

# **MySQL for Developers**

**SQL-4501 Release 2.2**

D61830GC10  
Edition 1.0

**ORACLE®**



# Day 1

- Overview
- Why MySQL?
- Installation
- Data Definition Language (DDL)
  - Database
  - Tables
- Data Manipulation Language (DML)  
& Transactions



# Day 1

- Data Retrieval Language (DRL)
  - SQL Expressions
  - Built in functions
    - Comparison
    - Control Flow
    - Cast
    - Numeric
    - String
    - Date / Time



# Overview

# History of MySQL



- Original development of MySQL by [Michael Widenius](#) and [David Axmark](#) beginning in 1994 and First internal release on 23 May 1995.
- In October 2005, [Oracle Corporation](#) acquired [Innobase OY](#), the company that developed the [InnoDB](#) storage engine that allows MySQL to provide such functionality as transactions and foreign keys.

# History of MySQL



- In February 2006, Oracle Corporation acquired Sleepycat Software, makers of the Berkeley DB, a database engine providing the basis for another MySQL storage engine.
- Sun Microsystems acquired MySQL AB on 26 February 2008 .





# History of MySQL



- In [April 2009](#), [Oracle Corporation](#) entered into an agreement to purchase [Sun Microsystems](#). Sun's board of directors unanimously approved the deal, it was also approved by Sun's shareholders, and by the U.S. government on [August 20, 2009](#).
- It is the world's most popular open source database and has won the Linux Journal Readers' Choice Award on a number of occasions.
- It is used in [Google](#), [Wikipedia](#), [Facebook](#) and [Yahoo!](#)



# You Are in Good Company!

 <p><b>Web / Web 2.0</b></p>	 <p><b>OEM / ISV's</b></p>	
 <p><b>OnDemand, SaaS, Hosting</b></p>	 <p><b>Telecommunications</b></p>	 <p><b>Enterprise 2.0</b></p>

Open-source is powering the World!



# MySQL Tool (Example)

- MySQL Workbench
  - Visual database design tool
    - Used to efficiently design, manage and document databases





**Why MySQL?**

# Why MySQL?



- Ease of Use
- Source Code
- Low Cost
- Availability of Support
- Portability
  - MySQL can be used on many different Unix systems as well as under Microsoft Windows.

# 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
  - QNX
  - Open BSD
  - SGI Irix
  - Novell NetWare
  - Source Code
  - Special Builds



Let's start MySQL 😊  
*Installation*

# Installing & Logging



- For Ubuntu/Debian :

```
$ Sudo apt-get install mysql-server mysql-client
```

- To Log into MySQL:

```
$ mysql -h hostname -u username -p
```





# Data Definition Language “DDL”



# Creating Database

# Database Objects

- Objects belonging to a database
  - Table data and record of relationships
  - Views
  - Index
  - Stored procedures / Functions
  - Triggers
  - Events

# Creating Databases (1/2)

- CREATE DATABASE statement

## Examples

```
CREATE DATABASE mydb;
```

```
CREATE DATABASE IF NOT EXISTS mydb;
```

- Optional clauses
  - CHARACTER SET (column setting)
  - COLLATE

## Example

```
CREATE DATABASE mydb CHARACTER SET utf8 COLLATE utf8_danish_ci;
```

## Creating Databases (2/2)

- Using a database in **mysql**

```
USE mydb;
```

- Displaying a database creation

```
SHOW CREATE DATABASE world\G
```

```
***** 1. row *****  
Database: world  
Create Database: CREATE DATABASE `world`  
/*!40100 DEFAULT CHARACTER SET latin1 */
```



# Altering Databases

- ALTER DATABASE statement
- Examples

```
ALTER DATABASE mydb COLLATE utf8_polish_ci;
```

```
ALTER DATABASE mydb CHARACTER SET latin1 COLLATE  
latin1_swedish_ci;
```

- Affects new tables only





# Dropping Databases

- DROP DATABASE statement
- Examples

```
DROP DATABASE mydb;
```

```
DROP DATABASE IF EXISTS mydb;
```

- Full or empty databases dropped



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



# Using the Right Database



- To Select a database

```
use db_name;
```

- Alternatively, you can do that when you log in:

```
mysql -D dbname -h hostname -u  
username -p
```

- You can also use qualified names that identify both the database and the table:

```
SELECT * FROM db_name.tbl_name;
```

- To Know which database is selected:

```
SELECT DATABASE();
```



# Tables

# Creating a Table

- General syntax for creating a table

```
CREATE TABLE <table> (  
  <column name> <column type> [<column options>],  
  [<column name> <column type> [<column options>], ..., ]  
  [<index list>]  
  [<table options>];
```

