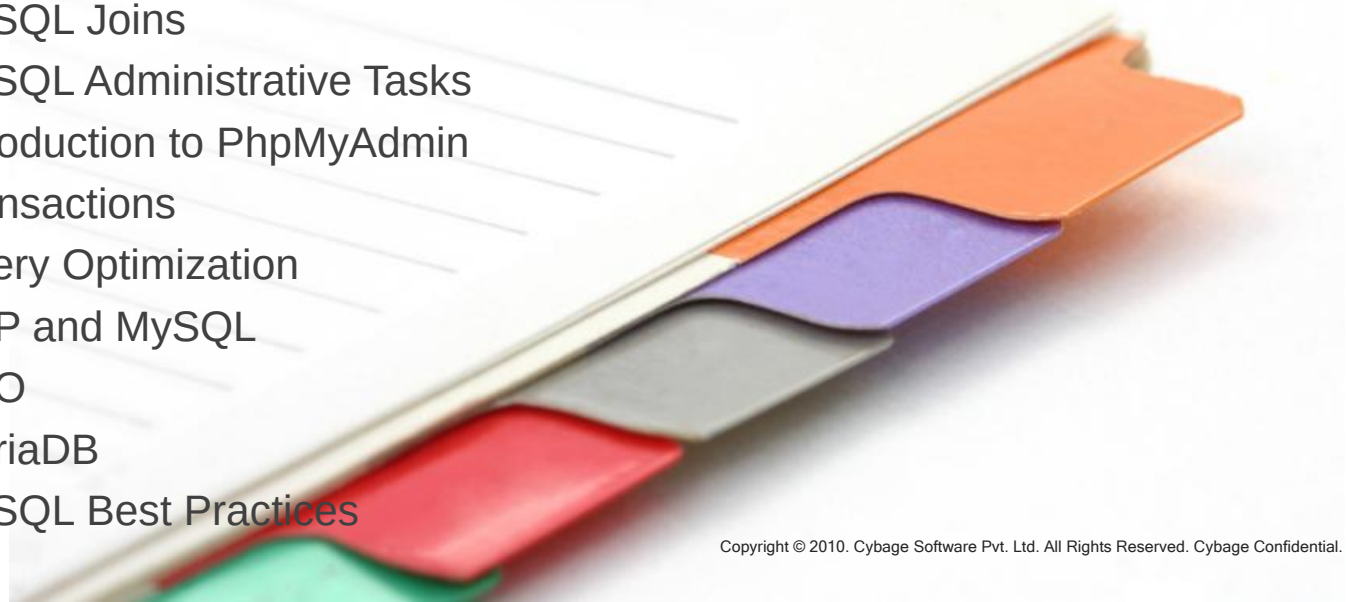
# PHP and MySQL

## Presented by : Sushil Wanjari

# Agenda

- Introduction to RDBMS
- Introduction to MySQL
- Key features of MySQL
- MySQL GUI and Tools
- MySQL Reserved Words
- MySQL Data Types
- MySQL Storage Engines
- MySQL Basic Commands and Functions
- MySQL Joins
- MySQL Administrative Tasks
- Introduction to PhpMyAdmin
- Transactions
- Query Optimization
- PHP and MySQL
- PDO
- MariaDB
- MySQL Best Practices



# Introduction to RDBMS

- A Database (DB) is a organized collection of data (which can easily be accessed, managed and updated)
- A Database Management System (DBMS) is a software program that interacts with the user, other applications and the database itself to capture, analyze and manage data
- In a relational model, data is represented in terms of tuples grouped into relations. Database organized in terms of relational model is a relational database.
- It represents all data in the database as simple tables in the row-column format
- RDBMS is a DBMS that is based on the relational model

# Introduction to MySQL

- The world's most popular open source database software
- MySQL is a RDBMS
- MySQL software is Open Source
- For commercial use, several paid editions are available and offer additional functionality
- MySQL database server is very fast, reliable and easy to use
- It works in client/server or embedded systems
- A large amount of contributed MySQL software is available
- Latest stable version: MySQL 5.6

# Key Features of MySQL

- Written in C and C++
- Works on many different platforms
- Designed to be fully multi-threaded, to easily use multiple CPUs if available
- Provides transactional and non-transactional storage engines
- Uses a very fast thread-based memory allocation system
- Executes very fast joins using an optimized nested-loop join
- Implements SQL functions using a highly optimized class library that should be as fast as possible

# Key Features of MySQL

- Security
  - A privilege and password system that is very flexible and secure
  - Password security by encryption of all password traffic when you connect to a server
- Scalability and Limits
  - Support for large databases
  - Support for upto 64 indexes per table
- Connectivity
  - Clients can connect to MySQL server using server protocols
  - Clients can connect using TCP/IP sockets on any platform
- Data Types
  - Many data types

# Key Features of MySQL

- Statements and Functions
  - Full operator and function support in the SELECT list and WHERE clause of queries
  - Full support for SQL GROUP BY and ORDER BY clauses
  - Support for GROUP function
  - Support for JOINs
  - Support for CRUD operations
  - Support for MySQL specific SHOW statements that retrieve information about databases, storage engines, tables and indexes
  - An EXPLAIN statement to show how optimizer resolves a query

