Lecture-37-38

Introduction to MySQL Database

Introduction of MySQL

- MySQL is an SQL (Structured Query Language) based relational database management system (DBMS)
- MySQL is compatible with standard SQL
- MySQL is frequently used by PHP and Perl
- Commercial version of MySQL is also provided (including technical support)

Resource

- MySQL and GUI Client can be downloaded from
  http://dev.mysql.com/downloads/

Command for accessing MySQL

Start MySQL

>mysql –u [username] –p

>Enter password:[password]

Entering & Editing commands

- Prompt mysql>
  - issue a command
  - Mysql sends it to the server for execution
  - displays the results
  - prints another mysql>
- a command could span multiple lines
- A command normally consists of SQL statement followed by a semicolon

Command prompt

prompt	meaning
mysql>	Ready for new command.
->	Waiting for next line of multiple-line command.
‘>	Waiting for next line, waiting for completion of a string that began with a single quote (“'”).
“>	Waiting for next line, waiting for completion of a string that began with a double quote (“"”).
` >	Waiting for next line, waiting for completion of an identifier that began with a backtick (“` ”).
/*>	Waiting for next line, waiting for completion of a comment that began with /*.

MySQL commands

- help \h
- Quit/exit \q
- Cancel the command \c
- Change database use

MySQL data types

- MySQL uses many different data types broken into three categories: numeric, date and time, and string types.

Numeric Data Types:

- **INT** - A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** - A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** - A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range. is from 0 to 65535. You can specify a width of up to 5 digits

- **MEDIUMINT** - A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** - A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
- **FLOAT(M,D)** - A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.

- **DOUBLE(M,D)** - A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
- **DECIMAL(M,D)** - An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

Date and Time Types:

- **DATE** - A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
- **DATETIME** - A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP** - A timestamp between midnight, January 1, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).
- **TIME** - Stores the time in HH:MM:SS format.
- **YEAR(M)** - Stores a year in 2-digit or 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be 1970 to 2069 (70 to 69). If the length is specified as 4, YEAR can be 1901 to 2155. The default length is 4.

String Types:

- **CHAR(M)** - A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
- **VARCHAR(M)** - A variable-length string between 1 and 255 characters in length; for example VARCHAR(25). You must define a length when creating a VARCHAR field.
- **BLOB or TEXT** - A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data; the difference between the two is that sorts and comparisons on stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields. You do not specify a length with BLOB or TEXT.

- **TINYBLOB or TINYTEXT** - A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
- **MEDIUMBLOB or MEDIUMTEXT** - A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
- **LONGBLOB or LONGTEXT** - A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.
- **ENUM** - An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

General Commands

- **USE database_name** : Change to this database. We need to change to some database when we first connect to MySQL.
- **SHOW DATABASES** : Lists all MySQL databases on the system.
- **SHOW TABLES [FROM database_name]** : Lists all tables from the current database or from the database given in the command.
- **DESCRIBE table_name**
- **SHOW FIELDS FROM table_name**
- **SHOW COLUMNS FROM table_name**

  These commands all give a list of all columns (fields) from the given table, along with column type and other info.

.

Table Commands

- **CREATE TABLE table_name (create_clause1, create_clause2, ...)**
- Creates a table with columns as indicated in the create clauses.
- **create_clause** column name followed by column type, followed optionally by modifiers. For example, "gene_id INT AUTO_INCREMENT PRIMARY KEY" (without the quotes) creates a column of type integer with the modifiers described below.

create_clause modifiers

- **AUTO_INCREMENT :** Each data record is assigned the next sequential number when it is given a NULL value.

- **PRIMARY KEY :** Items in this column have unique names, and the table is indexed automatically based on this column. One column must be the PRIMARY KEY, and only one column may be the PRIMARY KEY. This column should also be NOT NULL.

- **NOT NULL : No** values are allowed in this column: a NULL generates an error message as the data is inserted into the table.
- **NULL**
- **DEFAULT value :** If a NULL value is used in the data for this column, the default value is entered instead.

- **DROP TABLE table_name** Removes the table from the database.
- **ALTER TABLE table_name ADD (create_clause1, create_clause2, ...)** Adds the listed columns to the table.
- **ALTER TABLE table_name DROP column_name** Drops the listed columns from the table.
- **ALTER TABLE table_name MODIFY create_clause** Changes the type or modifiers to a column. Using MODIFY means that the column keeps the same name even though its type is altered. MySQL attempts to convert the data to match the new type: this can cause problems.
- **ALTER TABLE table_name CHANGE column_name create_clause** Changes the name and type or modifiers of a column. Using CHANGE (instead of MODIFY) implies that the column is getting a new name.

Data Commands

- **INSERT [INTO] table_name VALUES (value1, value2, ...)**

  Insert a complete row of data, giving a value (or NULL) for every column in the proper order.

- **INSERT [INTO] table_name (column_name1, column_name2, ...) VALUES (value1, value2, ...)INSERT [INTO] table_name SET column_name1=value1, column_name2=value2, ...**

  Insert data into the listed columns only. Alternate forms, with the SET form showing column assignments more explicitly.

- **INSERT [INTO] table_name (column_name1, column_name2, ...) SELECT list_of_fields_from_another_table FROM other_table_name WHERE where_clause**

  Inserts the data resulting from a SELECT statement into the listed columns. Be sure the number of items taken from the old table match the number of columns they are put into!

- **DELETE FROM table_name WHERE where_clause** Delete rows that meet the conditions of the where_clause. If the WHERE statement is omitted, the table is emptied, although its structure remains intact.

- **UPDATE table_name SET column_name1=value1, column_name2=value2, ... [WHERE where_clause]** Alters the data within a column based on the conditions in the where_clause.

Privilege Commands

- **GRANT USAGE ON *.* TO user_name@localhost [IDENTIFIED BY 'password']**

  Creates a new user on MySQL, with no rights to do anything. The IDENTIFED BY clause creates or changes the MySQL password, which is not necessarily the same as the user's system password. The @localhost after the user name allows usage on the local system, which is usually what we do; leaving this off allows the user to access the database from another system. User name NOT in quotes.

