Optimizing Queries for Performance

Query optimization involves improving the efficiency and performance of database queries. Here are some strategies to optimize queries

- **Use indexes:** Ensure that tables are properly indexed to speed up query execution.
- **Limit result sets:** Retrieve only the necessary columns and rows to minimize data retrieval overhead.
- **Optimize JOIN operations:** Use appropriate JOIN types and avoid unnecessary JOINs.
- **Use WHERE and HAVING clauses effectively:** Filter data as early as possible in the query execution process.
- **Avoid subqueries when possible**: Instead, consider using JOINs or temporary tables.
- **Optimize database schema**: Normalize tables, eliminate redundant data, and use appropriate data types.

Maharah
Code Your Future

Query Optimization Techniques

Additional techniques for query optimization include:

- Caching query results.
- Analyzing query execution plans.
- Using stored procedures or prepared statements.
- Monitoring and adjusting database configuration parameters.

Maharah
Code Your Future

User Accounts in MySQL

User accounts are essential for controlling access to a MySQL database. Each user account has its own set of privileges, which determine what actions they can perform within the database.

Creating User Accounts:

To create a user account, you can use the CREATE USER statement. Specify the username and password for the new account. For example:

```
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
```

Managing User Accounts & Privileges

User accounts can be managed using various statements, such as:

- ALTER USER: Modify user account properties, such as the password or account lock status.
- DROP USER: Delete a user account from the database.

Granting Privileges:

Privileges determine the actions that a user can perform within the database. To grant privileges to a user, use the GRANT statement. Specify the privileges and the target database or tables. For example:

```
GRANT SELECT, INSERT ON database.table TO 'username'@'localhost';
```

Revoking Privileges:

Privileges determine the actions that a user can perform within the database. To grant privileges to a user, use the GRANT statement. Specify the privileges and the target database or tables. For example:

```
REVOKE INSERT ON database.table FROM 'username'@'localhost';
```

Maharah
Code Your Future

Best Practices for MySQL Security

Ensure the security of your MySQL database by following these best practices:

- Use strong, complex passwords for user accounts.
- Regularly update and patch your MySQL installation.
- Limit access to the database server to trusted users and hosts.
- Regularly review and audit user privileges.
- Implement encryption protocols for secure data transmission.
- Backup your database regularly to protect against data loss.

Maharah
Code Your Future

PHP Data Object (PDO)

The PHP Data Objects (PDO) extension provides a consistent interface for accessing databases in PHP. It is a database access layer that offers a set of methods and classes for performing database operations, allowing developers to interact with databases in a secure and efficient manner.

Connecting to PDO

To connect to the database with PDO, you create an instance of the PDO class. The constructor accepts parameters for the database source (DSN) and the username/ password if these are required.

```php
<?php
try {
 $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
} catch (PDOException $e) {
 echo "Error connecting to database: " . $e->getMessage();
}
```

- If there are errors connecting to the database then a **PDOException** will be thrown.
- It is very important to note that the stack trace of the exception will probably contain the full database connection details. Make sure that you catch it and don't let it be displayed RAW.
- To close a connection when you're done with it, you can set the PDO variable to null.
- Database connections are automatically closed at the end of your running script unless you make them persistent. Persistent database connections are not closed but are instead cached for another instance of the script to use. This reduces the overhead of needing to connect to the database every time your web application runs.

Maharah
Code Your Future

Fetching PDO Results

We use the PDO::fetch() method to retrieve data from a PDO result. PDO will maintain a cursor to traverse the result set and will use this to determine which element to return to you.

PDO::FETCH_ASSOC: Returns an associative array with your database columns as keys.
PDO::FETCH_NUM: Returns an array indexed by column number as returned by your result set.
PDO::FETCH_BOTH: Returns an array with both the indexes of ASSOC and NUM style fetches.

```php
<?php
$sth = $dbh->query('SELECT * FROM `users` WHERE age > :age');
$rows = $sth->fetch(PDO::FETCH_ASSOC);
foreach ($rows as $row) {
 // Process each row
}
```

Maharah
Code Your Future

Fetching PDO All

In PHP, to fetch the remaining rows from a result set after retrieving a portion of the data, you can use the fetchAll method.

This method retrieves all remaining rows from the result set as an array of associative arrays, where each array represents a row of data.

```php
<?php
$sth = $dbh->query('SELECT * FROM `users` WHERE age > :age');
$rows = $sth->fetchAll(PDO::FETCH_ASSOC);
printr($rows);
```

Maharah
Code Your Future

Executing SQL Queries (CRUD Operations)

PDO provides a convenient and secure way to execute SQL queries and perform CRUD operations (Create, Read, Update, Delete) with databases in PHP.

SELECT Query

Use PDOStatement and execute() method to execute SELECT queries.

```php
<?php
$sql = 'SELECT * FROM users WHERE age > :age';
$stmt = $pdo->prepare($sql);
$stmt->execute(['age' => 18]);

$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach ($rows as $row) {
 // Process each row
}
```

Maharah
Code Your Future

Executing SQL Queries (CRUD Operations)

INSERT Query

Use prepared statements to execute INSERT queries safely.

```php
<?php
$sql = 'INSERT INTO users (name, email) VALUES (:name, :email)';
$stmt = $pdo->prepare($sql);

$name = 'John Doe';
$email = 'john@example.com';
$stmt->execute(['name' => $name, 'email' => $email]);
$lastInsertId = $pdo->lastInsertId();
```

Maharah
Code Your Future

Executing SQL Queries (CRUD Operations)

UPDATE Query

Use prepared statements to execute UPDATE queries.

```php
<?php
$sql = 'UPDATE users SET email = :email WHERE id = :id';
$stmt = $pdo->prepare($sql);

$email = 'newemail@example.com';
$id = 1;
$stmt->execute(['email' => $email, 'id' => $id]);
$rowCount = $stmt->rowCount();
```

Maharah
Code Your Future

Executing SQL Queries (CRUD Operations)

DELETE Query

Use prepared statements to execute DELETE queries.

```php
<?php
$sql = 'DELETE FROM users WHERE id = :id';
$stmt = $pdo->prepare($sql);

$id = 1;

$stmt->execute(['id' => $id]);
$rowCount = $stmt->rowCount();
```

Maharah
Code Your Future

Prepared Statements in PDO

Not all database engines support prepared statements and this is the only feature that PDO will emulate for adapters that don't.
The syntax for a prepared statement in PDO is very similar to using a native function.
Prepared statements with parameter binding provide protection against SQL injection attacks.

```php
<?php
$sql = 'INSERT INTO users (name, email) VALUES (:name, :value)';

$stmt = $dbh->prepare($sql);
$stmt->bindValue(':name', 'mohamed');
$stmt->bindValue(':email','mohamed@maharah.com', PDO::PARAM_STR);
$stmt->execute();
```

Maharah
Code Your Future

PDO Reference

https://www.php.net/manual/en/class.pdostatement.php

Maharah
Code Your Future

```sql
SELECT `question`, `answer`

FROM `questions`;
```

This project is funded by the European Union

UNDP

Maharah
Code Your Future

LibyanSpider