- 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)  
)ENGINE = InnoDB COMMENT='Lists Language Spoken';
```

# Table Properties

- Add table options to CREATE TABLE statement
- Several options available

- ENGINE
- COMMENT
- CHARACTER SET
- COLLATE



**COLLATE** can also be used in **SELECT** queries.

- Example

```
CREATE TABLE CountryLanguage (  
  
    ) ENGINE=InnoDB COMMENT='Lists Language Spoken' CHARSET  
    utf8 COLLATE utf8_unicode_ci;
```

## Column Options (1/2)

- Add column options to CREATE TABLE statement
- Several options available
  - NULL
  - NOT NULL
  - DEFAULT
  - AUTO\_INCREMENT
- Constraints
  - Restrictions placed on one or more columns
  - Primary Key
  - Foreign Key
  - Unique



## Column Options (2/2)

- Column options example

```
CREATE TABLE City (  
    ID int(11) NOT NULL AUTO_INCREMENT,  
    Name char(35) NOT NULL DEFAULT '',  
    CountryCode char(3) NOT NULL DEFAULT '',  
    District char(20) NOT NULL DEFAULT '',  
    Population int(11) NOT NULL DEFAULT '0',  
    PRIMARY KEY (ID)  
) ENGINE=InnoDB CHARSET=latin1
```

# SHOW CREATE TABLE

- Viewing the exact statement used to create a table
- Example

```
SHOW CREATE TABLE City\G
```

```
***** 1. row *****
```

```
Table: City
```

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



# Creating Tables from Existing Tables

- **CREATE TABLE...SELECT** will create a new table to fit and store the result set returned by the **SELECT**

```
CREATE TABLE CityCopy1 AS SELECT * FROM City;
```

# Creating Tables from Existing Tables

- **CREATE TABLE LIKE** creates a structurally equivalent table (alas no foreign keys), but does not copy any data

## Example

```
CREATE TABLE t
  (i INT NOT NULL AUTO_INCREMENT,
   PRIMARY KEY (i))
ENGINE = InnoDB;
```

```
CREATE TABLE copy1 SELECT * FROM t WHERE 0;
```

```
CREATE TABLE copy2 LIKE t;
```

# Add a Column

- Use an ALTER TABLE statement *with* ADD
- Example

```
ALTER TABLE City ADD COLUMN LocalName VARCHAR(35) CHARACTER SET utf8
NOT NULL DEFAULT '' COMMENT 'The local name of this City';
```

- Structure Change

```
DESCRIBE City;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	auto_increment
Name	char(35)	NO			
Population	int(11)	NO		0	
<b>LocalName</b>	<b>varchar(35)</b>	<b>NO</b>			

## Remove a Column

- Use an ALTER TABLE statement *with* DROP
- Example

```
ALTER TABLE City DROP COLUMN LocalName;
```



# Renaming Tables

- Use an ALTER TABLE statement *with* RENAME
- Examples

```
ALTER TABLE t1 RENAME TO t2;
```

```
RENAME TABLE t1 TO t2;
```

```
RENAME TABLE t1 TO tmp, t2 TO t1, tmp TO t2;
```

# The DROP TABLE Command

- Remove a table
- Full or empty table
- **IF EXISTS** to avoid error



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

- Examples:

```
DROP TABLE table1;
```

```
DROP TABLE IF EXISTS table1;
```



# Creating Foreign Key Constraints

- Foreign keys constraints may be specified as part of the CREATE TABLE syntax

```
CREATE TABLE City (  
    ID INT NOT NULL, Name CHAR(35) NOT NULL,  
    CountryCode CHAR(3) NOT NULL, District CHAR(20) NOT NULL,  
    Population INT NOT NULL, PRIMARY KEY (ID),  
    FOREIGN KEY (CountryCode) REFERENCES Country (Code)  
) ENGINE=InnoDB
```

- Alternatively they can be added to existing tables using an ALTER TABLE statement

```
ALTER TABLE City ADD FOREIGN KEY (CountryCode)  
REFERENCES Country (Code)
```

- The InnoDB engine is currently the only supported engine that provides a foreign key implementation

```
ALTER TABLE City ENGINE = InnoDB;
```

## Creating Foreign Key Constraints (3/3)

- Optional elements
  - The constraint name
  - DELETE rule - specifies what should happen to the referencing rows in case a referenced row is removed
    - **CASCADE** means that the **DELETE** must be propagated to any referencing rows
    - **NO ACTION** means that a **DELETE** of a row from the referenced table must not occur if there are still referencing rows
    - **RESTRICT** means the same as **NO ACTION**
    - **SET NULL** means that the referencing columns in the referencing rows are changed to **NULL**
  - UPDATE rule - specifies what should happen to the referencing rows in case a referenced row is changed
    - Uses similar rules as those used for DELETE

