

# MySQL for Beginners

## Student Guide

D61918GC30

Edition 3.0

May 2013

D82065

**ORACLE**

## Author

Mark Lewin

## Technical Contributors and Reviewers

Andrew Morgan

Martin Hansson

Bob Falsco

Mat Keep

Georgi Kodinov

Mike Lischke

Guilhem Bichot

Ford Brockman

Sanjay Manwani

Roy Lyseng

Craig McBride

Jeremy Smyth

## Editors

Smita Kommini

Vijayalakshmi Narasimhan

Raj Kumar

Arijit Ghosh

## Graphic Designer

Rajiv Chandrabhanu

## Publishers

Sujatha Nagendra

Srividya Rameshkumar

Copyright © 2013, 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 to MySQL

- Course Goals 1-2
- Course Lesson Map 1-4
- Introductions 1-6
- Classroom Environment 1-7
- MySQL Is Everywhere 1-8
- Why MySQL Makes Sense for Oracle 1-9
- Industry Leaders Rely on MySQL 1-10
- MySQL Database Server Editions 1-11
- MySQL Enterprise Edition 1-12
- MySQL Connectors and APIs 1-13
- MySQL Services 1-14
- Community Support 1-15
- Oracle Premier Support for MySQL 1-16
- MySQL-Supported Operating Systems 1-17
- MySQL Websites 1-18
- MySQL Curriculum Footprint 1-19
- MySQL Certification 1-20
- MySQL Online Documentation 1-21
- Example Databases 1-22
- Summary 1-23
- Practice 1-1 Overview: Accessing MySQL Resources Online 1-24

## 2 MySQL Server and Client

- Objectives 2-2
- MySQL Client/Server Model 2-3
- Communication Protocols 2-5
- MySQL Connectors 2-7
- LAMP Stack 2-9
- Installing MySQL 2-10
- MySQL Server and Command-Line Client 2-11
- Starting the MySQL Command-Line Client 2-12
- Startup Command-Line Options 2-13
- Keyboard Editing 2-14
- tee File 2-16

Quiz 2-17

Summary 2-18

Practice 2-1 Overview: Installing and Starting the MySQL Server 2-19

Practice 2-2 Overview: Using the Keyboard Editing and Tee Commands 2-20

Practice 2-3 Overview: Installing the world\_innodb Database 2-21

### **3 Database Basics**

Objectives 3-2

RDBMS: Overview 3-3

Spreadsheet Versus Database 3-4

Entities and Relationships 3-5

Relationship Categories 3-6

RDBMS Database Structure 3-7

Using SQL with Databases 3-8

SQL Statements: Data Definition Language 3-9

SQL Statements: Data Manipulation Language 3-10

MySQL: Benefits 3-11

Summary 3-12

Practice 3-1 Overview: Quiz – Database Basics 3-13

Practice 3-2 Overview: Identifying the Structure of a Table 3-14

### **4 Database Design**

Objectives 4-2

Database Modeling 4-3

ERD: Structure Diagram 4-4

ERD: Cardinality Diagram 4-5

Cardinality Diagram: Examples 4-6

Keys 4-7

Cardinality ERD: Example 4-8

Normalization 4-9

Advantages of Normalization 4-10

Disadvantages of Normalization 4-11

Eliminating Data Inconsistencies 4-12

Normal Forms 4-14

Normalization Process: Example 4-15

First Normal Form: 1NF 4-16

Second Normal Form: 2NF 4-17

Third Normal Form: 3NF 4-18

Normalized “Furniture Stores” Database 4-20

Quiz 4-21

Database Design Considerations 4-22

- Database Design Plan 4-23
- Database Design Plan Diagram 4-24
- Viewing a Database 4-25
- Evaluating a Database Design 4-26
- Summary 4-29
- Practice 4-1 Overview: Quiz – Database Design 4-30
- Practice 4-2 Overview: Evaluating a Database 4-31
- Practice 4-3 Overview: Creating a Structure Diagram 4-32

## **5 Table Data Types**

- Objectives 5-2
- Data Types as Part of Database Design 5-3
- Data Type Categories 5-4
- Numeric Data Types 5-5
  - Numeric: Integer Data Types 5-6
  - Numeric: Integer Data Types Comparison 5-7
  - Numeric: Floating-Point Data Types 5-8
  - Numeric: Floating-Point Data Types Comparison 5-9
  - Numeric: Fixed-Point Data Types 5-10
  - Numeric: Bit Data Types 5-11
- Temporal Data Types 5-12
  - Temporal Data Types Comparison 5-14
- Character String Data Types 5-15
  - Character String: Text Class Data Types 5-16
  - Character String: Text Class Data Types Comparison 5-17
  - Character String: Integer Class Data Types 5-18
- Character Set and Collation Support 5-19
- Binary String Data Types 5-21
  - Binary String: Data Types Comparison 5-22
- Quiz 5-23
- Choosing Data Types 5-24
- Setting Data Types to NULL 5-25
- When to Use NULL 5-26
- Summary 5-27
- Practice 5-1 Overview: Quiz – Data Types 5-28
- Practice 5-2 Overview: Explaining the Use of Data Types 5-29

## **6 Database and Table Creation**

- Objectives 6-2
- Creating a Database 6-3
- MySQL Naming Conventions 6-4

- Creating a Table 6-5
- Showing How a Table Was Created 6-7
- Column Options 6-8
- Table Options 6-9
- Table Indexes 6-10
- MySQL Indexing 6-11
- Showing Table Indexes 6-12
- Table Constraints 6-13
- Quiz 6-14
- Summary 6-15
- Practice 6-1 Overview: Displaying Table Creation Information 6-16
- Practice 6-2 Overview: Creating a Database 6-17

## **7 Basic Queries**

- Objectives 7-2
- SELECT Statement 7-3
- Using SELECT Clauses 7-5
- Using SELECT with DISTINCT 7-6
- SELECT DISTINCT with Null Values 7-8
- SELECT with WHERE 7-9
- SELECT with ORDER BY 7-13
- SELECT with ORDER BY with ASC and DESC 7-15
- SELECT with LIMIT 7-17
- SELECT with LIMIT and Skip Count 7-18
- SELECT with LIMIT and ORDER BY 7-19
- SELECT Usage Tips 7-20
- Troubleshooting 7-21
- SQL Modes for Syntax Checking 7-23
- Common SQL Modes 7-24
- Quiz 7-25
- MySQL Workbench for SQL Development 7-26
- Summary 7-27
- Practice 7-1 Overview: Performing Basic Queries 7-28
- Practice 7-2 Overview: Performing Basic Queries by Using MySQL Workbench 7-29
- Practice 7-3 Overview: Performing Basic Queries on the Pets Database 7-30

## **8 Database and Table Maintenance**

- Objectives 8-2
- DROP DATABASE Statement 8-3
- Creating a New Table by Using an Existing Table 8-4

Confirming the Creation of a New Table	8-5
Copying an Existing Table Structure	8-6
Creating a Temporary Table	8-7
DROP TABLE Statement	8-8
Adding a Table Column	8-9
Removing a Table Column	8-11
Modifying a Table Column	8-12
Modifying a Table Column: Row Changes	8-13
Adding Indexes and Constraints	8-14
Adding a Column Index	8-15
Dropping a Column Index	8-17
Quiz	8-18
Summary	8-19
Practice 8-1 Overview: Removing a Database	8-20
Practice 8-2 Overview: Creating a New Table and Removing a Table	8-21
Practice 8-3 Overview: Altering Table Columns	8-22
Practice 8-4 Overview: Modifying Table Indexes and Constraints	8-23
Practice 8-5 Overview: Further Practice	8-24

## **9 Table Data Manipulation**

Objectives	9-2
Manipulation of Table Row Data	9-3
INSERT Statement	9-4
Using INSERT for Multiple Rows	9-5
Column Value Assignment with INSERT	9-6
REPLACE Statement	9-7
REPLACE Results	9-8
REPLACE Algorithm	9-9
Quiz	9-10
UPDATE Statement	9-11
UPDATE Statement Ordering	9-12
UPDATE with LIMIT	9-13
When UPDATE Has No Effect	9-14
Difference Between UPDATE and REPLACE	9-15
DELETE Statement	9-16
DELETE with ORDER BY and LIMIT	9-17
Summary	9-19
Practice 9-1 Overview: Inserting and Replacing Table Row Data	9-20
Practice 9-2 Overview: Modifying and Deleting Table Row Data	9-21
Practice 9-3 Overview: Manipulating Table Row Data in the Pets Database	9-22

## 10 Functions

- Objectives 10-2
- Functions in MySQL Expressions 10-3
- Using Functions 10-4
- Types of Functions 10-5
- String Functions 10-6
- String Functions (Numeric): Examples (CHAR\_LENGTH, INSTR, STRCMP) 10-7
- String Functions: Examples (CONCAT, REVERSE, LEFT, RIGHT) 10-8
- String Functions: Examples (LOWER, UPPER, LPAD, RPAD) 10-9
- String Functions: Examples (TRIM) 10-10
- String Functions: Examples (SUBSTRING) 10-11
- String Functions: Examples (SUBSTRING\_INDEX) 10-12
- Temporal Functions 10-13
- Temporal Functions: Date/Time Formats 10-14
- Temporal Functions: Function Types 10-15
- Temporal Functions: Examples 10-16
- Numeric Functions 10-18
- Numeric Functions: Examples 10-19
- Numeric Functions: Additional Functions 10-20
- Control Flow Functions 10-21
- Control Flow Functions: CASE Functions 10-22
- Control Flow Functions: CASE Function Syntax 10-23
- Control Flow Functions: CASE Function Examples 10-24
- Quiz 10-25
- Aggregate Functions 10-26
- Aggregate Function Types 10-27
- Aggregate Functions: COUNT Function Examples 10-28
- Aggregate Functions: GROUP BY Clause 10-29
- Aggregate Functions: GROUP BY Clause and GROUP\_CONCAT Function 10-30
- Aggregate Functions: GROUP BY and HAVING Clauses 10-31
- Aggregate Functions: GROUP BY Clause and WITH ROLLUP Modifier 10-32
- Aggregate Functions: Super-Aggregate Operation 10-33
- Spaces in Function Names 10-34
- Summary 10-35
- Practice 10-1 Overview: Quiz 10-36
- Practice 10-2 Overview: Using Built-In, String, and Temporal Functions 10-37
- Practice 10-3 Overview: Using Numeric and Control Flow Functions 10-38
- Practice 10-4 Overview: Using Aggregate Functions 10-39



## 11 Exporting and Importing Data

- Objectives 11-2
- Exporting Data 11-3
- Exporting with a Query 11-4
- Exporting with a Query: INTO OUTFILE 11-5
- Exporting with a Query: CSV Format 11-6
- Exporting with a MySQL Utility 11-7
- Specifying Entities to Dump 11-8
- mysqldump Utility Options 11-9
- Dumping to a Text File 11-10
- Text File Contents 11-11
- Importing Data 11-12
- Importing from a Data File 11-13
- Importing with the LOAD DATA INFILE Statement 11-14
- Importing with LOAD DATA INFILE: CSV Format 11-15
- Importing with LOAD DATA INFILE: File Control 11-16
- Importing with a MySQL Utility 11-17
- mysqlimport Utility Options 11-18
- mysqlimport Option: Examples 11-19
- Summary 11-20
- Practice 11-1 Overview: Quiz 11-21
- Practice 11-2 Overview: Exporting Files by Using a Query 11-22
- Practice 11-3 Overview: Importing Files from a Data File 11-23
- Practice 11-4 Overview: Backing Up Database Files with a Utility 11-24

## 12 Joining Tables

- Objectives 12-2
- Combining Multiple Tables 12-3
- Table Joins 12-4
- Cross Joins 12-5
- Multiple Tables in the FROM Clause 12-6
- INNER JOIN Keyword 12-8
- JOIN Keyword 12-9
- Outer Joins 12-10
- Finding Mismatches with LEFT JOIN 12-12
- Finding Mismatches with RIGHT JOIN 12-13
- Outer Joins: USING and NULL 12-14
- Table Name Aliases 12-15
- Quiz 12-16
- Summary 12-17
- Practice 12-1 Overview: Performing Inner and Outer Joins 12-18

Practice 12-2 Overview: Creating Queries Requiring Joins 12-19

Practice 12-3 Overview: Additional Optional Practice 12-20

### **13 Table Subqueries**

Objectives 13-2

Subquery: Overview 13-3

Basic Subquery: Example 13-4

Advantages of Using a Subquery 13-5

Placement of Subqueries 13-6

Subquery Categories 13-7

Non-Correlated Subquery: Example 13-8

Correlated Subquery: Example 13-9

Subquery Result Table Types 13-10

Subquery Type/Placement Chart 13-11

Scalar Subqueries 13-12

Column Designation Subquery 13-13

FROM Subquery 13-14

WHERE Subquery 13-15

WHERE Subquery with Operators 13-16

Subqueries in Comparisons 13-17

WHERE with IN or NOT IN 13-18

WHERE with ALL, ANY, and SOME 13-19

ANY: Example 13-20

ANY: Example Subquery 13-21

WHERE with EXISTS and NOT EXISTS 13-22

Finding Table Mismatches 13-23

Modifying Table Data With Subqueries 13-24

Quiz 13-25

Subquery Tips 13-26

Converting Joins to Subqueries 13-27

Summary 13-28

Practice 13-1 Overview: Performing Different Types of Subqueries 13-29

Practice 13-2 Overview: Performing Several Advanced Subqueries 13-30

### **14 MySQL Graphical User Interface Tools**

Objectives 14-2

MySQL GUIs 14-3

MySQL Workbench 14-4

MySQL Workbench Functionality 14-5

MySQL Workbench: SQL Development 14-7

MySQL Workbench: Data Modeling 14-8

MySQL Workbench: Model Editor	14-9
MySQL Workbench: Model Overview Panel	14-10
MySQL Workbench: Data Modeling EER Diagram	14-11
MySQL Workbench: Server Administration	14-12
MySQL Workbench: Server Administration Features	14-13
MySQL Enterprise Monitor	14-14
Enterprise Monitor Features	14-15
Enterprise Monitor: Dashboard	14-17
Enterprise Monitor: Advisors	14-18
Quiz	14-20
Summary	14-21
Practice 14-1 Overview: Creating a Data Model by Using MySQL Workbench	14-22
Practice 14-2 Overview: Creating a Server Instance by Using MySQL Workbench	14-23
Practice 14-3 Overview: Viewing MySQL Enterprise Monitor Demonstrations	14-24
Practice 14-4 Overview: (Optional) Viewing the MySQL Workbench Demonstration	14-25
Practice 14-5 Overview: (Optional) Creating a Model for the Pets Database by Using MySQL Workbench	14-26

## **15 Supplementary Information**

Objectives	15-2
Storage Engines and MySQL	15-3
Storage Engines Available on the Server	15-4
Displaying Storage Engine Setting: Using SHOW CREATE TABLE	15-5
Displaying Storage Engine Setting: Using SHOW TABLE STATUS LIKE	15-6
Setting the Storage Engine to InnoDB	15-7
InnoDB Storage Engine	15-8
Other Storage Engines	15-9
Quiz	15-10
Creating Views	15-11
CREATE VIEW: Examples	15-12
Displaying View Information	15-13
Views: Showing Table Types	15-14
View Definition Restrictions	15-15
Quiz	15-16
Transactions	15-17
Transactions: ACID	15-18
Transaction Diagram	15-19
Transaction SQL Control Statements	15-20
Starting a Transaction	15-21

Quiz 15-22  
Retrieving Metadata 15-23  
Metadata: SHOW Statements 15-24  
Metadata: INFORMATION\_SCHEMA Database 15-25  
Metadata: INFORMATION\_SCHEMA Tables 15-26  
Metadata: Viewing INFORMATION\_SCHEMA 15-28  
Metadata: SCHEMATA 15-29  
Quiz 15-30  
MySQL Performance Schema 15-31  
Performance Schema: Implementation 15-32  
Performance Schema: SHOW Database Tables 15-33  
Quiz 15-34  
MySQL Enterprise Backup 15-35  
MySQL Enterprise Backup: Preparation 15-36  
MySQL Enterprise Backup: Implementation 15-37  
MySQL Enterprise Backup: Running mysqlbackup 15-38  
MySQL Enterprise Backup: Some mysqlbackup Options 15-39  
MySQL Enterprise Backup: Recovering or Restoring a Database 15-40  
Quiz 15-41  
Summary 15-42  
Practice 15-1 Overview: Displaying Storage Engine Information 15-43  
Practice 15-2 Overview: Displaying and Creating Views 15-44  
Practice 15-3 Overview: Obtaining Metadata 15-45  
Practice 15-4 Overview: (Optional) Creating a Backup of MySQL Databases 15-46

## **16 Conclusion**

Course Goals 16-2  
MySQL Curriculum Path 16-4  
MySQL Resources 16-5  
Your Evaluation 16-6  
Thank You 16-7  
Q&A Session 16-8

# 1

## Introduction to MySQL

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Course Goals

After completing this course, you will be able to:

- Describe the features and benefits of MySQL
- Explain the basics of relational databases
- Explain MySQL connectors and their major features and differences
- Describe the SQL and MySQL languages
- Describe data/column types with regard to efficient database design
- View a database design structure and content
- Create a database design by using an efficient structure
- Extract basic database information by using the **SELECT** command
- Troubleshoot and identify typical syntax warnings and errors

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Course Goals

- Delete or modify a database and table row data
- Group query data with aggregation
- Connect data from multiple table rows with a join
- Perform nested subqueries
- Use simple functions (String, Date, Numerical)
- Explain MySQL storage engines and transactions, and the features of the common engines
- Export and import database data
- Obtain database metadata
- Monitor server database performance
- Describe MySQL graphical user interface tools
- Perform database backup and recovery

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Course Lesson Map

## Day 1

1. Introduction to MySQL
2. MySQL Server and Client
3. Database Basics
4. Database Design
5. Table Data Types

## Day 2

6. Database and Table Creation
7. Basic Queries
8. Database and Table Maintenance
9. Table Data Manipulation

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@oracle.com) has a non-transferable license to use this Student Guide.



# Course Lesson Map

## Day 3

- 9. Table Data Manipulation (*continued*)
- 10. Functions
- 11. Exporting/Importing Data
- 12. Joining Tables

## Day 4

- 13. Table Subqueries
- 14. MySQL Graphical User Interface Tools
- 15. Supplementary Information
- 16. Conclusion

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# 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 © 2013, 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 © 2013, Oracle and/or its affiliates. All rights reserved.

# MySQL Is Everywhere

- The world's most popular open source database
- Over 15 million estimated active installations
- The M of the LAMP stack
- Used by 9 of the top 10 websites in the world
- Embedded by over 3,000 ISVs and OEMs
- The leading database in the cloud
- Highly popular on social media (Facebook, Twitter, and so on)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**\*Acronyms:** linux, apache, MySQL, PHP/Perl/Python (LAMP); independent software vendor (ISV); original equipment manufacturer (OEM)

# Why MySQL Makes Sense for Oracle

- Complete Solutions
- Best of Breed at Every Level
- On Premises and in the cloud
- MySQL: Web, Mobile, Embedded



## Oracle Drives MySQL Innovation

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL makes sense for Oracle. MySQL represents Oracle's best-of-breed database solution for web-based applications, and is also a great choice as an embedded database. MySQL therefore completes Oracle's offerings, and is very complementary to the Oracle database. This is why Oracle significantly invests in MySQL—to deliver MySQL solutions powering next-generation web, mobile, and embedded applications.

## Industry Leaders Rely on MySQL



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL is the world's most popular open source database, and the number one database for web-based applications, powering leading websites worldwide including Facebook, Twitter, and YouTube. MySQL is also widely used by enterprise and governmental organizations for web applications as well as custom enterprise applications or data marts. Those are very complementary use cases to the Oracle databases that are typically found in ERPs and other such high-end enterprise applications.

MySQL is also an extremely popular choice as embedded database, distributed by thousands of ISVs/OEMs. They include software vendors such as Sage embedding MySQL in accounting applications, or eClinicalWorks shipping MySQL within medical practice management software. MySQL is also integrated in security solutions delivered by Cisco or Checkpoint, for example, and in numerous telecom offerings including Alcatel Lucent, Avaya, and many others.

Finally, MySQL has become the leading database in the cloud, offered by the majority of cloud service providers, and powering countless SaaS applications.

**\*Acronyms:** enterprise resource planning (ERP), software as a service (SaaS)

# MySQL Database Server Editions

## GPL

- **MySQL Community Edition**
  - Open source, noncommercial
- **MySQL Cluster Community Edition**
  - Open source, noncommercial, cluster

## Commercial

- **MySQL Classic Edition**
  - Embedded database for OEMs, ISVs, VARs
- **MySQL Standard Edition**
  - High-performance, scalable OLTP applications, MySQL Workbench SE
- **MySQL Enterprise Edition**
  - Enterprise features: monitoring, backup, scalability, security, auditing, high availability
  - Oracle product integrations
- **MySQL Cluster Carrier Grade Edition**
  - High-availability, fault-tolerance
  - MySQL Cluster Manager
  - Superset of MySQL Commercial Editions

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL Classic Edition is well suited for embedded and read-intensive, non-OLTP applications.

MySQL Standard and Enterprise Editions are well suited for read-intensive and OLTP applications that require high performance, high availability, and consistent crash recovery.

In addition to the Community edition, OEMs can license or subscribe to all editions, and end users can subscribe to Standard, Enterprise, and Cluster Carrier Grade editions.

The commercial editions build on each other:

- Standard Edition includes Classic Edition plus the additional features listed in the slide.
- Enterprise Edition includes Standard Edition plus the additional features listed in the slide.
- Carrier Grade Edition includes Enterprise Edition plus the additional features listed in the slide.

For more information about licensing and subscription options, see:

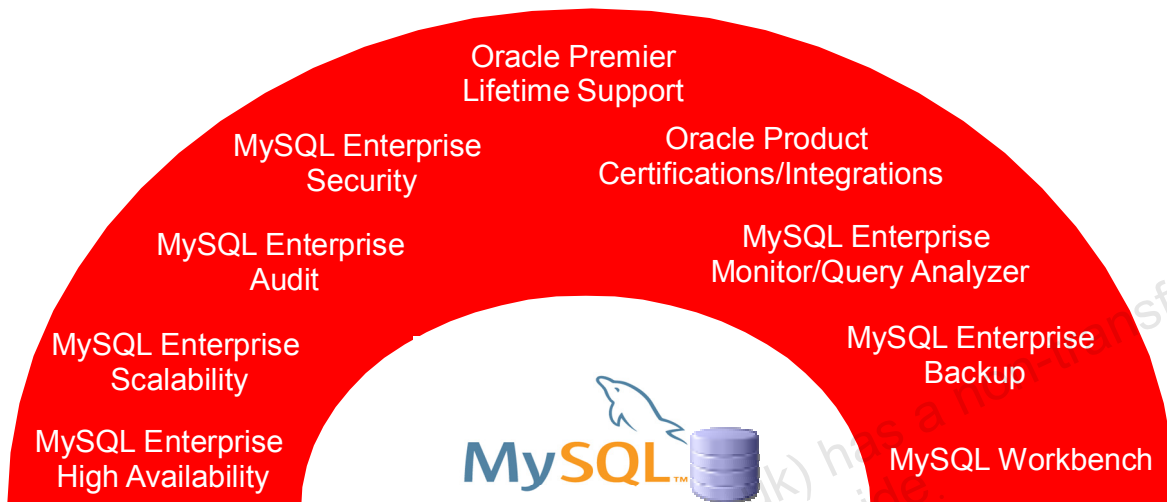
- <http://mysql.com/products/>
- <http://www.mysql.com/about/contact/>
- <https://shop.oracle.com/> – Select Databases > MySQL.

**\*Acronyms:** general public license (GPL), original equipment manufacturer (OEM), independent software vendor (ISV), value-added reseller (VAR), online transactional processing (OLTP)



# MySQL Enterprise Edition

Highest levels of MySQL scalability, security, and uptime



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

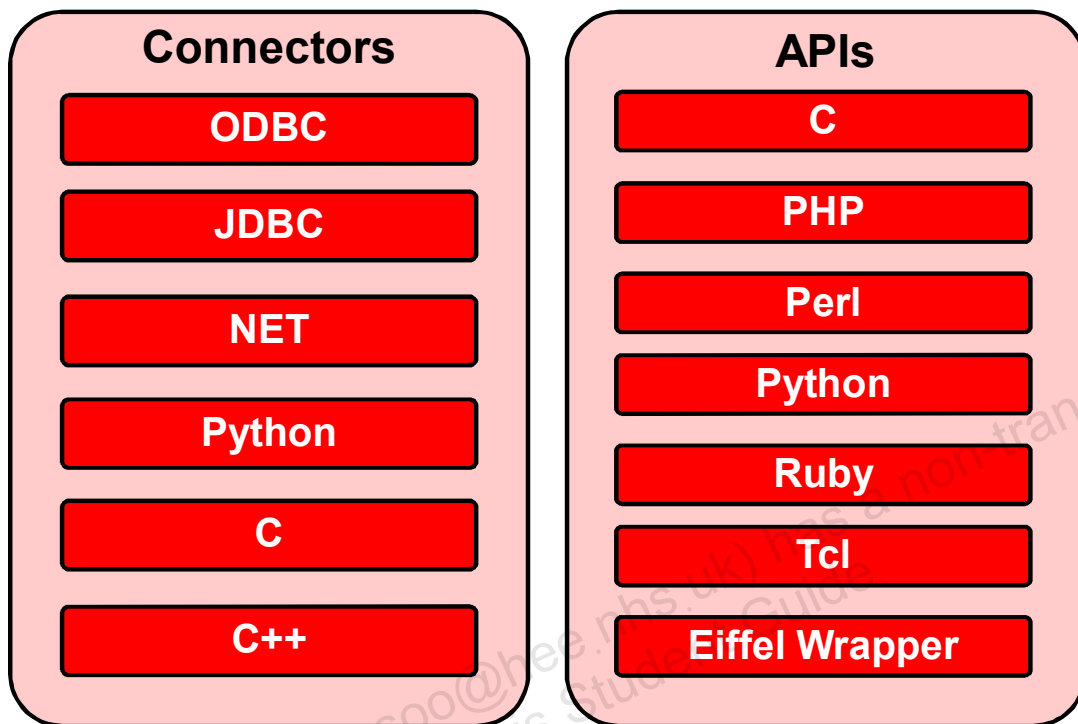
MySQL Enterprise Edition includes the most comprehensive set of advanced features, management tools, and technical support to achieve the highest levels of MySQL scalability, security, reliability, and uptime. It reduces the risk, cost, and complexity in developing, deploying, and managing business-critical MySQL applications.

MySQL Workbench is available in both GPL and commercial editions.

More details about these tools are presented later in this course.



# MySQL Connectors and APIs



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to the MySQL protocol and MySQL resources. Both the connectors and the APIs enable you to connect and execute MySQL statements from another language or environment.

Third-party connectors that MySQL supports:

- **PHP:** `mysql`, `ext/mysql`, `PDO_MYSQLND`, `PHP_MYSQLND`
- **Perl:** `DBD::mysql`
- **Python:** `MySQLdb`
- **Ruby:** `DBD::MySQL`, `ruby-mysql`
- **C++ Wrapper:** For MySQL C API (`MySQL++`)

The embedded MySQL server library (`libmysqld`) is also supported. `libmysqld` makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and simpler management for embedded applications.

You can download connectors and their documentation from the following page:  
<http://mysql.com/products/connector/>.

For more information about connectors, see the *MySQL Reference Manual*:  
<http://dev.mysql.com/doc/refman/5.6/en/connectors-apis.html>.

# MySQL Services

## MySQL training

- Comprehensive set of MySQL training courses

## MySQL certification

- High-quality certification for MySQL developers and database administrators

## MySQL consulting

- Full range of consulting services from startup to optimization

## Oracle Premier Lifetime Support

- Offerings to save time and ensure that you achieve the highest levels of performance, reliability, and uptime

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Visit <http://www.oracle.com/education/mysql> to see the full range of MySQL training options.

# Community Support

- Mailing lists
- Forums
- Community articles
- PlanetMySQL blogs
- Twitter
- Physical and virtual events, including:
  - Developer Days
  - Tech Tours
  - Webinars
- Bug tracking
- Launchpad repositories

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- Mailing lists (<http://lists.mysql.com/>)
- Forums (<http://forums.mysql.com>)
- Community articles (<http://dev.mysql.com/tech-resources/articles>)
- Planet MySQL blogs (<http://planet.mysql.com/>)
- Twitter ([http://twitter.com/mysql\\_community](http://twitter.com/mysql_community) and <http://twitter.com/MySQL>)
- Physical and virtual events (<http://www.mysql.com/news-and-events>)
- Bug tracking (<http://bugs.mysql.com>)
- Launchpad repositories

# Oracle Premier Support for MySQL

## Rely on the experts, get unique benefits.

- Straight from the source
- Largest team of MySQL experts
- Backed by MySQL developers
- Forward-compatible hot fixes
- MySQL maintenance releases
- MySQL support in 29 languages
- 24/7/365
- Unlimited incidents
- Knowledge Base
- MySQL consultative support

**Only From**  
**Oracle**

"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 © 2013, Oracle and/or its affiliates. All rights reserved.

The MySQL support organisation is the largest team of MySQL experts, backed directly by the engineers that develop MySQL. Premier support is available 24x7 in 29 languages.

The best practices Knowledge Base offers good insight into how to configure, provision, and manage highly available MySQL environments. Knowledge Base benefits include:

- Unlimited support incidents
- Hot fixes
- Maintenance releases, bug fixes, patches, and updates
- MySQL consultative support

Through tools and support, you can get the most out of your big data pipeline.

# 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, x86\_64, x86)
  - Mac OS X (x86, x86\_64)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A full list of supported operating systems is at:

<http://www.mysql.com/support/supportedplatforms/database.html>

# MySQL Websites

- <http://dev.mysql.com> includes:
  - Developer Zone (articles, forums, PlanetMySQL, and more)
  - Downloads (GA and development release)
  - Documentation
- <http://www.mysql.com> includes:
  - Downloads (GA)
  - Services
  - Consulting
  - Resources
  - White papers
  - Webinars

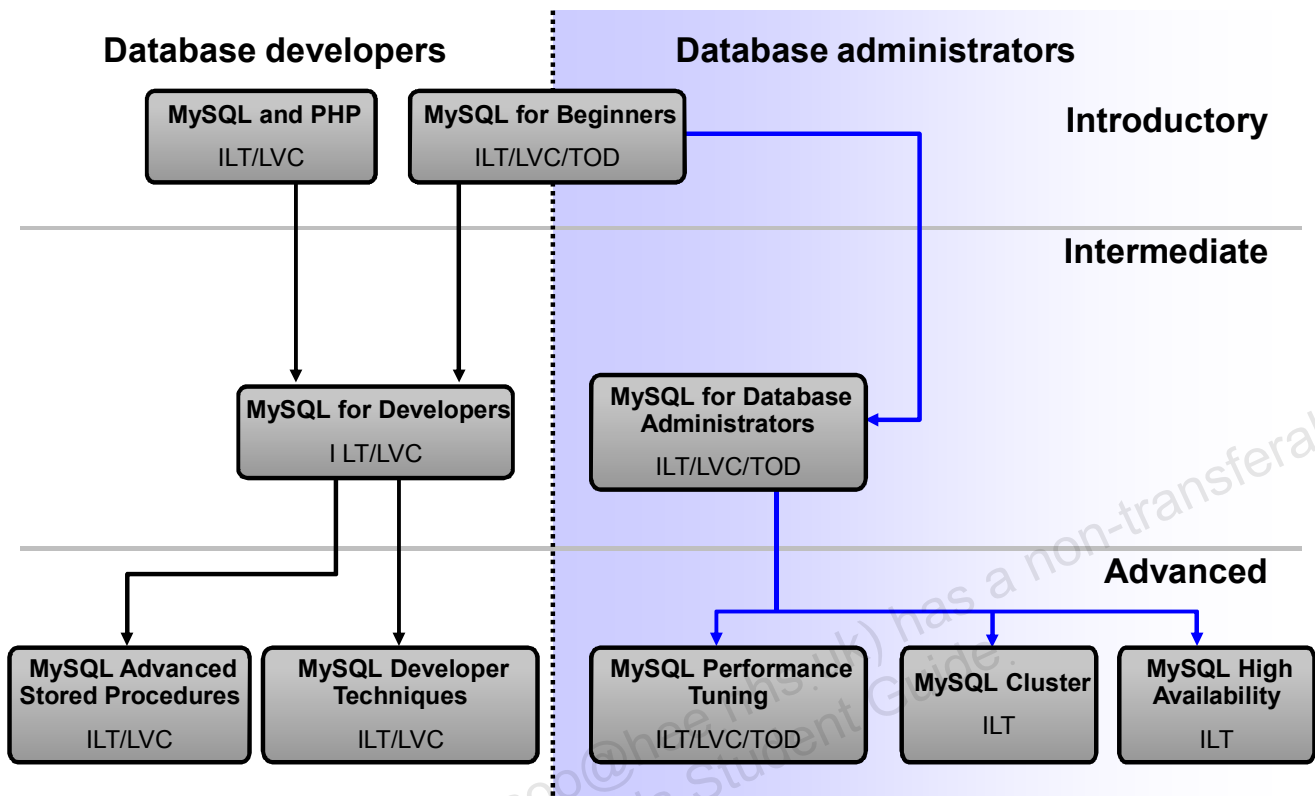
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For more information, see:

- <http://dev.mysql.com>
- <http://www.mysql.com/>

# MySQL Curriculum Footprint



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Course Types

- **Instructor-Led Training (ILT):** Delivered in a classroom with an 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 takes traditional classroom training (complete with all classroom content including lectures, white boarding, and lab videos) and makes it available in a video-based, online format so you can start your customized training at your convenience.

Some of these courses are available in combinations. For full details about the MySQL curriculum, go to <http://www.oracle.com/education/MySQL>.

# MySQL Certification

The Oracle Certification Program validates various levels of MySQL expertise:

- **Introductory: Certified Associate**
  - Oracle Certified Associate: MySQL
- **Intermediate: Certified Professional**
  - Oracle Certified Professional: MySQL Database Administrator
  - Oracle Certified Professional: MySQL Developer
- **Advanced: Certified Expert**
  - Oracle Certified Expert: MySQL Cluster Database Administrator

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For complete information about these and other offerings of the Oracle Certification Program, go to <http://education.oracle.com/certification>.



# MySQL Online Documentation

- *MySQL Reference Manual*
- Topic guides
- Expert guides
- MySQL help tables
- Example databases
- Meta documentation
- Community-contributed documentation
- Printed books
- Additional resources
- Tutorials
- Benchmark suites

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The MySQL documentation is available at <http://dev.mysql.com/doc>.

## Example Databases

- MySQL provides several example/test databases:
  - world\_innodb
  - world
  - employee
  - menagerie
  - sakila
- They are available for download from the MySQL Developer Zone website: <http://dev.mysql.com/doc/index-other.html>.
- The world\_innodb and sakila databases are used for demonstration and practices in this course.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For more information about example database installation, see the *MySQL Reference Manual* at <http://dev.mysql.com/doc/index-other.html>.

# Summary

In this lesson, you should have learned how to:

- Describe the course goals
- Explain the origin and status of the MySQL product
- List the available MySQL products and professional services
- List the currently supported operating systems
- Describe how to access MySQL information and services on Oracle and MySQL websites
- List the available MySQL courses
- Describe the MySQL Certification program
- Obtain MySQL online documentation and example databases

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 1-1 Overview: Accessing MySQL Resources Online

This practice covers reviewing the MySQL and Oracle websites that contain information about MySQL products and services.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# 2

## MySQL Server and Client

### Connecting to the Database

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you will be able to:

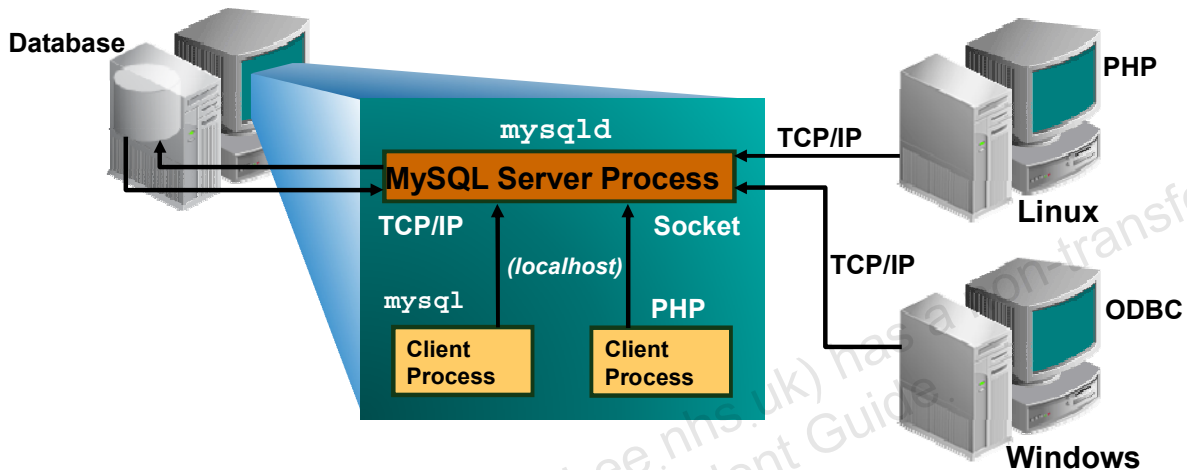
- Explain the MySQL client/server model
- Describe the communication protocols used to interact with the MySQL server
- Explain how MySQL works with connectors
- Identify the components of a LAMP stack and their roles
- Install the MySQL server
- Start the MySQL server and the `mysql` client
- Install the `world_innodb` example database
- Use command-line editing methods
- Use a text file to log your `mysql` client session

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# MySQL Client/Server Model

## MySQL client/server architecture



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The MySQL database system uses a client/server architecture. The MySQL server is the central program that manages database contents, and the client programs connect to the server to retrieve or modify data. To access the data in a database, a MySQL server must be running. The MySQL server program (`mysqld`) listens on a particular port for incoming connection requests—the default port for MySQL is 3306. You can change this port number to a different available port, if desired. You can also return it to the default port (3306), if it has been changed.

The client (in MySQL) is any application that sends a request to the database via the server. A client is used to communicate with the MySQL server any time a command statement is sent, whether that is a PHP script or one of the standard MySQL clients. The client connects, does what it needs to do, and then disconnects. The server can handle multiple client connections at once.