# MySQL GUI and Tools

- **PhpMyAdmin** is the web based graphical interface that allows users to use functionalities of MySQL database
- The **MySQL Migration Toolkit** is a graphical tool provided by MySQL for migrating schema and data from various relational database systems to MySQL
- **MySQL Administrator** is a program to perform administrative operations such as configuring the server, monitoring its status and performance, starting and stopping it, user and connection management, backup/restore etc.
- **MySQL Query Browser** is a tool provided by MySQL for creating, executing and optimizing queries. It is designed to help help writing queries and analyze data stored within your MySQL database

# MySQL Reserved Words

- Certain words such as SELECT, DELETE or  BEGIN are reserved and can not be as it is used as identifiers such as table or column names as it is
- The same stands true for the built in function names
- Reserved words can be used as identifiers if you quote them
- List of Reserved Words

# MySQL Data Types

- Data type is an attribute of data that tells the computer and the programmer about what kind of data is being dealt with

- The main data types are
  - String Types
  - Numeric Types
  - Date / Time Types
  - Misc Types

# MySQL Data Types - String

| Data Type | Description |
|---|---|
| CHAR() | A fixed section from 1 to 255 characters long (1 byte) *(length is optional)* |
| VARCHAR() | A variable section from 1 to 255 characters long (1 byte) *(length is mandatory)* |
| BINARY() | Same as char(), except that they contain binary strings (1 byte) |
| VARBINARY() | Same as varchar(), except that they contain binary strings (1 byte) |
| TINYTEXT | A string with a maximum length of 255 characters. (1 byte) |
| TEXT | A string with a maximum length of 65535 characters. (2 byte) |
| BLOB | A string with a maximum length of 65535 characters. (2 byte) |
| MEDIUMTEXT | A string with a maximum length of 16777215 characters. (3 byte) |
| MEDIUMBLOB | A string with a maximum length of 16777215 characters. (3 byte) |
| LONGTEXT | A string with a maximum length of 4294967295 characters. (4 byte) |
| LONGBLOB | A string with a maximum length of 4294967295 characters. (4 byte) |

# MySQL Data Types - Numeric

| Data Type | Description |
|---|---|
| TINYINT( ) | -128 to 127 normal 0 to 255 UNSIGNED - (1 byte) |
| SMALLINT( ) | -32768 to 32767 normal 0 to 65535 UNSIGNED (2 byte) |
| MEDIUMINT( ) | -8388608 to 8388607 normal 0 to 16777215 UNSIGNED (3 byte) |
| INT( ) | -2147483648 to 2147483647 normal 0 to 4294967295 UNSIGNED (4 byte) |
| BIGINT( ) | -9223372036854775808 to 9223372036854775807 normal 0 to 18446744073709551615 UNSIGNED (8 byte) |
| FLOAT | A small number with a floating decimal point – 4 byte |
| DOUBLE( , ) | A large number with a floating decimal point – 8 byte |
| DECIMAL( , ) | A DOUBLE stored as a string , allowing for a fixed decimal point. |
| BIT() | The BIT data type is used to store bit-field values (0,1) |

- The integer types have an extra option called UNSIGNED
- Normally the integer goes from an negative to positive value. Thus using an UNSIGNED command will move that range up, as it starts at zero instead of a negative number

# MySQL Data Types – Date / Time

| Data Type | Description |
|---|---|
| DATE | YYYY-MM-DD |
| DATETIME | YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | YYYYMMDDHHMMSS |
| TIME | HH:MM:SS |
| YEAR | YYYY or YY, 1901 to 2199 range |

# MySQL Data Types – Misc

| Data Type | Description |
|---|---|
| ENUM ( ) | Short for ENUMERATION which means that each column may have one of a specified possible values. |
| SET | Similar to ENUM except each column may have more than one of the specified possible values. |

- You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted
- SET is similar to ENUM except SET may contain up to 64 list items and can store more than one choice

# MySQL Storage Engines

- MySQL supports several storage engines that act as handlers for different table types
- This includes both that handle transaction-safe tables as well as non transaction-safe tables
- Storage Engines
  - Transaction Safe
    - INNODB
    - BDB
  - Non Transaction Safe
    - MyISAM
    - ISAM
    - MERGE
    - MEMORY (HEAP)

# MySQL Storage Engines - Transaction-safe

- InnoDB
  - ACID compliant (Atomicity, Consistency, Isolation, Durability)
  - Has commit, rollback and crash-recovery capabilities to protect user data
  - Row-level locking increases multi-user concurrency and performance
  - Supports foreign key referential integrity constraints to maintain data integrity
  - Designed for performance when processing large data
  - Tables can be very large even on operating systems where file size is limited
- BDB (Berkeley DB)
  - Tables may have a greater chance of surviving crashes and are also capable of commit and rollback operations on transactions