# Comments on Database Objects

- Table comments
  - Comments can be added to the **CREATE TABLE** statement with the **COMMENT** keyword

```
CREATE TABLE `CountryLanguage` (  
  
    ) ENGINE=MyISAM COMMENT 'Lists Languages Spoken'
```

- Column comments
  - Column comments can be included in **CREATE TABLE** statements too

```
CREATE TABLE `CountryLanguage` (  
    CountryCode CHAR(3) NOT NULL  
        COMMENT 'The code that identifies the Country',  
    Language CHAR(30) NOT NULL  
        COMMENT 'The name of the language spoken in the  
        Country',  
    ...)
```



# Data Types

# Numeric Data Types

- Store numeric data
- Types
  - Integer
  - Floating-Point
  - Fixed-Point
  - BIT
- Precision and scale

# Integer Types

- Whole numbers
- Types
  - TINYINT
  - SMALLINT
  - MEDIUMINT
  - INT
  - BIGINT
- Example
  - **World** database, **City** table, **Population** column  
Population **INT (11)**
  - Largest value output (uses 8, 11 allowed)  
**10500000**



Integer Syntax:

**INT (M)**



# Integer Type Comparison

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

# 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** entity  
GNP **FLOAT(10,2)**
  - Largest value output (uses 7, 10 allowed; 2 to right of decimal)  
8510700.00



Syntax:

**FLOAT (M,D)**

# Float Type Comparison

Column Type	Storage	Range
<b>FLOAT</b>	4 bytes	-3.402823466E+38 to -1.175494351E-38, 0 and 1.175494351E-38 to 3.402823466E+38
<b>DOUBLE REAL DOUBLE PRECISION</b>	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)**

# Character String Data Types

- Sequence of alphanumeric characters
- Used to store text or integer data
- Factors to consider when choosing type

Comparison Values	Type	Description
Text	<b>CHAR</b>	Fixed-length character string
	<b>VARCHAR</b>	Variable-length character string
	<b>TEXT</b>	Variable-length character string
Integer	<b>ENUM</b>	Enumeration consisting of a fixed set of legal values
	<b>SET</b>	Set consisting of a fixed set of legal values

## Text Types (1/2)

- CHAR/VARCHAR
  - CHAR
  - VARCHAR
- Example
  - **World** database, **CountryLanguage** table, **Language** entity  
Language **CHAR(30)**
  - Largest value output (uses 25, 30 allowed)  
Southern Slavic Languages

## Text Types (2/2)

- TEXT
  - TINYTEXT
  - TEXT
  - MEDIUMTEXT
  - LONGTEXT

# Text Type Summary

Type	Storage Required	Maximum Length
<b>CHAR(<i>M</i>)</b>	<i>M</i> characters	255 characters
<b>VARCHAR(<i>M</i>)</b>	#characters plus 1 or 2 bytes	65,535 bytes (subject to limitations)
<b>TINYTEXT</b>	#characters + 1 byte	255 bytes
<b>TEXT</b>	#characters + 2 bytes	65,535 bytes
<b>MEDIUMTEXT</b>	#characters + 3 bytes	16,777,215 bytes
<b>LONGTEXT</b>	#characters + 4 bytes	4,294,967,295 bytes



# Structured Character String Types

- ENUM

- Enumeration

- Example

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

- SET

- List of string values

- Example

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

# Character Set and Collation Support (1/3)

- Character set is a named encoded character Repertoire
  - Governed by Rules of Collation
- Collation is a names collating sequence
  - Defines character sort order

## Character Set and Collation Support (2/3)

- MySQL offers several character sets
  - Proper choice can make a big performance impact
  - Use **SHOW CHARACTER SET** to view list

### SHOW CHARACTER SET;

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3

## Character Set and Collation Support (3/3)

- A character set may have several collations
  - Use **SHOW COLLATION** to view available collations

**SHOW COLLATION LIKE 'latin1%';**

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5			0
latin1_swedish_ci	latin1	8	Yes	Yes	1
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	1
latin1_general_ci	latin1	48			
latin1_general_cs	latin1	49			
latin1_spanish_ci	latin1	94			



# Temporal Data Types (1/2)

- 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

## Temporal Data Types (2/2)

Type	Storage Required	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	1 byte	1901 to 2155 (for YEAR(4)), 1970 to 2069 (for YEAR(2))





