MySQL for Developers

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

Day 2

- DRL
    - Joins
    - Subquery
- Views
- Indexes
- Meta Data
- DCL

Let’s start MySQL ☺

Installation

Installing

- For Ubuntu/Debian :

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

Logging

- 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 EXIST 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 */
```

Character Set & Collations

A **character set** is a set of symbols and encodings.

A **collation** is a set of rules for comparing characters

in a character set.

Character Set & Collations (Example)

Suppose that we have an alphabet with four letters:

A, B, a, b.

We give each letter a number:

A = 0, B = 1, a = 2, b = 3

The letter A is a symbol

the number 0 is the encoding for A,

the combination of all four letters and their encodings is a **character set.**

Character Set & Collations Example

To compare two string values, A and B.

1- look at the encodings: 0 for A and 1 for B.
Because 0 is less than 1, we say A is less than B.

The **collation** is a set of rules (only one rule in this case): "compare the encodings"

We call this simplest of all possible collations a **binary collation.**

Character Set & Collations Example

2- if we want to say that the lowercase and uppercase letters are equivalent?
Then we would have at least two rules:

- treat the lowercase letters a and b as equivalent to A and B;
- then compare the encodings.

We call this a **case-insensitive collation**. It is a little more complex than a binary collation.

Character Set & Collations

MySQL can do these things for you:

- Store strings using a variety of character sets.
- Compare strings using a variety of collations.
- Mix strings with different character sets or collations in the same server, the same database, or even the same table.
- Enable specification of character set and collation at any level.

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;
```

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

- Full or empty databases dropped

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>];
```

Creating a Table

- 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
- Example

COLLATE can also be used in SELECT queries.

```
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
  - 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)
)
```

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=MyISAM 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 (no foreign keys), but does not copy any data

Example

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

```
CREATE TABLE copy1 SELECT * FROM t WHERE 0;
CREATE TABLE copy2 LIKE t;
```

Alter Table - Add a Column

```
ALTER TABLE City
ADD COLUMN LocalName VARCHAR(35)NOT NULL
COMMENT 'local name of 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       |                |
| LocalName   | varchar(35) | NO   |     |         |                |
+-------------+-------------+------+-----+---------+----------------+
```

Alter Table - Remove a Column

- Example

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

Alter Table - Modifying Columns

- Example

```
ALTER TABLE City
MODIFY ID BIGINT NOT NULL AUTO_INCREMENT;
```

Renaming Tables

- 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

```
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))
```

Creating Foreign Key Constraints

- 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

- **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 -** similar rules as those used for DELETE

Comments on Database Objects

- **Table comments**

```
CREATE TABLE `CountryLanguage` (
) COMMENT 'Lists Languages Spoken'
```

- **Column comments**

```
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
- 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
`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** entity

Syntax:

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

```
GNP  FLOAT(10,2)

8510700.00
```

Float Type Comparison

Column Type	Storage	Range
`FLOAT`	4 bytes	-3.402823466E+38 to -1.175494351E-38, 0 and 1.175494351E-38 to 3.402823466E+38

`DOUBLE`
`REAL`
`DOUBLE PRECISION`
	8 bytes	-1.7976931348623157E+308 to -2.2250738585072014E-308, 0 and 2.2250738585072014E-308 to 1.7976931348623157E+308

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	`CHAR`	Fixed-length character string
	`VARCHAR`	Variable-length character string
	`TEXT`	Variable-length character string
Integer	`ENUM`	Enumeration consisting of a fixed set of legal values
	`SET`	Set consisting of a fixed set of legal values

Text Types

- 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

- TEXT
  - TINYTEXT
  - TEXT
  - MEDIUMTEXT
  - LONGTEXT

Text Type Summary

Type	Storage Required	Maximum Length
CHAR(*M*)	*M* characters	255 characters
VARCHAR(*M*)	#characters plus 1 or 2 bytes	65,535 bytes (subject to limitations)
TINYTEXT	#characters + 1 byte	255 bytes
TEXT	#characters + 2 bytes	65,535 bytes
MEDIUMTEXT	#characters + 3 bytes	16,777,215 bytes
LONGTEXT	#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')
```

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

- Emptying a table completely

