MySQL for Beginners

Electronic Presentation

SQL-4401 Rev 2.1

D61918GC10
Edition 1.0
D61918GC10_EP

ORACLE

MySQL® for Beginners

Revision2.1

Time for Individual Presentations!

- What is your background?
  - Your name and your organization
  - How does your organization use MySQL?
- Why are you here?
  - What do you expect from the course?
  - What do you want to learn?

- Have experience using MySQL?
- Have experience from other SQL dialects?
- Have installed MySQL yourself?
- Use MySQL under Linux? Windows? Solaris? MacOSX? Other?
- Use MySQL 3.21? 3.22? 3.23? 4.0? 4.1? 5.0? 5.1?
- Use MySQL with PHP? Perl? Java? C? ODBC? Others?
- Are you MySQL Certified? Which certifications?

Practical Checklist

- Participant list -- email addresses -- checkup email
- Breaks -- timing, duration
- Where are the toilets/restrooms?
- Lunch arrangements
- Time allotted for exercises -- Solutions
- When does the day end? Can you stay longer?
- When do the days begin? Can you come earlier?
- Evaluation forms

Slide Format (Topic)

- Presentation sub-topics
  - More sub-topic information

Topics and Sub-Topics, presented in detail by instructor, Following Student Guide content

Course Name and Version

```
shell> mysql -h training.mysql.com -u wld world
```

Code Examples

Current Chapter Title

Current <*chapter.section*> Numbers and Title

Course Objectives (1/2)

- List the features and benefits of MySQL
- Find information about products, support and training
- Install and start the MySQL server
- Explain the basics of relational databases
- Distinguish the SQL language from MySQL language extensions
- Explain data/column types with regard to efficient database design
- Inspect design, structure and content of databases
- Design and Create an efficiently structured database
- Retrieve data using the SELECT statement

Course Objectives (2/2)

- Troubleshoot typical warnings and errors
- Modify and delete a database
- Modify and delete table row data
- Use data aggregation in queries
- Combine data from multiple tables using JOIN
- Write subqueries
- List simple functions (String, Date, Numerical)
- Understand the primary methods for exporting and importing data
- Describe MySQL connectors and their features
- Explain MySQL storage engine concepts

Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Explain the origin and status of the MySQL product
- List the MySQL products and professional services
- Describe the MySQL Enterprise subscription
- List the currently supported operating systems
- Describe the MySQL Community web page
- Describe the MySQL Certification program
- List all the available MySQL courses

MySQL Overview

- Relational Database Management System Program Suite
- World's most popular open source database
  - Fastest growing with over 70,000 downloads per day
- Originally developed by MySQL AB (Sweden)

MySQL is installed on every continent in the world

(Yes, even Antarctica!)

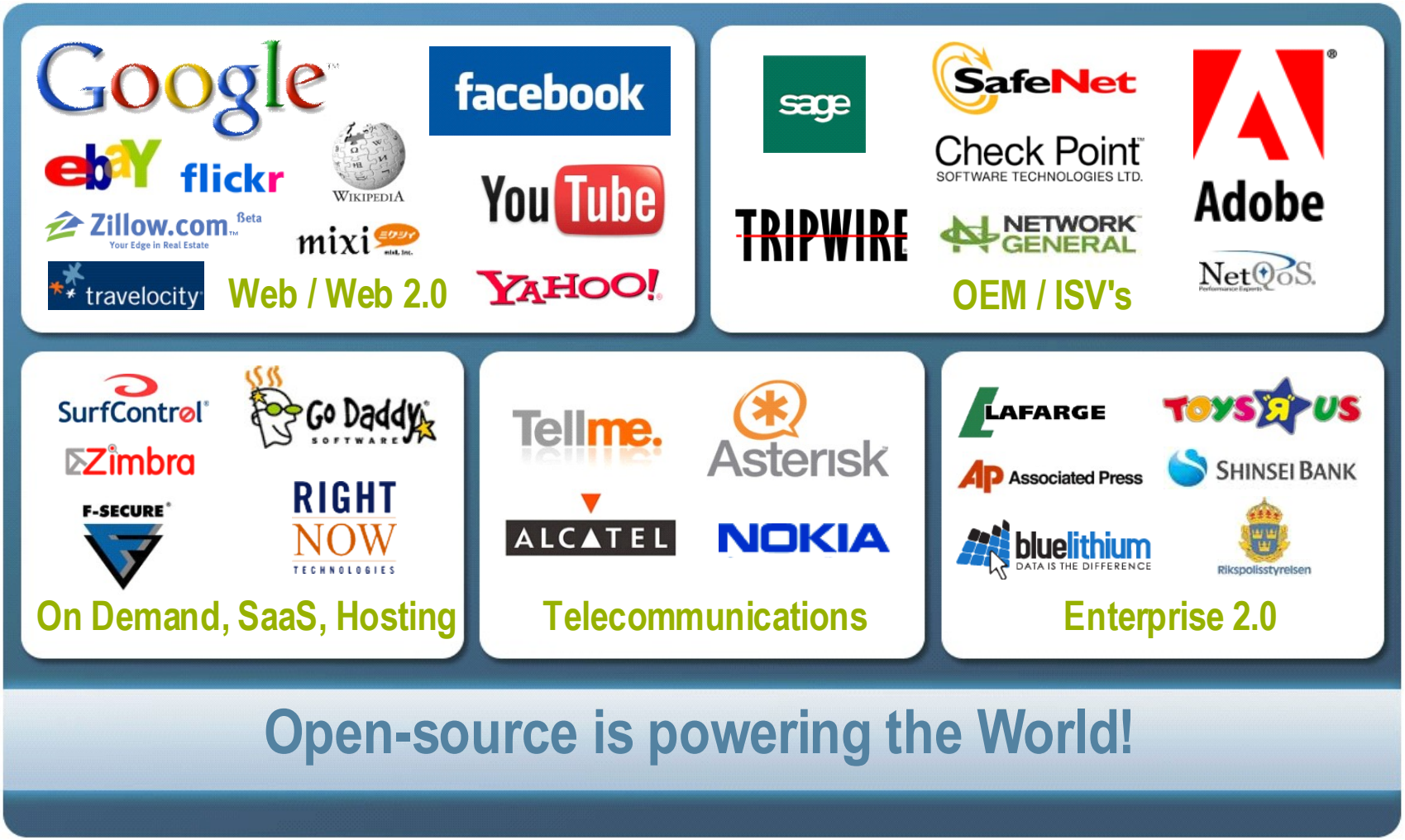
Sun Acquisition of MySQL

- Acquired by Sun Microsystems in 2008
  - Giving MySQL the considerable resources of a major mainstream company
- Sun and MySQL together provide powerful, global product and services
  - Enterprise class support 24x7x365
  - More resources to draw from
  - More platform choices now available
- Both companies are strong open source supporters

Sun/MySQL Mission:

Make superior database software available and affordable to all!

You Are in Good Company!



MySQL Database Server Products

- Enterprise Server
  - Enterprise-grade database
- Community Server
  - Database server for open source developers

- Embedded Database
  - Database server for OEMs/ISVs to bundle cost-effectively
- Cluster
  - Fault tolerant database clustering architecture
  - Carrier-grade availability and performance

MySQL GUI Tools

- Graphical User Interfaces to your MySQL database
- EOL (end-of-life) for MySQL GUI Tools Bundle
  - **MySQL Migration Toolkit**
    - Migration GUI Wizard
  - **MySQL Administrator**
    - Administration console
  - **MySQL Query Browser**
    - Create databases, execute and optimize SQL queries

- MySQL Workbench will contain the above GUI features starting with the 5.2 revision
  - Added to the existing Workbench features

Other MySQL Tools

- MySQL Workbench
  - Visual database design tool
    - Used to efficiently design, manage and document databases
  - Available in two editions
    - Community edition
    - Standard edition

- MySQL Proxy
  - A program that communicates between a client and a MySQL server
    - Can monitor, analyze or transform communication

MySQL Connectors

- MySQL Developed Connectors
  - MySQL C API
  - MySQL Connector/ODBC
  - MySQL Connector/J
  - MySQL Connector/NET
  - MySQL Connector/C
  - MySQL Connector/C++
- Developed by Community
  - PHP
  - Perl
  - Python
  - Ruby
  - C++ Wrapper

Solutions for Embedding MySQL

- MySQL also provides libraries to embed a database
- libmysqld
  - Embedded edition of the **`mysqld`** server program wrapped in a shared library
  - Allows the MySQL to be embedded in C programs
- MySQL MXJ
  - A JAR wrapper around **`mysqld`** binaries
  - Allows java programs and J2EE environments to instantiate (and install) a MySQL server

MySQL Services

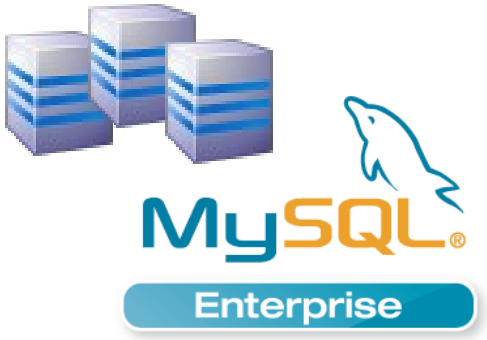
- MySQL Training
  - Comprehensive set of MySQL training courses
- MySQL Certification
  - High quality certification for MySQL Developers and Database Administrators
- MySQL Consulting
  - Full range of consulting services from start-up to optimization
- MySQL Support
  - Community
  - Enterprise (and other levels of purchased support)

MySQL Enterprise Subscription

- Annual offering
- Access to Enterprise Server and other premium products
- Enterprise-grade software, support and services
- Highest level of reliability, security and availability
- Pro-active services help eliminate problems *before* they occur

MySQL Enterprise Server

- Most reliable, secure, updated version of MySQL
- Updates and Service Packs
- Emergency Hot Fix builds
- Drivers (Connectors)
- Many different supported platforms

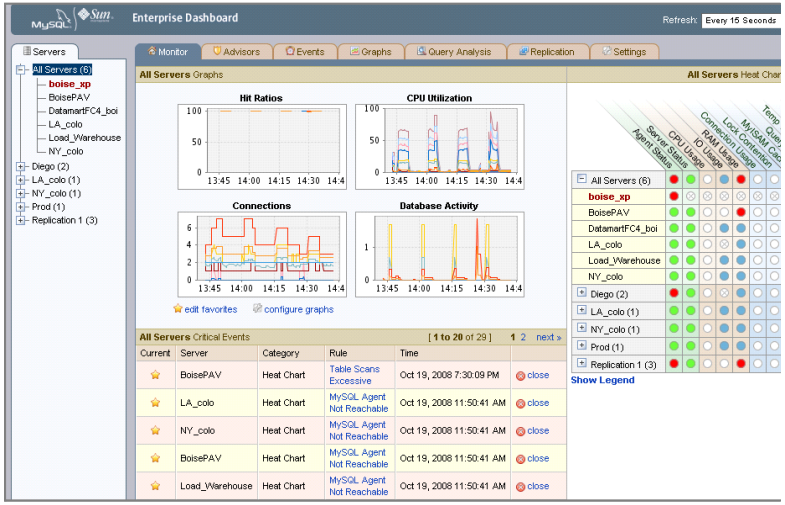


MySQL Enterprise Support

- 24x7 Production-level support
- Flexible service levels available
- High priority problem resolution
- Consultative Support
- Technical Account Manager (TAM)
- Online Knowledge Base

MySQL Enterprise Monitor

- Web-based monitoring and advising system
- Virtual DBA assistant
- Runs completely within corporate firewall
- Principle features:
  - Enterprise Dashboard
  - Server/group management
  - "At-a-glance" monitoring
  - MySQL & custom advisors/rules
  - Advisor rule scheduler
  - Customizable thresholds & alerts
  - Events and Alert history
  - Specialized scale-out assistance
  - Query Analyzer



Obtaining a MySQL Enterprise Subscription

- Contact sales personnel
- Purchase from our website
  - https://shop.mysql.com/enterprise/
- 30 day trial
  - Limited features
  - http://www.mysql.com/trials/

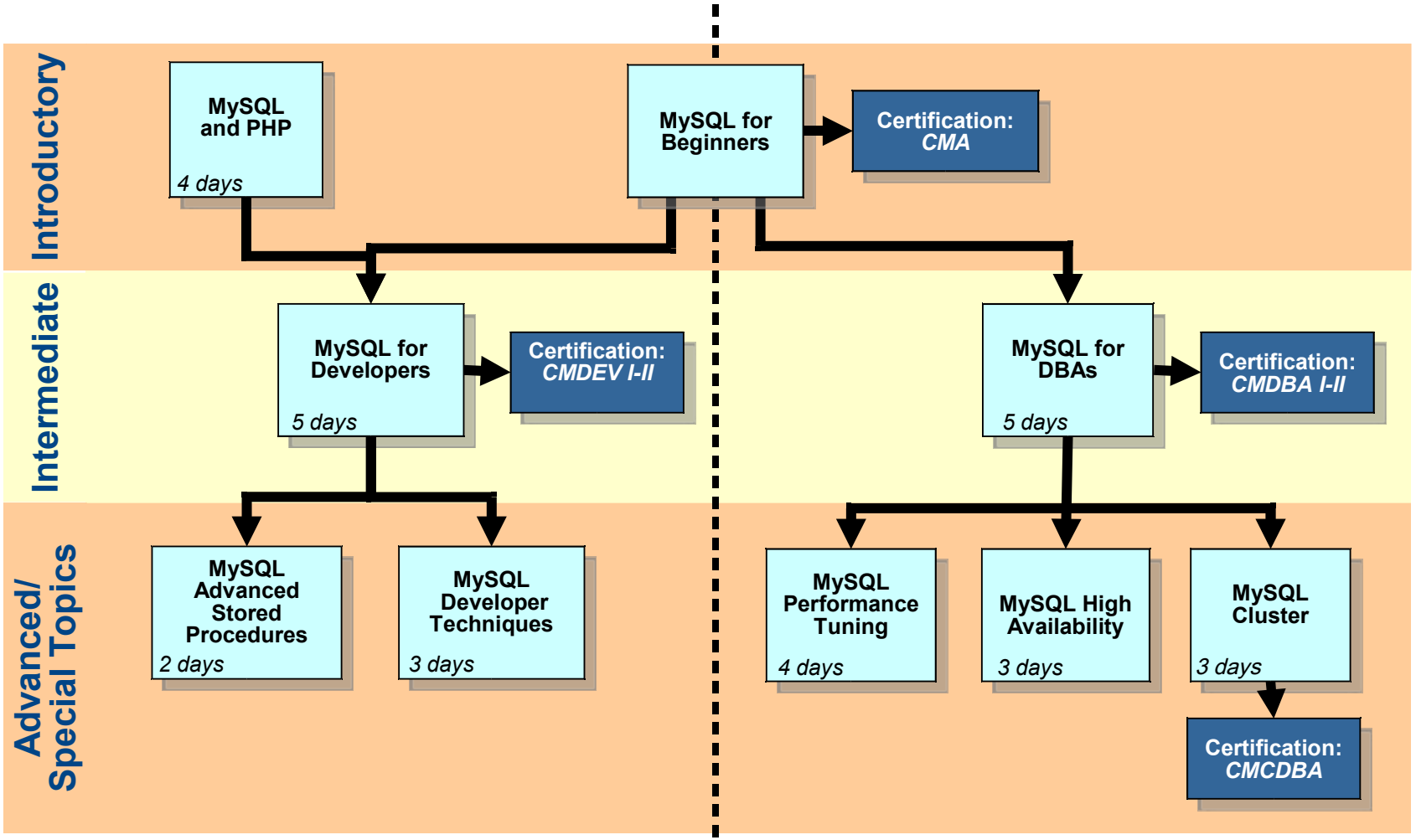
MySQL Supported Operating Systems

- More than 20 platforms
- Control and flexibility for users
- Currently available for MySQL download:
  - Windows (multiple)
  - Linux (multiple)
  - Solaris
  - FreeBSD
  - Mac OS X
  - HP-UX
  - IBM AIX and i5
  - Open BSD
  - SGI Irix
  - Novell NetWare
  - Source Code
  - Special Builds

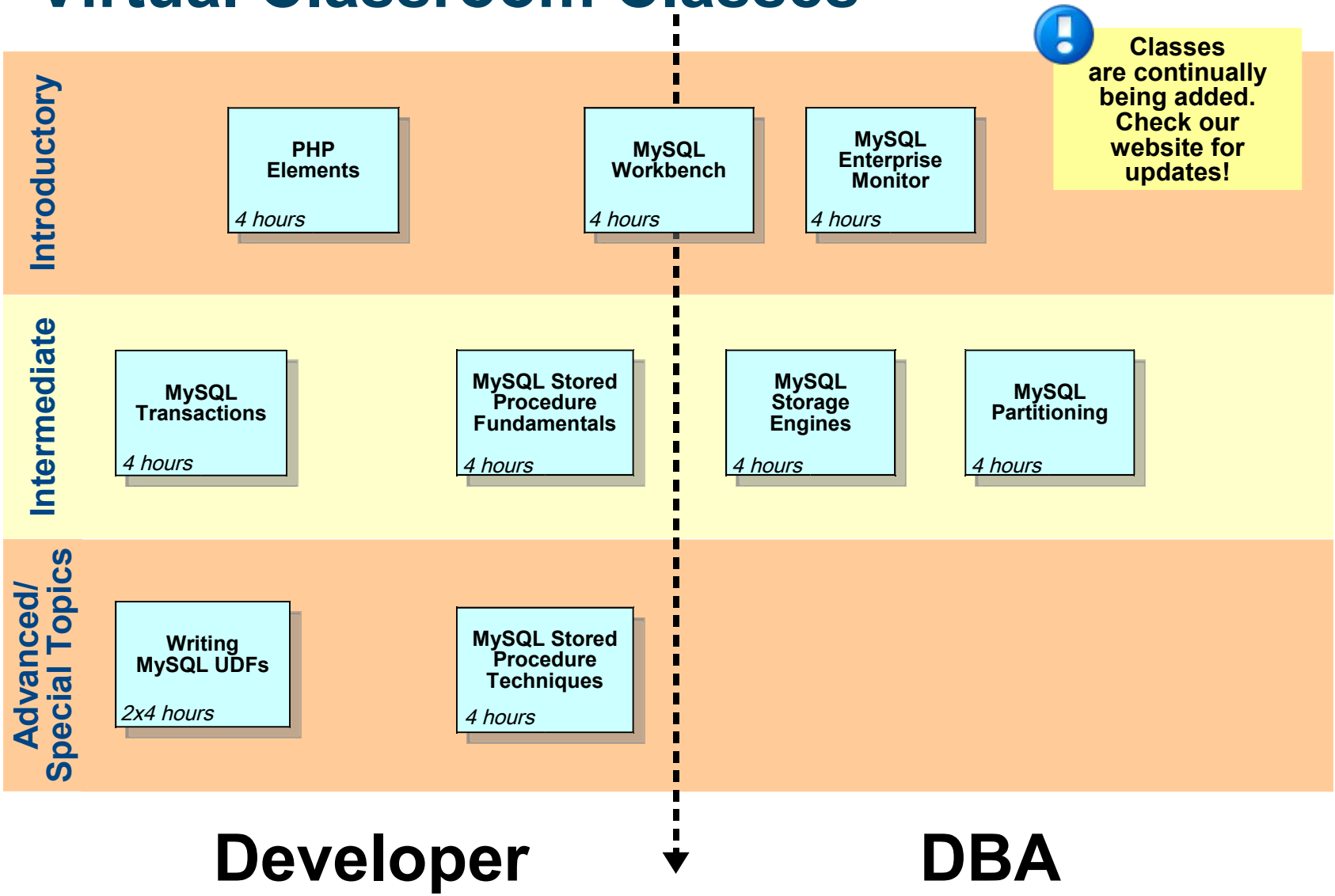
Certification Program Overview

- MySQL certifications available
  - Certified MySQL Associate (CMA)
  - Certified MySQL 5.0 Developer (CMDEV)
  - Certified MySQL 5.0 Database Administrator (CMDBA)
  - Certified MySQL Cluster DBA (CMCDBA)
- Certification web page
  - http://www.mysql.com/certification/
- Exam administration
  - Administered in several locations world-wide

Curriculum Paths

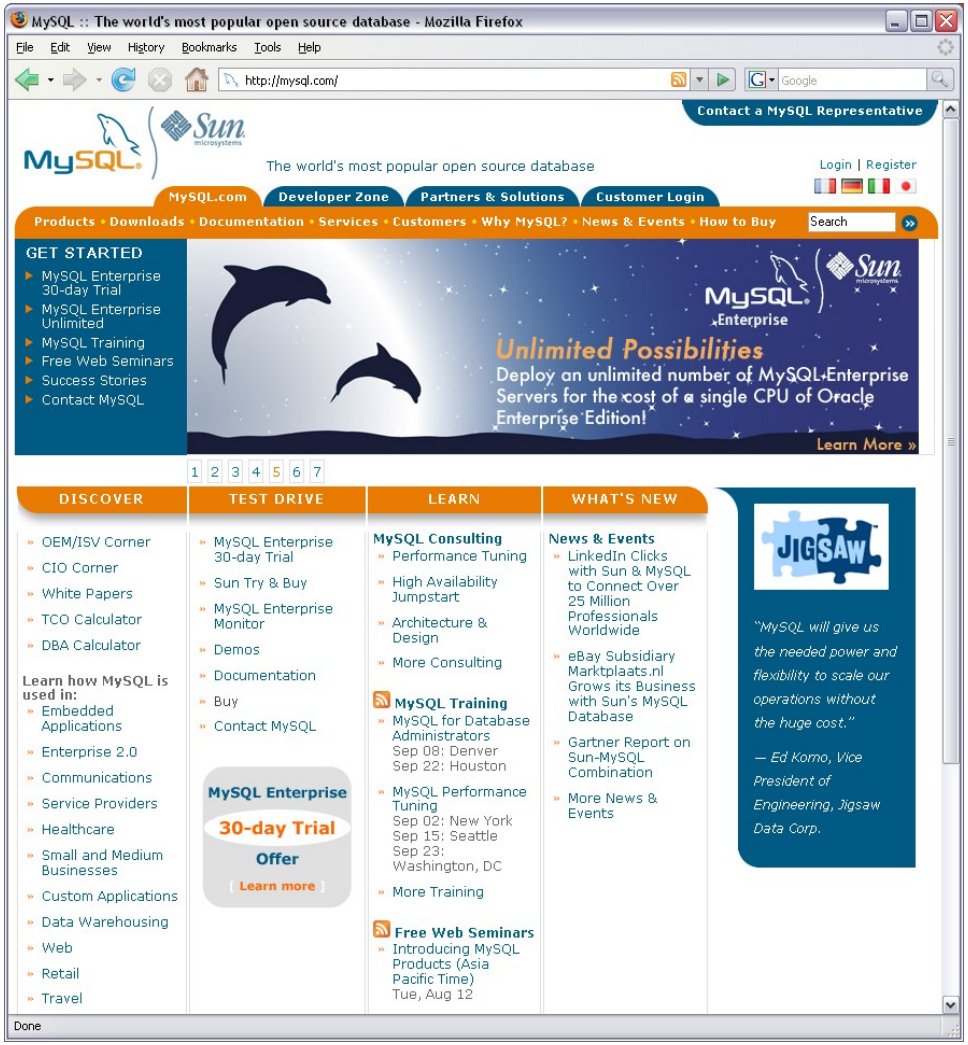


Virtual Classroom Classes



MySQL Website

- http://www.mysql.com/



MySQL Community Web Page (1/2)

