

**Critical Threats in Modern Systems**  
**Step 3 - Implementation of the Database**  
**Application System (MySQL)**

Jair Ramirez

CS 4332 Introduction to Database Systems  
Texas State University

April 13<sup>th</sup>, 2025

## Abstract

In this step, we bring our conceptual schema to life in MySQL by mapping the ER model into fully normalized tables with all primary keys, foreign keys, and constraints enforced. We then convert our ten user-centric queries into SQL, execute each one, and document the results with screenshots. This implementation validates the practical functionality of our design and lays the groundwork for the next phase of application integration.

## Section A - E/R Diagram

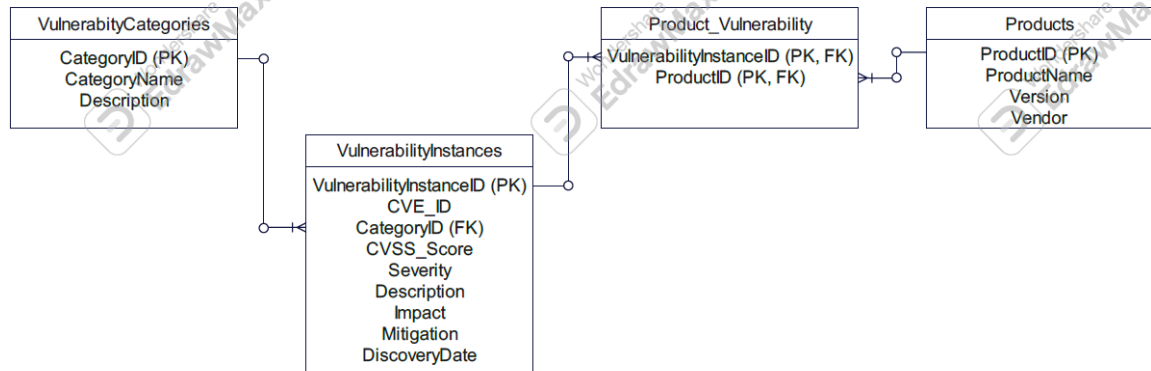


Figure 1: E/R Diagram

### VulnerabilityCategories

- **CategoryID** (PK)
- CategoryName
- Description
- **Relationship:** One-to-many with *VulnerabilityInstances* (each category can have many vulnerabilities).

### VulnerabilityInstances

- **VulnerabilityInstanceID** (PK)
- CVE\_ID
- **CategoryID** (FK referencing *VulnerabilityCategories.CategoryID*)
- CVSS\_Score
- Severity
- Description
- Impact
- Mitigation
- DiscoveryDate
- **Relationship:** Each instance is linked to one category; participates in a many-to-many relationship with *Products* through the linking table *Product\_Vulnerability*.

### Products

- **ProductID** (PK)
- ProductName
- Version
- Vendor

- **Relationship:** Many-to-many with *VulnerabilityInstances* (decomposed via the *Product\_Vulnerability* table).

#### **Product\_Vulnerability** (Linking Table)

- **VulnerabilityInstanceID** (PK, FK referencing *VulnerabilityInstances.VulnerabilityInstanceID*)
- **ProductID** (PK, FK referencing *Products.ProductID*)
- **Relationship:** Resolves the many-to-many relationship between *VulnerabilityInstances* and *Products* by using a composite primary key.

## **Section B - First Implementation**

This section implements our English queries from Step 3 in the chosen DBMS, MySQL. We show DDL for table creation, followed by each SQL statement and its execution output.

1. **Query 1:** Retrieve a list of all vulnerability instances, showing each CVE ID, the corresponding vulnerability type, severity, and discovery date, sorted by discovery date in descending order (most recent first).

```
SELECT
    VI.CVE_ID,
    VC.CategoryName AS VulnerabilityType,
    VI.Severity,
    VI.DiscoveryDate
FROM
    VulnerabilityInstances VI
JOIN
    VulnerabilityCategories VC
    ON VI.CategoryID = VC.CategoryID
ORDER BY
    VI.DiscoveryDate DESC;
```

This query provides a comprehensive overview of all vulnerabilities recorded in the system. By retrieving the CVE IDs, vulnerability types, severity levels, and discovery dates, it allows security analysts and decision makers to quickly identify the most recent vulnerabilities. This information is crucial for prioritizing remediation efforts and ensuring that the latest threats are addressed promptly.