# MySQL Storage Engines – Non Transaction-safe

- MyISAM
  - Default storage engine (prior to 5.5.5)
  - Table is stored on disk in three files
    - .frm – stores the table format
    - .myd – data file
    - .myi – index file
- Merge
  - Also known as MRG_MyISAM engine
  - Collection of identical MyISAM tables that can be used as one
  - “Identical” means tables that have identical column and index information
- Memory (Heap)
  - Creates tables with contents that are stored in memory
  - Vulnerable to crashes, hardware issues and power failures
  - Can be used as temporary work areas or read-only caches for data pulled from other tables

# MySQL Basic Commands and Functions

- Create a new database
  - CREATE DATABASE db_name;
- Drop the database
  - DROP DATABASE db_name;
- Show all database
  - SHOW databases

# MySQL Basic Commands and Functions

- Create a table
  - CREATE TABLE table_name (column_name1 column_type1, column_name2 column_type2);

  - CREATE TABLE *IF NOT EXISTS* Student
    (id int(10) auto_increment, name varchar(100) not null, primary key(id));

  - CREATE TABLE *IF NOT EXISTS* Student_Details
    (id int(10), address text,
    foreign key(id) references Student(id) *ON DELETE CASCADE*);

- Drop the table
  - DROP TABLE *IF EXISTS* table_name;

- Alter the table
  - ALTER TABLE table_name ADD COLUMN (column_name column_type);

  - ALTER TABLE Student add roll_number int(5);
  - ALTER TABLE Student drop roll_number;
  - ALTER TABLE Student modify name varchar(50);
  - ALTER TABLE Student_Details RENAME TO Student_Address

# MySQL Basic Commands and Functions

- SELECT

  - SELECT [ DISTINCT | * ] column1, column2, ….
    [ FROM table_name ]
    [ WHERE where_clause ]
    [ GROUP BY expression ]
    [ HAVING expression ]
    [ ORDER BY column1, column2, … ]

  - SELECT * FROM Student;
  - SELECT name FROM Student;
  - SELECT name FROM Student WHERE id = 10;
  - SELECT * FROM Student ORDER BY name ASC LIMIT 0,10;
  - SELECT SUM(marks) FROM Students WHERE roll_number = 10;
  - SELECT COUNT(id) FROM Student;
  - SELECT name FROM Student GROUP BY class;
  - SELECT min(marks) FROM Student WHERE subject = ‘english’ AND class = ‘10’;
  - SELECT DISTINCT last_name from Student;

# MySQL Basic Commands and Functions

- INSERT
  - INSERT INTO table_name [ (col_name1, col_name2, …) ]
    VALUES (expression1, expression2, …);
  - INSERT INTO Student (name, roll_number)
    VALUES (‘George’, 23);
- UPDATE
  - UPDATE table_name SET col_name1 = expression1, col_name2 = expression2, …
    [ WHERE expression ]
  - UPDATE Student SET name = ‘George’ WHERE roll_number = 23;
- DELETE
  - DELETE FROM table_name [ WHERE expression ]
  - DELETE FROM Student WHERE roll_number = 5;

# MySQL Basic Commands and Functions

- WHERE

  • SELECT col_name1, col_name2 … FROM table_name WHERE condition1 [ AND |OR condition2]

  • You can specify any condition using WHERE clause
  • You can specify more than one conditions using AND or OR operators
  • Can be used along with UPDATE or DELETE commands to specify the condition

  • SELECT * FROM Student WHERE roll_number = 10;

  • SELECT roll_number, name, ‘fail’ AS result FROM Student WHERE marks < 35

  • UPDATE Student SET name = ‘George’ WHERE roll_number = 23;

  • UPDATE Employee SET salary = 5000 WHERE salary < 5000 AND department = ‘engineering’

  • DELETE FROM Student WHERE roll_number = 5;

# MySQL Basic Commands and Functions

- LIKE

  • SELECT col_name1, col_name2 … FROM table_name WHERE col_name1 LIKE expression1;

  • % sign is used like a meta character
  • Underscore (_) character is used to match a single character in the string
  • You can specify more than one conditions using AND or OR operators
  • Can be used along with UPDATE or DELETE commands to specify the condition

  • SELECT * FROM Student WHERE name LIKE ‘George’;

  • SELECT * FROM Student WHERE name LIKE ‘%esh’;

  • SELECT * FROM Student WHERE name LIKE ‘Ra_esh’;

  • SELECT * FROM Student WHERE name LIKE ‘Ra_esh’ AND class = 10;

  • UPDATE Student SET found = 1 WHERE name LIKE ‘%esh’;

  • DELETE FROM Student WHERE name LIKE ‘%abc%’;

# MySQL Basic Commands and Functions

- ORDER BY

  • SELECT col_name1, col_name2 … FROM table_name [ WHERE expression ] ORDER BY col_name1 [ ,col_name2 … ] [ ASC | DESC ];

  • Result can be sorted based on any field if that field is being listed out
  • Result can be sorted on more than one field
  • ASC or DESC keyword is used to specify ascending or descending order. Default is ASC

  • SELECT * FROM Student WHERE class = 10 ORDER BY name;