- Developer Zone
- http://dev.mysql.com/

MySQL Community Web Page (2/2)

- Current product/service promotions
- Get Started with MySQL
- Developing with...
- MySQL Librarian
- MySQL Server Community Edition
- New Releases
- Software Previews
- What's New
- MySQL Training
- MySQL Quickpoll
- Stay Connected
- Resources

MySQL Online Documentation

- MySQL Reference Manual
- Excerpts from the Reference Manual
- MySQL GUI Tools Manuals
- Expert Guides
- MySQL Help Tables
- Example Databases
- Meta Documentation
- Community Contributed Documentation
- Printed Books
- Additional Resources

Installing MySQL

- Use the MySQL website to download
  - http://dev.mysql.com/downloads
- Several different platforms are supported
- “Windows” used for this course

Installing 'world' Database

- MySQL provides three example databases
- Can be downloaded from our website
- '**world**' database will be used for demonstration and exercises in this course

Chapter Summary

- Explain the origin and status of the MySQL product
- List the MySQL products and professional services
- Describe the MySQL Enterprise subscription
- List the currently supported operating systems
- Describe the MySQL Community web page
- Describe the MySQL Certification program
- List all the available MySQL courses

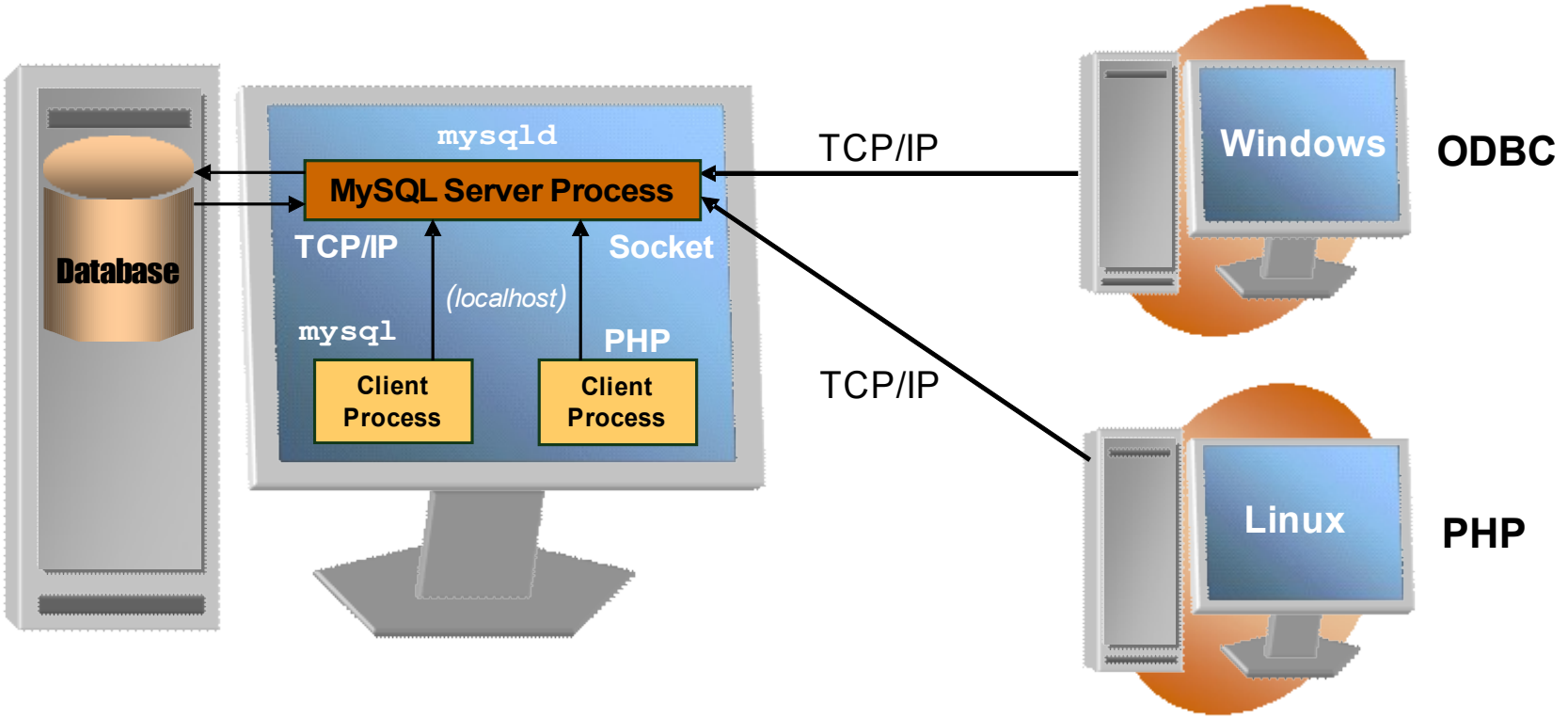
Course Chapter Map

Learning Objectives

- Explain the MySQL client/server model
- Describe the communication protocols used to interact with the MySQL server
- Explain how MySQL works with API's
- Identify the components of a LAMP stack, and their roles
- Start the MySQL server and the MySQL client

The Client/Server Model

- Server - the central database management program
- Client - program(s) connect to the server to retrieve or modify data



Communication Protocols

- TCP/IP
  - **T**ransmission **C**ontrol **P**rotocol
  - **I**nternet **P**rotocol
- Unix Socket
- Shared Memory
- NT Pipes

Connectors and MySQL

- MySQL provides several API's to the server
- Supports Windows, Unix and Mac OS X formats
- Officially supported
  - ODBC
  - J
  - NET
- Third-party client interfaces
- C API based

The LAMP Stack

- **L**inux
  - Operating system
- **A**pache
  - Web server
- **M**ySQL
  - Relational Database Management System
- **P**HP / **P**erl / **P**ython
  - Programming languages
- Offers many degrees of freedom
- Go from LAMP -> **W**AMP (Windows)

MySQL Server and the MySQL Client

- Server connects during installation
- Command line client can ...
  - Send queries
  - Receive query results

Starting the MySQL Command Line Client

- Standard 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: 5.1.39-community MySQL
Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to
clear the buffer.
mysql>
```

MySQL Start-Up Command Line Options

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

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

- Some primary options

```
-h hostname or IP
-u username
-ppassword
database
< input_script
> filename
-b
```

- Example

```
shell> mysql -h training.mysql.com -u wld world -p
< sqlcommands.sql
```

Keyboard Editing

- Keyboard directional arrows
- Command history preserved during session
- Interrupt a query
- Full readline capabilities under Linux
- Close quotes
- Copy/paste into window

Chapter Summary

- Explain the MySQL client/server model
- Describe the communication protocols used to interact with the MySQL server
- Explain how MySQL works with API's
- Identify the components of a LAMP stack, and their roles
- Start the MySQL server and the MySQL client

Course Chapter Map

Learning Objectives

- Define Relational Database (and RDBMS)
- Describe the structure of an RDBMS database
- Explain the use of MySQL with relational databases
- Define and use Data Definition Language
- Define and use Data Manipulation Language

What is a RDBMS?

- Relational Database Management System
  - Manages data according to a relational model
  - Organizes and stores data in the form of tables
    - Bank example: tables for currency data, customer info, account info
- Database
  - A collection of data
  - Synonymous with ‘Schema’
- The relational model was invented to solve problems with the hierarchical database model
  - Data logically organized as a tree
  - Time consuming and difficult to develop applications and perform required operations

Entities and Relationships (1/2)

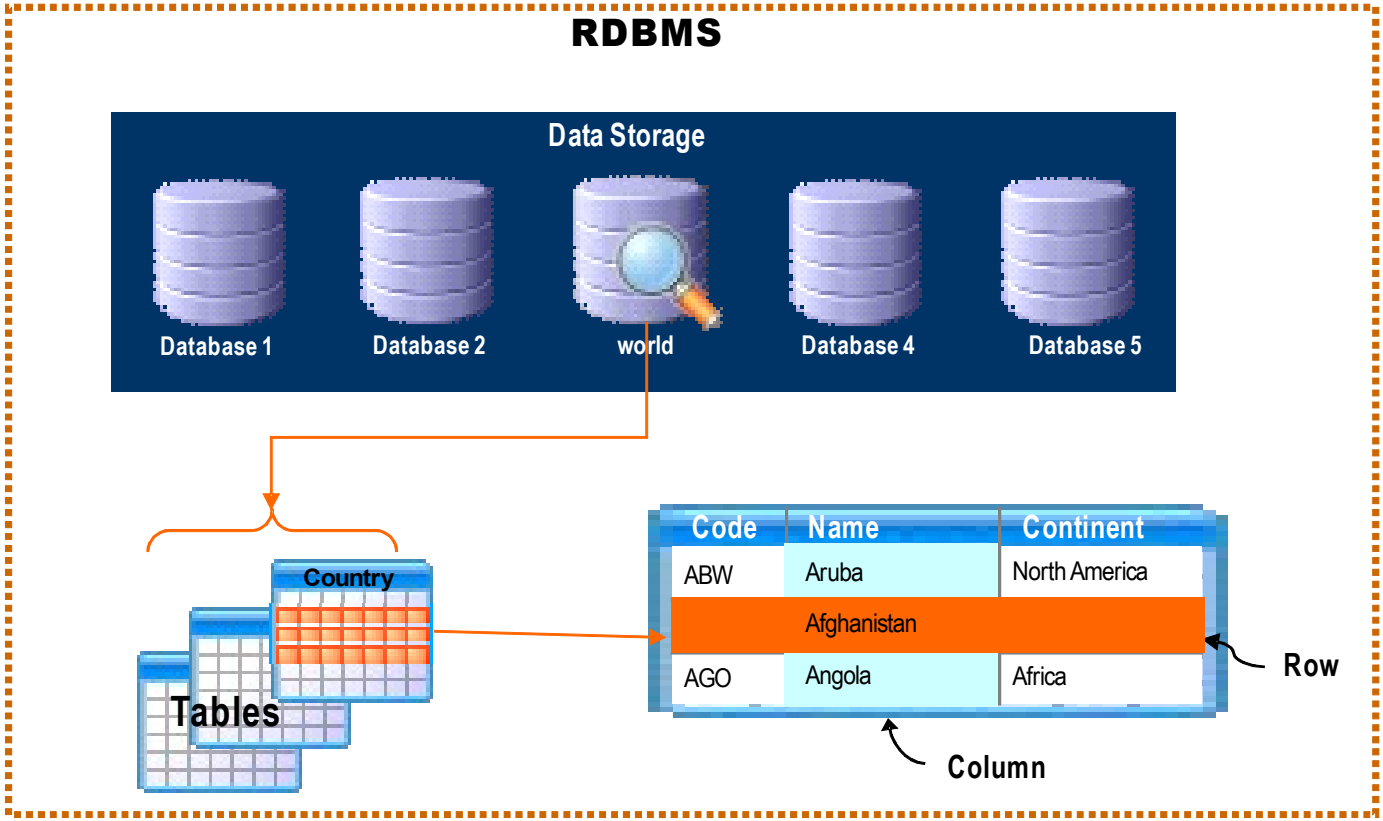
- Entities
  - Things in the real world that we will store information about in the database
- Relationships
  - Links between entities
- Tables typically store data representing one type of entity
  - For instance, a bank database has a customer table specifically for customer information

Entities and Relationships (2/2)

- Relationship categories
  - One-to-One
  - One-to-Many
  - Many-to-One
  - Many-to-Many
- Relationship examples
  - A bank customer can have multiple accounts
    - The relationship between customers and accounts is *one-to-many*
  - Students can take multiple courses, and courses can be attended by multiple students
    - The relationship between students and courses is *many-to-many*
  - A car has one engine, and engine belongs to one car
    - The relationship between car and engine is *one-to-one*

RDBMS Database Structure

- Data storage
  - Two-dimensional, columns and rows



The Use of SQL with Databases

- **S**tructured **Q**uery **L**anguage
- Standardized language
- Based on the English language (phrases)
- Categories of phrase types
  - **DDL**
  - **DML**

Data Definition Language (DDL)

- Creation and deletion of database

```
CREATE DATABASE ...
DROP DATABASE ...
```

- Creation and deletion of database table

```
CREATE TABLE ...
DROP TABLE ...
```

- Modification of databases and tables

```
ALTER DATABASE ...
ALTER TABLE ...
```

Data Manipulation Language (DML)

- Retrieve data