```
DELETE FROM table_name
```

- Remove specific rows of data

```
DELETE FROM table_name
[WHERE where_condition]
```

- **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
```

- 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 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 *all* 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

What is a Transaction? (1/2)

- In database programming, a transaction is a collection of data manipulation execution steps that are treated as a single unit of work
  - Execution steps are performed as if there were a single specialized command that accomplishes exactly that combination of actions

Non-Transactional Executions

Remove $1000 from account #10001

Write to database

Deposit $1000 into account #10243

Write to database

Transactional Executions

Remove $1000 from account #10001

Deposit $1000 into account #10243

Write to database

What is a Transaction? (2/2)

- All of the data manipulation steps must be carried out
- If any portion fails, action must be taken to:
  - Permanently retain those operations that did succeed
    - **- or -**
  - Disregard those operations that did succeed

Non-Transactional Executions

Remove $1000 from account #10001

Write to database

Deposit $1000 into account #10243

Transactional Executions

Remove $1000 from account #10001

Deposit $1000 into account #10243

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 changes made by transaction that complete successfully are recorded properly in database--Changes are *not* lost

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
  - **Set autocommit = 0;**

AUTOCOMMIT Mode (1/2)

- Determines how and when new transactions are started
- Autocommit enabled
  - A single SQL statement implicitly starts a new transaction by default
    - The transaction is automatically committed if the statement executes successfully
    - If the statement does not execute successfully, the transaction is automatically rolled back
  - Transactions can still be started explicitly using the START TRANSACTION statement
- Autocommit disabled
  - Transactions span multiple statements by default
  - Transactions can be explicitly committed or rolled back
  - A new transaction is implicitly started after termination of previous

AUTOCOMMIT Mode (2/2)

- Autocommit disabled
  - Transactions span multiple statements by default
  - Transactions must be explicitly committed or rolled back
  - A new transaction is implicitly started after successful termination of previous transaction
  - Unsuccessful statements will result in any potential changes by that statement being undone
    - The transaction continues to remain open until committed or rolled back as a whole

Controlling AUTOCOMMIT Mode (2/2)

- By default, autocommit is enabled
  - Disable if transactions that span multiple statements are required
- Server configuration default behavior can be changed

```
SET AUTOCOMMIT = 0
```

- Determining current autocommit setting

```
SELECT @@autocommit;
```

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**

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

View Available Storage Engines

- Check for a Transactional Storage Engine

```
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
```

Locking Concepts

- A Locking Mechanism Prevents Problems with Concurrent Data Access
- Locks are Managed By the Server
  - Allows access to one client and locks others out
- Locking Depends on Access Type
  - READ vs. WRITE

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

```
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 (#.## sec)
```

SELECT/ORDER BY

```
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

```
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 (#.## sec)
```

SELECT/LIMIT

```
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

- Use with ORDER BY for ordered output

```
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:
`MIN()`	Find the smallest value
`MAX()`	Find the largest value
`SUM()`	Summarize numeric value totals
`AVG()`	Summarize numeric value averages
`STD()`	Returns the population standard deviation
`COUNT()`	Counts rows, non-null values, or the number of distinct values
`GROUP_CONCAT()`	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

```
    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

String Expressions (1/3)

- Literal strings are quoted
  - Single or double quotes
  - ANSI_QUOTES sql mode special
- Data types
- Comparison operations

Operator:	Definition:
<	Less than
<=	Less than or equal to
=	Equal to
<=>	Equal to (works even for **NULL** values)
<> or !=	Not equal to
>=	Greater than or equal to
>	Greater than
BETWEEN <x AND Y>	Indicate a range of numerical values

String Expressions (2/3)

