MySQL for Developers

SQL-4501 Release 2.2

D61830GC10
Edition 1.0

ORACLE

Indexes

Creating Indexes

- Table *with* index

```
CREATE TABLE HeadOfState
(  ID              INT NOT NULL,
   LastName        CHAR(30) NOT NULL,
   FirstName       CHAR(30) NOT NULL,
   CountryCode     CHAR(3) NOT NULL,
   Inauguration    DATE NOT NULL,
   INDEX (Inauguration));
```

Creating Indexes

- Table with composite index

```
CREATE TABLE HeadOfState
(  ID              INT NOT NULL,
   LastName        CHAR(30) NOT NULL,
   FirstName       CHAR(30) NOT NULL,
   CountryCode     CHAR(3) NOT NULL,
   Inauguration    DATE NOT NULL,
   INDEX (LastName, FirstNam));
```

Creating Indexes

- Table with multiple indexes

```
CREATE TABLE HeadOfState
(  ID              INT NOT NULL,
   LastName        CHAR(30) NOT NULL,
   FirstName       CHAR(30) NOT NULL,
   CountryCode     CHAR(3) NOT NULL,
   Inauguration    DATE NOT NULL,
   INDEX (LastName, FirstNam),
   INDEX (Inauguration));
```

Creating Indexes

- Table *with UNIQUE* index

```
CREATE TABLE HeadOfState
(  ID              INT NOT NULL,
   LastName        CHAR(30) NOT NULL,
   FirstName       CHAR(30) NOT NULL,
   CountryCode     CHAR(3) NOT NULL,
   Inauguration    DATE NOT NULL,
   UNIQUE INDEX (ID));
```

Creating Indexes

- Primary key versus unique index
  - Primary key cannot contain NULL
  - Only one primary key is allowed per table
  - Primary key is a type of unique index
  - Unique Index is not always a primary key

Naming Indexes

- Include name just before column list

```
CREATE TABLE HeadOfState
(  ID            INT NOT NULL,
   LastName      CHAR(30) NOT NULL,
   FirstName     CHAR(30) NOT NULL,
   CountryCode   CHAR(3) NOT NULL,
   Inauguration DATE NOT NULL,
   INDEX NameIndex (LastName, FirstName),
   UNIQUE INDEX IDIndex (ID));
```

- Default name
- Primary key always named **PRIMARY**

Adding Indexes to Existing Tables

```
ALTER TABLE HeadOfState ADD PRIMARY KEY (ID);

ALTER TABLE HeadOfState ADD INDEX (LastName,FirstName);

ALTER TABLE HeadOfState ADD PRIMARY KEY (ID),
   ADD INDEX (LastName,FirstName);
```

Adding Indexes to Existing Tables

- **CREATE INDEX** examples
  - Must provide name for index
  - Only single index per statement

```
CREATE UNIQUE INDEX IDIndex

ON HeadOfState (ID);
```

```
CREATE INDEX NameIndex

ON HeadOfState (LastName,FirstName);
```

Dropping Indexes

- Dropping a **PRIMARY KEY** is easy

```
ALTER TABLE HeadOfState DROP PRIMARY KEY
```

- To drop another kind of index, you must specify its name

```
ALTER TABLE HeadOfState DROP INDEX NameIndex;
```

Dropping Indexes

```
DROP INDEX NameIndex ON t;

DROP INDEX `PRIMARY` ON t;
```

Creating Users
Data Control Language
“DCL”

Creating Users

- Create One or more users:

```
CREATE USER account
[IDENTIFIED BY [PASSWORD] 'password']
[,account [IDENTIFIED BY[PASSWORD]
'password'] ] ...
```

Example

```
CREATE USER 'open_source'@'localhost'
IDENTIFIED BY 'os123';
```

Grant - Revoke

- The `GRANT` and `REVOKE` commands enable you to give rights to and take them away from MySQL users at these four levels of privilege:
  - Global
  - Database
  - Table
  - Column

Grant

- The `GRANT` command creates users and gives them privileges. The general form is

```
GRANT privileges ON item
TO user_name [IDENTIFIED BY 'password']

[WITH GRANT OPTION]
```

Example

```
GRANT ALL
ON *.*
TO open_source IDENTIFIED BY 'os123'
WITH GRANT OPTION;

REVOKE ALL privileges, grant option
FROM open_source;
```