**\*Acronyms:** php: hypertext processor (PHP), open database connectivity (ODBC), transmission control protocol/internet protocol (TCP/IP)

## Connection Methods

- On Windows, local connections can use a shared pipe or shared memory. However, most local connections use TCP/IP.
- UNIX/Linux local connections can use a UNIX socket file.
- Remote connections must be made via TCP/IP. However, TCP/IP can also be used for a local connection.

The `d` in `mysqld` is for daemon; a program or process that sits idly in the background until it is invoked to perform its task.

**Note:** Any information in this course that does not apply to all operating systems is identified as platform-specific. Information that works for Linux generally applies to all UNIX-like operating systems.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.



# Communication Protocols

Protocol	Types of Connections	Supported Operating Systems
TCP/IP	Local, remote	All
UNIX socket file	Local only	UNIX only
Shared memory	Local only	Windows only
Named pipes	Local only	Windows only

- Protocols are implemented in the client libraries and drivers.
- The connection speed depends on the protocol used and how it is implemented.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- **TCP/IP (Transmission Control Protocol/Internet Protocol):** TCP/IP is the suite of communication protocols used to connect hosts on the Internet, but you can also use it for local connections. This is the best MySQL connection type for Windows but you can use it for all supported operating systems.
- **UNIX socket file:** This protocol allows processes on the same machine to communicate with each other. It relies on a physical file (the “socket”), which the processes access by using a special set of functions. The protocol allows easy exchange of data in both directions and is the best choice for local MySQL connections on Linux.
- **Shared memory:** This is an efficient means of passing data between processes running on the same Windows machine. Windows makes an area of memory available, which trusted processes can read and write to. It is very efficient because programs can access this memory directly without making calls to the operating system. Shared memory is disabled by default. To enable shared-memory connections, you must start the server with the `--shared-memory` option.

- **Named pipes:** This protocol, like shared memory, lets one process use a region of memory to pass information to another process. Unlike shared memory, the named pipe protocol works not just between processes running on the same machine but on the same Local Area Network. Named pipes are disabled by default. To enable named-pipe connections, the server must be started with the `--enable-named-pipe` option.

# MySQL Connectors

- Application programming interfaces (APIs)
- Most are compatible with industry standards ODBC and JDBC.
- Available for multiple operating systems
- Official MySQL-supported connectors:
  - Connector/ODBC
  - Connector/J
  - Connector/NET
  - Connector/Python
  - Connector/C and Connector C++
- Embedded MySQL server library: `libmysqld`
- Multiple PHP extensions and APIs supported

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL works with several application programming interfaces also known as “database drivers”. These APIs and drivers act as bridges to the MySQL server for client programs that communicate using a particular protocol. The drivers comprise the family of MySQL connectors. They are available as separate packages.

The MySQL connectors are available for Windows, Linux/UNIX, and Mac OS X. To use a connector, you must install it on the client host. The server does not have to be on the same machine or running the same operating system as the client. Therefore, MySQL connectors are very useful for providing MySQL connectivity in heterogeneous environments. For example, people who use Windows machines can run client applications that access MySQL databases on a Linux server host.

Most connectors are based on the C client library and provide a binding for some other language. The MySQL C client library is the standard client/server protocol implementation from MySQL, (which is considered to be the reference implementation).

The embedded MySQL server library `libmysqld` allows you to run a full-featured MySQL server inside a client application. The main benefits are increased speed and simpler management of embedded applications.

\***Acronyms:** java database connectivity (JDBC)

There are several MySQL PHP extensions. These extensions can use either the `mysqlnd` (MySQL Native Driver) or `libmysql` (the C client library) to connect to MySQL from PHP. Oracle recommends using `mysqlnd` which is optimized for PHP (but not compatible with ODBC or JDBC).

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# LAMP Stack

- MySQL is the database component in the LAMP stack, which is the preferred stack for web applications:
  - **Linux:** Operating system
  - **Apache:** Web server
  - **MySQL:** Relational Database Management System
  - **PHP/Perl/Python:** Programming languages
- LAMP allows different deployment options for MySQL.
- The Windows alternative is WAMP.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Though the originators of these open source programs did not design them all to work specifically with each other, the combination has become popular because its components are free to acquire and used widely. Together, they represent a solution stack of technologies that support application servers.

This technology allows the user of a web browser to execute a program on the web server, which can then serve dynamic as well as static content.

The LAMP stack allows several different options for MySQL deployments:

- Use Windows instead of Linux (WAMP).
- Put the MySQL server process on a separate computer.
- Replicate the MySQL server process on several computers.
- Run several MySQL server processes on one server.

# Installing MySQL

- Download locations for MySQL:
  - Community: <http://dev.mysql.com/downloads>
  - Enterprise: <https://edelivery.oracle.com/>
- This course uses the Windows operating system.
- Oracle recommends the MySQL Installer for Windows:
  - Installs several MySQL products
- Example databases for testing MySQL features:
  - This course uses the `world_innodb` database for examples and practices.
  - This and other sample databases are available online.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can download all licensable Oracle products from the Oracle Software Delivery Cloud website at <https://edelivery.oracle.com/>.

MySQL Installer for Windows installs the following MySQL products:

- MySQL Server
- All supported connectors
- Workbench and sample models
- Sample databases
- Documentation

For more information about MySQL installation, see the MySQL Reference Manual: <http://dev.mysql.com/doc/refman/5.6/en/installing.html>.

# MySQL Server and Command-Line Client

- The server starts automatically after standard installation on the Windows platform.
- Connect to the server with the `mysql` command-line client.
- You use the command-line client primarily for:
  - Executing queries
  - Retrieving query results

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use the `mysql` command-line client to:

- Create databases
- Check syntax
- Simplify program development
- Create tables
- Show database structure
- Optimize queries
- Alter table descriptions
- Test `SELECT` statements
- Perform backup and restore
- Create indexes
- Do bulk updates
- Grant access rights

# Starting the MySQL Command-Line Client

Start the `mysql` client with the following command:

```
shell> mysql -u root -p
Enter password: <password>
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.10 MySQL Community Server (GPL)
...
Type 'help;' or '\h' for help. Type '\c' to clear the
current input statement.
mysql>
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

After invoking the `mysql` client, take note of the following:

- Connection ID and server version
- `help` command

At this point, the only available account is the root account (and possibly an anonymous account). The root account has global permissions and makes it easy to delete data and system information by mistake. In a production environment, create new user accounts with appropriate permissions.



# Startup Command-Line Options

- Get a full list of options by using `--help`:

```
shell> mysql --help
```

- Some primary options:
  - `-h`: Host name or IP address
  - `-u`: Username
  - `-p`: Password
  - `<database>`: Database name
  - `<`: Input script
  - `>`: Output file name
  - `-b`: Beeps
- Syntax example:

```
shell> mysql -h 192.145.66.10 -u wld -pmypass  
world_innodb < sqlcommands.sql
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The primary options are:

- `-h <hostname>` or `<IP>`: Specifies the host machine where the MySQL server resides. You can usually omit this if the client and the server are on the same host.
- `-u<username>`: Specifies a username other than the default (ODBC on Windows or your login name on UNIX). You can put a space between the option and the login name, but you do not have to.
- `-p<password>`: Specifies the user password. If you do not supply `<password>` after the `-p`, you will be prompted for it. You cannot have a space between the option and the password.
- `<database>`: (Optional) Sets the default database for the current session
- `< <input_script>`: Connects to the MySQL server and executes the SQL statements in this file
- `> <filename>`: Saves the output sent by the MySQL server to this file
- `-b`: Disables the beeps the client makes for errors

The example in the slide executes the `mysql` client, connects to the host IP 192.145.66.10 as the user `wld`, sets the default database to `world_innodb`, and issues the SQL statements found in the `sqlcommands.sql` file.

# Keyboard Editing

To edit `mysql` client commands, you can use:

- Keyboard directional arrows
- The command history preserved during the session
- Special character sequences to interrupt a query
- Full readline capabilities under Linux
- Enforced quote closure
- Copy and paste

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Some editing methods:

- The four keyboard arrows:
  - Right and left arrows ( $\leftarrow$ ,  $\rightarrow$ ) move the cursor one step forward and backward.
  - Up and down arrows ( $\uparrow$ ,  $\downarrow$ ) let you cycle your way through the command history.
- Command history is preserved between sessions.
  - History clears if you close the command-prompt window in Windows.
  - History persists when you close the shell-prompt window in Linux.
- Cancel a query before you have finished entering it with `\c`.
- The `mysql` client has full `readline` capabilities under Linux.
  - `Ctrl-A`: Moves to the beginning of the line (Anfang)
  - `Ctrl-E`: Moves to the end of the line (Ende)
  - `Ctrl-K`: Kills from the cursor onward
  - `Ctrl-R`: Searches through the command history in reverse

**Note:** MySQL commercial versions use `libedit` for command-line editing instead of `readline` but the two are compatible.

- Close the quotes (otherwise, the prompt does not let you continue):
  - ' >
  - " >
- Copy from the DOS window.
  - Right-click and select Mark. Click the first character to copy it.
  - Press and hold the Shift key and click the last character to copy it.
  - Press Enter to place the selection in the paste buffer.
- Paste into DOS window.
  - Place the cursor at the prompt.
  - Right-click and select Paste.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## tee File

- The `tee` file is a text file that logs all `mysql` client statements and their output.
- Everything you see displayed on the screen is appended into a file that you specify.
- You can enable this feature interactively within the `mysql` client with the `tee` command:

```
mysql> tee session_tee_log.txt
```

- You can disable `tee` file logging with the `notee` command.
- Executing `tee` again re-enables logging.
- View the `tee` file contents in your favorite text editor.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you write to an existing `tee` file, it appends the new statement information to the current file content.

## Quiz

What command do you use to list and define the options available from the `mysql` command-line client?

- a. `mysql -u root -p -h`
- b. `mysql --help`
- c. Use the down-arrow key to step through the command history.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: b**

# Summary

In this lesson, you learned how to:

- Explain the MySQL client/server model
- Describe the communication protocols used to interact with the MySQL server
- Explain how MySQL works with connectors
- Identify the components of a LAMP stack and their roles
- Install the MySQL server
- Start the MySQL server and the `mysql` client
- Install the `world_innodb` example database
- Use command-line editing methods
- Use a text file to log your `mysql` client session

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 2-1 Overview: Installing and Starting the MySQL Server

In this practice, you:

- Install the MySQL server by using the MySQL Installer for Windows.
- Start the `mysql` client from the command-line prompt.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 2-2 Overview: Using the Keyboard Editing and Tee Commands

In this practice, you:

- Use the keyboard editing methods
- Create and view a `tee` file

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.



## Practice 2-3 Overview: Installing the `world_innodb` Database

In this practice, you create and populate the `world_innodb` database.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# 3

## Database Basics

### Relational Database

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Define a relational database (and RDBMS)
- Describe the structure of an RDBMS database
- Identify the differences between databases and spreadsheets
- Explain the use of SQL and MySQL with relational databases
- Define and use data definition language (DDL)
- Define and use data manipulation language (DML)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# RDBMS: Overview

- A Relational Database Management System (RDBMS):
  - Manages data according to the relational model
  - Organizes and stores data in the form of tables
  - Example: A bank database has tables for currency data, customer information, account information, and so on.
- A database is a collection of data placed in tables.
  - Synonymous with “schema”
- The relational model
  - Organizes data logically as tables and the relationships between them
  - Is an improvement over the hierarchical database model, which:
    - Requires additional steps to coordinate the data and programs
    - Is difficult to work with

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In some RDBMSs, a single database can have multiple “schemas.” In MySQL a database and a schema are the same thing.

In the relational model, the fundamental structure for organizing data is the “relation,” which is where it gets its name. For most purposes, the term “table” is interchangeable with the more abstract term “relation.” The term “table” is used in the remainder of this course.

For example, a database for a bank might contain a table to store currency data, a table to store customer data, and a table for its accounts. These tables are obviously related: a customer can have one or more accounts, and an account can express its balance in a particular currency. However, the existence of these relationships is not a defining characteristic of a relational database. The only reason for a DBMS to be called relational is that it stores its data in tables whether or not these tables are related to one another.

Continuing with the bank example, a hierarchical representation would require a list of customers and, within each customer entry, a list of bank accounts, each of which uses a different currency. To list all currencies used in the database, you must first examine all the customers and then, for each customer, you must look at all accounts. The relational model makes it much easier to do this.

## Spreadsheet Versus Database

- Spreadsheets are easy to use and provide an instant view of the data.
  - Focus on numerical data and manipulation
  - Allow any data to be entered into a cell
  - Can display data as graphs and charts
  - Format directly into printable reports
- Databases are much more powerful than spreadsheets due to their ability to query and manipulate data:
  - Retrieve all records that match certain criteria
  - Cross-reference records between tables
  - Update records in bulk
  - Perform complex aggregate calculations

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Entities and Relationships

- **Entities:** Things in the real world that you store information about in a database
  - Tables store data representing one type of entity.
  - Example:
    - A bank database has a `customer` table to store customer information.
    - The `customer` table stores this information as a set of attributes (columns within the table) for each customer.
- **Relationships:** Links between entities that have something to do with each other
  - Example:
    - The customer name is related to the customer account number and contact information, which might be in the same table.
    - There can also be relationships between separate tables (for example, `customer` to `accounts`).

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you talk about storing information about customers, bank accounts, currencies, and other real-world entities in your database, what you really mean is that you are storing data that represents these things.

In a relational database, one table stores data about one particular type of entity. In the bank example, the `customers` table stores data about the bank's customers.

# Relationship Categories

Relationship categories and examples:



Relationship between one car and one engine



Relationship between many animals and one zoo



Relationship between one bank customer and many accounts



Relationship between many students and many courses

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can categorize relationships according to how many entities can be involved at either end of the relationship.

In the slide example of a one-to-one relationship, a car has one engine, and the engine can belong to just one car. There is a one-to-one relationship between the car and the engine. Often these types of relationships exist within a single table, but not always.

In the slide example of a one-to-many relationship, an individual customer of a bank can have more than one account. Note that this allows for customers to have zero, one, or more bank accounts. The word “many” refers to the maximum number of possible bank accounts for each customer.

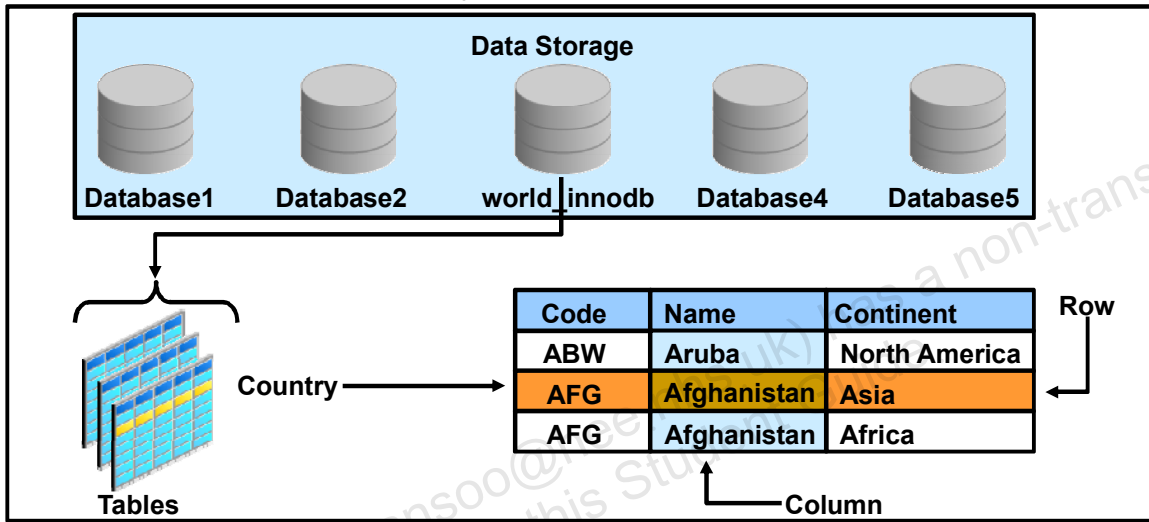
An example of a many-to-one relationship is between bank branches and customers. Many customers could belong to one branch.

There is a many-to-many relationship between students and courses. A student can attend more than one course and a course can be attended by many students.



# RDBMS Database Structure

- Two-dimensional tables, columns, and rows:
  - An ordered set of columns (1 or more)
  - An unordered set of rows (0 or more)
- Each row has exactly one value in each column.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Row-Column Relationship

- A row is a collection of data items that describes an entity (such as a country).
- A column represents one particular piece of information about an entity; for example, the name and continent columns in the country table. Each column has a name and a data type and there are usually several columns in a table.
- The intersection of row and column contains individual data items called values. Values can only contain one piece of data. In this way, a table is like a spreadsheet.

The relational model requires that each row is unique: no two entities can be identical or you would not be able to distinguish one from another. There must be some combination of columns (possibly, all columns) called a key, whose values uniquely identify each row. In practice, most RDBMSs are less strict. Nearly all allow a table to contain duplicate rows, and to exist without a key.

**Note:** In SQL, tables do not have to have primary keys. In other words, the relational model does not enforce this requirement in SQL. However, it is enforced as soon as a primary key is added to the table. This course covers primary keys in Lesson 6, "Database and Table Creation".

# Using SQL with Databases

SQL is:

- **Structured Query Language**
- A standardized language
- The “core” of the relational database
- Based on the English language (phrases)

Categories of phrase types:

- DDL: Data definition language
- DML: Data manipulation language

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL provides an easy, efficient way for users to add, delete, modify, and retrieve the data in their databases by using:

- The SQL language
- A powerful server
- Flexible program connections

SQL is a standardized language for communicating with relational databases. The majority of database vendors have add-ons and extensions to the syntax, but most SQL dialects provide a reasonably adequate subset of the features defined in the standard. The language is designed to mimic the English language and most commonly uses phrases instead of individual words. These phrases fall into two different categories, which are discussed next.

# SQL Statements: Data Definition Language

- Create and delete database:

```
CREATE DATABASE
DROP DATABASE
```

– DATABASE is synonymous with SCHEMA.

- Create and delete database table:

```
CREATE TABLE
DROP TABLE
```

- Modify databases and tables:

```
ALTER DATABASE
ALTER TABLE
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DDL SQL statements define database and table data. The statements shown in the slide are followed by options to specify data operations.

For example, to create a bank database with a table for customer information, you can enter the following statements:

```
CREATE DATABASE mybank
CREATE TABLE customers
```

Because SQL is based on phrases, the order of words is important. For example, CREATE mybank DATABASE and DATABASE mybank CREATE are not correct SQL statements. Only CREATE DATABASE mybank is correct.

In the CREATE DATABASE and DROP DATABASE statements, SCHEMA can be used instead (CREATE SCHEMA/DROP SCHEMA).

# SQL Statements: Data Manipulation Language

- Retrieve data:

```
SELECT ... FROM
```

- Add new data:

```
INSERT INTO
```

- Modify data:

```
UPDATE
```

- Remove data:

```
DELETE ... FROM
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DML SQL statements are used to manipulate table data. Options (such as table name) follow the statements shown in the slide.

For example, to select all the data from the `customer` table, enter the following statement:

```
SELECT * FROM customers
```

**Note:** This course covers the specifics of DDL and DML statements in later lessons.

## MySQL: Benefits

The following are some of the benefits of MySQL:

- Supports the SQL “dialect,” which is a powerful extension of the standard SQL language
- Runs on many different operating systems
- Has a flexible and secure authorization system
- Supports very large databases
- Is highly customizable
- Is built for speed
- Is free or inexpensive
- Provides open source and commercial licensing options
- Has Enterprise tools
- Provides exceptional support

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Summary

In this lesson, you learned how to:

- Define a relational database (and RDBMS)
- Describe the structure of an RDBMS database
- Identify the differences between databases and spreadsheets
- Explain the use of SQL and MySQL with relational databases
- Define and use data definition language (DDL)
- Define and use data manipulation language (DML)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 3-1 Overview: Quiz – Database Basics

In this practice, you answer questions about:

- Database basics
- MySQL-specific SQL and RDBMS topics

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## Practice 3-2 Overview: Identifying the Structure of a Table

In this practice, you identify the relational database characteristics of a given table.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# 4

## Database Design

### Table Design

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Describe database modeling
- Explain how to use keys to identify a row
- Explain how to use foreign keys to achieve referential integrity
- Define and perform normalization
- Prepare a design plan for a new database
- View a database structure
- Evaluate a database design

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Database Modeling

Database modeling is the process of defining the logical structure of a database. This structure determines how data is stored, organized, and manipulated.

- It encourages you to understand your data before building the database.
- Consider what questions your data needs to answer.
- The entity relationship model (ERM) is the most common model for an RDBMS:
  - Top-down method
  - Used to produce an entity relationship diagram (ERD)

ORACLE

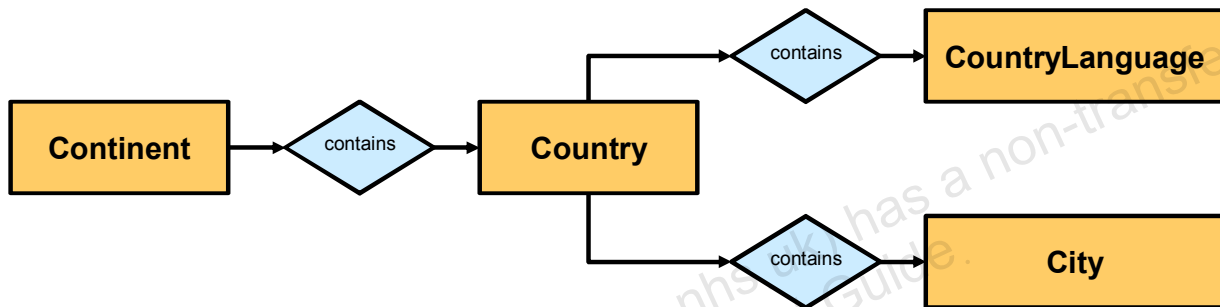
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

An entity relationship model describes a relational database by using a “top-down” method. This method starts with a high-level view, and then adds details as you refine the process. The result is a diagram called an entity relationship diagram (ERD). There are several types of ERDs. You use different types at different stages of the process and for different reasons.

**Note:** The information provided here gives a high-level, conceptual view of database modeling. This is a big topic that could easily be a course on its own.

## ERD: Structure Diagram

- Visualizes the content and relationships of a database
- Organizes data into groups of similar entities (to be placed in tables)
- Example: High-level diagram of the `world_innodb` database:



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## ERD: Cardinality Diagram

- Indicates the type of relationship between tables
- Uses lines and specific symbols (notation) to show these relationships
- There are several ERD notations
- Example:

- Zero or one
- || Exactly one (one and only one)
- Zero or many
- ⌞ One or many

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The type of notation shown in the slide is called “crow's foot” notation, and is very common. Others include Barker, IDEF1X and Unified Modeling Language (UML).

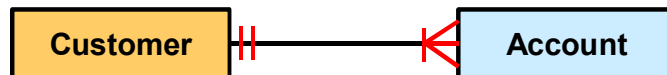
To be precise, the “zero or many” symbol can also mean “zero or one or many”.

## Cardinality Diagram: Examples

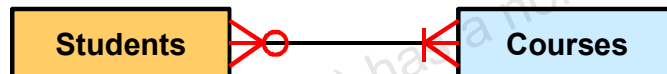
- **One-to-one:** One car can have only one engine.



- **One-to-one (or many):** A bank customer can have one or more accounts.



- **Zero (or many)-to-one (or many):** Zero or many students can take one or many courses.



- **Zero (or one)-to-zero (or many):** Zero or one person can rent zero or many DVDs.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To complete the other side of the cardinality for each relationship:

- **One-to-one:** One engine can be installed in only one car at a time.
- **One-to-one (or many):** One or many accounts must be assigned to one bank customer.
- **One (or many)-to-zero (or many):** One or many courses can be taken by zero or many students.
- **Zero (or one)-to-zero (or many):** Zero or many DVDs can be rented by zero or one renter at a time.

# Keys

- Candidate key:
  - A column that can uniquely identify a row
    - There can be multiple candidate keys for a table.
- Primary key:
  - The chosen candidate key column(s)
    - One primary key per table
    - Prevents duplicate values
    - Cannot contain null values
    - Can help to decrease lookup time for large tables
- Foreign key:
  - A column that matches a candidate key in another table, to create a relationship between the tables
    - Enforces referential integrity

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

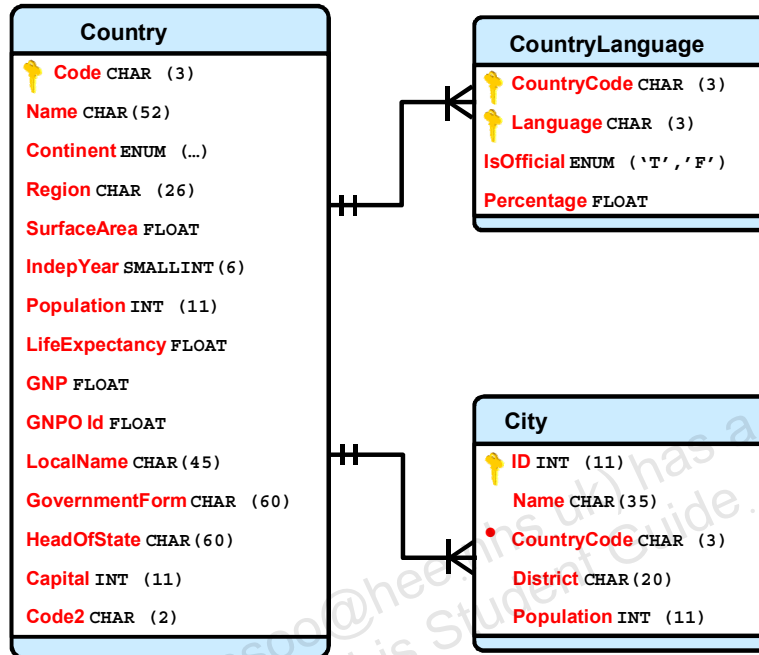
A table consists of several rows, each of which corresponds to a record. A key is a unique identifier for each record. The database uses keys to organize records.

A primary key can be a single column, or several columns in a “composite” primary key.

Referential integrity requires that the foreign key in any referencing table must always refer to a valid row in the referenced table. Referential integrity ensures that data in related tables remains consistent during updates and deletes.

# Cardinality ERD: Example

world\_innodb database ERD, showing keys



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A primary key is a single column (for example, ID in the City table) or a combination of columns (CountryCode + Language in the CountryLanguage table). A table has only one primary key.

The world\_innodb ERD shows a relationship between the Country and CountryLanguage tables. The CountryCode column in the CountryLanguage table is being used as a foreign key referring to the Country table's primary key (Code).

The Country and City tables are also related. The CountryCode column in the City table is used as a foreign key referring to Code in the Country table.

The lines and symbols between the tables describe their relationships. The double bar notation on the Country table side means "one and only one." The bar and crow's foot combination on the referencing tables (CountryLanguage and City) means "one or many."

In the Country/CountryLanguage relationship, a country has only one code. The code links the country to one or more national languages. In the Country/City relationship, the country code associates a country with one or many cities.

**Note:** You will do more with ERDs later in the course.



# Normalization

Normalization is the process of organizing data to avoid duplication and redundancy.

- Ensures that each individual data item is stored only once
- Prevents invalid or inconsistent data after modification
- “Decomposes” database into smaller tables
- Stops one column from containing multiple values
- Avoids repeating groups (“tables within tables”)
- Helps with good database design

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Normalization is a technique used to create well-designed relational databases. A normalized database gives you a lot of flexibility when accessing your data and stops modifications from making the data inconsistent.

Repeating groups and columns containing more than one value lead to “tables within tables”. You cannot access the inner table without first accessing the outer table. This is a characteristic of hierarchical databases, which the relational model is designed to avoid.

Although this course focuses on the use of SQL to create and use databases, even the best-written SQL code cannot fix a badly-designed database. Spend the necessary time and effort on proper database modeling and normalization before building your database.

**Note:** Normalization is a big topic. This course provides a high-level view.

# Advantages of Normalization

- Better database organization:
  - More tables with smaller rows
  - Efficient data access
  - Greater flexibility for queries
  - Ability to quickly find the information you need
  - Easier to implement security
  - Easier to maintain as your needs change
- Reduction of redundant and duplicate data:
  - More compact database
  - Easier to ensure consistent data after modification

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Disadvantages of Normalization

Despite the numerous advantages of normalization, consider these disadvantages:

- Even simple queries often need access to several tables.
- Tables often contain codes rather than the real data.
  - Need to look up codes in related tables
  - Optimized for applications, not general querying
- Join operations can reduce database performance.
- The normalization process is complex and time-consuming.
- De-normalization results in unusable data.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Eliminating Data Inconsistencies

Normalization avoids insertion, deletion, and modification problems by ensuring that each column in a table:

- Is relevant to the entity that the table represents
- Contains a single value
- Contains the smallest amount of useful information
- Does not contain a calculated or concatenated value
- Is unique within the database
- Is identical to the key it refers to in a table relationship

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Avoid using table columns that might lead to data inconsistencies. Ensure that each column:

- **Is relevant to the entity the table represents:** For example, a table representing a dinner menu would not include shoe sizes of the guests.
- **Contains a single value:** If a column contains more than one value, that value is harder to work with and potentially redundant.
- **Contains the smallest amount of useful information:** For example, a column containing an address. Divide this “multipart” column into separate columns for building number, street name, city, postal code, and so on. This allows the component parts to be filtered, sorted, and otherwise manipulated easily.

- **Does not contain a calculated or concatenated value:** Columns must not derive their value from other column data. These derived values need to be updated every time the columns they depend on are updated. Instead, create the concatenated or derived values dynamically as required.
- **Is unique within the database:** If a data item is duplicated, there is the risk that it is updated in one place but not another, leading to data corruption.
- **Is identical to the key it refers to in a table relationship:** The primary and foreign key columns must be of the same data type.

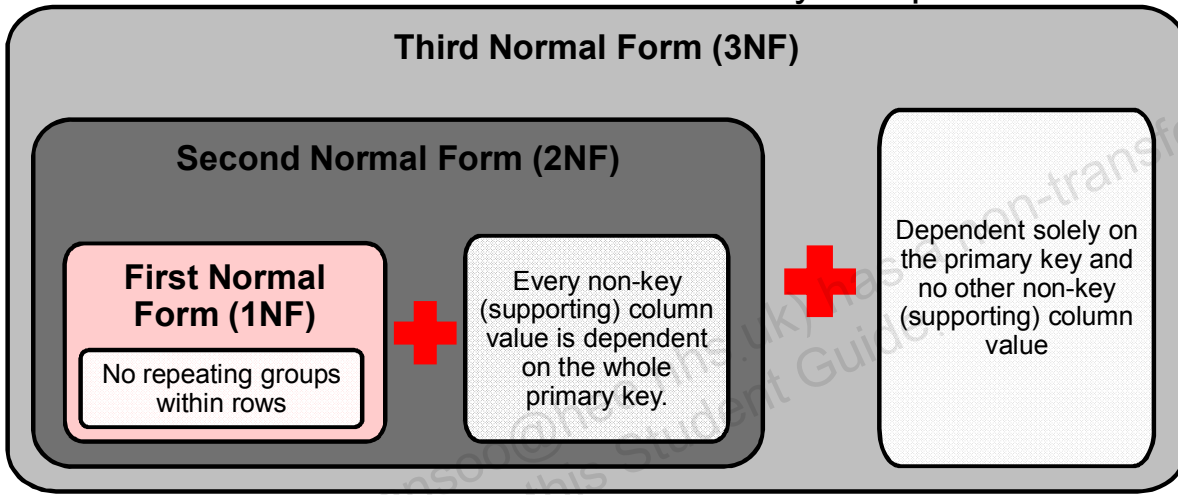
**Note:** Columns are sometimes referred to as “fields” in MySQL, and the term can be confusing if you have worked with other RDBMS.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Normal Forms

There are many successive levels of normalization. These are called **normal forms**.

- Each consecutive normal form depends on the previous one.
- The first three normal forms are usually adequate.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Start with the first normal form and build upon it. The first three normal forms are adequate for most databases.

- First normal form (1NF): A table in the first normal form:
  - Has columns containing a single value only
  - Does not have repeating groups within rows
- Second normal form (2NF): A table is in the second normal form if:
  - It is already in 1NF
  - All its non-key attributes depend on the primary key (and on every column in a composite primary key)

A table is already in 2NF if it has a single column primary key.

- Third normal form (3NF): A table is in third normal form if:
  - It is already in 2NF
  - All its non-key attributes are not dependent on any non-key column values

# Normalization Process: Example

Inventory database for a chain of furniture stores:

- Turn the inventory spreadsheet into a database, so you can:
  - Execute intelligent searches
  - Make modifications
  - Prepare for future growth of your business
- Original inventory spreadsheet:

Inventory									
sID	sLoc	sPostal	pID1	pName1	pQty1	pID2	pID	pName2	pQty2
1	Holtsville	00501	1	bed	15	2	2	chair	4
2	Waukesha	53146	1	bed	4	3	3	table	6
3	Waukesha	53146	2	chair	8	4	4	sofa	4
4	Ketchikan	99950	2	chair	24	4	4	sofa	10

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Imagine a chain of furniture stores that stores its inventory in a spreadsheet and wants to turn that spreadsheet data into a MySQL database.

The slide shows the original inventory spreadsheet. It contains information about stores (spreadsheet columns prefixed by “s”) and the items held in those stores (prefixed by “p”). This spreadsheet is the basis for the next several slides, which demonstrate the normalization process.

# First Normal Form: 1NF

Inventory								
sID	sLoc	sPostal	pID1	pName	pQty1	pID2	pName2	pQty2
1	Holtsville	00501	1	bed	15	2	chair	4
2	Waukesha	53146	1	bed	4	3	table	6
3	Waukesha	53146	2	chair	8	4	sofa	4
4	Ketchikan	99950	2	chair	24	4	sofa	10



Inventory_1NF					
sID	sLoc	sPostal	pID*	pName	pQty
1	Holtsville	00501	1	bed	15
1	Holtsville	00501	2	chair	4
2	Waukesha	53146	1	bed	4
2	Waukesha	53146	3	table	6
3	Waukesha	53146	2	chair	8
3	Waukesha	53146	4	sofa	4
4	Ketchikan	99950	2	chair	24
4	Ketchikan	99950	4	sofa	10

This column would not usually appear in a first normal form; however, it is here to demonstrate a connection to the second normal form.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the inventory spreadsheet, the items represented by pID1 (a bed) and pID2 (a chair) are both in the same row. This is a “repeating group” and repeating groups cause problems.

For example, you cannot remove item pID1 from the Holtsville store without removing the entire row. This deletes information about pID2, too (removing the bed also removes the chair).

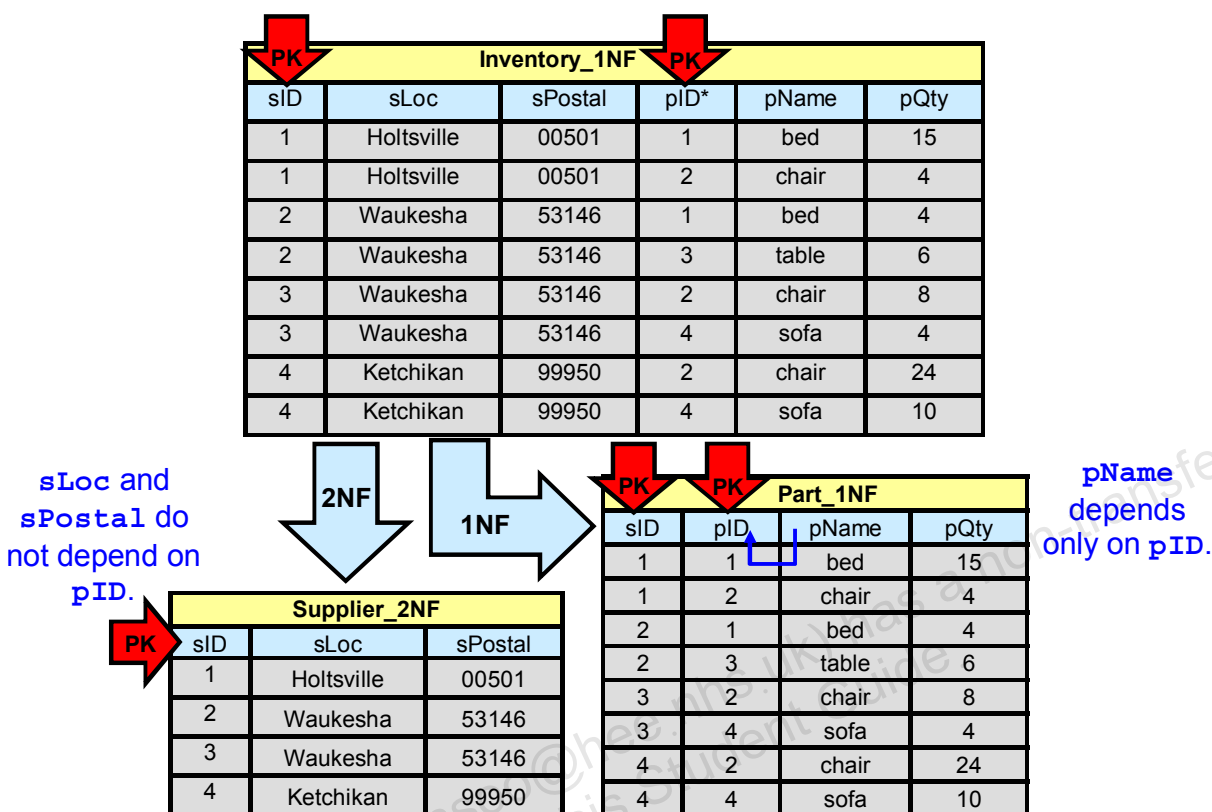
To prevent this, take the wide spreadsheet (with its nine columns), and turn it into a narrower table (with six columns). Now each row contains information about only one item of furniture. In the Inventory\_1NF table, the chair remains if you remove the bed.

The Inventory\_1NF table has a composite primary key made up of the sID and pID columns. There is no single column that can be used as a primary key to avoid repeating groups, so redundancy can still exist. Redundancies can result in inconsistent data, so this table should be put through second normal form.

**Note:** Successful transformation to a normal form never results in loss of data. It only involves re-organization of data.