  • SELECT * FROM Student WHERE class = 10 ORDER BY marks DESC;

  • SELECT * FROM Student WHERE class = 10 ORDER BY marks DESC, name ASC;

# MySQL Basic Commands and Functions

- GROUP BY
  - Used to group values from a column
  - Calculations can be performed n the column
  - You can use COUNT, SUM, AVG etc functions on the grouped column
  - SELECT col_name1, col_name2 … FROM table_name GROUP BY col_name;
  - SELECT class, count(*) FROM Students GROUP BY class;
- HAVING
  - Used to specify the filter conditions for group of rows or aggregations
  - When using GROUP BY clause, you can apply a filter condition to the columns appearing in the GROUP BY clause
  - SELECT roll_number, SUM(marks) as total FROM Students GROUP BY roll_number HAVING total > 70;

# MySQL Basic Commands and Functions

- IN
  - Used to replace many OR conditions
  - SELECT col_name1, col_name2 … FROM table_name WHERE col_name IN (expression1, expression2, …);
  - SELECT * FROM Students WHERE class in (8, 9, 10);
- BETWEEN
  - Used to replace a combination of “greater than equal AND less than equal” conditions
  - SELECT col_name1, col_name2 … FROM table_name WHERE col_name BETWEEN expression1 AND expression2
  - SELECT roll_number, name FROM Students WHERE marks BETWEEN 30 AND 40;

# MySQL Basic Commands and Functions

- Introduction to Index

  • A data structure that improves the speed of operations on a table

  • Can be created using one or more columns

  • INSERT and UPDATE statements take more time on table whereas SELECR statements become fast on those tables

  • UNIQUE INDEX

    CREATE UNIQUE INDEX index_name ON table_name (column1, column2, …);

  • PRIMARY INDEX

    ALTER TABLE table_name ADD PRIMARY KEY(id);

    ALTER TABLE table_name DROP PRIMARY KEY;

  • NORMAL INDEX

    CREATE INDEX index_name ON table_name (column_name);

# MySQL Basic Commands and Functions

- Aggregate Functions

  - AVG(expression)
    Calculates average value
    SELECT AVG(marks) FROM Students;

  - COUNT(expression)
    Returns number of rows in a table or number of rows matching the condition
    SELECT COUNT(id) FROM Students WHERE result = ‘pass’;

  - SUM(expression)
    Calculates sum of a set of values or expression
    SELECT SUM(marks) FROM Students WHERE roll_number = 10 and class = 8;

  - MIN(expression)
    Returns minimum value in a set of values
    SELECT MIN(marks) FROM Students WHERE subject = ‘english’;

  - MAX(expression)
    Returns maximum value in a set of values
    SELECT MAX(marks) FROM Students WHERE subject = ‘english’;

# MySQL Basic Commands and Functions

- String Functions

  - CONCAT(str1, str2,…)
    Returns the string by concatenating the arguments
    SELECT CONTACT(first_name, ‘ ‘, last_name) as full_name FROM Student;

  - CONCAT_WS(separator, str1, str2,…)
    Returns the string by concatenating the arguments with a separator
    SELECT CONTACT_WS(‘ ‘, first_name, middle_name, last_name) as full_name FROM Student;

  - LENGTH(str)
    Returns the length of the string
    SELECT LENGTH(first_name) FROM Student;

# MySQL Basic Commands and Functions

- String Functions

  - INSTR(str, substr)

    Returns position of first occurrence of substring in the original string. Position start from 1. If substring is not found, then returns “0”

    SELECT INSTR(‘esh’, first_name) FROM Student;

  - REPLACE(str, string_to_find, string_to_replace)

    Replaces a string by another one in the specified column field

    SELECT REPLACE(name, ‘Daviid’, ‘David’) FROM Student;

  - SUBSTR(str, position, length)

    Returns a substring starting from the position to the end of string

    SELECT SUBSTR(first_name, 1, 1) as initials FROM Student;

# MySQL Basic Commands and Functions

- Date and Time Functions

  - CURDATE()
    Returns today's date
    SELECT CURDATE();

  - CURTIME()
    Returns the current time
    SELECT CURTIME();

  - NOW()
    Returns the current date and time
    SELECT NOW();

  - DATE_FORMAT(date, format)
    Formats the date value according to the format string
    SELECT DATE_FORMAT('2014-09-01 09:02:00', '%W %M %Y');
    Monday September 2014
    SELECT DATE_FORMAT('2014-09-01 09:02:00', '%H:%i:%s');
    09:02:00

# MySQL Basic Commands and Functions

- Date and Time Functions

  - STR_TO_DATE(str, format)
    Converts the str into a date value based on the format string
    SELECT STR_TO_DATE('01,10,2014', '%d,%m,%Y');
    2014-01-10

  - DATEDIFF(date1, date2)
    Calculates number of days between two dates
    SELECT DATEDIFF('2014-08-17', '2014-08-08');
    9

# MySQL Basic Commands and Functions

- Control Flow Functions

  • IF(expression, if_true_expression, if_false_expression)

  If expression evaluates to true (i.e. not null and not 0), then if_true_expression is returned. Else if_false_expression is returned