Revoke

The REVOKE Command : The opposite of GRANT is REVOKE. You use it to take privileges away from a user. It is similar to GRANT in syntax:

```
REVOKE privileges [(columns)]
ON item
FROM 'user_name'
```

- If you have given the WITH GRANT OPTION clause, you can revoke this (along with all other privileges) by adding:

```
REVOKE All , GRANT OPTION
FROM 'user_name'
```

Managing MySQL Users

- **CREATE USER, DROP USER,** and **RENAME USER** create, remove, and rename MySQL accounts.
- **GRANT** specifies account privileges (and creates accounts if they do not exist).
- **REVOKE** removes privileges from existing MySQL accounts.
- **SET PASSWORD** assigns passwords to existing accounts.
- **SHOW GRANTS** displays the privileges held by existing accounts.

Managing MySQL Users

```
SHOW GRANTS;

SHOW GRANTS FOR CURRENT_USER();
SHOW GRANTS FOR 'open_source'@localhost'

SET PASSWORD FOR 'open_source'@'localhost'
= PASSWORD('ositi');
```

Meta Data

Metadata Access Methods

- Information about database structure is metadata
- Methods
  - INFORMATION_SCHEMA
  - SHOW
  - DESCRIBE
- Metadata for several database aspects
- INFORMATION_SCHEMA was introduced in 5.0

SHOW Statements (1/8)

- MySQL supports many **SHOW** statements
- Commonly used statements
  - `SHOW DATABASES`
  - `SHOW [FULL] TABLES`
  - `SHOW [FULL] COLUMNS from table_name`
  - `SHOW INDEX from table_name`
  - `SHOW CHARACTER SET`
  - `SHOW COLLATION`

SHOW Statements (2/8)

- SHOW DATABASE example

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| menagerie          |
| mysql              |
| test               |
| world              |
+--------------------+
```

SHOW Statements (3/8)

- SHOW TABLES examples

```
SHOW TABLES;
+-----------------+
| Tables_in_world |
+-----------------+
| City            |
| Country         |
| CountryLanguage |
+-----------------+
```

```
SHOW TABLES FROM mysql;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |

| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
23 rows in set (#.## sec)
```

SHOW Statements (4/8)

- SHOW COLUMNS example

```
SHOW COLUMNS FROM CountryLanguage;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| CountryCode | char(3)       | NO   | PRI |         |       |
| Language    | char(30)      | NO   | PRI |         |       |
| IsOfficial  | enum('T','F') | NO   |     | F       |       |
| Percentage  | float(4,1)    | NO   |     | 0.0     |       |
+-------------+---------------+------+-----+---------+-------+
```

SHOW Statements (5/8)

- SHOW FULL COLUMNS example

```
SHOW FULL COLUMNS FROM CountryLanguage\G
*************************** 1. row ***************************
Field: CountryCode
Type: char(3)
Collation: latin1_swedish_ci
Null: NO
Key: PRI
Default:
Extra:
Privileges: select,insert,update,references
Comment:
*************************** 2. row ***************************
Field: Language
Type: char(30)
Collation: latin1_swedish_ci
Null: NO
Key: PRI
```

SHOW Statements (6/8)

- SHOW with LIKE example

```
SHOW DATABASES LIKE 'm%';
+---------------+
| Database (m%) |
+---------------+
| menagerie     |
| mysql         |
+---------------+
```

- SHOW with WHERE example

```
SHOW COLUMNS FROM Country WHERE `Default` IS NULL;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| IndepYear      | smallint(6) | YES  |     | NULL    |       |
| LifeExpectancy | float(3,1)  | YES  |     | NULL    |       |
| GNP            | float(10,2) | YES  |     | NULL    |       |
| GNPOld         | float(10,2) | YES  |     | NULL    |       |
| HeadOfState    | char(60)    | YES  |     | NULL    |       |
| Capital        | int(11)     | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
```

SHOW Statements (7/8)

- SHOW INDEX example

```
SHOW INDEX FROM City\G
*************************** 1. row ***************************
       Table: City
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 1
Column_name: ID
   Collation: A
 Cardinality: 4079
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
```

SHOW Statements (8/8)

- SHOW CHARACTER SET/COLLATION examples