# Data Manipulation Language “DML”

# The INSERT Statement

- The INSERT statement is a common method for adding new rows of data into a table

```
INSERT INTO table_name (column_list) VALUES (row_list);
```

## Example:

```
INSERT INTO City (ID, Name, CountryCode)  
VALUES (NULL, 'Essaouira', 'MAR'),  
        (NULL, 'Sankt-Augustin', 'DEU');
```



## INSERT ... SET

- The INSERT ... SET clause can also be used to indicate column names and values

```
INSERT INTO City (ID, Name, CountryCode)
VALUES (NULL, 'Essaouira', 'MAR'),
       (NULL, 'Sankt-Augustin', 'DEU');
```

- The above example can also be written with SET as follows;

```
INSERT INTO City SET ID=NULL, Name='Essaouira',
CountryCode='MAR';
```

```
INSERT INTO City SET ID=NULL, Name='Sankt-Augustin',
CountryCode='DEU';
```

# INSERT ... SELECT

- The INSERT...SELECT syntax is useful for copying rows from an existing table, or (temporarily) storing a result set from a query

```
INSERT INTO Top10Cities (ID, Name, CountryCode)
SELECT ID, Name, CountryCode FROM City
ORDER BY Population DESC LIMIT 10;
```

# The DELETE Statement (1/2)

- Emptying a table completely

```
DELETE FROM table_name
```

- Remove specific rows of data

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

- Example

```
DELETE FROM CountryLanguage WHERE IsOfficial='F'
```

- The DELETE statement removes entire rows
  - Does not include a specification of columns

# The UPDATE Statement

- Modifies contents of existing rows

```
UPDATE table_name SET column=expression(s)  
[WHERE where_condition] [ORDER BY...] [LIMIT row_count];
```

- Use with the SET clause for column assignments
- Optionally use WHERE
- Example

```
UPDATE Country SET Population = Population * 1.1;  
Query OK, 232 rows affected (#.## sec)  
Rows matched: 239 Changed: 232 Warnings:0
```



# The UPDATE Statement

## • Examples

```
UPDATE people
SET id=id-1;
```

*Does not put the id's in order  
After subscription occurs  
(4 to 3, 3 to 2) ...*



id	name	age
2	Susan	15
1	Victor	21
3	Victor	31

```
UPDATE people
SET id=id-1
ORDER BY id;
```

*Solves ordering issue...*



id	name	age
1	Victor	21
2	Susan	15
3	Victor	31

```
UPDATE people
SET name='Vic'
WHERE name='Victor'
LIMIT 1;
```

*After id renumbering is finalized,  
this update changes one name and  
limits output to only changed row...*



id	name	age
1	Vic	21

1 row in set (0.00 sec)

# The REPLACE Statement (1/2)

- MySQL extension to SQL standard
- Exactly the same as INSERT
  - Except when it is a PRIMARY KEY or UNIQUE constraint

- General syntax

```
REPLACE INTO table_name (column_list) VALUES (value_list);
```

- Example

```
REPLACE INTO people (id,name,age) VALUES (12,'Bruce',25);
```

- *Only* useful with PRIMARY KEY or UNIQUE

## The REPLACE Statement (2/2)

- Returns sum of rows deleted and inserted
- REPLACE algorithm
  - Try to insert the new row into the table
  - While the insertion fails because a duplicate-key error occurs for a primary key or unique index:
    - Delete from the table the conflicting row that has the duplicate key value
    - Try again to insert the new row into the table



# The TRUNCATE TABLE Statement

- Always removes all records
- General syntax

**TRUNCATE TABLE** *table\_name*;

- DELETE vs. TRUNCATE TABLE

DELETE	TRUNCATE TABLE
Can delete specific rows with WHERE	Cannot delete specific rows, deletes <i>all</i> rows
Usually executes more slowly	Usually executes more quickly
Returns a true row count	May return a row count of zero
Transactional	May reset AUTO_INCREMENT
	Not Transactional







# Transactions

# Transaction Control Statements

- **START TRANSACTION (or BEGIN)**
  - Begins a new transaction
- **COMMIT**
  - Commits the current transaction, making its changes permanent
- **ROLLBACK**
  - Rolls back the current transaction, canceling its changes
- **SET AUTOCOMMIT**
  - Disables or enables the default autocommit mode for the current connection