MySQL Workbench

Project 6 - Warning - not supp... x

File Edit View Query Database Server Tools Scripting Help

Project6 x

Limit to 1000 rows

```

182 SELECT
183     VI.CVE_ID,
184     VC.CategoryName AS VulnerabilityType,
185     VI.Severity,
186     VI.DiscoveryDate
187 FROM
188     VulnerabilityInstances VI
189 JOIN
190     VulnerabilityCategories VC
191     ON VI.CategoryID = VC.CategoryID
192 ORDER BY
193     VI.DiscoveryDate DESC;
194

```

Result Grid

	CVE_ID	VulnerabilityType	Severity	DiscoveryDate
▶	CVE-2025-2951	SQL Injection	Medium	2025-03-30
	CVE-2025-2074	SQL Injection	Medium	2025-03-28
	CVE-2025-2699	Cross-Site Scripting	Medium	2025-03-24
	CVE-2024-13903	Buffer Overflow	High	2025-03-21
	CVE-2025-2088	SQL Injection	Medium	2025-03-13
	CVE-2024-12035	Remote Code Execution	High	2025-03-07
	CVE-2025-2061	Cross-Site Scripting	Medium	2025-03-06
	CVE-2025-1899	Buffer Overflow	High	2025-03-03
	CVE-2024-50310	Information Disclosure	High	2024-11-12

Result 6 x

Output

Action Output

#	Time	Action	Message
1	17:56:19	SELECT VI.CVE_ID, VC.CategoryName AS VulnerabilityType, VI.Severity, VI.DiscoveryDate FR...	9 row(s) returned

Figure 2: Execution Output for Query 1 - Vulnerability Overview Dashboard

- Query 2:** Display a list of products including the vendor and version, along with the number of vulnerabilities affecting each product, sorted by the vulnerability count in descending order.

```

SELECT
    P.ProductName,
    P.Vendor,
    P.Version,
    COUNT(PV.VulnerabilityInstanceID) AS VulnerabilityCount
FROM
    Products P
JOIN
    Product_Vulnerability PV ON P.ProductID = PV.ProductID
GROUP BY
    P.ProductID, P.ProductName, P.Vendor, P.Version
ORDER BY
    VulnerabilityCount DESC;

```

This query aggregates data from the Products and Product\_Vulnerability tables. By joining these tables, the query counts the number of vulnerabilities associated with each product. It then displays the product's name, vendor, and version along with this count, sorted from highest to lowest vulnerability count. This is useful for identifying which products are at greatest risk, allowing security teams to prioritize remediation efforts.