```
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 |
```

```
SHOW COLLATION;
+---------------------+----------+-----+---------+----------+---------+
| Collation           | Charset  | Id  | Default | Compiled | Sortlen |
+---------------------+----------+-----+---------+----------+---------+
| big5_chinese_ci     | big5     |   1 | Yes     | Yes      |       1 |
| big5_bin            | big5     |  84 |         | Yes      |       1 |
| dec8_swedish_ci     | dec8     |   3 | Yes     |          |       0 |
```

DESCRIBE Statements

- Equivalent to SHOW COLUMNS
- Can be abbreviated as DESC

```
DESCRIBE table_name;
DESC table_name;
SHOW COLUMNS FROM table_name;
```

- DESCRIBE does not support FROM
- Shows INFORMATION_SCHEMA table information

```
DESCRIBE INFORMATION_SCHEMA.CHARACTER_SETS;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| CHARACTER_SET_NAME   | varchar(64) | NO   |     |         |       |
| DEFAULT_COLLATE_NAME | varchar(64) | NO   |     |         |       |
| DESCRIPTION          | varchar(60) | NO   |     |         |       |
| MAXLEN               | bigint(3)   | NO   |     | 0       |       |
+----------------------+-------------+------+-----+---------+-------+
```

INFORMATION_SCHEMA Database (1/2)

- Database/schema that serves as a central repository for metadata
- Virtual database
- Use `SELECT` to obtain information

INFORMATION_SCHEMA Database (2/2)

- Tables example

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'INFORMATION_SCHEMA'
ORDER BY TABLE_NAME;
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| KEY_COLUMN_USAGE                      |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
```

INFORMATION_SCHEMA Tables (1/3)

- Table contents
  - CHARACTER_SETS -- available character sets
  - COLLATIONS -- collations for each character set
  - COLLATION_CHARACTER_SET_APPLICABILITY -- which character set applies to each collation
  - COLUMNS -- columns in tables
  - COLUMN_PRIVILEGES -- column privileges held by MySQL user accounts
  - ENGINES -- storage engines
  - EVENTS -- scheduled events
  - FILES -- the files in which MySQL NDB Disk Data tables are stored
  - KEY_COLUMN_USAGE -- constraints on key columns

INFORMATION_SCHEMA Tables (2/3)

- Table contents
  - PARTITIONS -- table partitions
  - PLUGINS -- server plugins
  - PROCESSLIST -- which threads are running
  - REFERENTIAL_CONSTRAINTS -- foreign keys
  - ROUTINES -- stored procedures and functions
  - SCHEMATA -- databases
  - SCHEMA_PRIVILEGES -- database privileges held by MySQL user accounts
  - STATISTICS -- table indexes
  - TABLES -- tables in databases
  - TABLE_CONSTRAINTS -- constraints on tables

INFORMATION_SCHEMA Tables (3/3)

- Table contents
  - TABLE_PRIVILEGES -- table privileges held by MySQL user accounts
  - TRIGGERS -- triggers in databases
  - USER_PRIVILEGES -- global privileges held by MySQL user accounts
  - VIEWS -- views in databases

Displaying INFORMATION_SCHEMA Tables

- Can use all the normal `SELECT` features
  - Specify columns
  - Restrict rows with the WHERE clause
  - Group or Sort with GROUP BY and ORDER BY
  - Use joins, unions and subqueries
  - Can feed results in another table
  - Create views on top of INFORMATION_SCHEMA tables

INFORMATION_SCHEMA

- VIEWS table in database

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS
WHERE TABLE_NAME = 'CityView'
AND TABLE_SCHEMA = 'world'\G
```

```
*************************** 1. row ***************************
  TABLE_CATALOG: NULL
   TABLE_SCHEMA: world
     TABLE_NAME: CityView
VIEW_DEFINITION: select `world`.`City`.`ID` AS `ID`,
                 `world`.`City`.`Name`AS `Name` from `world`.`City`
   CHECK_OPTION: NONE
   IS_UPDATABLE: YES
```

Stored Routines

What is a Stored Routine?

- Set of SQL statements that can be stored in server
- Types
  - Stored procedures
    - A procedure is invoked using a call statement, and can only pass back values using output variables
  - Stored functions
    - A function can be called from inside a statement and can return a scalar value

Creating Procedures

```
drop procedure if exists display_emp_info;
delimiter $
CREATE PROCEDURE display_emp_info(p_id integer)
BEGIN
   Select ename, salary
   from emp
    where id = p_id;
END$
delimiter ;
```

Invoking Procedure