# Implicit COMMIT's

- COMMIT explicitly commits the current transaction
- Other statements that cause commit's
  - **START TRANSACTION**
  - **SET AUTOCOMMIT = 1** (or ON)
- Statements that have the potential to cause commit's
  - Data definition statements (**ALTER**, **CREATE**, **DROP**)
  - Data access and user management statements (**GRANT**, **REVOKE**, **SET PASSWORD**)
  - Locking statements (**LOCK TABLES**, **UNLOCK TABLES**)
- DML statements that cause implicit commit's
  - **TRUNCATE TABLE**, **LOAD DATA INFILE**

# Transaction Demo: ROLLBACK

**START TRANSACTION;**

**SELECT** name **FROM** City **WHERE** id=3803;

```
+-----+
| name   |
+-----+
| San Jose |
+-----+
```

**DELETE FROM** City **WHERE** id=3803;

Query OK, 1 row affected (#.## sec)

**SELECT** name **FROM** City **WHERE** id=3803;

Empty set (#.## sec)

**ROLLBACK;**

**SELECT** name **FROM** City **WHERE** id=3803;

```
+-----+
| name   |
+-----+
| San Jose |
+-----+
```



# Data Retrieval Language “DRL”

# The SELECT Statement (1/2)

- Most commonly used command for queries
- Retrieves rows from tables in a database
- General syntax

```
SELECT [<clause options>] <column list> [FROM] <table>  
      [<clause options>];
```

# The SELECT Statement (2/2)

- Examples

```
SELECT Name FROM Country;
```

```
+-----+
| Name |
+-----+
| Afghanistan |
| Netherlands |
:           :
| French Southern Territories |
| Unites States Minor Outlying Islands |
+-----+
```

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

```
SELECT 1+2;
```

```
+-----+
| 1+2 |
+-----+
| 3 |
+-----+
```

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



# Basic Uses of SELECT

- Clauses used to yield specific results

- DISTINCT
- FROM
- WHERE
- ORDER BY
- LIMIT

- Syntax example:

```
SELECT DISTINCT values_to_display  
FROM table_name  
WHERE expression  
ORDER BY how_to_sort  
LIMIT row_count;
```

## SELECT Tips



- Commands (and clauses) are not case-sensitive (unless host is set as such)
- Use \c to abort a command
- Use \G in place of the ;) to return results by the row
- Use of \* (all row data) can give random results and waste resources
- Keep clauses in proper order of precedence



# SELECT/DISTINCT

- Removes duplicate rows
- Example

```
SELECT Continent FROM Country;
```

Continent
Asia
Europe
North America
Europe
Africa
Oceania
Europe
Africa
:
Antarctica
Oceania

239 rows in set (#.## sec)

--->

```
SELECT DISTINCT Continent  
FROM Country;
```

Continent
Asia
Europe
North America
Africa
Oceania
South America
Antarctica

7 rows in set (#.## sec)

# SELECT/WHERE

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

# SELECT/WHERE

- Example

```
SELECT Name, Population FROM Country
      WHERE Population > 50000000 AND
      (Continent = 'Europe' OR Code = 'USA');
```

```
+-----+-----+
| Name                | Population |
+-----+-----+
| United Kingdom      | 59623400  |
| Italy                | 57680000  |
| France              | 59225700  |
| Germany              | 82164700  |
| Ukraine              | 50456000  |
| Russian Federation  | 146934000 |
| United States        | 278357000 |
+-----+-----+
7 rows in set (0.31 sec)
```

# SELECT/WHERE

- Example

```
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/ORDER BY

- Will return output rows in a specific order
- Example

```
SELECT Name FROM Country ORDER BY Name;
```

```
+-----+
| Name          |
+-----+
| Afghanistan   |
| Albania       |
| Algeria       |
| American Samoa |
| Andorra       |
| Angola        |
| Anguilla      |
| Antarctica    |
| Antigua and Barbuda |
```

# SELECT/ORDER BY

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

```
SELECT Name FROM Country ORDER BY Name DESC;
```

```
+-----+
| Name          |
+-----+
| Zimbabwe     |
| Zambia        |
| Yugoslavia    |
| Yemen         |
| Western Sahara |
| Wallis and Futuna |
| Virgin Islands, U.S. |
```

# SELECT/ORDER BY

- Sort multiple columns simultaneously
- Example

```
SELECT Name, Continent FROM Country
ORDER BY Continent DESC, Name ASC;
```

Name	Continent
<b>Argentina</b>	<b>South America</b>
Bolivia	South America
Brazil	South America
Chile	South America
:	:
Uzbekistan	Asia
Vietnam	Asia
Yemen	Asia

239 rows in set (#.## sec)



# SELECT/LIMIT

- Specify number of rows output
- Example