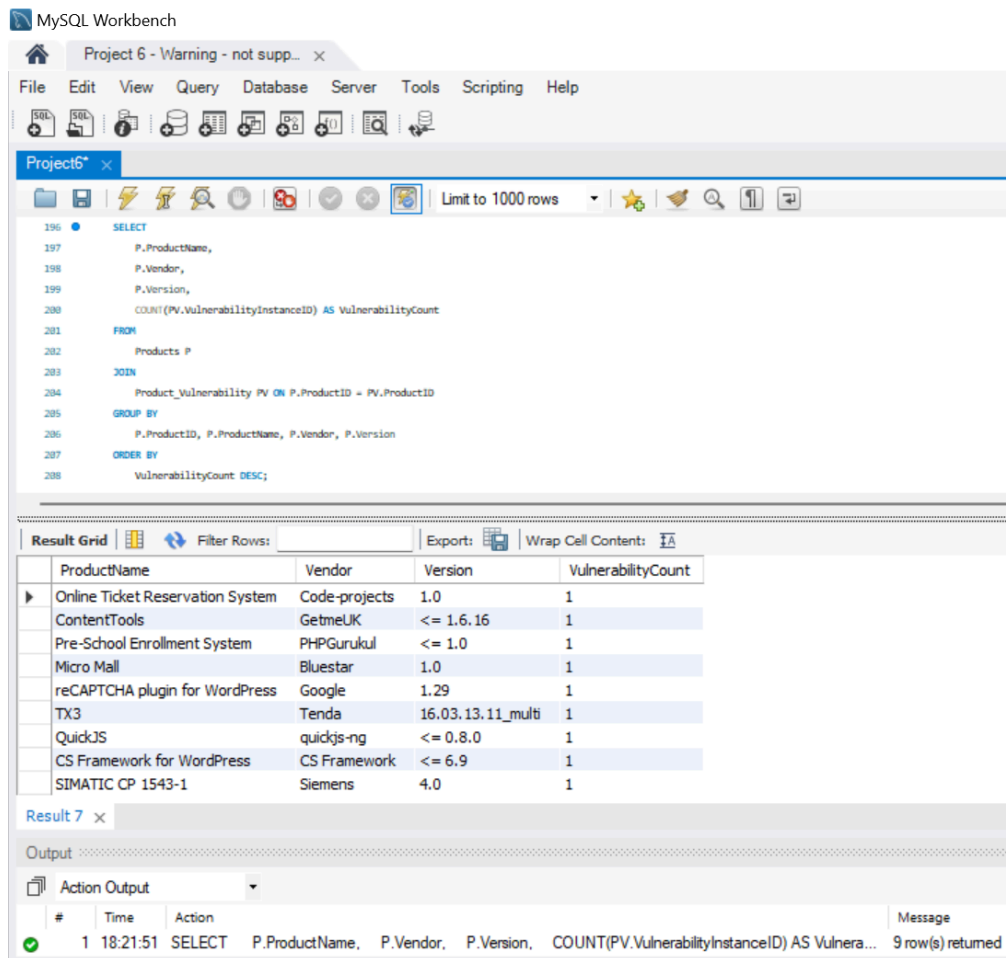


Figure 3: Execution Output for Query 2 – Vulnerability Count by Product

- Query 3:** Retrieve details for all vulnerabilities of type 'SQL Injection'—including the CVE ID, CVSS score, severity, impact, and mitigation details—and list the names of all products affected by each of these vulnerabilities.

```

SELECT
    VI.CVE_ID,
    VI.CVSS_Score,
    VI.Severity,
    VI.Impact,
    VI.Mitigation,
    GROUP_CONCAT(P.ProductName SEPARATOR ', ') AS AffectedProducts
FROM
    VulnerabilityInstances VI
JOIN
    VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
JOIN
    Product_Vulnerability PV ON VI.VulnerabilityInstanceID = PV.VulnerabilityInstanceID
JOIN
    Products P ON PV.ProductID = P.ProductID
WHERE
    VC.CategoryName = 'SQL Injection'
GROUP BY
    VI.VulnerabilityInstanceID;

```

This query narrows down the results to only include vulnerabilities classified as 'SQL Injection.' For each case, it pulls essential details like the CVE ID, CVSS score, severity level, potential impact, and recommended mitigation steps. It also compiles the names of all affected products using the GROUP\_CONCAT function, creating a unified view. This output gives security teams

a clearer understanding of the risks tied to SQL Injection flaws and highlights exactly which products need focused remediation.