```
Call display_emp_info(1);
```

Creating Function

```
drop function if exists tax;

delimiter $

CREATE FUNCTION tax(p_id integer)

RETURNS int(11)

BEGIN

  RETURN p_id * 0.1 ;

END$

delimiter ;
```

Invoking Function

```
Select Tax (1000);

Select Tax(Salary) from emp;
```

Creating Function

```
drop function if exists thank_you;

delimiter $

CREATE FUNCTION thank_you(p_name char(50))

RETURNS char(100)

BEGIN

RETURN CONCAT('Thank You, ',p_name,'!');

END$

delimiter ;
```

Invoking Function

```
Select thank_you(name) from emp;
```

Compound statements

```
drop procedure if exists multitask;
delimiter $
CREATE procedure multitask()
BEGIN
  select * from emp;
  select * from dept;
  call display_emp_info(1);
  select tax(salary) from emp;
  Select thank_you(name) from emp;
END$
delimiter ;
```

Declaring Variables

- **DECLARE**
  - Declaring
  - Scope

```
DELIMITER //
CREATE FUNCTION add_tax (total_charge FLOAT(9,2))
RETURNS FLOAT(10,2)
BEGIN
  DECLARE tax_rate FLOAT (3,2) DEFAULT 0.07;
  RETURN total_charge + total_charge * tax_rate;
END//
DELIMITER ;
```

Assign Variables (SELECT ... INTO / SET)

```
CREATE procedure display_dept_name(p_id integer)
BEGIN
  Declare v_dno integer;
  Declare v_name varchar(50);
  SET v_name = (select ename from emp where id =
p_id);
  select deptno into v_dno from emp where id = p_id;
  /* print*/
  select thank_you(v_name);
  select dname from dept where deptno = v_dno;
END$
delimiter ;
```

Examine Stored Routines

- `SHOW CREATE PROCEDURE` / `FUNCTION`
  - MySQL specific
  - Returns exact code string
- `SHOW PROCEDURE` / `FUNCTION STATUS`
  - MySQL specific
  - Returns characteristics of routines
- `INFORMATION_SCHEMA.ROUTINES`
  - Standard SQL
  - Returns a combination of the `SHOW` commands

Delete Stored Routines

- **DROP PROCEDURE**

```
DROP PROCEDURE [IF EXISTS] procedure_name;
```

  - Example

```
DROP PROCEDURE proc_1;
```

- **DROP FUNCTION**

```
DROP FUNCTION [IF EXISTS] function_name;
```

  - Example

```
DROP FUNCTION IF EXISTS func_1;
```

Flow Control Statements

- Statements and constructs that control order of operation execution
- Common flow controls
  - Choices
    - **`IF`** and **`CASE`**
  - Loops
    - **`REPEAT`**, **`WHILE`** and **`LOOP`**

IF

- The most basic of all choice flow controls or conditional constructs

```
IF (test condition) THEN
ELSEIF (test condition) THEN
ELSE
END IF
```

CASE

- **CASE** provides a means of developing complex conditional constructs
- **CASE** works on the principle of comparing a given value with specified constants and acting upon the first constant that is matched

```
CASE case_value
  WHEN value THEN

  ELSE

END CASE
```

OR

```
CASE
  WHEN test_condition THEN

  ELSE

END CASE
```

REPEAT

- The **REPEAT** statement repeats the statements between the **REPEAT** and **UNTIL** keywords until the condition after the **UNTIL** keyword becomes **TRUE**
- A **REPEAT** loop always iterates at least once
- Optional Labels
  - Begin
  - End

```
my_label: REPEAT

UNTIL test_condition
END REPEAT my_label;
```

WHILE

- **WHILE** repeats the statements between the **DO** and **END WHILE** keywords as long as the condition appearing after the **WHILE** keyword remains **TRUE**
- A **WHILE** loop may never iterate (if the condition is initially **FALSE**)

```
my_label: WHILE test_condition
DO
END WHILE my_label;
```

LOOP

- The statements between the **LOOP** and **END LOOP** keywords are repeated.
- The loop must be explicitly exited, and usually this is accomplished with a **LEAVE** statement.
- A valid label must appear after the **LEAVE** keyword.