```
SELECT Name FROM Country LIMIT 8;
```

```
+-----+
| name                |
+-----+
| Afghanistan         |
| Netherlands         |
| Netherlands Antilles |
| Albania             |
| Algeria             |
| American Samoa      |
| Andorra             |
| Angola              |
+-----+
8 rows in set (#.## sec)
```



**MySQL specific keyword.**



# SELECT/LIMIT

- Specify skip rows
- Example

```
SELECT name, population FROM country LIMIT 20,8;
```

+-----+-----+	
name	population
+-----+-----+	
<b>Belgium</b>	<b>10239000</b>
Belize	241000
Benin	6097000
Bermuda	65000
Bhutan	2124000
Bolivia	8329000
Bosnia and Herzegovina	3972000
Botswana	1622000
+-----+-----+	

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

# SELECT/LIMIT)

- Use with ORDER BY for ordered output
- Examples

```
SELECT * FROM t ORDER BY id LIMIT 1;
```

```
SELECT name, population FROM country
ORDER BY population DESC LIMIT 5;
```

name	population
China	1277558000
India	1013662000
United States	278357000
Indonesia	212107000
Brazil	170115000

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



# Why Use Aggregate Functions? (1/2)

- Summary functions
  - Perform summary operations on a set of values
- Returns single value based on group of values
  - Turn many rows into one value
- Only NON NULL

Aggregate Functions:	Definition:
<code>MIN ()</code>	Find the smallest value
<code>MAX ()</code>	Find the largest value
<code>SUM ()</code>	Summarize numeric value totals
<code>AVG ()</code>	Summarize numeric value averages
<code>STD ()</code>	Returns the population standard deviation
<code>COUNT ()</code>	Counts rows, non-null values, or the number of distinct values
<code>GROUP_CONCAT ()</code>	Concatenates a set of strings to produce a single string

# Why Use Aggregate Functions? (2/2)

- Examples

```
SELECT COUNT(*) FROM Country;
```

```
+-----+
```

```
| COUNT(*) |
```

```
+-----+
```

```
|      239 |
```

```
+-----+
```

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

```
SELECT COUNT(Capital) FROM Country;
```

```
+-----+
```

```
| COUNT(Capital) |
```

```
+-----+
```

```
|           232 |
```

```
+-----+
```

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

# Grouping with SELECT/GROUP BY

- Use GROUP BY for sub-group
- Based on values on one + columns of rows
- Example

```
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 (#.## sec)



# SQL expressions

## Using LIKE for Pattern Matching (1/2)

- Comparisons based on similarity
- Use LIKE pattern-matching operator
  - Percent character '%'
  - Underscore character '\_'
- NOT LIKE opposite comparison

## Using LIKE for Pattern Matching (2/2)

- Examples (LIKE vs. NOT LIKE)

```
SELECT Name FROM Country
WHERE Name LIKE 'United%';
```

Name
United Arab Emirates
United Kingdom
United States Minor Outlying Isl.
United States

4 rows in set (#.## sec)

```
SELECT Name FROM Country
WHERE Name NOT LIKE 'United%';
```

Name
Aruba
Zambia
Zimbabwe

235 rows in set (#.## sec)





## Built in functions

# Built in Functions



- The Multi row functions are categorized according to the mode of action and argument's data type into the following :
  - Comparison Functions
  - Control Flow Functions
  - Cast Functions
  - Managing Different Types of Data



# Comparison functions

# Comparison Functions

- Test relative values or membership value
- Functions
  - LEAST( ) returns the smallest value from a set
  - GREATEST( ) returns the largest value from a set
- Examples

```
SELECT LEAST(4,3,8,-1,5), LEAST('cdef','ab','ghi');
```

LEAST(4,3,8,-1,5)	LEAST('cdef','ab','ghi')
-1	ab

```
SELECT GREATEST(4,3,8,-1,5),  
       GREATEST('cdef','ab','ghi');
```

GREATEST(4,3,8,-1,5)	GREATEST('cdef','ab','ghi')
8	ghi



# Control Flow functions

# Flow Control Functions

- Choose between different values based on the result of an expression
- IF() tests the expression
  - Examples

True

```
SELECT IF(1 > 0, 'YES', 'NO');
```

False

IF(1 > 0, 'YES', 'NO')
YES

1 row in set (0.00 sec)

# Flow Control Functions

- CASE/WHEN provides branching flow control
- General syntax

```
CASE case_expr  
  WHEN when_expr THEN result  
  [WHEN when_expr THEN result] ...  
  [ELSE result]  
END
```