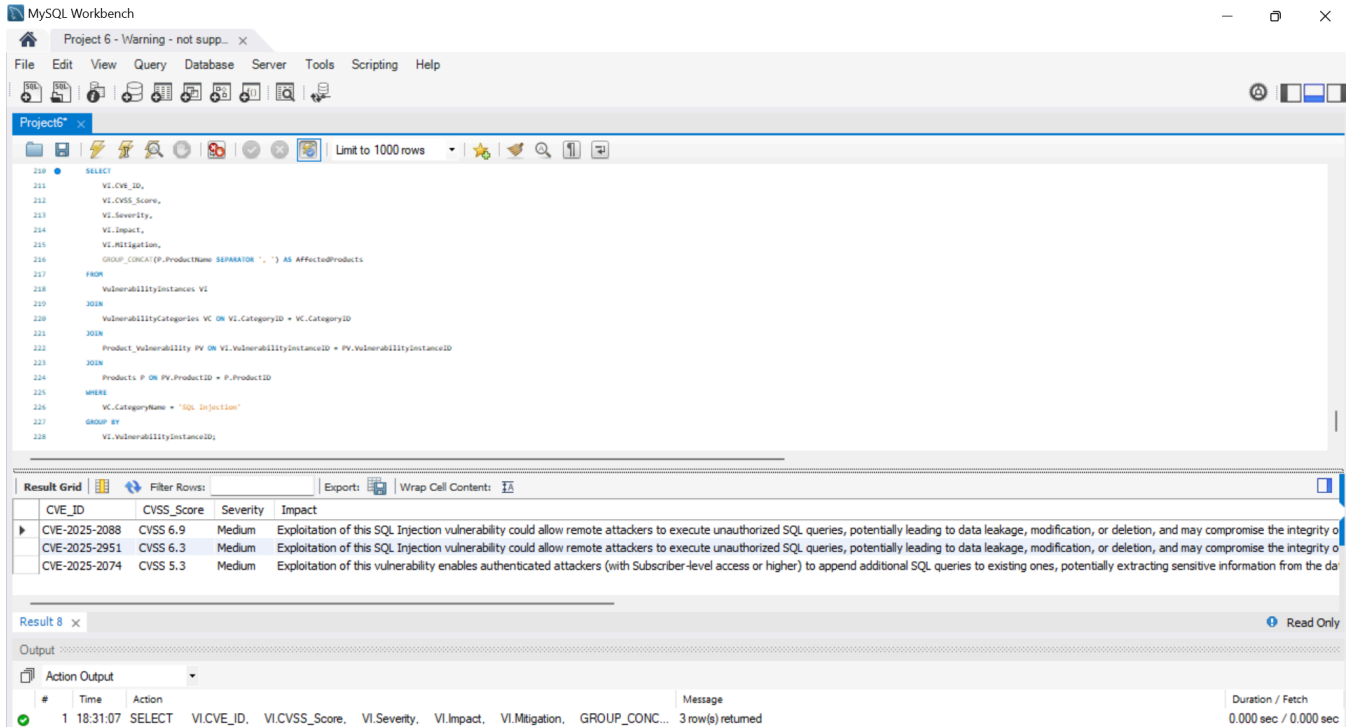


Figure 4: Execution Output for Query 3 – SQL Injection Vulnerabilities and Affected Products

**Mitigation**

Implement robust input validation and sanitization for the parameters (fullname, emailid, mobileNumber). Use parameterized queries or prepared statements to prevent SQL injection attacks.

Implement strict input validation and sanitization for the "Search" parameter in the /api/data.php file. Use parameterized queries or prepared statements to prevent SQL injection attacks, and restrict database privileges to the minimum necessary.

Implement robust input sanitization and parameterized queries to securely handle the "sSearch" parameter. Ensure that any user-supplied data is properly escaped and validated before use in SQL queries.

Figure 5: Execution Output for Query 3 – SQL Injection Vulnerabilities and Affected Products (2)