```
SELECT ... FROM <table> ...
```

- Add new data

```
INSERT INTO <table> ...
```

- Modify data

```
UPDATE <table> ...
```

- Remove data

```
DELETE FROM <table> ...
```

MySQL Added-Value

- Powerful extension of SQL language
  - Runs on many different operating systems
  - Flexible and secure authorization system
  - Supports very large databases
  - Very customizable
  - Built for speed
  - Free or Inexpensive
  - Open Source and Commercial licenses available
  - Exceptional support available
  - *…and more!*

Chapter Summary

- Define Relational Database (and RDBMS)
- Describe the structure of an RDBMS database
- Explain the use of MySQL with relational databases
- Define and use Data Definition Language
- Define and use Data Manipulation Language

Course Chapter Map

Learning Objectives

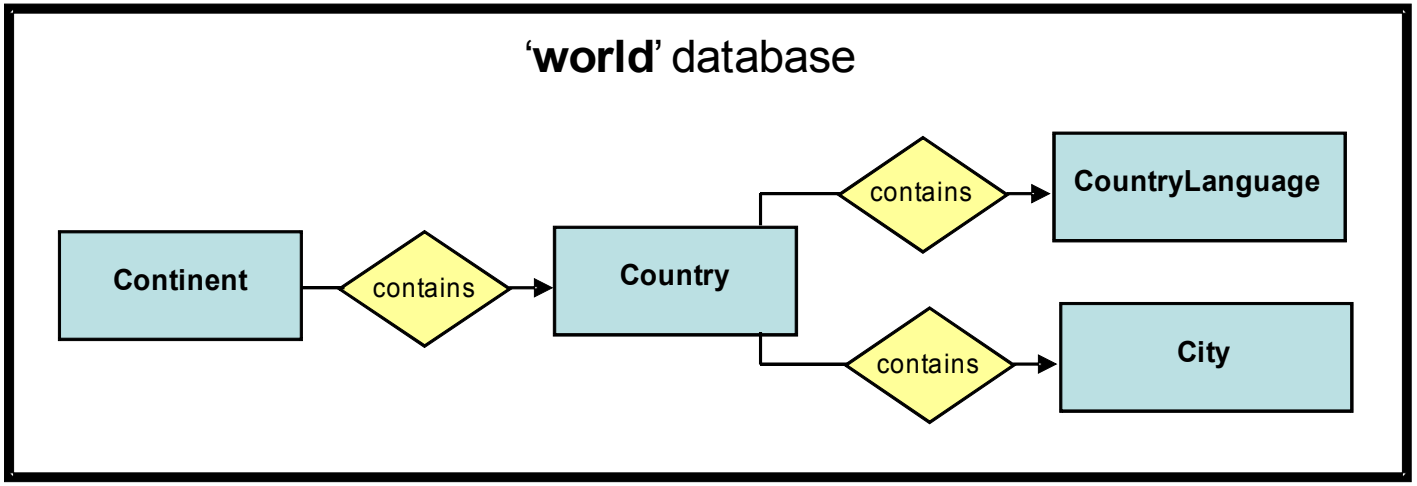
- Explain database modeling
- Describe using keys
- Define and use normalization
- List MySQL accepted data types and their uses
- How to view a database structure
- How to evaluate the database design

Database Modeling

- Prior to building a database some database structure modeling should occur
- Process specifies the structure of the tables based upon specific use of data
- Entity-relational model (ERM) is the most common model for RDBMSs

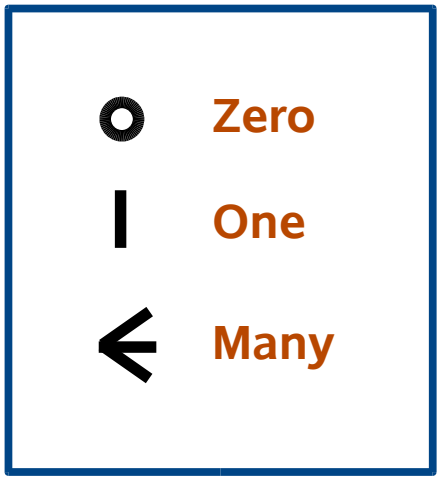
Structure Diagram (ERD)

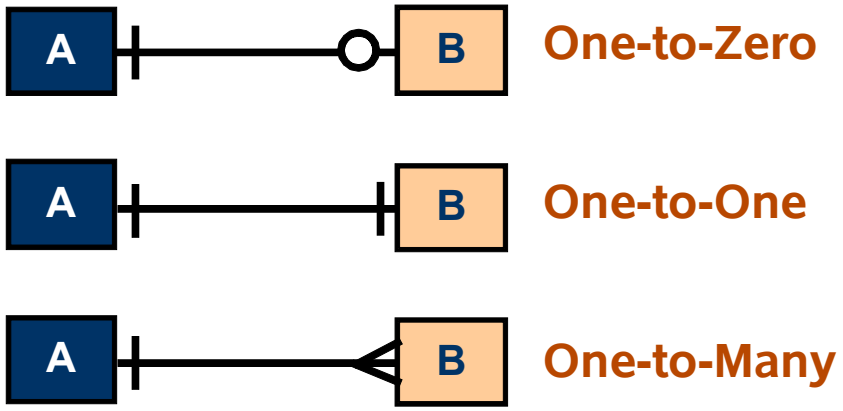
- Used to better visualize the content and hierarchy of a database
  - Organize data into logical groups
- Example of a high-level diagram of the **world** database



Cardinality Diagram (ERD)

- Indicates relationships between tables
- Uses lines and specific symbols:



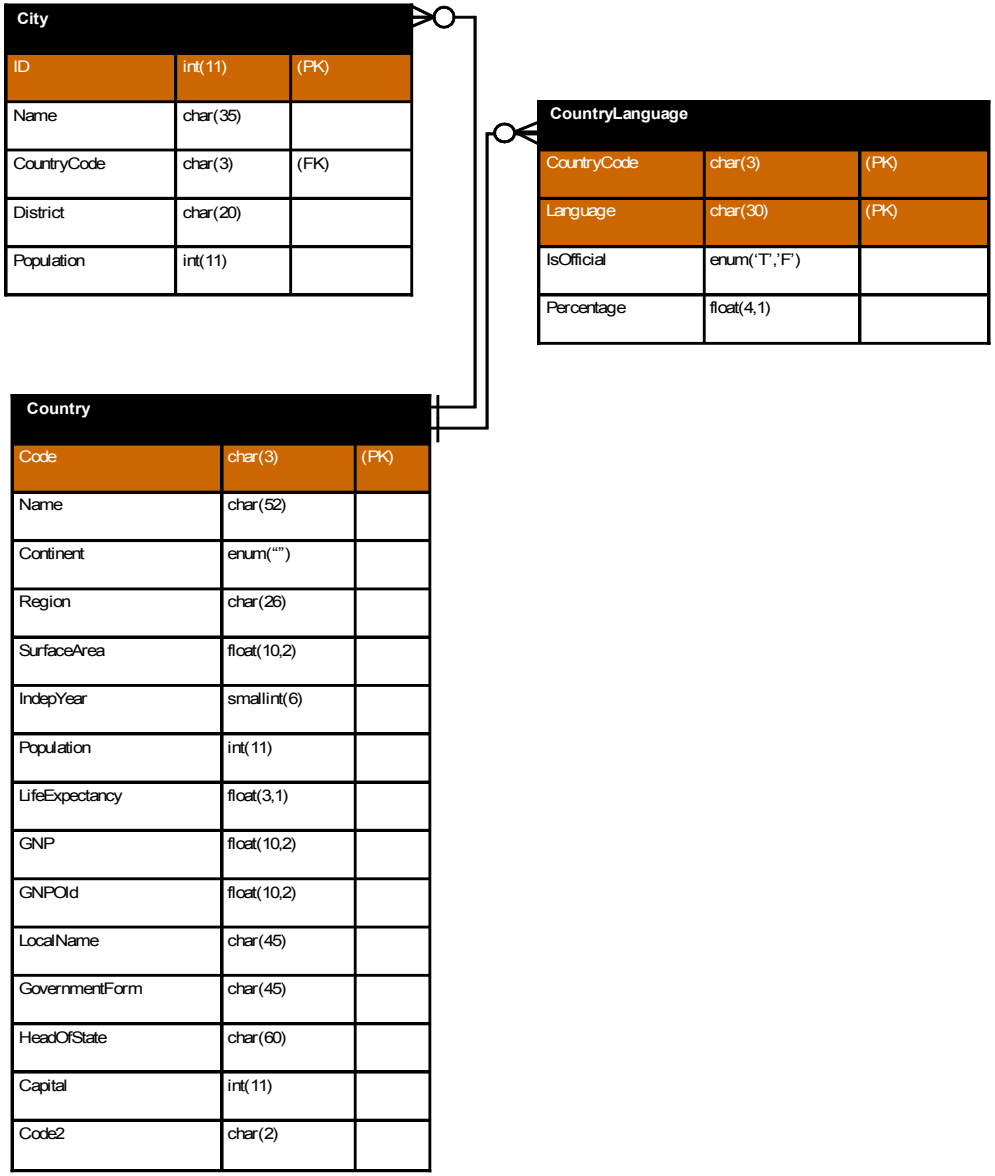


Keys

- Row identification
- Candidate key
  - Column(s) that uniquely identify a row
- Primary key
  - A candidate key that best defines exactly one unique row
  - Ensures that duplicate values do not occur
  - Cannot contain NULL
  - Can decrease lookup time for large databases
- Foreign key
  - References a primary key in another table

Key Example

- '**world**' database



Normalization

- Widely used as a guide in designing relational databases
- Eliminate redundant data
- Eliminate modifications that make the data inconsistent
- Often includes a process of 'decomposition' into a set of smaller tables
  - Removes multi-valued attributes and repeating groups
- Good database design is very important
  - Although SQL usage is very important, it cannot fix a badly designed database

Advantages and Disadvantages of Normalizing

- Advantages
  - ER diagrams
  - Avoid update anomalies
  - Compact
  - Data access
  - Joins
- Disadvantages
  - Numerous tables
  - Maintenance
  - Resource usage

Eliminating Data Inconsistencies

- Anomalies
  - Can occur from poorly designed table structures is duplicate data, redundant data and difficulties in using the data
- Ideal Fields
  - The best way to detect fields that may lead to anomalies is ensuring that each field designed meets the following criteria:
    - Represents a characteristic of the table subject
    - Contains only a single value
    - Should not be de-constructed into smaller components
    - Does not contain a calculated or concatenated value
    - Unique within the entire database structure
    - Identical field types in relationship

Normal Forms

- Many successive levels of normalization
  - Provides stronger guarantees for data updates
- First three are most common
  - **First Normal Form** (**1NF**) -- contains no repeating groups within rows
  - **Second Normal Form** (**2NF**) -- normalized at the first level and every non-key (supporting) column value is dependent on the primary key
    - A non-key value must depend on every column in a *composite* primary key, as well
  - **Third Normal Form** (**3NF**) -- normalized at the first and second level, dependent solely on the primary key and no other non-key (supporting) column value

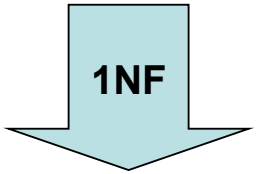
Normalization Process Example

- Furniture store inventory
  - You want to turn your inventory spreadsheet into a database, so you can...
    - do intelligent searches
    - Modifications
    - prepare for future growth of your business.
  - Very simple example to demonstrate normalization process
- Original inventory spread sheet

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

First Normal Form

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



Inventory_1NF					
PK 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 second normal form.

Second Normal Form

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

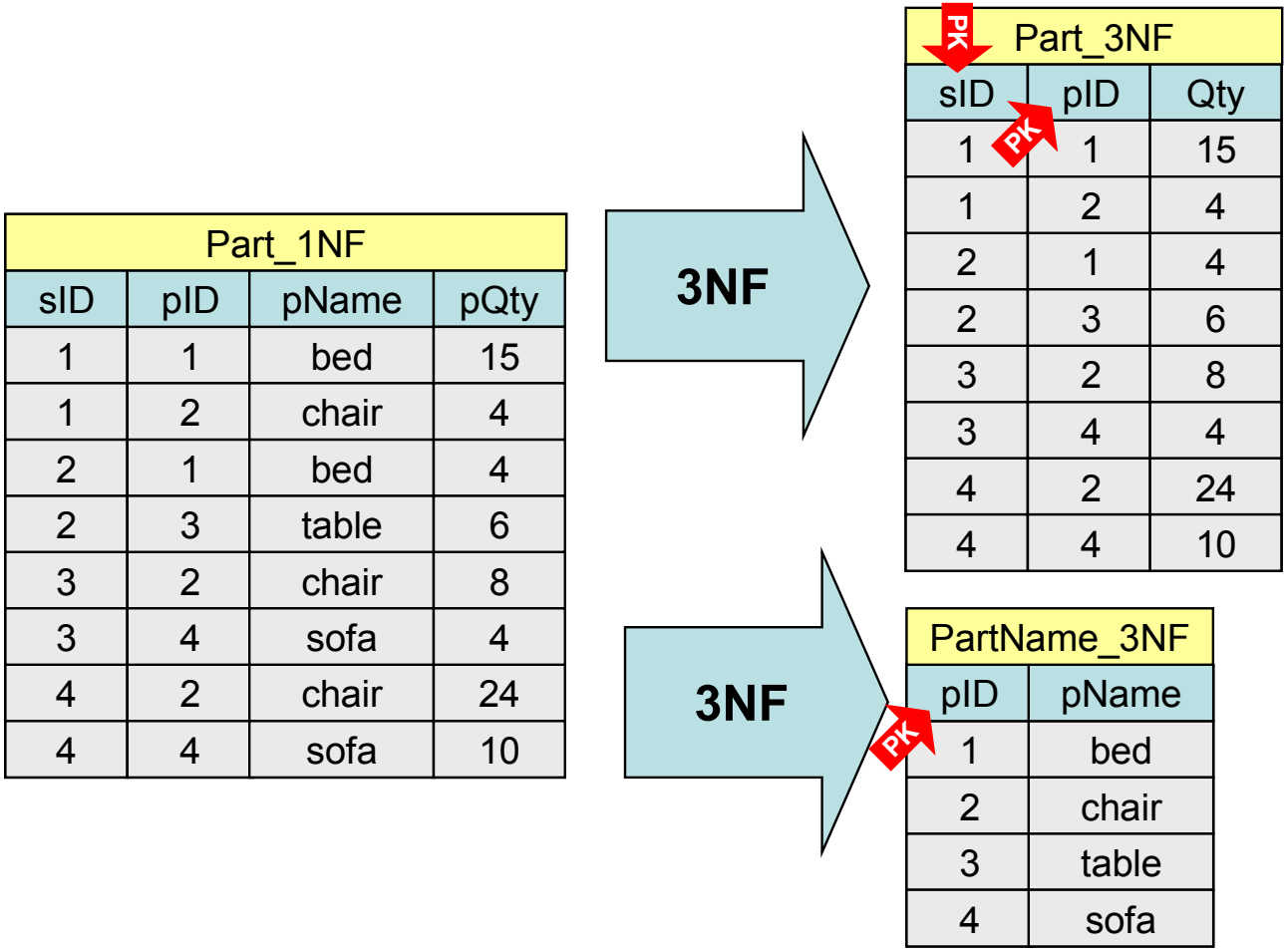
2NF

1NF

Supplier_2NF		
PK sID	sLoc	sPostal
1	Holtsville	00501
2	Waukesha	53146
3	Waukesha	53146
4	Ketchikan	99950

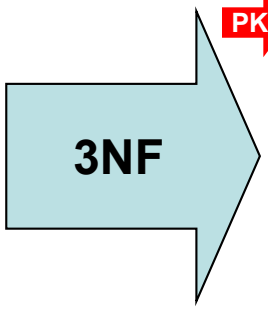
Part_1NF			
PK sID	PK pID	pName	pQty
1	1	bed	15
1	2	chair	4
2	1	bed	4
2	3	table	6
3	2	chair	8
3	4	sofa	4
4	2	chair	24
4	4	sofa	10

Third Normal Form (1/2)

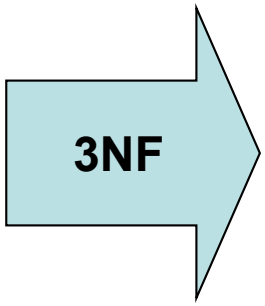


Third Normal Form (2/2)

Supplier_2NF		
sID	sLoc	sPostal
1	Holtsville	00501
2	Waukesha	53146
3	Waukesha	53146
4	Ketchikan	99950



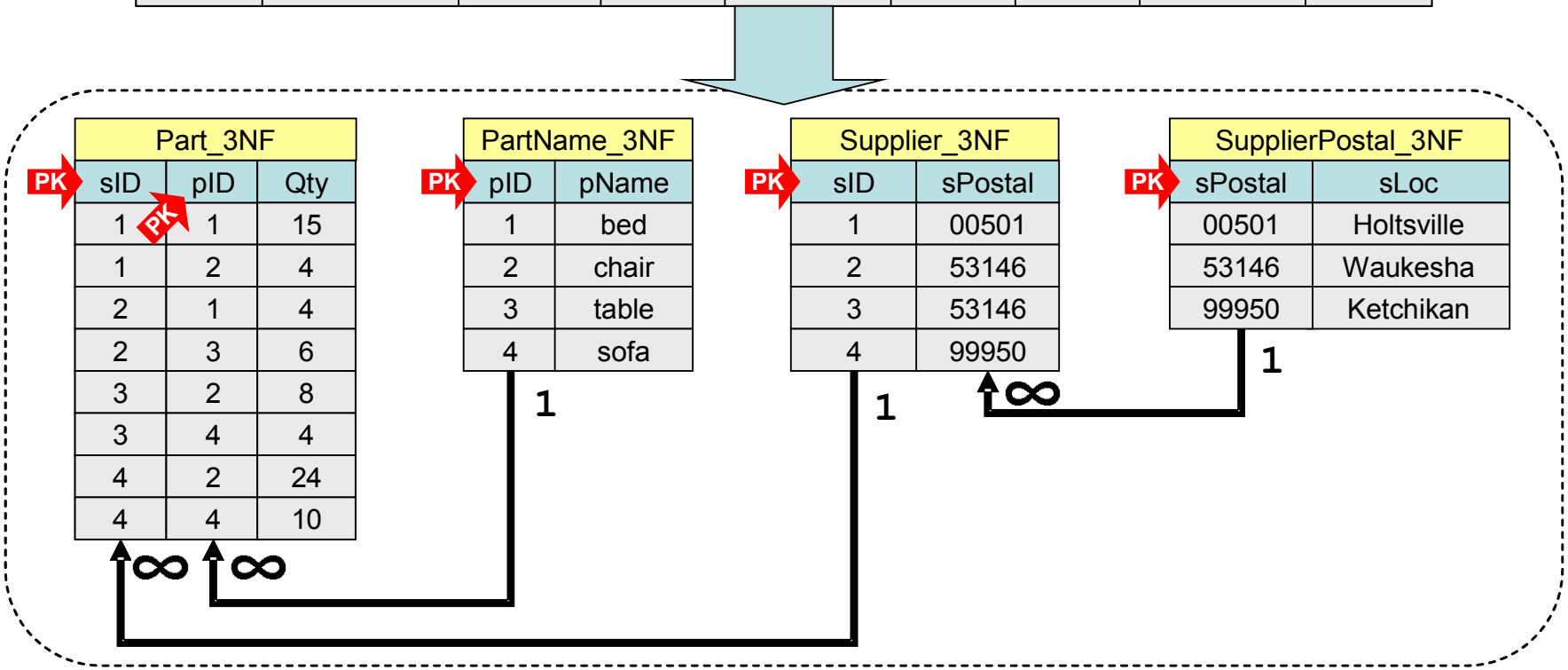
SupplierPostal_3NF	
PK sPostal	sLoc
00501	Holtsville
53146	Waukesha
99950	Ketchikan



PK Supplier_3NF		
sID	sPostal	sName
1	00501	KidzRooms
2	53146	Zurniture, Inc.
3	53146	MyRoom
4	99950	Furventure, Inc.

Normalized 'Furniture Store' 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



Making Intelligent Choices

- Normalization is just common sense
- Put “pen to paper” and document the process
- Eliminate repeating groups
- Make sure each column contains only one value
- Use candidate keys
- Sensible column naming
- Create new tables to better organize (if needed)

Data Types

- Table columns can be one of several different types
  - Depends on the form of the data
  - Example
    - **PartName_3NF** table, the **pName** column values are all *character* data types, whereas the **pID** column values are all *numeric* data types
- Four major categories
  - Numeric
  - Character
  - Binary
  - Temporal
- ABC's of data types
  - **A**pt
  - **B**rief
  - **C**omplete

PROPER DATATYPE USAGE IS VERY IMPORTANT! It can make a big performance difference.

Numeric Data Types

- Store numeric data
  - Integer
  - Floating-Point
  - Fixed-Point
  - BIT
- Factors to consider with Numeric data types
  - Range of values the data type represents
  - Amount of space column values require
  - Column precision and scale (floating-point and fixed-point)
- Whole numbers

Integer Data Types (1/2)

- Types
  - **TINYINT**
  - **SMALLINT**
  - **MEDIUMINT**
  - **INT**
  - **BIGINT**
- Example
  - **World** database, **City** table, **Population** column

```
Population INT(11)
```

  - Largest value output (uses 8, 11 allowed)

```
10500000
```

- **UNSIGNED** allows only positive numbers
- **ZEROFILL** replaces padding spaces with zeros

Integer Data Types (2/2)

- Integer data type comparison

Column Type	Storage	Signed	Unsigned
`TINYINT`	1 byte	-128 to 127	0 to 255
`SMALLINT`	2 bytes	-32,768 to 32,767	0 to 65,535
`MEDIUMINT`	3 bytes	-8,388,608 to 8,388,607	0 to 16,777,215
`INTEGER`	4 bytes	-2,147,483,648 to 2,147,483,647	0 to 4,294,967,295
`BIGINT`	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

Floating-Point Types

- Used for approximate-value numbers
  - Integer, Fractional or both
- Types
  - **FLOAT**
  - **DOUBLE**
- May declare with precision and scale
- Example
  - **World** database, **Country** table, **GNP** column

```
GNP FLOAT(10,2)
```

  - Largest value output (uses 7, 10 allowed; 2 to right of decimal)

```
8510700.00
```

Syntax:

```
FLOAT(M,D)
```

M = maximum decimal digits
D = digits after the decimal

Floating-Point Types Summary

Type	Storage	Signed Range	Unsigned Range
`FLOAT`	4 bytes	-3.402823466E+38 to -1.175494351E-38	0 and 1.175494351E-38 to 3.402823466E+38
`DOUBLE`	8 bytes	-1.7976931348623157E+308 to -2.2250738585072014E-308	0 and 2.2250738585072014E-308 to 1.7976931348623157E+308

Fixed-Point Types

- Exact-value numbers
  - Integer, Fractional or both
- Types
  - **DECIMAL**
  - **NUMERIC**
- Example
  - To represent currency values such as dollars and cents

```
cost DECIMAL(10,2)
```

  - Example value output

```
650.88
```

Syntax:

```
DECIMAL(M,D)
```

M = maximum significant digits
D = digits after the decimal

Bit Types

- 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

Syntax:

```
BIT(M)
```

Rule of thumb:
`n=8 -> 1 Byte`

Temporal Data Types

- TIME
  - HH:MM:SS > `12:59:02`
- YEAR
  - Two or Four digit > `2006`
- DATE
  - YYYY-MM-DD > `2006-08-04`
- DATETIME
  - YYYY-MM-DD HH:MM:SS > `2006-08-04 12:59:02`
- TIMESTAMP > `2006-08-04 12:59:02`
  - **DEFAULT CURRENT_TIMESTAMP**

Temporal Data Type Comparison

Type	Storage	Range
`DATE`	3 bytes	1000-01-01 to 9999-12-31
`TIME`	3 bytes	-838:59:59 to 838:59:59
`DATETIME`	8 bytes	1000-01-01 00:00:00 to 9999-12-31 23:59:59
`TIMESTAMP`	4 bytes	1970-01-01 00:00:00 to mid-year 2037
`YEAR` `YEAR(4)`	1 byte	1901 to 2155 (for YEAR(4))
`YEAR(2)`	1 byte	1970 to 2069 (for YEAR(2))

Character String Data Types (1/2)

- Used for storing character strings
  - Sequence of characters
  - A character is a position in a character set (or alphabet)
- Available string data types

Category	Type
Text	`CHAR`
(comparison values)	`VARCHAR`
	`TINYTEXT`
	`TEXT`
	`MEDIUMTEXT`
	`LONGTEXT`
Integer	`ENUM`
(comparison values)	`SET`

Character String Data Types (2/2)

- Character sets
  - Every character has an associated symbol, per character set
  - Positions in the character set dictate sorting order
  - Column definitions can use a **CHARSET** attribute
- Collation
  - MySQL supports several alternative sorting orders (collations)
  - String operations are affected by collation
  - Column definitions can use a **COLLATE** attribute
- Example

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

Text Category Data Types (1/2)

- **CHAR**
  - Fixed-length string, always right-padded with specified spaces
  - `CHAR(M)`
- **VARCHAR**
  - Variable-length string, stores the string value and length together
  - `VARCHAR(M)`
- Example
  - **World** database, **CountryLanguage** table, **Language** column

```
Language CHAR(30)
```

  - Largest value output (uses 25, 30 allowed)

```
Southern Slavic Languages
```

Text Category Data Types (2/2)

- **TINYTEXT**
  - Variable-length
  - `TINYTEXT`
- **TEXT**
  - Variable-length
  - `TEXT`
- **MEDIUMTEXT**
  - Variable-length
  - `MEDIUMTEXT`
- **MEDIUMTEXT**
  - Variable-length
  - `LONGTEXT`

Text Category Comparison

Type	Storage Required	Maximum Length
`CHAR(M)`	`M` characters	255 characters
`VARCHAR(M)`	`L` characters plus 1 or 2 bytes	65,535 characters (subject to limitations)
`TINYTEXT`	`L` + 1 bytes, where `L` < 2^8	255 bytes
`TEXT`	`L` + 2 bytes, where `L` < 2^{16}	65,535 bytes
`MEDIUMTEXT`	`L` + 3 bytes, where `L` < 2^{24}	16,777,215 bytes
`LONGTEXT`	`L` + 4 bytes, where `L` < 2^{32}	4,294,967,295 bytes

Integer Category Data Types

- **ENUM**
  - `ENUM` – Enumeration
  - Example
    - **World** database, **Country** table, **Continent** column

```
Continent ENUM('Asia', 'Europe', 'North America',
'Africa', 'Oceana', 'Antarctica', 'South America')
```

    - Largest value output

```
North America, Asia, Africa, Europe, South America,
Oceania, Antarctica
```

- **SET**
  - `SET` – List of string values
  - Example
    - **Allergy** table, **Symptom** column

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

Binary String Data Types

- Sequence of bytes
  - Binary digits (Bits) grouped in eights (octets)
- Example binary uses
  - Compiled computer programs/applications
  - Image and sound files
- Binary types
  - `BINARY(M)`
  - `VARBINARY(M)`
  - `TINYBLOB`
  - `BLOB`
  - `MEDIUMBLOB`
  - `LONGBLOB`

Binary String Type Comparison

Type	Storage	Maximum Length
`BINARY(M)`	*M* bytes	255 bytes
`VARBINARY(M)`	*M* bytes plus 1 or 2 bytes	65,535 bytes (subject to limitations)
`TINYBLOB`	*L* + 1 bytes	255 bytes
`BLOB`	*L* + 2 bytes	65,535 bytes
`MEDIUMBLOB`	*L* + 3 bytes	16,777,215 bytes
`LONGBLOB`	*L* + 4 bytes	4,294,967,295 bytes

Data Type Considerations

- Data storage and retrieval is a large issue
- Use appropriate data type
- Use appropriate integer length
- Use appropriate character length

The Meaning of NULL

- Can set data types to allow missing values
- Can be an empty query result
- Conceptually has several meanings
  - “no value”
  - “unknown value”
  - “missing value”
  - “out of range”
  - “not applicable”
- Two categories
  - Unknown
  - Not Applicable

When to Use NULL

- During database design
- Cases where column information is not available
  - Determine whether **NULL** values can be allowed
- Can also be changed in an existing table
- Example from **world**
  - **Country** table contains countries with no value for this column

```
`LifeExpectancy` float(3,1) DEFAULT NULL
```

- Use **NOT NULL** if column will not hold nulls
  - Ensures integrity of the data

Viewing a Database in 5 Easy Steps

- SQL statements for viewing a database

1) `SHOW DATABASES`
2) `USE <database_name>`
3) `SHOW TABLES`
4) `DESCRIBE <table_name>`
5) `SELECT * FROM <table_name>`

Evaluating a Design (1/3)

- Use ‘5 Easy Steps’ to view database structure

Step 1

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| world              |
+--------------------+
```

--->

Step 2

```
mysql> USE world;
Database changed
```

Evaluating a Design (2/3)

Step 3

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

Step 4

```
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       |                |
+-------------+----------+------+-----+---------+----------------+
```

Evaluating a Design (3/3)

Step 5

```
mysql> SELECT * FROM City;
+------+---------------+-------------+---------------+------------+
| ID   | Name          | CountryCode | District      | Population |
+------+---------------+-------------+---------------+------------+
|    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 |
...
| 4079 | Rafah         | PSE         | Rafah         |      92020 |
+------+---------------+-------------+---------------+------------+
```

Chapter Summary

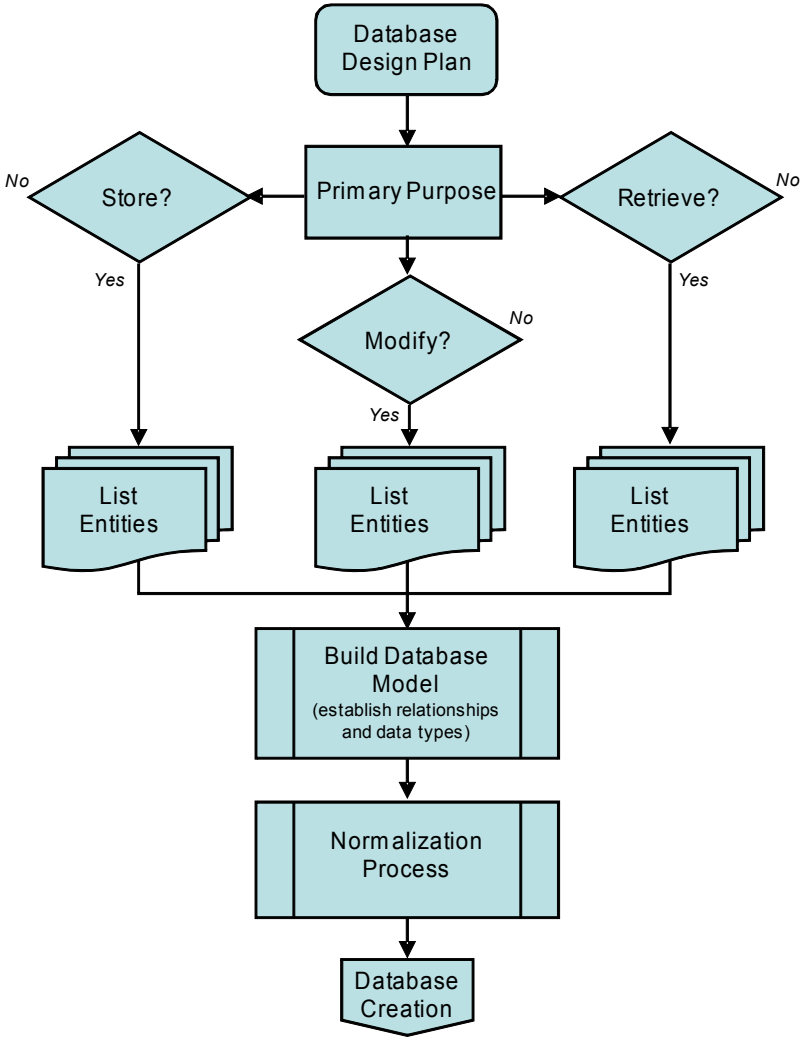
- Explain database modeling
- Describe using keys
- Define and use normalization
- List MySQL accepted data types and their uses
- How to view a database structure
- How to evaluate the database design

Course Chapter Map

Learning Objectives

- Prepare a design plan for a new database
- Define columns for a new database
- Use the **CREATE DATABASE** statement
- Describe and set column and table options
- Use the **SHOW CREATE DATABASE** statement
- Describe and use indexing and constraints
- Use the **SHOW INDEX** statement
- Use the **CREATE TABLE** statement to add tables to a database

Database Design Plan



Creating a Database

- Create database using SQL DDL statement

```
mysql> CREATE DATABASE mydatabase
```

… the response will look similar to the following…

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

- MySQL naming conventions
  - Generally not case-sensitive
  - Some characters cannot be used
    - ASCII(**0**), ASCII(**255**), **/**, ****, and **.**
  - Names cannot have more than 64 characters
  - Reserved words and special characters can be used in quotes

Creating a Table

- General syntax for creating a table

```
CREATE TABLE <table_name> (
    <column name> <column type> [<column options>]
    [, <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)
    );
```

SHOW CREATE TABLE

- Viewing 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=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Column Options

- Table must have at least one column
- Add options to **CREATE TABLE** statement
- Several options available
  - **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(Country, Language)
    )
```

Table Options

- Add options to **CREATE TABLE** statement
- Several options available
  - **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(Country, Language)
    )ENGINE = MyISAM COMMENT 'Lists Language Spoken'
```

Table Indexing

- Original data not put in any specific order/location
  - Server must scan *all* data to find relevant rows
- Indexes assist in finding rows quickly and easily
  - Only contain column info for locating rows
- Assign Indexes when creating a table, or add later
- Composite indexes
  - Multiple columns

MySQL Indexes

- Commonly used indexes
  - **PRIMARY KEY**
  - **UNIQUE**

Most indexes are Stored in B-trees.

