**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | 11/06/2020 | Name: | Gaganashree P |
| Course: | Vlsi | USN: | 4al15ec024 |
| Topic: | MOS transistor basics-II and III Source:NPTEL | Semester & Section: | 8th &"A" section |
| Github Repository: | Gaganashree-P |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report–**  1.**MOS transistor basics-II and III** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DAILY ASSESSMENT FORMAT**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | Date: | 11-06-20 | Name: | Gaganashree P | | Course: | Beginner PHP and MySQL Tutorial | USN: | 4AL15EC024 | | Topic: | PHP and MySQL Tutorial | Semester & Section: | 8th A | | Github Repository: | Gaganashree-P |  |  | | | | | | |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Report –**  1. Php functions  PHP MySQLi Functions  PHP MySQLi Introduction  The MySQLi functions allows you to access MySQL database servers.  Note: The MySQLi extension is designed to work with MySQL version 4.1.13 or newer.  Installation / Runtime Configuration  For the MySQLi functions to be available, you must compile PHP with support for the MySQLi extension.  The MySQLi extension was introduced with PHP version 5.0.0. The MySQL Native Driver was included in PHP version 5.3.0.  For installation details, go to: http://php.net/manual/en/mysqli.installation.php  For runtime configuration details, go to: http://php.net/manual/en/mysqli.configuration.php  PHP MySQLi Functions  Function Description  affected\_rows() Returns the number of affected rows in the previous MySQL operation  autocommit() Turns on or off auto-committing database modifications  begin\_transaction() Starts a transaction  change\_user() Changes the user of the specified database connection  character\_set\_name() Returns the default character set for the database connection  close() Closes a previously opened database connection  commit() Commits the current transaction  connect() Opens a new connection to the MySQL server  connect\_errno() Returns the error code from the last connection error  connect\_error() Returns the error description from the last connection error  data\_seek() Adjusts the result pointer to an arbitrary row in the result-set  debug() Performs debugging operations  dump\_debug\_info() Dumps debugging info into the log  errno() Returns the last error code for the most recent function call  error() Returns the last error description for the most recent function call  error\_list() Returns a list of errors for the most recent function call  fetch\_all() Fetches all result rows as an associative array, a numeric array, or both  fetch\_array() Fetches a result row as an associative, a numeric array, or both  fetch\_assoc() Fetches a result row as an associative array  fetch\_field() Returns the next field in the result-set, as an object  fetch\_field\_direct() Returns meta-data for a single field in the result-set, as an object  fetch\_fields() Returns an array of objects that represent the fields in a result-set  fetch\_lengths() Returns the lengths of the columns of the current row in the result-set  fetch\_object() Returns the current row of a result-set, as an object  fetch\_row() Fetches one row from a result-set and returns it as an enumerated array  field\_count() Returns the number of columns for the most recent query  field\_seek() Sets the field cursor to the given field offset  get\_charset() Returns a character set object  get\_client\_info() Returns the MySQL client library version  get\_client\_stats() Returns statistics about client per-process  get\_client\_version() Returns the MySQL client library version as an integer  get\_connection\_stats() Returns statistics about the client connection  get\_host\_info() Returns the MySQL server hostname and the connection type  get\_proto\_info() Returns the MySQL protocol version  get\_server\_info() Returns the MySQL server version  get\_server\_version() Returns the MySQL server version as an integer  info() Returns information about the last executed query  init() Initializes MySQLi and returns a resource for use with real\_connect()  insert\_id() Returns the auto-generated id from the last query  kill() Asks the server to kill a MySQL thread  more\_results() Checks if there are more results from a multi query  multi\_query() Performs one or more queries on the database  PHP mysqli\_connect function  The PHP mysql connect function is used to connect to a MySQL database server.  It has the following syntax.  <?php;  $db\_handle = mysqli\_connect($db\_server\_name, $db\_user\_name, $db\_password);  ?