AffectedProducts

Pre-School Enrollment System

Micro Mail

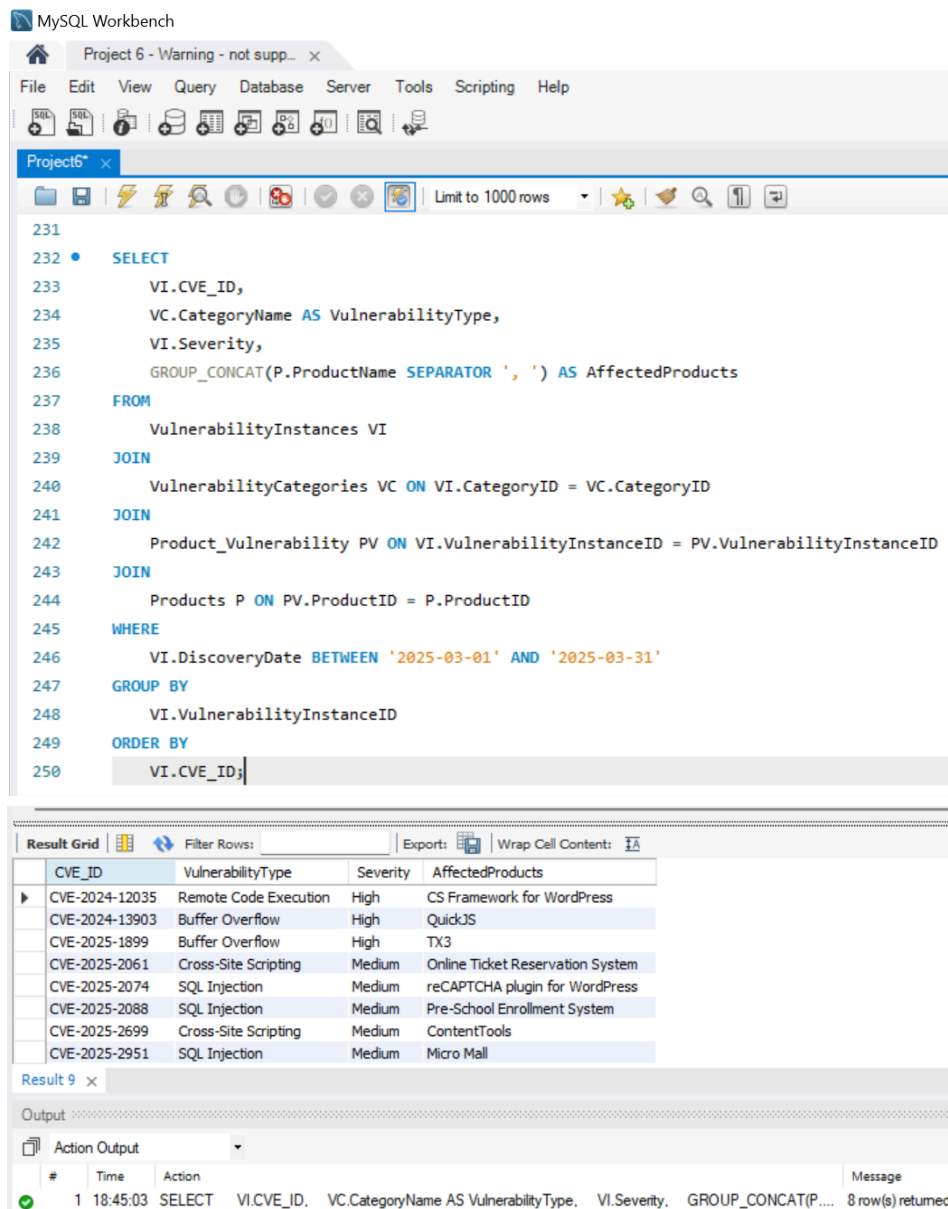
reCAPTCHA plugin for WordPress

Figure 6: Execution Output for Query 3 – SQL Injection Vulnerabilities and Affected Products (3)

- Query 4:** Retrieve all vulnerabilities that were discovered in March 2025, displaying their CVE IDs, vulnerability types, severity levels, and the names of the products affected.

```
SELECT
    VI.CVE_ID,
    VC.CategoryName AS VulnerabilityType,
    VI.Severity,
    GROUP_CONCAT(P.ProductName SEPARATOR ', ') AS AffectedProducts
FROM
    VulnerabilityInstances VI
JOIN
    VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
JOIN
    Product_Vulnerability PV ON VI.VulnerabilityInstanceID = PV.VulnerabilityInstanceID
JOIN
    Products P ON PV.ProductID = P.ProductID
WHERE
    VI.DiscoveryDate BETWEEN '2025-03-01' AND '2025-03-31'
GROUP BY
    VI.VulnerabilityInstanceID
ORDER BY
    VI.CVE_ID;
```

This query is useful for focusing on a specific time period, allowing security teams to analyze vulnerabilities discovered during that month and quickly identify the risk associated with the affected products.