- Indexes are optional and must be created by 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(Country, Language)
    )ENGINE = MyISAM COMMENT 'Lists Language Spoken'
```

SHOW INDEX (1/2)

- View status of indexes in a table
- General syntax

```
SHOW INDEX FROM <table>
```

SHOW INDEX (2/2)

- 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
      Comment:
```

Table Constraints

- A restriction placed on one or more column values of a table
  - Actively enforces integrity rules
- Implemented using indexes
- Types of constraints
  - **PRIMARY KEY**
  - **FOREIGN KEY**
  - **UNIQUE**
- MySQL generates indexes to enforce above constraints

Further Practice: Chapter 5

- Comprehensive exercises

Chapter Summary

- Prepare a design plan for a new database
- Define columns for a new database
- Use the **CREATE DATABASE** statement
- Describe and set column and table options
- Use the **SHOW CREATE DATABASE** statement
- Describe and use indexing and constraints
- Use the **SHOW INDEX** statement
- Use the **CREATE TABLE** statement to add tables to a database

Course Chapter Map

Learning Objectives

- Retrieve database data using the **SELECT** query statement
- Use **SELECT** statement clauses: **FROM**, **WHERE**, **LIMIT**, **DISTINCT**, **ORDER BY**
- Trouble-shoot problems using warning and error messages
- Describe and use SQL Modes to change interpretation of SQL syntax

The SELECT Statement (1/2)

- Most commonly used command for queries
- Retrieves rows from tables in a database
- Returns rows in a tabular form called a ‘result set’
- Simple syntax

```
SELECT [<clause options>] <column list>
[FROM <table_name>] [<other_clauses>]
```

- The *column_list* is a list of column names that make up the result set

The SELECT Statement (2/2)

- Example

```
mysql> SELECT Name FROM Country;
+-----------------------------------+
| Name                              |
+-----------------------------------+
| Afghanistan                       |
| Netherlands                       |
| Netherlands Antilles              |
| Albania                           |
...
| French Southern Territories       |
| Unites States Minor Outlying Islands |
+-----------------------------------+
239 rows in set (0.02 sec)
```

Basic Uses of SELECT

- Clauses used to yield specific results
  - **DISTINCT**
  - **FROM**
  - **WHERE**
  - **ORDER BY**
  - **LIMIT**
- Syntax example

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

SELECT Tips

- **Best to treat table/databases names as case-sensitive**
- **Use \c to abort a statement**
- **Use \G in place of the ;) to return results by the row**
- **Use of * (all row data) can give random results and waste resources**

SELECT with DISTINCT (1/2)

- Removes duplicate rows
- Example

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

--->

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

SELECT with DISTINCT (2/2)

- Treats **NULL** as the same
- Example

```
mysql> SELECT i, j FROM t;
+-------------+
| i     | j     |
+-------------+
|     1 |     2 |
|     1 | NULL  |
|     1 | NULL  |
+-------------+
```

--->

```
mysql> SELECT DISTINCT i, j FROM t;
+-------------+
| i     | j     |
+-------------+
|     1 |     2 |
|     1 | NULL  |
+-------------+
```

SELECT with WHERE (1/4)

- Filters out unwanted rows
- Specify column values
- Example

```
mysql> SELECT ID, Name, District FROM City
    -> WHERE Name = 'New York';
+------+----------+----------+
| ID   | Name     | District |
+------+----------+----------+
| 3793 | New York | New York |
+------+----------+----------+
1 row in set (0.00 sec)
```

SELECT with WHERE (2/4)

- Operators used with WHERE
  - Arithmetic
  - Comparison
  - Logical
- Arithmetic
  - **+**, **-**, *****, **/**, **DIV**, **%**
- Comparison
  - **<**, **<=**, **=**, **<=>**, **<> or !=**, **>=**, **>**, **BETWEEN**
- Logical
  - **AND**, **OR**, **XOR**, **NOT**
- Additional options
  - **IN**, **IS NULL**, **LIKE**, **etc.**

SELECT with WHERE (3/4)

- Example

```
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)
```

SELECT with WHERE (4/4)

- Example

```
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 |
| Dominica            | North America |
...
| Belarus             | Europe        |
| Estonia             | Europe        |
+---------------------+---------------+
35 rows in set (0.00 sec)
```

SELECT with ORDER BY (1/3)

- Will return output rows in a specific order
- Example

```
mysql> SELECT Name, Continent FROM Country
    -> WHERE GovernmentForm = 'Republic'
    -> AND (Continent='North America'
        OR Continent='Europe')
    -> ORDER BY Continent;
+--------------------+---------------+
| Name               | Continent     |
+--------------------+---------------+
| Albania            | Europe        |
| Malta              | Europe        |
| Moldova            | Europe        |
...
| Nicaragua          | North America |
| Panama             | North America |
| Honduras           | North America |
+--------------------+---------------+
35 rows in set (0.00 sec)
```

SELECT with ORDER BY (2/3)

- Ascending order is default
- Specify order with **ASC** and **DESC**
- Example

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

SELECT with ORDER BY (3/3)

- Sort multiple columns simultaneously
- Example

```
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)
```

SELECT with LIMIT (1/3)

- Specify number of rows output
- Example

```
mysql> SELECT Name FROM Country LIMIT 8;
+----------------------+
| name                 |
+----------------------+
| Afghanistan          |
| Netherlands          |
| Netherlands Antilles |
| Albania              |
| Algeria              |
| American Samoa       |
| Andorra              |
| Angola               |
+----------------------+
8 rows in set (0.00 sec)
```

SELECT with LIMIT (2/3)

- Specify skip rows
- Example

```
mysql> SELECT name, population FROM country LIMIT 20,8;
+--------------------------+------------+
| name                     | population |
+--------------------------+------------+
| Belgium                  |   10239000 |
| Belize                   |     241000 |
| Benin                    |    6097000 |
| Bermuda                  |      65000 |
| Bhutan                   |    2124000 |
| Bolivia                  |    8329000 |
| Bosnia and Herzegovina   |    3972000 |
| Botswana                 |    1622000 |
+--------------------------+------------+
8 rows in set (0.00 sec)
```

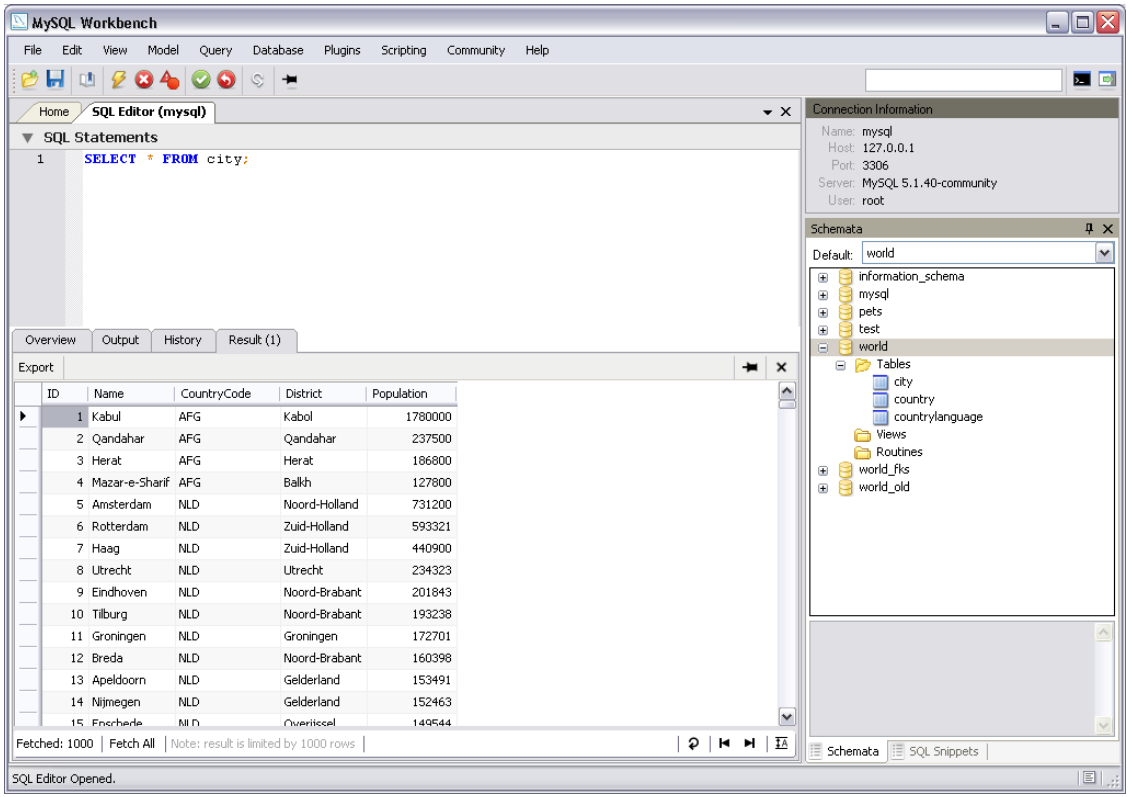
SELECT with LIMIT (3/3)

- Use with **ORDER BY** for ordered output
- 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)
```

MySQL Workbench for SQL Development

- GUI used to query and analyze data on MySQL server
- Download from MySQL website
- Free and available for Windows, Linux and Mac OS



Warnings and Errors (1/2)

- Many resources to assist with query problems
- Example

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

- Three components of error message
- MySQL Reference Manual
  - http://mysql.com/<*keyword*>

Warnings and Errors (2/2)

- Statements for viewing warnings and errors
  - **SHOW WARNINGS**
  - **SHOW ERRORS**
- Reference manual error documentation
- Correct errors and run SQL statement again

SQL Modes for Syntax Checking

- Customize server operating mode

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

- View current SQL mode value

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

- SQL mode values
  - **ANSI**
  - **STRICY_TRANS_TABLES, STRICT_ALL_TABLES**
  - **TRADITIONAL**

Further Practice: Chapter 6

- Comprehensive exercises

Chapter Summary

- Retrieve database data using the **SELECT** query statement
- Use **SELECT** statement clauses: **FROM**, **WHERE**, **LIMIT**, **DISTINCT**, **ORDER BY**
- Trouble-shoot problems using warning and error messages
- Describe and use SQL Modes to change interpretation of SQL syntax

Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Use the **DROP DATABASE** statement to delete a database
- Discuss the issues related to dropping a database
- Add a new table to a database using **SELECT**
- Delete an entire table
- Add and remove columns of a table
- Modify table columns
- Add and remove indexes and constraints

The DROP DATABASE Statement

- Removes a database
- Full or empty database
- Returns row count which is number of tables deleted
- Examples

```
DROP DATABASE mydb

DROP DATABASE IF EXISTS mydb
```

DROP DATABASE has no UNDO feature, so be cautious when deleting an entire database!

Adding a Table From an Existing Table (1/2)

- Creating from values of an existing table
- View original table using **SHOW CREATE TABLE**
- Use **SELECT** statement with **CREATE TABLE**
- Examples

```
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
```

Adding a Table From an Existing Table (2/2)

- New table added

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

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

```
mysql> SELECT * FROM EU_Countries;
+-------------------------------+---------------+
| Name                          | NewPopulation |
+-------------------------------+---------------+
| Netherlands                   |    23796000.0 |
| Albania                       |     5101800.0 |
| Andorra                       |      117000.0 |
| Belgium                       |    15358500.0 |
| Bosnia and Herzegovina        |     5958000.0 |
| United Kingdom                |    89435100.0 |
...
| Russian Federation            |   220401000.0 |
| Estonia                       |     2158800.0 |
+-------------------------------+---------------+
```

Creating a Temporary Table

- **CREATE TEMPORARY TABLE**
  - Table kept until the client disconnects
  - Great way to work with summary data
  - Only visible to the client that created it
  - Does not conflict with other hosts using same data
  - Overrides existing table until the connection dropped
- Examples

```
mysql> CREATE TEMPORARY TABLE EU_Countries_TEMP
    -> 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
```

Copying an Existing Table Structure

- Copying a table and it's structure
  - Indexes
  - Column options
  - No row data
- Examples

```
CREATE TABLE new_table LIKE old_table
```

The DROP DATABASE Statement

- Remove a table
- Full or empty table
- **IF EXISTS** to avoid error
- **DROP TEMPORARY TABLE**
- Examples

DROP TABLE has no UNDO feature, so be cautious when deleting an entire table!

```
DROP TABLE table1
DROP TABLE IF EXISTS table1
DROP TEMPORARY TABLE EU_Countries_TEMP
```

Adding a Column (1/2)

- **ALTER TABLE** statement with **ADD COLUMN**
- Examples

```
ALTER TABLE EU_Countries ADD COLUMN Id INT NOT NULL
```

- Structure change

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

Adding a Column (2/2)

- Review row contents

```
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 |
...
```

Removing a Column

- **ALTER TABLE** statement with **DROP COLUMN**
- Examples

```
ALTER TABLE EU_Countries DROP COLUMN Id
```

- Structure change

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

Modifying Columns (1/2)

- **ALTER TABLE** statement with **MODIFY COLUMN**
- Examples

```
ALTER TABLE EU_Countries MODIFY COLUMN NewPopulation
INT UNSIGNED NOT NULL
```

- Structure change

```
+---------------+------------------+------+-----+---------+-------+
| Field         | Type             | Null | Key | Default | Extra |
+---------------+------------------+------+-----+---------+-------+
| Name          | char(52)         | NO   |     |         |       |
| NewPopulation | int(10) unsigned | NO   |     |         |       |
+---------------+------------------+------+-----+---------+-------+
```

Modifying Columns (2/2)

- Review row contents

```
+--------------------------+---------------+
| name                     | NewPopulation |
+--------------------------+---------------+
| Albania                  |       5101800 |
| Andorra                  |        117000 |
| Austria                  |      12137700 |
| Belgium                  |      15358500 |
| Bulgaria                 |      12286350 |
| Bosnia and Herzegovina   |       5958000 |
...
```

Adding Index Constraints (1/2)

- **ALTER TABLE** statement with **ADD INDEX**
- 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)
```

- Example

```
ALTER TABLE City ADD INDEX Pop (Population)
```

Adding Index Constraints (2/2)

- Use **SHOW CREATE TABLE** to confirm alteration

```
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 `Pop` (`Population`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Dropping Index Constraints

- **ALTER TABLE** statement with **DROP INDEX**

```
ALTER TABLE City DROP INDEX Pop
```

- Confirm drop

```
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',
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Further Practice: Chapter 7

- Comprehensive exercises

Chapter Summary

- Use the **DROP DATABASE** statement to delete a database
- Discuss the issues related to dropping a database
- Add a new table to a database using **SELECT**
- Delete an entire table
- Add and remove columns of a table
- Modify table columns
- Add and remove indexes and constraints

Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Delete and modify table data
- Use the **INSERT** and **REPLACE** statements to add new data to a table
- Use the **UPDATE** statement to modify data in a table
- Use the **DELETE** statement to remove table rows

Delete/Modify Table Row Data

- Row data may require changes

SAFETY TIPS

Database manipulation can be tricky!

- **Administrators should restrict access to only required users**
- **Keep up-to-date backups**
- **Make ad hoc back-ups as needed**
- **Set a "Safe Updates" option**
- **Always think in terms of rows**
  - **entire row associated with data will be affected**
- **Verify that the WHERE criteria returns the right rows**

The INSERT Statement (1/3)

- Populate tables with data
- General syntax
  - **INTO** clause

```
INSERT INTO table_name (<column_list>) VALUES (<value_list>)
```

- Single row example

```
INSERT INTO CountryLanguage (CountryCode, Language,
   Percentage) VALUES ('ALB', 'Chinese', 20);
```

- Row contents

```
+-------------+-----------+-----------+------------+
| CountryCode | Language  | IsOfficial | Percentage |
+-------------+-----------+-----------+------------+
...
| ALB         | Chinese   | F          |       20.0 |
...
```

The INSERT Statement (2/3)

- Multi-row example

```
INSERT INTO CountryLanguage (CountryCode, Language)
    VALUES ('GRL', 'MySQL'), ('FJI', 'MySQL'), ('BEL', 'MySQL');
```

- Row contents

```
+-------------+----------+------------+------------+
| CountryCode | Language | IsOfficial | Percentage |
+-------------+----------+------------+------------+
| BEL         | MySQL    | F          |        0.0 |
| FJI         | MySQL    | F          |        0.0 |
| GRL         | MySQL    | F          |        0.0 |
+-------------+----------+------------+------------+
```

Remember:
String and Temporal Data Types must be enclosed in single quotes.

The INSERT Statement (3/3)

- **NULL** condition
  - If values are *not* specified and columns have no default value...
    - If column accepts **NULL**, set to **NULL**
    - If not, set to implicit default
    - **AUTO_INCREMENT** can accept **NULL**

The REPLACE Statement (1/2)

- Exactly the same as **INSERT**
  - Except when it is a **PRIMARY KEY** or **UNIQUE** constraint
- MySQL extension
- General syntax
  - **INTO** clause

```
REPLACE INTO table_name (<column_list>) VALUES (<value_list>)
```

- Example

```
REPLACE INTO CountryLanguage (CountryCode, Language,
Percentage) VALUES ('ALB', 'Albaniana', 78.1);
```

REPLACE is only useful when table has a PRIMARY KEY or UNIQUE constraint

The REPLACE Statement (2/2)

- Returns sum of rows deleted and inserted
- **REPLACE** algorithm
  - Try to insert the new row into the table
  - When insertion fails due to an error, delete conflicting row then redo insert

The UPDATE Statement (1/4)

- Modifies contents of existing rows
- Use with **SET** clause for column assignments
- 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 = 'SEI';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings:0
```

The UPDATE Statement (2/4)

- Updates can have no effect
  - Matches no rows
  - No change to column values
- Differences between **UPDATE** and **REPLACE**
  - **UPDATE** requires existing row, **REPLACE** does not
  - **UPDATE** can change existing row, **REPLACE** discards row

The UPDATE Statement (3/4)

- Inconsistent ordering by default
  - Can cause errors

```
mysql> UPDATE City SET ID = ID+1;
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

- Use **ORDER BY** to control order

```
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
```

- Row contents

```
+----+---------------+-------------+---------------+------------+
| ID | Name          | CountryCode | District      | Population |
+----+---------------+-------------+---------------+------------+
|  2 | Kabul         | AFG         | Kabol         |    1780000 |
|  3 | Qandahar      | AFG         | Qandahar      |     237500 |
|  4 | Herat         | AFG         | Herat         |     186800 |
...
```

The UPDATE Statement (4/4)

- Use **LIMIT** to control number of rows updated

```
UPDATE City SET ID = ID+1 LIMIT 1;
```

- Row contents

```
+----+----------------+-------------+---------------+------------+
| ID | Name           | CountryCode | District      | Population |
+----+----------------+-------------+---------------+------------+
|  1 | Kabul          | AFG         | Kabol         |    1780000 |
|  3 | Qandahar       | AFG         | Qandahar      |     237500 |
|  4 | Herat          | AFG         | Herat         |     186800 |
...
```

- **UPDATE** can include **ORDER BY** used *with* **LIMIT**

The DELETE Statement (1/2)

- Removes specified table rows
- General syntax

```
DELETE FROM tbl_name
   [WHERE where_condition]
   [ORDER BY ...]
   [LIMIT row_count]
```

- Use **WHERE** to indicate which rows to remove
- Do ***not*** indicate columns
- Removing ***all*** table rows

```
DELETE FROM tbl_name
```

DELETE FROM has no UNDO feature, so be cautious when deleting an entire table of rows!