- **GRANT SELECT ON *.* TO user_name@localhost** In general, unless data is supposed to be kept private, all users should be able to view it. A debatable point, and most databases will only grant SELECT privileges on particular databases. There is no way to grant privileges on all databases EXCEPT specifically enumerated ones.
- **GRANT ALL ON database_name.* TO user_name@localhost** Grants permissions on all tables for a specific database (database_name.*) to a user. Permissions are for: ALTER, CREATE, DELETE, DROP, INDEX, INSERT, SELECT, UPDATE.

Example:

- **Returns the columns and column information pertaining to the designated table.**

  mysql> show columns from [table name];

- **Show certain selected rows with the value "whatever".**

  mysql> SELECT * FROM [table name] WHERE [field name] = "whatever";

- **Show all records containing the name "Bob" AND the phone number '3444444'.**

  mysql> SELECT * FROM [table name] WHERE name = "Bob" AND phone_number = '3444444';

- **Show all records not containing the name "Bob" AND the phone number '3444444' order by the phone_number field.**

  mysql> SELECT * FROM [table name] WHERE name != "Bob" AND phone_number = '3444444' order by phone_number;

- **Show all records starting with the letters 'bob' AND the phone number '3444444'.**

  mysql> SELECT * FROM [table name] WHERE name like "Bob%" AND phone_number = '3444444';

- **Show all records starting with the letters 'bob' AND the phone number '3444444' limit to records 1 through 5.**

  mysql> SELECT * FROM [table name] WHERE name like "Bob%" AND phone_number = '3444444' limit 1,5;

- **Use a regular expression to find records. Use "REGEXP BINARY" to force case-sensitivity. This finds any record beginning with a.**

  mysql> SELECT * FROM [table name] WHERE rec RLIKE "^a";

Lecture-39-40

Introduction of Mysql phpmyadmin

Introduction of MySQL PhpMyadmin

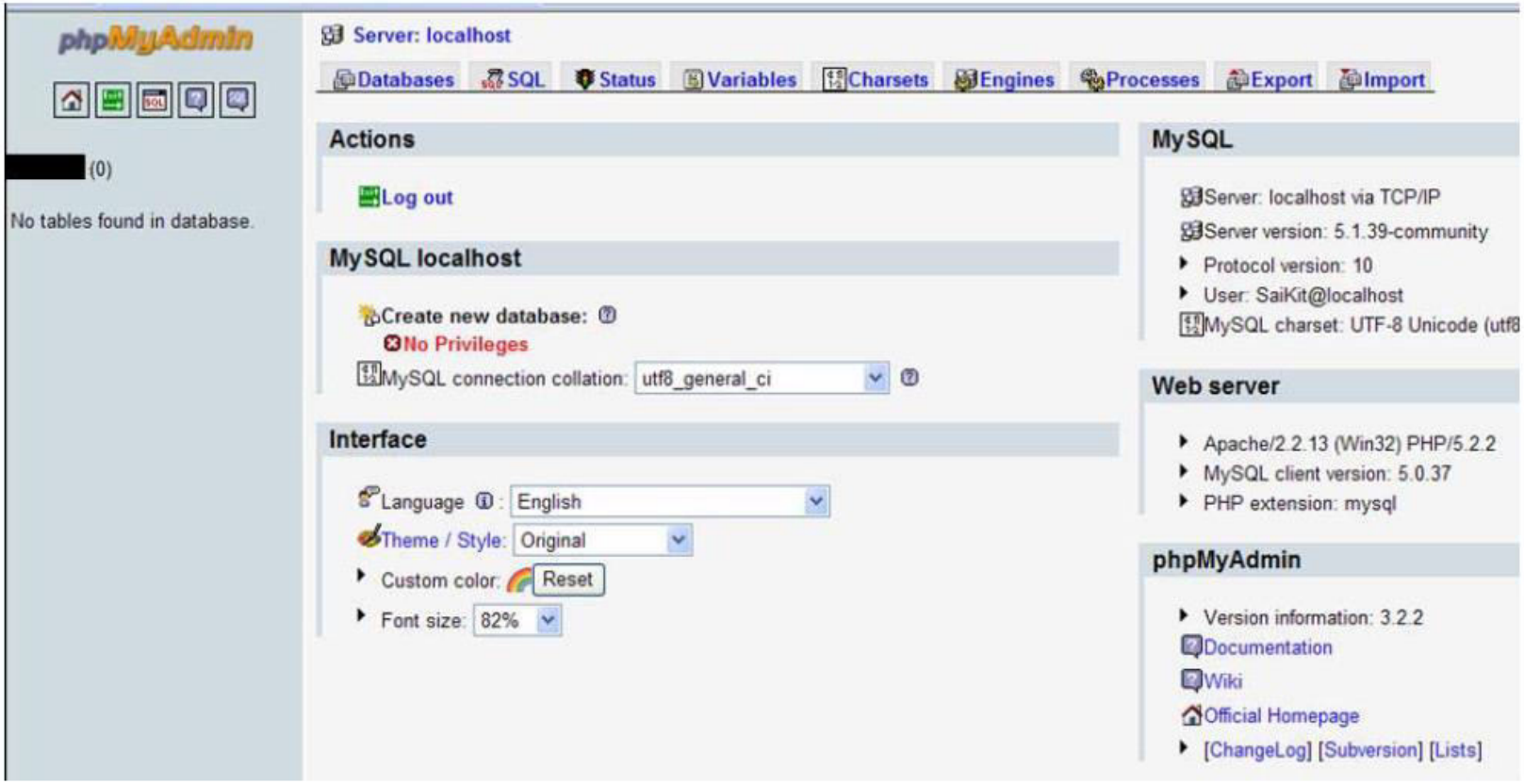
- phpMyAdmin is an open source web application, written in PHP for managing MySQL databases.
- Currently it can create and drop databases, create/drop/alter tables, delete/edit/add fields, execute any SQL statement, manage keys on fields, manage privileges, export data into various formats and is available in 50 languages.