```
my_label: LOOP

  LEAVE my_label;
END LOOP my_label;
```

Triggers

What Are Triggers?

- Named database objects
- Activated when table data is modified
- Bring a level of power and security to table data
- Trigger scenario using the world database
  - What would you do after changing the Country table code column?
  - Since the code is stored in all three world database tables, it is best to change all 3 at once
  - A trigger can accomplish this task
- Trigger features

Creating Triggers

- Syntax

```
CREATE TRIGGER trigger_name
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON table_name
  FOR EACH ROW
  triggered_statement
```

```
create table deleted_emp like emp;
```

```
CREATE TRIGGER emp_deletion_log
AFTER DELETE ON emp
FOR EACH ROW
     INSERT INTO Deleted_emp (ID, eName)
     VALUES (OLD.ID, OLD.eName);
```

To test the trigger

```
delete from emp where id = 6;

select * from deleted_emp;
```

Delete Triggers

- **DROP TRIGGER**

```
DROP TRIGGER trigger_name;
DROP TRIGGER schema_name.trigger_name;
```

If you drop a table, the triggers are automatically dropped also.

Events

```
CREATE EVENT delete_changes
ON SCHEDULE EVERY 48 HOUR
DO
   DELETE FROM changes;
```

Make sure that event scheduler variable is ON

```
select @@global.event_scheduler;
set @@global.event_scheduler =1;
```

GUI Tools

Self Study

- Import & Export
- Storage Engines
- SQL modes
- MySQL Architecture

Exporting & Importing Data

Backup Database into file

- Create a dump file using:

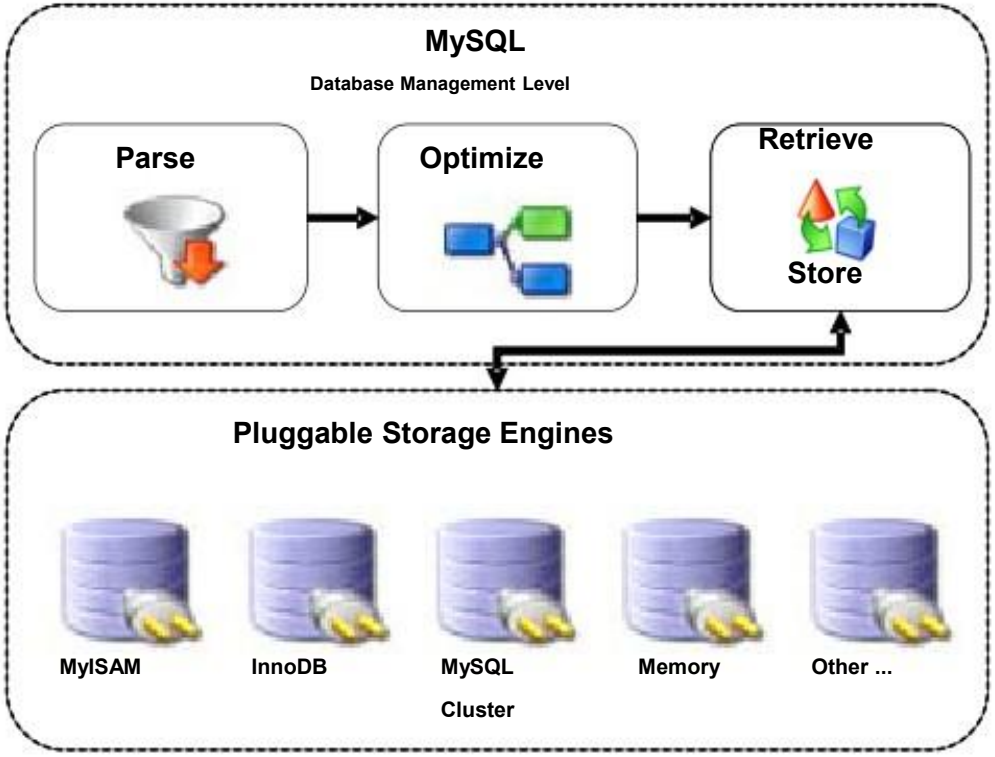
```
mysqldump grades -u root -p> d:\grades.sql
```

- load the dump file into the MySQL