  SELECT roll_number, IF(marks > 35, ‘pass’, ‘fail’) FROM Students;

  • IFNULL(expression1, expression2)

  Returns first argument if it is not null. Else returns the second argument

  SELECT roll_number, IFNULL(result, ‘on hold’) FROM Students;

  • NULLIF(expression1, expression2)

  Returns null if the first argument is equal to the second argument, otherwise returns the first argument

  SELECT 1/NULLIF(expression, 0);

# MySQL Joins

- Cross Join
  - Very basic type of join, which matches each row from one table to every row from another table
- Inner Join
  - Is used when certain values contained in one table are matched with values contained within a second table
  - INNER JOIN and "," are equivalent in absence of a join condition. Both produce a Cartesian product
- Left Join
  - All rows from the first table (left table) are included. Unmatched rows from the right table are omitted
- Right Join
  - All rows from the second table (right table) are included. Unmatched rows in the left table are omitted
- Self Join
  - Performed by joining a particular table to itself

# MySQL Joins

| id | name |
|---|---|
| 1 | John |
| 2 | Jacob |
| 3 | Norman |
| 4 | Jim |
| 5 | Luis |

| id | address |
|---|---|
| 1 | Miami |
| 2 | Chicago |
| 5 | New York |

- SELECT s.id, s.name, sd.address FROM Student s, Student_Details sd WHERE s.id = sd.id;

| id | name | address |
|---|---|---|
| 1 | John | Miami |
| 2 | Jacob | Chicago |
| 5 | Luis | New York |

# MySQL Joins

| id | name |
|---|---|
| 1 | John |
| 2 | Jacob |
| 3 | Norman |
| 4 | Jim |
| 5 | Luis |

| id | address |
|---|---|
| 1 | Miami |
| 2 | Chicago |
| 5 | New York |

- SELECT s.id, s.name, sd.address FROM Student s LEFT JOIN Student_Details sd ON s.id = sd.id;

| id | name | address |
|---|---|---|
| 1 | John | Miami |
| 2 | Jacob | Chicago |
| 3 | Norman | NULL |
| 4 | Jim | NULL |
| 5 | Luis | New York |

# MySQL Administrative Tasks

- Logging in to MySQL: mysql –h hostname –u username –p password
- View Databases: SHOW DATABASES;
- Backup and Restore:
  mysqldump [options] database [tables] > [dump_file]
  mysqldump –u username –p password [-no-data] db_name > path_to_file
- Table Maintenance and Crash Recovery:
  mysqlcheck [options] database [tables]
  mysqlcheck [options] –databases *-r – Repair, -a – Analyze, -o – Optimize*
- To repair database:
  mysqlcheck –auto –repair –u username –p password database_name
- Restart mysql service:
  /etc/init.d/mysql restart

# MySQL Administrative Tasks

- ANALYZE TABLE
  To update “key distribution” for a table
  ANAYZE TABLE table_name;

- OPTIMIZE TABLE:
  Defragment the physical storage for a table
  OPTIMIZE TABLE table_name;

- CHECK TABLE:
  Check the integrity of a table
  CHECK TABLE table_name;

- REPAIR TABLE:
  Repair some errors in a table
  REPAIR TABLE table_name;

# MySQL Administrative Tasks

- Create a new user:
  CREATE USER username@hostnameIDENTIFIED BY password;
  INSERT INTO user (host, user, password)
  VALUES ('localhost', 'newuser', PASSWORD('password'));

- Change Password:
  SET PASSWORD FOR 'user'@'host' = PASSWORD(password);
  UPDATE USER SET password = PASSWORD(newpassword)
  WHERE user = 'user' AND host = 'host';

# MySQL Administrative Tasks

- Show all tables and columns:

  SELECT DISTINCT table_name FROM INFORMATION_SCHEMA.COLUMNS

  SELECT column_name FROM INFORMATION_SCHEMA.COLUMNS WHERE table_schema = ‘database_name' AND table_name = ‘table_name';

  SHOW COLUMNS FROM db_name.table_name;

  SHOW COLUMNS FROM table_name FROM db_name;

# PhpMyAdmin

# Transactions

- A transaction comprises of a unit of work performed within a DBMS against a DB
- A transaction is a sequential group of database manipulation operations, which is performed as if it were a single work unit
- Two main purposes:
  - To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in case of system failure, when execution stops and many operations upon a database remain uncompleted
  - To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the program's outcome are possibly erroneous
- Transactions provide an "all-or-nothing" proposition, stating that each work-unit performed in a database must either complete in its entirety or have no effect

# Transactions

- A DB transaction, by definition, must be **A**tomic, **C**onsistent, **I**solation and **D**urable
  - Atmoicity: Each transaction is “all or nothing”. If one part of transaction fails, the entire transaction fails, and the DB is left unchanged
  - Consistency: Ensures that any transaction will bring the DB from one valid state to another. Any data written to the DB must be valid according to all defined rules like constraints, cascades, triggers
  - Isolation: Ensures that the current execution of transactions results in a system state that would be obtained if transactions were executed one after other
  - Durability: Once a transaction has been committed, it will remain so, in the event of power loss, crashes or errors
- In MySQL, transactions begin with **BEGIN WORK** statement and end with either **COMMIT** or **ROLLBACK** statement
- When a successful transaction is completed, the COMMIT statement should be issued so that changes to all involved tables will take effect
- If a failure occurs, a ROLLBACK command should be issues to return every table to its previous state