- We can connect to and manipulate our database in two ways:

  (1) Use phpMyAdmin to manage your MySQL database directly.

  (2) Use PHP programming language to write a CGI program on a web server (namely: the iHome server) that can directly access our MySQL database.

The default welcome screen for phpMyAdmin:



Once PhpMyAdmin application open , we see different areas.

phpMyAdmin

- emp (1)
- information_schema (28)
- mysql (23)
- suman (2)

Please select a database

Server: localhost

Databases | SQL | Status | Variables | Charsets | Engines | Privileges | Binary log | Processes | Export | Import

Actions

MySQL localhost

Create new database

Collation | Create

MySQL connection collation: utf8_general_ci

Interface

Language : English

Theme / Style: Original

Custom color: Reset

Font size: 82%

MySQL

Server: localhost (MySQL host info: localhost via TCP/IP)

Server version: 5.1.36-community-log

- Protocol version: 10
- User: root@localhost

MySQL charset: UTF-8 Unicode (utf8)

Web server

- Apache/2.2.11 (Win32) PHP/5.3.0
- MySQL client version: mysqlnd 5.0.5-dev - 081106 - $Revision: 1.3.2.27 $
- PHP extension: mysqli

phpMyAdmin

- Version information: 3.2.0.1

Documentation

Wiki

Official Homepage

- [ChangeLog] [Subversion] [Lists]

phpMyAdmin

- In the upper part we find the server hostname.
- The databases which you will manage are stored on the same server as the software and the hostname is: localhost.
- Under it there is information regarding the MySQL server, the MySQL client and the PhpMyAdmin version.
- Next, the MySQL charset and able to define the MySQL connection collation.
- In the right column we can change the default language, alter the style, customize the theme color and the font size. Also there is a notice links to PhpMyAdmin resources.

Engines

- The Storage Engines link opens a list with all the engines supported by the MySQL server. The default one is MyISAM.
- Another popular storage engine, used by many databases is InnoDB.

localhost

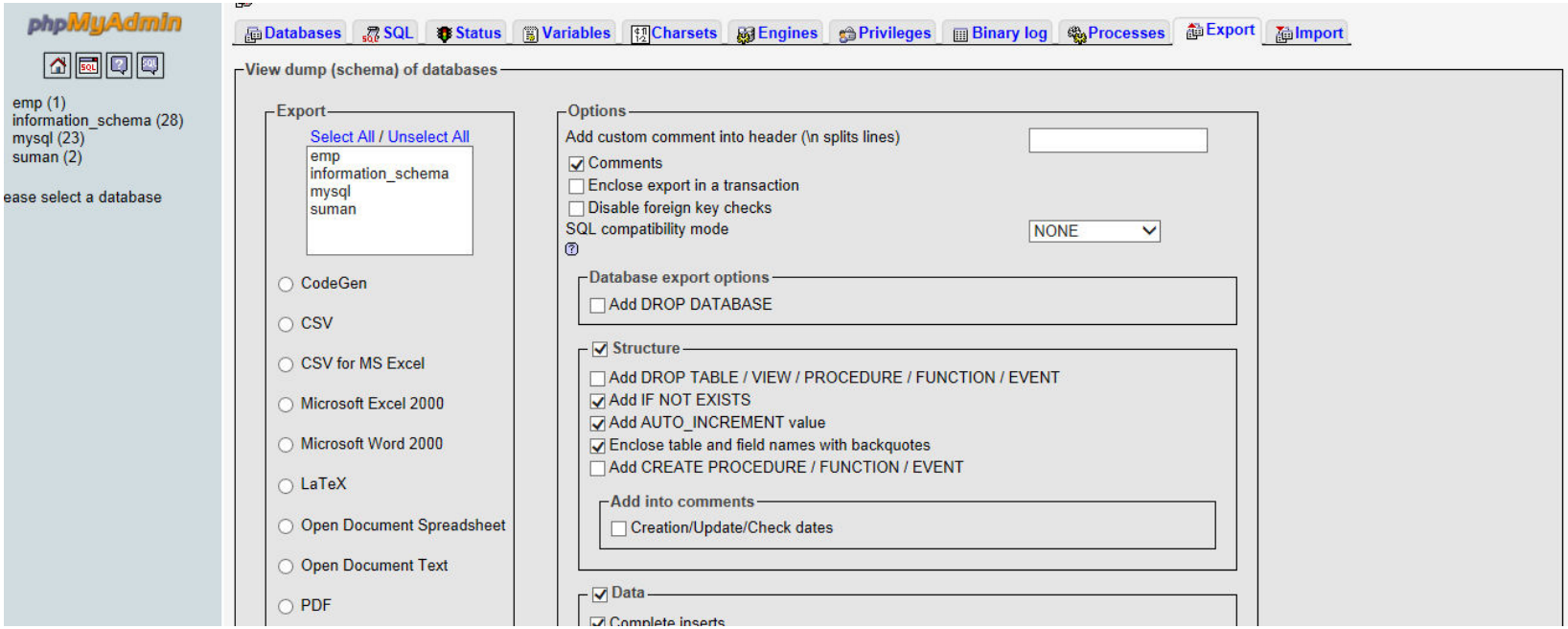
Databases | SQL | Status | Processes | Export | Import | Variables | Cha

Storage Engines

Storage Engine	Description
MyISAM	MyISAM storage engine
CSV	CSV storage engine
MRG_MYISAM	Collection of identical MyISAM tables
BLACKHOLE	/dev/null storage engine (anything you write to it disappears)
MEMORY	Hash based, stored in memory, useful for temporary tables
FEDERATED	Federated MySQL storage engine
ARCHIVE	Archive storage engine
InnoDB	Percona-XtraDB, Supports transactions, row-level locking, and foreign keys
PERFORMANCE_SCHEMA	Performance Schema

Export

In the Export section we can export our database tables content in different formats (CSV, SQL, PDF, Microsoft Excel, Microsoft Word, XML, and many more). we can select all the database tables or just pick some of them.