The screenshot shows the MySQL Workbench interface. The top toolbar includes icons for File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The main editor displays a SQL query (lines 231-250) that selects vulnerability information for March 2025. The query includes columns for CVE ID, vulnerability type, severity, and affected products, joined across multiple tables (VulnerabilityInstances, VulnerabilityCategories, Product\_Vulnerability, and Products). The WHERE clause filters for discovery dates between '2025-03-01' and '2025-03-31'. The results are grouped by VulnerabilityInstanceID and ordered by CVE ID.

Below the query editor, the 'Result Grid' shows 8 rows of data. The columns are CVE\_ID, VulnerabilityType, Severity, and AffectedProducts. The data includes CVEs from 2024 and 2025, with various severity levels and affected products like WordPress, QuickJS, and TX3.

CVE_ID	VulnerabilityType	Severity	AffectedProducts
CVE-2024-12035	Remote Code Execution	High	CS Framework for WordPress
CVE-2024-13903	Buffer Overflow	High	QuickJS
CVE-2025-1899	Buffer Overflow	High	TX3
CVE-2025-2061	Cross-Site Scripting	Medium	Online Ticket Reservation System
CVE-2025-2074	SQL Injection	Medium	reCAPTCHA plugin for WordPress
CVE-2025-2088	SQL Injection	Medium	Pre-School Enrollment System
CVE-2025-2699	Cross-Site Scripting	Medium	ContentTools
CVE-2025-2951	SQL Injection	Medium	Micro Mall

The 'Output' section at the bottom shows the execution details: 1 row(s) returned, 18:45:03, and a message indicating 8 row(s) returned.

Figure 7: Execution Output for Query 4 – Vulnerabilities Discovered in March 2025

- Query 5:** Retrieve all vulnerabilities where the mitigation strategy includes the phrase 'input validation'. For each matching record, display the CVE ID, vulnerability type, detailed mitigation steps, and the names of all products affected by these vulnerabilities.

```
SELECT
    VI.CVE_ID,
    VC.CategoryName AS VulnerabilityType,
    VI.Mitigation,
    GROUP_CONCAT(P.ProductName SEPARATOR ', ') AS AffectedProducts
FROM
    VulnerabilityInstances VI
JOIN
    VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
JOIN
    Product_Vulnerability PV ON VI.VulnerabilityInstanceID = PV.VulnerabilityInstanceID
JOIN
    Products P ON PV.ProductID = P.ProductID
WHERE
    VI.Mitigation LIKE '%input validation%'
```

```

Products P ON PV.ProductID = P.ProductID
WHERE
VI.Mitigation LIKE '%input validation%'
GROUP BY
VI.VulnerabilityInstanceID;

```

This query filters the vulnerability records to include only those where the mitigation strategy mentions 'input validation'. This is useful for quickly identifying vulnerabilities that rely on input validation as part of their remediation strategy, helping security teams focus on a common mitigation approach.

