DOSSIER M4

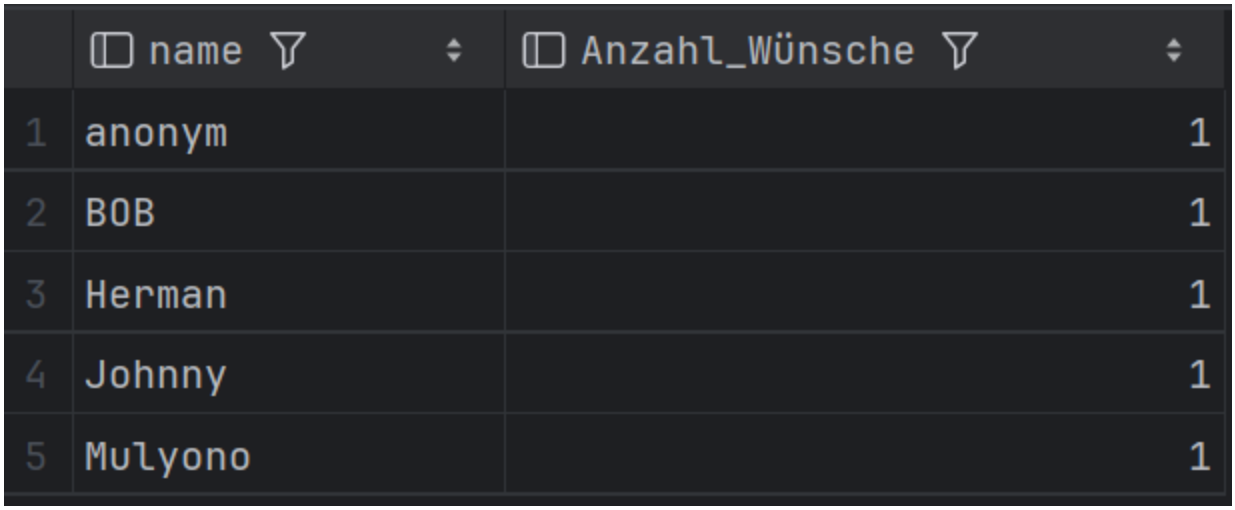
Adriano Ferane, Gunawan, 3659313
Michael Xhristiano, Espranata, 3658655

Table

Aufgaben	Geschätzte Zeit	Ursprüngliche Zeit	Gründe für die Abweichung
Aufgabe 1	30 minuten	30minuten	
Aufgabe 2	30 minuten	15 minuten	
Aufgabe 4	1 stunden	1 stunden	
Aufgabe 6	30 minuten	25 minuten	
Aufgabe 7	2 stunden	4 stunden	
Aufgabe 8	3 stunden	5 stunden	

SELECT * FROM Wunschgericht ORDER BY creation_date DESC LIMIT 5;

	id	name	description	creation_date	creator_id
1	1	chicken teriyaki	chicken teriyaki mit reis	2024-12-05	1
2	2	Currywurst	mit pommes bitte	2024-12-05	2
3	3	burger	Burger mit pommes	2024-12-05	3
4	4	pasta	aglio olio	2024-12-05	4
5	5	nasi uduk	campur ayam	2024-12-05	5



	name	Anzahl_Wünsche
1	anonym	1
2	BOB	1
3	Herman	1
4	Johnny	1
5	Mulyono	1

```
SELECT Ersteller.name, COUNT(Wunschgericht.id) AS
Anzahl_Wünsche
FROM Ersteller
LEFT JOIN Wunschgericht ON Ersteller.id = Wunschgericht.creator_id
GROUP BY Ersteller.name;
```

AUFGABE 2

1. CSRF-Schutz

- **Früherer Code**:
  - Enthielt keine Mechanismen, um CSRF-Angriffe (Cross-Site Request Forgery) zu verhindern.
- **Verbesserter Code**:

Ein CSRF-Token wurde hinzugefügt. Dies erstellt ein eindeutiges Token, speichert es in der Session und fügt es als verstecktes Feld in das Formular ein. Bei der Formularübermittlung wird das Token überprüft:

```
if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
```

```
<input type="hidden" name="csrf_token" value="<?php echo
$_SESSION['csrf_token']; ?>">

if (!hash_equals($_SESSION['csrf_token'], $_POST['csrf_token']))
{
    die("Ungültiges CSRF-Token!");
}
```

- **Warum?**
  - Dies stellt sicher, dass nur Anfragen, die von Ihrer Anwendung stammen, verarbeitet werden. Ohne diesen Schutz könnten Angreifer schädliche Links oder Formulare erstellen, die Aktionen im Namen eines Benutzers ausführen.