)

Import

In the Import section we can import our database tables from a file, saved on our local computer

phpMyAdmin

- emp (1)
- information_schema (28)
- mysql (23)
- suman (2)

Please select a database

Server: localhost

Databases | SQL | Status | Variables | Charsets | Engines | Privileges | Binary log | Processes | Export | Import

File to import

Location of the text file [ ] Browse... (Max: 2,048 KiB)

Character set of the file: utf8

Imported file compression will be automatically detected from: None, gzip, zip

Partial import

☑ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit. This might be good way to import large files, however it can break transactions.

Number of records (queries) to skip from start 0

Format of imported file

◉ SQL

Options

SQL compatibility mode NONE

☑ Do not use AUTO_INCREMENT for zero values

Go

Open new phpMyAdmin windo

)

Creating tables in our MySQL database.

1. Using the GUI of phpMyAdmin program.
2. Using a SQL "CREATE TABLE …" command issued from phpMyAdmin GUI.
3. Using a CGI program to connect to your MySQL DB, and sending the command from our CGI program

Before create any table(s)

- Names of all the tables.
- All the attributes for each table
- The domain constraint(s) for each attribute
- The primary key for each table
- The referential constraints (Foreign keys)

Conventions: Try to use a consistent convention for all names that you will assign. For example:

- All table names: First letter capitalized with no underscores: e.g. Employee, WorksOn,…
- All attribute names: lower case with underscores: name, ssn, birth_date, …
- All constraint names: lower case, underscored; for example, a foreign key constraints from

  Employee to Department table

  →fk_employee_department.

Data Type to Use

- For all integer values, use Type = INT
- For all real numbers, use Type = FLOAT
- For all text fields, use Type = VARCHAR, and Length = 50 (or some other reasonable number)
- For Dates (e.g. Birth Date), use Type = DATE

Create a database in phpmyadmin

phpMyAdmin

- emp (1)
- information_schema (28)
- mysql (23)
- suman (2)

Please select a database

Server: localhost

Databases | SQL | Status | Variables | Charsets | Engines | Privileges | Binary log | Processes | Export | Import

Actions

MySQL localhost

Create new database

Collation | Create

MySQL connection collation: utf8_general_ci

Interface

Language : English

Theme / Style: Original

Custom color: Reset

Font size: 82%

MySQL

Server: localhost (MySQL host info: localhost via TCP/IP)

Server version: 5.1.36-community-log

Protocol version: 10

User: root@localhost

MySQL charset: UTF-8 Unicode (utf8)