The DELETE Statement (2/2)

- Use **ORDER BY** and **LIMIT** to remove specific rows
  - Example with **DESC**

```
DELETE FROM CountryLanguage WHERE Language = 'MySQL'
  ORDER BY CountryCode DESC LIMIT 1;
...
+-------------+----------+------------+------------+
| CountryCode | Language | IsOfficial | Percentage |
+-------------+----------+------------+------------+
| BEL         | MySQL    | F          |        0.0 |
| FJI         | MySQL    | F          |        0.0 |
| GRL         | MySQL    | F          |        0.0 |
+-------------+----------+------------+------------+
```

< *Removed*

  - Example with **ASC**

```
DELETE FROM CountryLanguage WHERE Language = 'MySQL'
  ORDER BY CountryCode ASC LIMIT 1;
...
| BEL         | MySQL    | F          |        0.0 |
| FJI         | MySQL    | F          |        0.0 |
...
```

< *Removed*

Further Practice: Chapter 8

- Comprehensive exercises

Chapter Summary

- Delete and modify table data
- Use the **INSERT** and **REPLACE** statements to add new data to a table
- Use the **UPDATE** statement to modify data in a table
- Use the **DELETE** statement to remove table rows

Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Delete and modify table data
- Use the **INSERT** and **REPLACE** statements to add new data to a table
- Use the **UPDATE** statement to modify data in a table
- Use the **DELETE** statement to remove table rows

Functions in MySQL Expressions

- Several categories of functions
- Invoked within an expression
- General function syntax

```
function_name(<arg1> [, <arg2>, …, <argn>])
```

- Examples

```
SELECT NOW()

SELECT VERSION()
```

Function Tips

- **Functions can be used anywhere a value expression is accepted**
- **Columns can be used as arguments with the correct data type**
- **A function output can be the input of another function**
- **An expression with NULL always produces a NULL**
- **Mathematical functions will return NULL on error (i.e., division by zero)**

String Functions

- Perform operations on strings
- Two categories
  - Returning numbers
  - Returning strings

Returning Numbers

- Examples

```
mysql> SELECT CHAR_LENGTH('MySQL');
+----------------------+
| CHAR_LENGTH('MySQL') |
+----------------------+
|                    5 |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT INSTR('MySQL', 'SQL');
+----------------------+
| INSTR('MySQL', 'SQL') |
+----------------------+
|                    3 |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT STRCMP('abc','def'),
    -> STRCMP('def','def'),
    -> STRCMP('def','abc');
+---------------------+---------------------+---------------------+
| STRCMP('abc','def') | STRCMP('def','def') | STRCMP('def','abc') |
+---------------------+---------------------+---------------------+
|                  -1 |                   0 |                   1 |
+---------------------+---------------------+---------------------+
1 row in set (0.00 sec)
```

Returning Strings (1/5)

- Examples

```
mysql> SELECT CONCAT('A', '-', 'Z');
+----------------------+
| CONCAT('A', '-', 'Z') |
+----------------------+
| A-Z                  |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT RIGHT('MySQL', 3);
+------------------+
| RIGHT('MySQL', 3) |
+------------------+
| SQL              |
+------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT REVERSE('MySQL');
+-----------------+
| REVERSE('MySQL') |
+-----------------+
| LQSyM           |
+-----------------+
1 row in set (0.01 sec)
```

```
mysql> SELECT LEFT('MySQL', 3);
+-----------------+
| LEFT ('MySQL', 3) |
+-----------------+
| MyS             |
+-----------------+
1 row in set (0.00 sec)
```

Returning Strings (2/5)

- Examples *(continued)*

```
mysql> SELECT LOWER('MySQL');
+----------------+
| LOWER('MySQL') |
+----------------+
| mysql          |
+----------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT UPPER('MySQL');
+----------------+
| UPPER('MySQL') |
+----------------+
| MYSQL          |
+----------------+
1 row in set (0.01 sec)
```

```
mysql> SELECT LPAD('MySQL', 15, '.');
+-----------------------+
| LPAD('MySQL', 15, '.') |
+-----------------------+
| .........MySQL        |
+-----------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT RPAD('MySQL', 14,
    '_.');
+-----------------------+
| RPAD('MySQL', 14, '_.') |
+-----------------------+
| MySQL_._._._._        |
+-----------------------+
1 row in set (0.00 sec)
```

Returning Strings (3/5)

- Examples *(continued)*

```
mysql> SELECT TRIM('   MySQL    ') AS str;
+-------+
| str   |
+-------+
| MySQL |
+-------+
1 row in set (0.00 sec)


mysql> SELECT TRIM(LEADING 'Q' FROM 'QQQMySQLQQQ');
+---------------------------------------+
| TRIM(LEADING 'Q' FROM 'QQQMySQLQQQ') |
+---------------------------------------+
| MySQLQQQ                              |
+---------------------------------------+
1 row in set (0.00 sec)
```

Returning Strings (4/5)

- Examples *(continued)*

```
mysql> SELECT SUBSTRING('MySQL', 3);
+-----------------------+
| SUBSTRING('MySQL', 3) |
+-----------------------+
| SQL                   |
+-----------------------+
1 row in set (0.00 sec)


mysql> SELECT SUBSTRING('MySQL', 2, 2);
+--------------------------+
| SUBSTRING('MySQL', 2, 2) |
+--------------------------+
| yS                       |
+--------------------------+
1 row in set (0.00 sec)
```

Returning Strings (5/5)

- Examples *(continued)*

```
mysql> SELECT SUBSTRING_INDEX('training@mysql.com', '@', 1);
+-----------------------------------------------+
| SUBSTRING_INDEX('training@mysql.com', '@', 1) |
+-----------------------------------------------+
| training                                      |
+-----------------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
+-------------------------------------------+
| SUBSTRING_INDEX('www.mysql.com', '.', -2) |
+-------------------------------------------+
| mysql.com                                 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

Temporal Functions

- Time and date
- Several forms of description
- Generated in many ways
  - Copying existing
  - Execute built-in function
  - Build a string representation
- Date components

Type:	Default Format:
DATE	YYYY-MM-DD
TIME	HH:MM:SS
DATETIME	YYYY-MM-DD HH:MI:SS
TIMESTAMP	YYYY-MM-DD HH:MI:SS
YEAR	YYYY

Temporal Function Types

- NOW()
- CURDATE()
- CURTIME()
- YEAR(<*date_expression*>)
- MONTH(<*date_expression*>)
- DAYOFMONTH(<*date_expression*>), DAY(<*date_expression*>)
- DAYNAME(<*date_expression*>)
- HOUR(<*time_expression*>)
- MINUTE(<*time_expression*>)
- SECOND(<*time_expression*>)

Extracting Temporal Data (1/2)

- Examples

```
mysql> SELECT CURDATE(), CURTIME(), DAYNAME(NOW());
+------------+-----------+----------------+
| CURDATE()  | CURTIME() | DAYNAME(NOW()) |
+------------+-----------+----------------+
| 2006-06-09 | 17:55:40  | Friday         |
+------------+-----------+----------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT NOW(), NOW() + INTERVAL 5 DAY;
+---------------------+-----------------------+
| NOW()               | NOW() + INTERVAL 5 DAY |
+---------------------+-----------------------+
| 2006-06-09 18:02:35 | 2006-06-14 18:02:35    |
+---------------------+-----------------------+
1 row in set (0.00 sec)
```

Extracting Temporal Data (2/2)

- Examples

```
mysql> SELECT NOW(), DATE_FORMAT(NOW(), '%W the %D of %M');
+---------------------+----------------------------------------+
| NOW()               | DATE_FORMAT(NOW(), '%W the %D of %M') |
+---------------------+----------------------------------------+
| 2004-04-30 11:59:15 | Friday the 30th of April               |
+---------------------+----------------------------------------+
1 row in set (0.00 sec)
```

Numeric Functions (1/2)

- Mathematical operations
- Examples

```
mysql> SELECT ABS(-42), ABS(42);
+----------+---------+
| ABS(-42) | ABS(42) |
+----------+---------+
|       42 |      42 |
+----------+---------+
1 row in set (0.00 sec)
```

```
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 |
+-----------+----------+---------+---------+----------+
1 row in set (0.06 sec)
```

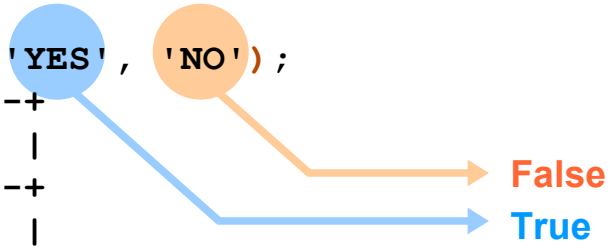
Numeric Functions (2/2)

- More numeric functions
  - **TRUNCATE(<*number*>, <*decimals*>)**
  - **FLOOR(<*number*>)**
  - **CEILING(<*number*>)**
  - **ROUND(<*number*>)**
- Mathematical functions
  - Geometry
  - Trigonometry
  - Other

Flow Control Functions (1/3)

- Choose between different values based on the result of an expression

```
mysql> SELECT IF(1 > 0, 'YES', 'NO');
+-------------------------+
| IF(1 > 0, 'YES', 'NO') |
+-------------------------+
| YES                     |
+-------------------------+
1 row in set (0.03 sec)
```



Flow Control Functions (2/3)

- **CASE/WHEN** provides branching flow control
- General syntax

```
CASE value
    WHEN <compare_value> THEN <result>
    [WHEN <compare_value> THEN <result> ...]
    [ELSE <result>]
END
```

...and...

```
CASE
    WHEN <condition> THEN <result>
    [WHEN <condition> THEN <result> ...]
    [ELSE <result>]
END
```

Flow Control Functions (3/3)

- 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 pet_name, owner, pet_gender,
CASE
    WHEN pet_gender = 'm' THEN 'Boy'
    WHEN pet_gender = 'f' THEN 'Girl'
    ELSE 'Unknown'
END
FROM pet_info
ORDER BY pet_gender;
```

Why Use Aggregate Functions? (1/2)

- Summary functions
  - Perform summary operations on a set of values
- Returns Single Value Based on Group of Values
- Only **NON NULL**

Why Use Aggregate Functions? (2/2)

- Aggregate function types
  - **MIN()**
  - **MAX()**
  - **SUM()**
  - **AVG()**
  - **COUNT()**
  - **GROUP_CONCAT()**
- Examples

```
mysql> SELECT COUNT(*)
    -> FROM Country;
+----------+
| COUNT(*) |
+----------+
|      239 |
+----------+
1 row in set (0.00 sec)
```

AND

```
mysql> SELECT COUNT(Capital)
    -> FROM Country;
+----------------+
| COUNT(Capital) |
+----------------+
|            232 |
+----------------+
1 row in set (0.00 sec)
```

Grouping with SELECT...GROUP BY (1/2)

- Use **GROUP BY** for sub-group
- Based on values on one or more columns of rows
- 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  |
+---------------+------------------+
7 rows in set (0.00 sec)
```

Grouping with SELECT...GROUP BY (2/2)

- Use **GROUP_CONCAT()** to concatenate 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
4 rows in set (0.00 sec)
```

GROUP BY...HAVING

- Eliminates rows based on aggregate values
- Only for comparisons against aggregated values
- Example

```
mysql> SELECT Continent, SUM(Population) AS pop
    -> FROM Country
    -> GROUP BY Continent
    -> HAVING SUM(Population) > 100000000;
+---------------+------------+
| Continent     | pop        |
+---------------+------------+
| Asia          | 3705025700 |
| Europe        |  730074600 |
| North America |  482993000 |
| Africa        |  784475000 |
| South America |  345780000 |
+---------------+------------+
5 rows in set (0.00 sec)
```

GROUP BY...WITH ROLLUP (1/2)

- Multiple levels of summary values
- Example

```
mysql> SELECT Continent, SUM(Population) AS pop
    -> FROM Country
    -> GROUP BY Continent WITH ROLLUP;
+---------------+------------+
| Continent     | pop        |
+---------------+------------+
| Asia          | 3705025700 |
| Europe        |  730074600 |
| North America |  482993000 |
| Africa        |  784475000 |
| Oceania       |   30401150 |
| Antarctica    |          0 |
| South America |  345780000 |
| NULL          | 6078749450 |
+---------------+------------+
8 rows in set (0.53 sec)
```

GROUP BY...WITH ROLLUP (2/2)

- Super aggregate operation
- Example

```
mysql> SELECT Continent, AVG(Population) AS avg_pop
    -> FROM Country
    -> GROUP BY Continent WITH ROLLUP;
+---------------+---------------+
| Continent     | avg_pop       |
+---------------+---------------+
| 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 |
+---------------+---------------+
8 rows in set (0.06 sec)
```

IGNORE SPACE

- Example

```
mysql> SELECT PI ();
ERROR 1305 (42000): FUNCTION world.PI does not exist

mysql> SET sql_mode = 'IGNORE_SPACE';
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT PI ();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
1 row in set (0.05 sec)
```

Further Practice: Chapter 9

- Comprehensive exercises

Chapter Summary

- Delete and modify table data
- Use the **INSERT** and **REPLACE** statements to add new data to a table
- Use the **UPDATE** statement to modify data in a table
- Use the **DELETE** statement to remove table rows

Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Combine data from multiple tables, using the **JOIN** keyword
- Explain the concept of a **JOIN**
- Use **OUTER** and **INNER** joins

Combining Multiple Tables (1/2)

- **SELECT**s can be use to retrieve data from multiple tables
  - Some questions cannot be answered by querying only one table
- A *join* operation is used to combine tables for a query
  - Joins data in one table with data in another table

Combining Multiple Tables (2/2)

- Tables to be joined

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)
```

- Joined tables *(cross join results in a 'Cartesian Product')*

```
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 example is a *cross join*, which results in a *Cartesian Product.* Also known as an *Unqualified Join*.

Categories of Joins

- Cross join
  - Combines all rows from one table with all rows of another table
  - Unqualified join
- Inner join
  - matching rows from two tables
  - Qualified join
- Outer join
  - matching and non-matching rows from two tables
  - Qualified join
- Qualified joins
  - Retain only specific row pairs according to the 'join condition'
    - A **city** *(**City** table)* is a capital of a **country** *(**Country** table)* *and* a **country** *(**Country** table)* has **cities** *(**City** table)*

Inner Joins

- Identifies combinations of matching rows from two tables
- Two different types of syntax
  - Comma separated
  - **INNER JOIN** Keywords

Comma Joins

- List tables to be joined with a comma separator
- Two separate queries can be joined into one

```
mysql> SELECT Code, Name, Language
    -> FROM Country, CountryLanguage
    ->      WHERE Continent='Africa'
    ->      AND Code = CountryCode;
+------+-------------+-----------------+
| Code | Name        | Language        |
+------+-------------+-----------------+
| DZA  | Algeria     | Arabic          |
| DZA  | Algeria     | Berberi         |
| AGO  | Angola      | Ambo            |
| AGO  | Angola      | Chokwe          |
...
| BEN  | Benin       | Adja            |
| BEN  | Benin       | Aizo            |
...
| BWA  | Botswana    | Khoekhoe        |
| BWA  | Botswana    | Ndebele         |
...
```

Table Name Aliases

- Table reference can be aliased
  - *tbl_name* **AS** *alias_name*
  - *tbl_name alias_name*
- Examples

```
mysql> SELECT t1.Name, t2.CountryCode
    -> FROM Country AS t1, City AS t2
    -> WHERE t1.Name = t2.Name;
+------------+-------------+
| Name       | CountryCode |
+------------+-------------+
| Djibouti   | DJI         |
| Mexico     | PHL         |
| Gibraltar  | GIB         |
| Armenia    | COL         |
| Kuwait     | KWT         |
| Macao      | MAC         |
| San Marino | SMR         |
| Singapore  | SGP         |
+------------+-------------+
8 rows in set (0.47 sec)
```

OR

```
mysql> SELECT t1.Name, t2.CountryCode
    -> FROM Country t1, City t2
    -> WHERE t1.Name = t2.Name;
```

INNER JOIN Keywords

- **INNER JOIN** replaces the comma separator
- **FROM** clause
- Examples

```
mysql> SELECT t1.Name, t2.CountryCode
  -> FROM Country AS t1 INNER JOIN City AS t2
  -> ON t1.Name = t2.Name;
+------------+-------------+
| Name       | CountryCode |
+------------+-------------+
| Djibouti   | DJI         |
| Mexico     | PHL         |
| Gibraltar  | GIB         |
| Armenia    | COL         |
| Kuwait     | KWT         |
| Macao      | MAC         |
| San Marino | SMR         |
| Singapore  | SGP         |
+------------+-------------+
8 rows in set (0.34 sec)
```

OR

```
mysql> SELECT t1.Name, t2.CountryCode
    -> FROM Country AS t1
    -> INNER JOIN City AS t2
    -> USING(Name);
```

JOIN Keyword

- Equivalent to **INNER JOIN**
- 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 (0.42 sec)
```

- **ON** condition -> *How*
- **WHERE** clause -> *Which*

OUTER JOIN Keywords

- Finds tables with and without matching rows
- **LEFT JOIN**
- **RIGHT JOIN**
- Example →

```
mysql> SELECT Name, Language
    -> FROM Country
    -> LEFT JOIN CountryLanguage
    -> ON Code = CountryCode;
+----------------------------+-----------------+
| Name                       | Language        |
+----------------------------+-----------------+
| Aruba                      | Dutch           |
| Aruba                      | English         |
| Aruba                      | Papiamento      |
| 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)
```

LEFT JOIN

- Comparison for join based on the first (or left) table
- Use **WHERE** clause to find mismatches
- **LEFT JOIN** example

```
mysql> SELECT Name, Language
    -> FROM Country
    -> LEFT JOIN CountryLanguage
    -> ON Code = CountryCode
    -> WHERE CountryCode IS NULL;
+-----------------------------------------+----------+
| Name                                    | Language |
+-----------------------------------------+----------+
| Antarctica                              | NULL     |
| Bouvet Island                           | NULL     |
| British Indian Ocean Territory          | NULL     |
| South Georgia and the South Sandwich Islands | NULL |
| Heard Island and McDonald Islands       | NULL     |
| French Southern territories             | NULL     |
+-----------------------------------------+----------+
6 rows in set (0.01 sec)
```

RIGHT JOIN

- Roles of tables 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)
```

OUTER JOIN: USING and NULL

- USING
  - Single/Multiple columns must exist in both tables

    … `a LEFT JOIN b USING (c1,c2,c3)`

- **NULL**
  - No matching row for right table of LEFT JOIN gives NULL result
  - Used to find rows with no counterpart in other table
  - Example

```
SELECT table1.id * FROM table1
LEFT JOIN table2
ON table1.id=table2.id
WHERE table2.id IS NULL
```

Further Practice: Chapter 10

- Comprehensive exercises

Chapter Summary

- Combine data from multiple tables, using the **JOIN** keyword
- Explain the concept of a **JOIN**
- Use **OUTER** and **INNER** joins

Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Export data using a query
- Export using the ‘mysqldump’ database backup client
- Import data with MySQL statement files
- Import using the ‘mysqlimport’ client program
- Import data from non-MySQL command files
- Import data using ODBC

Exporting Data

- Back ups are important!
  - Back up principles
  - Perform regularly
  - Consistent/Meaningful naming scheme
  - Expire back up files
  - Include with regular system backups
- Copying databases
  - From one server to another
  - Between RDBMS's

Export with a Query (1/2)

- SELECT with INTO OUTFILE
  - Writes result set directly into a file
- Example

```
SELECT * INTO OUTFILE ' Country.txt'  FROM Country
```

- INTO OUTFILE changes SELECT operation
  - File written to server host, not to the executing client
  - Data is written to a new file only
  - User that executes the statement must have FILE privilege
  - File is created with filesystem access permissions
  - File contains one line per row select by the statement

Export with a Query (2/2)

- Create CSV format text file

```
SELECT * INTO OUTFILE '/tmp/data-out.txt'
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\r'
FROM t
```

Exporting with 'mysqldump' (1/2)

- MySQL utility to export (dump) table contents
  - Full structure
  - Data only
  - Table structure only
  - In standard format
  - MySQL specifics for optimized speed
  - Compressed
- Three ways to invoke **mysqldump**

```
mysqldump [options] db_name [tables]

mysqldump [options] --databases db_name1
          [db_name2 db_name3...]

mysqldump [options] --all-databases
```

Exporting with 'mysqldump' (2/2)

- Export a database …

```
mysqldump world
```

- Export a multiple tables …

```
mysqldump world City Country
```

- Export a multiple databases …

```
mysqldump --all-databases (or -A)
```

```
mysqldump --databases world db2
```

If you do not specify a table name, the *entire* database will be dumped.

'mysqldump' Options

- `--help`
- `--opt`
- `-C (--compress)`
- `--tab=dir_name`
- `-d (--no-data)`
- `-X (--xml)`
- `--compatible=name`

Dumping to a Text File (1/2)

- Export a database file

```
mysqldump guestdb > guestdb.txt
```

- Export a multiple tables …

```
mysqldump guestdb guestTbl > guestdb.txt
```

- Export a multiple databases …

```
mysqldump -p --user=username --add-drop-table
          guestdb guestTbl

Enter password: ********
```

Dumping to a Text File (2/2)

- Text file contents

```
guestdb.txt
# MySQL dump 8.11
#
# Host: localhost Database:
guestdb#--------------------------------------------------------
# Server version 3.23.28-gamma
#
# Table structure for table 'guestTbl'
#
DROP TABLE IF EXISTS guestTbl;CREATE TABLE guestTbl (msgID int(11) NOT
NULL auto_increment,name varchar(30),email varchar(30),msgFrom
varchar(30),msgDate timestamp(14),msgBody text,PRIMARY KEY (msgID));
#
# Dumping data for table 'guestTbl'
#
INSERT INTO guestTbl VALUES (1,'Test
User1','testuser@test.com','Smallville USA',20010101182157,'This is the
body for message 1.');INSERT INTO guestTbl VALUES (2,'Test
User2','testuser2@test.com','Smallville USA',20010101182352,'This is
the body for message 2.');INSERT INTO guestTbl VALUES (3,'Test
User3','testuser3@test.com','Smalltown USA',20010101182451,'This is the
body for message 3.');INSERT INTO guestTbl VALUES (4,'Test
User4','testuser4@test.com','Smallcity USA',20010101182519,'This is the
body for message 4.');
```

Importing with SQL Statement Files

- Importing table data using **mysql** command
  - Example

```
mysql –u root -p world < world.sql
```

- Loading the data file in **mysql**
  - Example

```
SOURCE C:/files/world.sql
```

Should not import over existing database file with same name.