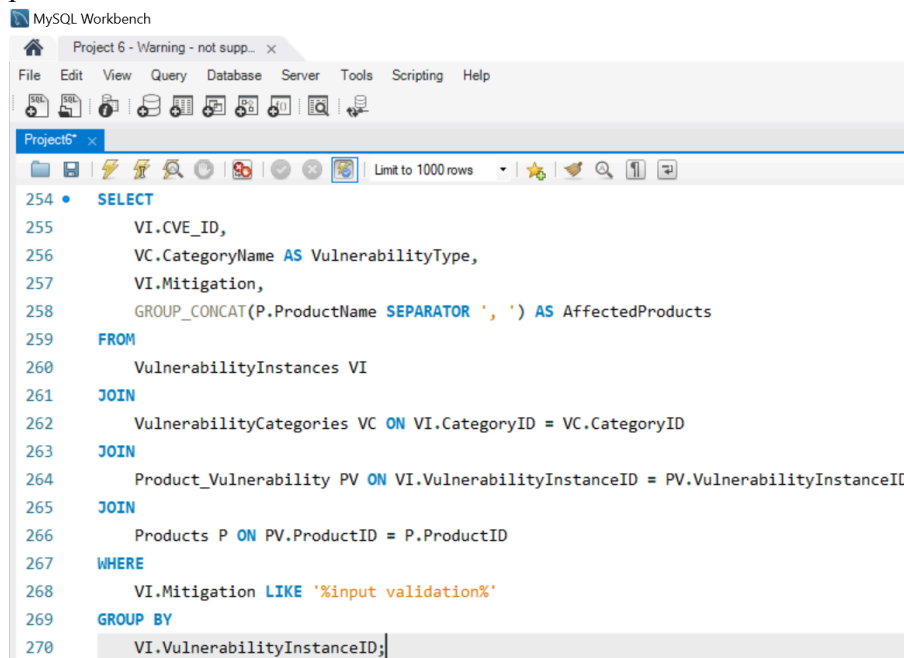


Figure 8: Execution Output for Query 5 – Vulnerabilities with 'Input Validation' in Mitigation

CVE_ID	VulnerabilityType	Mitigation
CVE-2025-2061	Cross-Site Scripting	Implement proper input validation and output encoding on the affected file (/passenger.php). Sanitize all user-supplied data to ensure special characters are handled safely, and apply secure coding practices to pre
CVE-2025-2699	Cross-Site Scripting	Apply strict input validation and output encoding for the onload attribute in the Image Handler component. Additionally, review and patch the affected component in GetmeUK ContentTools.
CVE-2025-2088	SQL Injection	Implement robust input validation and sanitization for the parameters (fullname, emailid, mobileNumber). Use parameterized queries or prepared statements to prevent SQL injection attacks.
CVE-2025-2951	SQL Injection	Implement strict input validation and sanitization for the "Search" parameter in the /api/data.php file. Use parameterized queries or prepared statements to prevent SQL injection attacks, and restrict database privi
CVE-2025-1899	Buffer Overflow	Implement strict bounds checking and input validation on the "list" parameter in the /goform/setPptpUserList functionality. Ensure secure memory handling.
CVE-2024-13903	Buffer Overflow	Review and enforce secure coding practices, such as proper input validation and bounds checking, to prevent similar vulnerabilities.

Figure 8: Execution Output for Query 5 – Vulnerabilities with 'Input Validation' in Mitigation (2)

AffectedProducts
Online Ticket Reservation System
ContentTools
Pre-School Enrollment System
Micro Mall
TX3
QuickJS

Figure 8: Execution Output for Query 5 – Vulnerabilities with 'Input Validation' in Mitigation (3)

- Query 6:** Retrieve all products affected by 'Buffer Overflow' vulnerabilities. For each product, display the product name, vendor, and the count of 'Buffer Overflow' vulnerabilities associated with it, ordered by the vulnerability count in descending order.

```

SELECT