```
Mysql –D grades –u root –p< d:\grades.sql
```

Storage Engines

Storage Engine Breakdown (1/2)



Storage Engine Breakdown (2/2)

- Storage medium
- Transactional capabilities
- Locking
- Backup and recovery
- Optimization
- Special features
- MySQL server operates same for all storage engines
  - SQL commands independent of engine

Storage Engines and MySQL

- Can choose specific storage engine when creating a table
- Best fit for your application
- Each have different characteristics and implications

Available Storage Engines

- MySQL provides and maintains several storage engines
- Also compatible with many third party engines
- MySQL developed
  - MyISAM
  - Falcon
  - NDB/Cluster
  - MEMORY
  - ARCHIVE
  - BLACKHOLE
  - CSV
- Third party engines
  - InnoDB
  - solidDB
  - InfoBright- BrightHouse
  - Nitro
  - PBXT

Common Storage Engines

- MyISAM

  - Fast
  - Data stored in table
  - Table-level locking
- InnoDB

  - Transactional
  - Foreign keys
  - Row-leveling locking
  - Backups
- Memory

  - Data is in memory ONLY

Storage Engines Available on Server

- View available storage engines…

```
SHOW ENGINES\G
*************************** 1. row ***************************
   Engine: MyISAM
   Support: DEFAULT
   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: YES
   Comment: Supports transactions, row-level locking, foreign
   keys
```

Setting the Storage Engine

- Specify engine using **CREATE TABLE**
  - MySQL uses system default engine if not specified
- Change engine for existing table with

  ALTER TABLE
- Examples

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

ALTER TABLE t ENGINE = MEMORY;
```

The InnoDB Storage Engine

- Manages tables with specific characteristics
  - Represented on disk by a `.frm` format file as well as data and index storage
  - Supports transactions
  - ACID compliant
  - Auto-recovery after a crash
  - MVCC and non-locking reads
  - Supports foreign keys and referential integrity
  - Supports consistent and online logical backup

The InnoDB Tablespace and Logs (1/2)

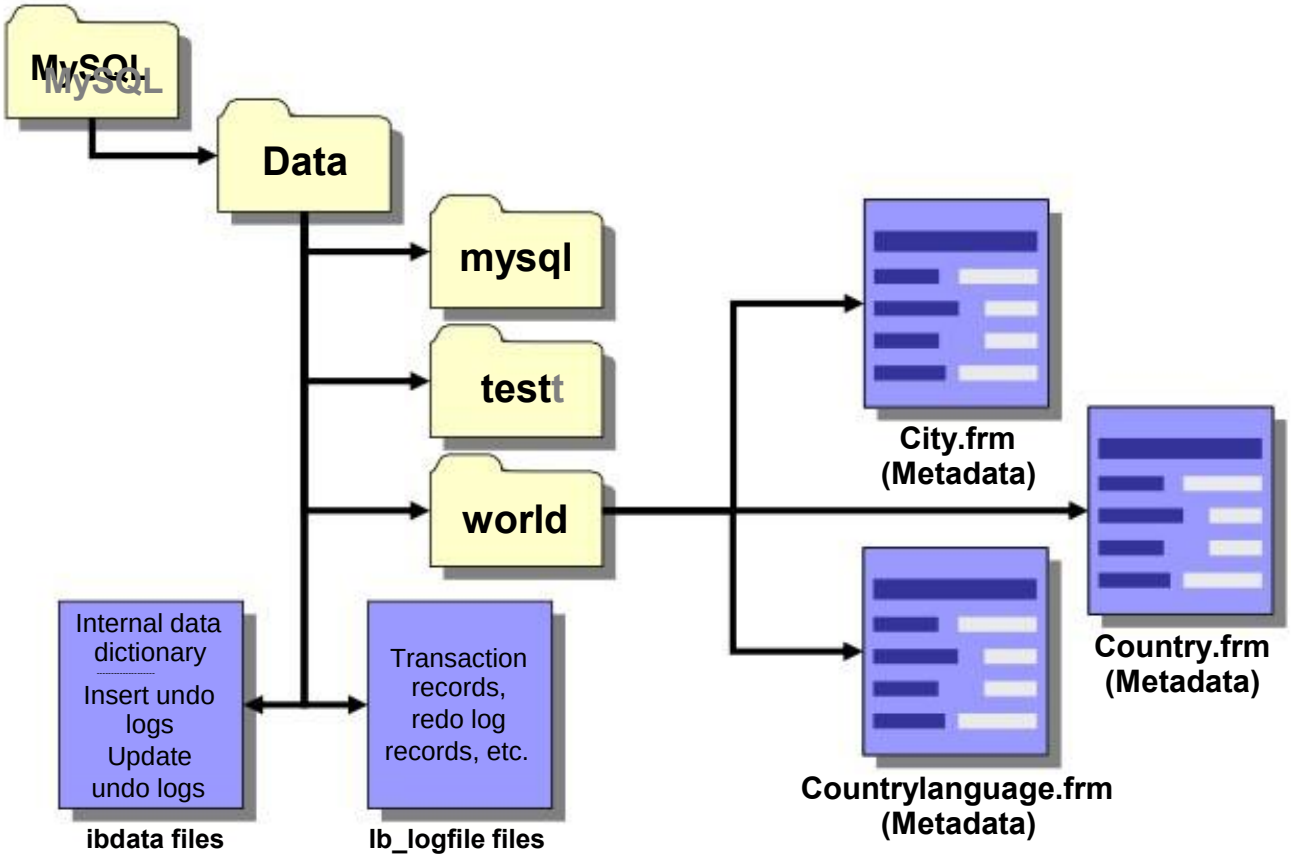
- Tablespace for storing table contents
- Log files for recording transaction activity
- Format file (`.frm`)
- Logical storage area can contain multiple files
- Table-specific file (`.ibd`)

```
--innodb-file-per-table
```