Importing with LOAD DATA INFILE (1/3)

- Tab delimited or comma separated files
- Characteristics you need to know about input file
- Syntax

```
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 (2/3)

- Simple syntax example

```
LOAD DATA INFILE 'file_name' INTO TABLE table_name
```

- CSV example

```
LOAD DATA INFILE '/tmp/data.txt' INTO TABLE t
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\r'
```

Importing with LOAD DATA INFILE (3/3)

- MySQL assumptions
- **LOAD DATA INFILE** clauses control…
  - Skipping data file lines
  - Loading specific table columns
  - Skipping or transforming column values
  - Control of duplicate records
  - Tracking of records
  - Control privileges
  - Data file format specification
  - Importing NULL values

Importing with 'mysqlimport'

- MySQL utility to load data files into tables
- Command line interface to **LOAD DATA INFILE**
- General syntax

```
mysqlimport <options> db_name <input_file>
```

- Matches file name with table name
- Tables must already exist

'mysqlimport' Options (1/2)

- `--help`
- `--lines-terminated-by=string`
- `--fields-terminated-by=string`
- `--fields-enclosed-by=char`
- `--ignore or -replace`
- `--local`

'mysqlimport' Options (2/2)

- Examples

```
mysqlimport --lines-terminated-by="\r\n" world City.txt

mysqlimport --fields-terminated-by=","
            --lines-terminated-by="\n" world City.txt

mysqlimport --fields-enclosed-by='"' world City.txt
```

Import with ODBC

- ODBC
  - Open Database Connectivity
  - Standard API for connecting to SQL database servers
- MySQL Connector/ODBC
  - Family of drivers
    - Formerly known as **MyODBC** drivers
  - Bridge between MySQL server and ODBC clients
  - Microsoft Excel
  - Microsoft Access
  - Visual Basic
- Protocols
- Available for Windows and Linux

Further Practice: Chapter 11

- Comprehensive exercises

Chapter Summary

- Export data using a query
- Export using the ‘mysqldump’ database backup client
- Import data with MySQL statement files
- Import using the ‘mysqlimport’ client program
- Import data from non-MySQL command files
- Import data using ODBC

Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Describe and use methods for performing nested queries (subqueries)
- Recognize the different types of subqueries
- Explain and perform the process of converting subqueries to joins

What is a Subquery? (1/2)

- Query nested inside another query
- An alternative to a join
- Enclosed in parenthesis ( )
- Subquery table type determines usage
- Discarded like a temporary table

What is a Subquery? (2/2)

- Example

```
mysql> SELECT Language
    -> FROM CountryLanguage
    -> WHERE CountryCode = (SELECT Code
    ->                      FROM Country
    ->                      WHERE Name='Finland');
+----------+
| Language |
+----------+
| Estonian |
| Finnish  |
| Russian  |
| Saame    |
| Swedish  |
+----------+
```

Why use a Subquery?

- When a join will not work
- Structured readability
  - Puts the "**S**" in "**SQL**"
- Similar to English language
  - Read subordinate clause from inside -> out
  - Focus on subquery first, then final analysis

Placement of Subqueries

- Column designation/list
- **FROM** clause
- **WHERE** clause
- Example

```
mysql> SELECT 1 AS a, (SELECT 1+1) AS b;
+---+---+
| a | b |
+---+---+
| 1 | 2 |
+---+---+
1 row in set (0.11 sec)
```

Categories of Subqueries

- Non-correlated
  - Does *not* reference outer query
  - Can stand alone
- Correlated
  - References columns in the outer query
  - Cannot stand alone

Non-Correlated Subqueries

- 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  |
...
```

Correlated Subqueries

- 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 |
...
```

The WHERE expression in the subquery makes the subquery correlated; values for the expression must be supplied by the outer query for the subquery to execute

Result Table Types

- Scalar
  - Zero or one row, single column
- Row
  - Zero or one row, one or more columns
- Column
  - Zero or one row, with one or more rows
- Table
  - Zero or more rows, multiple columns

Scalar Subqueries (1/2)

- Go almost anywhere a Scalar value is allowed
  - Functions
  - Mathematical operators
  - *And more*
- Example of Scalar subquery using **WHERE**

```
mysql> SELECT name FROM City
    -> WHERE population >
    ->    (SELECT population FROM City WHERE
            name='Tokyo');
+------------------+
| name             |
+------------------+
| São Paulo        |
| Jakarta          |
| Mumbai (Bombay)  |
| Shanghai         |
...
```

Scalar Subqueries (2/2)

- Running the subquery to confirm results

```
mysql> SELECT population FROM City WHERE name='Tokyo';
+------------+
| population |
+------------+
|    7980230 |
+------------+
1 row in set (0.00 sec)
```

Subquery Type/Placement Chart

- Cross-references types with placement

	Scalar	Row	Column	Table
Column	X			
`FROM`	X	X	X	X
`WHERE: =`	X	X		
`WHERE: <,>`	X			
`WHERE: IN, ALL, ANY, SOME`	X	X	X	
`WHERE: EXISTS`	X	X	X	X

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 LIMIT 8;
+----------------------+-------------+
| name                 | LargestCity |
+----------------------+-------------+
| Afghanistan          |     1780000 |
| Netherlands          |      731200 |
| Netherlands Antilles |        2345 |
| Albania              |      270000 |
| Algeria              |     2168000 |
| American Samoa       |        5200 |
| Andorra              |       21189 |
| Angola               |     2022000 |
+----------------------+-------------+
8 rows in set (0.00 sec
```

FROM Subquery

- Creates a derived table
  - Must be given a table alias name
    - You can use the **AS** clause to name the 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 |
+----------------+
```

WHERE Subquery

- Most common location for subqueries
  - Per type/placement chart
  - All table type results can be obtained

	Scalar	Row	Column	Table
Column	X			
`FROM`	X	X	X	X
`WHERE: =`	X	X		
`WHERE: <,>`	X			
`WHERE: IN, ALL, ANY, SOME`	X	X	X	
`WHERE: EXISTS`	X	X	X	X

WHERE 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 |
+---------------+--------------------+------------+
```

Subqueries in Comparisons

- Constructs commonly used in the **WHERE** clause
  - **IN/NOT IN**
  - **ANY**
  - **ALL**
  - **SOME**
  - **EXISTS/NOT EXISTS**

WHERE ... IN/NOT IN

- Using **IN/NOT IN**
- '**IN**' is functionally equivalent to '**=ANY**'
- Example of **IN**

```
mysql> SELECT Name, Population FROM City
    -> WHERE CountryCode IN(SELECT Code FROM Country
    ->                      WHERE Continent = 'Europe')
    -> ORDER BY Name LIMIT 10;
+----------------------+------------+
| Name                 | Population |
+----------------------+------------+
| A Coruña (La Coruña) |     243402 |
| Aachen               |     243825 |
| Aalborg              |     161161 |
| Abakan               |     169200 |
| Aberdeen             |     213070 |
| Aix-en-Provence      |     134222 |
| Albacete             |     147527 |
| Alcalá de Henares    |     164463 |
| Alcorcón             |     142048 |
| Alessandria          |      90289 |
+----------------------+------------+
10 rows in set (0.05 sec)
```

WHERE ... ALL, ANY and SOME (1/3)

- Using **ALL**, **ANY** and **SOME**
- Quantified comparison for 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           |
+-----------------+
3 rows in set (0.01 sec)
```

WHERE ... ALL, ANY and SOME (2/3)

- Comparison using ANY
  - *Any* qualifying subquery values
- Example of **ANY**

```
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  |
+---------+
```

WHERE ... ALL, ANY and SOME (2/3)

- Run subquery to confirm results

```
mysql> SELECT CountryCode FROM CountryLanguage
    -> WHERE Language = 'Spanish';
+-------------+
| CountryCode |
+-------------+
| ABW         |
| AND         |
...
| USA         |
| VEN         |
| VIR         |
+-------------+
28 rows in set (0.00 sec)
```

WHERE ... EXISTS/NOT EXISTS

- Using **EXISTS/NOT EXISTS**
- Tests whether a subquery returns 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 (0.03 sec)
```

NOT EXISTS >>

```
+---------------+
| Continent     |
+---------------+
| Asia          |
| Europe        |
| North America |
| Africa        |
| Oceania       |
| Antarctica    |
| South America |
+---------------+
7 rows in set (0.01 sec)
```

Finding Mismatches

- **SELECT...NOT IN**

```
SELECT Name FROM Country
WHERE Code NOT IN
    (SELECT  CountryCode FROM CountryLanguage);
```

- **SELECT...LEFT JOIN**

```
SELECT Name FROM Country
LEFT JOIN CountryLanguage
ON Code = CountryCode
WHERE CountryCode IS NULL;
```

Modifying Tables Using Subqueries

- Subqueries not limited to **SELECT**
- Can use **DELETE** and **UPDATE**
- **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);
```

Converting Joins to Subqueries

- More easily constructed query

```
mysql> SELECT DISTINCT Language
    -> FROM Country
    -> JOIN CountryLanguage
    -> ON CountryCode = Code
    -> WHERE Continent='Africa';
+----------------------+
| Language             |
+----------------------+
| Ambo                 |
| Chokwe               |
...
| Nsenga               |
| Tongan               |
+----------------------+
215 rows in set (0.08 sec)
```

OR

```
SELECT DISTINCT Language
FROM CountryLanguage
WHERE CountryCode IN
    (SELECT Code FROM Country
     WHERE Continent='Africa');
```

Subquery Tips

- May need to run several times through the data in order to achieve desired result
- A temporary table alias may be necessary
- Subqueries can be used in **SELECT**, **UPDATE**, **DELETE**, and **INSERT** statements
- Used in the **SELECT**, **FROM**, **WHERE**, **HAVING**, and **ORDER BY** clauses of queries
- Used in conditions that utilize comparison operators and special-purpose operators
- Arithmetic operators can be appear on either side of a subquery; **=**, **<>**, **<**, **>**,**<=**, **>=**

Converting Subqueries to Joins (1/2)

- Joins can be more efficient than subqueries
  - Convert a subquery to a join, if subquery is running slow
- Subquery versus Join…

```
mysql> SELECT Name FROM Country
    -> WHERE Code IN
    ->  (SELECT CountryCode
    ->  FROM CountryLanguage);
+----------------------------+
| Name                       |
+----------------------------+
| Afghanistan                |
| Netherlands                |
| Netherlands Antilles       |
| Albania                    |
| Algeria                    |
| American Samoa             |
...
```

--->

```
mysql> SELECT Name
    -> FROM Country,
          CountryLanguage
    -> WHERE Code=CountryCode;
+----------------------+
| Name                 |
+----------------------+
| Afghanistan          |
| Afghanistan          |
| Afghanistan          |
| Afghanistan          |
| Netherlands          |
| Netherlands          |
...
```

Converting Subqueries to Joins (2/2)

- List each name once

```
mysql> SELECT DISTINCT Name
    -> FROM Country, CountryLanguage
    -> WHERE Code=CountryCode;
+----------------------------+
| Name                       |
+----------------------------+
| Afghanistan                |
| Netherlands                |
| Netherlands Antilles       |
| Albania                    |
| Algeria                    |
| American Samoa             |
...
```

Further Practice: Chapter 12

- Comprehensive exercises

Chapter Summary

- Describe and use methods for performing nested queries (subqueries)
- Recognize the different types of subqueries
- Explain and perform the process of converting subqueries to joins

Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Create views of tables with specific data
- Explain the basics of transactions and transaction statements
- Describe and use ACID transactions
- List the features of the MySQL storage engines
- Name the storage engines used by MySQL
- Define and use the INFORMATION_SCHEMA database
- Retrieve metadata using INFORMATION_SCHEMA
- Describe the primary features of the Enterprise Monitor and Workbench GUIs

Creating Views

- A view is a database object defined by **SELECT**
- Several benefits to views
- Using the **CREATE VIEW** statement
- Simple syntax

```
CREATE
    [OR REPLACE]
    VIEW view_name [<column_list>]
    AS select_expression
```

CREATE VIEW...SELECT

- **CREATE VIEW** example

```
CREATE VIEW world.Ctry_View AS SELECT * FROM Country
```

- Can use many types of **SELECT** statements
  - Example

```
CREATE VIEW per_capita_v AS
    SELECT Name, GNP, Population, GNP/Population
           AS per_capita FROM Country;

mysql> SELECT * from per_capita_v;
+---------------------+----------+------------+------------+
| Name                | GNP      | Population | per_capita |
+---------------------+----------+------------+------------+
| Afghanistan         |   5976.00 |   22720000 |   0.000263 |
| Netherlands         | 371362.00 |   15864000 |   0.023409 |
| Netherlands Antilles |   1941.00 |     217000 |   0.008945 |
| Albania             |   3205.00 |    3401200 |   0.000942 |
...
```

Displaying VIEW Information

- Base versus View
- Base table statements work for views
  - **DESCRIBE**
  - **SHOW TABLES**
  - **SHOW TABLE STATUS**
- Showing **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)
```

Showing the Table Types

- Using **SHOW FULL TABLES**
  - Base and view types
  - Example

```
mysql> SHOW FULL TABLES FROM world;
+-----------------+------------+
| Tables_in_world | Table_type |
+-----------------+------------+
| city            | BASE TABLE |
| country         | BASE TABLE |
| countrylanguage | BASE TABLE |
| ctry_view       | VIEW       |
| gelderlanddist  | BASE TABLE |
| per_capita_v    | VIEW       |
+-----------------+------------+
6 rows in set (0.00 sec)
```

View Definition Restrictions

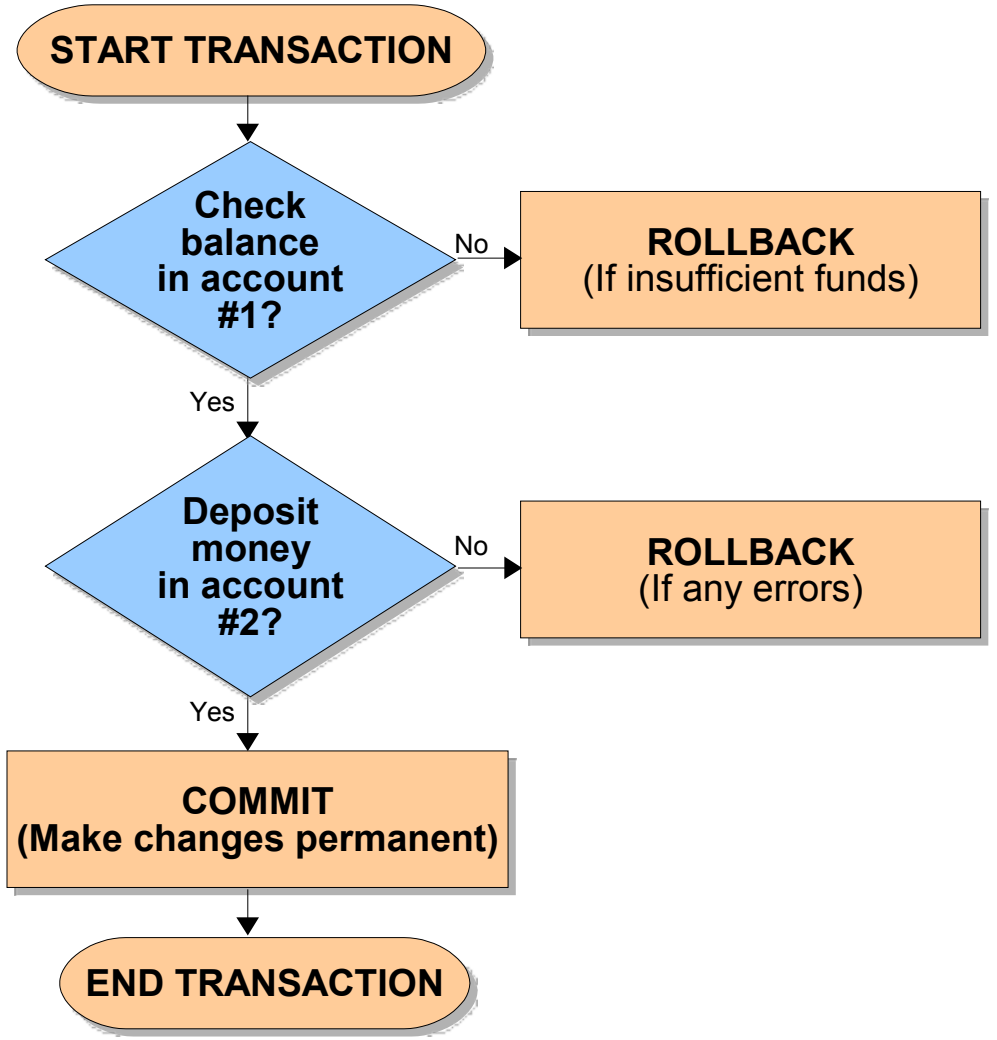
- **SELECT** statement restrictions
  - Cannot contain a subquery in the **FROM** clause
  - Cannot refer to system or user variables
  - Cannot refer to prepared statement parameters
- A stored routine cannot refer to routine parameters or local variables
- Any referenced table or view must exist
- Cannot refer to a temporary table
- Cannot create a temporary view
- Tables named in view must exist

Basics of Transactions (1/2)

- When concurrent statements are needed
- Means for executing one or more statement as single unit
- *All* or *none* succeed
- Execute if *all* good
- Cancel if error or incomplete

Basics of Transactions (2/2)

- Banking transaction flow chart



Transaction Statements

- **START TRANSACTION**
- **COMMIT**
- **ROLLBACK**
- **SET AUTOCOMMIT**
  - Autocommit mode is enabled by default
    - Each statement is considered a transaction and updates data
  - Disable for transactional engine with `SET AUTOCOMMIT=0`
  - Disable with `START TRANSACTION`
  - Check autocommit setting with `SELECT @@AUTOCOMMIT`

What is ACID?

- **A**tomic
  - All statements execute successfully or are canceled as a unit
- **C**onsistent
  - Database that is in a consistent state when a transaction begins, is left in a consistent state by the transaction
- **I**solated
  - One transaction does not affect another
- **D**urable
  - All committed changes are stored persistently (not lost), and all rolled back changes guaranteed to be gone

Storage Engines and MySQL

- Several storage engines to choose from
- Best fit for your application
- What types of queries will you use?
- Specify engine using **CREATE TABLE**
  - Example

```
CREATE TABLE t (i INT) ENGINE = InnoDB
```

- Uses system default if not set

Displaying Storage Engines (1/3)

- View current engine setting
  - **SHOW CREATE TABLE**
  - **SHOW TABLE STATUS**
- 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=MyISAM DEFAULT CHARSET=latin1
    1 row in set (0.00 sec)
```

Displaying Storage Engines (2/3)

- **SHOW TABLE STATUS** Example

```
mysql> SHOW TABLE STATUS LIKE 'CountryLanguage'\G
*********************** 1. row ***********************
Name: CountryLanguage
 Engine: MyISAM
 Version: 10
 Row_format: Fixed
 Rows: 984
 Avg_row_length: 39
 Data_length: 38376
 Max_data_length: 167503724543
 Index_length: 22528
 Data_free: 0
 Auto_increment: NULL
 Create_time: 2005-04-26 22:15:35
 ...
```

Displaying Storage Engines (3/3)

- Table management is engine independent
- Storage engine implementation “Lower Tier”
- Good to know which engine manages a table
- Engine must be compiled and enabled

Storage Engines Available on Server

- View available storage engines

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
Engine: MyISAM
 Support: YES
 Comment: Default engine as of MySQL 3.23 with great
 performance
...
*************************** 2. row ***************************
Engine: MEMORY
 Support: YES
 Comment: Hash based, stored in memory, useful for
temporary tables
...
*************************** 3. row ***************************
Engine: InnoDB
 Support: DEFAULT
Comment: Supports transactions, row-level locking, and foreign keys
...
```

Common Storage Engines

- MyISAM 
  - Fast
  - Data stored in **table.MYD** file
  - Table-level locking
- InnoDB 
  - Transactional
  - Foreign keys
  - Row-level locking
  - Backups
- Memory 
  - Data is in memory *only*

The MyISAM Storage Engine

- Manages tables with specific characteristics
  - Represented by three files
  - Most Flexible **AUTO_INCREMENT**
  - Fast, compressed, read-only tables save space
  - Manages contention between queries
  - Portable storage format
  - Specify preference for number of rows for a table
  - Disable updating of non-unique indexes and enable the indexes
  - Tables take up very little space

MyISAM Row Storage Formats (1/2)

- Fixed-row format
  - All rows have the same size
  - Rows stored as multiples of the row size, for easy to look up
  - Fixed-size rows take more space
  - All columns must have fixed width data types
- Dynamic-row format
  - Rows take varying amounts of space
  - Rows cannot be looked up as efficiently
  - Take less space since rows are not padded to a fixed size
  - Fragmentation can occur more easily than for fixed-row tables
  - Repair/corruption problems can occur

MyISAM Row Storage Formats (2/2)

- Compressed format
  - Packed to save space
  - Storage is optimized for quick retrieval
  - Read-only tables
  - Reversible operation

The InnoDB Storage Engine (1/2)

- Manages tables with specific characteristics
  - Represented by `.frm` file in the database directory
  - Data and indexes stored in the *tablespace*
  - Supports **COMMIT** and **ROLLBACK**
  - Full ACID compliance
  - Auto-recovery after a crash
  - Multi-versioning and row-level locking
  - Supports foreign keys and referential integrity
  - Tablespace storage format is portable
  - Supports consistent and online backups

The InnoDB Storage Engine (2/2)

- Log files
  - Information about ongoing transactions
  - Cached in memory
  - Log buffer written and flushed to log files at COMMIT (or as specified)
  - Crash auto-recovery
- Support is standard in binary distributions

The MEMORY Storage Engine

- Uses tables stored in memory
- Fixed-Length rows
- Manages tables with specific characteristics
  - Represented by `.frm` file and in memory
  - In-memory storage very fast
  - Table contents do not survive restart
  - Cannot use different character sets for different columns
  - Limited file size
  - Manages contention using table-level locking
  - Cannot contain **TEXT** or **BLOB** columns
  - Can use different character sets for different columns
- Formerly HEAP Engine

Storage Engine Summary

	MyISAM	MEMORY	InnoDB
Usage	Fastest for read heavy apps	In-Memory storage	Fully ACID compliant transactions
Locking	Large-grain table locks, no non-locking reads	Large grain table locks	Multi-versioning, Row-level locking
Transactions	NO	NO	YES
Durability	Table recovery	No disk I/O or persistence	Durability recovery

Other Engines

- Optional storage engines
- Select when configuring MySQL
- Many more engines available
  - FEDERATED
  - NDB
  - EXAMPLE
  - ARCHIVE
  - CSV
  - BLACKHOLE

Using INFORMATION_SCHEMA (1/2)

- Metadata
  - Tables of all tables
  - Tables of columns
- Information about all MySQL server databases
  - System views, *not* data
- Database cannot be manipulated

Using INFORMATION_SCHEMA (2/2)

- Two methods for obtaining metadata
  - **SHOW** statement
  - **SELECT...FROM** with **INFORMATION_SCHEMA**
- Advantages of **INFORMATION_SCHEMA**
- Advantages of **SHOW**

INFORMATION_SCHEMA Options (1/2)

- CHARACTER_SETS
- COLLATIONS
- COLLATION_CHARACTER_SET_APPLICABILITY
- COLUMNS
- COLUMN_PRIVILEGES
- KEY_COLUMN_USAGE
- ROUTINES
- SCHEMATA

INFORMATION_SCHEMA Options (2/2)

- SCHEMA_PRIVILEGES
- STATISTICS
- TABLES
- TABLE_CONSTRAINTS
- TABLE_PRIVILEGES
- TRIGGERS
- USER_PRIVILEGES
- VIEWS

Find an ER Diagram of the INFORMATION_SCHEMA database online.