## Second Normal Form: 2NF



ORACLE

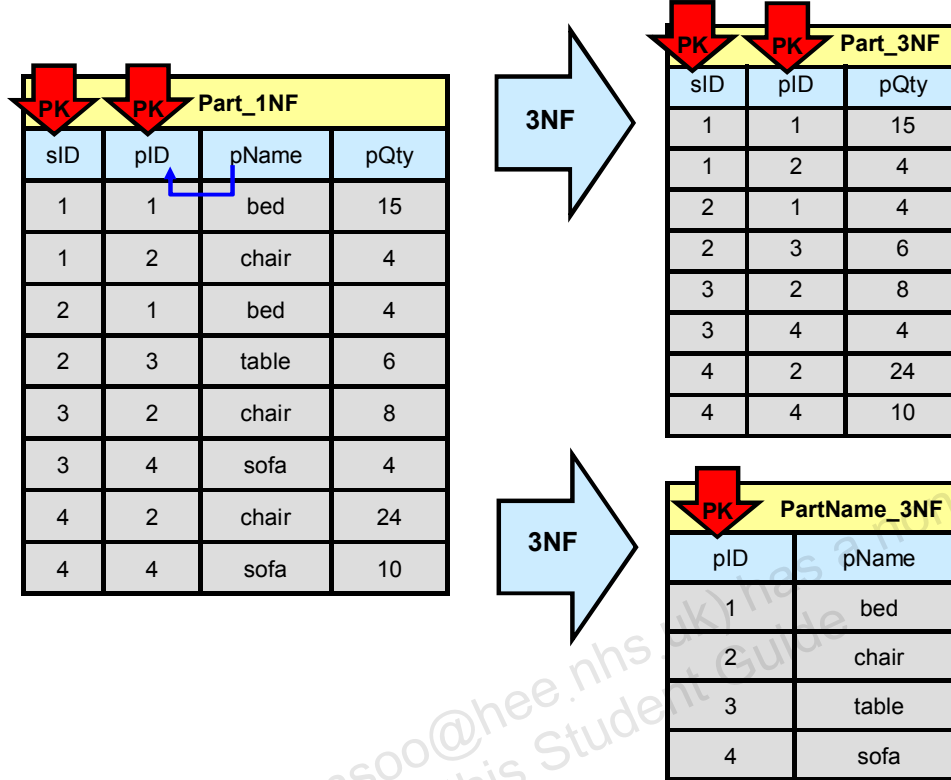
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `Inventory_1NF` table has a composite primary key made up of the `sID` and `pID` columns. Every non-key (supporting) value depends on the primary key value. The goal for second normal form is to remove any non-key attributes that only depend on part of this composite key (that is, `sID` or `pID` but not both) to a new table.

`Inventory_1NF` has been split into two tables, `Supplier_2NF` and `Part_1NF`:

- `Supplier_2NF`: Has one primary key (`sID`) with two supporting columns (`sLoc` and `sPostal`), which depend on the primary key. This table is in second normal form.
- `Part_1NF`: Has a composite primary key made up of `sID` and `pID`. The supporting column `pName` does not depend on both parts of the primary key (it is dependent only on `pID`), so the table is still in first normal form.

# Third Normal Form: 3NF



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `Part1_1NF` table contains columns that are not completely dependent on both parts of the composite primary key. This can cause problems with updates and insertions:

- **Updates:** Consider the “chair” record where `sID` is 1 and `pID` is 2. If you change its `pName` value to “armchair,” the other chair records still have a `pName` of “chair.”
- **Insertions:** If you add a new furniture item to the inventory with a `pID` of 5 and a `pName` of “loveseat,” `pQty` and `sID` will not have values.
- There are now three different suppliers, so the `sID` and `pQty` columns need to have multiple rows.

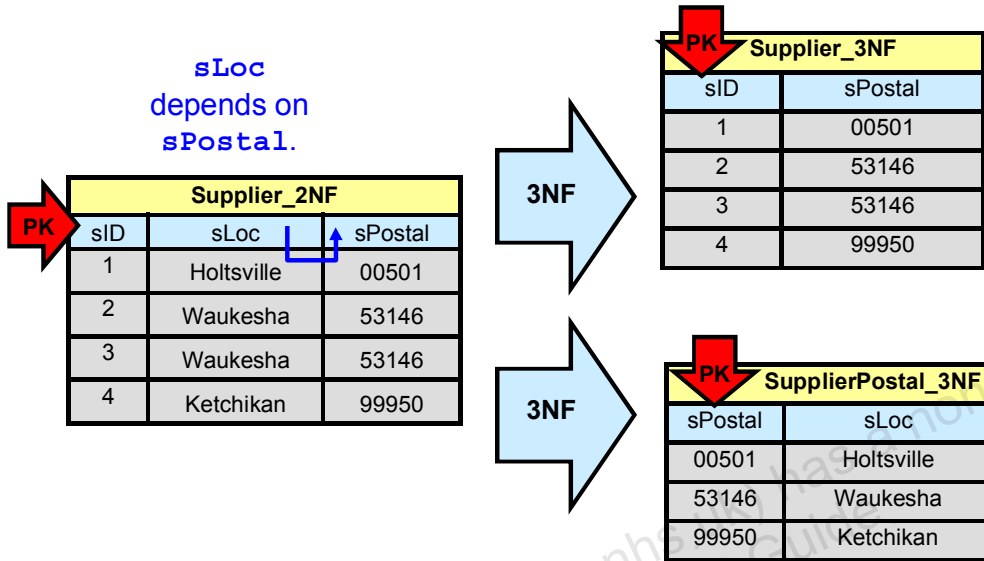
These problems occur because `pName` is dependent only on `pID` and not `sID`.

Third normal form removes columns that are not dependent on both parts of the composite key and puts them into separate tables.

Because of the simplicity of the `Part_1NF` table, the second normal form can be passed over and the third normal form applied.

The two resulting tables (`Part_3NF` and `PartName_3NF`) meet all the criteria of the third normal form. Now if you change chair to armchair, it is changed everywhere consistently.

# Third Normal Form: 3NF



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

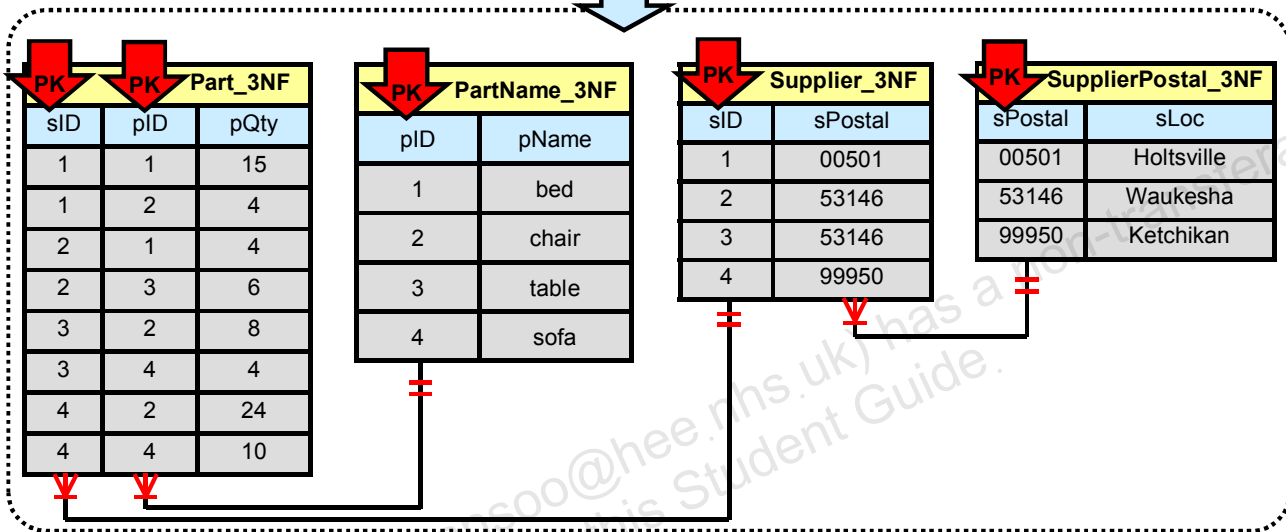
In the `Supplier_2NF` table, `sLoc` depends on both `sID` (the primary key) and `sPostal`. This is because there are two suppliers in the same postal zone (`sIDs` 2 and 3). The `sPostal` column is not a primary key, so the table is not in third normal form.

The `Supplier_2NF` table is normalized into two separate tables (`SupplierPostal_3NF` and `Supplier_3NF`), making future updates less problematic.

To help distinguish between different suppliers in the same postal zone, add an extra column to the `Supplier_3NF` table for supplier name (`sName`). This makes the table easier to understand, because most people think in terms of names and not numbers. However, if you want the most efficient table, you would not do this.

# Normalized "Furniture Stores" Database

Inventory								
sID	sLoc	sPostal	pID1	pName1	pQty1	pID2	pName2	pQty2
1	Holtsville	00501	1	bed	15	2	chair	4
2	Waukesha	53146	1	bed	4	3	table	6
3	Waukesha	53146	2	chair	8	4	sofa	4
4	Ketchikan	99950	2	chair	24	4	sofa	10



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This slide shows the ERD resulting from the original spreadsheet after normalization. The database now consists of four tables, which have been sufficiently normalized.

The process has ensured that insertion, modification, deletion, and querying of data is efficient and leaves the database in a consistent state.

## Quiz

The first three normalization forms are cumulative and build on each other, starting with the first normal form (or 1NF).

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

## Database Design Considerations

- Always design your database before creating it, even if you do not use formal methods.
- Give tables and columns sensible names.
- Use normalization to organize data in a new database.
  - Or evaluate and improve the design of an existing database.
  - Normalize only to the point where it is useful.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Database Design Plan

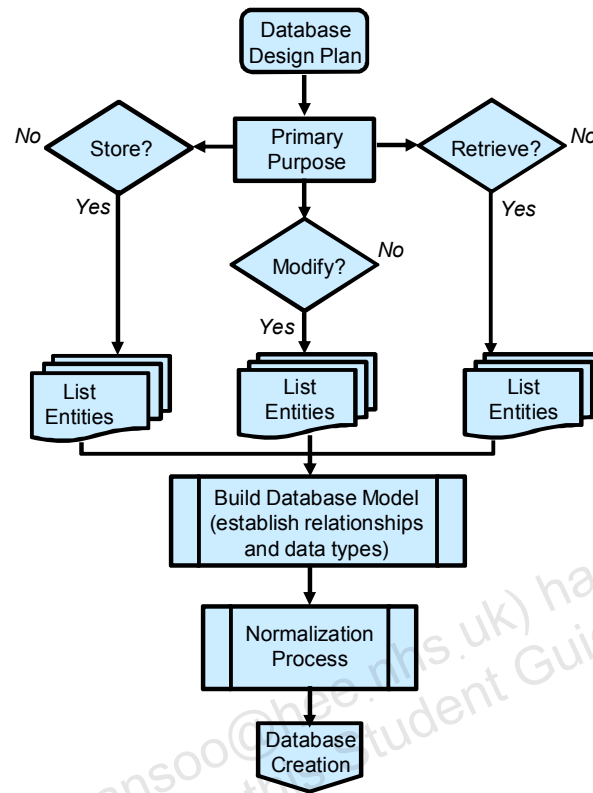
To start the design process, ask these questions about your database:

- What is the primary purpose of this database?
- What entities need to be stored, retrieved, and/or modified?
- How do those entities relate to each other?

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Database Design Plan Diagram



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Database design plan steps:

- Determine the purpose of your database.
  - Start a structure diagram for a visual representation of your design plan.
  - Identify the key entities to be managed by the database. List (or draw) a table for each key entity.
  - List or draw the attributes of each entity. These will become columns.
  - Consider which columns could uniquely identify a record. These are your candidate keys. (You might require an additional column specifically for this purpose.)
  - Determine the storage requirements for each column. Consider whether to use fixed or variable length types because this determines which storage engine you use.
    - For InnoDB (the default) use variable data types.
    - For MyISAM, use fixed data types.
    - Use bit data type when flags are needed.
- Note:** Data types are covered in detail in the “Table Data Types” lesson.
- Normalize to the third normal form (or higher).

Continue to refine your database after construction as required.



## Viewing a Database

Issue the following SQL statements from the `mysql` client:

- `SHOW DATABASES`
- `USE <database_name>`
- `SHOW TABLES`
- `DESCRIBE <table_name>`
- `SELECT * FROM <table_name>`

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Evaluating a Database Design

1. Connect to the database server:

```
shell> mysql -uroot -p  
Enter password: <password>
```

2. Display all databases/schemas:

```
mysql> SHOW DATABASES;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| test |  
| world_innodb |  
+-----+
```

3. Change the default database/schema:

```
mysql> USE world_innodb  
Database changed
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Evaluating a Database Design

## 4. Display all tables in the database/schema:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_world_innodb |
+-----+
| city                    |
| country                 |
| countrylanguage        |
+-----+
```

## 5. Display all the columns in the specified table:

```
mysql> DESCRIBE City;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| ID         | int(11)   | NO   | PRI | NULL    | auto_increment |
| Name      | char(35)  | NO   |     |         |               |
| CountryCode | char(3)   | NO   |     |         |               |
| District   | char(20)  | NO   |     |         |               |
| Population | int(11)   | NO   |     | 0       |               |
+-----+-----+-----+-----+-----+-----+

```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

DESCRIBE returns the following information for a table:

- **Field:** Column name
- **Type:** Column data type
- **Null:** Contains YES if NULL values can be stored in the column; NO if not
- **Key:** Indicates whether the column is indexed:
  - **PRI:** The column is the primary key, or one of the columns in a composite primary key.
  - **MUL:** The column is the first column of a non-unique index or a unique-valued index that can contain NULL values.
- **Default:** Default value that is assigned to the column
- **Extra:** Contains additional information for a column. The example in the slide shows `auto_increment`, which means that this column was created with the `AUTO_INCREMENT` keyword.

For more information about the `DESCRIBE` statement, see the MySQL Reference Manual:  
<http://dev.mysql.com/doc/refman/5.6/en/show-columns.html>

## Evaluating a Database Design

### 6. Display all the data in the specified table:

```
mysql> SELECT * FROM City;
```

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
...				
4079	Rafah	PSE	Rafah	92020

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Repeat steps 5 and 6 for the `Country` and `CountryLanguage` tables. Note the differences in structure and content.

# Summary

In this lesson, you learned how to:

- Describe database modeling
- Explain how to use keys to identify a row
- Explain how to use foreign keys to achieve referential integrity
- Define and perform normalization
- Prepare a design plan for a new database
- View a database structure
- Evaluate a database design

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 4-1 Overview: Quiz – Database Design

In this practice, you answer questions about database design.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 4-2 Overview: Evaluating a Database

In this practice, you evaluate the `world_innodb` database.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 4-3 Overview: Creating a Structure Diagram

In this practice, you create a complete structure diagram for the `world_innodb` database, including all its tables and columns.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# 5

## Table Data Types

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Explain the use of data types in database design
- List the four MySQL data type categories
- Choose the correct data types for columns
- Describe the special considerations for using data types
- Explain the use of `NULL`

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Data Types as Part of Database Design

A well-designed database uses the most suitable data type for each data item.

- Think about the data each column needs to store.
- Assign column data types accordingly:
  - Example: `PartName_3NF` in the `Inventory` database. The `pName` column contains character data and the `pID` column contains numeric data.
- Follow the ABCs of data types:
  - **A**ppropriate
  - **B**rief
  - **C**omplete

PartName_3NF	
pID	pName
1	bed
2	chair
3	table
4	sofa

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ensure that a column's data type is appropriate, brief, and complete:

- **Appropriate:** Choose the most suitable data type for the data the column contains.
- **Brief:** Choose the data type that uses the least amount of storage space. This conserves resources and improves performance.
- **Complete:** Ensure that the data type can store the largest possible value for each column without data loss.

Within each category, there are numerous sub-types that use varying amounts of memory and disk space and affect performance. This can be negligible at the record level, but significant in large databases. Make the right decisions early in the design process to avoid performance issues later on.

# Data Type Categories

- **Numeric:** Numeric values
- **Temporal:** Time and dates
- **Character:** Text strings
- **Binary:** Binary data strings

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For more information about data types, see the MySQL Reference Manual:  
<http://dev.mysql.com/doc/refman/5.6/en/data-types.html>

# Numeric Data Types

- Factors to consider with numeric data types:
  - Range of values to be stored
  - Amount of storage space required
  - Column precision and scale (floating-point and fixed-point)
- Classes of numeric data types:
  - Integer: Whole numbers
  - Floating-point: Approximate-value fractional numbers
  - Fixed-point: Exact-value fractional numbers
  - Bit: Bit-field values

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL uses the following data types to store numeric data:

- **Integer:** Integers are whole numbers without a fractional part, that is, nothing after the decimal point.
- **Floating-point:** Approximate value numbers that have an integer part, a fractional part, and an exponent. This data type class represents values in the native binary floating-point format (IEEE 754) used by the server host's CPU. This type is very efficient for storage and computation, but values are subject to rounding errors.
- **Fixed-point:** Exact value numbers that have an integer part, a fractional part, or both.
- **Bit:** Binary data only, from 1 to 64 bits.

Precision and scale are terms that apply to floating-point and fixed-point values, which can have both an integer part and a fractional part.

- **Precision:** The number of significant digits.
- **Scale:** The number of digits to the right of the decimal point.

# Numeric: Integer Data Types

- Data types:
  - **TINYINT**: Very small integer data type
  - **SMALLINT**: Small integer data type
  - **MEDIUMINT**: Medium-sized integer data type
  - **INT, INTEGER**: Normal (average) size integer data type
  - **BIGINT**: Large integer data type
- Example:
  - world\_innodb database, City table, Population column: **Population INT(11)**
  - Largest value stored is 10500000 (uses 8; 11 allowed)
- Attributes:
  - **UNSIGNED** no negative numbers
  - **ZEROFILL** replaces padding spaces with zeroes

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Assign integer data types with the following syntax:

```
INT [(M)] [UNSIGNED] [ZEROFILL]
```

- M is the maximum display width for integer types. The maximum legal display width is 255. Display width is unrelated to the range of values a type can contain.
- M is required only when using the **ZEROFILL** attribute.

All integer types can use the optional attribute **UNSIGNED**. Use it when you are certain you do not need to store negative numbers in a column and need a larger upper numeric range for that column. For example, if an **INT** column is **UNSIGNED**, the range of possible values changes from -2147483648 to 2147483647 up to 0 to 4294967295.

When you use an integer data type with the optional attribute **ZEROFILL**, the default padding of spaces is replaced with zeroes. For example, if you declare a column as **INT(5) ZEROFILL**, a value of 4 is retrieved as 00004.

## Note

- Combining unsigned and signed numbers in expressions might cause conversion problems.
- If you specify **ZEROFILL** for a numeric column, MySQL automatically adds the **UNSIGNED** attribute to that column.

## Numeric: Integer Data Types Comparison

Type	Storage Required	Signed Range	Unsigned Range
<b>TINYINT</b>	1 byte	-128 to 127	0 to 255
<b>SMALLINT</b>	2 bytes	-32,768 to 32,767	0 to 65,535
<b>MEDIUMINT</b>	3 bytes	-8,388,608 to 8,388,607	0 to 16,777,215
<b>INT</b>	4 bytes	-2,147,683,648 to 2,147,483,647	0 to 4,294,967,295
<b>BIGINT</b>	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0 to 18,446,744,073,709, 551,615

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Numeric: Floating-Point Data Types

- Used for approximate-value numbers: Integer, fractional, or both
- Data types:
  - **FLOAT**: Very small numeric data type
  - **DOUBLE**: Small numeric data type
- Can declare with precision and scale
- Can specify an exponent
- Example:
  - world\_innodb database, Country table, GNP column:  
**GNP FLOAT(10,2)**
  - Largest value stored is 8510700.00 (uses 7, 10 allowed, 2 decimal places):

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A single precision floating-point value (a **FLOAT**) is accurate to approximately seven decimal places. Assign with the following syntax:

**FLOAT** (M, D)

- M is the maximum number of decimal digits available.
- D is the number of digits to the right of the decimal point.

The storage requirement for **FLOAT** values is 4 bytes.

A double precision floating point value (a **DOUBLE**) is accurate to approximately 15 decimal places. Assign with the following syntax:

**DOUBLE** (M, D)

- M is the maximum number of decimal digits available.
- D is the number of digits to the right of the decimal point.

**Note:** For both **FLOAT** and **DOUBLE** the M and D options do not affect the accuracy of the stored numbers, but how many significant digits are displayed. Floating-point values are also subject to platform or implementation dependencies. For maximum portability, code requiring storage of approximate numeric data values should use **FLOAT** or **DOUBLE** with no specification of precision or number of digits. See the MySQL Reference Manual for issues with floating-point data types: <http://dev.mysql.com/doc/refman/5.6/en/problems-with-float.html>



## Numeric: Floating-Point Data Types Comparison

Type	Storage Required	Signed Range	Unsigned Range
<b>FLOAT</b>	4 bytes	-3.402823466E+38 to -1.175494351E-38 to 1.175494351E-38 to 3.402823466E+38	0 and 1.175494351E-38 to 3.402823466E+38
<b>DOUBLE</b>	8 bytes	-1.7976931348623157E+308 to -2.2250738585072014E-308 and 2.2250738585072014E-308 to 1.7976931348623157E+308	0 and 2.2250738585072014E- 308 to 1.7976931348623157E +308

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Numeric: Fixed-Point Data Types

- Used for exact-value numbers: Integer, fractional, or both
- Data types:
  - **DECIMAL** (also **NUMERIC**):
    - Fixed-decimal, binary storage format
    - Preserve exact precision
- Example:
  - To represent currency values, such as dollars and cents:  
`cost DECIMAL(10,2)`
  - Example value output:  
650.88

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

All values in a **DECIMAL** column have the same number of decimal places. **DECIMAL** values are stored exactly as they appear, when possible. **DECIMAL** values are not processed as efficiently as **FLOAT** or **DOUBLE** values, (which use the processor's native binary format), but are not subject to rounding errors, so are more accurate.

The **DECIMAL** data type is a popular choice for financial applications involving currency calculations, because accuracy is important.

Assign using the following syntax:

**DECIMAL** (M, D)

- M is the number of digits that can be stored (precision: up to 65). Any attempt to store a number with greater precision normally results in truncation of the value (depending on the operating system).
- D is the number of digits following the decimal point (scale). It must be less than or equal to the precision, and cannot be larger than 30.

## Numeric: Bit Data Types

- Data type:
  - **BIT**
- Bit-field values:
  - Column Width (**m**) is number of bits per value
  - 1 to 64 bits
- Example:
  - Storing 4 and 20 bits:  
`bit_col1 BIT(4)`  
`bit_col2 BIT(20)`
- Can assign values using Numeric expressions

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For a `BIT(n)` column, the range of values is 0 to  $2^n-1$ , and the storage requirement is approximately  $\text{INT}((n+7)/8)$  bytes per value.

You can assign `BIT` column values using numeric expressions. To write literal bit values in binary format, use the literal value notation `b'val'`, where `val` is a value consisting of the binary digits 0 and 1. Example:

- `b'1111' = 15`
- `b'1000000' = 64`

**Note:** A general rule for storage size is  $n=8$  occupies 1 byte.

# Temporal Data Types

- **TIME:**
  - HH:MM:SS as in 12:59:02
- **YEAR:**
  - A four digit (YYYY) as in 1969
- **DATE:**
  - YYYY-MM-DD as in 2013-01-04
- **DATETIME:**
  - YYYY-MM-DD HH:MM:SS as in 2013-01-04 12:59:02
- **TIMESTAMP:**
  - Current date and time as in 2013-01-04 12:59:02
  - **DEFAULT CURRENT\_TIMESTAMP**

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL provides several different data types for storing date and time data (temporal data). Here, YYYY, MM, DD, HH, MM, and SS stand for year, month, day of month, hour, minute, and second, respectively.

## TIME

- Can express time of day or a duration
- Range of values allowed is -838:59:59 to 838:59:59. (The standard compliant range of values for TIME is 00:00:00.000000 to 23:59:59.999999. Allowing values outside of that range is a MySQL extension).
- Displayed as HH:MM:SS
- TIME values can be assigned by using either strings or numbers.
- The storage requirement for a TIME value is 3 bytes.

## YEAR

- The YEAR data type is assigned by using the following syntax: YEAR [(4)].
- The range of values allowed is 1901 to 2155, and 0000. MySQL retrieves YEAR values in YYYY format.

#### DATE

- The range permitted is 1000-01-01 to 9999-12-31.
- Values can be assigned as strings or numbers. For example, January 9, 2014 can be represented as 2014-01-09.
- Storage requirement is 3 bytes.

#### DATETIME

- A combination of date and time
- The range of values allowed is 1000-01-01 00:00:00 to 9999-12-31 23:59:59.
- Can include fractions of seconds: 2014-12-11 23:59:59.542545
- Values can be assigned as strings or numbers.
- Storage requirement is 8 bytes.

#### TIMESTAMP

- A special `DATETIME` value used to store the current date and time
- The range of values allowed is 1970-01-01 00:00:01.000000 to 2038-01-19 03:14:07.999999.
- To obtain the value as a number, add +0 to the time stamp column.
- `TIMESTAMP` values are converted from the current time zone to UTC for storage, and converted back from UTC to the current time zone for retrieval.
- Storage requirement is 4 bytes.
- Display width is fixed at 19 characters.

**Note:** You can optionally extend `TIME`, `DATETIME` and `TIMESTAMP` with a precision between 0 and 6, indicating fractional seconds.

# Temporal Data Types Comparison

Type	Storage Required	Range
<b>DATE</b>	3 bytes	1000-01-01 to 999-12-31
<b>TIME</b>	3 bytes	-838:59:59 to 838:59:59
<b>DATETIME</b>	8 bytes	1000-01-01 00:00:00 to 9999-12-31 23:59:59
<b>TIMESTAMP</b>	4 bytes	1970-01-01 00:00:00 to mid-year 2037
<b>YEAR</b>	1 byte	1901 to 2155

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Character String Data Types

A character string data type:

- Represents a sequence of alphanumeric characters that belong to a given character set
- Stores text or binary data
- Is implemented in nearly every programming language
- Supports character sets and collation
- Belongs to one of the following classes:
  - Text: Unstructured text (freeform strings of characters)
  - Integer: Structured text (a set of predefined strings that you supply)
  - Binary: Stores sequences of bytes

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Character string data types differ in terms of:

- Whether data is stored in a fixed or variable length format
- The maximum length that can be stored
- Whether unstructured string values are allowed

# Character String: Text Class Data Types

- Data types:
  - **CHAR**: Fixed-length (static)
  - **VARCHAR**: Variable-length (dynamic)
  - **TINYTEXT**: Variable-length
  - **TEXT**: Variable-length
  - **MEDIUMTEXT**: Variable-length
  - **LONGTEXT**: Variable-length
- Example:
  - world\_innodb database, CountryLanguage table, Language column:  
**Language CHAR(30)**
  - Largest value (uses 25, 30 allowed):  
Southern Slavic Languages

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The main character string data types are **CHAR** and **VARCHAR**.

**CHAR**:

- Fixed-length: String is stored using a fixed number of bytes.
- Assignment: **CHAR (M)**, where **M** is the length of the character string (0 to 255)
- Unused characters are right-padded with spaces.
- Trailing spaces are removed when **CHAR** values are retrieved.

**VARCHAR**:

- Variable-length: String value and string length stored together
- Assignment: **VARCHAR (M)**, where **M** is the maximum number of characters
- Maximum value for **M** depends on the character set used.
- Can be used to store strings with trailing spaces (unlike **CHAR**)
- Stored as a one- or two-byte prefix plus the actual value itself
  - Prefix part depends on the number of bytes required to store the value (255 bytes or less: prefix is one byte; more than 255 bytes: prefix is two bytes)
  - Value part storage requirement depends on the character set used

**TINYTEXT**, **TEXT**, **MEDIUMTEXT**, and **LONGTEXT** are nonstandard alternatives to **VARCHAR** with pre-defined storage capabilities.



## Character String: Text Class Data Types Comparison

Type	Description	Maximum Length
<b>CHAR</b>	M characters	255 characters
<b>VARCHAR</b>	L characters plus 1 or 2 bytes	65,535 characters <i>(subject to limitations)</i>
<b>TINYTEXT</b>	L + 1 bytes, where L < 2 <sup>8</sup>	255 bytes
<b>TEXT</b>	L + 2 bytes, where L < 2 <sup>16</sup>	65,535 bytes
<b>MEDIUMTEXT</b>	L + 3 bytes, where L < 2 <sup>24</sup>	16,777,215 bytes
<b>LONGTEXT</b>	L + 4 bytes, where L < 2 <sup>32</sup>	4,294,967,295 bytes

**ORACLE**

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- M represents the declared column length in characters.
- L represents the actual length in bytes of a string value.

# Character String: Integer Class Data Types

- Data types:
  - **ENUM**: List of enumerated string values
  - **SET**: List of zero or more string values
- **ENUM** example:
  - Country table, Continent column:  

```
Continent ENUM('Asia', 'Europe', 'North America',  
              'Africa', 'Oceania', 'Antarctica', 'South America')
```
- **SET** example:
  - Allergy table, Symptom column:  

```
Symptom SET('sneezing', 'runny nose', 'stuffy head',  
            'red eyes')
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

An **ENUM** is a list of up to 65,535 distinct strings, which is assigned to a column when the table is created. Every entry in the **ENUM** column must be one of the strings in the list.

A **SET** is an unordered list of strings considered as a whole. A **SET** can contain up to 64 strings. Each value in a **SET** column is a list containing any or all of the strings within the **SET**.

The **ENUM** and **SET** data types are represented internally as integers.

# Character Set and Collation Support

- **Character set:** A named, encoded group of characters
  - All character strings belong to a specific character set.
- **Collation:** A named collating sequence for a character set
  - Defines character sort order
  - Affects comparison of characters and strings
- Column definitions for string data types can specify a character set or collation for the column.
- Character set and collation example:

```
CREATE TABLE t
( c1 VARCHAR(20) CHARACTER SET utf8,
  c2 TEXT CHARACTER SET latin1
  COLLATE latin1_general_cs );
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When computer systems process characters, they use the numeric codes for the characters instead of their graphical representation. The process of assigning these numeric codes is called encoding.

For example, when MySQL stores the letter A, it actually stores a numeric code that the software interprets as the letter A.

A group of characters (for example, alphabetic characters, ideographs, symbols, punctuation marks, and control characters) can be encoded as a character set.

Character strings have the following characteristics:

- Every character is associated with a particular symbol. The symbol depends on the character set.
- The positions in the character set dictate an implicit sorting order (binary).
- MySQL supports several alternative sorting orders called collations.
- Characters in the same sort position are effectively equal to one another.
- Character string operations (such as comparison) are also affected by the collation.

MySQL provides the following support for character sets and collations:

- The character set for many string data types can be specified in the column definition with the `CHARACTER SET` (or `CHARSET`) attribute.
- Use the `COLLATE` attribute to specify a collation for the character set. In the example, the table definition creates a column named `c1` that uses `utf8` encoding and its default collation, and a column named `c2` that uses the `latin1` character set and a case-sensitive collation.

**Note:** The `CHARACTER SET` attribute applies to the `CHAR` and `VARCHAR` data types, the `TEXT` data types, and `ENUM` and `SET` data types.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## Binary String Data Types

- Similar to character strings, but store a sequence of bytes instead of characters
- Can store binary values that represent data, such as:
  - Compiled computer programs and applications
  - Video and audio files
- Data types:
  - **BINARY**: Fixed-length (static)
  - **VARBINARY**: Variable-length (dynamic)
  - **TINYBLOB**: Variable-length
  - **BLOB**: Variable-length
  - **MEDIUMBLOB**: Variable-length
  - **LOBLOB**: Variable-length

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Binary data types store bytes of data (binary digits [bits] grouped in eights [octets]). They are stored like character strings.

Unlike character strings, the textual representation of a binary string usually has nothing to do with its real meaning. Therefore, binary strings do not have character sets or collations.

Binary data types are suitable for storing images, multimedia, audio, and sometimes executable code.

A **BLOB** (binary large object) is a collection of binary data stored as a single value in a database.

## Binary String: Data Types Comparison

Type	Description	Maximum Length
<b>BINARY</b>	M bytes	255 characters
<b>VARBINARY</b>	L bytes plus 1 or 2 bytes	65,535 characters
<b>TINYBLOB</b>	L + 1 bytes	255 bytes
<b>BLOB</b>	L + 2 bytes	65,535 bytes
<b>MEDIUMBLOB</b>	L + 3 bytes	16,777,215 bytes
<b>LOB</b>	L + 4 bytes	4,294,967,295 bytes

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- M represents the maximum length of a column.
- L represents the actual length in of a value (0 to M).

## Quiz

Identify the best data type for the `PartName_3NF` table column `pID` in the inventory database example.

- a. INT
- b. CHAR
- c. TINYINT
- d. BLOB

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

# Choosing Data Types

When selecting a data type, consider:

- What sort of data it will store. For example, use a numeric data type for a number (instead of a character string).
- How much data it stores:
  - Max size of integers. Example:
    - Age: `TINYINT UNSIGNED` (0 to 255)
    - A country's population: `INTEGER UNSIGNED` (0 to 4,294,967,295)
  - Max length of character strings. Example:
    - Student grade: `CHAR (2)` (A+, B-)
    - Last name: `VARCHAR (64)`

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The correct use of data types is essential for database integrity and performance. Consider the following when choosing a data type:

- Use the appropriate data type. For example, storing numbers as strings. "09" and "9" are the same number but different strings. This can result in invalid data and require additional effort to convert the string to a numeric value before performing calculations.
- Always consider the maximum size of the column before setting the data type. Use a `TINYINT UNSIGNED` data type (0 to 255) for a person's age. Using an `INTEGER UNSIGNED` data type (0 to 4,294,967,295) here would be a waste of storage and could affect performance.
- Consider the minimum and maximum number of characters required for columns with variable length character strings. For example, a `VARCHAR (64)` is an acceptable length for a column containing customers' last names.



## Setting Data Types to NULL

**NULL** is a SQL keyword used to define data types that allow missing values. It means one of two things:

- **Unknown:** There is a value, but the precise value is not known at this time.
- **Not applicable:** If a value is specified, it would not be accurately representative.

Use **NULL**:

- In place of a real value in a number of situations
  - For example: “no value,” “unknown value,” “missing value,” “out of range,” “not applicable,” and so on
- To represent an empty query result

Plan for null values during database design.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In SQL, expressions can evaluate to a null value. A null value is one that cannot be computed or is not known. A null is not the same as zero (for a numeric data type), an empty string, or any other default value. Plan for the use of null values during database design when assigning data types.

## When to Use NULL

- During database design, in cases where column information is not available, determine whether null values are allowed.
- For example, the `world_innodb` database's `Country` table contains countries with no life expectancy data and should allow nulls.
- Nulls are allowed by default.
- Use `NOT NULL` if a column must not allow nulls, to ensure data integrity.
- You can apply `NULL` and `NOT NULL` to existing column definitions.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### When to use null values

When designing a database and considering what data must be stored, you can find that data is not available for all columns. Decide whether to allow null values in these cases. The default behavior is to allow nulls.

You can change this for an existing table if you detect a problem due to null values in the column.

### When not to use null values

Do not allow null values for any column that must have a value for the database design to make sense. Use `NOT NULL` to enforce this. A primary key must never contain null values.

# Summary

In this lesson, you learned how to:

- Explain the use of data types in database design
- List the four MySQL data type categories
- Choose the correct data types for columns
- Describe the special considerations for using data types
- Explain the use of `NULL`

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 5-1 Overview: Quiz – Data Types

In this practice, you answer questions about MySQL data types.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 5-2 Overview: Explaining the Use of Data Types

This practice demonstrates the use of data types in a `world_innodb` database table.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# 6 Database and Table Creation



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Use the `CREATE DATABASE` statement
- Use the `CREATE TABLE` statement to add tables to a database
- Use the `SHOW CREATE TABLE` statement
- Describe and set column and table options
- Describe and use table indexing
- Use the `SHOW INDEX` statement
- Describe table constraints

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# Creating a Database

- General syntax for creating a database:

```
CREATE DATABASE <database_name> [<options>]
```

- Example:

```
CREATE DATABASE mydatabase
```

– The response is:

```
Query OK, 0 rows affected (0.00 sec)
```

- Can use **CREATE SCHEMA** instead

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Options include the **IF NOT EXISTS** option statement. This stops you from getting an error if the database already exists.

For more information about creating a database, see the MySQL Reference Manual at <http://dev.mysql.com/doc/refman/5.6/en/create-database.html>.

# MySQL Naming Conventions

- Case sensitivity of database and table names depends on the host operating system.
- Names cannot have more than 64 characters.
- You cannot use certain characters, including ASCII(0), ASCII(255), /, \, and .
- You can use reserved words and special characters, but only if you use backticks, for example:

```
CREATE TABLE my table (ID INT);  
SELECT STATUS FROM this_table;
```

Causes an error. Replace with:

```
CREATE TABLE `my table` (ID INT);  
SELECT `STATUS` FROM this_table;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Databases and tables in MySQL are created by using directories and files in the host file system. Because of this, if the host operating system is case-sensitive in its treatment of directory and file names, so is MySQL. Windows file names are not case-sensitive, so a server running on Windows does not treat database and table names as case-sensitive. MySQL servers running on UNIX usually treat database and table names as case-sensitive because UNIX file names are case-sensitive.

Table names can be reserved words or contain illegal characters if the name is quoted in backticks (`).

# Creating a Table

- General syntax for creating a table:

```
CREATE TABLE <table_name> (  
    <column name> <column type> [<column options>]  
    ,...  
    [, <primary key definition>])
```

- Example:

```
CREATE TABLE CountryLanguage (  
    CountryCode CHAR(3) NOT NULL,  
    Language CHAR(30) NOT NULL,  
    IsOfficial ENUM('True','False') NOT NULL DEFAULT  
        'False',  
    Percentage FLOAT(3,1) NOT NULL,  
    PRIMARY KEY(CountryCode, Language)  
);
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The syntax for the `CREATE TABLE` statement is complex because it includes table and column options.

A line-by-line description of the `CREATE TABLE` example:

1. The SQL statement in the slide example creates a table called `CountryLanguage` ("(" is the beginning of the table structure definition that ends with ")").
2. The column named `CountryCode` is assigned the data type of `CHAR` and a maximum length of 3 characters. Nulls are not allowed. The comma at the end of the line indicates that the next statement is a new column or primary key definition.
3. The column named `Language` is assigned the `CHAR` data type, a length of 30 and no null values are allowed.