# Transactions

- An example workflow of Transaction
- Begin transaction by using BEGIN WORK command
- Issue one or more commands like SELECT, INSERT, UPDATE or DELETE
- Check if there is no error and everything is according to your requirement
- If there is any error, then issue ROLLBACK command otherwise issue a COMMIT command

# Query Optimization

- EXPLAIN statement can be used to obtain information about how MySQL executes a statement
- When you proceed a SELECT statement with EXPLAIN, MySQL displays information from the optimizer about the statement execution plan
- With the help of EXPLAIN
  - You can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows
  - You can also check whether the optimizer joins the tables in an optimal order
- Output of EXPLAIN
  - select_type: Type of SELECT query
  - table: Table referred by the row
  - type: How MySQL joins the tables being used. It can indicate missing index or how the query should be reconsidered
  - possible_keys: Keys that can be used to find rows from table
  - key: Indicates the actual index used by MySQL
  - rows: Number of records that were examined to produce the output
  - Extra: Additional information about the query execution plan

# Query Optimization

- Adjusting internal variables
  - Index Buffer Size (key_buffer): Controls the size of buffer when handling table indices. Recommended to increase as much as you can afford to ensure best performance of indexed tables
  - Table Buffer Size (read_buffer_size): Size of the buffer that is allocated when MySQL scans a table sequentially
  - Number of Maximum Open Tables (table_cache): Controls the maximum number of tables that can be open at any time, and controls the ability of server to respond to incoming requests
  - Time Limit for Long Queries (long_query_time): Maximum time limit for query execution
- Use ANALYZE to make the indexes usable by query optimizer
- Add INDEX for improving SELECT query performance
- Make use of WHERE clause in such a way that will reduce intermediate results

# PHP and MySQL

- Three main API options when considering connecting to a MySQL database server from PHP

- PHP's MySQL Extension *(deprecated as of PHP 5.5.0)*
- PHP's MySQLi Extension *(MySQL improved version)*
- PHP Data Objects (PDO)

# PHP and MySQL

- MySQL Extension
  - Original extension designed to allow users to develop PHP application that interact with MySQL database
  - Provides procedural interface and intended to use only with MySQL < 4.1.3
  - deprecated as of PHP 5.5.0

# PHP and MySQL

- mysql_connect
- To connect to a database
- Syntax
  - resource mysql_connect ([ string $server [, string $username [, string $password ]]] )
- Returns
  - MySQL link identifier on success or FALSE on failure
- Example

```
<?php
    $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
    if (!$link) {
        die('Could not connect: ' . mysql_error());
    }
    echo 'Connected successfully';
    mysql_close($link);
    ?>
```

# PHP and MySQL

- mysql_pconnect
- To make a persistent connection to the database
- Syntax
  - resource mysql_connect ([ string $server [, string $username [, string $password ]]] )
- Returns
  - MySQL link identifier on success or FALSE on failure
- Example

```
<?php
    $link = mysql_pconnect('localhost', 'mysql_user', 'mysql_password');
    if (!$link) {
        die('Could not connect: ' . mysql_error());
    }
    echo 'Connected successfully';
    mysql_close($link);
    ?>
```

# PHP and MySQL

- mysql_select_db
- To select the particular database on the server
- Syntax
  - bool mysql_select_db ( string $database_name [, resource $link_identifier = NULL ] )
- Returns
  - TRUE on success or FALSE on failure

# PHP and MySQL

- mysql_select_db

- Example

```
<?php

    $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
    if (!$link) {
        die('Not connected : ' . mysql_error());
    }

    // make foo the current db
    $db_selected = mysql_select_db('foo', $link);
    if (!$db_selected) {
        die ('Can\'t use foo : ' . mysql_error());
    }
    ?>
```

# PHP and MySQL

- mysql_close
- To close the connection to MySQL server that is associated with the specified link
- Syntax
  - bool mysql_close ([ resource $link_identifier = NULL ] )
- Returns
  - TRUE on success or FALSE on failure
- Example

```
<?php
    $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
    if (!$link) {
        die('Could not connect: ' . mysql_error());
    }
    echo 'Connected successfully';
    mysql_close($link);
    ?>
```

# PHP and MySQL

- mysql_drop_db
- To drop/delete a database associated with the link identifier
- Syntax
  - bool mysql_drop_db ( string $database_name [, resource $link_identifier = NULL ] )
- Returns
  - TRUE on success or FALSE on failure

# PHP and MySQL

- mysql_drop_db

- Example

```
<?php
    $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
    if (!$link) {
        die('Could not connect: ' . mysql_error());
    }

    $sql = 'DROP DATABASE my_db';
    if (mysql_query($sql, $link)) {
        echo "Database my_db was successfully dropped\n";
    } else {
        echo 'Error dropping database: ' . mysql_error() . "\n";
    }
    ?>
```