Viewing INFORMATION_SCHEMA

- **INFORMATION_SCHEMA** example

```
mysql> SELECT table_name, engine
    ->       FROM INFORMATION_SCHEMA.TABLES
    ->       WHERE table_schema = 'world'
    ->       ORDER BY table_name DESC;
+-----------------+--------+
| table_name      | engine |
+-----------------+--------+
| per_capita_v    | NULL   |
| gelderlanddist  | InnoDB |
| ctry_view       | NULL   |
| countrylanguage | MyISAM |
| country         | MyISAM |
| city            | MyISAM |
+-----------------+--------+
```

SCHEMATA

- Table in **INFORMATION_SCHEMA** database
- Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA
    -> WHERE SCHEMA_NAME = 'world'\G
*************************** 1. row ***************************
                CATALOG_NAME: NULL
                 SCHEMA_NAME: world
DEFAULT_CHARACTER_SET_NAME: latin1
    DEFAULT_COLLATION_NAME: latin1_swedish_ci
                  SQL_PATH: NULL
```

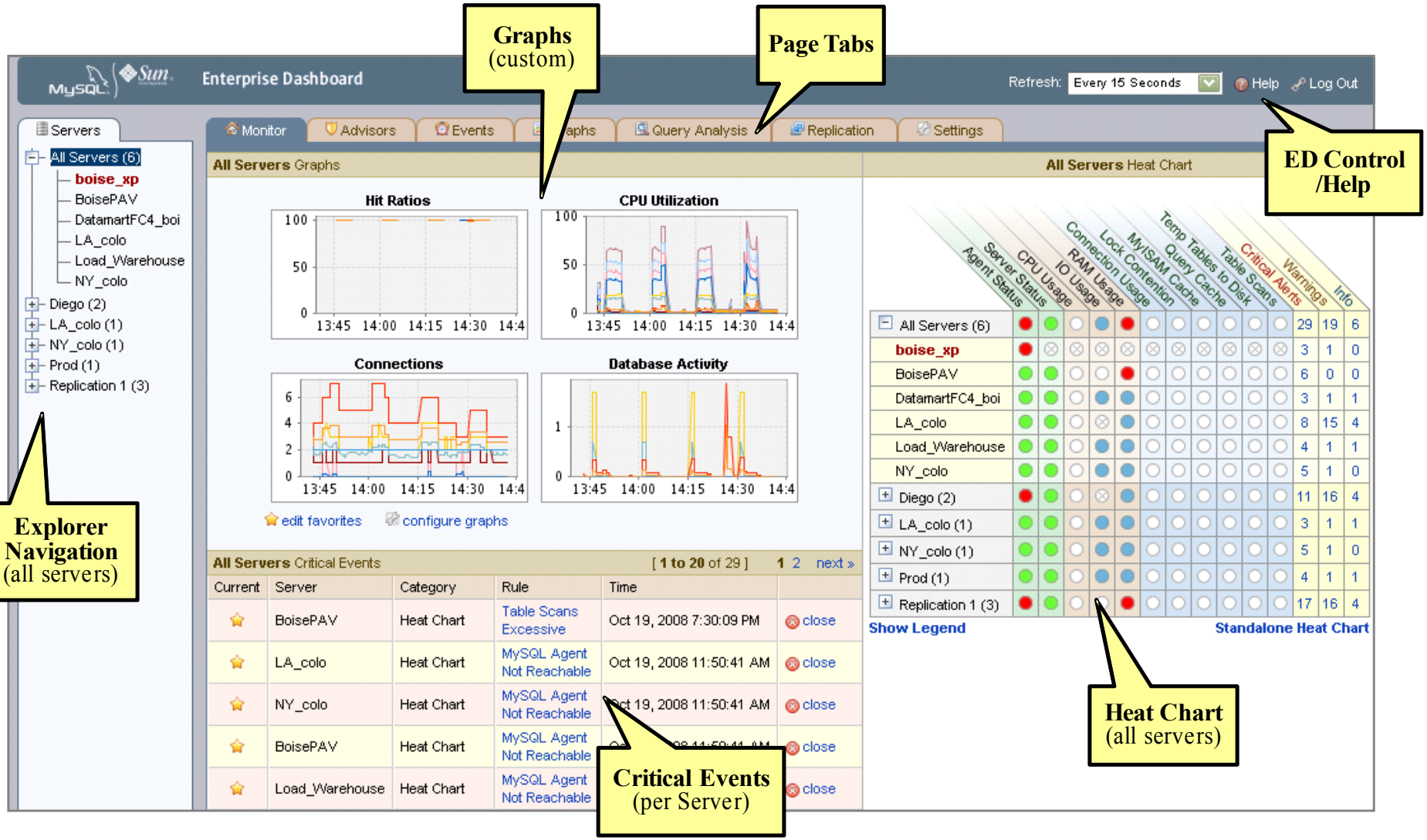
MySQL GUI's

- MySQL provides several graphical user interface tools
- **MySQL Enterprise Monitor** and **MySQL Workbench** are of particular use to beginning MySQL users
- This course gives only a cursory view of these GUI's
  - For more information see our website and other MySQL courses

MySQL Enterprise Monitor

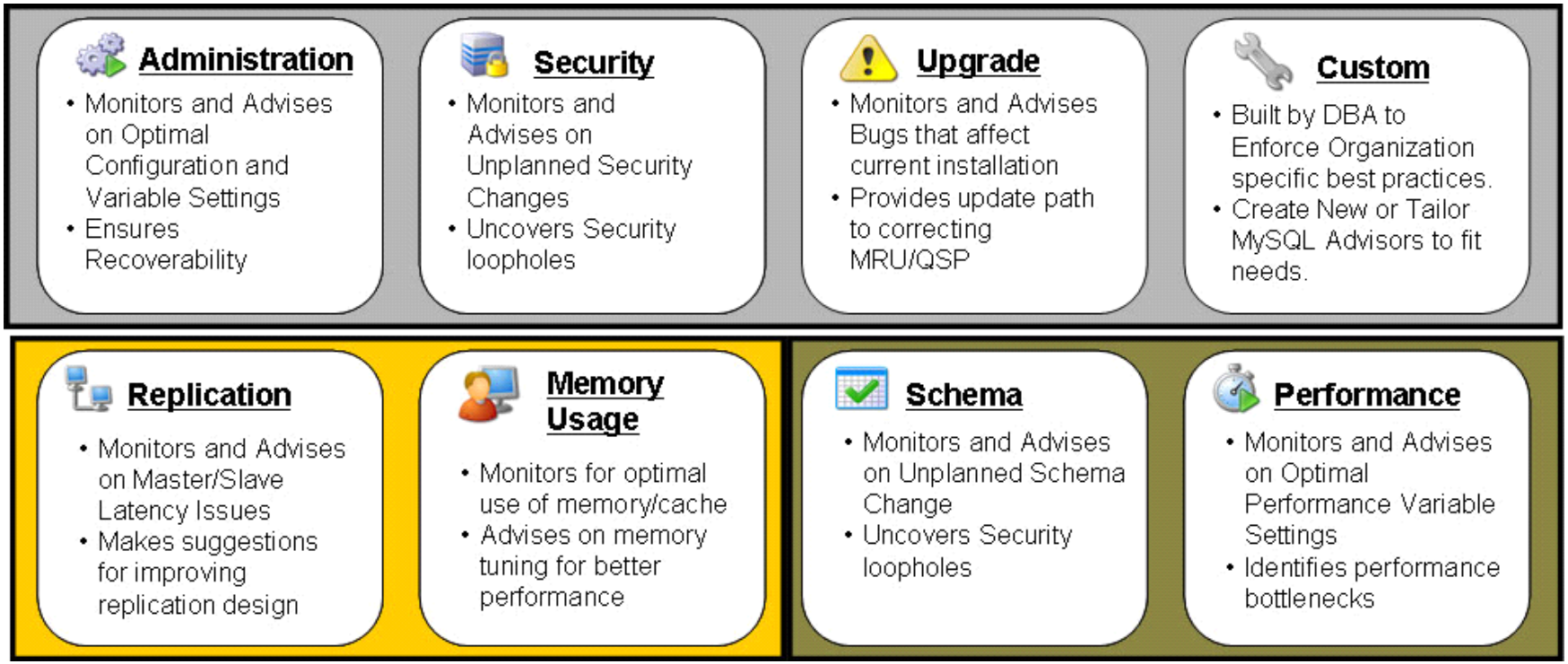
- Web-based monitoring and advising system
  - Virtual DBA assistant
  - Runs completely within corporate firewall
- Principle features:
  - Enterprise Dashboard
  - Server/group management
  - "At-a-glance" monitoring
  - MySQL and custom advisors and rules
  - Advisor rule scheduler
  - Customizable thresholds and alerts
  - Events and Alert history
  - Replication
  - Query Analyzer

Enterprise Dashboard



Enterprise Advisors

- Advisors are divided into categories by function



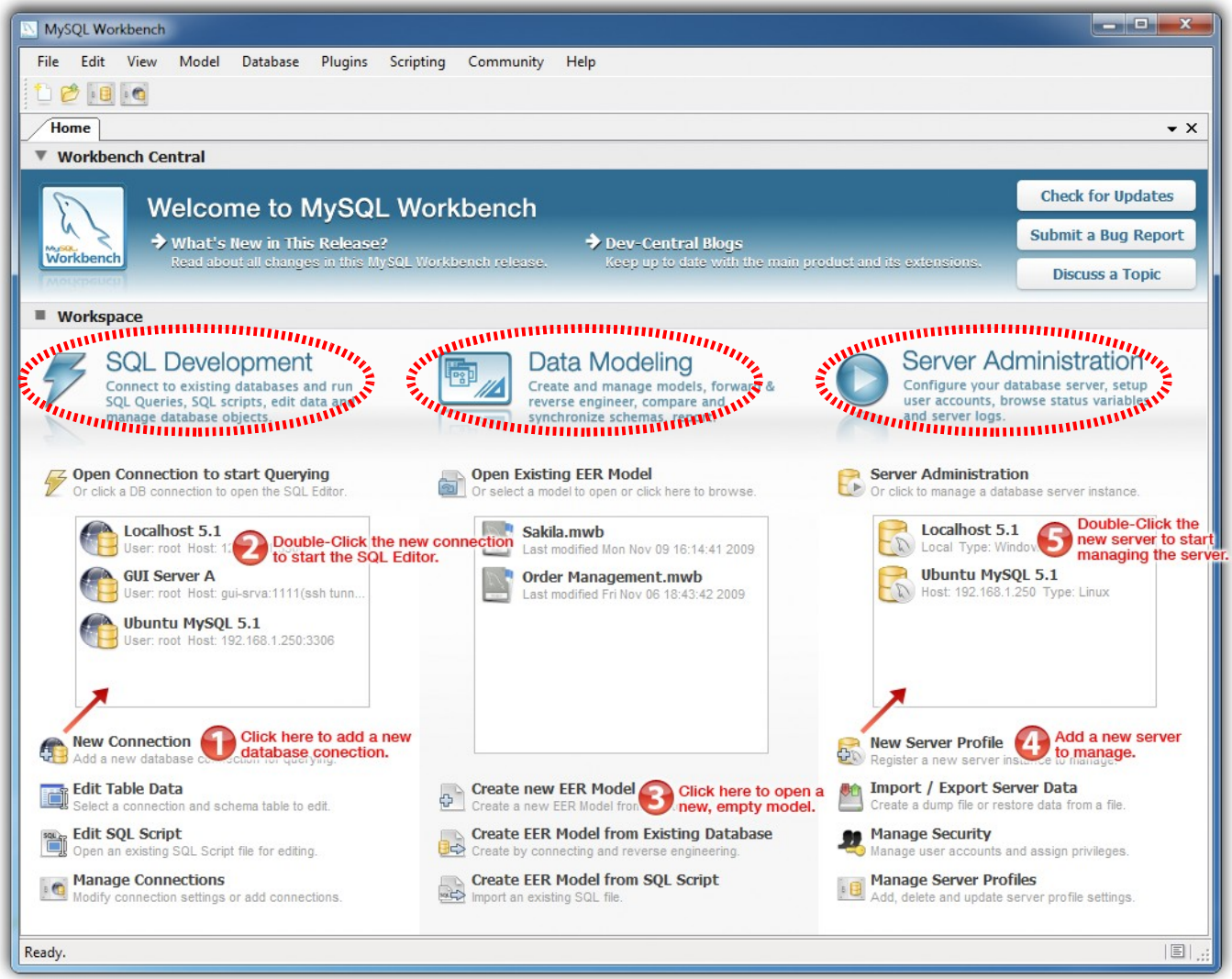
Enterprise Monitor Subscription Levels

	Silver	Gold	Platinum
Enterprise Dashboard	✓	✓	✓
Notifications and Alerts	✓	✓	✓
Custom Advisor	✓	✓	✓
Upgrade Advisor	✓	✓	✓
Administration Advisor	✓	✓	✓
Security Advisor	✓	✓	✓
Replication Monitor		✓	✓
Replication Advisor		✓	✓
Query Analysis Advisor		✓	✓
Memory Usage Advisor		✓	✓
Schema Advisor			✓
Performance Advisor			✓

MySQL Workbench

- Cross-platform GUI for the MySQL Server
  - Current BETA (5.2) version
- Three separate functional modules:
  - SQL Development
    - Edit and execute SQL queries and scripts
    - Create or alter database objects
    - Edit table data
  - Data Modeling
    - EER Modeling
    - Design, generate and manage databases
  - Server Administration
    - Start/stop server
    - Edit database server configuration
    - Manage users
    - Data import/export

MySQL Workbench GUI Window



MySQL Workbench Editions

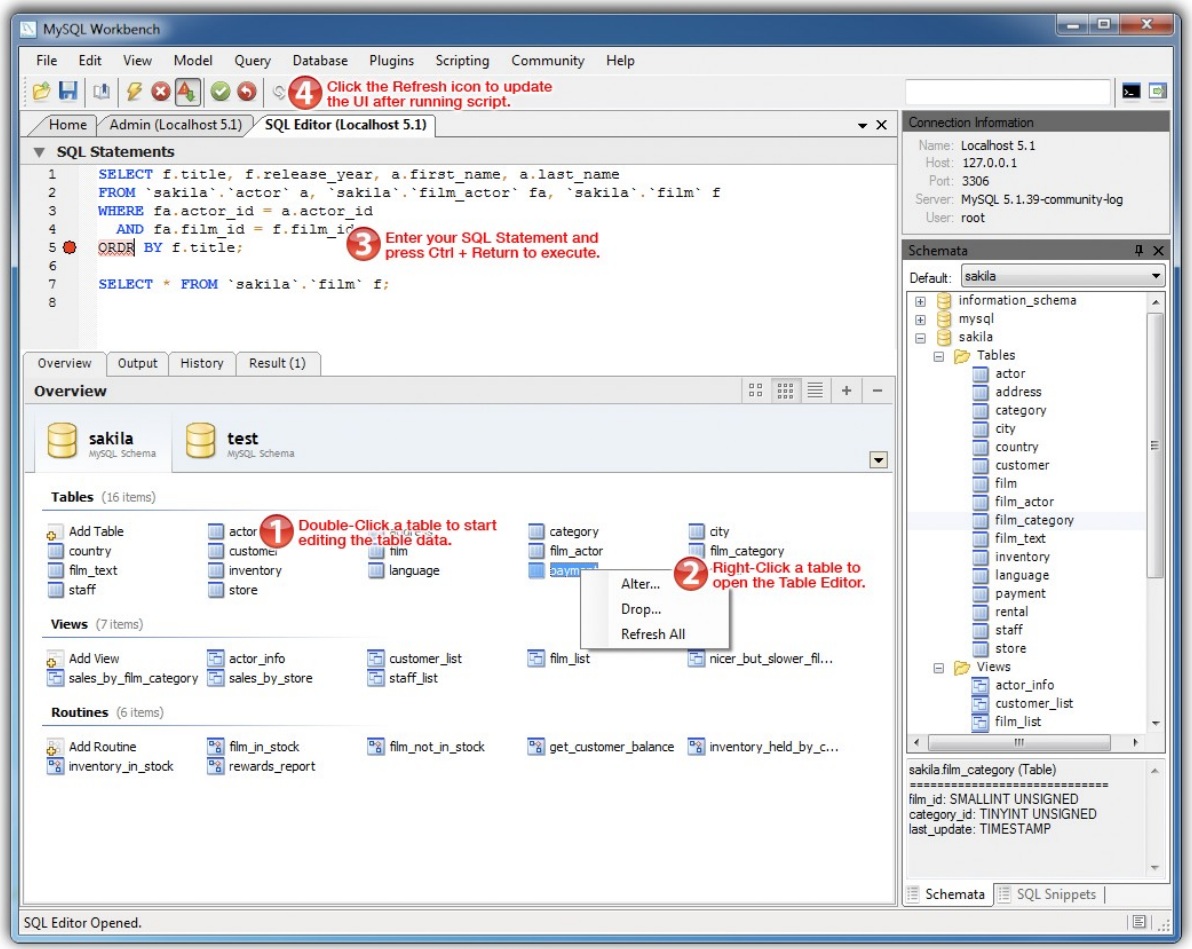
- Two versions of Workbench are available
- Community Edition (OSS)
  - Foundation of all Workbench editions
  - Full-featured, powerful database management tool
  - Open Source GPL available for FREE from our website
- Standard Edition (SE)
  - Commercial extension of OSS version
  - Advanced features
  - Available for purchase from our website

MySQL Workbench Installation

- Available for the following OS platforms
  - Windows
  - Linux
  - Mac OS X
- Requirements for installation
  - Listed in the **MySQL Workbench Reference Manual**
    http://dev.mysql.com/doc/workbench/en/index.html
- Go to MySQL Workbench download web page
  - **http://dev.mysql.com/downloads/workbench/**
- Quick-Start Tutorial on Development page
  - **http://wb.mysql.com/**

SQL Development Module

- Provides GUI for querying and analyzing data using SQL
- The **SQL Editor** is accessed from the main window of the **MySQL Workbench**



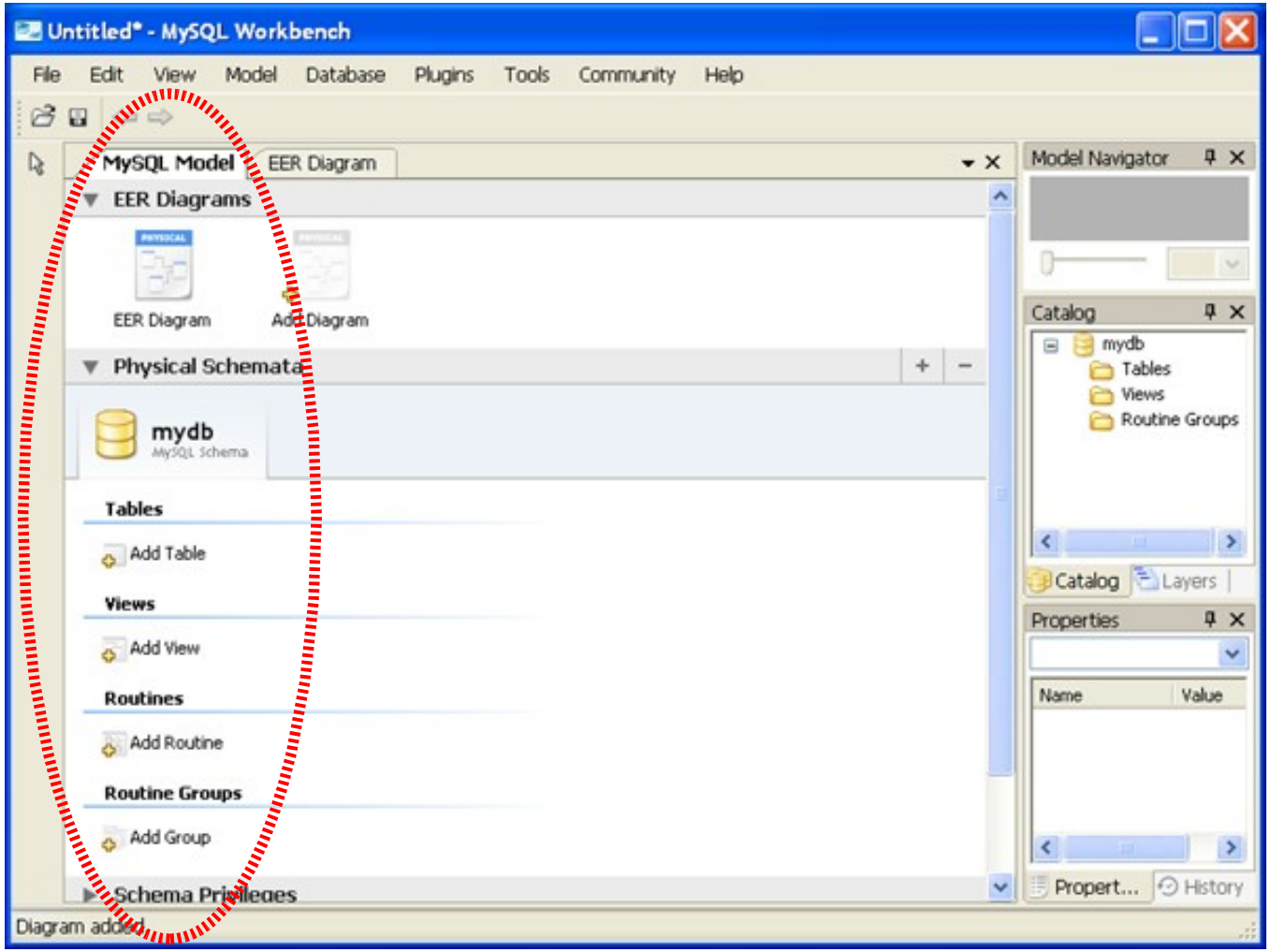
Data Modeling Module

- Intuitive visual database modeling and design tool
- Enables users to design, generate and manage all types of databases
- Model-driven database design
  - Modeling purpose and relationships of data for existing and new databases
  - Best means for representing the definition of data elements
  - Use of Extended Entity-Relationship diagrams (EER)
- Everything a data modeler needs