- Manages InnoDB-specific log files
- Log files used for auto-recovery

The InnoDB Tablespace and Logs (2/2)

- File locations



The InnoDB ACID Compliance and Locking

- Satisfies ACID conditions
- General locking properties
  - Does not need to set locks to achieve consistent reads
  - Uses row-level locking per concurrency properties
  - May acquire row locks as necessary to improve concurrency
  - Deadlock is possible
- Supports two locking modifiers
  - Convert non-locking into locking reads
  - `LOCK IN SHARE MODE` places a shared lock on each selected row
  - `FOR UPDATE` places an exclusive lock on selected rows
- REPEATABLE READ isolation level allows modifiers

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
Durability	Table recovery	No disk I/O or persistence	Durability recovery
Supports Transactions	NO	NO	YES

SQL Modes

SQL Modes for Syntax Checking

- View current SQL mode value

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

SESSION is default operating mode.

- Check SQL mode setting

```
SELECT @@sql_mode;
+-----------------------------------------------+
| @@sql_mode                                    |
+-----------------------------------------------+
| STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO |
+-----------------------------------------------+
```

SQL Mode Values

- Commonly used modes
  - ANSI
  - ONLY_FULL_GROUP_BY
  - ERROR_FOR_DIVISION_BY_ZERO
  - STRICT_TRANS_TABLES, STRICT_ALL_TABLES
  - NO_ZERO_DATE, NO_ZERO_IN_DATE
  - TRADITIONAL

MySQL Architecture

MySQL General Architecture

MySQL General Architecture

- Networked environment using client/server
- Components of MySQL installation
  - MySQL server
  - Client programs
  - Non-client programs