4. The column named `IsOfficial` is assigned the `ENUM` data type. 'True' or 'False' are the only values allowed. The addition of `NOT NULL` means that each row must contain one of these values. If no value is provided, it is set to 'False' by the phrase `DEFAULT 'False'`.
5. The `Percentage` column is assigned the `FLOAT` data type. It contains three digits, including one to the right of the decimal point. No null values are allowed.
6. It defines the key and the column(s) it applies to. In this case it defines a composite `PRIMARY KEY` using the `CountryCode` and `Language` columns. Together, these define a unique identifier for each row.
7. `)` terminates the table structure definition. `;` terminates the MySQL statement and marks it as ready for execution.

**Note:** You can also use the `CREATE TABLE` statement to create temporary tables. Temporary tables are covered in detail later in the course.

For more information about creating a table, see the MySQL Reference Manual at <http://dev.mysql.com/doc/refman/5.6/en/create-table.html>.

## Showing How a Table Was Created

- View the exact statement used to create a table.
- Example:

```
mysql> SHOW CREATE TABLE City\G
***** 1. row *****
      Table: City
Create Table: CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `SHOW CREATE TABLE` statement produces a statement that can create a table with the same properties as the table specified. This helps you understand the structure of a table. You can use it to create a new table with the same structure.

The example in the slide displays the statement used to create the `City` table.

# Column Options

- Each table must have at least one column.
- You can add options to the `CREATE TABLE` statement's column definitions, including:
  - `NULL`
  - `DEFAULT`
  - `NOT NULL`
  - `AUTO_INCREMENT`
- Example:

```
CREATE TABLE CountryLanguage (  
    CountryCode CHAR(3) NOT NULL,  
    Language CHAR(30) NOT NULL,  
    IsOfficial ENUM('T', 'F') NOT NULL DEFAULT 'F',  
    Percentage FLOAT(3,1) NOT NULL,  
    PRIMARY KEY(CountryCode, Language)  
);
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `CREATE TABLE` column options modify how MySQL handles the associated column. Some common column options:

- `NULL`: Allows null values. This is the default behavior.
- `NOT NULL`
  - Disallows null values
  - Improves performance and, in some cases, disk space
  - Is required for a `PRIMARY KEY`

**(Note:** `NULL` and `NOT NULL` are mutually exclusive.)

- `DEFAULT <value>`: If the user does not supply a value, the specified value is stored.
- `AUTO_INCREMENT`
  - A “running number” option for indexed columns
  - The column must be indexed.
  - Used for `INTEGER` data type columns only
  - If you do not specify a value or assign a null to this column, the next available number in the sequence is inserted automatically.
  - Only one permitted per table

**Note:** Indexes are covered later in this lesson.

# Table Options

- You can add options to the `CREATE TABLE` statement, such as:
  - `ENGINE`
  - `COMMENT`
  - `DEFAULT CHARACTER SET`
- Example:

```
CREATE TABLE CountryLanguage (  
    CountryCode CHAR(3) NOT NULL,  
    Language CHAR(30) NOT NULL,  
    IsOfficial ENUM('T', 'F') NOT NULL DEFAULT 'F',  
    Percentage FLOAT(3,1) NOT NULL,  
    PRIMARY KEY(CountryCode, Language)  
)  
ENGINE = InnoDB COMMENT 'Lists Language Spoken'
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `CREATE TABLE` table options modify how MySQL handles the entire table. Some common table options:

- `ENGINE = <storage_engine_name>`: The storage engine the table uses
- `COMMENT`: Up to 60 characters of free-form text
- `DEFAULT CHARACTER SET <character set>`: The default character set for the table

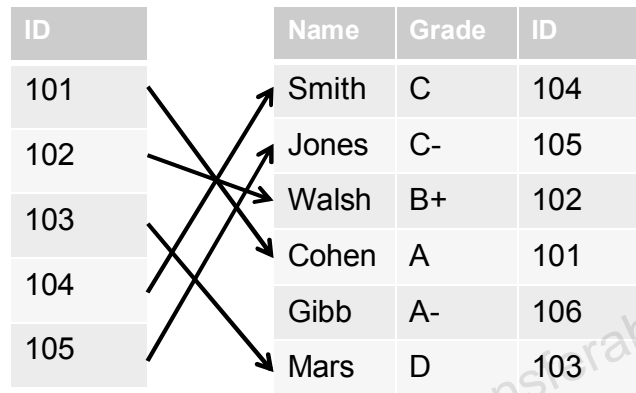
# Table Indexes

An index is a collection of pointers to records in a table.

An index helps to:

- Locate rows quickly
- Avoid full table scans
- Improve query performance

ID	Name	Grade	ID
101	Smith	C	104
102	Jones	C-	105
103	Walsh	B+	102
104	Cohen	A	101
105	Gibb	A-	106
	Mars	D	103



Unnecessary indexes are wasteful.

- Use them only for performance-critical queries.
- Avoid using them for low specificity/high cardinality values.

Key is synonymous with index in MySQL.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

By default, when MySQL attempts to locate a record, it scans the entire table until a match is found. This can really slow down large queries.

An index helps MySQL find specific column values quickly. The index entries act as pointers to the rows that match these criteria so they can be accessed without a full table scan.

You can assign indexes when you create a table, or at any time afterwards. You can create indexes on single columns or multiple columns (composite indexes). For example, you can use the phone number for an individual in a phone book as a single column index, or you can use the last name and first name to create a composite index.

You can index any table column, but do not overdo it. Too many indexes can slow query performance, so only create indexes for queries where performance is critical. Avoid indexing fields with low specificity (tendency to identify a single row) and high cardinality (there are lots of them).



# MySQL Indexing

- Commonly used indexes:
  - **PRIMARY KEY**: Only one allowed
  - **UNIQUE**: Multiple allowed
- Indexes are optional and must be created by the user.
- Example:

```
CREATE TABLE CountryLanguage (  
    CountryCode CHAR(3) NOT NULL,  
    Language CHAR(30) NOT NULL,  
    IsOfficial ENUM('T', 'F') NOT NULL DEFAULT 'F',  
    Percentage FLOAT(3,1) NOT NULL,  
    PRIMARY KEY (CountryCode, Language)  
);
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The following indexes are commonly used:

## PRIMARY KEY

- Only one allowed per table
- Uniquely identifies a single row in the table
- Null values are not allowed

## UNIQUE

- You can have multiple **UNIQUE** indexes within a table.
- Each value that is not null uniquely identifies a single row.
- Null values are allowed.

Duplicate values are not allowed in **PRIMARY KEY** or **UNIQUE** indexes. If you attempt to insert or update a duplicate, MySQL returns an error.

For more information about how MySQL uses indexes, see the MySQL Reference Manual at <http://dev.mysql.com/doc/refman/5.6/en/mysql-indexes.html>.

## Showing Table Indexes

- General syntax for displaying table indexes:

```
SHOW INDEX FROM <table_name>
```

- Example:

```
mysql> SHOW INDEX FROM Country\G
***** 1. row *****
      Table: Country
      Non_unique: 0
      Key_name: PRIMARY
      Seq_in_index: 1
      Column_name: Code
      Collation: A
      Cardinality: 239
      Sub_part: NULL
      Packed: NULL
      Null:
      Index_type: BTREE
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The result of the `SHOW INDEX` statement example in the slide shows that there is a `PRIMARY` index on the `Code` column of the `Country` table, of type `BTREE`. The `Cardinality` column shows the number of distinct `Code` values in the table: 239 in this case.

**Note:** Most indexes are stored in B-trees. Exceptions include spatial data types, (which use R-trees) and `MEMORY` tables, (which also support hash indexes).

For more information about the `SHOW INDEX` statement, see the *MySQL Reference Manual* at <http://dev.mysql.com/doc/refman/5.6/en/show-index.html>.

# Table Constraints

- A constraint is a restriction placed on one or more column values of a table to enforce integrity rules.
- You implement constraints by using indexes.
- Types of constraints:
  - PRIMARY KEY
  - FOREIGN KEY
  - UNIQUE
- MySQL generates indexes to enforce the preceding constraints.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Table constraints help enforce data integrity. The following indexes implement these constraints:

- PRIMARY KEY: Defines the columns that guarantee that each record in the table is unique
- FOREIGN KEY: Restricts column values to ensure a match in another table's PRIMARY KEY or UNIQUE columns
- UNIQUE: The same as PRIMARY KEY except that the columns can contain null values, and there can be multiple UNIQUE constraints in a table

For example, if the MySQL server allowed you to modify the `world_innodb` database by changing the `CountryCode` value in the `CountryLanguage` table without changing the corresponding `CountryCode` value in the `City` table, the result is rows that no longer point to valid country records (known as orphaned rows). With constraints in place (FOREIGN KEY, in this case), the server returns an error if an attempt is made to modify or delete data that is referenced by other tables, or it propagates the changes to other tables for you.

**Note:** Only the InnoDB storage engine supports FOREIGN KEY constraints. Other storage engines ignore these constraints in table definitions.

## Quiz

Which of the following SQL statements do you use to create a database after completing a database design?

- a. SHOW CREATE DATABASE
- b. CREATE TABLE
- c. CREATE DATABASE

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: c**

# Summary

In this lesson, you learned how to:

- Use the `CREATE DATABASE` statement
- Use the `CREATE TABLE` statement to add tables to a database
- Use the `SHOW CREATE TABLE` statement
- Describe and set column and table options
- Describe and use table indexing
- Use the `SHOW INDEX` statement
- Describe table constraints

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 6-1 Overview: Displaying Table Creation Information

This practice covers the following topics:

- Examining the SQL statement used to create the `Country` table in the `world_innodb` database
- Using the output from a `SHOW CREATE TABLE` statement to create a new table
- Showing the indexes used by a table

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## Practice 6-2 Overview: Creating a Database

In this practice, you use your knowledge of database design and creation to create the `Pets` database and its tables.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.



# 7

## Basic Queries

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Retrieve database data by using the `SELECT` statement
- Use the `SELECT` statement clauses: `FROM`, `DISTINCT`, `WHERE`, `ORDER BY`, and `LIMIT`
- Troubleshoot problems by using warning and error messages
- Describe and use SQL modes to change the interpretation of SQL syntax
- Use the MySQL Workbench GUI tool to submit queries

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# SELECT Statement

- **SELECT** is the most commonly used DML command for queries:
  - Retrieves rows from tables in a database
  - Returns rows as a “result set” in the form of a table
- General syntax:

```
SELECT [<clause options>] <column_list>  
[FROM <table_name>] [<other_clauses>]
```

- **column\_list** is a list of column names that make up the result set.
  - Separate the items in the list with a comma separator (,).

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In MySQL, **SELECT** is the most commonly used data manipulation language (DML) statement. The **SELECT** statement describes the data to be retrieved from the database. It does not specify exactly how to retrieve that data. The database server returns a table that contains only the rows that meet the **SELECT** criteria.

You can use **SELECT** to evaluate expressions, too. Add the expression, (which does not have to refer to a column or table) to the column list, for example:

```
mysql> SELECT 2+2;  
-> 4
```

For more information about the **SELECT** statement, see the MySQL Reference Manual at <http://dev.mysql.com/doc/refman/5.6/en/select.html>.

# SELECT Statement

Example: querying world\_innodb with **SELECT**:

```
mysql> SELECT Name, Continent FROM Country;
+-----+-----+
| Name          | Continent          |
+-----+-----+
| Aruba         | North America     |
| Afghanistan  | Asia               |
| Angola        | Africa             |
| Anguilla      | North America     |
| Albania       | Europe             |
| ...           | ...                |
| Yemen         | Asia               |
| Yugoslavia    | Europe             |
| South Africa  | Africa             |
| Zambia        | Africa             |
| Zimbabwe     | Africa             |
+-----+-----+
239 rows in set (0.02 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows a typical multiple-column **SELECT** statement, which retrieves the Name and Continent column values from all the rows in the Country table. You can **SELECT** one, multiple, or all columns from one or more tables.

Use “\*” to include all columns without listing them individually:

```
mysql> SELECT * FROM Country;
```

**Note:** A **SELECT** statement is often called a query.

# Using SELECT Clauses

Use optional clauses alone (or in combination) to generate specific query results.

- Types of clauses:
  - **DISTINCT**: Eliminates duplicate rows from the result set
  - **FROM**: Specifies which tables to retrieve data from
  - **WHERE**: Filters rows according to specified criteria
  - **ORDER BY**: Sorts rows in a specified order
  - **LIMIT**: Limits the maximum number of rows in the result set
- The clauses must be used in the correct order:

```
SELECT DISTINCT <values_to_return>  
FROM <table_name>  
WHERE <condition>  
ORDER BY <how_to_sort>  
LIMIT <row_count>;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows the correct order and syntax of the optional clauses. If you do not use them in the correct order, you get a syntax error.

This statement:

- Selects **DISTINCT** (unique) rows
- **FROM** a named table
- **WHERE** some condition defines which rows to return
- Returns the results in the specified order (**ORDER BY**)
- **LIMITS** the number of rows to be retrieved to a given number

## Using SELECT with DISTINCT

- Removes duplicate rows, so every result set row is unique
- Difference between **SELECT** with and without **DISTINCT**:

```
mysql> SELECT Continent
-> FROM Country;
+-----+
| Continent |
+-----+
| Asia      |
| Europe    |
| North America |
| Europe    |
| Africa    |
| Oceania   |
| Europe    |
| Africa    |
...
239 rows in set (0.01 sec)
```

**compared to** →

```
mysql> SELECT DISTINCT Continent
-> FROM Country;
+-----+
| Continent |
+-----+
| Asia      |
| Europe    |
| North America |
| Africa    |
| Oceania   |
| South America |
| Antarctica |
...
7 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Example of the difference between **SELECT** without and with **DISTINCT** for multiple columns:

```
mysql> SELECT Continent, Region FROM Country;
+-----+-----+
| Continent | Region |
+-----+-----+
| North America | Caribbean |
| Asia          | Southern and Central Asia |
| Africa        | Central Africa |
| North America | Caribbean |
...
| Oceania       | Polynesia |
| Oceania       | Polynesia |
...
| Africa        | Eastern Africa |
| Africa        | Eastern Africa |
+-----+-----+
239 rows in set (0.00 sec)
```

... compared to ...

```
mysql> SELECT DISTINCT Continent, Region FROM Country;
```

Continent	Region
North America	Caribbean
Asia	Southern and Central Asia
Africa	Central Africa
Europe	Southern Europe
Asia	Middle East
South America	South America
Oceania	Polynesia
Antarctica	Antarctica
Oceania	Australia and New Zealand
Europe	Western Europe
Africa	Eastern Africa
Africa	Western Africa
Europe	Eastern Europe
North America	Central America
North America	North America
Asia	Southeast Asia
Africa	Southern Africa
Asia	Eastern Asia
Europe	Nordic Countries
Africa	Northern Africa
Europe	Baltic Countries
Oceania	Melanesia
Oceania	Micronesia
Europe	British Islands
Oceania	Micronesia/Caribbean

25 rows in set (0.00 sec)

## SELECT DISTINCT with Null Values

- Treats all nulls as the same value
- Retrieval of null values with and without DISTINCT:

<pre>mysql&gt; SELECT i, j -&gt; FROM t;</pre> <table border="1"><thead><tr><th>i</th><th>j</th></tr></thead><tbody><tr><td>1</td><td>2</td></tr><tr><td>1</td><td>NULL</td></tr><tr><td>1</td><td>NULL</td></tr></tbody></table>	i	j	1	2	1	NULL	1	NULL	<p>compared to</p>	<pre>mysql&gt; SELECT DISTINCT i, j -&gt; FROM t;</pre> <table border="1"><thead><tr><th>i</th><th>j</th></tr></thead><tbody><tr><td>1</td><td>2</td></tr><tr><td>1</td><td>NULL</td></tr></tbody></table>	i	j	1	2	1	NULL
i	j															
1	2															
1	NULL															
1	NULL															
i	j															
1	2															
1	NULL															

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The NULL values in the second column are the same, so the second and third rows are identical. If you add DISTINCT to the query it eliminates one of them as a duplicate.



## SELECT with WHERE

Expressions in a **WHERE** clause can use the following types of operators:

- Arithmetic: **+**, **-**, **\***, **/**, **DIV**, **%**
- Comparison: **<**, **<=**, **=**, **<=>**, **<>** or **!=**, **>=**, **>**, **BETWEEN**
- Logical: **AND**, **OR**, **XOR**, **NOT**
- Additional options: **IN**, **IS NULL**, **LIKE**, **()**

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### Arithmetic Operators

Perform mathematical operations in column value expressions:

- **+** Addition
- **-** Subtraction
- **\*** Multiplication
- **/** Division
- **DIV** Integer division
- **%** Modulo (remainder after division)

## Comparison Operators

Compare column value expressions

- + addition
- < less than
- <= less than or equal to
- = equal to
- <=> equal to (works even for null values)
- <> or != not equal to
- >= greater than or equal to
- > greater than
- BETWEEN <value1> AND <value2> indicates a range of values (inclusive)

## Logical Operators

Combine Boolean expressions

- AND: logical AND
- OR: logical OR
- XOR: logical exclusive-OR
- NOT: logical negation

## Additional Options in a WHERE Condition:

- IN is equivalent to a list of comparisons with OR, but more readable and efficient.
- Use IS NULL to check whether a value is null (WHERE <column\_name> IS NULL).
- Use the LIKE operator for pattern matching (WHERE <value> LIKE '<pattern>').
- Use parentheses (<expression>) to group parts of an expression. MySQL evaluates the part in parentheses first and uses that value to evaluate the rest of the expression. If more than one part of an expression is in parentheses, the evaluation order is from left to right. MySQL evaluates the innermost expressions first if parentheses are nested. Operators can be applied to almost all types of value expressions such as literals, columns, function calls, and so on.
- You can combine several criteria with logical operators.

## SELECT with WHERE

### Example of WHERE with IN:

```
mysql> SELECT ID, Name, District FROM City
      -> WHERE Name IN ('New York', 'Rochester',
      'Syracuse');
```

```
+-----+-----+-----+
| ID    | Name      | District |
+-----+-----+-----+
| 3793  | New York  | New York |
| 3871  | Rochester | New York |
| 3935  | Syracuse  | New York |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide demonstrates the use of the IN expression in a WHERE condition to find a list of the specified city names, along with their corresponding ID and District, from the City table.

## SELECT with WHERE

Example of **WHERE** with **AND** and **OR**:

```
mysql> SELECT Name, Continent FROM Country
       -> WHERE GovernmentForm = 'Republic'
       -> AND (Continent = 'North America'
       -> OR Continent = 'Europe');
```

```
+-----+-----+
| Name          | Continent |
+-----+-----+
| Albania       | Europe   |
| Bulgaria      | Europe   |
| Costa Rica    | North America |
| ...          |          |
| Trinidad and Tobago | North America |
| Ukraine       | Europe   |
+-----+-----+
35 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide uses **AND** (with an **OR** expression) in a **WHERE** condition to list all countries in North America and Europe whose government is a Republic, from the **Country** table.

## SELECT with ORDER BY

- Returns output rows in a specific order
- Example of ORDER BY:

```
mysql> SELECT Name FROM Country ORDER BY Name;
+-----+
| Name                |
+-----+
| Afghanistan         |
| Albania              |
| Algeria              |
| American Samoa      |
| Andorra              |
| Angola               |
| Anguilla             |
| Antarctica           |
| Antigua and Barbuda |
| Argentina            |
| ...                  |
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

By default, the server returns the rows in the `SELECT` statement to the client in no particular order. The actual order depends on various factors, such as the order MySQL stores the rows in the table, or which indexes are used to process the query. If you want the rows to be returned in a specific order, use the `ORDER BY` clause.

The example in the slide sorts the names of the countries in the `Country` table in alphabetical order by `Name`.

The following example uses `ORDER BY` in a more complex `WHERE` query:

```
mysql> SELECT Name, Continent FROM Country
-> WHERE GovernmentForm = 'Republic'
-> AND (Continent='North America' OR Continent='Europe')
-> ORDER BY Name;
```

Name	Continent
Albania	Europe
Belarus	Europe
Bulgaria	Europe
Costa Rica	North America
Croatia	Europe
Czech Republic	Europe
...	
Romania	Europe
San Marino	Europe
Slovakia	Europe
Slovenia	Europe
Trinidad and Tobago	North America
Ukraine	Europe

35 rows in set (0.00 sec)

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## SELECT with ORDER BY with ASC and DESC

- Specify order with **ASC** and **DESC**:
  - **ASC**: Ascending order (default)
  - **DESC**: Descending order
- Example of **ORDER BY** with **DESC**:

```
mysql> SELECT Name FROM Country
      -> ORDER BY Name DESC;
+-----+
| Name          |
+-----+
| Zimbabwe     |
| Zambia       |
| Yugoslavia   |
| Yemen        |
| Western Sahara |
| Wallis and Futuna |
| ...         |
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**ORDER BY** sorts values in ascending order by default. You can specify the sort order explicitly using the **ASC** or **DESC** keywords after the column names.

The statement in the example in the slide sorts the names of the countries in the `Country` table in descending alphabetical order.

**Note:** **ORDER BY** also works with functions, mathematical operations, and any value expression.

## SELECT with ORDER BY with ASC and DESC

- You can sort multiple columns simultaneously.
- Example of ORDER BY with both ASC and DESC:

```
mysql> SELECT Name, Continent FROM Country
      -> ORDER BY Continent DESC, Name ASC;
+-----+-----+
| Name          | Continent      |
+-----+-----+
| Argentina     | South America  |
| Bolivia       | South America  |
| Brazil        | South America  |
| Chile         | South America  |
| ...          | ...            |
| Uzbekistan    | Asia           |
| Vietnam       | Asia           |
| Yemen         | Asia           |
+-----+-----+
239 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide sorts the continents in the `Country` table in descending alphabetical order, and the country names in ascending order for each continent. The `Continent` column is an `ENUM`. Therefore, the order of its values depends on their placement in the list, which just happens to be alphabetic in this instance.



## SELECT with LIMIT

- Specifies the number of rows to output in result set
- Example of **LIMIT**:

```
mysql> SELECT Name FROM Country LIMIT 8;
+-----+
| Name |
+-----+
| Aruba |
| Afghanistan |
| Angola |
| Anguilla |
| Albania |
| Andorra |
| Netherlands Antilles |
| United Arab Emirates |
+-----+
8 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The **LIMIT** clause constrains the number of rows that appear in the result set.

You can use the **LIMIT** clause with either one or two arguments:

- ... **LIMIT** <row\_count>
- ... **LIMIT** <skip\_count>, <row\_count>

You must represent each argument as an integer constant. You cannot use expressions, user variables, and so on.

When followed by a single integer <row\_count>, **LIMIT** returns the first <row\_count> rows from the beginning of the result set.

As shown in the example in the slide, to select just the first 8 rows of the **Name** column from the **Country** table, use **LIMIT 8**.

## SELECT with LIMIT and Skip Count

- Specifies how many rows to skip before starting result set
- Example of skipped LIMIT:

```
mysql> SELECT Name, Population FROM Country
-> LIMIT 20,8;
```

Name	Population
Burkina Faso	11937000
Bangladesh	129155000
Bulgaria	8190900
Bahrain	617000
Bahamas	307000
Bosnia and Herzegovina	3972000
Belarus	10236000
Belize	241000

```
8 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When followed by two integers, `<skip_count>` and `<row_count>`, LIMIT skips the first `<skip_count>` rows from the beginning of the result set, and returns the number of rows specified by `<row_count>`.

As shown in the example in the slide, to select the next 8 rows that come after the first 20 rows of the Name and Population columns, use LIMIT 20, 8.

## SELECT with LIMIT and ORDER BY

- Returns a specific number of rows in a specific order
- Example:

```
mysql> SELECT Name FROM Country
      -> ORDER BY Name LIMIT 8;
+-----+
| Name          |
+-----+
| Afghanistan  |
| Albania      |
| Algeria       |
| American Samoa |
| Andorra       |
| Angola        |
| Anguilla      |
| Antarctica    |
+-----+
8 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use `LIMIT` in conjunction with the `ORDER BY` clause to put the specified number of rows in a particular order. MySQL applies `ORDER BY` first and then `LIMIT`. One common use for this is to find the row containing the smallest or largest value in a particular column.

The example in the slide combines the previous `LIMIT` example with `ORDER BY` to return the first 8 rows of the `Name` column, in alphabetical order.

## SELECT Usage Tips

- Treat table and database names as case-sensitive.
- From the mysql command-line client:
  - Use `\c` to abort a SQL statement
  - Use `\G` in place of `;` to see results with each column in a row, instead of as a table, for example:

```
mysql> SELECT Name, Continent FROM Country\G
***** 1. row *****
      Name: Afghanistan
      Continent: Asia
***** 2. row *****
      Name: Angola
      Continent: Africa
      . . .
```

- Use `*` (all columns) with caution; it can waste resources and give unpredictable results due to changes in columns.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use `\G` at the end of a statement when the output is too wide for the screen.

You can use the clauses covered in this lesson with statements other than `SELECT`. Some of them are covered later in the course.

Using `*` (meaning all columns) can waste resources (time, disk space, and so on) and result in potential maintenance problems. For example, if you add a column to a table, `SELECT *` queries on the table will automatically retrieve values for the new column. This might not be what you want.

# Troubleshooting

- MySQL provides error and warning messages.
- Error message example:

```
mysql> SELECT Name FROM City LIMIT;
```

**MySQL error code** **SQLSTATE**

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1
```

**error message**

- This error is due to the missing `<row_count>` value for the `LIMIT` clause.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

There are three components of these messages:

- A MySQL-specific code
- A SQLSTATE error code. A SQLSTATE error code is defined by standard SQL and ODBC.
- A text message that describes the problem

In the example in the slide, the message indicates that there is a syntax error in your statement and recommends that you check the manual.

The information string for multiple-row statements is a summary rather than a complete listing of diagnostics.

# Troubleshooting

- The following statements control display of errors and warnings:
  - **SHOW WARNINGS:** Displays any error, warning, and informational messages generated by the last statement
  - **SHOW ERRORS:** Displays error messages only
- Refer to the error code definitions in the MySQL Reference Manual.
- After you understand the cause of the error or warning, make any needed changes and run the SQL statement again.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Many MySQL statements generate warnings and errors. These are not always displayed by default. Use the following statements to turn on error and warning output:

- **SHOW WARNINGS:** Shows information about any errors, warnings, and notes that resulted from the last statement in the current session that generated messages. It shows nothing if the last statement used a table and did not generate any messages. (That is, a statement that uses a table but generates no messages clears the message list.) Statements that do not use tables and do not generate messages have no effect on the message list.
- **SHOW ERRORS:** The list of messages is reset by each new statement that uses a table.

You can also find out what an error code means by using the `pererror` utility, which displays a description for a given error code:

```
cmd> pererror [options] errorcode
```

Use the MySQL Reference Manual to look up the meanings of error codes and get help for common problems:

- **Error codes/messages:** <http://dev.mysql.com/doc/refman/5.6/en/error-handling.html>
- **Problems and common errors:** <http://dev.mysql.com/doc/refman/5.6/en/problems.html>

# SQL Modes for Syntax Checking

You can customize the MySQL server behavior by using SQL modes.

- SQL modes govern behavior such as:
  - Which SQL syntax MySQL supports
  - What data validation checks it performs
- Customize server operating mode at run time:

```
SET [SESSION|GLOBAL] sql_mode='mode_value'
```

- View current global or session SQL mode value:

```
SELECT @@global.sql_mode
```

```
SELECT @@session.sql_mode
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

SQL mode settings can make it easier to use MySQL in different environments and in conjunction with other database servers.

You can set the default SQL mode by starting `mysqld` with the

`--sql-mode = 'mode_value'` option.

Setting a variable with the `GLOBAL` options requires the `SUPER` privilege and affects all clients that connect from that time on. Setting the `SESSION` variable affects only the current client. Any client can change its own `session sql_mode` value at any time.

For more information about SQL modes, see the MySQL Reference Manual:  
<http://dev.mysql.com/doc/refman/5.6/en/server-sql-mode.html>.

## Common SQL Modes

- **ANSI:** Change syntax and behavior to conform more closely to standard SQL.
- **STRICT\_TRANS\_TABLES** and **STRICT\_ALL\_TABLES:** Enable “strict mode.”
  - Transactional tables: Abort a statement and roll back its effect on the database if a value could not be inserted as given.
  - Nontransactional tables: Abort a statement if the value occurs in a single-row statement or the first row of a multiple-row statement.
- **TRADITIONAL:** Make MySQL behave like a “traditional” SQL database system.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- **TRADITIONAL** mode shows errors instead of warnings if you try to insert an incorrect value into a column.
- **STRICT\_TRANS\_TABLES** enables strict mode for transactional storage engines (such as InnoDB, the default), and when possible for nontransactional storage engines (such as MyISAM).
- **STRICT\_ALL\_TABLES** enables strict mode for all storage engines. Invalid data values are rejected.



## Quiz

Using `SELECT` with the `DISTINCT` clause returns a result set in alphabetical order.

- a. True
- b. False

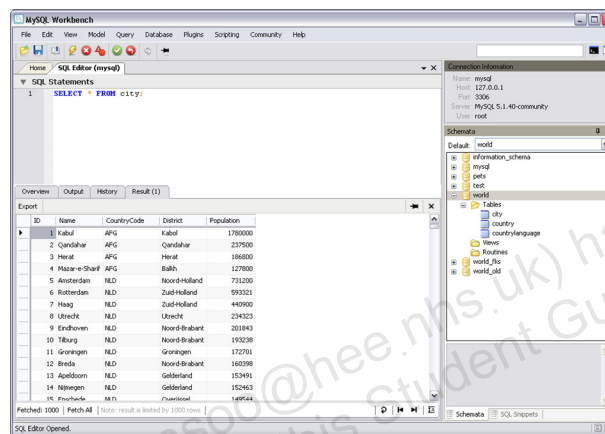
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: b**

# MySQL Workbench for SQL Development

- A GUI tool that includes the SQL Development module:
  - Use it to query and analyze data on a MySQL server.
- Download MySQL Workbench from the MySQL website
- Available for Windows, Linux, and Mac OS:
  - Community and commercial versions



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL Workbench provides an alternative environment in which to write and execute MySQL code. You can enter SQL statements in a graphical user interface and save them to a file to use later.

# Summary

In this lesson, you learned how to:

- Retrieve database data by using the `SELECT` statement
- Use the `SELECT` statement clauses: `FROM`, `DISTINCT`, `WHERE`, `ORDER BY`, and `LIMIT`
- Troubleshoot problems by using warning and error messages
- Describe and use SQL modes to change the interpretation of SQL syntax
- Use the MySQL Workbench GUI tool to submit queries

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 7-1 Overview: Performing Basic Queries

In this practice, you query the `world_innodb` database by using `SELECT` statements from the `mysql` command-line program.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 7-2 Overview: Performing Basic Queries by Using MySQL Workbench

In this practice, you query the `world_innodb` tables by using `SELECT` statements, within the MySQL Workbench tool.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 7-3 Overview: Performing Basic Queries on the Pets Database

In this practice, you query the `Pets` database using `SELECT` statements. You can use either the `mysql` command-line client or the MySQL Workbench tool.

# Database and Table Maintenance

# 8

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Delete (or “drop”) a database
- Understand the issues related to dropping a database
- Create a new table from existing table data
- Delete a table
- Add and remove table columns
- Modify table columns
- Add and remove indexes and constraints

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



## DROP DATABASE Statement

- Deletes a database:
  - Includes any tables and their data
  - Requires the `DROP` privilege on the database
- Returns a row count, which is the number of tables deleted
- Cannot be reversed; use with extreme caution!
- Examples:

```
DROP DATABASE my_database
```

- Returns an **error** if the database does not exist

```
DROP DATABASE IF EXISTS my_database
```

- Returns a **warning** if the database does not exist

**ORACLE**

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Before dropping the database, MySQL removes any objects that it contains, such as tables, stored routines, and triggers.

A successful `DROP DATABASE` returns a row count that indicates the number of tables dropped. (This is actually the number of `.frm` files removed, which amounts to the same thing.) Issue a `SHOW DATABASES` statement after the drop to confirm the deletion.

The host file system stores the database in its own directory under the data directory. When you drop the database, the server deletes only the files and directories it created and leaves others intact. If you or another process created files in the database directory, the server cannot delete those files and they prevent the server from deleting the directory. As such, the database still shows up in the results of a `SHOW DATABASES` statement. You need to remove the database directory and any remaining files manually.

Use the `SHOW WARNINGS` statement to display warnings generated by `DROP DATABASE`.

## Creating a New Table by Using an Existing Table

- Use the `CREATE TABLE` statement with a `SELECT` on an existing table(s).
  - Creates a new table with the results of the query
- Use existing columns or create new columns with the `AS` keyword.
- Example:

```
mysql> CREATE TABLE EU_Countries
-> SELECT Name,
-> Population * 1.5 AS NewPopulation
-> FROM Country
-> WHERE Continent = 'Europe';
Query OK, 46 rows affected (0.42 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

*new table name* (points to `EU_Countries`)  
*column alias* (points to `AS NewPopulation`)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `SELECT` statement can reference multiple tables with joins, subqueries, or unions. Or you can use a `SELECT` statement without any table.

The example shown creates a new table called `EU_Countries` from a `SELECT` query on the `Country` table. The new table has two columns: The `Name` column data from the `Country` table and a new column called `NewPopulation`. The `NewPopulation` column derives its values from an expression that uses the `Country` table's `Population` column data. The `WHERE` clause limits the source data to European countries only.

**Note:** The tables created with `CREATE TABLE...SELECT` are based solely on the output of the `SELECT`. They do not include table options, such as indexes and constraints. Also, the newly created tables do not always contain the data types you expect, particularly if the column is created based on an expression.

For more information about using `CREATE TABLE` with `SELECT`, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.6/en/create-table-select.html>.

## Confirming the Creation of a New Table

- Show the new table:
- Confirm the structure of the new table:

```
mysql> SHOW TABLES;  
+-----+  
| Tables_in_world_innodb |  
+-----+  
| city                    |  
+-----+  
| ..                      |  
| eu_countries            |  
+-----+
```

```
mysql> DESCRIBE EU_Countries;  
+-----+-----+-----+-----+-----+  
| Field          | Type          | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+  
| Name           | char(52)      | NO   |     |         |       |  
| NewPopulation  | decimal(12,1) | NO   |     | 0.0     |       |  
+-----+-----+-----+-----+-----+
```

- Select data from the new table:

```
mysql> SELECT * FROM EU_Countries;  
+-----+-----+  
| Name           | NewPopulation |  
+-----+-----+  
| Albania        | 5101800.0     |  
| Andorra        | 117000.0      |  
| Austria        | 12137700.0    |  
| ..             |               |  
| Yugoslavia     | 15960000.0    |  
+-----+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the slide example, the SHOW TABLES, DESCRIBE, and SELECT statements confirm the creation of the new EU\_Countries table and its contents.

## Copying an Existing Table Structure

- Use **CREATE TABLE** with the **LIKE** keyword to create a table with the same structure as another table:
  - Indexes
  - Column options
- Only creates the table structure
  - Does not copy any data
- Example:

```
mysql> CREATE TABLE NewCity LIKE City;
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT * FROM NewCity;
Empty set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Creating a Temporary Table

- Use **CREATE TEMPORARY TABLE** for a table that:
  - Exists only for the duration of the client session
  - Is visible to only the client that created it
  - Does not affect other clients that are using the same data
  - Can be used to override an existing permanent table
- Use temporary tables for storing summary data.
- Example:

```
mysql> CREATE TEMPORARY TABLE EU_CountriesTemp
-> SELECT Name, Population * 1.5
-> AS NewPopulation
-> FROM Country
-> WHERE Continent = 'Europe';
Query OK, 46 rows affected (0.42 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you create a temporary table with the same name as an existing permanent table, the temporary table overrides the existing permanent table within the client session. This lasts until you drop the temporary table or disconnect the client session.

You can examine the table created by the statement in the slide example by using the **DESCRIBE** statement:

```
mysql> DESC EU_CountriesTemp;
```

Field	Type	Null	Key	Default	Extra
Name	char(52)	NO			
NewPopulation	decimal(12,1)	NO		0.0	

```
2 rows in set (0.08 sec)
```

Use **CREATE TEMPORARY TABLE ... LIKE** to create a temporary table with the same structure as an existing table.

## DROP TABLE Statement

- Removes one or more tables:
  - The table can be empty or contain data.
  - This requires the `DROP` privilege on the table.
- Cannot be reversed; use with extreme caution!
- Examples:

```
DROP TABLE table1, table2, table3
```

- Returns an error if the table does not exist

```
DROP TABLE IF EXISTS table1
```

- Returns a warning if the table does not exist

```
DROP TEMPORARY TABLE table1_temp
```

- Removes only a temporary table

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use `IF EXISTS` to prevent an error if you drop tables that do not exist. This generates a warning instead which can be displayed with the `SHOW WARNINGS` statement.

Using `DROP TABLE` with the `TEMPORARY` keyword:

- Drops only temporary tables
- Does not end an ongoing transaction. (Transactions are covered later in the course.)
- Does not check access rights. (A `temporary` table is visible only to the client that created it, so no check is necessary.)

Use `DROP TEMPORARY TABLE` to ensure that you do not accidentally drop a permanent table with the same name.

# Adding a Table Column

- Use the **ALTER TABLE** statement with **ADD COLUMN**.
- Example:

```
mysql> ALTER TABLE EU_Countries
      -> ADD COLUMN ID INT NOT NULL;
Query OK, 46 rows affected (0.11 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

- Adds **ID** as the last column in the table:

```
mysql> DESC EU_Countries;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Name           | char(52)      | NO   |     |         |       |
| NewPopulation  | decimal(12,1) | NO   |     | 0.0     |       |
| ID             | int(11)       | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use **ALTER TABLE ... ADD COLUMN** with the appropriate clause that specifies the column's definition. Use the same **ADD COLUMN** syntax as you would for **CREATE TABLE**. You can use **ALTER TABLE** to:

- Add or remove a column
- Add or remove an index
- Change an existing column's definition

Column names within a table must be unique, so you cannot add a column if one of the same name already exists in the table. Also, column names are not case-sensitive, so if the table already contains a column named **ID**, you cannot add a new column using any of these names: **ID**, **id**, **Id**, or **iD**. They are all considered to be the same.

## Adding a Table Column

- Adding a column to a table populates the rows with **NULL**, the specified default value, or the implicit default for the data type.
- Example: The **ID** column uses the default for the **INT** data type (zero).

```
mysql> SELECT * FROM EU_Countries;
```

Name	NewPopulation	Id
Albania	5101800.0	0
Andorra	117000.0	0
Austria	12137700.0	0
Belgium	15358500.0	0
Bulgaria	12286350.0	0
Bosnia and Herzegovina	5958000.0	0
Belarus	15354000.0	0
...		

