```php
foreach ($result as $row) {
    echo "<tr>";
    echo "<td>" . ($row['title']) . "</td>";
    echo "<td>" . ($row['count']) . "</td>";
    echo "</tr>";
}
```

```php
foreach ($result as $row) {
    echo "<tr>";
    echo "<td>" . htmlspecialchars($row['title']) . "</td>";
    echo "<td>" . htmlspecialchars($row['count']) . "</td>";
    echo "</tr>";
}
```

Основные моменты для защиты от XSS атак:

1. **Экранирование пользовательского ввода:** Всегда экранируйте специальные символы перед отображением данных, введенных пользователем.
2. **Использование библиотек и фреймворков:** Множество современных фреймворков и библиотек уже включают встроенные механизмы защиты от XSS. Например, в Django, Angular, React и других.
3. **Валидация и фильтрация ввода:** Ограничивайте и проверяйте данные, вводимые пользователями, чтобы убедиться, что они соответствуют ожиданиям (например, только алфавитные символы в имени).

```php
try {
    $stmt = $db->prepare("SELECT * FROM p_languages;");
    $stmt->execute();
    $languages = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    print($e->getMessage());
    exit();
}
```

```php
try {
    $stmt = $db->prepare("SELECT * FROM p_languages;");
    $stmt->execute();
    $languages = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    print("Ошибка подключения к базе данных");
    exit();
}
```

1. Мы убрали раскрытие внутренней информации об ошибке.
2. Обрабатываем исключения так, чтобы они не выдавали подробные технические детали пользователю.

```
$stmt = $db->prepare("SELECT * FROM user
where user=".$_POST['login']);
$stmt -> execute();
```

```
$stmt = $db->prepare("SELECT * FROM user
where user=?");
$stmt -> execute([$_POST['login']]);
```

1. Подготавливает SQL-запрос, который выбирает все записи из таблицы **user**, где значение в колонке **user** совпадает с указанным.
2. Выполняет этот запрос, используя значение, переданное через POST-запрос с ключом **'login'**.

```
else if(($_SERVER['REQUEST_METHOD'] == 'POST')){
```

```
else if(($_SERVER['REQUEST_METHOD'] == 'POST')){
  if (!empty($_POST['csrf_token']) && hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) {
```

Во втором примере используется CSRF токен, который генерируется и сохраняется в сессии при загрузке формы. Этот токен затем включается в скрытое поле формы. Когда форма отправляется, сервер проверяет, соответствует ли токен значению, хранящемуся в сессии. Если они не совпадают, запрос отклоняется. Это предотвращает атаки CSRF, так как злоумышленник не может предугадать или получить действительный CSRF токен пользователя

```
include('form.php');
```

Так как значение подключаемого файла является константным, то код не поддевержен Include и Upload атакам