Web server

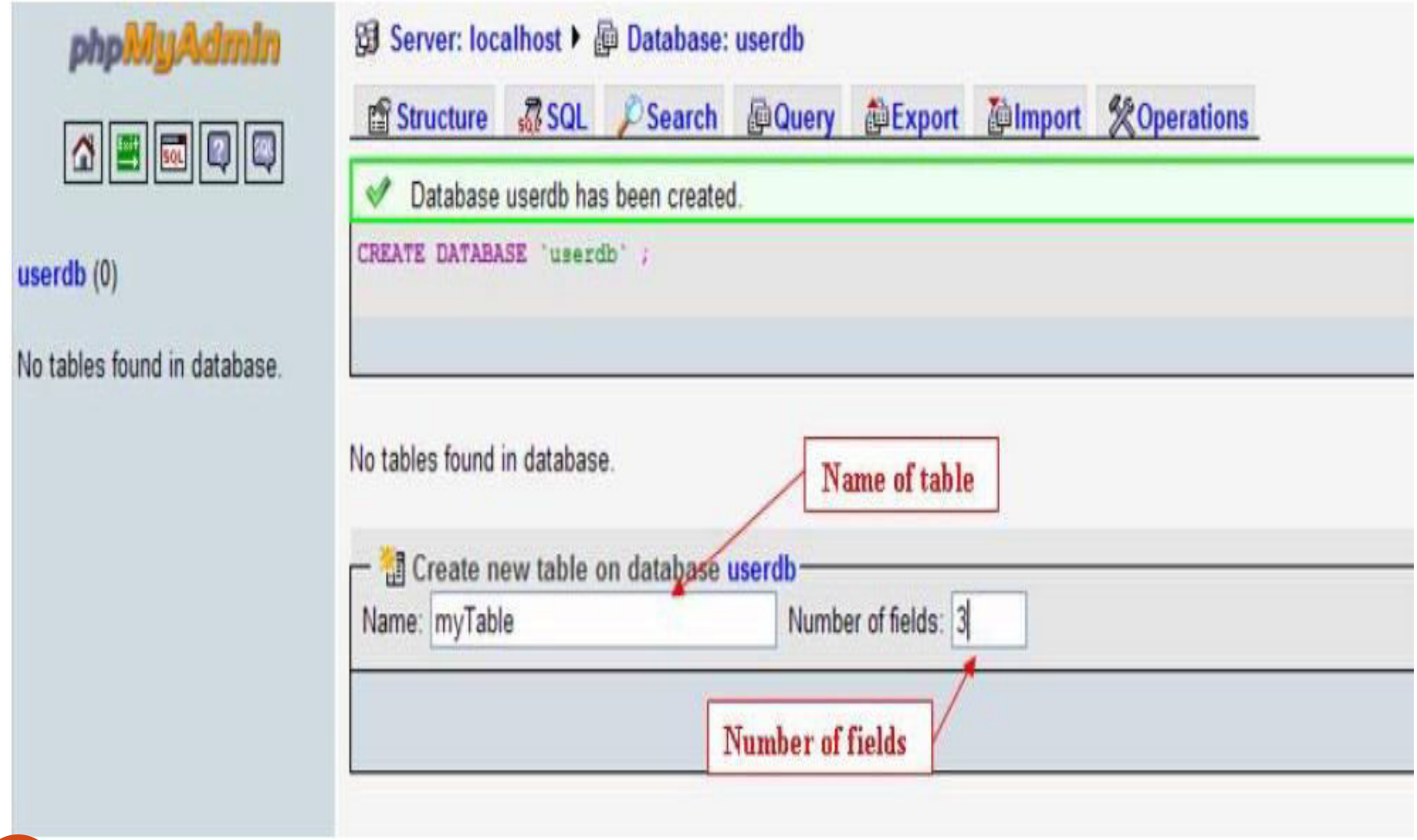
Apache/2.2.11 (Win32) PHP/5.3.0

MySQL client version: mysqlnd 5.0.5-dev - 081106 - $Revision: 1.3.2.27 $

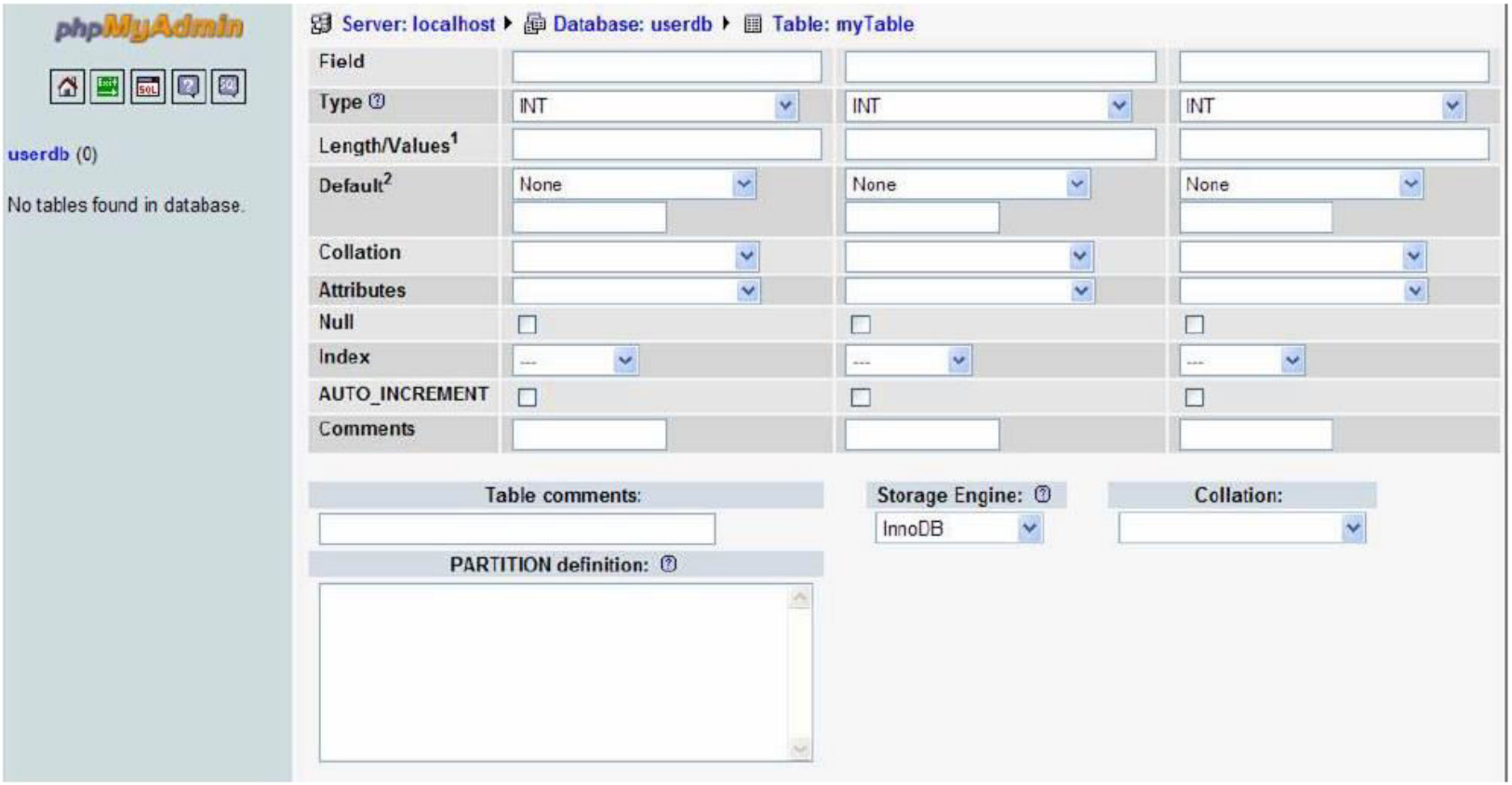
PHP extension: mysqli

phpMyAdmin

After creating a database than create a table in the database by using database by selecting database or by sql command use database name



Enter name of the field and specify the type, length, set the primary key, etc.



After creating database and table

phpMyAdmin

Server: localhost ▸ Database: userdb ▸ Table: myTable

Browse | Structure | SQL | Search | Insert | Export | Import

Table `userdb`.`myTable` has been created.

```
CREATE TABLE `userdb`.`myTable` (
  `Name` VARCHAR( 20 ) NOT NULL ,
  `SID` VARCHAR( 8 ) NOT NULL ,
  `Gender` VARCHAR( 1 ) NOT NULL
) ENGINE = INNODB;
```

userdb (1)

mytable

	Field	Type	Collation	Attributes	Null	Default	Extra
☐	Name	varchar(20)	latin1_swedish_ci		No	*None*	
☐	SID	varchar(8)	latin1_swedish_ci		No	*None*	
☐	Gender	varchar(1)	latin1_swedish_ci		No	*None*	