  - Creating Extended-Entity-Relationship (EER) models
  - Change management
  - Documentation

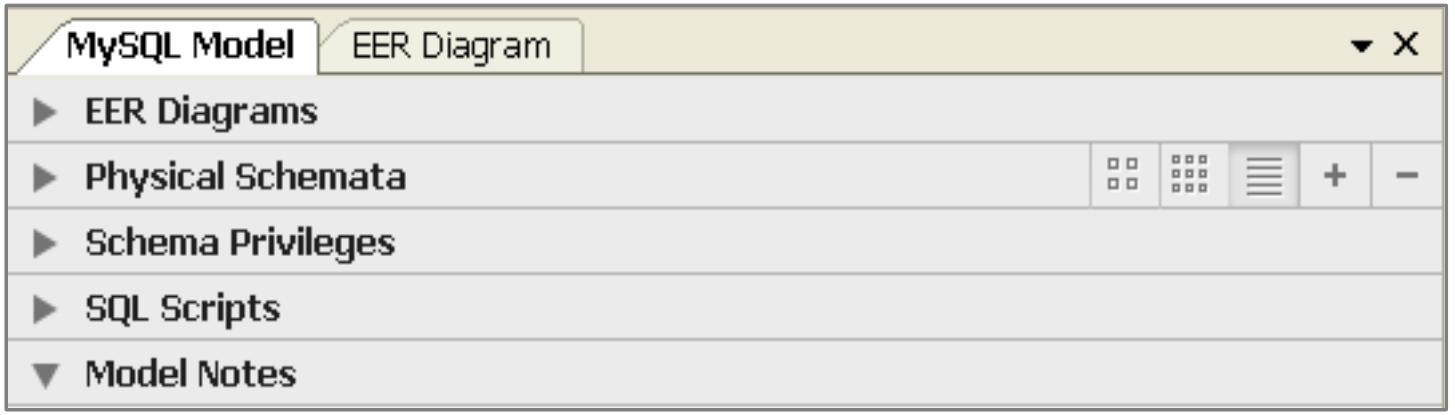
Data Modeling Feature Categories

- Database Design
- Forward and Reverse Engineering
- Database Maintenance
- Reporting and Documentation

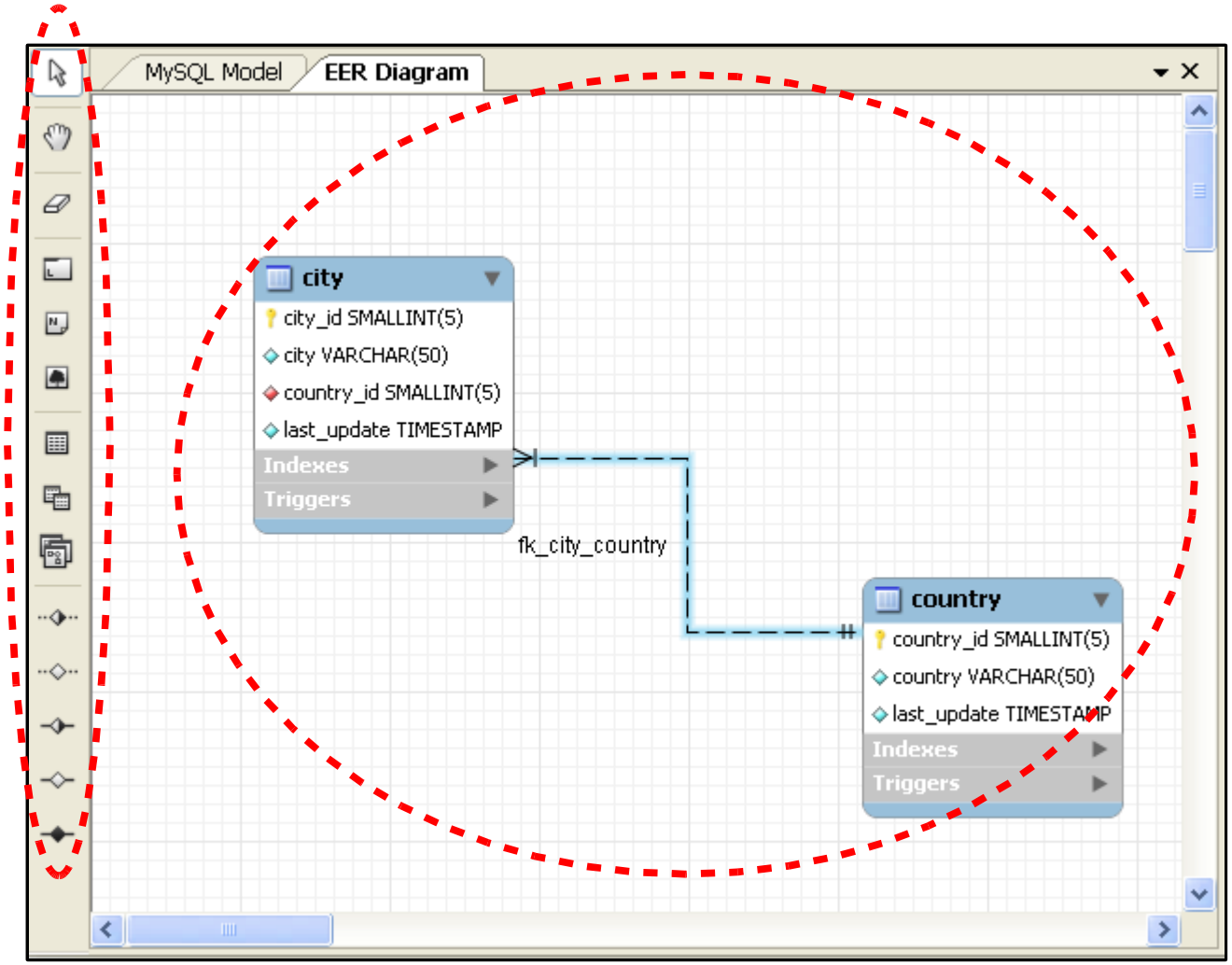
Data Modeling GUI



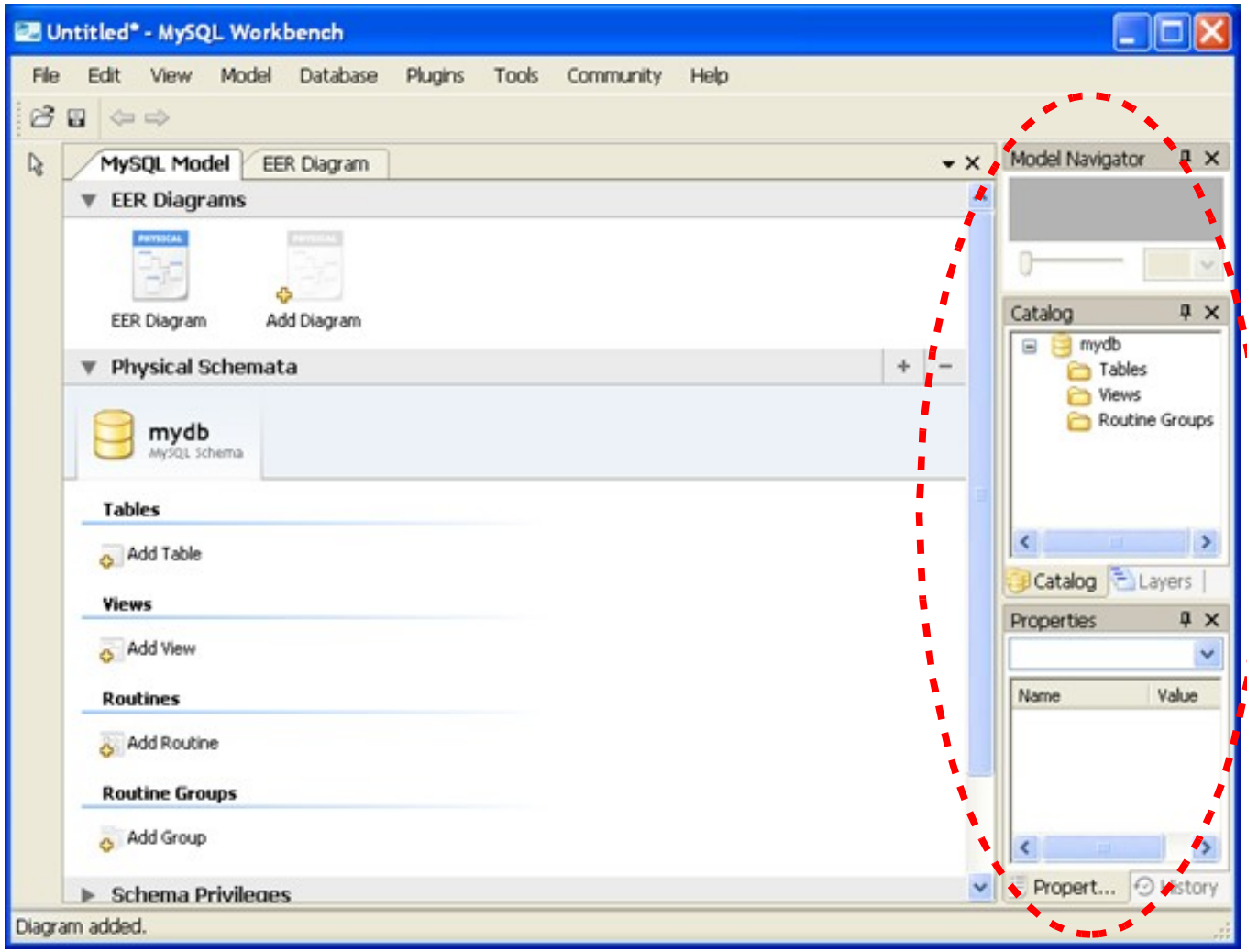
MySQL Model Window



EER Diagram Window



Functional Palettes



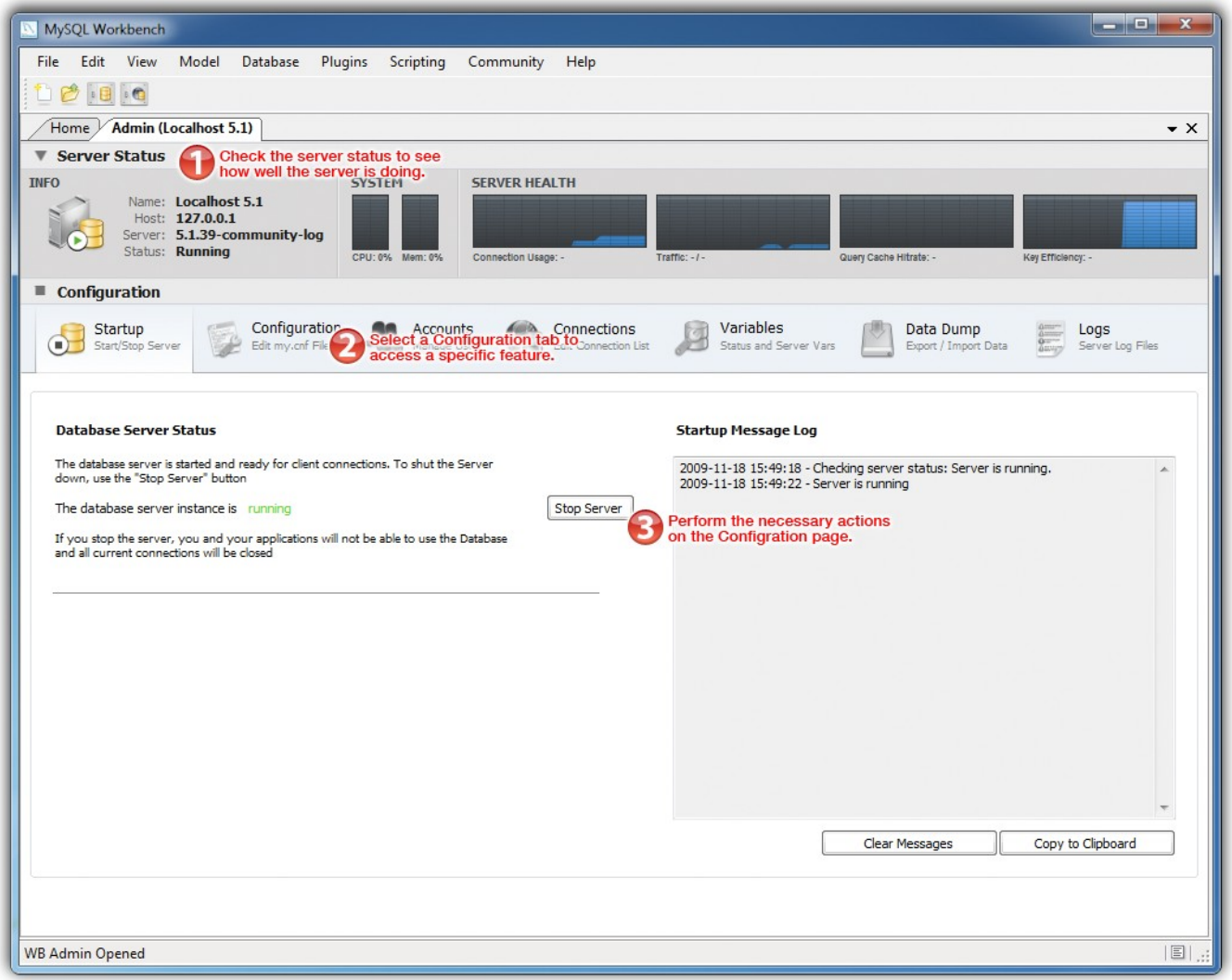
Server Administration Module (1/2)

- Previously available as the stand-alone **MySQL Administrator** GUI
- Server function categories
  - Startup
  - Configuration
  - Accounts
    - A **user** account is required to log in and to restrict access to various resources
  - Connections
  - Variables
  - Data Dump
  - Logs

Server Administration Module (2/2)

- System status
  - CPU utilization
  - Memory usage
  - Connection health
- Server “health”
  - Connection Usage
  - Traffic
  - Query Cache Hit Rate
  - Key Efficiency

Server Administration GUI



Chapter Summary

- Create views of tables with specific data
- Explain the basics of transactions and transaction statements
- Describe and use ACID transactions
- List the features of the MySQL storage engines
- Name the storage engines used by MySQL
- Define and use the INFORMATION_SCHEMA database
- Retrieve metadata using INFORMATION_SCHEMA
- Describe the primary features of the Enterprise Monitor and Workbench GUI's

Course Summary (1/2)

- List the features and benefits of MySQL
- Find information about products, support and training
- Install and start the MySQL server
- Explain the basics of relational databases
- Distinguish the SQL language from MySQL language extensions
- Explain data/column types with regard to efficient database design
- Inspect design, structure and content of databases
- Design and Create an efficiently structured database
- Retrieve data using the SELECT statement

Course Summary (2/2)

- Troubleshoot typical warnings and errors
- Modify and delete a database
- Modify and delete table row data
- Use data aggregation in queries
- Combine data from multiple tables using JOIN
- Write subqueries
- List simple functions (String, Date, Numerical)
- Understand the primary methods for exporting and importing data
- Describe MySQL connectors and their features
- Explain MySQL storage engine concepts

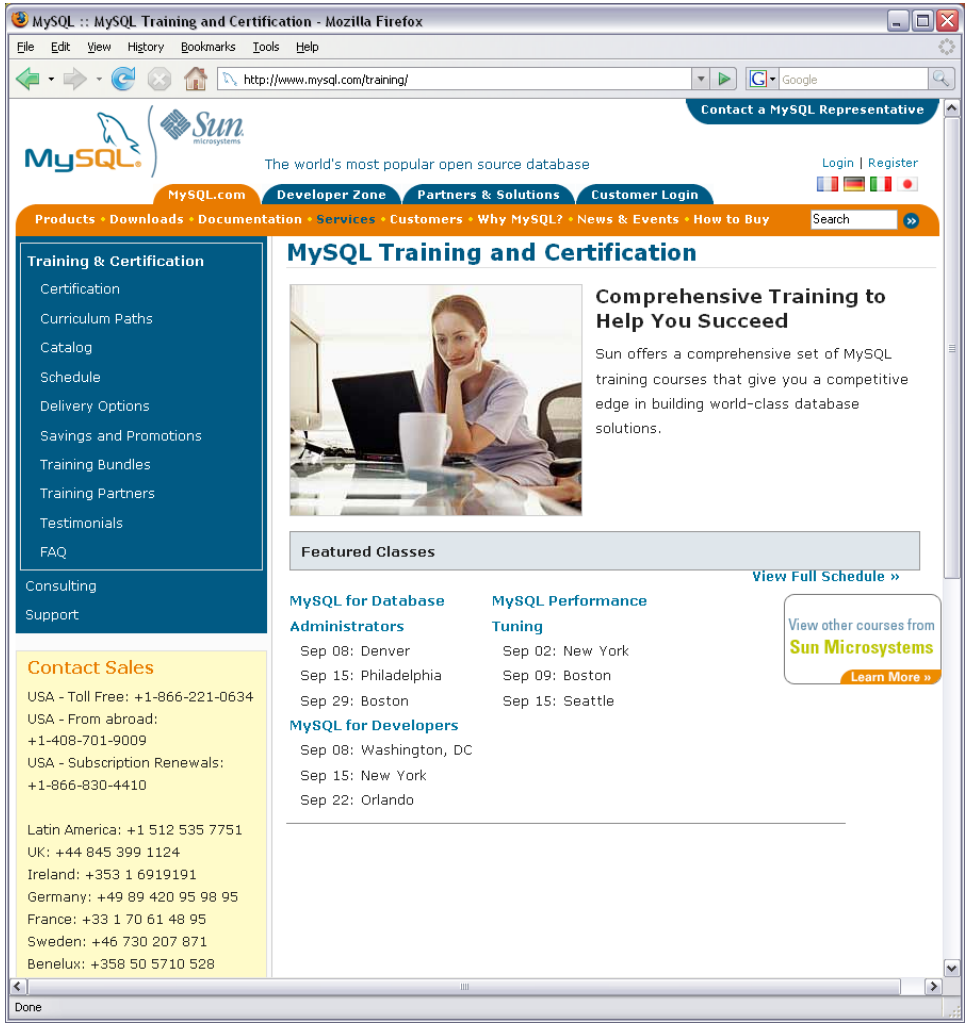
Course Chapter Map

1. INTRODUCTION
2. MySQL CLIENT/SERVER MODEL
3. DATABASE BASICS
4. DATABASE DESIGN
5. DATABASE CREATION
6. BASIC QUERIES
7. DATABASE MAINTENANCE
8. DATA MANIPULATION
9. FUNCTIONS
10. JOINING TABLES
11. EXPORTING AND IMPORTING DATA
12. SUBQUERIES
13. SUPPLEMENTAL INFORMATION
14. CONCLUSION

Learning Objectives

- Describe specific contents of the MySQL training and certification web pages
- Complete a course evaluation, to aid in continuing improvements to MySQL courses
- Use the various contact information for additional training information and support
- Find additional training information
- Q & A course wrap-up

Training and Certification Web Page



- Certification
- Curriculum Paths
- Catalog
- Schedule
- Delivery Options
- Savings and Promotions
- Training Bundles
- Training Partners
- Testimonials
- FAQ

http://www.mysql.com/training/

We Need Your Evaluation!

- Please take time to give us your opinions
  - http://www.mysql.com/training/evaluation.php
  - Requires your **first name**, **last name** and **email** address
    - Must be the ***exact*** same as used to sign up for the course

THANK YOU!!!

- We appreciate your attendance and participation!
- Contact us regarding training issues
  - **Website: http://www.mysql.com/training/**
  - **Email: training@mysql.com**
  - **Phone: USA Toll Free: 1-866-697-7522**
    Worldwide: 1-208-514-4780

Q&A Session

- Question and Answers
- More questions after class?
  - Get answers on our FAQ Reference Manual online
  - http://dev.mysql.com/doc/refman/5.1/en/faqs.html
- Want to do the labs on your own?
  - Download example databases from our website
  - http://dev.mysql.com/doc/
    - Under "**Example Databases**"