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Note:** The statement for populating a table with data is covered in detail later in the course.



## Removing a Table Column

- Use the `ALTER TABLE` statement with `DROP COLUMN`.
- Example:

```
mysql> ALTER TABLE EU_Countries
-> DROP COLUMN ID;
Query OK, 46 rows affected (0.11 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

- Removes the `ID` column:

```
mysql> DESC EU_Countries;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Name           | char(52)      | NO   |     |         |       |
| NewPopulation  | decimal(12,1) | NO   |     | 0.0     |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If a table contains only one column, you cannot drop the column. If you intend to remove the table, use `DROP TABLE` instead.

Do not remove a column from a table if it is a primary key. You cannot remove a column that is a foreign key referencing another table.

For more information about using `ALTER TABLE`, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.6/en/alter-table.html>.

# Modifying a Table Column

- Use the **ALTER TABLE** statement with **MODIFY COLUMN**.
- Example:

```
mysql> ALTER TABLE EU_Countries
      -> MODIFY COLUMN NewPopulation
      -> INT UNSIGNED NOT NULL;
Query OK, 46 rows affected (0.11 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

- The `NewPopulation` column data type changes to `INT`:

```
mysql> DESC EU_Countries;
+-----+-----+-----+-----+-----+-----+
| Field          | Type                | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Name           | char(52)            | NO   |     |         |       |
| NewPopulation | int(10) unsigned    | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.06 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows how to change the `NewPopulation` column's data type from `DECIMAL` to `INT`, restricting column values to whole numbers, disallowing negative values with the `UNSIGNED` attribute and null values with `NOT NULL`.

When you modify a table column you have to reapply all the attributes you want to keep from the old column definition. For example, the old column definition did not permit null values. If you want to disallow nulls in the new column definition you need to specify `NOT NULL` again.

You cannot modify a column if it is a primary key and if a foreign key from another table references the column.

## Modifying a Table Column: Row Changes

The previous `ALTER TABLE...MODIFY COLUMN` statement removed the decimal part of the `NewPopulation` values:

```
mysql> SELECT * FROM EU_Countries;
```

Name	NewPopulation
Albania	5101800
Andorra	117000
Austria	12137700
Belgium	15358500
Bulgaria	12286350
Bosnia and Herzegovina	5958000
Belarus	15354000
Switzerland	10740600
...	

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Anything that can be added by using `CREATE TABLE` can be changed by using `ALTER TABLE`.

For more information about using `ALTER TABLE`, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.6/en/alter-table.html>.

**Note:** The statement for populating a table with data is covered in detail later in the course.

## Adding Indexes and Constraints

- Use the **ALTER TABLE** statement with **ADD** options to add indexes and constraints to columns.
- General syntax:

```
ALTER TABLE table_name ADD INDEX [index_name]  
      (index_columns)
```

```
ALTER TABLE table_name ADD UNIQUE [index_name]  
      (index_columns)
```

```
ALTER TABLE table_name ADD PRIMARY KEY  
      (index_columns)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use **ALTER TABLE** to add indexes and constraints to existing columns. The index name is optional.

## Adding a Column Index

- Use the **ALTER TABLE** statement with **ADD INDEX**.
- Example:

```
mysql> ALTER TABLE NewCity
      -> ADD INDEX Pop (Population);
Query OK, 0 rows affected (0.22 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

- Adds the index **Pop** for the **Population** column

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Adding a Column Index

Confirm the change by using `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE NewCity\G
***** 1. row *****
Table: City
Create Table: CREATE TABLE 'city' (
  'ID' int(11) NOT NULL auto_increment,
  'Name' char(35) NOT NULL default '',
  'CountryCode' char(3) NOT NULL default '',
  'District' char(20) NOT NULL default '',
  'Population' int(11) NOT NULL default '0',
  PRIMARY KEY ('ID'),
  KEY `CountryCode` (`CountryCode`),
  KEY 'Pop' ('Population')
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Dropping a Column Index

- Use the **ALTER TABLE** statement with **DROP INDEX**.
- Example:

```
mysql> ALTER TABLE NewCity DROP INDEX Pop;
Query OK, 0 rows affected (0.22 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

- Removes the **Pop** index

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Confirm the change in the slide example with **SHOW CREATE TABLE**:

```
mysql> SHOW CREATE TABLE NewCity\G
***** 1. row *****
      Table: NewCity
Create Table: CREATE TABLE `newcity` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.08 sec)
```

You can remove constraints in the same way, for example:

```
ALTER TABLE City DROP FOREIGN KEY CountryCode;
```

## Quiz

When you add a new column to a table with **ALTER TABLE . . . ADD COLUMN**, it adds the new column and also populates it with data.

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: b**



# Summary

In this lesson, you learned how to:

- Delete (or “drop”) a database
- Understand the issues related to dropping a database
- Create a new table from existing table data
- Delete a table
- Add and remove table columns
- Modify table columns
- Add and remove indexes and constraints

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 8-1 Overview: Removing a Database

In this practice, you drop an unwanted database.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 8-2 Overview: Creating a New Table and Removing a Table

This practice covers the following topics:

- Creating a table from an existing table
- Removing an entire table, including its data

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 8-3 Overview: Altering Table Columns

This practice covers the following topics:

- Modifying the data type of a table column
- Adding a new column to a table

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 8-4 Overview: Modifying Table Indexes and Constraints

This practice covers the following topics:

- Adding an index to a table
- Dropping an index from a table
- Adding a primary key constraint to a table

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## Practice 8-5 Overview: Further Practice

This practice covers the following topics:

- Creating a new table
- Adding, modifying, and removing column indexes and constraints
- Removing the table

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# 9

## Table Data Manipulation

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Delete and modify table data
- Add new data to a table by using the `INSERT` and `REPLACE` statements
- Modify data in a table by using the `UPDATE` statement
- Remove table rows by using the `DELETE` statement

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# Manipulation of Table Row Data

- Data in a database is dynamic:
  - Row data can be inserted, updated, replaced, or deleted.
  - Care must be taken to prevent data loss or damage.
- Take these precautions:
  - Grant change access only to users that need it.
  - Keep regular backups.
  - Make additional backups as required.
  - Set the “safe updates” option.
  - Always think in terms of rows (the entire record is affected).
  - Test the scope of a change first with a `SELECT . . . WHERE` statement.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Data manipulation can be unsafe. Take the following precautions with your data:

- Do not grant users (including yourself) more permissions than they need. For example, if you are running some queries on a database for the finance department, do not use the MySQL `root` account. Instead, create a user who has permission to run only the required `SELECT` queries and then log in as that user.
- Keep daily backups.
- Create backups before you make any major changes or use any unfamiliar features.
- The safe updates option is useful for beginners. Enable it with the command-line client option `--safe-updates`, or `SET SQL_SAFE_UPDATES=1` within the `mysql` client program. This option stops you from running a `DELETE` statement without a `WHERE` clause. If you do not set this option and execute a `DELETE` statement without a `WHERE` clause, it will delete every record in the table.
- Test statements that will affect your data on a copy of the table before running them on the real table.
- Use a `SELECT` with the same `WHERE` clause to make sure you retrieve the right records before you change them with a `DELETE` or `UPDATE` statement.

# INSERT Statement

- Populates a table with row data
- Syntax:

```
INSERT INTO table_name (<column_list>
VALUES (<value_list>)
```

- Example (inserts a single row):

```
INSERT INTO CountryLanguage (CountryCode,
Language, Percentage)
VALUES ('ALB', 'Chinese', 20);
```

- Adds the following record:

CountryCode	Language	IsOfficial	Percentage
...	...	...	...
ALB	Chinese	F	20.0
...	...	...	...

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The statement syntax to insert a single record is `INSERT INTO`, followed by the table name, the columns affected, and the values to be inserted into those columns. The number of columns and values must be the same.

The example does not provide an entry for the `IsOfficial` column, so it is given the default value for that column, which is `F` (false).

## Using INSERT for Multiple Rows

- Specify values for each row.
- Example:

```
INSERT INTO CountryLanguage (CountryCode, Language)
VALUES ('GRL', 'MySQL'), ← 1st row
      ('FJI', 'MySQL'), ← 2nd row
      ('BEL', 'MySQL'); ← 3rd row
```

– Content of the new row:

CountryCode	Language	IsOfficial	Percentage
BEL	MySQL	F	0.0
FJI	MySQL	F	0.0
GRL	MySQL	F	0.0

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The statement in the example adds new `CountryLanguage` records for Greenland, Fiji, and Belgium. It assigns a country code and “MySQL” as the language for each. Because it does not assign values for the `Percentage` or `IsOfficial` columns, they receive the column defaults.

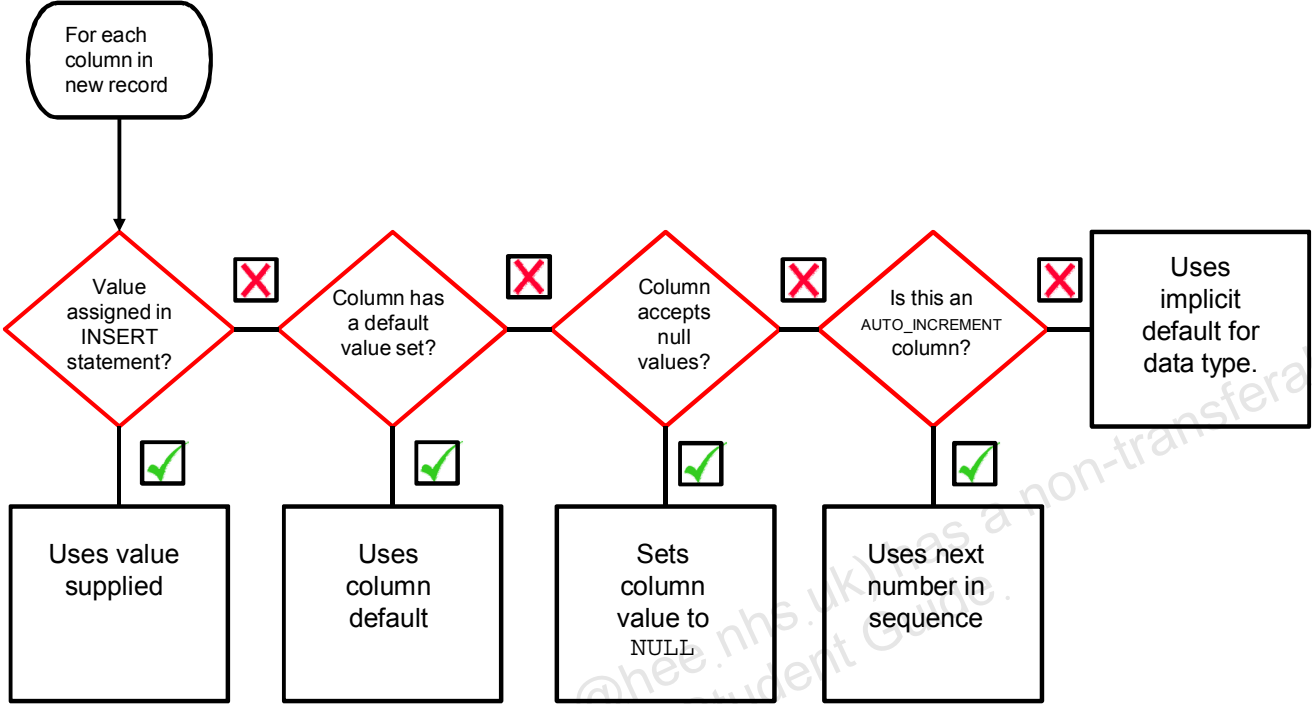
The row data for each of the three rows is enclosed in parentheses.

The following query lists the new records:

```
SELECT * FROM CountryLanguage
WHERE CountryCode IN ('GRL', 'FJI', 'BEL')
AND LANGUAGE = 'MySQL';
```

**Note:** You must enclose column values for string and temporal data types in single quotation marks.

# Column Value Assignment with INSERT



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

## REPLACE Statement

- Use in tables with primary key or unique constraint.
  - It replaces the existing row with new values.
  - In tables without a primary key or unique constraint, it acts in the same way as an `INSERT` statement.
- `REPLACE` is a MySQL extension to the SQL standard.
- General syntax:

```
REPLACE INTO table_name (<column_list>
VALUES (<value_list>)
```

- Example:

```
REPLACE INTO CountryLanguage (CountryCode,
Language, Percentage)
VALUES ('ALB', 'Albaniana', 78.1);
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Unless the table has a `PRIMARY KEY` or a `UNIQUE` constraint, using a `REPLACE` statement makes no sense. It is equivalent to `INSERT` because MySQL needs an index to determine if the new row duplicates an existing one.

You supply column values in the `REPLACE` statement. Like `INSERT`, any missing columns are set to their default values. You cannot refer to values from the current row and use them in the new row.

In the example in the slide, you use the `REPLACE` statement to overwrite the existing Albaniana record with a new one. The new record compensates for the addition of Chinese in an earlier step because it has a different `Percentage` value for Albaniana.

The results of this change are as follows:

```
mysql> SELECT * FROM CountryLanguage WHERE CountryCode = 'ALB';
+-----+-----+-----+-----+
| CountryCode | Language | IsOfficial | Percentage |
+-----+-----+-----+-----+
| ALB        | Albaniana | F          | 78.1      |
| ALB        | Chinese   | F          | 20.0      |
| ALB        | Greek     | F          | 1.8       |
| ALB        | Macedonian | F         | 0.1       |
+-----+-----+-----+-----+
```

## REPLACE Results

Example:

```
Query OK, 2 rows affected (0.06 sec)
```

- Returns a count to indicate the number of rows affected
- Count is the sum of the rows deleted and inserted:
  - Count of 1: One row was inserted. No rows were deleted.
  - Count greater than 1: One or more old rows were deleted before the new row was inserted.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A single row can replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

## REPLACE Algorithm

MySQL performs the following steps for **REPLACE**:

1. It tries to insert the new row into the table.
2. If insertion fails because a duplicate-key error occurs (for a primary key or unique constraint), it deletes the conflicting row.
3. It tries again to insert the new row into the table.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Caution:** If a table has more than one constraint (unique or primary key), replacing one row can delete several rows, if there are conflicts on more than one constraint.

The main difference between **REPLACE** and **INSERT** is that **INSERT** only adds new data, but **REPLACE** can also change existing data.

To use **REPLACE**, you must have **INSERT** and **DELETE** privileges on the table.

## Quiz

The only difference between using **INSERT** and **REPLACE** is that **REPLACE** always deletes a row before inserting a new one.

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: b**



# UPDATE Statement

- Modifies contents of existing rows
- Is used with the **SET** clause to assign new values
- General syntax:

```
UPDATE table_name SET column=expression  
[,column=expression,...]  
[WHERE condition] [other_clauses]
```

- Example:

```
mysql> UPDATE Country  
-> SET Population = Population * 2,  
->      Region = 'Dolphin Country'  
-> WHERE Code = 'SWE';  
Query OK, 1 row affected (0.03 sec)  
Rows matched: 1  Changed: 1  Warnings: 0
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example shows an **UPDATE** of the **Country** table. The **Population** column value is doubled and the **Region** changed for Sweden (**Code** = **SWE**).

The results show that only one row was affected.

- Matched: One row was retrieved for the **WHERE** clause.
- Changed: One row's values were updated.

## UPDATE Statement Ordering

- There is no guarantee about the order in which rows are updated. This can result in errors:

```
mysql> UPDATE City SET ID = ID+1;
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

- Use **ORDER BY** to control the order of updates:

```
mysql> UPDATE City SET ID = ID+1
-> ORDER BY ID DESC;
Query OK, 4079 rows affected (0.13 sec)
Rows matched: 4079 Changed: 4079 Warnings: 0
```

- Content of the new row:

ID	Name	CountryCode	District	Population
2	Kabul	AFG	Kabul	1780000
3	Qandahar	AFG	Qandahar	237500
4	Herat	AFG	Herat	186800
...				

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use an **ORDER BY** clause to force row updates to occur in a particular order.

The results shown in the slide are a result of the following query:

```
mysql> SELECT * FROM City ORDER BY ID;
+----+-----+-----+-----+-----+
| ID  | Name          | CountryCode | District      | Population |
+----+-----+-----+-----+-----+
| 2   | Kabul         | AFG         | Kabul         | 1780000   |
| 3   | Qandahar     | AFG         | Qandahar     | 237500    |
| 4   | Herat        | AFG         | Herat        | 186800    |
| 5   | Mazar-e-Sharif | AFG         | Balkh        | 127800    |
| 6   | Amsterdam    | NLD         | Noord-Holland | 731200    |
...
| 4079 | Nablus       | PSE         | Nablus       | 100231    |
| 4080 | Rafah        | PSE         | Rafah        | 92020     |
+----+-----+-----+-----+-----+
4079 rows in set (0.02 sec)
```

**Note:** Because of the update, the first ID is now 2.

## UPDATE with LIMIT

- Use **LIMIT** to control the number of rows updated:

```
mysql> UPDATE City SET ID = ID-1 LIMIT 1;
```

- The content of the row:

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
3	Qandahar	AFG	Qandahar	237500
4	Herat	AFG	Herat	186800
...				

- Note that only the first ID was affected by the update.

- **UPDATE** can include **ORDER BY** used with **LIMIT**.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide example demonstrates using a **LIMIT** of 1 to affect the first record in the table and leave the rest unchanged.

**Note:** It is not a good practice to update a primary key value.

## When UPDATE Has No Effect

**UPDATE** has no effect when:

- No rows are matched:
  - None match the **WHERE** clause.
  - The table is empty.
- No column values are changed:
  - The new value is the same as the existing one.
  - No rows in the table contain the specified key values.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Difference Between UPDATE and REPLACE

When handling rows with unique key values, **UPDATE** and **REPLACE** are not equivalent:

UPDATE	REPLACE
Does nothing if there is no existing row with specified key values	Adds a record if there is no existing row with specified key values
Can change some column values in an existing row and leave others unchanged	Replaces the entire row*

\*If you want to keep the same values for any columns, you need to supply those values in the **REPLACE** statement.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## DELETE Statement

- Removes entire rows from a table
- General syntax:

```
DELETE FROM table_name  
[WHERE condition]  
[ORDER BY ...]  
[LIMIT row_count]
```

- Use **WHERE** to specify which rows to remove.
- Use the table name without a **WHERE** clause to remove all rows in the table:

```
DELETE FROM my_table
```

- Use **DELETE** with extreme caution; it cannot be reversed.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The **DELETE** statement returns the number of rows affected.

## DELETE with ORDER BY and LIMIT

- Control the order and number of records deleted by using **ORDER BY** and **LIMIT**:
- Example (using **ORDER BY ... DESC**):

```
DELETE FROM CountryLanguage
WHERE Language = 'MySQL'
ORDER BY CountryCode
DESC LIMIT 1;
```

- Affects the following row:

	CountryCode	Language	IsOfficial	Percentage
	BEL	MySQL	F	0.0
	FJI	MySQL	F	0.0
Removed →	GRL	MySQL	F	0.0

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**LIMIT** restricts the number of records that will be deleted. They are deleted from the top of the result set, so use **ORDER BY** to ensure that they are in the correct order.

The **DELETE** statement in the slide retrieves all records from the **CountryLanguage** table where the **Language** is “MySQL,” in descending order of **CountryCode**. It then deletes the first record.

A **SELECT** statement listing the remaining records in the default sort order confirms the deletion:

```
mysql> SELECT * FROM CountryLanguage WHERE Language = 'MySQL';
+-----+-----+-----+-----+
| CountryCode | Language | IsOfficial | Percentage |
+-----+-----+-----+-----+
| BEL        | MySQL   | F          | 0.0        |
| FJI        | MySQL   | F          | 0.0        |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

**Note:** If you do not use the **ORDER BY** clause in conjunction with **LIMIT**, MySQL cannot guarantee which one of the items will be deleted.

## DELETE with ORDER BY and LIMIT

- Example (using ORDER BY ... ASC):

```
DELETE FROM CountryLanguage
WHERE Language = 'MySQL'
ORDER BY CountryCode
ASC LIMIT 1;
```

- Affects the following row:

	CountryCode	Language	IsOfficial	Percentage
Removed →	BEL	MySQL	F	0.0
	FJI	MySQL	F	0.0
	GRL	MySQL	F	0.0

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this example, records are retrieved in ascending order of CountryCode and the first one is deleted.



# Summary

In this lesson, you learned how to:

- Delete and modify table data
- Add new data to a table by using the `INSERT` and `REPLACE` statements
- Modify data in a table by using the `UPDATE` statement
- Remove table rows by using the `DELETE` statement

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 9-1 Overview: Inserting and Replacing Table Row Data

In this practice you:

- Add table data by using `INSERT`
- Replace existing row data by using `REPLACE`

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## Practice 9-2 Overview: Modifying and Deleting Table Row Data

In this practice you:

- Modify existing row data
- Delete specific rows from a table

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 9-3 Overview: Manipulating Table Row Data in the `Pets` Database

In this practice you work with the `Pets` database to:

- Add row data
- Replace row data
- Modify existing rows
- Delete rows

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# 10

## Functions

### Built-In and Group Functions

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Use built-in functions in MySQL
- Describe and use:
  - String functions
  - Temporal functions
  - Numeric functions
  - Control flow functions
- Use aggregate functions with the `SELECT` statement

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Functions in MySQL Expressions

- Functions perform calculations on data.
- A function returns a value that can be used as part of an expression.
- General syntax:

```
function_name([<arg1> [, <arg2>, ..., <argn>]])
```

- You must include the parentheses, even if there are no arguments.
- Separate multiple arguments with commas.
- Examples:
  - **SELECT NOW()**: Returns the current date and time
  - **SELECT VERSION()**: Returns the MySQL Server version currently being used on the host

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The general syntax of a function call is the name of the function followed by parentheses. The parentheses contain any arguments that the function requires, separated by commas.

You can put spaces around function arguments but there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns with the same name as a function.

For more information about functions, see the MySQL Reference Manual at <http://dev.mysql.com/doc/refman/5.6/en/functions.html>.

## Using Functions

- Functions can be used anywhere a value expression is accepted.
- Most functions require arguments.
- Columns (of the appropriate data type) can be used as arguments.
- The output of one function can be used as the input of another function.
- An expression with null values always produces a null output.
  - With rare, documented exceptions
- Mathematical functions return a null value on error
  - For example, division by zero

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# Types of Functions

- **String:** Operations on character strings
- **Temporal:** Operations on dates and times
- **Numeric:** Mathematical operations
- **Control Flow:** Choose between different values based on the result of an expression
- **Aggregate:** Return a single value based on multiple values in a column

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The following slides cover these different types of functions in detail.

# String Functions

- Perform operations on strings, such as:
  - Calculating string lengths
  - Extracting pieces of strings
  - Searching for or replacing substrings
  - Performing case conversion
- String functions are divided into two categories:
  - **Numeric:** Return numbers
  - **String:** Return strings

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## String Functions (Numeric): Examples (CHAR\_LENGTH, INSTR, STRCMP)

- Returns the number of characters in the string:

```
mysql> SELECT CHAR_LENGTH('MySQL');
+-----+
| CHAR_LENGTH('MySQL') |
+-----+
|                      5 |
+-----+
```

- Returns the position in the string where substring occurs:

```
mysql> SELECT INSTR('MySQL', 'SQL');
+-----+
| INSTR('MySQL', 'SQL') |
+-----+
| 3 |
+-----+
```

- Returns results of string sort comparison (0=same, -1=smaller, 1=other):

```
mysql> SELECT STRCMP('abc','def'),
-> STRCMP('def','def'),
-> STRCMP('def','abc');
+-----+-----+-----+
| STRCMP('abc','def') | STRCMP('def','def') | STRCMP('def','abc') |
+-----+-----+-----+
| -1 | 0 | 1 |
+-----+-----+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### Usage:

- CHAR\_LENGTH(<string>)
- INSTR(<string>, <substring>): Returns 0 if the substring is not in the specified string
- STRCMP(<arg1>, <arg2>): Returns 0 if arguments are the same, -1 if arg1 is smaller than arg2 (according to the current sort order), and 1 otherwise

## String Functions: Examples (CONCAT, REVERSE, LEFT, RIGHT)

- Concatenates the given arguments into one string:

```
mysql> SELECT CONCAT('A', '-', 'Z');
+-----+
| CONCAT('A', '-', 'Z') |
+-----+
| A-Z                    |
+-----+
```

- Returns string with the characters in reverse order:

```
mysql> SELECT REVERSE('MySQL');
+-----+
| REVERSE('MySQL')      |
+-----+
| LQSyM                 |
+-----+
```

- Returns the specified number of characters from the left of a string

```
mysql> SELECT LEFT('MySQL', 3);
+-----+
| LEFT('MySQL', 3)     |
+-----+
| MyS                  |
+-----+
```

- Returns the specified number of characters from the right of a string

```
mysql> SELECT RIGHT('MySQL', 3);
+-----+
| RIGHT('MySQL', 3)    |
+-----+
| SQL                  |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### Usage:

- CONCAT(<arg1> [, <arg2>, ..., <argn>])
- REVERSE(<string>)
- RIGHT(<string>, <length>)
- LEFT(<string>, <length>)

## String Functions: Examples (LOWER, UPPER, LPAD, RPAD)

- Returns the string with all characters in lowercase:

```
mysql> SELECT LOWER('MySQL');
+-----+
| LOWER('MySQL') |
+-----+
| mysql          |
+-----+
```

- Returns the string with all characters in uppercase:

```
mysql> SELECT UPPER('MySQL');
+-----+
| UPPER('MySQL') |
+-----+
| MYSQL          |
+-----+
```

- Returns string left-padded with the specified characters:

```
mysql> SELECT LPAD('MySQL', 8, '.');
+-----+
| LPAD('MySQL', 8, '.') |
+-----+
| ...MySQL             |
+-----+
```

- Returns string right-padded with the specified characters:

```
mysql> SELECT RPAD('MySQL', 8, '.');
+-----+
| RPAD('MySQL', 8, '.') |
+-----+
| MySQL...              |
+-----+
```

**ORACLE**

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### Usage:

- LOWER(<string>)
- UPPER(<string>)
- LPAD(<string>, <length>, <padding>)
- RPAD(<string>, <length>, <padding>)

## String Functions: Examples (TRIM)

- Removes all leading and trailing spaces from the string:

```
mysql> SELECT TRIM('  MySQL  ') AS str;
+-----+
| str   |
+-----+
| MySQL |
+-----+
```

- Removes the specified characters from the beginning of the string:

```
mysql> SELECT TRIM(LEADING 'Q' FROM 'QQQMySQLQQQ');
+-----+
| TRIM(LEADING 'Q' FROM 'QQQMySQLQQQ') |
+-----+
| MySQLQQQ                             |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### Usage:

- TRIM( [[LEADING | TRAILING] <string> FROM ] <fullstring>):
  - TRIM without the FROM clause removes all spaces from the beginning and end of <fullstring>.
  - With the FROM clause, TRIM removes all <string> characters from the beginning and/or end (LEADING or TRAILING or BOTH) of <fullstring>.

## String Functions: Examples (SUBSTRING)

- Returns part of a string
  - From a specified starting position to the end of a string

```
mysql> SELECT SUBSTRING('MySQL', 3);
+-----+
| SUBSTRING('MySQL', 3) |
+-----+
| SQL                 |
+-----+
```

- A specified number of characters from the starting position.

```
mysql> SELECT SUBSTRING('MySQL', 2, 2);
+-----+
| SUBSTRING('MySQL', 2, 2) |
+-----+
| yS                       |
+-----+
```

ORACLE

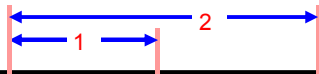
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### Usage:

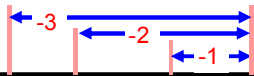
SUBSTRING(<string>, <startpos> [, <chars>])

## String Functions: Examples (SUBSTRING\_INDEX)

- Returns part or all of a string based on a delimiter
- You provide the delimiter character(s) and a count of delimiter “stops”.



```
mysql> SELECT SUBSTRING_INDEX('training@mysql.com', '@', 1);
+-----+
| SUBSTRING_INDEX('training@mysql.com', '@', 1) |
+-----+
| training                                     |
+-----+
```



```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
+-----+
| SUBSTRING_INDEX('www.mysql.com', '.', -2) |
+-----+
| mysql.com                                 |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

### Usage:

SUBSTRING\_INDEX(<string>, <delimiter>, <count>): The search is from left to right if <count> is positive and right to left if <count> is negative.



# Temporal Functions

- Perform operations such as:
  - Extracting parts of dates and times
  - Reformatting values
  - Converting values to seconds or days
- You can generate temporal data in many ways:
  - Copy existing data.
  - Use built-in functions.
  - Build a string representation for the server to evaluate.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Temporal Functions: Date/Time Formats

Type	Default Format
<b>DATE</b>	YYYY-MM-DD
<b>TIME</b>	HH:MM:SS
<b>DATETIME</b>	YYYY-MM-DD HH:MM:SS
<b>TIMESTAMP</b>	YYYY-MM-DD HH:MM:SS
<b>YEAR</b>	YYYY

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide lists the date and time formats you can use with temporal functions.

See the lesson about data types for specific information about the values and ranges for these types.

Functions that expect date values usually accept `DATETIME` values and ignore the time part.

Functions that expect time values usually accept `DATETIME` values and ignore the date part.

## Temporal Functions: Function Types

Function Syntax	Return value
<b>NOW ()</b>	Current date and time as set on the server host ( <b>DATETIME</b> format)
<b>CURDATE ()</b>	Current date as set on the server host ( <b>DATE</b> format)
<b>CURTIME ()</b>	Current time as set on the server host ( <b>TIME</b> format)
<b>YEAR (&lt;date_expression&gt;)</b>	Year in four-digit <b>YEAR</b> format, from the date you supply as an argument.
<b>MONTH (&lt;date_expression&gt;)</b>	Month of the year in integer format, from the date you supply as an argument
<b>DAYOFMONTH (&lt;date_expression&gt;)</b> or <b>DAY (&lt;date_expression&gt;)</b>	Day of the month in integer format, from the date you supply as an argument
<b>DAYNAME (&lt;date_expression&gt;)</b>	Day of the week in natural language format, from the date you supply as an argument
<b>HOURL (&lt;date_expression&gt;)</b>	Hour of the day in integer format (in 0–23 range), from the date you supply as an argument
<b>MINUTE (&lt;date_expression&gt;)</b>	Minute of the day in integer format, from the date you supply as an argument
<b>SECOND (&lt;date_expression&gt;)</b>	Second of the minute in integer format, from the date you supply as an argument

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Temporal Functions: Examples

- Return current date, time, and day of week:

```
mysql> SELECT CURDATE(), CURTIME(), DAYNAME(NOW());
+-----+-----+-----+
| CURDATE() | CURTIME() | DAYNAME(NOW()) |
+-----+-----+-----+
| 2012-02-06 | 17:55:40 | Friday          |
+-----+-----+-----+
```

- Add specified number of days to current date:

```
mysql> SELECT NOW(), NOW() + INTERVAL 5 DAY;
+-----+-----+
| NOW() | NOW() + INTERVAL 5 DAY |
+-----+-----+
| 2012-02-06 18:02:35 | 2006-02-11 18:02:35 |
+-----+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Temporal Functions: Examples

Change the output date format:

```
mysql> SELECT NOW(), DATE_FORMAT(NOW(), '%W the %D of %M');
+-----+-----+
| NOW()                | DATE_FORMAT(NOW(), '%W the %D of %M') |
+-----+-----+
| 2012-02-06 18:02:35 | Friday the 6th of February           |
+-----+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Syntax and additional information for the temporal functions:

- DATE\_FORMAT (<date>, <format>)
- Prefix the date/time formats with %.
  - Others include %H (hour), %p (AM or PM), %T (24 hour clock).
  - For other formats, see the MySQL Reference Manual at [http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function\\_date-format](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_date-format).

Ashley Mansoo (ashley.mansoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Numeric Functions

Perform mathematical operations such as:

- Rounding
- Truncation
- Trigonometric calculations
- Generating random numbers

Function Syntax	Returns
<b>ABS</b> (<number>)	Absolute value of the number
<b>SIGN</b> (<number>)	-1, 0, or 1 depending on whether the number is negative, zero, or positive
<b>TRUNCATE</b> (<number>, <decimals>)	The number truncated to decimals
<b>FLOOR</b> (<number>)	The number rounded down to the closest lower integer
<b>CEILING</b> (<number>)	The number rounded up to the closest higher integer
<b>ROUND</b> (<number>)	The number rounded to the closest integer

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**ROUND** (<number>):

- Works with **DECIMAL** data type
- Defaults to zero, if not specified
- For exact-value numbers, a value with a fractional part less than .5 is rounded down to the next integer if positive, or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that a value with any fractional part is rounded to the nearest even integer.

## Numeric Functions: Examples

- Returns the absolute value of the negative and positive values:

```
mysql> SELECT ABS(-42), ABS(42);
+-----+-----+
| ABS(-42) | ABS(42) |
+-----+-----+
|         42 |         42 |
+-----+-----+
```

- Returns sign (-1=negative, 0=zero, 1=positive):

```
mysql> SELECT SIGN(-42), SIGN(-1), SIGN(0), SIGN(1), SIGN(42);
+-----+-----+-----+-----+-----+
| SIGN(-42) | SIGN(-1) | SIGN(0) | SIGN(1) | SIGN(42) |
+-----+-----+-----+-----+-----+
|      -1   |      -1   |       0   |       1   |       1   |
+-----+-----+-----+-----+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The absolute value of a number is its distance from zero, without considering which direction from zero the number lies. An absolute number is always zero or positive.

## Numeric Functions: Additional Functions

- Geometric functions:
  - `DEGREES ()`, `PI ()`, `RADIANS ()`
- Trigonometric functions:
  - `COS ()`, `SIN ()`, `COT ()`
  - `ACOS ()`, `ASIN ()`, `ATAN ()`, `ATAN2 ()`
- Other functions:
  - `EXP ()`, `LN ()`, `LOG ()`, `LOG2 ()`, `LOG10 ()`
  - `POWER ()`, `SQRT ()`
  - `MOD ()`

ORACLE

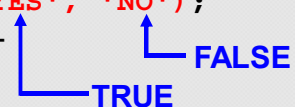
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# Control Flow Functions

- Choose between different values based on the result of an expression.
- **IF ()** function example:

```
mysql> SELECT IF(1 > 0, 'YES', 'NO');
+-----+
| IF(1 > 0, 'YES', 'NO') |
+-----+
| YES                     |
+-----+
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## Control Flow Functions: CASE Functions

- The **CASE** function executes a series of conditional tests.
- Each test returns a Boolean result.
- The result determines the flow of control.
- It can be used in two different ways:
  - To compare expression values
  - To compare expression conditions

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Control Flow Functions: CASE Function Syntax

- Expression value comparison example:

```
CASE value
  WHEN <compare_value> THEN <result>
  [WHEN <compare_value> THEN <result> ...]
  [ELSE <result>]
END
```

- Expression condition evaluation example:

```
CASE
  WHEN <condition> THEN <result>
  [WHEN <condition> THEN <result> ...]
  [ELSE <result>]
END
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The first example in the slide returns the result where `value = compare_value`. The second slide example returns the result for the first condition it tests that is true. If no conditions are true, it returns the result after `ELSE`, or `NULL` if there is no `ELSE`.

# Control Flow Functions: CASE Function Examples

```
mysql> SELECT CASE 3 WHEN 1 THEN 'one'
-> WHEN 2 THEN 'two' ELSE 'more' END;
'more'
```

```
mysql> SELECT CASE WHEN 1>0 THEN 'true'
-> ELSE 'false' END;
'true'
```

```
mysql> SELECT CASE BINARY 'B'
-> WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
NULL
```

```
SELECT pName, oID, pGender,
CASE
WHEN pGender = 'm' THEN 'Boy'
WHEN pGender = 'f' THEN 'Girl'
ELSE 'Unknown'
END
FROM pet_info
ORDER BY pGender;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The default return type of a CASE expression is the compatible aggregated type of all return values, but the return type also depends on the context in which you use CASE. If you use it in a string context, the result is returned as a string. If used in a numeric context, the result is returned as a decimal, real, or integer value.

The last example in the slide assumes that you want to display a pet's gender as Boy or Girl instead of M or F. You can use the CASE function to return "Boy" when pGender = 'M' and "Girl" when pGender = 'F':

pName	oID	pGender	CASE
Claws	2	M	Boy
Fluffy	1	F	Girl
Buffy	1	F	Girl

## Quiz

The `CHAR_LENGTH()` function is part of the \_\_\_\_\_ function category.

- a. String
- b. Temporal
- c. Numeric
- d. Control Flow
- e. Aggregate

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

# Aggregate Functions

- Perform summary operations on a set of values, such as:
  - Counting
  - Averaging
  - Finding minimum or maximum values
- Return a single value based on multiple values from different rows
- Include only non-null values in results

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Aggregate functions are also known as group functions. Use them when you want a summary of the data in all rows, rather than the individual row values. These functions consider only values that are not null, except `COUNT (*)`, which includes all rows.

# Aggregate Function Types

Some of the aggregate function types:

Function Syntax	Definition
<b>MIN</b> ( <i>&lt;column_name&gt;</i> )	Finds the smallest value
<b>MAX</b> ( <i>&lt;column_name&gt;</i> )	Finds the largest value
<b>SUM</b> ( <i>&lt;column_name&gt;</i> )	Calculates the sum of values in <i>&lt;column_name&gt;</i>
<b>AVG</b> ( <i>&lt;column_name&gt;</i> )	Calculates the average value of <i>&lt;column_name&gt;</i>
<b>COUNT</b> ( <i>&lt;column_name&gt;</i> )	Counts non-null values in <i>&lt;column_name&gt;</i> . COUNT (*) includes records with null values.
<b>GROUP_CONCAT</b> ( <i>&lt;column_name&gt;</i> )	Concatenates a set of strings to produce a single string

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

All the functions can use the `DISTINCT` keyword, although it is not useful for the `MAX()` and `MIN()` functions.

`DISTINCT` examples:

```
SUM(DISTINCT <column_name>)
```

```
AVG(DISTINCT <column_name>)
```

```
COUNT(DISTINCT <column_name>)
```

```
GROUP_CONCAT(DISTINCT <column_name>)
```

## Aggregate Functions: COUNT Function Examples

- Retrieves a count of all rows in the `Country` table:

```
mysql> SELECT COUNT(*) FROM Country;
+-----+
| COUNT(*) |
+-----+
|         239 |
+-----+
```

- Retrieves a count of all non-null values in the `Capital` column:

```
mysql> SELECT COUNT(Capital) FROM Country;
+-----+
| COUNT(Capital) |
+-----+
|             232 |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The second example in the slide returns a different result, because not every country has a capital city associated with it. The count does not include null values in the `Capital` column.