Check All / Uncheck All *With selected:*

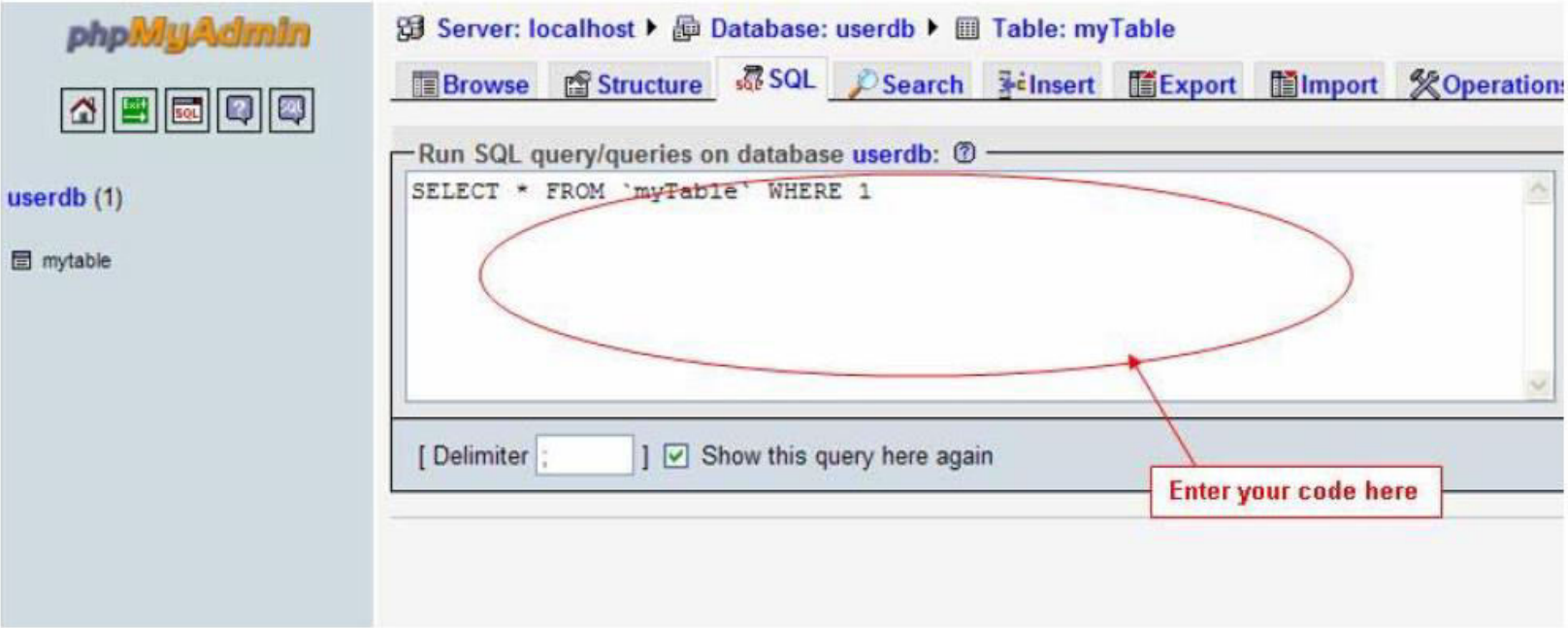
Print view | Relation view | Propose table structure

Add 1 field(s) ◉ At End of Table ○ At Beginning of Table ○ After Name Go

How to use SQL command to create your own table?

1> Click “SQL” in the middle

2> Type your SQL query in the box



)

Note: The keyword clustered is needed since the primary key has more than one attribute. If the primary key has only one attribute, don't need the word "**clustered**"

phpMyAdmin

Server: localhost ▸ Database: userdb ▸ Table: myTable

Browse | Structure | SQL | Search | Insert | Export | Import | Operations | Em

Your SQL query has been executed successfully (Query took 0.0708 sec)

```
CREATE TABLE sales(
  stor_idCHAR( 4 ) NOT NULL ,
  ord_num VARCHAR( 20 ) NOT NULL ,
  DATE DATETIME NOT NULL ,
  PRIMARY KEY clustered( stor_id, ord_num )
)
```

userdb (2)

- mytable
- sales

Run SQL query/queries on database userdb:

```
create table sales
(stor_id char(4) not null,
ord_num varchar(20) not null,
date datetime not null,
primary key clustered (stor_id, ord_num))
```

Fields

Name
SID
Gender

[Delimiter ;] Show this query here again

)

Example 2:

```
create table salesdetail
(stor_id char(4) not null,
ord_num varchar(20) not null,
title_id int not null references titles(title_id),
qty smallint default 0 not null,
discount float not null,
constraint salesdet_constr
foreign key (stor_id, ord_num) references sales(stor_id,
   ord_num))
```

phpMyAdmin

Server: localhost ▸ Database: userdb ▸ Table: salesdetail

Browse | Structure | SQL | Search | Insert | Export | Import | Opera

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 sec)

```
SELECT *
FROM `salesdetail`
LIMIT 0 , 30
```

Profiling [Ed

userdb (3)

- mytable
- sales
- salesdetail

	Field	Type	Collation	Attributes	Null	Default	Extra		
☐	stor_id	char(4)	latin1_swedish_ci		No	None			
☐	ord_num	varchar(20)	latin1_swedish_ci		No	None			
☐	title_id	int(11)			No	None			
☐	qty	smallint(6)			No	0			
☐	discount	float			No	None			

Check All / Uncheck All *With selected:*

Print view | Relation view | Propose table structure

Add 1 field(s) ◉ At End of Table ○ At Beginning of Table ○ After stor_id Go

+ Details...

Note:

1. We can specify default values on attributes (that is, if a record is created without this value specified, then the value is set to the default, as in the case of attribute qty.
2. There is one referential constraint on the attribute **title_id. It refers to the attribute title_id in a table called titles. Such a constraint may be set if the attribute being referred (title_id) is not the primary key for the referred table (titles).**
3. We shall only use referential constraints that are foreign keys.
4. Each constraint must have a unique name

Lecture-42-46

Mysql Connection with PHP

Creating Database Tables

- Create **database to hold the tables:**

  CREATE DATABASE database name;

- To create a table called notes:

  USE database name;

The Basic Queries

- CREATE

create databases and tables

- SELECT

select table rows based on certain conditions

- DELETE

delete one or more rows of a table

- INSERT

insert a new row in a table

- UPDATE

update rows in a table

- ALTER

alter the structure of a table

PHP MySQL Functions

- Connecting to a Database
- Making a query
- Using results of a query
- freeing resources
- closing the connection

PHP Connect to the MySQL Server

- Use the PHP mysql_connect() function to open a new connection to the MySQL server.

