```php
<?php
$conn = mysqli_connect("localhost", "root", "", "class_db");
$id = isset($_POST['id']) ? intval($_POST['id']) : 0;
$sql = "SELECT * FROM students WHERE id = $id";
$res = mysqli_query($conn, $sql) or die(mysqli_error($conn));
$r = mysqli_fetch_assoc($res);
echo isset($r['first_name']) ? $r['first_name'] : "Student not found";
?>
```

```php
<?php
$conn = mysqli_connect("localhost","root","","class_db");
$fname = isset($_POST['fname']) ? $_POST['fname'] : '';
$fname= mysqli_real_escape_string($conn, $fname);
$sql = "SELECT * FROM students WHERE first_name = '$fname'";
$res = mysqli_query($conn, $sql) or die(mysqli_error($conn));
?>
```

```php
<?php
$conn = mysqli_connect("localhost", "root", "", "class_db");
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$age = $_GET['age'] ?? null;
if (!ctype_digit($age)) {
   die("Invalid age parameter.");
```

```
}
$stmt = $conn->prepare("SELECT * FROM students WHERE age = ?");
$stmt->bind_param("i", $age);
$stmt->execute();
$res = $stmt->get_result();

while ($row = $res->fetch_assoc()) {
   echo $row['name'] . " - " . $row['age'] . "<br>";
}
$stmt->close();
$conn->close();
?>
```

EXPLANATION:

Scenario 1

The script takes an id from the POST data and converts it into an integer. It then looks up the student with that ID in the database. While using intval() helps block some injection attacks, it isn't completely secure. A better approach would be to use prepared statements, which keep the SQL code separate from user input.

Scenario 2

The script grabs fname from the POST data and escapes any special characters using mysqli_real_escape_string(). It then searches the database for students with that first name. Escaping characters helps reduce the risk of SQL injection, but using prepared statements would be a safer and more reliable method.

Scenario 3

The script gets age from the GET data and makes sure it only contains digits. It then uses a prepared statement to find students with that age. By binding the age parameter, the input can't interfere with the SQL, making this approach fully secure.