# Flow Control Functions

- Example

```
SELECT name FROM country
ORDER BY
```

```
  CASE code
```

```
    WHEN 'USA' THEN 1
```

```
    WHEN 'CAN' THEN 2
```

```
    WHEN 'MEX' THEN 3
```

```
  ELSE 4 END, name;
```

```
+-----+
```

```
| name |
```

```
+-----+
```

```
| United States |
```

```
| Canada |
```

```
| Mexico |
```

```
| Afghanistan |
```

```
| Albania |
```

```
| Algeria |
```

```
| American Samoa |
```

```
| Zimbabwe |
```

```
+-----+
```

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





# Managing Data Types



# String functions

# String Functions

- INSTR(), LOCATE() and POSITION()

```
SELECT INSTR('Alice and Bob', 'and'),
LOCATE('and', 'Alice and Bob'),
POSITION('and' IN 'Alice and Bob')\G
*****1. row*****
      INSTR('Alice and Bob', 'and'): 7
      LOCATE('and', 'Alice and Bob'): 7
      POSITION('and' IN 'Alice and Bob'): 7
```

```
SELECT LOCATE(' ', 'Alice and Bob', 7);
+-----+
| LOCATE(' ', 'Alice and Bob', 7) |
+-----+
|                                10 |
+-----+
```

# String Functions

- Perform operations on strings
- LENGTH()/CHAR\_LENGTH() examples

```
SELECT LENGTH('MySQL'), CHAR_LENGTH('MySQL');
```

LENGTH('MySQL')	CHAR_LENGTH('MySQL')
5	5

```
SELECT LENGTH(CONVERT('MySQL' USING ucs2)) AS length,  
       CHAR_LENGTH(CONVERT('MySQL' USING ucs2)) AS c_length;
```

length	c_length
10	5

# String Functions

- CONCAT() and CONCAT\_WS() examples

```
SELECT CONCAT('See','spot','run');
```

```
+-----+
| CONCAT('See','spot','run') |
+-----+
| Seespotrun                  |
+-----+
```

```
SELECT CONCAT_WS(' ','See','spot','run');
```

```
+-----+
| CONCAT_WS(' ','See','spot','run') |
+-----+
| See spot run                      |
+-----+
```

# String Functions

- SUBSTRING()

```
SELECT SUBSTRING('Alice and Bob', 1, 5);
+-----+
| SUBSTRING('Alice and Bob', 1, 5) |
+-----+
| Alice                             |
+-----+
```

# String Functions

- LEFT() and RIGHT()

```
SELECT LEFT('Alice and Bob', 5);
```

```
+-----+  
| LEFT('Alice and Bob', 5) |  
+-----+  
| Alice                    |  
+-----+
```

```
SELECT RIGHT('Alice and Bob', 3);
```

```
+-----+  
| RIGHT('Alice and Bob', 3) |  
+-----+  
| Bob                      |  
+-----+
```

# String Functions

- REPLACE()

```
SELECT REPLACE('Alice & Bob', '&', 'and');
+-----+
| REPLACE('Alice & Bob', '&', 'and') |
+-----+
| Alice and Bob                      |
+-----+
```





# Numeric functions

# Numeric Functions (1/5)

- Mathematical operations
- Common functions
  - TRUNCATE()
  - FLOOR()
  - CEILING()
  - ROUND()
  - ABS()
  - SIGN()
  - SIN(), COS(), TAN()

## Numeric Functions (2/5)

- **ROUND** examples

```
SELECT ROUND(28.5), ROUND(-28.5);
```

+	-----	+	-----	+
	ROUND(28.5)		ROUND(-28.5)	
+	-----	+	-----	+
	29		-29	
+	-----	+	-----	+

## Numeric Functions (3/5)

- FLOOR/CEILING examples

```
SELECT FLOOR(-14.7), FLOOR(14.7);
```

```
+-----+-----+
| FLOOR(-14.7) | FLOOR(14.7) |
+-----+-----+
|           -15 |           14 |
+-----+-----+
```

```
SELECT CEILING(-14.7), CEILING(14.7);
```

```
+-----+-----+
| CEILING(-14.7) | CEILING(14.7) |
+-----+-----+
|           -14 |           15 |
+-----+-----+
```