Open a Connection to the MySQL Server

- Before we can access data in a database, we must open a connection to the MySQL server.
- In PHP, this is done with the function. :
  - mysql_connect()

Syntax

mysql_connect(host, username, password);

parameter	description
host	**Either a host name or an IP address**
username	The MySQL user name
password	The password to log in with

Error Reporting in PHP

- mysql_error(link)
  - Return an error string or error number
  - the link is optional
  - if not supplied the last opened link is used.
  - Empty string is returned if there is no error.

Example

- **mysql_error();**

mysql_no(link)

- Return the error number
- the link is optional
- if not supplied the last opened link is used.
- 0 is returned if there is no error.

Example

- **mysql_no();**

The default error handling in PHP is very simple. An error message with filename, line number and a message describing the error is sent to the browser.

- PHP Error Handling

- When creating scripts and web applications, error handling is an important part.
- different error handling methods:
  - Simple "die()" statements
  - Custom errors and error triggers
  - Error reporting

Basic Error Handling: Using the die() function

```
<?php
  $file=fopen(“emp.txt“ ,"r");
  ?>
```

If the file does not exist you might get an error like this:

- **Warning**: fopen(emp.txt) [function.fopen]: failed to open stream: No such file or directory in **C:\wamp\www\gargidatabase\file.php** on line **2**

To prevent the user from getting an error message like the one above, we test whether the file exist before we try to access it:

```
<?php
if(!file_exists(“emp.txt"))
  {
  die("File not found");
  }
else
{
  $file=fopen(“emp.txt","r");
  }
?>
```

- Now if the file does not exist you get an error like this:

  File not found

Creating a Custom Error Handler

- Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.
- This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax

error_function(error_level ,error_message, error_file,error_line,error_context)

Parameter	Description
error_level	Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels
error_message	Required. Specifies the error message for the user-defined error
error_file	Optional. Specifies the filename in which the error occurred
error_line	Optional. Specifies the line number in which the error occurred
error_context	Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Error Report levels

These error report levels are the different types of error the user-defined error handler can be used for:

Value	Constant	Description
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

Example:

```
<?php
function customError($errno, $errstr)
  {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
  }
?>
```

Set Error Handler

- The default error handler for PHP is the built in error handler.
- It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. Example:

  ```
  set_error_handler("customError");
  ```

- Since we want our custom function to handle all errors, the set_error_handler() only needed one parameter, a second parameter could be added to specify an error level.

Example:

```
<?php
//error handler function
function customError($errno, $errstr)
  {
  echo "<b>Error:</b> [$errno] $errstr";
  }

//set error handler
set_error_handler("customError");

//trigger error
echo"$test";
?>
```

output

- **Error:** [8] Undefined variable: test /* if(!file_exists("emp1.txt")) { die("File not found"); } else { $file=fopen("emp1.txt","r"); }*/ ?>

Trigger an Error

- In a script where users can input data it is useful to trigger errors when an illegal input occurs.
- In PHP, this is done by the trigger_error() function.

Example:

```
<?php
$test=2;
if ($test>1)
{
trigger_error("Value must be 1 or below");
}
?>
```

output

- **Notice**: Value must be 1 or below in **C:\wamp\www\database\filecheck.php** on line **6** /*//error handler function function customError($errno, $errstr) { echo "**Error:** [$errno] $errstr"; } //set error handler set_error_handler("customError"); //trigger error echo"$test";*/ /* if(!file_exists("emp1.txt")) { die("File not found"); } else { $file=fopen("emp1.txt","r"); }*/ ?>

Possible error types:

- E_USER_ERROR - Fatal user-generated run-time error. Errors that can not be recovered from. Execution of the script is halted
- E_USER_WARNING - Non-fatal user-generated run-time warning. Execution of the script is not halted
- E_USER_NOTICE - Default. User-generated run-time notice. The script found something that might be an error, but could also happen when running a script normally

Example:

```
<?php
//error handler function
function customError($errno, $errstr)
  {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
  }
//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1)
  {
  trigger_error("Value must be 1 or below",E_USER_WARNING);

?>
```

Output:

- **Error:** [512] Value must be 1 or below
  Ending Script

Example: connect Mysql database with php

```
<html>
<body>
<?php
echo"database connection and open a database<br>";
$link=mysql_connect("localhost","root","") or die("could not
    connect:".mysql_error());
echo"connected successfully<br>";
$db=mysql_select_db("gargi") or die("could not select database:
    ".mysql_error());
echo "database selected successfully<br>";
mysql_close($link);a
?>
</body>
</html>
```

Create a Database

- The CREATE DATABASE statement is used to create a database table in MySQL.
- We must add the CREATE DATABASE statement to the mysql_query() function to execute the command.