## Aggregate Functions: GROUP BY Clause

- The **GROUP BY** clause places rows into groups.
  - Each group consists of rows that have the same value in one or more columns.
  - It calculates an aggregate value for each group.
- Example:

```
mysql> SELECT Continent, AVG(Population)
-> FROM Country
-> GROUP BY Continent;
```

Continent	AVG(Population)
Asia	72647562.7451
Europe	15871186.9565
North America	13053864.8649
Africa	13525431.0345
Oceania	1085755.3571
Antarctica	0.0000
South America	24698571.4286

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If a **WHERE** clause selects 20 rows and **GROUP BY** arranges them into four groups of five rows each, aggregate functions produce a value for each of the four groups.

Without a **GROUP BY** clause, an aggregate function calculates the aggregate value based on all selected rows. That is, MySQL treats all rows as a single group:

```
mysql> SELECT AVG(Population) FROM COUNTRY;
```

```
+-----+
| AVG(Population) |
+-----+
| 25434098.1172 |
+-----+
```

```
1 row in set (0.00 sec)
```

The example in the slide groups the **Country** table rows by continent, and calculates the average population of countries in each continent.

## Aggregate Functions: GROUP BY Clause and GROUP\_CONCAT Function

- The GROUP\_CONCAT () function concatenates results.
- Example:

```
mysql> SELECT GovernmentForm, GROUP_CONCAT(Name)
-> AS Countries
-> FROM Country WHERE Continent = 'South America'
-> GROUP BY GovernmentForm\G
***** 1. row *****
GovernmentForm: Dependent Territory of the UK
Countries: Falkland Islands
***** 2. row *****
GovernmentForm: Federal Republic
Countries: Argentina,Venezuela,Brazil
***** 3. row *****
GovernmentForm: Overseas Department of France
Countries: French Guiana
***** 4. row *****
GovernmentForm: Republic
Countries: Chile,Uruguay,Suriname,Peru,Paraguay,Bolivia,
Guyana,Ecuador,Colombia
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide creates a list of the countries for each form of government on the South American continent.

## Aggregate Functions: GROUP BY and HAVING Clauses

- Use the **HAVING** clause to filter rows based on aggregate values.
  - Evaluated after the grouping implied by **GROUP BY**
- Example:

```
mysql> SELECT Continent, SUM(Population)
-> FROM Country
-> GROUP BY Continent
-> HAVING SUM(Population) > 100000000;
```

Continent	SUM(Population)
Asia	3705025700
Europe	730074600
North America	482993000
Africa	784475000
South America	345780000

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use the **HAVING** modifier to require that the groups a **GROUP BY** clause produces satisfy particular criteria.

It is similar to the **WHERE** clause. The difference is that MySQL evaluates the **HAVING** clause after the grouping implied by the **GROUP BY** clause. This means that the **HAVING** condition can refer to aggregate functions. Do not put any part of a condition in the **HAVING** clause that could also appear in the **WHERE** clause. A good **HAVING** clause is always based on aggregate functions (because these are not allowed in the **WHERE** clause).

The slide example results in a list of continents whose sum (aggregate value) of country populations is greater than 100,000,000.

## Aggregate Functions: GROUP BY Clause and WITH ROLLUP Modifier

- Use the WITH ROLLUP modifier to produce multiple levels of aggregate values.
- Example:

```
mysql> SELECT Continent, SUM(Population)
-> FROM Country
-> GROUP BY Continent
-> WITH ROLLUP;
```

Continent	SUM(Population)
Asia	3705025700
Europe	730074600
North America	482993000
Africa	784475000
Oceania	30401150
Antarctica	0
South America	345780000
NULL	6078749450

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

WITH ROLLUP adds extra rows to the aggregate output. These extra rows display the results of higher-level (or super-aggregate) aggregate operations.

The example in the slide shows the population of each continent and the total population of all continents.

Another way to do this is to run one query to get the per-continent totals and another to get the total for all continents. Using WITH ROLLUP lets you use a single query to get both.

# Aggregate Functions: Super-Aggregate Operation

- Use the `WITH ROLLUP` and the `AVG()` function.
  - Produce a final line that comprises the application of the given aggregate function
- Example:

```
mysql> SELECT Continent, AVG(Population)
-> FROM Country
-> GROUP BY Continent WITH ROLLUP;
```

Continent	AVG(Population)
Asia	72647562.7451
Europe	15871186.9565
North America	13053864.8649
Africa	13525431.0345
Oceania	1085755.3571
Antarctica	0.0000
South America	24698571.4286
NULL	25434098.1172

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Adding a `WITH ROLLUP` modifier to the `GROUP BY` clause performs a super-aggregate operation. It forces the query to produce another row that shows the overall average, rather than just the sum of all the averages.

**Note:** `ORDER BY` does not work with `WITH ROLLUP`.

## Spaces in Function Names

- By default, there must be no space between a function name and the parenthesis:

```
mysql> SELECT PI ();  
ERROR 1305 (42000): FUNCTION world.PI does not exist
```

- You can change this by using the `IGNORE SPACE` SQL mode:

```
mysql> SET sql_mode = 'IGNORE_SPACE';  
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> SELECT PI ();  
+-----+  
| PI () |  
+-----+  
| 3.141593 |  
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Summary

In this lesson, you learned how to:

- Use built-in functions in MySQL
- Describe and use:
  - String functions
  - Temporal functions
  - Numeric functions
  - Control flow functions
- Use aggregate functions with the `SELECT` statement

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 10-1 Overview: Quiz

In this practice, you answer questions about MySQL functions.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



## Practice 10-2 Overview: Using Built-In, String, and Temporal Functions

In this practice you use built-in, string, and temporal functions.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 10-3 Overview: Using Numeric and Control Flow Functions

In this practice, you use numeric and control flow statements.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 10-4 Overview: Using Aggregate Functions

This practice covers the following topics:

- Using aggregate functions
- Using the `GROUP BY` clause with options

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# 11

## Exporting and Importing Data

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Export data by using a query
- Export data by using the `mysqldump` database backup client
- Import data with MySQL statement files
- Import data by using the `mysqlimport` client program

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Exporting Data

- Database backups are important!
  - Back up regularly.
  - Use a consistent and meaningful naming scheme.
  - Expire backup files.
  - Include with regular system backups.
- Other reasons for exporting data:
  - Copying databases from one server to another:
    - From within the same host
    - To another host
  - Testing a new MySQL release with real data
  - Transferring data from one RDBMS to another

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use SQL statements or dedicated MySQL backup client programs to import and export data.

You must back up your databases regularly. If a serious system crash occurs, you want to be able to restore your tables to the state they were in at the time of the crash and minimize data loss.

You can also use database backups to copy databases between servers. Usually this is a server running on another host, but can be a different server running on the same host. For example, you might want to test a new version of MySQL with real data from your production server, or load data into external applications. You can also transfer data between different RDBMSs.

## Exporting with a Query

- Write query results directly into a file by using **SELECT** with the **INTO OUTFILE** clause.
- Example:

```
SELECT * INTO OUTFILE 'D:/DataBackup/Country.txt'  
FROM Country
```

- Specify the output file location on the host.
- Default file format:
  - Values are delimited by tabs.
  - Lines are terminated by the newline character.
- Change the file format with **INTO OUTFILE** options.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Put the **INTO OUTFILE** clause before the **FROM** clause.

The example in the slide uses **SELECT . . . INTO FILE** to write the contents of the entire **Country** table into a file named **Country.txt**.



## Exporting with a Query: INTO outfile

**INTO outfile** changes the standard **SELECT** operation:

- It writes the query to the output file instead of returning them to the client.
- The process creates a new output file. It cannot already exist.
- Each line in the output file represents one row of data.
- The user that executes the statement must have the **FILE** privilege.
- It creates the file with file system access permissions.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Exporting with a Query: CSV Format

- The CSV (comma-separated values) format:
  - Separates values with commas
  - Encloses values in double quotation marks
  - Terminates lines with carriage returns
- Example:

```
SELECT * INTO OUTFILE '/tmp/table1.txt'  
FIELDS TERMINATED BY ',' ENCLOSED BY '"'  
LINES TERMINATED BY '\r'  
FROM table1
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Exporting with a MySQL Utility

The `mysqldump` utility “dumps” table contents to text.

- Dump files can contain:
  - Logical backups (SQL statements)
    - Statements re-create table structure, table data, or both.
  - Physical backups (data only)
    - CSV, other delimited text files or XML format
- Control behavior with options, including:
  - Specify entities to dump
  - Specify output format
  - Compress output file

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The dump file usually contains SQL statements to create the table, populate it, or both. However, `mysqldump` can also be used to generate files in CSV, other delimited text, or XML format.

There are three ways to invoke `mysqldump` at the command prompt:

```
mysqldump [options] db_name [tables]
mysqldump [options] --databases db_name1 [db_name2 db_name3...]
mysqldump [options] --all-databases
```

## Specifying Entities to Dump

- Export an entire database (for example, the `world_innodb` database and all its tables):

```
mysqldump -uusername -ppassword world_innodb
```

- Export specific tables (for example, the `City` table in the `world_innodb` database):

```
mysqldump -uusername -ppassword world_innodb City
```

- Export multiple databases (for example, the `world_innodb` and `sakila` databases):

```
mysqldump -uusername -ppassword --databases  
world_innodb sakila
```

- All databases on the MySQL server:

```
mysqldump -uusername -ppassword --all-databases
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you do not specify tables, `mysqldump` dumps the entire database.

## mysqldump Utility Options

- List available options with `mysqldump --help`
- Common options include:
  - `--opt`: Optimizes `mysqldump` for best performance; enabled by default
  - `-C` (`--compress`): Uses compression (good for slow connections)
  - `--tab=dir_name`: Produces tab-separated text-format data files
  - `-d` (`--no-data`): Dumps empty table structures only; no data
  - `-X` (`--xml`): Dumps the data as well-formed XML
  - `--compatible=name`: Works with other databases or older versions of MySQL

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use the `--tab` option to export to multiple files, with one set of files (`.sql` or `.txt`) for each table. The option value is the directory to write the files to.

The `--comment` option writes additional information in the dump file, such as program version, server version, and host. This option is enabled by default. To suppress this additional information, use `--skip-comments`.

The `--compatible` option does not guarantee compatibility with other servers, but sets various SQL modes to help with compatibility. For example, `--compatible=oracle` does not map data types to Oracle types and does not use Oracle comment syntax.

`mysqldump` does not dump the `INFORMATION_SCHEMA` database by default. To dump `INFORMATION_SCHEMA`, name it explicitly on the command line and use the `--skip-lock-tables` option.

## Dumping to a Text File

Dump databases to a text file (.txt) using `mysqldump` with the redirect operator (>) to indicate the file name and location:

- Example: Export the entire `world_innodb` database.

```
mysqldump -uusername -ppassword world_innodb >
world_innodb.txt
```

- Example: Export the `City` and `Country` tables in the `world_innodb` database.

```
mysqldump -uusername -ppassword world_innodb City
Country > world_innodb.txt
```

- Example: Drop the `City` table if it already exists on the target server.

```
mysqldump -uusername -ppassword --add-drop-table
world_innodb City > world_innodb.txt
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

`mysqldump` creates SQL code that assumes that the destination tables do not already exist. The `--add-drop-table` option adds statements to drop a table if it exists in the new database when the file is imported.

Connect to the MySQL server and execute the SQL dump from a client program. For example, use the `mysql` command-line client `SOURCE` command.

`mysqldump` exports the data to a file on the server host. To export data to the client's local machine, use the `mysql` client:

```
cmd> mysql -uroot -p world -e"SELECT * FROM country" > filename
```

For more information about the `mysqldump` utility, see the MySQL Reference Manual: <http://dev.mysql.com/doc/refman/5.6/en/mysqldump.html>.

# Text File Contents

Output from the `--add-drop-table` example statement:

```
-- MySQL dump 10.13  Distrib 5.6.10, for Win64 (x86_64)
--
-- Host: localhost    Database: world_innodb
-- Server version 5.6.10-enterprise-commercial-advanced
--
-- Table structure for table `city`
--
/* ... */
DROP TABLE IF EXISTS `city`;
CREATE TABLE `city` (
  `ID` int(11) NOT NULL AUTO INCREMENT,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
  CONSTRAINT `city_ibfk_1` FOREIGN KEY (`CountryCode`) REFERENCES `country` (`Code`)
) ENGINE=InnoDB AUTO_INCREMENT=4080 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Dumping data for table `city`
--
INSERT INTO `city` VALUES
(1, 'Kabul', 'AFG', 'Kabol', 1780000), (2, 'Qandahar', 'AFG', 'Qandahar', 237500), (3, 'Herat', 'AFG', 'Herat', 186800), (4, 'Mazar-e-Sharif', 'AFG', 'Balkh', 127800),
(5, 'Amsterdam', 'NLD', 'Noord-Holland', 731200), (6, 'Rotterdam', 'NLD', 'Zuid-Holland', 593321), (7, 'Haag', 'NLD', 'Zuid-Holland', 440900), ...
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This shows the partial output from a dump of the `world_innodb` `City` table, with the `--add-drop-table` option, as per the last example in the previous slide.

## Importing Data

You can import data into a database by using a MySQL (.sql) statement file, which contains all the information needed to create tables.

- Example: Import the `world_innodb` database from a statement file by using the `mysql` client program:

```
cmd> mysql -uusername -ppassword world_innodb <  
world_innodb.sql
```

– Use the input operator (<) to indicate the SQL file name.

- Example: Import the data file from within the `mysql` client:

```
mysql> SOURCE D:/DataBackup/world_innodb.sql
```

**Caution:** Do not try to replace an existing database file with an import.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Shell-level client commands do not require a semicolon (;) at the end of the statement.

You can also import data from a text (.txt) file.



## Importing from a Data File

You should know the following characteristics of the data file:

- Column value separator
- Order of the columns
- Row separator
- File system where the file resides
- What characters enclose the values (example: double quotation marks)
- Does the file contain the column names?
- Is there a header indicating rows of the table to skip before importing?
- Do you need certain privileges to access the file?

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Importing with the LOAD DATA INFILE Statement

- **LOAD DATA INFILE** is the opposite of **SELECT...INTO OUTFILE**.
  - However, it uses similar clauses and format specifiers.
- It reads row values from a file into a table.
- Files can be in tab-delimited or comma-separated format.
- Example:

```
LOAD DATA INFILE 'D:/DataBackup/City.txt'  
INTO TABLE City
```

- MySQL assumes that the file resides on the server host in the database data directory.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In its simplest form, the **LOAD DATA INFILE** statement needs only the name of the data file and the table to load it into, as shown in the slide example.

The syntax for **LOAD DATA INFILE** is as follows. Optional items are indicated by square brackets:

```
LOAD DATA [LOCAL] INFILE 'file_name'  
[IGNORE | REPLACE]  
INTO TABLE table_name  
format_specifiers  
[IGNORE n LINES]  
[(column_list)]  
[SET (assignment_list)]
```

## Importing with LOAD DATA INFILE: CSV Format

- Import a text file containing data values separated by commas.
- Example:

```
LOAD DATA INFILE '/tmp/table1.txt'  
INTO TABLE table1  
FIELDS TERMINATED BY ',' ENCLOSED BY ''''  
LINES TERMINATED BY '\r'
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The file name is given as a quoted string. On Windows, the path name separator character is a backslash (“\”), but MySQL treats a backslash in a string as the escape character, so use “/” or “\\” as separators in Windows pathnames. Example:

```
LOAD DATA INFILE 'D:/mydata/data.txt' INTO TABLE t  
LOAD DATA INFILE 'D:\\mydata\\data.txt' INTO TABLE t
```

## Importing with LOAD DATA INFILE: File Control

- MySQL file assumptions and defaults:
  - File resides on the server host.
  - File is in CSV format.
  - Each input line contains a value for each column in the table.
- Optional clauses give you more control:
  - Skipping data file rows
  - Loading specific table columns
  - Skipping or transforming column values
  - Control of duplicate records/rows
  - Tracking of records/rows
  - Control privileges
  - Data file format specification (field and line terminators)
  - Whether to import nulls

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You need the FILE privilege to use LOAD DATA INFILE with server files. To import files on the client host, use LOAD DATA LOCAL INFILE. The client program then reads the data file from the local machine and sends its contents over the network to the server.

## Importing with a MySQL Utility

The `mysqlimport` utility is a command-line interface to `LOAD DATA INFILE`.

- General syntax:

```
mysqlimport <options> db_name <textfile1> [<textfile2...>]
```

- `mysqlimport` matches the file name with the table name, and then uses `LOAD DATA INFILE` to load the file to the table.
- You can import several files at once.
- Input files must contain only data values, not SQL statements.
- Tables must already exist.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Because `mysqlimport` uses `LOAD DATA INFILE` internally, become familiar with the `LOAD DATA INFILE` statement before using it.

`mysqlimport` uses each input file name to determine which table to load the file contents into. It removes any extension and uses the result as the table name. For example, a file named `City.txt` or `City.dat` is input for the `City` table.

## mysqlimport Utility Options

- List available options with `mysqlimport --help`.
- Common options include:
  - `--lines-terminated-by=string`: Specifies the character sequence that each input line ends with. The default is `\n` (newline).
  - `--fields-terminated-by=string`: Specifies the delimiter between data values. The default is `\t` (tab).
  - `--fields-enclosed-by=char`: Specifies character(s) that enclose each value. The double quote character (") is common.
  - `--ignore` or `--replace`: Use to import records with unique key values that are already present in the table.
  - `--local`: Use a local data file on the client host.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## mysqlimport Option: Examples

Examples:

```
mysqlimport --lines-terminated-by="\r\n"  
world_innodb City.txt  
mysqlimport --fields-terminated-by=","  
--lines-terminated-by="\n"  
world_innodb City.txt  
mysqlimport --fields-enclosed-by='''  
world_innodb City.txt
```

Imports different versions of the City.txt file into the City table.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Summary

In this lesson, you learned how to:

- Export data by using a query
- Export data by using the `mysqldump` database backup client
- Import data with MySQL statement files
- Import data by using the `mysqlimport` client program

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



## Practice 11-1 Overview: Quiz

In this practice, you answer questions about exporting and importing data.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 11-2 Overview: Exporting Files by Using a Query

In this practice, you use the `SELECT...INTO OUTFILE` statement to export database data.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 11-3 Overview: Importing Files from a Data File

In this practice, you use the `LOAD DATA INFILE` statement to import database data.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 11-4 Overview: Backing Up Database Files with a Utility

In this practice, you use the `mysqldump` utility to back up a database.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# 12

## Joining Tables

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Explain the concept of a join
- Use the `JOIN` keyword to query multiple tables
- Execute outer and inner joins

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Combining Multiple Tables

- Some queries need data from multiple tables.
- You use a join operation to combine data from different tables.
- A join creates a temporary resultset that contains combined data.
- To join tables, there must be a relationship between certain columns in these tables.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `SELECT` queries shown so far in this course retrieve data from a single table. You cannot answer all questions in this way.

If you want to find the details of records referenced in a foreign key, you combine data from two or more tables with a table join.

# Table Joins

Category	Type	Description
Unqualified Join	Cross Join	Combines all the rows from one table with all the rows from another table
Qualified Join	Inner Join	Identifies combinations of matching rows from two tables
Qualified Join	Outer Join	Identifies combinations of matching and mismatching rows from two tables

- **Unqualified join:** Includes all row pairs
- **Qualified join:** Includes specific row pairs only, according to a particular “join condition”

Example:

- A city (`City` table) is a capital of a country (`Country` table), and a country (`Country` table) has cities (`City` table).

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Most joins are qualified and are used to combine related tables. The join condition expresses a meaningful relationship between the data sets.



# Cross Joins

- Two separate tables:

```
table1
+----+----+
| i1 | c1 |
+----+----+
| 1  | a  |
| 2  | b  |
| 3  | c  |
+----+----+
3 rows in set (0.00 sec)
```

AND

```
table2
+----+----+
| i2 | c2 |
+----+----+
| 2  | c  |
| 3  | b  |
| 4  | a  |
+----+----+
3 rows in set (0.00 sec)
```

- Tables joined by SELECT:

```
SELECT * FROM table1, table2;
+----+----+----+----+
| i1 | c1 | i2 | c2 |
+----+----+----+----+
| 1  | a  | 2  | c  |
| 2  | b  | 2  | c  |
| 3  | c  | 2  | c  |
| 1  | a  | 3  | b  |
| 2  | b  | 3  | b  |
| 3  | c  | 3  | b  |
| 1  | a  | 4  | a  |
| 2  | b  | 4  | a  |
| 3  | c  | 4  | a  |
+----+----+----+----+
9 rows in set (0.00 sec)
```

- This join results in a Cartesian product.
- The result includes all combinations of rows from table1 and table2.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you cross-join two tables, you combine each row from one table with each row from the other table. This yields all possible combinations of rows and is known as the Cartesian product. Joining tables this way can produce a very large number of rows because the row count is the product of the number of rows in each table.

A cross-join between two tables each containing 1000 rows returns  $1000 \times 1000 = 1$  million rows. That is a lot of rows, even though the individual tables are small. This type of join is also called an “unqualified join”.

## Multiple Tables in the FROM Clause

- Example of two separate queries that can be joined:

```
SELECT Code, Name
FROM Country
WHERE Continent = 'Africa';
```

AND

```
SELECT CountryCode, Language
FROM CountryLanguage;
```

- Joined tables:

```
mysql> SELECT Code, Name, Language ← CountryLanguage
-> FROM Country, CountryLanguage
-> WHERE Continent='Africa'
-> AND Code = CountryCode;
```

Code	Name	Language
AGO	Angola	Ambo
...		
BDI	Burundi	French
...		
ZWE	Zimbabwe	Shona

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can join two or more tables by listing them in the FROM clause of the SELECT statement. Separate multiple table names by commas. Use the WHERE clause to indicate the relationship between the tables.

Imagine that you want to list all the languages spoken in every African country by using the world\_innodb database. You cannot get this information from just one table. You have to retrieve the country names from the Country table and the languages from the CountryLanguage table.

The join creates a new “virtual” table, which exists only for the duration of the statement. This table contains the Code and country Name from the Country table, and the corresponding Language from the CountryLanguage table.

This type of join can use any of the constructs allowed in a single-table SELECT statement.

Without a join, you have to execute two separate queries in which you use the code in each as a reference, and manually match up the country names to the languages (using the common country codes) as shown:

```
mysql> SELECT Code, Name
-> FROM Country
-> WHERE Continent = 'Africa';
```

Code	Name
AGO	Angola
BDI	Burundi
BEN	Benin
BFA	Burkina Faso

...

```
mysql> SELECT CountryCode, Language
-> FROM CountryLanguage;
```

CountryCode	Language
ABW	Dutch
ABW	English
ABW	Papiamentu
ABW	Spanish
AFG	Balochi

...

Using a join is a more efficient approach.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# INNER JOIN Keyword

- Alternative to listing multiple tables in the FROM clause
- Use with the ON or USING clause
- Examples:

```
mysql> SELECT Country.Name, City.CountryCode
  -> FROM Country INNER JOIN City
  -> ON Country.Name = City.Name;
```

Name	CountryCode
Djibouti	DJI
Mexico	PHL
Gibraltar	GIB
Armenia	COL
Kuwait	KWT
Macao	MAC
San Marino	SMR
Singapore	SGP

8 rows in set (0.19 sec)

**OR**

```
SELECT Country.Name,
City.CountryCode
FROM Country INNER JOIN City
  USING (Name);
```

Annotations: "table" points to "City" in the FROM clause; "column" points to "City.CountryCode" in the SELECT clause.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use the `INNER JOIN` keyword instead of listing join tables in the `FROM` clause. You then specify how rows are matched between the tables in the `FROM` clause, not in the `WHERE` clause. The `ON` or `USING` clause conditions instruct MySQL how to perform the join.

Recall that the purpose of the previous example was to join the `City` and `Country` tables on matching city names. The first example in the slide uses `INNER JOIN` and `FROM . . . ON` to get the same result. If the name of the joined column is the same in both tables, you can use `USING ()` instead of `ON` as per the second example in the slide, and list the names of the join columns within its parentheses.

For example, in the join query in the slide, you match the two tables by columns with the same name (`Name`). Therefore, you can rewrite this statement with `USING ()`.

**Note:** `ON` and `USING` are not functionally identical. `USING` treats the columns from the two tables as the same. `ON` treats them as two different columns.

# JOIN Keyword

- JOIN is equivalent to INNER JOIN
- Use with the ON and WHERE clauses
- Example:

```
mysql> SELECT COUNT(City.Name)
      -> FROM City
      -> JOIN Country
      -> ON CountryCode = Code
      -> WHERE Continent = 'South America';
+-----+
| COUNT(City.Name) |
+-----+
|                470 |
+-----+
1 row in set (0.05 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The JOIN keyword is equivalent to the INNER JOIN keyword and you can use either to achieve the same result.

The query in the example counts the cities in South America. The City table does not contain continent data. This information is in the Country table, so you need to join the City and Country tables.

The expression following the ON clause must be a condition. In this example, the ON clause states that a City row matches a Country row only when the CountryCode column value in the City table equals the value of the Code column in the Country table.

You can filter the resultset further with the WHERE clause. In this example, the join applies only when the continent is South America.

Use the ON clause for conditions that specify how to join tables, and use the WHERE clause to restrict which rows you want in the result set.

## Outer Joins

Identify combinations of matching and mismatching rows from two tables.

**LEFT JOIN:** Returns all rows from the left table, even if there are no matches in the right table

**RIGHT JOIN:** Returns all rows from the right table, even if there are no matches in the left table

```
mysql> SELECT column_name(s)
-> FROM left_table
-> LEFT JOIN right_table
-> ON left_table.column_name =
   right_table.column_name;
```

```
mysql> SELECT column_name(s)
-> FROM left_table
-> RIGHT JOIN right_table
-> ON left_table.column_name =
   right_table.column_name;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

There are two types of outer join: LEFT JOIN and RIGHT JOIN. They answer the same kinds of questions, but differ slightly in syntax. A LEFT JOIN can always be rewritten as an equivalent RIGHT JOIN.

For example, an inner join can match country names in the `Country` table with the languages spoken in those countries through a join with the `CountryLanguage` table, based on country codes. But it cannot tell you which countries do not have an associated language in the `CountryLanguage` table.

To answer this question, you need to identify which country codes in the `Country` table are not present in the `CountryLanguage` table. An outer join gives you this information. In the following example, the LEFT JOIN shows nulls for countries that do not have an associated language:

```
mysql> SELECT Name, Language
-> FROM Country
-> LEFT JOIN CountryLanguage
-> ON Code = CountryCode;
```

```
+-----+-----+
| Name                | Language          |
+-----+-----+
| Aruba               | Dutch             |
| Aruba               | English           |
| Aruba               | Papiamentu       |
| Aruba               | Spanish           |
| Afghanistan        | Balochi           |
| ...
| Antarctica          | NULL            |
| French Southern territories | NULL            |
| Antigua and Barbuda | Creole English    |
| Antigua and Barbuda | English           |
| Australia           | Arabic            |
| Australia           | Canton Chinese    |
| Australia           | English           |
| Australia           | German            |
| ...
990 rows in set (0.00 sec)
```

**Note:** An outer join is not the same as a cross join, because in an outer join only the rows in the referenced table (left or right) appear in the results, with any matched entries from the other table. A cross join includes every row from both tables.

## Finding Mismatches with LEFT JOIN

- LEFT JOIN uses the WHERE clause to find mismatches.
- Example:

```
mysql> SELECT Name, Language
-> FROM Country
-> LEFT JOIN CountryLanguage
-> ON Code = CountryCode
-> WHERE CountryCode IS NULL;
```

Name	Language
Antarctica	NULL
French Southern territories	NULL
Bouvet Island	NULL
Heard Island and McDonald Islands	NULL
British Indian Ocean Territory	NULL
South Georgia and the South Sandwich Islands	NULL

6 rows in set (0.00 sec)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

With a LEFT JOIN, the comparison that takes place between the join tables is based on the first (or left-most) table referenced in the SELECT statement.

If you are interested in only mismatches, use an appropriate WHERE clause to check which rows in the left table do not have a matching row in the right table.

In the slide example, countries with no entry in the CountryLanguage table have a null value for CountryCode.



## Finding Mismatches with RIGHT JOIN

- **RIGHT JOIN** uses the **WHERE** clause to find mismatches.
- Roles of tables are reversed from **LEFT JOIN**.
- Example:

```
mysql> SELECT Name, Language
-> FROM Country
-> RIGHT JOIN CountryLanguage
-> ON Code = CountryCode
-> WHERE CountryCode IS NULL;
Empty set (0.00 sec)
```

- The join is based on the `CountryCode` table.
- Every row in `CountryCode` has a matching row in the `Country` table, so the resultset is empty.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a **RIGHT JOIN** the roles of the tables are reversed. A **RIGHT JOIN** produces a result for each row in the right table, irrespective of whether it has any matches in the left table.

The previous example demonstrated a **LEFT JOIN** between the `Country` and `CountryLanguage` tables. This example uses a **RIGHT JOIN** to join the same two tables. Because all rows in the `CountryLanguage` table have matching rows in the `Country` table, the query produces an empty resultset.

## Outer Joins: USING and NULL

- **USING:** Easier way to reference columns that have the same name in both tables. Use instead of **ON**.
  - For example, when tables `a` and `b` both contain columns `c1`, `c2`, and `c3`, the following join compares the corresponding columns from each table:

```
... a LEFT JOIN b USING (c1,c2,c3) ...
```

- **NULL:** Identifies rows with no counterpart in the other table
  - If there are no matching rows, the join condition returns null.
  - Example:

```
SELECT table1.id * FROM table1
LEFT JOIN table2
ON table1.id=table2.id
WHERE table2.id IS NULL
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

`USING ()` is a succinct way of expressing the join condition when column names are the same in both tables. The equivalent statement using `ON` is:

```
mysql> SELECT table1.id * FROM table1
-> LEFT JOIN table2
-> ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
-> WHERE table2.id IS NULL;
```

If there is no matching row for the right table in the `ON` or `USING` part of a `LEFT JOIN`, the right table returns a row with all columns set to `NULL`.

The example in the slide lists all rows in `table1` with an `id` value that is not present in `table2` (that is, all rows in `table1` with no corresponding row in `table2`). This assumes that the `table2.id` column is declared `NOT NULL`.

For more information about join syntax, see the MySQL Reference Manual:  
<http://dev.mysql.com/doc/refman/5.6/en/join.html>.

# Table Name Aliases

- Table references can be aliased:
  - `tbl_name AS alias_name`
  - `tbl_name alias_name`
    - The `AS` keyword aids clarity but is optional.
- Examples:

```
mysql> SELECT t1.Name, t2.CountryCode
-> FROM Country AS t1, City AS t2
-> WHERE t1.Name = t2.Name;

OR

mysql> SELECT t1.Name, t2.CountryCode
-> FROM Country t1, City t2
-> WHERE t1.Name = t2.Name;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide joins the `Country` and `City` tables on city names. The `Country` table is given an alias of `t1`, and the `City` table is given an alias of `t2` although, typically, aliasing is used to abbreviate table names rather than giving them meaningless names. The query results in a list of country names and codes:

Name	CountryCode
Djibouti	DJI
Mexico	PHL
Gibraltar	GIB
Armenia	COL
Kuwait	KWT
Macao	MAC
San Marino	SMR
Singapore	SGP

8 rows in set (0.47 sec)

## Quiz

Inner joins identify combinations of matching rows from two tables, as well as the instances where a row in one table has no match in another table.

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: b**

# Summary

In this lesson, you learned how to:

- Explain the concept of a join
- Use the `JOIN` keyword to query multiple tables
- Execute outer and inner joins

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 12-1 Overview: Performing Inner and Outer Joins

In this practice you execute:

- Inner joins by using the **INNER JOIN** keyword
- Outer joins by using the **LEFT JOIN** and **RIGHT JOIN** keywords

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 12-2 Overview: Creating Queries Requiring Joins

In this practice, you create join queries to answer specific questions.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 12-3 Overview: Additional Optional Practice

In this optional practice, you are asked more in-depth questions about your answers to Practice 12-2.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# 13

## Table Subqueries

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Recognize the different types of subqueries
- Use subqueries in a variety of SQL statements
- Convert joins to subqueries

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Subquery: Overview

A subquery is a query within a query. It is:

- A powerful tool that can be used in many SQL statements
- An alternative to a join
- Also known as an “inner query”
- Executed as part of the main or “outer query”
- Always enclosed in parentheses ( )
- Allowed in only certain parts of a statement depending on the data it returns
- Discarded after execution (similar to a temporary table)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use subqueries within a variety of SQL statements. This course focuses on subqueries in `SELECT` statements.

# Basic Subquery: Example

```
mysql> SELECT Language
-> FROM CountryLanguage
-> WHERE CountryCode = (SELECT Code
-> FROM Country
-> WHERE Name='Finland');
```

Language
Estonian
Finnish
Russian
Saame
Swedish

← Main/outer query

Subquery/  
Inner query

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide uses a subquery instead of a join to find out which languages are spoken in Finland. The subquery, or inner query (in parentheses) searches the `Country` table and returns a result that is used in the `WHERE` condition of the main, or outer, query.

## Advantages of Using a Subquery

- Subqueries are useful in cases when a join does not work or is not optimal.
  - Are often easier to express than a complex join
- Subqueries generally make SQL statements easier to read.
  - Puts the “S” in SQL

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You use subqueries in cases where a join cannot solve a problem, or instead of a join to make a SQL statement easier to understand. Subqueries make structured query language statements more structured.

## Placement of Subqueries

You can use subqueries in different parts of a `SELECT` statement:

- Column designation/list

– Example:

```
mysql> SELECT 1 AS a, (SELECT 1+1) AS b;
+---+---+
| a | b |
+---+---+
| 1 | 2 |
+---+---+
1 row in set (0.11 sec)
```

- **FROM** clause
- **WHERE** clause

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Subquery Categories

- Non-correlated:
  - Does not reference outer query
  - Does not depend on outer query
  - Can stand alone (is a valid query in its own right)
- Correlated:
  - References columns in the outer query
  - Depends on the outer query
  - Cannot stand alone

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Non-Correlated Subquery: Example

```
mysql> SELECT Name FROM City WHERE CountryCode IN  
      ->      (SELECT Code FROM Country  
      ->      WHERE Continent = 'Oceania');
```

```
+-----+  
| Name  |  
+-----+  
| Tafuna|  
| Fagatogo|  
| Sydney|  
| ...   |
```

**AND THE STAND-ALONE SUBQUERY...**

```
mysql> SELECT Code FROM Country  
      -> WHERE Continent = 'Oceania';
```

```
+-----+  
| Code  |  
+-----+  
| ASM   |  
| AUS   |  
| ...   |
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The first example in the slide uses a non-correlated subquery to list the names of all the cities in the continent of Oceania, using the `City` and `Country` tables.

First, the inner query (subquery) returns the country code for each country in Oceania. Then, the outer query matches these codes to records in the `City` table.

The subquery does not reference the `City` table in the outer query and can be run stand-alone, as shown in the second example. Therefore it is a non-correlated subquery.



## Correlated Subquery: Example

```
mysql> SELECT Country.Name,
->         (SELECT COUNT(*) FROM City
->         WHERE CountryCode = Country.Code)
-> AS CityCount FROM Country;
```

Name	CityCount
Aruba	1
Afghanistan	4
Angola	5
Anguilla	2
...	

↑  
Values from outer table

- The **WHERE** expression makes the subquery correlated; the subquery expression relies on values from the outer query.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The query example in the slide uses a correlated subquery to count the number of cities in the `City` table that have a matching country code in the `Country` table. The outer query selects the country names and their respective city counts. The subquery reference to the `Country` table in the outer query is what makes it a correlated subquery. Because of this reference, you cannot execute the subquery as a stand-alone query. If you try, you get an error:

```
mysql> SELECT COUNT(*) FROM City
-> WHERE CountryCode = Country.Code;
ERROR 1054 (42S22): Unknown column 'Country.Code' in 'where clause'
```

## Subquery Result Table Types

Subqueries can return different amounts and types of table data.

- **Scalar:** Zero or one row, with a single column:

--

- **Row:** Zero or one row, with one or more columns:

--	--	--

- **Column:** Zero or more rows, with one column:


- **Table:** Zero or more rows, with multiple columns:


ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Subqueries return table data. The size and type of the table data a subquery returns determines which operators the outer query can use to interact with it. These tables are discarded after the outer statement has been executed.

## Subquery Type/Placement Chart

Subquery result types and where in a **SELECT** statement you can use them:

Placement	Scalar	Row	Column	Table
Column/Select list	X			
<b>FROM</b>	X	X	X	X
<b>WHERE: =</b>	X	X		
<b>WHERE: &lt;, &gt;</b>	X	X		
<b>WHERE: IN, ALL, ANY, SOME</b>	X	X	X	X
<b>WHERE: EXISTS</b>	X	X	X	X

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The chart shows the different types of subquery results and the parts of a **SELECT** statement that can accept these results.

For example, the column list in a **SELECT** statement can use only a scalar result. However, the **FROM** clause can use all subquery result types.

# Scalar Subqueries

- Use anywhere a scalar value is allowed, such as:
  - Function parameters
  - Mathematical operators
- Example:

```
mysql> SELECT Name FROM City
      -> WHERE Population >
      -> (SELECT Population FROM City
      -> WHERE Name='Tokyo');
+-----+
| Name          |
+-----+
| São Paulo     |
| Jakarta       |
| Mumbai (Bombay) |
| Shanghai      |
| ...           |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the slide example, the subquery retrieves the population of Tokyo and the outer query uses this value in its WHERE condition to identify larger cities.

If you execute the subquery as a stand-alone statement, it returns a scalar value: a single row in a single column:

```
mysql> SELECT Population FROM City WHERE Name='Tokyo';
+-----+
| Population |
+-----+
| 7980230    |
+-----+
1 row in set (0.00 sec)
```

## Column Designation Subquery

- Runs once for each row of the subquery result
- Correlated example:

```
mysql> SELECT Name,
      -> (SELECT MAX(Population) FROM City
      -> WHERE City.CountryCode = Country.Code)
      -> AS LargestCity
      -> FROM Country;
```

Name	LargestCity
Aruba	29034
Afghanistan	1780000
Angola	2022000
Anguilla	961
Albania	270000
Andorra	21189
Netherlands Antilles	2345
...	

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The subquery is evaluated once for each row produced by the outer query. Each result is a scalar value.

The slide example is a correlated subquery because it cannot execute without the `Country` table, referenced in the outer query.

**Note:** When you use a subquery in the column designation, you need to create a table alias for the result. The example uses an `AS` clause to achieve this.

## FROM Subquery

- A **FROM** subquery creates a derived table:
  - Special type of subquery that is used only in the **FROM** clause
- You must give the subquery a table alias.
- Example:

```
mysql> SELECT AVG(cont_sum)
      -> FROM (SELECT Continent, SUM(Population)
      ->         AS cont_sum FROM Country
      ->         GROUP BY Continent)
      -> AS t;
```

```
+-----+
| AVG(cont_sum) |
+-----+
| 868392778.5714 |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A derived table is a special type of subquery that appears only in the **FROM** clause. Derived tables are sometimes called “virtual tables” or “inline views”.