>  HERE,  “$db\_handle” is the database connection resource variable.  “mysqli\_connect(…)” is the function for php database connection  “$server\_name” is the name or IP address of the server hosting MySQL server.  “$user\_name” is a valid user name in MySQL server.  “$password” is a valid password associated with a user name in MySQL server.  PHP mysqli\_query function  The mysqli\_query function is used to execute SQL queries.  The function can be used to execute the following query types;  Insert  Select  Update  delete  It has the following syntax.  <?php  mysqli\_query($db\_handle,$query) ;  ?>  HERE,  “mysqli\_query(…)” is the function that executes the SQL queries.  “$query” is the SQL query to be executed  “$link\_identifier” is optional, it can be used to pass in the server connection link  2.Using external files and images  Introduction  A Binary Large Object (BLOB) is a MySQL data type that can store binary data such as images, multimedia, and PDF files.  When creating applications that require a tightly-coupled database where images should be in sync with related data (for example, an employee portal, a student database, or a financial application), you might find it convenient to store images such as students’ passport photos and signatures in a MySQL database alongside other related information.  This is where the MySQL BLOB data type comes in. This programming approach eliminates the need for creating a separate file system for storing images. The scheme also centralizes the database, making it more portable and secure because the data is isolated from the file system. Creating backups is also more seamless since you can create a single MySQL dump file that contains all your data.  Retrieving data is faster, and when creating records you can be sure that data validation rules and referential integrity are maintained especially when using MySQL transactions.  In this tutorial, you will use the MySQL BLOB data type to store images with PHP on Ubuntu 18.04.  Prerequisites  To follow along with this guide, you will need the following:  An Ubuntu 18.04 server configured using the Initial Server Setup with Ubuntu 18.04 and a non-root user with sudo privileges.  Apache, MySQL, and PHP set up by following the guide on How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04. For this tutorial, it isn’t necessary to create virtual hosts, so you can skip Step 4.  Step 1 — Creating a Database  You’ll start off by creating a sample database for your project. To do this, SSH in to your server and then run the following command to log in to your MySQL server as root:  sudo mysql -u root -p  Enter the root password of your MySQL database and hit ENTER to continue.  Then, run the following command to create a database. In this tutorial we’ll name it test\_company:  CREATE DATABASE test\_company;  Once the database is created, you will see the following output:  Output  Query OK, 1 row affected (0.01 sec)  Next, create a test\_user account on the MySQL server and remember to replace PASSWORD with a strong password:  CREATE USER 'test\_user'@'localhost' IDENTIFIED BY 'PASSWORD';  You’ll see the following output:  Output  Query OK, 0 rows affected (0.01 sec)  To grant test\_user full privileges on the test\_company database, run:  GRANT ALL PRIVILEGES ON test\_company.\* TO 'test\_user'@'localhost';  Make sure you get the following output:  Output  Query OK, 0 rows affected (0.01 sec)  Finally, flush the privileges table in order for MySQL to reload the permissions:  FLUSH PRIVILEGES;  Ensure you see the following output:  Output  Query OK, 0 rows affected (0.01 sec)  Now that the test\_company database and test\_user are ready, you’ll move on to creating a products table for storing sample products. You’ll use this table later to insert and retrieve records to demonstrate how MySQL BLOB works.  Log out from the MySQL server:  QUIT;  Then, log back in again with the credentials of the test\_user that you created:  mysql -u test\_user -p  When prompted, enter the password for the test\_user and hit ENTER to continue. Next, switch to the test\_company database by typing the following:  USE test\_company;  Once the test\_company database is selected, MySQL will display:  Output  Database changed  Next, create a products table by running:  CREATE TABLE `products` (product\_id BIGINT PRIMARY KEY AUTO\_INCREMENT, product\_name VARCHAR(50), price DOUBLE, product\_image BLOB) ENGINE = InnoDB;  This command creates a table named products. The table has four columns:  product\_id: This column uses a BIGINT data type in order to accommodate a large list of products up to a maximum of 2⁶³-1 items. You’ve marked the column as PRIMARY KEY to uniquely identify products. In order for MySQL to handle the generation of new identifiers for inserted columns, you have used the keyword AUTO\_INCREMENT.  