# PHP and MySQL

- mysql_errno
- Returns the numerical value of the error message from previous MySQL operation
- Syntax
  - int mysql_errno ([ resource $link_identifier = NULL ] )
- Returns
  - Error number from the last MySQL function or 0 (zero) if no error occurred

# PHP and MySQL

- mysql_error
- Returns the text of the error message from previous MySQL operation
- Syntax
  - string mysql_error ([ resource $link_identifier = NULL ] )
- Returns
  - Error message from the last MySQL function or empty string if no error occurred

# PHP and MySQL

- mysql_errno and mysql_error

- Example

```
<?php
    $link = mysql_connect("localhost", "mysql_user", "mysql_password");

    mysql_select_db("nonexistentdb", $link);
    echo mysql_errno($link) . ": " . mysql_error($link). "\n";

    mysql_select_db("kossu", $link);
    mysql_query("SELECT * FROM nonexistenttable", $link);
    echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
    ?>
```

# PHP and MySQL

- mysql_query
- Sends the query to the currently active database on the server associated with the specified link identifier
- Syntax
  - mixed mysql_query ( string $query [, resource $link_identifier = NULL ] )
- Returns
  - Resource (result set) for SELECT, SHOW, DESCRIBE, EXPLAIN queries
  - Use mysql_num_rows() to find out how many rows were returned from above statements
  - TRUE on success or FALSE on failure for INSERT, UPDATE, DELETE, DROP queries
  - Use mysql_affected_rows() to find out how many rows were affected by above statements

# PHP and MySQL

- mysql_query

- Example

```
<?php
    $result = mysql_query('SELECT firstname, lastname, address, age FROM friends');
    if (!$result) {
        die('Invalid query: ' . mysql_error());
    }
    ?>
```

# PHP and MySQL

- mysql_fetch_row
- Gets a result row as an enumerated array one by one
- Syntax
  - array mysql_fetch_row ( resource $result )
- Returns
  - Array corresponding to the fetched row or FALSE if there are no more rows

# PHP and MySQL

- mysql_fetch_row

- Example

```
<?php
    $result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
    if (!$result) {
        echo 'Could not run query: ' . mysql_error();
        exit;
    }
    $row = mysql_fetch_row($result);

    echo $row[0]; // 42
    echo $row[1]; // the email value
    ?>
```

# PHP and MySQL

- mysql_fetch_array
- Fetch a result row as an associative array, a numeric array, or both
- Syntax
  - array mysql_fetch_array ( resource $result [, int $result_type = MYSQL_BOTH ] )
- Returns
  - Array corresponding to the fetched row or FALSE if there are no more rows
  - Note:

    By using MYSQL_BOTH (default), you'll get an array with both associative and number indices

    Using MYSQL_ASSOC, you only get associative indices (as mysql_fetch_assoc() works)

    using MYSQL_NUM, you only get number indices (as mysql_fetch_row() works)

# PHP and MySQL

- mysql_fetch_array

- Example

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password") or
        die("Could not connect: " . mysql_error());
    mysql_select_db("mydb");

    $result = mysql_query("SELECT id, name FROM mytable");

    while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
        printf("ID: %s  Name: %s", $row[0], $row[1]);
    }

    mysql_free_result($result);
    ?>
```

# PHP and MySQL

- mysql_fetch_assoc
- Fetch a result row as an associative array
- Syntax
  - array mysql_fetch_assoc ( resource $result )
- Returns
  - Associative array corresponding to the fetched row or FALSE if there are no more rows

# PHP and MySQL

- mysql_fetch_assoc

- Example

```
<?php
    mysql_connect("localhost", "mysql_user", "mysql_password") or
        die("Could not connect: " . mysql_error());
    mysql_select_db("mydb");

    $result = mysql_query("SELECT id, name FROM mytable");

    while ($row = mysql_fetch_assoc($result, MYSQL_NUM)) {
        printf("ID: %s  Name: %s", $row[“id”], $row[“name”]);
    }

    mysql_free_result($result);
    ?>
```

# PHP and MySQL

- mysql_fetch_object
- Fetch a result as an object
- Syntax
  - object mysql_fetch_object ( resource $result [, string $class_name [, array $params ]] )
- Returns
  - Object with properties corresponding to the fetched row or FALSE if there are no more rows

# PHP and MySQL

- mysql_fetch_object

- Example

```
<?php
    mysql_connect("hostname", "user", "password");
    mysql_select_db("mydb");
    $result = mysql_query("select * from mytable");
    while ($row = mysql_fetch_object($result)) {
        echo $row->user_id;
        echo $row->fullname;
    }
    mysql_free_result($result);
    ?>
```

# PHP and MySQL

- mysql_free_result

- Free all memory associated with the result identifier
- Only needs to be called if you are concerned about how much memory is being used by large result set queries
- All associated s=result memory is automatically freed at the end of script execution

- Syntax
  - bool mysql_free_result ( resource $result )

- Returns
  - TRUE on success or FALSE on failure

# PHP and MySQL

- mysql_insert_id
- Get the ID generated from previous INSERT operation
- Syntax
  - int mysql_insert_id ([ resource $link_identifier = NULL ] )
- Returns
  - TRUE on success or FALSE on failure
- Example