---

2. XSS-Abwehr

- **Ihr Code**:

Nutzte Benutzereingaben direkt (`$_POST['...']`), ohne sie zu bereinigen oder zu escapen:

```
$creator_name = !empty($_POST['creator_name']) ?
$_POST['creator_name'] : 'anonym';
$dish_name = $_POST['name'];
$description = $_POST['description'];
```

- **Verbesserter Code**:

Alle Benutzereingaben wurden mit `htmlspecialchars()` escaped:

```
$creator_name = !empty($_POST['creator_name']) ?
htmlspecialchars($_POST['creator_name'], ENT_QUOTES, 'UTF-8') :
'anonym';
```

```
$dish_name = htmlspecialchars($_POST['name'], ENT_QUOTES,
'UTF-8');
$description = htmlspecialchars($_POST['description'],
ENT_QUOTES, 'UTF-8');
```

- **Warum?**
  - Ohne Escaping könnten Angreifer schädliches HTML/JavaScript in die Datenbank einfügen, was zu Cross-Site Scripting (XSS) führen würde. Escaping stellt sicher, dass Eingaben wie `<script>alert('XSS')</script>` als reiner Text behandelt werden und nicht ausgeführt werden.

---

3. Verhinderung von SQL-Injection

- **Ihr Code**:

Nutzt bereits vorbereitete Statements für `INSERT`-Abfragen, was gut ist. Allerdings wurden Benutzereingaben vor dem Binden nicht validiert oder bereinigt:
php
Copy code
```
$stmt = $conn->prepare("INSERT INTO Ersteller (name, email)
VALUES (?, ?)");
$stmt->bind_param("ss", $creator_name, $creator_email);
```

- **Verbesserter Code**:

Beibehaltung von vorbereiteten Statements, zusätzlich aber Eingabebereinigung und Validierung, insbesondere für E-Mails:

```
$creator_email = filter_var($_POST['creator_email'],
FILTER_SANITIZE_EMAIL);


if (!filter_var($creator_email, FILTER_VALIDATE_EMAIL)) {
    die("Ungültige E-Mail-Adresse!");
}
```

    -
- **Warum?**
    - Vorbereitete Statements schützen bereits vor SQL-Injection. Die zusätzliche Validierung und Bereinigung von Eingaben bietet jedoch eine weitere Sicherheitsebene:
        - `FILTER_SANITIZE_EMAIL` entfernt ungültige Zeichen.
        - `FILTER_VALIDATE_EMAIL` prüft, ob das E-Mail-Format korrekt ist.

---

4. Bereinigung von Eingaben

- **Ihr Code**:
    - Validierte oder bereinigte Benutzereingaben wie `description` oder `creator_name` nicht.
- **Verbesserter Code**:

Alle Eingaben wurden mit `htmlspecialchars()` bereinigt, um spezielle Zeichen wie `<`, `>` und `&` zu neutralisieren:

```
$dish_name = htmlspecialchars($_POST['name'], ENT_QUOTES,
'UTF-8');
```

    -
- **Warum?**
    - Verhindert potenzielles XSS und stellt sicher, dass nur saubere Daten in der Datenbank gespeichert werden.

---

5. Validierung der E-Mail-Eingabe

- **Ihr Code**:
    - Validierte die vom Benutzer eingegebene E-Mail-Adresse nicht.
- **Verbesserter Code**:

Hinzufügen der E-Mail-Validierung mit:

```
if (!filter_var($creator_email, FILTER_VALIDATE_EMAIL)) {
    die("Ungültige E-Mail-Adresse!");
}
```

    -
- **Warum?**
    - Verhindert ungültige oder schädliche Daten im E-Mail-Feld und stellt sicher, dass nur ein gültiges Format wie `user@example.com` akzeptiert wird.

AUFGABE 4

1)Unique Combination of Gericht and Kategorie:
sql

```
ALTER TABLE gericht_hat_kategorie
ADD CONSTRAINT unique_gericht_kategorie UNIQUE (gericht_id,
kategorie_id);
```

2) Index for Gericht Name:
sql

```
CREATE INDEX idx_gericht_name ON gericht (name);
```

3) Cascade Deletes for Gericht:
sql

```
ALTER TABLE gericht_hat_kategorie
ADD CONSTRAINT fk_gericht_kategorie
FOREIGN KEY (gericht_id) REFERENCES gericht (id)
ON DELETE CASCADE;

ALTER TABLE allergen_hat_gericht
ADD CONSTRAINT fk_allergen_gericht
FOREIGN KEY (gericht_id) REFERENCES gericht (id)
ON DELETE CASCADE;
```

4) Deletion Restriction for Kategorie:
sql

```
ALTER TABLE gericht_hat_kategorie
ADD CONSTRAINT fk_gericht_kategorie
FOREIGN KEY (kategorie_id) REFERENCES kategorie (id)
ON DELETE RESTRICT;

ALTER TABLE kategorie
ADD CONSTRAINT fk_kategorie_parent
FOREIGN KEY (parent_id) REFERENCES kategorie (id)
ON DELETE RESTRICT;
```

5) Cascading Updates for Allergens:
sql

```
ALTER TABLE allergen_hat_gericht
ADD CONSTRAINT fk_allergen_code
FOREIGN KEY (allergen_code) REFERENCES allergen (code)
ON UPDATE CASCADE;
```

6) Primary Key for `gericht_hat_kategorie`:
sql

```
ALTER TABLE gericht_hat_kategorie
ADD PRIMARY KEY (gericht_id, kategorie_id);
```