Every table that appears in a **FROM** clause must have a name, so you must provide a table alias for the subquery results.

The slide example lists the average population of each continent, based on the total population of the countries within it. The subquery calculates a total for each continent and the outer query averages these totals.

**Note:** Subqueries in **FROM** clauses must be non-correlated. They cannot reference the outer query.

You can discover how MySQL optimizes subqueries in the **FROM** clause in the MySQL Reference Manual: <http://dev.mysql.com/doc/refman/5.6/en/from-clause-subquery-optimization.html>.

## WHERE Subquery

Most common location for subqueries:

- The **WHERE** clause can accommodate all subquery result types.

Placement	Scalar	Row	Column	Table
Column/Select list	X			
FROM	X	X	X	X
WHERE: =	X	X		
WHERE: <, >	X	X		
WHERE: IN, ALL, ANY, SOME	X	X	X	X
WHERE: EXISTS	X	X	X	X

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Most subqueries appear as part of a **WHERE** condition, which can accept most types of subquery results.

## WHERE Subquery with Operators

- Operators: =, <, >
- Compares outer query with subquery results
- Example:

```
mysql> SELECT Continent, Name, Population
-> FROM Country c
-> WHERE Population = (SELECT MAX(Population)
-> FROM Country c2
-> WHERE c.Continent = c2.Continent
-> AND Population > 0);
```

Continent	Name	Population
Oceania	Australia	18886000
South America	Brazil	170115000
Asia	China	1277558000
Africa	Nigeria	111506000
Europe	Russian Federation	146934000
North America	United States	278357000

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use the =, <, or > operators to compare one value from the outer query, with the result of the subquery.

The statement in the slide example displays the country with the largest population in each populated continent. The subquery returns the largest country population in the continent referred to in the outer query. The outer query looks up this value with the equality operator and retrieves the country information.

This is a correlated query, because the subquery references the `Country` table in the outer query. It also holds its own reference to the `Country` table, so the statement uses the aliases `c` and `c2` to distinguish between them.



# Subqueries in Comparisons

The **WHERE** clause commonly uses these constructs:

- **IN** and **NOT IN**
- **ANY**
- **ALL**
- **SOME**
- **EXISTS** and **NOT EXISTS**

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The following slides cover these constructs in detail.

## WHERE with IN or NOT IN

- **IN** is functionally equivalent to = **ANY**.
- It answers the question, “Does this value appear in the subquery results?”
- Example of **IN**:

```
mysql> SELECT Name, Population FROM City
-> WHERE CountryCode
-> IN(SELECT Code FROM Country
->     WHERE Continent = 'Europe')
-> ORDER BY Name;
```

Name	Population
A Coruña (La Coruña)	243402
Aachen	243825
Aalborg	161161
...	

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**IN** is functionally equivalent to = **ANY** in a subquery. Note that the equals “=” sign is part of the equivalence.

Many consider **IN** to be more readable than = **ANY**, because what you really want to know from the outer query is whether the value appears in the subquery results. If a row in the outer query matches any row in the subquery results, the condition is true.

**NOT IN** is the opposite of **IN**. It is true if the outer query does not match any row in the subquery results.

The example in the slide retrieves the populations of European cities. The subquery returns the country codes for European countries and the outer query uses the **IN** keyword to retrieve population details of cities with these country codes.

## WHERE with ALL, ANY, and SOME

- Allow quantified comparison with multiple rows:
  - Required for a comparison between a scalar value and a column subquery
- Example of ALL:

```
mysql> SELECT Name FROM City
-> WHERE Population >
-> ALL(SELECT Population FROM City
-> WHERE CountryCode = 'CHN');
+-----+
| Name          |
+-----+
| São Paulo     |
| Mumbai (Bombay) |
| Seoul         |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A column subquery returns several rows of data in a single column. If you want to compare a scalar value with the results of a column subquery, you must use a quantified comparison condition. The quantifier keywords ALL, ANY, and SOME compare scalar values with multiple rows. ALL limits the resultset to those rows where the comparison is true for every value returned by the subquery.

The slide example uses ALL to list cities with a population larger than all the cities in China.

**Note:** If you use the ANY construct in the example query instead of ALL, it lists all cities that are larger than any Chinese city, including the one with the smallest population.

The word SOME is an alias for ANY. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
```

```
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

## ANY: Example

- Allows *any* qualifying subquery values
- Example:

```
mysql> SELECT Name
-> FROM Country
-> WHERE Continent = 'Europe'
-> AND Code = ANY (SELECT CountryCode
->                  FROM CountryLanguage
->                  WHERE Language = 'Spanish')
-> ORDER BY Name;
+-----+
| Name   |
+-----+
| Andorra|
| France |
| Spain  |
| Sweden |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The ANY comparison is equivalent to putting an “or” between each of the values in the subquery results.

The example in the slide retrieves all the countries in Europe and, for each one, tests if the inhabitants speak Spanish. You can rewrite this query using the IN keyword:

```
mysql> SELECT Name FROM Country
-> WHERE Continent = 'Europe'
-> AND Code IN(SELECT CountryCode
->             FROM CountryLanguage
->             WHERE Language = 'Spanish') ORDER BY Name;
+-----+
| Name   |
+-----+
| Andorra|
| France |
| Spain  |
| Sweden |
+-----+
```

## ANY: Example Subquery

Run the subquery stand-alone to confirm the results:

```
mysql> SELECT CountryCode
      -> FROM CountryLanguage
      -> WHERE Language = 'Spanish';
+-----+
| CountryCode |
+-----+
| ABW         |
| AND         |
| ...         |
| USA         |
| VEN         |
| VIR         |
+-----+
28 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# WHERE with EXISTS and NOT EXISTS

- Tests if a subquery returns any rows
- Example of **EXISTS**:

```
mysql> SELECT Continent FROM Country
-> WHERE EXISTS (SELECT * FROM City
-> WHERE Code = CountryCode
-> AND Population > 8000000)
-> GROUP BY Continent;
```

Continent
Asia
Europe
North America
South America

4 rows in set (1.34 sec)

**NOT EXISTS**  
*result* →

Continent
Asia
Europe
North America
Africa
Oceania
Antarctica
South America

7 rows in set (0.05 sec)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The **EXISTS** and **NOT EXISTS** operators test if a subquery returns any rows. If it does return rows, **EXISTS** is true and **NOT EXISTS** is false.

The first example in the slide uses a subquery to retrieve cities with a population greater than eight million and lists the continents those cities are in.

If, as in the second example, you change the operator to **NOT EXISTS**, you get all continents that contain a country that has no city with more than eight million people.

## Finding Table Mismatches

- Use **NOT IN** to find mismatches:

```
SELECT Name FROM Country
WHERE Code NOT IN
      (SELECT CountryCode FROM CountryLanguage);
```

- Use **LEFT JOIN** to find mismatches:

```
SELECT Name FROM Country
LEFT JOIN CountryLanguage
ON Code = CountryCode
WHERE CountryCode IS NULL;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The first example uses **NOT IN** to identify countries that do not have any language entries in the `CountryLanguage` table.

The second example uses a **LEFT JOIN** to achieve the same result.

## Modifying Table Data With Subqueries

- Subqueries are not limited to **SELECT** statements.
- You can use subqueries in **DELETES** and **UPDATES**.
- **DELETE** example:

```
DELETE FROM NACities
WHERE CountryCode IN
    (SELECT Code FROM Country
     WHERE LifeExpectancy < 70.0);
```

- **UPDATE** example:

```
UPDATE table1 SET table1field =
    (SELECT MAX(table2.table2field)
     FROM table2
     WHERE table1.table1field = table2.table2field);
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Common scenarios for using subqueries to modify table data include:

- Deleting rows from one table if they match (or do not match) rows in another table
- Updating rows in one table by using row values from another table

The **DELETE** example in the slide removes city records from a table containing North American cities if those cities are in countries where the life expectancy is less than 70 years.

The **UPDATE** example in the slide performs a correlated subquery based on an aggregate function applied to another table. Another example is creating a temporary table to store the date of a specific customer's last order, based on a table containing orders for all customers.



## Quiz

When `IN` is used with a subquery, it is functionally equivalent to:

- a. `ALL`
- b. `= ALL`
- c. `NOT IN`
- d. `= ANY`

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: d**

## Subquery Tips

- You might need to provide a table alias for the results.
- You can use subqueries in:
  - **SELECT, UPDATE, DELETE, and INSERT** statements
  - **FROM, WHERE, HAVING, and ORDER BY** clauses
  - Conditions that use comparison and special-purpose operators: **IN, NOT IN, ANY, ALL, SOME, EXISTS, and NOT EXISTS**
- You can use arithmetic operators on either side of a subquery: **=, <>, <, >, <=, >=**

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Converting Joins to Subqueries

- Subqueries are easier to read and write than table joins.
- Example of join versus subquery:

```
mysql> SELECT DISTINCT Language
-> FROM Country
-> JOIN CountryLanguage
-> ON CountryCode = Code
-> WHERE Continent='Africa';
```

Language
Ambo
Chokwe
Kongo
Luchazi
Luimbe-nganguela
...

OR

```
SELECT DISTINCT Language
FROM CountryLanguage
WHERE CountryCode IN
  (SELECT Code FROM Country
   WHERE Continent='Africa')
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Both examples in the slide answer the question “which languages are spoken in Africa?”

# Summary

In this lesson, you learned how to:

- Recognize the different types of subqueries
- Use subqueries in a variety of SQL statements
- Convert joins to subqueries

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 13-1 Overview: Performing Different Types of Subqueries

In this practice, you use subqueries in the **FROM** and **WHERE** clauses of a **SELECT** statement.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 13-2 Overview: Performing Several Advanced Subqueries

In this practice, you create more advanced subqueries to answer specific questions.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# 14

## MySQL Graphical User Interface Tools

### Workbench and Enterprise Monitor

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Explain the primary features of the MySQL Workbench GUI tool
- Use the MySQL Workbench tool
- Explain the primary features of the MySQL Enterprise Monitor GUI tool

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# MySQL GUIs

- MySQL provides several graphical user interface tools.
- MySQL Workbench and MySQL Enterprise Monitor can help all MySQL users.
- This lesson provides only an overview of these tools.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For more information about MySQL Workbench, MySQL Enterprise Monitor, and other GUI tools, see the MySQL website, as well as other MySQL courses.

# MySQL Workbench

A unified GUI tool that helps database architects, developers, and DBAs manage the MySQL server and data.

- Use MySQL Workbench for:
  - SQL development
  - Data modeling
  - Server administration
  - Database migration
- Two versions of Workbench:
  - Community Edition (OSS)
  - Standard Edition (SE):
    - Schema/model validation and documentation tools
  - This lesson covers key features available in the Community Edition.
- Runs on Windows, Linux, Mac OS X

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL Workbench is used to write and execute SQL queries and scripts, edit data, design database models visually, manage the MySQL server, and help with data migration. Workbench runs on Windows, Linux, and Mac OS X operating systems, but can work with MySQL servers running on any supported operating system.

MySQL Workbench is available in two versions:

- **Community Edition (OSS):**
  - Foundation of all Workbench editions
  - Fully-featured, powerful database management tool
  - Open-source GPL (general public license), available free from the MySQL website
- **Standard Edition (SE):**
  - Commercial extension of the OSS version
  - Advanced features include MySQL-specific schema validation, model validation, general schema validation, and DBDoc documentation tools.
  - Available for purchase from the MySQL website

You can download MySQL Workbench from: <http://dev.mysql.com/downloads/workbench/>. For system requirements, refer to the MySQL Workbench Reference Manual at <http://dev.mysql.com/doc/workbench/en/index.html>.

# MySQL Workbench Functionality

- SQL Development
  - Edit and execute SQL queries and scripts.
  - Create or alter database objects.
  - Edit table data.
- Data Modeling
  - Extended entity-relationship (EER) modeling
  - Edit and execute SQL queries and scripts.
  - Design, generate, and manage databases.
- Server Administration
  - Start and stop the server.
  - Edit database server configuration.
  - Manage users.
  - Import and export data.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# MySQL Workbench Functionality

- Database Migration
  - Migrate tables and objects from other RDBMs to MySQL.
  - Convert existing applications to run on MySQL on Windows and other platforms.
  - Upgrade to later versions of MySQL.
- Access to MySQL Utilities
  - MySQL Utilities are python tools for working with MySQL Server.
  - MySQL Workbench provides a shell for working with these tools.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

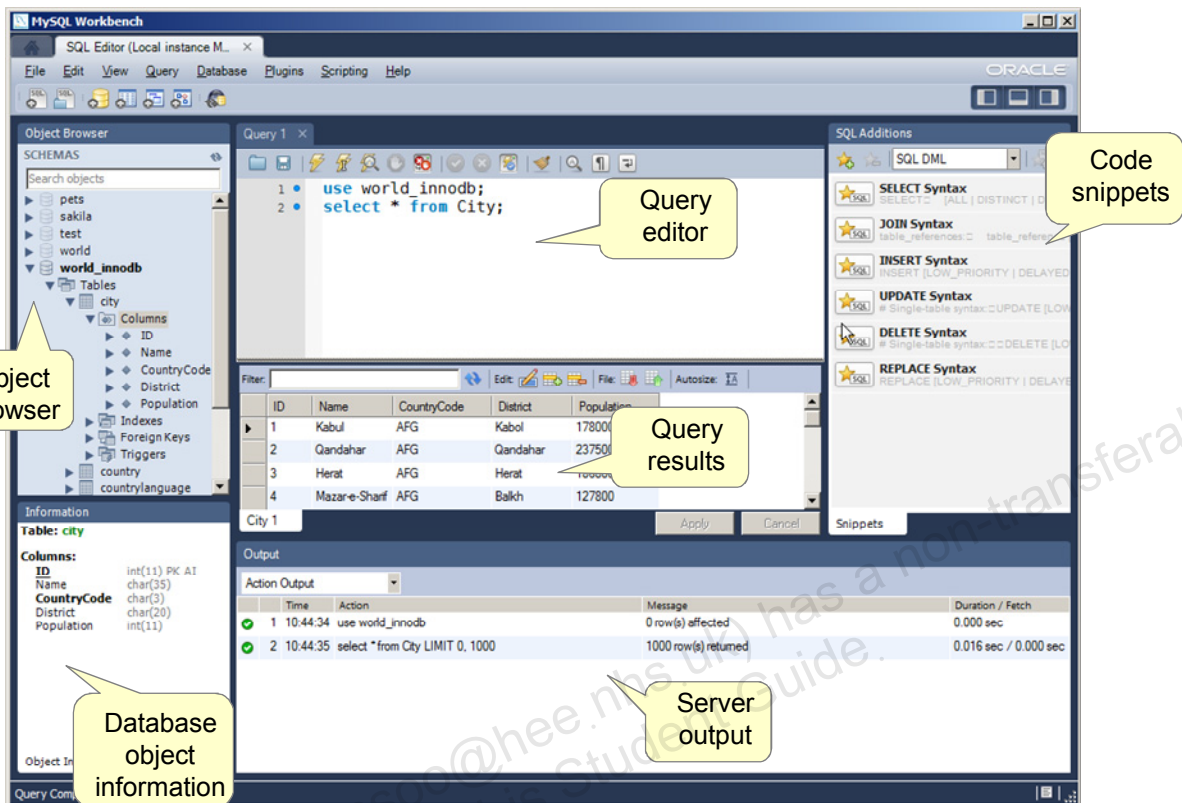
MySQL Workbench provides a complete, easy-to-use solution for migrating Microsoft SQL Server, Sybase ASE, PostgreSQL, and other RDBMS tables, objects, and data to MySQL. Developers and DBAs can quickly and easily convert existing applications to run on MySQL both on Windows and other platforms. Migration also supports migrating from earlier versions of MySQL to the latest releases.

For more information about MySQL Workbench, visit <http://mysql.com/products/workbench/>.

**Note:** Workbench can migrate views and stored procedures only for MySQL databases, not other RDBMSs. It cannot migrate objects that have no equivalent in MySQL.

For more information about MySQL Utilities, visit <http://dev.mysql.com/doc/workbench/en/mysql-utilities.html>.

# MySQL Workbench: SQL Development



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The SQL development features of MySQL Workbench include:

- **SQL Editor:** For composing and executing query statements, with syntax highlighting to make SQL easier to read. You can write multiple statements and execute them individually, together, or save them as a script. Plug-ins are available for specific tasks such as:
  - SQL code beautification and reformatting
  - Output to plain text (instead of the query results grid)
  - Copying SQL code as PHP code to the clipboard for quick re-use in PHP scripts
- **Query Results:** Appear in tabs at the bottom of the editor pane by default. You can redirect output to plain text via a plug-in. You can also edit certain resultsets directly.
- **SQL code snippets:** For faster development. You can use existing snippets in your code, or create your own.
- **Object Browser:** For getting information about tables, columns, and other entities in your database. You can live-edit database objects by using the `ALTER` syntax via a context menu.
- **Output:** Messages generated by the MySQL server, including information about query execution and fetch times, and query history. Output can be sent to plain text via a plug-in.

To develop SQL code in this environment, you need a database connection to the MySQL server, which you can create, connect to, and manage within MySQL Workbench.

# MySQL Workbench: Data Modeling

Includes tools for:

- Creating and manipulating data models graphically
  - Using extended entity-relationship (EER) diagrams to visualize data and relationships
- Forward/reverse engineering:
  - Turning a model into a live database or script
  - Turning a live database to a model
- Schema synchronization:
  - Comparing a data model against a target MySQL server, and performing a synchronization between the two
- Exporting and printing EER diagrams

ORACLE

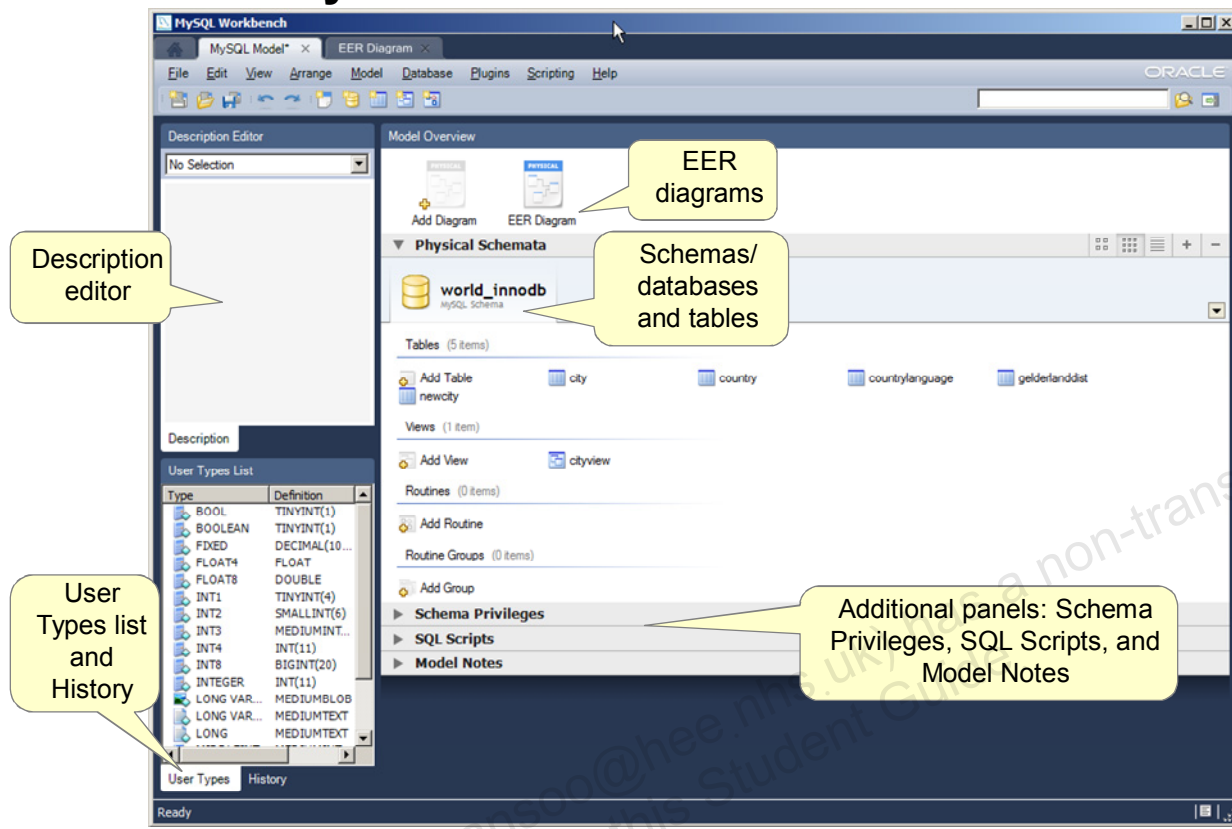
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Database modeling is acknowledged by experienced data professionals as the best way to design and implement databases. MySQL Workbench helps a data modeler create complex models and perform difficult and time-consuming change management and documentation tasks. It includes tools for creating extended entity-relationship diagrams (EERs) to visualize the structure of a database and understand the relationships between tables.

You can export EER diagrams as images (.png, .svg) or PDFs, or print them, with options to define printer page sizes and counts for large diagrams.

**Note:** Workbench does not support the full EER model, but stays close to the supported features of the MySQL server (so, for example, it does not support the inheritance modeling feature).

# MySQL Workbench: Model Editor



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You create a model for a new database from the beginning, or for an existing database by reverse-engineering it, and manage the model in the Model Editor (the MySQL Model window), shown in the slide. The Model Editor has three main panels:

- **Model Overview:** The main area for managing your modeling project and associated EER diagrams. The next slide describes this area.
- **Description Editor:** A free text area that you use to describe the entities in the database
- **User Types list and History:**
  - The User Types list shows a list of data types for column definitions. A dot symbol indicates that an item is already included in a diagram.
  - The History tab logs any activities that alter the MySQL model or change an EER diagram. The user can undo/redo actions by double-clicking entries.

There are also a number of useful plug-ins for special tasks, such as obfuscating object names, changing storage engines for all tables at once, and so on.

## MySQL Workbench: Model Overview Panel

The panels within the MySQL Model window are:

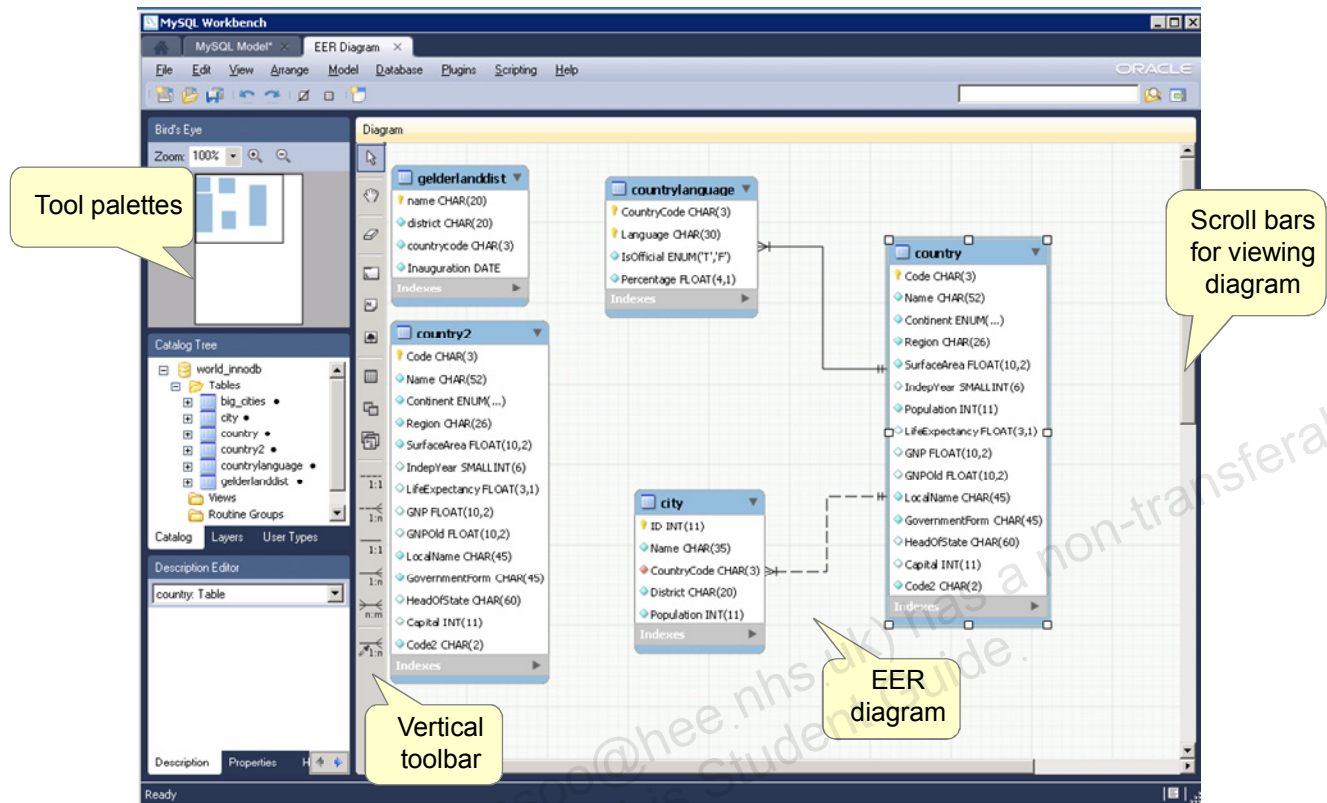
- **EER Diagram:** Each model can have multiple diagrams.
- **Physical Schemata:** Shows the active schemata (each schema in its own tab) and the objects that they contain:
  - Tables
  - Views
  - Routines
  - Routine Groups
- **Schema Privileges:** Manage users and privileges.
- **SQL Scripts:** Load and edit SQL scripts.
- **Model Notes:** Create and save project notes.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# MySQL Workbench: Data Modeling EER Diagram



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

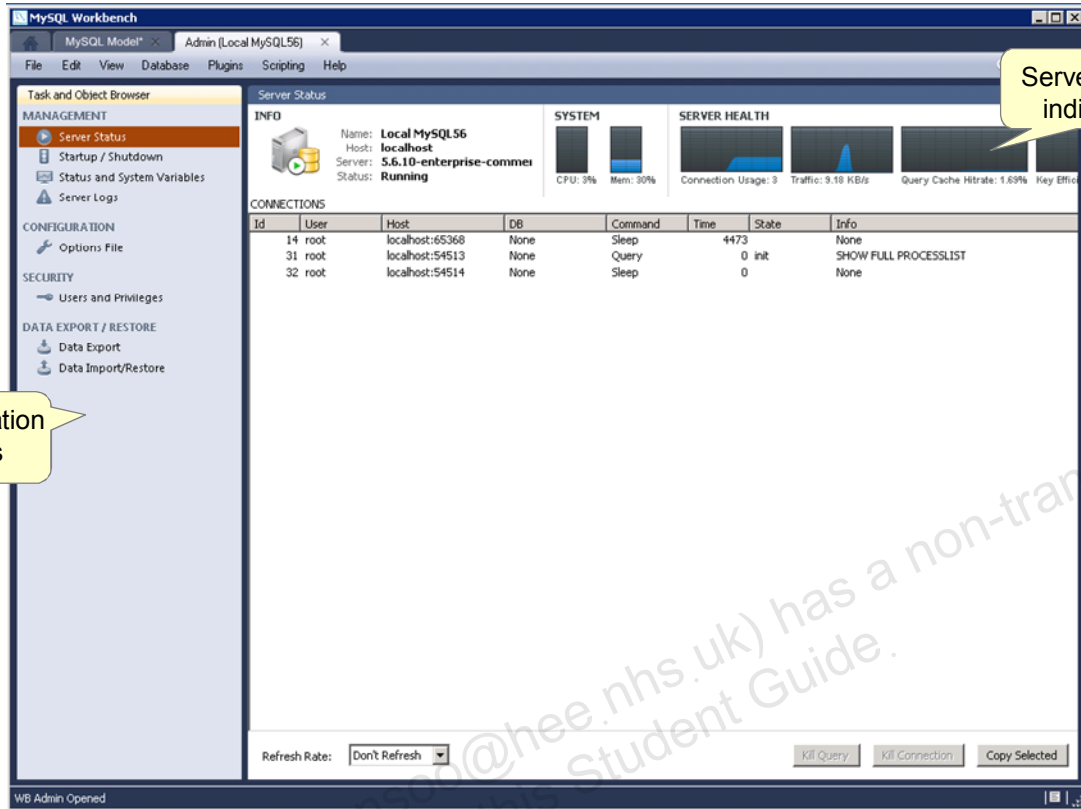
Clicking an EER diagram opens the canvas where you graphically manipulate data objects by using tool palettes and the vertical toolbar.

Tool palettes include:

- **Bird's Eye navigator palette:** Gives you an overview of the objects placed on an EER diagram canvas and makes large diagrams easy to navigate
- **Catalog Tree palette:** Shows all the schemata from the Physical Schemata panel of the MySQL Model window. It arranges the database objects in the following tree folders: Tables, Views, and Routine Groups.
- **Properties Palette:** Displays and edits the properties of objects on an EER diagram. It is especially useful for editing display objects such as layers and notes.

Additional palettes are available.

# MySQL Workbench: Server Administration



Administration options

Server status indicators

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# MySQL Workbench: Server Administration Features

- Management
  - Overall server “health”
  - Connection and query management
  - Server startup and shutdown
  - Status and system variables
  - Server logs
- Configuration
  - View/edit MySQL configuration file.
- Security
  - Manage users and privileges.
- Data Export/Restore
  - Graphical interface to `mysqldump` program

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Server Status option provides immediate view of the basic health indicators and counters in a MySQL environment:

- System
  - CPU utilization
  - Memory usage
- Server Health
  - Number of connections
  - Network traffic
  - Query cache hit rate
  - Key efficiency

# MySQL Enterprise Monitor

MySQL Enterprise Monitor (MEM) is a web-based monitoring and advising system that significantly improves productivity.

MEM:

- Runs entirely within the corporate firewall
- Recommends best practices to:
  - Eliminate security vulnerabilities
  - Improve replication
  - Optimize performance and more
- You can access MEM from the MySQL website:
  - It is available with MySQL Enterprise Edition and MySQL Cluster Carrier Grade Edition.
  - A fully-featured trial of MEM and prerecorded demonstrations are also available.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can use MEM to:

- Manage more MySQL servers in a scale-out environment
- Tune your current MySQL servers
- Find and fix problems with your MySQL database applications before they become serious problems or costly outages

Because MEM runs entirely within your corporate firewall, it proactively monitors your enterprise database environments. MEM provides expert advice on how MySQL can tighten security, optimize performance, and reduce down time. MEM accomplishes all this while reducing DBA time and effort.

For more information about MEM and product demonstrations, visit the MySQL Enterprise Monitor web page at <http://mysql.com/products/enterprise/monitor.html>.

## Enterprise Monitor Features

- **Enterprise Dashboard:** Manage all MySQL servers from a consolidated console.
- **Server/group management:** Supports auto-detection, grouping, and monitoring of replication and scale-out topologies
- **Monitoring page:** An at-a-glance global health check of key systems
- **MySQL and custom advisors and rules:** To enforce MySQL best practices
- **Advisor rule scheduler:** Specify which advisors should be active on which server or group.
- **Replication Monitor:** View real-time master/slave performance.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MEM provides a rich GUI Enterprise Dashboard with pages that give you an immediate, graphical view of administrative tasks, server and database status, and advisory information.

## Enterprise Monitor Features

- **Customizable thresholds, advisor rules, and alerts:** Help you to identify advisor rule violations and user-defined advisor rules
- **Events and alert history:** Lists all advisor rule executions
- **Query Analyzer:** Monitors query performance and accurately pinpoints SQL code that is causing a slowdown
- **Cluster Graphs and Advisor:** Automated, real-time monitoring of MySQL Cluster data nodes and best practice advice
- **MySQL and Operating System Graphs:** Visual monitoring of key system resources like database transactions and binlog cache efficiency

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# Enterprise Monitor: Dashboard

The screenshot shows the MySQL Enterprise Monitor Dashboard with several callouts highlighting key features:

- Page Tabs:** Located at the top, including Monitor, Advisors, Events, Graphs, Query Analyzer, Replication, Settings, and What's New?
- Refresh Setting/Help:** Located in the top right corner, showing a refresh interval of 'Every 15 Seconds' and a 'Help' button.
- Server Navigation:** A tree view on the left side showing the hierarchy of servers, including 'All Servers (6)', 'Cluster (2)', and 'Production 1 (2)'. The 'Production 1 (2)' folder is expanded to show 'NA\_Blade4403' and 'NY\_Blade4402'.
- Graphs:** A collection of six line graphs showing metrics over time (16:45 to 17:30):
  - Cluster Data Node Data Memory Use
  - Cluster Data Node Index Memory Use
  - Connections
  - CPU Utilization
  - Database Activity
  - Network - Database Throughput
- Heat Chart:** A grid-based chart showing the status of various components across different servers. The legend includes: Server Status, CPU Usage, RAM Usage, MySQL Content Change, MySQL Query Cache, Temp Tables to Disk, Table Locks, Critical Alerts, and Warning Alerts.
- Support Status:** A section on the right titled 'Waiting on Customer Support Issues' with a summary table:
 

Summary	Sev	Last Response
51831 - memory error piping mysqlbinlog to mysql	S3	Nov 9, 2010 11:00:41 AM
- Critical Events:** A table at the bottom showing production critical events:
 

Current	Server	Category	Rule	Time	Action
●	NA_Blade4403	Performance	Excessive Disk Temporary Table Usage Detected	Nov 15, 2010 3:00:06 AM	close
●	NA_Blade4403	Memory Usage	Key Buffer Size May Not Be Optimal For System RAM	Oct 25, 2010 4:48:08 AM	close
●	NA_Blade4403	Replication	Slave Has Been Stopped	Sep 13, 2010 11:44:06 AM	close
●	NA_Blade4403	Replication	Slave I/O Thread Not Running	Sep 13, 2010 11:44:06 AM	close

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Enterprise Monitor: Advisors

MEM advisors provide monitoring and advice on:

- **Upgrades:** Ensuring that you have deployed the most secure and up-to-date version of MySQL
- **Administration:** General database administration, recoverability, and performance configuration settings
- **MySQL Enterprise Backup:** MySQL Enterprise Backup operations, status, and MySQL servers that need backup
- **Security:** Tightening security vulnerabilities in the MySQL server
- **Replication:** Addressing problems relating to replication setup, security, and master/slave latency

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Having MySQL Enterprise Monitor and its advisors constantly checking the configuration and workload of monitored MySQL servers is like having a virtual MySQL DBA always on guard.



## Enterprise Monitor: Advisors

- **Memory Usage:** Memory-related server metrics (cache usage, hit ratios, and so on) and suggested configuration changes to improve performance
- **Performance:** Performance-related server metrics and suggested configuration and variable settings
- **Schema:** Unplanned changes to database schemas
- **MySQL Cluster:** Health, performance, and availability of MySQL Cluster data nodes
- **Custom:** Create your own best practice advisors and rules to fit your specific use of MySQL.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Quiz

You can automatically create an EER diagram from an existing MySQL database by using the MySQL Workbench tool.

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

# Summary

In this lesson, you learned how to:

- Explain the primary features of the MySQL Workbench GUI tool
- Use the MySQL Workbench tool
- Explain the primary features of the MySQL Enterprise Monitor GUI tool

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

## Practice 14-1 Overview: Creating a Data Model by Using MySQL Workbench

In this practice, you use MySQL Workbench data modeling functionality to create a data model for the `world_innodb` database.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 14-2 Overview: Creating a Server Instance by Using MySQL Workbench

In this practice, you use the MySQL Workbench server administration functionality to create a server instance with the `world_innodb` database.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 14-3 Overview: Viewing MySQL Enterprise Monitor Demonstrations

In this practice, you view web-based demonstrations that highlight the primary features of Enterprise Monitor.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 14-4 Overview: (Optional) Viewing the MySQL Workbench Demonstration

In this practice, you view a web-based demonstration that highlights the primary features of Workbench.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 14-5 Overview: (Optional) Creating a Model for the `Pets` Database by Using MySQL Workbench

In this practice, you:

- Create a backup of the `Pets` database
- Create a model for the `Pets` database in MySQL Workbench, including:
  - Creating an EER diagram
  - Defining table relationships

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



# 15

## Supplementary Information

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Objectives

After completing this lesson, you will be able to:

- Explain the general purpose of storage engines in MySQL
- Determine which storage engine a specific table uses
- Describe the key features of the InnoDB storage engine
- Create table views with specific data
- Explain and use transactional processing
- Define ACID compliance for transaction safety
- Describe the `INFORMATION_SCHEMA` database, and use it to obtain metadata
- Describe the `PERFORMANCE_SCHEMA` database
- Explain the key features of MySQL Enterprise Backup

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Storage Engines and MySQL

- InnoDB is the default storage engine for MySQL Server.
- NDB is the storage engine for MySQL Cluster.
- There are many other storage engines.
- Each storage engine has different characteristics and implications.
- You can choose a specific storage engine when you create a table.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you create tables with MySQL, you can specify the storage engine to be used for each table. Typically, you make this choice according to which storage engine offers features that best fit the needs of your application. Each storage engine has a particular set of operational characteristics. These characteristics include the types of locks it uses to manage query contention, and whether it supports transactions. Which engine you use has implications for query processing performance, concurrency, and deadlock prevention.

Although there are many other storage engines available, InnoDB is the best fit for most cases.

**Note:** MySQL Cluster is a technology providing shared-nothing clustering and auto-sharding for the MySQL database management system, which supports high read and write workloads. It uses its own dedicated storage engine NDB. For a comparison between NDB and InnoDB, see the MySQL Reference Manual: <http://dev.mysql.com/doc/refman/5.6/en/mysql-cluster-ndb-innodb-engines.html>.

