Mysqli SELECT query with prepared statements

Before running any query with mysqli, make sure you've got a properly configured mysqli connection variable that is required in order to run SQL queries and to inform you of the possible errors.

SELECT query with prepared statements

You **must** always use `prepared statements` for any SQL query that would contain a PHP variable. To do so, always follow the below steps:

- create a correct SQL SELECT statement. Test it in mysql console/phpmyadmin if needed
- **replace all variables in the query with with question marks** (called placeholders or parameters)
- Prepare the resulting query
- Bind all variables to the previously prepared statement
- Execute the statement
- get the mysqli result variable from the statement.
- fetch your data

Long story short, here is the code:

```
$sql = "SELECT * FROM users WHERE id=?"; // SQL with parameters
$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $id);
$stmt->execute();
$result = $stmt->get_result(); // get the mysqli result
$user = $result->fetch_assoc(); // fetch data
```

And have your SELECT query executed without a single syntax error or SQL injection.

Detailed explanation

Let's see what does every line of this code mean

```
$sql = "SELECT * FROM users WHERE id=?";
```

Like it was said above, first we are writing an SQL query where all variables are substituted with question marks.

IMPORTANT! there should be no quotes around question marks, you are adding placeholders, not strings.

```
$stmt= $conn->prepare($sql);
```

Then, the query is prepared. The idea is very smart. To avoid even a possibility of the SQL injection or a syntax error caused by the input data, the query and the data are sent to database server *separately*. So it goes on here: with `prepare()` we are sending the query to the database server ahead. A special variable, a *statement* is created as a result. We would use this variable from now on.

```
$stmt->bind_param("i", $id);
```

Then variables must be bound to the statement. The call consists of two parts - the string with types and the list of variables. With mysqli, you have to designate the type for each bound variable. It is represented by a single letter in the first parameter. The number of letters should be always equal to the number of variables. The possible types are

- i for integer
- d for double (float)
- s for string
- b for blobs

A HINT: you can almost always safely use "s" for any variable.

So now you can tell that "s" means "there would be 1 variable, of string type".

```
$stmt->execute();
```

Then the query finally gets executed. Means variables get sent to the database server and the query is actually executed.

NOTE that you don't have to check the execution result. Given you have the proper connection code mentioned above, in case of error mysqli will raise an error automatically.

```
$result = $stmt->get_result(); // get the mysqli result
```

Here we are calling a very smart function. By default, for some reason it's impossible to fetch a familiar array (like we did with `mysql_fetch_array()`) from a mysqli statement. So this function is here to help, getting a mysqli result from a mysqli statement.

```
$user = $result->fetch_assoc(); // fetch data
```

Finally, fetching a row using a familiar `fetch_assoc()` method.

Getting multiple rows into array

Generally, getting multiple rows would involve a familiar while loop:

```
while ($row = $stmt->fetch_assoc()) { echo $row['name']; }
```

However, in a modern web-application the database interaction is separated from the HTML output. It makes the code much cleaner and more flexible. It means that we should never print our data using a while loop but rather collect it into array and then use this array for the output.
And mysqli has a handy function that instantly returns an array from the query result: `mysqli_fetch_all()`.
So instead of four lines

```
$data = [];
while ($row = $stmt->fetch_assoc()) { data[] = $row; }
```

it could be just a single line:

```
$data = $result->fetch_all(MYSQLI_ASSOC);
```

By default this function returns enumerated arrays, so to get associative arrays the `MYSQLI_ASSOC` mode must be set explicitly.

SELECT query with a helper function

As you may noted, the code for a prepared statement is quite verbose. If you like to build a code like a Lego figure, with shining ranks of operators, you may keep it as is. If you, like me, hate useless repetitions and like to write concise and meaningful code, then there is a simple helper function. With it, the code will become two times shorter:

```
$sql = "SELECT * FROM users WHERE id=?"; // SQL with parameters
$stmt = prepared_query($conn, $sql, [$id]);
$user = $stmt->get_result()->fetch_assoc(); // fetch a row
// oder
$users = $stmt->get_result()->fetch_all(MYSQLI_ASSOC);

// fetch an array of rows
```

Only three lines instead of six!