```
<?php
    $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
    if (!$link) {
        die('Could not connect: ' . mysql_error());
    }
    mysql_select_db('mydb');

    mysql_query("INSERT INTO mytable (product) values ('kossu')");
    printf("Last inserted record has id %d\n", mysql_insert_id());
    ?>
```

# PHP and MySQL

- MySQLi Extension
  - MySQL improved extension
  - Object oriented interface
  - Support for prepared statements
  - Support for transactions
  - Enhanced debugging capabilities

# PHP and MySQL

- MySQLi Extension

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Select queries return a resultset */
if ($result = mysqli_query($link, "SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", mysqli_num_rows($result));

    /* free result set */
    mysqli_free_result($result);
}

mysqli_close($link);
?>
```

# PHP and MySQL

- MySQLi Extension

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}

/* Select queries return a resultset */
if ($result = $mysqli->query("SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", $result->num_rows);

    /* free result set */
    $result->close();
}

$mysqli->close();
?>
```

# PHP and MySQL

- PDO
  - Database abstraction layer for PHP applications
  - Provides a consistent API for PHP application regardless of the type of database that the application will connect to
  - In theory, if you are using PDO, you can switch the database server by doing minor changes to the PHP code
  - Provides clean, simple and portable API
  - Does not allow to use advanced features available with latest versions of MySQL database server
  - PDO_MYSQL is a driver that implements the PDO interface to enable access from PHP to MySQL databases

# PHP and MySQL

- The PDO class Represents a connection between PHP and a database server

| Method | Description |
|---|---|
| PDO::quote | Quotes a string for use in a query |
| PDO::prepare | Prepares a statement for execution and returns a statement object |
| PDO::query | Executes an SQL statement, returning a result set as a PDOStatement object |
| PDO::exec | Execute an SQL statement and return the number of affected rows |
| PDO::lastInsertId | Returns the ID of the last inserted row or sequence value |
| PDO::beginTransaction | Initiates a transaction |
| PDO::commit | Commits a transaction |
| PDO::inTransaction | Checks if inside a transaction |
| PDO::rollBack | Rolls back a transaction |
| PDO::errorCode | Fetch the SQLSTATE associated with the last operation on the database handle |
| PDO::errorInfo | Fetch extended error information associated with the last operation on the database handle |

# PHP and MySQL

- The PDOStatement class Represents a prepared statement and, after the statement is executed, an associated result set

| Method | Description |
|---|---|
| PDOStatement::execute | Executes a prepared statement |
| PDOStatement::fetch | Fetches the next row from a result set |
| PDOStatement::fetchAll | Returns an array containing all of the result set rows |
| PDOStatement::fetchObject | Fetches the next row and returns it as an object |
| PDOStatement::rowCount | Returns the number of rows affected by the last SQL statement |

# MariaDB

- A community-developed form of the MySQL
- Being led by its original developers
- Triggered by concerns over direction by MySQL's acquiring company "Oracle"
- Intent is to maintain high compatibility with MySQL, ensuring a "drop-in" replacement capability with library binary equivalency and exact matching with MySQL APIs and commands

# MariaDB

- MariaDB vs MySQL Features
- More Storage Engines
- Speed Improvements
- Extensions and New Features
  - Microseconds
  - Extended user statistics
  - Faster Joins and Subqueries
  - PCRE Regular Expressions
- Better testing
- Fewer bugs
- Truly Open Source
- Binary drop-in replacement for MySQL

# MySQL Best Practices

- You need to begin considering the best way to organize data before implementing the software
  - How will your table primarily be used
  - What kind of data types are required
  - Which columns will you be querying
- Use EXPLAIN to tune your queries
- LIMIT you selections as per need
- Make optimal use of indexes
- Adjust internal variables for optimal use
- Make use of “slow query log” to log the inefficient or misbehaving queries
- “Benchmark” the queries to check performance

# MySQL Best Practices

- Consider vertical partitioning of the tables
- Make proper use of joins and their order in a query
- Consider performing any operation at DB level as and when possible instead of writing code inside your code
- Make proper choice of storage engine while creating the tables
- Primary Keys should be used in most vital queries
- Separate lease frequently used data in different table and Use foreign keys
- Minimize use of OR keyword in WHERE clauses
- Use ANALYZE TABLE table_name to update the key distribution for the table

# MySQL Best Practices

- Check your code for SQL Injection vulnerabilities
- Disable or restrict remote access and hide MySQL from internet
- Change root username and password
- Enable logging
- LIMIT 1 when getting a unique row
- Avoid SELECT *
- Use ENUM instead of VARCHAR if the field contains only a few kind of values
- Use “prepared statements”
- Create “views” to simplify commonly used table joins

# Any Questions?

# Thank you!