Example: php script to create a database

```
<html>
<body>
<?php
echo"database connection and open a database<br>";
$link=mysql_connect("localhost","root","") or die("could not connect:".mysql_error());
echo"connected successfully<br>";
$query="create database student";
$result=mysql_query($query,$link);
echo"database is created";
echo"<hr>";
$db=mysql_select_db("student") or die("could not select database: ".mysql_error());
echo "database selected successfully<br>";
mysql_close($link);
?>
</body>
</html>
```

Example: PHP script to create a table in php

```
<html>
<body>
<?php
echo"database connection and open a database<br>";
$link=mysql_connect("localhost","root","") or die("could not
   connect:".mysql_error());
echo"connected successfully<br>";v
$db=mysql_select_db("gargi") or die("could not select database:
   ".mysql_error());
echo "database selected successfully<br>";
$query="create table test1(name varchar(30) not null ,age int)";
$result=mysql_query($query) or die ("Query failed:".mysql_error());
echo"table is created";
mysql_close($link);
?>
```

Example: php script to insert a value in table

```
<?php
echo"database connection and open a database<br>";
$link=mysql_connect ("localhost","root","") or die("could not
    connect:".mysql_error());
echo"connected successfully<br>";
$db=mysql_select_db("student") or die("could not select database:
    ".mysql_error());
    echo "database selected successfully<br>";
$sql=mysql_query("insert into t1(name, age)values('gargi','30')");
echo"value  is inserted";
mysql_close($link);
?>
```

Example: php script to select data from table

```
<?php
echo"database connection and open a database<br>";
$link=mysql_connect("localhost","root","") or die("could not connect:".mysql_error());
echo"connected successfully<br>";
$db=mysql_select_db("student") or die("could not select database: ".mysql_error());
echo "database selected successfully<br>";
$sql=mysql_query("select *from t1");
//$result=mysql_query($sql) or die ("Query failed:".mysql_error());
while($row = mysql_fetch_array($sql))
{
$name=$row["name"];
$age=$row["age"];
}
echo $name;
echo $age;
mysql_close($link);
?>
```

Example: PHP script for insert from form and select *from table.

```
<html>
<body>
<?php
if(isset($_REQUEST['t1']))
{

$link=mysql_connect("localhost","root","") or die("could not connect:".mysql_error());
echo"connected successfully<br>";
$db=mysql_select_db("company") or die("could not select database: ".mysql_error());
echo "database selected successfully<br>";
echo"<hr>";
$v1=$_REQUEST['t1'];
echo $v1;
$v2=$_REQUEST['t2'];
echo $v2;
$v3=$_REQUEST['t3'];
echo $v3;
```

```
mysql_query("insert into emp (name, desig,state) values('".$v1."','".$v2."','".$v3."')
}
?>
<form action="forminsert.php" method="post">
Employee name:<input type="text" name="t1" size=40>
Employee desig:<input type="text" name="t2" size=40>
Employee state:<input type="text" name="t3" size=30>
<input type="submit" value="ok"  name="ok" />
</form>
<?php
$ds=mysql_query("select*from emp");
while($dr=mysql_fetch_array($ds))
{
echo"$dr[0]-$dr[1]-$dr[2]-$dr[3] <br>";
}
?>
```

Example

```
<?php
echo"database connection and open a database<br>";
$link=mysql_connect ("localhost", "root“ ,"") or die("could not
   connect:".mysql_error());
echo"connected successfully<br>";
$db=mysql_select_db("student") or die("could not select
   database: ".mysql_error());
echo "database selected successfully<br>";
$fields=mysql_list_fields("student","t1");
$num_col=mysql_num_fields($fields);
echo "$fields <br>";
echo"$num_col";
?>
```

Following example to display all the records from tutorials_tbl table using MYSQL_NUM argument.

```
<?php
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
die('Could not connect: ' . mysql_error());
}
```

```
$sql = "SELECT empid, name FROM emp";
  mysql_select_db('company');
 $retval = mysql_query( $sql, $conn );
 if(! $retval )
  {
   die('Could not get data: ' . mysql_error());
  }

  while($row = mysql_fetch_array($retval, MYSQL_NUM))
   {
      echo "EMP ID :{$row[0]} <br> ".
          "Name: {$row[1]} <br> ". "--------------------------------<br>";

}
 mysql_free_result($retval);

 echo "Fetched data successfully\n";

mysql_close($conn);
?>
```

Updating Data Using PHP Script:

```
<?php
$dbhost = “localhost”;
$dbuser = “root”;
$dbpass = “”;
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'UPDATE tutorials_tbl
        SET tutorial_title="Learning JAVA"
        WHERE tutorial_id=3';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not update data: ' . mysql_error());
}
echo "Updated data successfully\n";
mysql_close($conn);
?>
```

Deleting Data Using PHP Script:

```
<?php
$dbhost = 'localhost;
$dbuser = 'root';
$dbpass = '';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
die('Could not connect: ' . mysql_error());
}
$sql = 'DELETE FROM tutorials WHERE tutorial_id=3';
 mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
 die('Could not delete data: ' . mysql_error());
}
echo "Deleted data successfully\n";mysql_close($conn);
?>
```

Releasing Memory:

It's a good practice to release cursor memory at the end of each SELECT statement. This can be done by using PHP function mysql_free_result().

```
<?php
$dbhost = 'localhost’;
$dbuser = 'root';
$dbpass = '';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
```

```
$sql = 'SELECT tutorial_id, tutorial_title,
            tutorial_author , submission_date
       FROM tutorials_tbl';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_NUM))
{
    echo "Tutorial ID :{$row[0]}  <br> ".
         "Title: {$row[1]} <br> ".
         "Author: {$row[2]} <br> ".
         "Submission Date : {$row[3]} <br> ".
         "--------------------------------<br>";
}
mysql_free_result($retval);
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```

Using LIKE clause inside PHP Script:

```
<? php
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$conn = mysql_connect ($dbhost, $dbuser, $dbpass);
if(! $conn )
{
  die('Could not connect: ' . mysql_error());
}
$sql = 'SELECT tutorial_id, tutorial_title,
              tutorial_author, submission_date
        FROM tutorials_tbl
        WHERE tutorial_author LIKE "%jay%"';

mysql_select_db('TUTORIALS');
$retval = mysql_query( $sql, $conn );
```

```
if(! $retval )
{
  die('Could not get data: ' . mysql_error());
}
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "Tutorial ID :{$row['tutorial_id']}  <br> ".
         "Title: {$row['tutorial_title']} <br> ".
         "Author: {$row['tutorial_author']} <br> ".
         "Submission Date : {$row['submission_date']} <br> ".
         "--------------------------------<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```