## Numeric Functions (4/5)

- ABS/SIGN examples

```
SELECT ABS(-14.7), ABS(14.7);
```

ABS(-14.7)	ABS(14.7)
14.7	14.7

```
SELECT SIGN(-14.7), SIGN(14.7), SIGN(0);
```

SIGN(-14.7)	SIGN(14.7)	SIGN(0)
-1	1	0



# Date/Time functions

# Temporal Functions (1/5)

- Time, Date, Year
- Perform many operations
- Functions

Functions	Definition
<code>NOW ()</code>	<i>Current date and time as set on the client host ( in <b>DATETIME</b> format)</i>
<code>CURDATE ()</code>	<i>Current date as set on the client host ( in <b>DATE</b> format)</i>
<code>CURTIME ()</code>	<i>Current time as set on the client host ( in <b>TIME</b> format)</i>
<code>YEAR ()</code>	<i>Year in <b>YEAR</b> format, per value indicated (can use <b>NOW()</b> function within parenthesis to get current year per client)</i>
<code>MONTH ()</code>	<i>Month of the year in integer format, per value indicated (can use <b>NOW()</b> as above)</i>
<code>DAYOFMONTH () or DAY ()</code>	<i>Day of the month in integer format, per value indicated (can use <b>NOW()</b> as above)</i>
<code>DAYNAME ()</code> <i>(English)</i>	<i>Day of the week in string format, per value indicated (can use <b>NOW()</b> as above)</i>
<code>HOURL ()</code>	<i>Hour of the Day in integer format, per value indicated (can use <b>NOW()</b> as above)</i>
<code>MINUTE ()</code>	<i>Minute of the Day in integer format, per value indicated (can use <b>NOW()</b> as above)</i>
<code>SECOND ()</code>	<i>Second of the Minute in integer format, per value indicated (can use <b>NOW()</b> as above)</i>
<code>GET _FORMAT ()</code>	<i>Returns a date format string, per values indicated for date-type and international format.</i>

## Temporal Functions (2/5)

- View current date and time

```
SELECT NOW();
```

NOW()
2004-04-30 11:59:15

1 row in set (#.## sec)

- View date format

```
SELECT GET_FORMAT(DATE, 'EUR');
```

GET_FORMAT(DATE, 'EUR')
%d.%m.%Y

1 row in set (#.## sec)



# Temporal Functions (3/5)

- Extracting parts of date/time examples

```
SELECT YEAR('2010-04-15'), MONTH('2010-04-15'), DAYOFMONTH('2010-04-15');
+-----+-----+-----+
| YEAR('2010-04-15') | MONTH('2010-04-15') | DAYOFMONTH('2010-04-15') |
+-----+-----+-----+
|                2010 |                4      |                15      |
+-----+-----+-----+
```

```
SELECT DAYOFYEAR('2010-04-15');
+-----+
| DAYOFYEAR('2010-04-15') |
+-----+
|                105      |
+-----+
```

```
SELECT HOUR('09:23:57'), MINUTE('09:23:57'), SECOND('09:23:57');
+-----+-----+-----+
| HOUR('09:23:57') | MINUTE('09:23:57') | SECOND('09:23:57') |
+-----+-----+-----+
|                9 |                23  |                57  |
+-----+-----+-----+
```

# Temporal Functions (5/5)

- Current dates/times examples

```
SELECT CURRENT_DATE(),
        CURRENT_TIME(),
        CURRENT_TIMESTAMP();
```

CURRENT_DATE()	CURRENT_TIME()	CURRENT_TIMESTAMP()
2005-05-31	21:40:18	2005-05-31 21:40:18

## NULL-Related Functions (1/2)

- Specifically for use with NULL
- ISNULL()/IFNULL() examples

```
SELECT ISNULL(NULL), ISNULL(0), ISNULL(1);
```

ISNULL(NULL)	ISNULL(0)	ISNULL(1)
1	0	0

```
SELECT IFNULL(NULL, 'a'), IFNULL(0, 'b');
```

IFNULL(NULL, 'a')	IFNULL(0, 'b')
a	0

## NULL-Related Functions (2/2)

- CONCAT with NULL examples

```
SELECT CONCAT('a','b'), CONCAT('a',NULL,'b');
```

CONCAT('a','b')	CONCAT('a',NULL,'b')
ab	NULL

```
SELECT CONCAT_WS('/', 'a', 'b'),
       CONCAT_WS('/', 'a', NULL, 'b');
```

CONCAT_WS('/', 'a', 'b')	CONCAT_WS('/', 'a', NULL, 'b')
a/b	a/b



# Comments in SQL Statements (1/2)

- MySQL supports three forms of syntax

- '#'

- '/\*' or '/\*!'

- '--'

- Examples

```
/* this is a comment */
```

```
/*  
  this  
  is a  
  comment,  
  too  
*/
```

## Comments in SQL Statements (2/2)

- C-style comments
- Examples

```
CREATE TABLE t (i INT) /*! ENGINE = MEMORY */;
```

```
SHOW /*!50002 FULL */ TABLES;
```

```
CREATE TABLE `CountryLanguage` (  
    ) ENGINE=MyISAM COMMENT 'Lists Languages Spoken'
```