# Storage Engines Available on the Server

View the available storage engines:

```
mysql> SHOW ENGINES\G
...
  Engine: InnoDB
  Support: DEFAULT
  Comment: Supports transactions, row-level locking,
           and foreign keys
  Transactions: YES
...
  Engine: MyISAM
  Support: YES
  Comment: MyISAM storage engine
  Transactions: NO
...
  Engine: MEMORY
  Support: YES
  Comment: Hash based, stored in memory, useful for
           temporary tables
  Transactions: NO
...
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To see what storage engines are compiled into your server and whether they are available at run time, use the `SHOW ENGINES` statement.

The value in the Support column is YES or NO, to indicate that the engine is or is not available, DISABLED if the engine is present but turned off, or DEFAULT for the storage engine that the server uses by default. The default engine is always available.

## Displaying Storage Engine Setting: Using SHOW CREATE TABLE

Example:

```
mysql> SHOW CREATE TABLE City\G
***** 1. row *****
      Table: City
Create Table: CREATE TABLE `city` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
  CONSTRAINT `city_ibfk_1` FOREIGN KEY
  (`CountryCode`) REFERENCES `country` (`Code`)
) ENGINE=InnoDB AUTO_INCREMENT=4080
  DEFAULT CHARSET=latin1
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Displaying Storage Engine Setting: Using SHOW TABLE STATUS LIKE

Example:

```
mysql> SHOW TABLE STATUS LIKE 'CountryLanguage'\G
***** 1. row *****
      Name: countrylanguage
      Engine: InnoDB
      Version: 10
      Row_format: Compact
      Rows: 820
      Avg_row_length: 119
      Data_length: 98304
      Max_data_length: 0
      Index_length: 65536
      Data_free: 12582912
      Auto_increment: NULL
      ...
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Setting the Storage Engine to InnoDB

- Set the server storage engine as part of the startup configuration file:

```
[mysqld]
default-storage-engine=InnoDB
```

- Set for the current client session using the **SET** command:

```
SET @@default_storage_engine=InnoDB;
```

- Specify using the **CREATE TABLE** statement:

```
CREATE TABLE t (i INT) ENGINE = InnoDB
```

- Change the table's current storage engine:

```
ALTER TABLE t ENGINE = InnoDB
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you create a table without using the **ENGINE** option to specify a storage engine explicitly, the MySQL server creates the table using the default engine. This is the value stored in the `default_storage_engine` system variable.

You can overwrite the server default for the current session by using the **SET** command, as shown in the slide.

# InnoDB Storage Engine

InnoDB is the default storage engine for MySQL. It is extremely reliable and offers the following additional advantages:

- Excellent performance with large volumes of data
- Transaction safe (ACID compliant)
- Multi-Versioning Concurrency Control (MVCC)
  - InnoDB row-level locking
  - Oracle-style consistent non-locking reads
- Tables arrange data on disk to optimize common queries
- Supports foreign key constraints for data integrity
- Can mix with tables by using different storage engines
- Fast auto-recovery after a crash
- Buffer pool for caching data and indexes in memory
- Supports on-line schema changes

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Not many disk-based relational database engines rival the efficiency of InnoDB. Here is additional information about the advantages of using InnoDB:

- **Transaction-safe:** Transaction commit, rollback, and crash-recovery capabilities are ACID compliant.
- **Foreign key support:** When a foreign key value is used, it must reference a valid, existing primary key in the parent table to ensure referential integrity. InnoDB supports `CASCADE` deletes and updates, and `RESTRICT` foreign key constraints to support this.
- **Recovery/Backup:** InnoDB supports consistent and online logical backup.
- **Mixing queries:** You can mix InnoDB tables with tables from other MySQL storage engines in queries. For example, you can use a join operation to combine data from `InnoDB` and `MEMORY` tables in a single query.

**Note:** Oracle does not recommend this practice, especially in replicated environments.

For more information about the InnoDB storage engine, see the MySQL Reference Manual: <http://dev.mysql.com/doc/refman/5.6/en/innodb-storage-engine.html>.

For a glossary of InnoDB terms, visit: <http://dev.mysql.com/doc/refman/5.6/en/glossary.html>.



## Other Storage Engines

There are many optional storage engines for specialized use cases.

- Optional engines include:
  - **MyISAM:** Primarily used in web, data warehousing, and other application environments
  - **MEMORY:** Creates tables with contents that are stored in memory
  - **ARCHIVE:** High-speed, compressed storage of seldom-read historical data, such as log files
  - **NDB:** For MySQL Cluster installations
- Include or exclude other engines with a custom build.
- Enable or disable optional engines at server startup.
- InnoDB is the best choice for most situations.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The default storage engine InnoDB is one of many storage engines MySQL supports. All other storage engines are optional, but can help in specialized situations. The availability of these optional engines depends on your build of MySQL Server. If your build of MySQL Server supports other storage engines, you can enable or disable them at run time with a server startup option. Some, like MyISAM, MERGE, and MEMORY, are included in every build. To include or exclude support for other storage engines, you need to configure and compile MySQL Server from the source code.

If you are compiling from source you can reduce memory overhead by removing support for unused engines. Or, if you are using a binary distribution that supports unwanted optional storage engines, you should disable them during server startup.

For more information about all supported storage engines, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.6/en/storage-engines.html>.

## Quiz

If you create a table without specifying the storage engine explicitly using the \_\_\_\_\_ option, the MySQL server creates the table with the default engine.

- a. INNODB
- b. ENGINE
- c. DEFAULT\_STORAGE\_ENGINE

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: b**

# Creating Views

A view is a virtual table defined by the result of a `SELECT` query.

- A custom view of your tables makes some operations simpler.
- Views provide additional benefits compared to selecting data directly from base tables.
- You can include regular tables or other views in your defining `SELECT` statement.
- Creation syntax:

```
CREATE VIEW view_name [<column_list>]  
AS SELECT_expression
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Views are also known as virtual tables. They are useful for providing custom views of tables for specific uses or users.

The `SELECT` statement that defines the view can select from regular tables or other views.

Selecting data directly from views instead of tables allows you to:

- Hide data complexity, such as complex joins
- Protect sensitive data by hiding certain columns and rows from certain users
- Customize the display of data. For example, use computed values or different column names.
- Preserve the appearance of the original table when you need to change the underlying table structure

In some cases, a view is updatable and you can use it with statements such as `UPDATE`, `DELETE`, or `INSERT` to modify the underlying tables. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. Certain constructs prevent you from updating views. For more information about these restrictions, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.6/en/view-updatability.html>

## CREATE VIEW: Examples

- Basic example:

```
CREATE VIEW Country_View
AS SELECT * FROM Country;
```

- More complex example, using computed values:

```
CREATE VIEW per_capita_v AS
SELECT Name, GNP, Population, GNP/Population
AS per_capita FROM Country;
```

- The contents of the view:

```
mysql> SELECT * from per_capita_v;
```

Name	GNP	Population	per_capita
Aruba	828.00	103000	0.008039
Afghanistan	5976.00	22720000	0.000263
Angola	6648.00	12878000	0.000516
...			

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A view belongs to a database. By default, the `CREATE VIEW` statement creates the view in the current database. To create or refer to a view in another database, either change the current database (with `USE <database>`) or create the view as `<db_name>.<view_name>`.

You create a view from a `SELECT` statement., which can refer to base tables or other views. It can use joins, unions, and subqueries. The `SELECT` does not even have to refer to any tables.

The first example in the slide creates a view containing the entire contents of the `Country` table from the `world_innodb` database.

The second example defines a view by using three columns from the `Country` table and an expression calculated from two of the columns.

## Displaying View Information

- Base table statements also work for views:
  - DESCRIBE
  - SHOW TABLES
  - SHOW TABLE STATUS
- Example using SHOW CREATE VIEW:

```
mysql> SHOW CREATE VIEW per_capita_v\G
***** 1. row *****
View: per_capita_v
Create View: CREATE ... VIEW 'per_capita_v'
AS select 'country'.' Name' AS 'Name','country'.'GNP'
AS 'GNP','country'.'Population' AS 'Population',
('country'.'GNP' / 'country'.'Population') AS 'per_capita'
  from 'country'
character_set_client: latin1
collation_connection: latin1_swedish_ci
1 row in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can also obtain information about views from the INFORMATION\_SCHEMA database's VIEWS table. Refer to the INFORMATION\_SCHEMA details later in this lesson.

## Views: Showing Table Types

- You can use `SHOW FULL TABLES` to differentiate base table and views.
- Example:

```
mysql> SHOW FULL TABLES FROM world_innodb;
+-----+-----+
| Tables_in_world_innodb | Table_type |
+-----+-----+
| city                    | BASE TABLE |
| country                 | BASE TABLE |
| country_view           | VIEW        |
| countrylanguage        | BASE TABLE |
| per_capita_v           | VIEW        |
+-----+-----+
5 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## View Definition Restrictions

- The view's **SELECT** statement cannot:
  - Contain a subquery in the **FROM** clause
  - Refer to system or user variables
  - Refer to prepared statement parameters
- Any table or view referred to in the definition must exist.
- The view definition cannot:
  - Refer to a temporary table
  - Create a temporary view

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Quiz

Views are sometimes called virtual tables. You create them with a `CREATE TABLES` statement.

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: b**



# Transactions

- A collection of statements that are treated as a single unit.
  - Groups multiple statements
  - Is useful when multiple clients can access data from the same table at the same time
- All or none of the steps succeed.
  - Execute if all steps are successful.
  - Cancel if any steps cause errors or do not complete.
- MySQL transactions are ACID compliant.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A transaction lets you execute one or more SQL statements as a single unit of work, so that either all or none of the statements succeed. This happens independently of work being done by any other transactions. If all the statements succeed, you commit the transaction to record their effect permanently in the database. If an error occurs during the transaction, you roll it back to cancel it. Any statements executed up to that point within the transaction are undone, leaving the database in the state it was in before the transaction.

**Note:** MySQL supports transactions only for tables that use a transactional storage engine (such as InnoDB). These statements have no noticeable effect on tables managed by non-transactional storage engines.

# Transactions: ACID

- **Atomic:** All statements execute successfully as a unit or are canceled as a unit.
- **Consistent:** A database that is in a valid state when a transaction begins remains in a valid state after the transaction.
- **Isolated:** One transaction does not affect another.
- **Durable:** All changes made by transactions that complete successfully are recorded properly in the database. Changes are not lost.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Transactional processing provides stronger guarantees about the outcome of database operations, but also requires more overhead in CPU cycles, memory, and disk space.

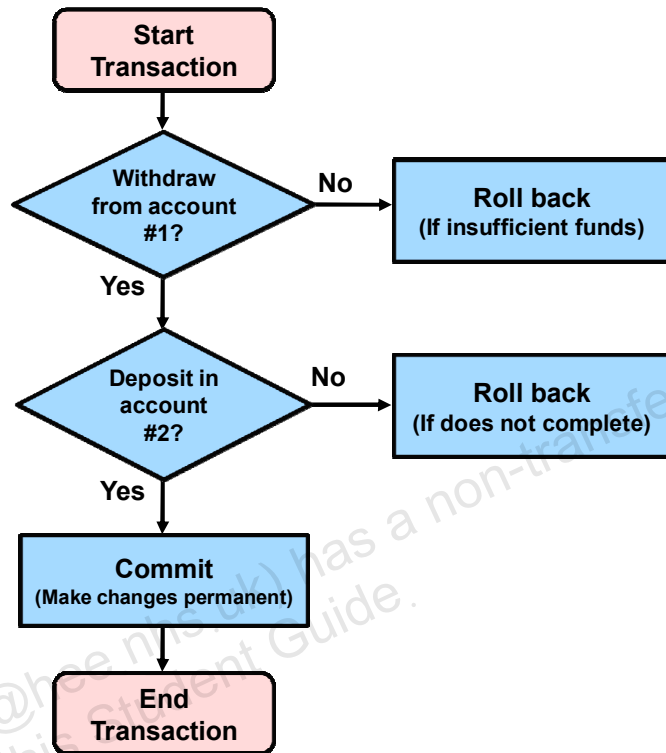
Transactional properties are essential for some applications and not for others, and you can choose which ones make the most sense for your applications.

For financial applications, the guarantee of data integrity provided by transactional processing is usually worth the additional overhead. Alternatively, an application that logs web page access can probably cope with the loss of a few records if the server crashes.

Some storage engines support transactions and others do not. Select the engine that best matches your application requirements.

# Transaction Diagram

Example of a banking transaction:



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide shows an attempted transfer of \$1,000 from your savings account to your checking account. You would not be happy if the money were successfully withdrawn from your savings account, but never reached your checking account.

To protect against this kind of error, the program that handles your transfer request begins a transaction, and then issues the SQL statements needed to move the money from your savings to your checking account. It only commits the transaction if everything succeeds. If a problem occurs, the program instructs the server to undo all the changes made since the transaction began.

# Transaction SQL Control Statements

- **START TRANSACTION** (or **BEGIN**): Explicitly begins a new transaction
- **COMMIT**: Makes the changes from the current transaction permanent
- **ROLLBACK**: Cancels the changes from the current transaction
- **SET AUTOCOMMIT**: Disables or enables the default `autocommit` mode for the current connection

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

By default, MySQL runs with `autocommit` mode enabled. This means that as soon as you execute a statement that modifies a table, MySQL stores the update on disk. In `autocommit` mode, each individual statement is considered to be a transaction, unless you issue a `START TRANSACTION` statement. After that, the transaction remains open until you close it with either a `COMMIT`, a `ROLLBACK`, or any statement that implicitly closes a transaction.

After disabling `autocommit` mode, changes to transaction-safe tables (such as InnoDB) are not made permanent immediately. You must use `COMMIT` to store your changes to disk or `ROLLBACK` to ignore the changes.

# Starting a Transaction

- `autocommit` is disabled when you start a transaction.
- Example:

```
START TRANSACTION;

SELECT Code FROM Country WHERE Name='Mexico';

UPDATE Country
  SET Name = 'World Cup Winner'
  WHERE Code = 'MEX';

COMMIT;
```

- `autocommit` is re-enabled when you end the transaction with **COMMIT** or **ROLLBACK**.
- Check the `autocommit` setting with **SELECT @@AUTOCOMMIT**.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If you are using a transactional storage engine such as InnoDB or NDB Cluster, you can also disable `autocommit` mode by using the following statement:

```
SET AUTOCOMMIT=0
```

After disabling `autocommit`, you must use **COMMIT** to store your changes to disk or **ROLLBACK** if you want to ignore the changes you have made since the beginning of your transaction. The `autocommit` mode then reverts to its previous state.

You can check the current setting as follows:

```
mysql> SELECT @@AUTOCOMMIT;
+-----+
| @@autocommit |
+-----+
| 1             |
+-----+
1 row in set (0.41 sec)
```

## Quiz

A transaction is a device for grouping together multiple SQL statements so that either all or none of the statements succeed.

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

## Retrieving Metadata

In an RDBMS, metadata is “data about data,” or information about the MySQL server, such as the name of a database or table, the data type of a column, and access privileges.

- There are two ways to obtain metadata:
  - Issue **SHOW** statements:
    - Provide information about databases, tables, and columns, as well as server status.
  - Query the **INFORMATION\_SCHEMA** database:
    - The **INFORMATION\_SCHEMA** database stores database metadata.
    - You query **INFORMATION\_SCHEMA** like any other database, with **SELECT**.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Metadata: SHOW Statements

- The **SHOW** statement is specific to MySQL.
- **SHOW** statement usage:
  - **SHOW DATABASES:** Databases on the MySQL server host
  - **SHOW TABLES:** Non-temporary tables in a database
  - **SHOW TABLE STATUS:** Like **SHOW TABLES**, but provides extra information about each non-temporary table
  - **SHOW COLUMNS:** Information about columns in a given table
  - **SHOW INDEX:** Table index information
  - **SHOW CREATE TABLE:** Shows the **CREATE TABLE** statement that creates a given table
- **SHOW** statements are concise and easy to remember.
- They cannot store results in other tables for later use.
- You can apply **WHERE** conditions to **SHOW** statements.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Some **SHOW** statements can obtain information that is unavailable in `INFORMATION_SCHEMA`, such as:

- **SHOW CREATE FUNCTION**
- **SHOW CREATE PROCEDURE**
- **SHOW MASTER/SLAVE STATUS**



## Metadata: INFORMATION\_SCHEMA Database

- Provides access to database metadata:
  - Schema and schema objects
  - Server statistics (status variables, settings, connections)
- Is a virtual database:
  - Tables are not “real” tables (base tables) but “system views.”
  - Tables are filled dynamically, according to user privileges.
- Is kept in a table format allowing flexible access:
  - Tables store metadata for all tables and columns
  - You use `SELECT` statements to query these tables
- Is part of the SQL standard—other RDBMSs can query it
- Allows query limits that `SHOW` does not
- You can store query results in other tables to use later.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The `INFORMATION_SCHEMA` database contains tables that hold information about:

- The tables in a database, their names, sizes, and number of rows in each table
- The table columns in a database, what tables they are used in, and the type of data each column contains

Database terminology refers to this set of metadata as the catalog. In the SQL standard, `INFORMATION_SCHEMA` is the means to access this catalog.

In `INFORMATION_SCHEMA`, there are several read-only tables, which are actually system views (not base tables) so there are no files associated with them.

Although `INFORMATION_SCHEMA` is a database, the server does not create a database directory with that name. You can issue a `USE` statement to make `INFORMATION_SCHEMA` the default database, but its tables are read only. You cannot insert, update, or delete records.

# Metadata: INFORMATION\_SCHEMA Tables

List all tables in the INFORMATION\_SCHEMA database:

```
mysql> SELECT TABLE_NAME
-> FROM INFORMATION_SCHEMA.TABLES
-> WHERE TABLE_SCHEMA = 'information_schema'
-> ORDER BY TABLE_NAME;
```

```
+-----+
| TABLE_NAME |
+-----+
| CHARACTER_SETS |
| COLLATIONS |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS |
| COLUMN_PRIVILEGES |
| ... |
| USER_PRIVILEGES |
| VIEWS |
+-----+
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The INFORMATION\_SCHEMA tables contain the following types of information:

## Table Information

- COLUMNS: Columns in tables and views
- ENGINES: Storage engines
- SCHEMATA: Databases
- TABLES: Tables in databases
- VIEWS: Views in databases

## Partitioning

- PARTITIONS: Table partitions
- FILES: Files in which MySQL NDB disk data tables are stored

## Privileges

- COLUMN\_PRIVILEGES: Column privileges held by MySQL user accounts
- SCHEMA\_PRIVILEGES: Database privileges held by MySQL user accounts
- TABLE\_PRIVILEGES: Table privileges held by MySQL user accounts
- USER\_PRIVILEGES: Global privileges held by MySQL user accounts

## Character Set Support

- `CHARACTER_SETS`: Available character sets
- `COLLATIONS`: Collations for each character set
- `COLLATION_CHARACTER_SET_APPLICABILITY`: Which collations are applicable to a particular character set

## Constraints and Indexes

- `KEY_COLUMN_USAGE`: Constraints on key columns
- `REFERENTIAL_CONSTRAINTS`: Foreign keys
- `STATISTICS`: Table indexes
- `TABLE_CONSTRAINTS`: Constraints on tables

## Server Settings and Status

- `KEY_COLUMN_USAGE`: Constraints
- `GLOBAL_STATUS`: The status values for all connections to MySQL
- `GLOBAL_VARIABLES`: The values used for new connections to MySQL
- `PLUGINS`: Server plug-ins
- `PROCESSLIST`: Indication of which threads are running
- `SESSION_STATUS`: The status values for the current connection to MySQL
- `SESSION_VARIABLES`: The values that are in effect for the current connection to MySQL

## Routines and Related Information

- `EVENTS`: Scheduled events
- `ROUTINES`: Stored procedures and functions
- `TRIGGERS`: Triggers in databases
- `PARAMETERS`: Stored procedure and function parameters, and stored functions

## InnoDB

- `INNODB_CMP` and `INNODB_CMP_RESET`: Status on operations related to compressed InnoDB tables
- `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET`: Status on compressed pages within the InnoDB buffer pool
- `INNODB_LOCKS`: Each lock that an InnoDB transaction has requested and holds
- `INNODB_LOCK_WAITS`: One or more row locks for each blocked InnoDB transaction
- `INNODB_TRX`: Every transaction currently executing inside InnoDB
- `TABLESPACES`: Active tablespaces

For more information about the `INFORMATION_SCHEMA` tables, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.6/en/information-schema.html>.

## Metadata: Viewing INFORMATION\_SCHEMA

Example of INFORMATION\_SCHEMA with TABLE\_SCHEMA:

```
mysql> SELECT table_name, engine
      -> FROM INFORMATION_SCHEMA.TABLES
      -> WHERE table_schema = 'world_innodb';
```

table_name	engine
city	InnoDB
country	InnoDB
country_view	NULL
countrylanguage	InnoDB
countrylanguage2	InnoDB
per_capita_v	NULL

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The example in the slide lists all the tables in the `world_innodb` database, showing the name of each table and its engine.

**Note:** All users have access to these tables, but only see rows for the objects they have access to.

## Metadata: SCHEMATA

Example of INFORMATION\_SCHEMA with SCHEMA\_NAME:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA
      -> WHERE SCHEMA_NAME = 'world_innodb'\G
***** 1. row *****
          CATALOG_NAME: def
          SCHEMA_NAME: world_innodb
DEFAULT_CHARACTER_SET_NAME: latin1
DEFAULT_COLLATION_NAME: latin1_swedish_ci
          SQL_PATH: NULL
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The INFORMATION\_SCHEMA database has a SCHEMATA table that contains database metadata (information about databases).

## Quiz

**INFORMATION\_SCHEMA** contains several tables, which store information which is similar to the results of **SHOW** statements.

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

# MySQL Performance Schema

The MySQL Performance Schema is a feature for monitoring MySQL server execution at a low level.

- Focuses primarily on performance data
- Monitors server events on all supported platforms
  - An “event” is anything the server does that takes time.
- Inspects internal execution of the server (at run time) by using:
  - Performance Schema storage engine
  - **PERFORMANCE\_SCHEMA** database
- Has minimal impact on server performance

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The MySQL Performance Schema stores information about server performance based on server events. A server event is any task the server performs that takes time and that has been instrumented to make this timing information available. An event can be a function call, a wait for the operating system, a single SQL statement, a group of statements, or stages within a statement, such as parsing or sorting. Event collection provides access to information about synchronization calls, file and table I/O, table locks, and so on for the server and for several storage engines.

This information lets you see how various low-level items factor in to overall database performance, see which ones are the busiest under various workloads and system configurations, and trace issues back to the relevant file and line in the source code so you can understand what is happening behind the scenes.

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). Performance Schema events are specific to a given instance of the MySQL server. In MySQL 5.6.9 and later, Performance Schema tables are considered local to the server, and changes to them are not replicated or written to the binary log.

For more information about Performance Schema features, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.6/en/performance-schema.html>.

# Performance Schema: Implementation

- The Performance Schema storage engine:
  - Collects event data by using “instrumentation points” in server source code.
    - It is easy to add new instrumentation points.
    - Instrumented code is backward-compatible when new versions are implemented.
- The **PERFORMANCE\_SCHEMA** database:
  - Stores collected events in tables
    - You query these tables by using `SELECT` statements.
  - Allows you to change how the schema collects data by updating tables using SQL statements
  - Uses views or temporary tables that are not permanent

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can customize data performance schema collects by modifying the server source code to add instrumentation. There are no separate threads associated with the Performance Schema, unlike other features such as replication or the Event Scheduler.



# Performance Schema: SHOW Database Tables

List all tables in the `PERFORMANCE_SCHEMA` database:

```
mysql> SHOW TABLES FROM PERFORMANCE_SCHEMA;
+-----+
| Tables_in_performance_schema |
+-----+
| accounts                     |
| cond_instances               |
| events_stages_current        |
| events_stages_history        |
| events_stages_history_long   |
| events_stages_summary_by_account_by_event_name |
| events_stages_summary_by_host_by_event_name |
| . . .                        |
| socket_instances             |
| socket_summary_by_event_name |
| socket_summary_by_instance   |
| table_io_waits_summary_by_index_usage |
| table_io_waits_summary_by_table |
| table_lock_waits_summary_by_table |
| threads                      |
| users                        |
+-----+
52 rows in set (0.00 sec)
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

There are many tables in the `PERFORMANCE_SCHEMA` database, which can be categorized as follows:

- **Setup tables**  
Who to monitor, what to monitor, how to monitor it, and where to store the results
- **Raw Data tables**  
The actual data for current events, objects, or instances of instruments, and a brief history
- **Current events tables**  
Summaries of event data over various dimensions. Useful for longer-term monitoring.

For detailed information about `PERFORMANCE_SCHEMA` and help with setup, troubleshooting, and diagnosis, refer to the MySQL Reference Manual:

<http://dev.mysql.com/doc/refman/5.6/en/performance-schema.html>.

## Quiz

The `PERFORMANCE_SCHEMA` database stores event data in tables. You can change how data is collected by updating tables using SQL statements.

- a. True
- b. False

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

# MySQL Enterprise Backup

The MySQL Enterprise Backup (MEB) product performs “hot backup” operations for MySQL databases.

- MEB executes efficient and reliable backups of InnoDB tables.
- It performs hot backups while the database is running.
- Use it when your database:
  - Is large and takes a long time to back up
  - Cannot be taken offline
- MEB lets you take complete “warm backups” for some non-InnoDB tables.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

“Hot” backups do not block normal database operations and capture changes that occur while the backup is happening. Hot backups are desirable when your database is large enough for backups to take a long time and important enough so that your application, website, or web service cannot be taken offline.

MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables that use MyISAM or other non-InnoDB storage engines it does a “warm” backup. In a warm backup, the database remains operational, but non-InnoDB tables cannot be modified while being backed up. The most efficient backup strategy is to use InnoDB as the storage engine for all tables.

For more information about MySQL Enterprise Backup, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.6/en/mysql-enterprise-backup.html>.

# MySQL Enterprise Backup: Preparation

Important tasks to perform before your first backup:

- Collect pertinent database information and decide on directory names.
- Grant the necessary privileges to the users who will administer the backup.
- Designate a location with ample storage for backup data.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Before backing up a particular database server for the first time, determine the following:

- Path to the MySQL configuration file
- MySQL server port number
- Path to the MySQL data directory
- Username and password of the MySQL user with the necessary privileges
- File system path to back up data to
- Owner and permission details of backup files

The `mysqlbackup` command usually connects to the MySQL server via the `--user` and `--password` options. The user performing the operation requires certain privileges. You can either create a dedicated user account for this purpose with a minimal set of privileges, or use an administrative account such as the root user.

Choose a directory where the backup files will go and make sure there is sufficient storage space.

# MySQL Enterprise Backup: Implementation

- Use the `mysqlbackup` command for all backup and restore operations. `mysqlbackup`:
  - Is easy to use
  - Backs up:
    - All InnoDB and MyISAM tables and indexes
    - Tables managed by other storage engines
    - Other files beneath the MySQL data directory
- In addition to creating backups, `mysqlbackup` can:
  - Pack and unpack backup data
  - Apply to the backup data any changes to InnoDB tables that occurred during the backup operation
  - Restore data, index, and InnoDB log files to their original positions

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

InnoDB table backups include:

- The InnoDB system tablespace (which, by default, contains all InnoDB tables)
- Any separate data files produced if the `innodb_file_per_table` option is set. Each file contains one table and its associated indexes. The files can use either the original Antelope or the new Barracuda file format.

Other files backed up from the MySQL data directory include the `.frm` files that record the structure of each table.

**Note:** Although the `mysqlbackup` command backs up InnoDB tables without interrupting database use, the final stage that copies non-InnoDB files (such as MyISAM tables and `.frm` files) temporarily puts the database into a read-only state. Therefore, do not run any long queries, and keep MyISAM tables small and for read-only or mostly read-only work.

For more information about backups with MEB, see the MySQL Enterprise Backup User's Guide at: <http://dev.mysql.com/doc/mysql-enterprise-backup/3.8/en/index.html>.

## MySQL Enterprise Backup: Running mysqlbackup

- You can include connection options on the command line, or store them in the MySQL configuration file and reference the file instead.
- A folder is created in the specified backup location with all data.
- Example of command-line options:

```
shell> mysqlbackup --user=dba --password=dba_pswd
--port=3306 --with-timestamp
--backup-dir=D:/mysql_backups backup

MySQL Enterprise Backup version 3.8.1 [Mon 01/28/2013 ]
...
-----
Backup Config Options:
-----
datadir                = D:\mysql_backups\2012-01-11_10-07-23\datadir
innodb_data_home_dir  = D:\mysql_backups\2012-01-11_10-07-23\datadir
...
mysqlbackup completed OK!
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can include the connection options in the [mysqlbackup] section of the MySQL configuration file and reference the file in mysqlbackup:

```
mysqlbackup --defaults-file=/usr/local/mysql/my.cnf backup
```

In fact, any option that you can specify on the command line you can alternatively include in the [mysqlbackup] section of the configuration file.

Alternatively, you can specify the configuration file as shown, but override some of those options on the command line.

The command reports the steps taken to complete the backup, as well as the location of the input data and the backup output files.

In the example in the slide, the dba user (with the password dba\_pswd) backs up all the existing MySQL server database data to the D:\mysql\_backups directory in a folder named after the date and time of the backup. The folder name is shown in the report.

Make sure that the user or cron job running mysqlbackup has the necessary permissions to copy files from the MySQL database directories to the backup directory.

Also, ensure that your connection timeouts are long enough for the command to keep the connection to the server open for the duration of the backup. mysqlbackup pings the server after copying each database to keep the connection alive.

## MySQL Enterprise Backup: Some `mysqlbackup` Options

- Subcommands:
  - `backup`: Performs the initial phase of a backup
  - `apply-log`: Applies incremental backups or InnoDB table changes during a backup, to a full backup.
- Standard options:
  - `--no-defaults`: Do not read default options from any file.
  - `--defaults-file=PATH`: Read options from only this file.
- Server repository options:
  - `--datadir=PATH`: Location of MySQL data files
  - `--innodb_data_file_path=VALUE`: Specifies InnoDB data file names and sizes (Example: `ibdata1:32M:autoextend`)
- Backup repository options:
  - `--backup_dir=PATH`: Location to store the backup data

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

MySQL Enterprise Backup separates the backup phase into two parts: “backup” and “applylog”. While the backup must be run on the server whose backup has to be taken, the applylogs can be run on any independent server. This allows users to offload an intensive portion of the backup process to an offline machine.

There are many more subcommands and options for `mysqlbackup`. See the MySQL Enterprise Backup User Guide for a full listing and explanation of these options:

<http://dev.mysql.com/doc/mysql-enterprise-backup/3.8/en/mysqlbackup.html>.

## MySQL Enterprise Backup: Recovering or Restoring a Database

You use backup data to recover from a database issue, or to create a clone of the original database in another location.

- Use `mysqlbackup` with the `copy-back` subcommand instead of `backup` to restore files to their original locations.
- Use MEB with a MySQL binary log to create a “point-in-time” recovery from the hot backup.
  - Dump all SQL activity after the binlog position of the backup using `mysqlbinlog` and pipe back into `mysql`.
- Use MEB to set up a replication slave database without stopping the master, by backing up the master and restoring that backup on a new slave server.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You use a backup to recover your data as quickly as possible after a database issue, or to create a clone of the original database for reporting, or to create a new replication slave.

You must shut down the database server before running `mysqlbackup` with `copy-back`. It copies the data files, logs, and other backed-up files from the backup directory back to their original locations, and performs any required post-processing on them.

To recover the database to a specific point in time:

- You must enable binary logging in MySQL before taking the backup that serves as the basis of the restore operation
- Find the binlog position that corresponds to the time of the backup. InnoDB stores the binlog position information to its tablespace only after committing a transaction.
- Pipe the output from `mysqlbinlog --start-position=...` directly to `mysql` to replay all the SQL statements after the last backup

You can also use replication to secure data at another location.



## Quiz

Use the following subcommand with `mysqlbackup` to run the initial phase of a backup.

- a. `backup`
- b. `--datadir`
- c. `copy-back`

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

**Answer: a**

# Summary

After completing this lesson, you learned how to:

- Explain the general purpose of storage engines in MySQL
- Determine which storage engine a specific table uses
- Describe the key features of the InnoDB storage engine
- Create table views with specific data
- Explain and use transactional processing
- Define ACID compliance for transaction safety
- Describe the `INFORMATION_SCHEMA` database, and use it to obtain metadata
- Describe the `PERFORMANCE_SCHEMA` database
- Explain the key features of MySQL Enterprise Backup

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 15-1 Overview: Displaying Storage Engine Information

In this practice, you use the following statements to retrieve information about storage engines:

- **SHOW CREATE TABLE**
- **SHOW TABLE STATUS**
- **SHOW ENGINES**

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Practice 15-2 Overview: Displaying and Creating Views

In this practice, you determine the types of tables in a database and create a view from an existing `world_innodb` table.

## Practice 15-3 Overview: Obtaining Metadata

In this practice, you use the `INFORMATION_SCHEMA` database to show metadata.

## Practice 15-4 Overview: (Optional) Creating a Backup of MySQL Databases

In this practice, you use MySQL Enterprise Backup to create a backup of your MySQL databases.

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

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# 16

## Conclusion

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Ashley Ransoo (ashley.ransoo@hee.nhs.uk) has a non-transferable license to use this Student Guide.

# Course Goals

In this course, you learned how to:

- Describe the features and benefits of MySQL
- Explain the basics of relational databases
- Describe MySQL connectors, and their major features and differences
- Explain the SQL and MySQL languages
- Describe data/column types with regard to efficient database design
- View a database design structure and content
- Create a database design by using an efficient structure
- Extract basic database information using the **SELECT** command
- Troubleshoot and identify typical syntax warnings and errors

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.



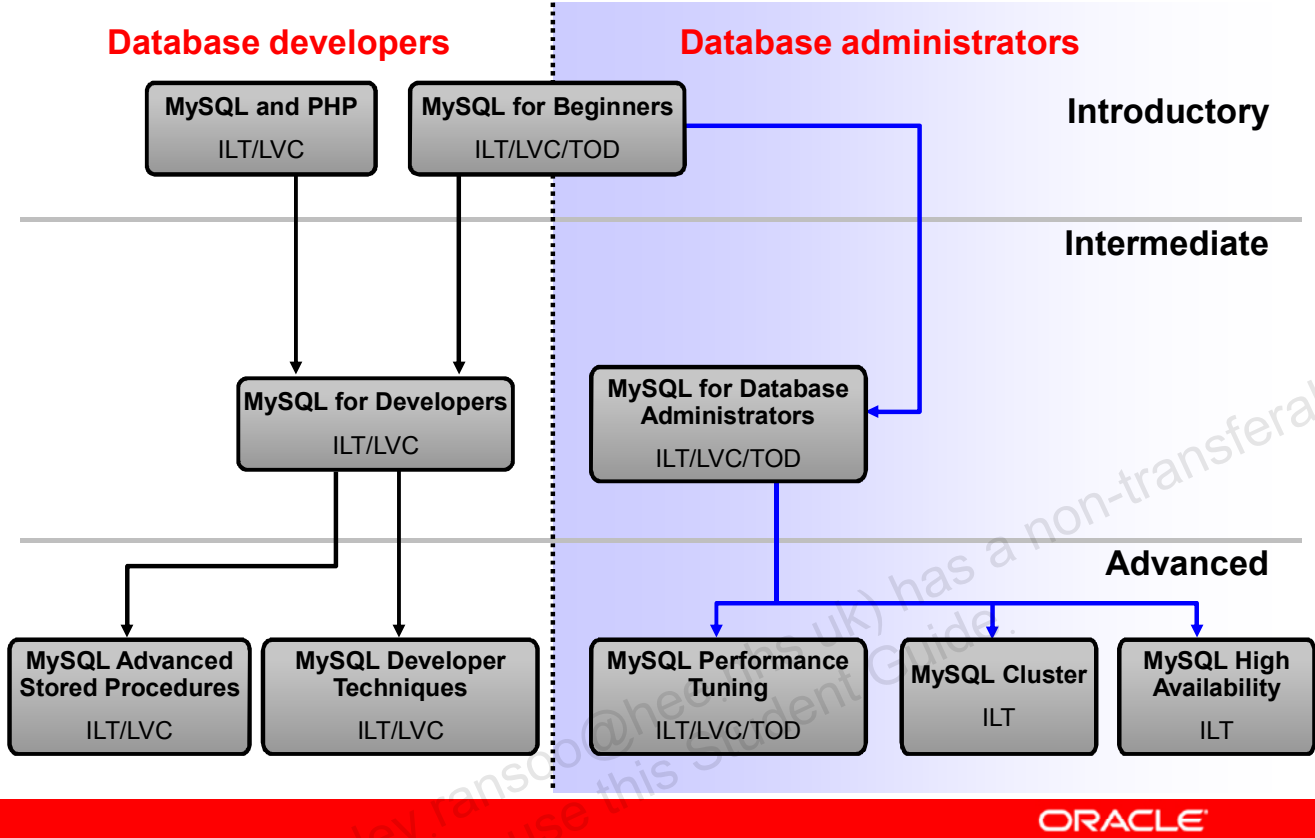
## Course Goals

- Delete or modify a database and table row data
- Group query data with aggregation
- Connect data from multiple table rows with a join
- Perform nested subqueries
- Use simple functions (String, Date, Numerical)
- Explain MySQL storage engines and transactions, and the features of the common engines
- Export and import database data
- Obtain database metadata
- Monitor server database performance
- Describe MySQL graphical user interface tools
- Perform database backup and recovery

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

# MySQL Curriculum Path



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

# MySQL Resources

- MySQL training
  - <http://www.oracle.com/education/mysql>
- MySQL certification
  - <http://www.oracle.com/certification>
- Additional learning resources
  - <http://www.mysql.com>
    - Webinars and demos
    - Articles
    - White papers
- Developer Zone
  - <http://dev.mysql.com/>
    - Product downloads
    - Community communication



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

## Your Evaluation

- Because courses are continually updated, your feedback is invaluable.
- Thank you for taking the time to give your opinions.

ORACLE

Copyright © 2013, 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 © 2013, Oracle and/or its affiliates. All rights reserved.

## Q&A Session

- Questions and answers
- Questions after class
  - Get answers from the online reference manual.  
<http://dev.mysql.com/doc/refman/5.6/en/faqs.html>
- Example databases
  - Download the `world_innodb` and other sample databases from our website:  
<http://dev.mysql.com/doc/index-other.html>  
(under Example Databases)

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.