- Function examples

```
SELECT CONCAT('abc','def',REPEAT('X',3));
+-----------------------------------+
| CONCAT('abc','def',REPEAT('X',3)) |
+-----------------------------------+
| abcdefXXX                         |
+-----------------------------------+

SELECT 'abc' || 'def';
+---------------+
| 'abc' || 'def' |
+---------------+
|             0 |
+---------------+
1 row in set, 2 warnings (#.## sec)
```

String Expressions (3/3)

- Function examples *(continued)*

```
SET sql_mode = 'PIPES_AS_CONCAT';
    Query OK, 0 rows affected (#.## sec)

SELECT 'abc' || 'def';
    +----------------+
    | 'abc' || 'def' |
    +----------------+
    | abcdef         |
    +----------------+
    1 row in set (#.## sec)
```

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 (IF / Case)

- 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 (#.## sec)
```

Flow Control Functions

- CASE/WHEN provides branching flow control
- General syntax

```
CASE
  WHEN when_expr THEN result

  [WHEN when_expr THEN result] ...

  [ELSE result]
END
```

Managing Data Types

Managing Data Types

According to the input data type they can be classified into

String functions:

- ASCII() Functions
- CHAR_LENGTH(), CHARACTER_LENGTH(), and LENGTH() Functions
- CHARSET() and COLLATION() Functions
- CONCAT() and CONCAT_WS() Functions
- INSTR() and LOCATE() Functions
- LCASE(), LOWER(), UCASE(), and UPPER() Functions
- LEFT() and RIGHT() Functions
- REPEAT() and REVERSE() Functions
- SUBSTRING() Function

Managing Data Types

According to the input data type they can be classified into

- Numeric functions
  - CEIL(), CEILING(), and FLOOR() Functions
  - COT() Functions
  - MOD() Function
  - POW() and POWER() Functions
  - ROUND() and TRUNCATE() Functions
  - SQRT() Function

Managing Data Types

According to the input data type they can be classified into

- Date time functions

  - ADDDATE(), DATE_ADD(), SUBDATE(), DATE_SUB(),and EXTRACT() Functions
  - DATE(), MONTH(), MONTHNAME(), and YEAR() Functions
  - DATEDIFF() and TIMEDIFF() Functions
  - DAY(), DAYOFMONTH(), DAYNAME(), DAYOFWEEK(), and DAYOFYEAR() Functions
  - SECOND(), MINUTE(), HOUR(), and TIME() Functions

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
```

7

7

7

String Functions

- Perform operations on strings
- LENGTH()

```
SELECT LENGTH('MySQL')
```

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

- INSERT() and REPLACE()

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

SELECT INSERT('Alice and Bob', 6, 5, ', Carol & ');
+-------------------------------------------+
| INSERT('Alice and Bob', 6, 5, ', Carol & ') |
+-------------------------------------------+
| Alice, Carol & Bob                        |
+-------------------------------------------+
```

Numeric functions

Numeric Functions (1/4)

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

Numeric Functions (2/4)

- **ROUND** examples

```
SELECT ROUND(28.5), ROUND(-28.5);
+-------------+--------------+
| ROUND(28.5) | ROUND(-28.5) |
+-------------+--------------+
| 29          | -29          |
+-------------+--------------+
```

Numeric Functions (3/4)

- 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 (4/4)

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

Temporal Functions (2/5)

- View current date and time

```
SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2004-04-30 11:59:15 |
+---------------------+
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 |
+--------------------+---------------------+--------------------------+
```

Temporal Functions (3/5)

- Extracting parts of date/time examples

```
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 (4/5)

- Composite dates/times examples

```
SELECT MAKEDATE(2010,105);
+--------------------+
| MAKEDATE(2010,105) |
+--------------------+
| 2010-04-15         |
+--------------------+

SELECT MAKETIME(9,23,57);
+-------------------+
| MAKETIME(9,23,57) |
+-------------------+
| 09:23:57          |
+-------------------+
```

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

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

Comments in SQL Statements

- MySQL supports three forms of syntax
  - '#'
  - /* or /*!
  - --
- Examples

```
/* this is a comment */

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