MySQL Server

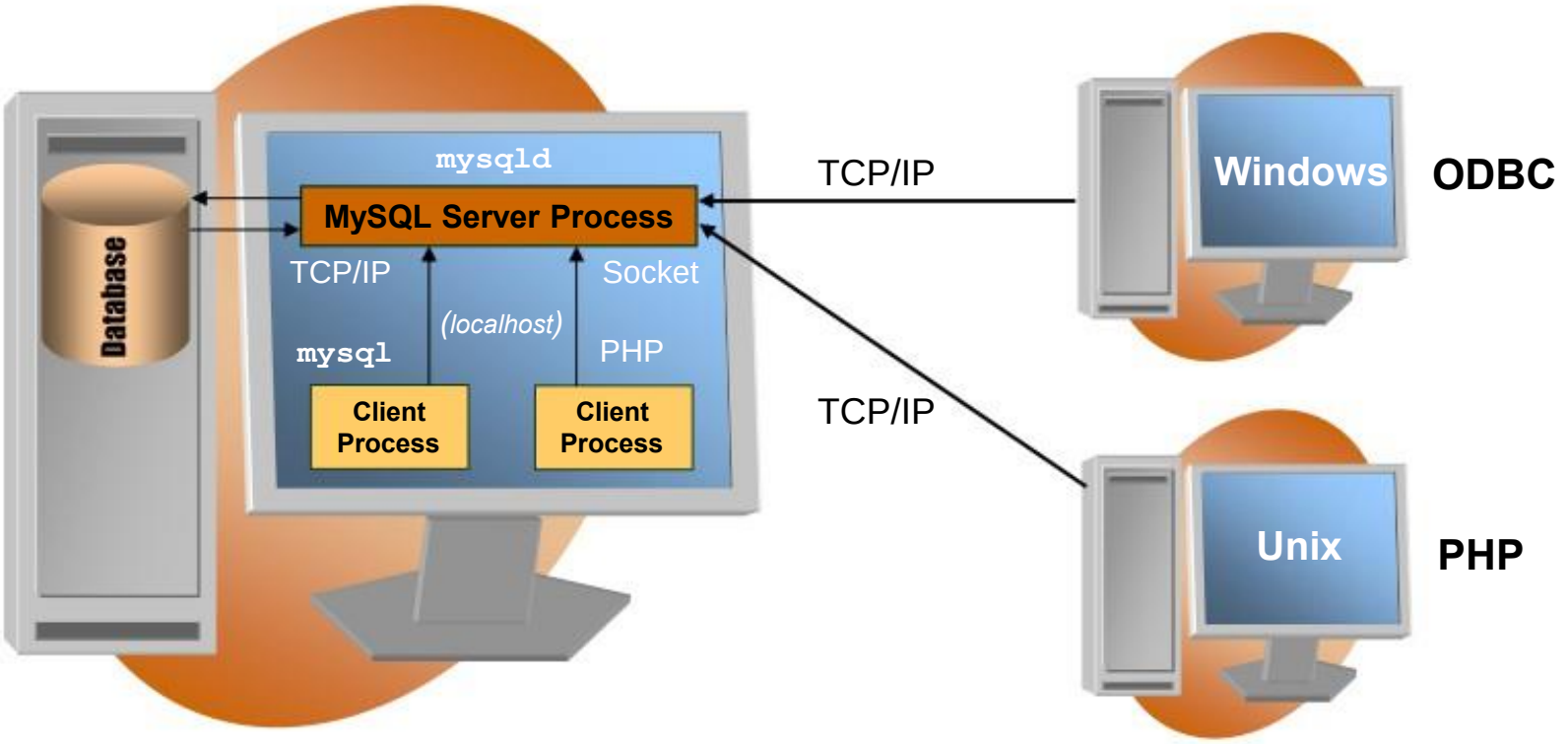
- `mysqld`
- Single process architecture
- Manages access to databases
- Multi-threaded connections
- Supports multiple storage engines
- Server vs. Host
  - *Server* is software
  - *Host* is physical machine which runs software

Client Programs

- Communicate with server to manipulate databases
- Common client programs
  - `mysql`
  - `mysqlimport`
  - `mysqldump`
  - `mysqladmin`
  - `mysqlcheck`

MySQL Client/Server Model

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



Connection Parameter Options (2/3)

- User identification

  `--user=user_name` or `-u user_name`

  `--password=pass_value` or `-ppass_value`

- Password options
  - Short form uses no space after option
  - Can omit password value to get prompt

```
shell> mysql -u user_name -p
Enter password:
```

Using `mysql` Interactively

- **`mysql`** enables server queries
  - Interactive
  - Batch Mode
- Execute within the **`mysql`** client

```
SELECT VERSION();
+------------------+
| VERSION()        |
+------------------+
| 5.1.30-community |
+------------------+
```

- Execute from the command line

```
shell> mysql -u user_name -ppassword -e "SELECT VERSION()"
+------------------+
| VERSION()        |
+------------------+
| 5.1.30-community |
+------------------+
```

MySQL Terminology

- MySQL operates using a client/server architecture.

The first program is the MySQL server, **mysqld:**

- The server runs on the machine where your databases are stored.
- It listens for client requests coming in over the network.

MySQL Terminology

- **Mysql** is an interactive client that lets you issue queries and see the results.
- Two administrative clients are:
  - **mysqldump**, a backup program that dumps table contents into a file.
  - **mysqladmin**, which enables you to check on the status of the server and performs other administrative tasks such as telling the server to shut down.

How MySQL stores data?

- A MySQL server can store several databases
- Databases are stored as directories
  - Default is at /usr/local/mysql/var/
  - Or /var/lib/mysql (Ubuntu /Debian)
- Tables are stored as files inside each database **(**directory**)**
- For each table, it has three files (MYISAM):
  - table**.FRM** file containing information about the table structure
  - table**.MYD** file containing the row data
  - table**.MYI** containing any indexes belonging with this table, as well as some statistics about the table.