product\_name: This column holds the names of the products. You’ve used the VARCHAR data type since this field will generally handle alphanumerics up to a maximum of 50 characters—the limit of 50 is just a hypothetical value used for the purpose of this tutorial.  price: For demonstration purposes, your products table contains the price column to store the retail price of products. Since some products may have floating values (for example, 23.69, 45.36, 102.99), you’ve used the DOUBLE data type.  product\_image: This column uses a BLOB data type to store the actual binary data of the products’ images.  You’ve used the InnoDB storage ENGINE for the table to support a wide range of features including MySQL transactions. After executing this for creating the products table, you’ll see the following output:  Output  Query OK, 0 rows affected (0.03 sec)  Log out from your MySQL server:  QUIT;  You will get the following output  Output  Bye  The products table is now ready to store some records including products’ images and you’ll populate it with some products in the next step.  Step 2 — Creating PHP Scripts for Connecting and Populating the Database  In this step, you’ll create a PHP script that will connect to the MySQL database that you created in Step 1. The script will prepare three sample products and insert them into the products table.  To create the PHP code, open a new file with your text editor:  sudo nano /var/www/html/config.php  Then, enter the following information into the file and replace PASSWORD with the test\_user password that you created in Step 1:  /var/www/html/config.php  <?php  define('DB\_NAME', 'test\_company');  define('DB\_USER', 'test\_user');  define('DB\_PASSWORD', 'PASSWORD');  define('DB\_HOST', 'localhost');  $pdo = new PDO("mysql:host=" . DB\_HOST . "; dbname=" . DB\_NAME, DB\_USER, DB\_PASSWORD);  $pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  $pdo->setAttribute(PDO::ATTR\_EMULATE\_PREPARES, false);  Save and close the file.  In this file, you’ve used four PHP constants to connect to the MySQL database that you created in Step 1:  DB\_NAME : This constant holds the name of the test\_company database.  DB\_USER : This variable holds the test\_user username.  DB\_PASSWORD : This constant stores the MySQL PASSWORD of the test\_user account.  DB\_HOST: This represents the server where the database resides. In this case, you are using the localhost server.  The following line in your file initiates a PHP Data Object (PDO) and connects to the MySQL database:  Step 3 — Displaying Products’ Information From the MySQL Database  With the products’ information and images in the database, you’re now going to code another PHP script that queries and displays the products’ information in an HTML table on your browser.  To create the file, type the following:  sudo nano /var/www/html/display\_products.php  Then, enter the following information into the file:  /var/www/html/display\_products.php  <html>  <title>Using BLOB and MySQL</title>  <body>  <?php  require\_once 'config.php';  $sql = "SELECT \* FROM products";  $stmt = $pdo->prepare($sql);  $stmt->execute();  ?>  <table border = '1' align = 'center'><caption>Products Database</caption>  <tr>  <th>Product Id</th>  <th>Product Name</th>  <th>Price</th>  <th>Product Image</th>  </tr>  <?php  while ($row = $stmt->fetch(PDO::FETCH\_ASSOC)) {  echo '<tr>';  echo '<td>' . $row['product\_id'] . '</td>';  echo '<td>' . $row['product\_name'] . '</td>';  echo '<td>' . $row['price'] . '</td>';  echo '<td>' .  '<img src = "data:image/png;base64,' . base64\_encode($row['product\_image']) . '" width = "50px" height = "50px"/>'  . '</td>';  echo '</tr>';  }  ?>  </table>  </body>  </html>  Save the changes to the file and close it.  Conclusion  In this guide, you used the MySQL BLOB data type to store and display images with PHP on Ubuntu 18.04. You’ve also seen the basic advantages of storing images in a database as opposed to storing them in a file system. These include portability, security, and ease of backup. If you are building an application such as a students’ portal or employees’ database that requires information and related images to be stored together, then this technology can be of great use to you. | | | |