```

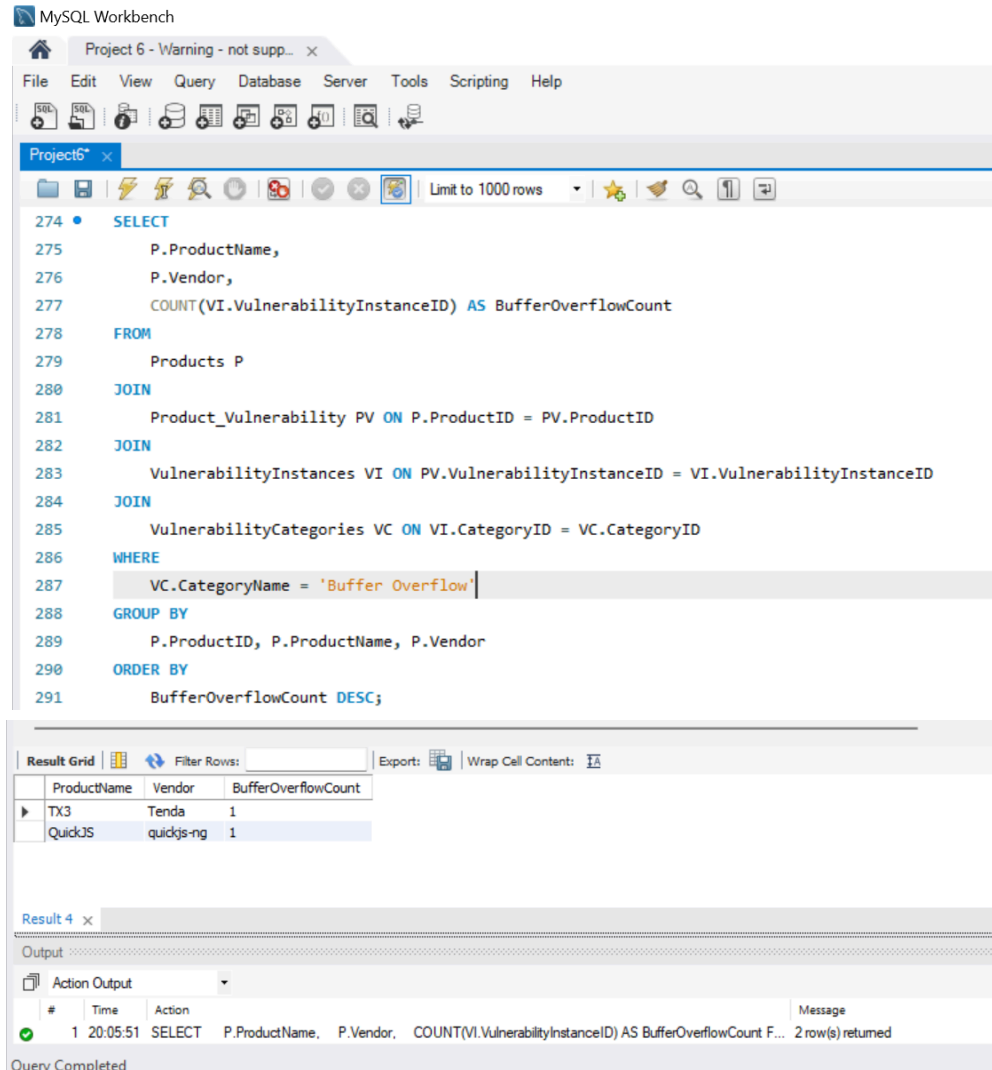


```

P.ProductName,
P.Vendor,
COUNT(VI.VulnerabilityInstanceID) AS BufferOverflowCount
FROM
Products P
JOIN
Product_Vulnerability PV ON P.ProductID = PV.ProductID
JOIN
VulnerabilityInstances VI ON PV.VulnerabilityInstanceID = VI.VulnerabilityInstanceID
JOIN
VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
WHERE
VC.CategoryName = 'Buffer Overflow'
GROUP BY
P.ProductID, P.ProductName, P.Vendor
ORDER BY
BufferOverflowCount DESC;

```

This query focuses on identifying products that are affected by 'Buffer Overflow' vulnerabilities. It joins the Products, Product\_Vulnerability, and VulnerabilityInstances tables along with the VulnerabilityCategories table to filter for vulnerabilities that are classified as 'Buffer Overflow'.



MySQL Workbench

Project 6 - Warning - not supp... x

File Edit View Query Database Server Tools Scripting Help

Project6\* x

Limit to 1000 rows

```

274 • SELECT
275     P.ProductName,
276     P.Vendor,
277     COUNT(VI.VulnerabilityInstanceID) AS BufferOverflowCount
278 FROM
279     Products P
280 JOIN
281     Product_Vulnerability PV ON P.ProductID = PV.ProductID
282 JOIN
283     VulnerabilityInstances VI ON PV.VulnerabilityInstanceID = VI.VulnerabilityInstanceID
284 JOIN
285     VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
286 WHERE
287     VC.CategoryName = 'Buffer Overflow'
288 GROUP BY
289     P.ProductID, P.ProductName, P.Vendor
290 ORDER BY
291     BufferOverflowCount DESC;

```

Result Grid

ProductID	ProductName	Vendor	BufferOverflowCount
TX3	TX3	Tenda	1
QuickJS	QuickJS	quickjs-ng	1

Result 4 x

Output

Action Output

#	Time	Action	Message
1	20:05:51	SELECT P.ProductName, P.Vendor, COUNT(VI.VulnerabilityInstanceID) AS BufferOverflowCount F...	2 row(s) returned

Query Completed

Figure 9: Execution Output for Query 6 – Products Affected by 'Buffer Overflow' Vulnerabilities

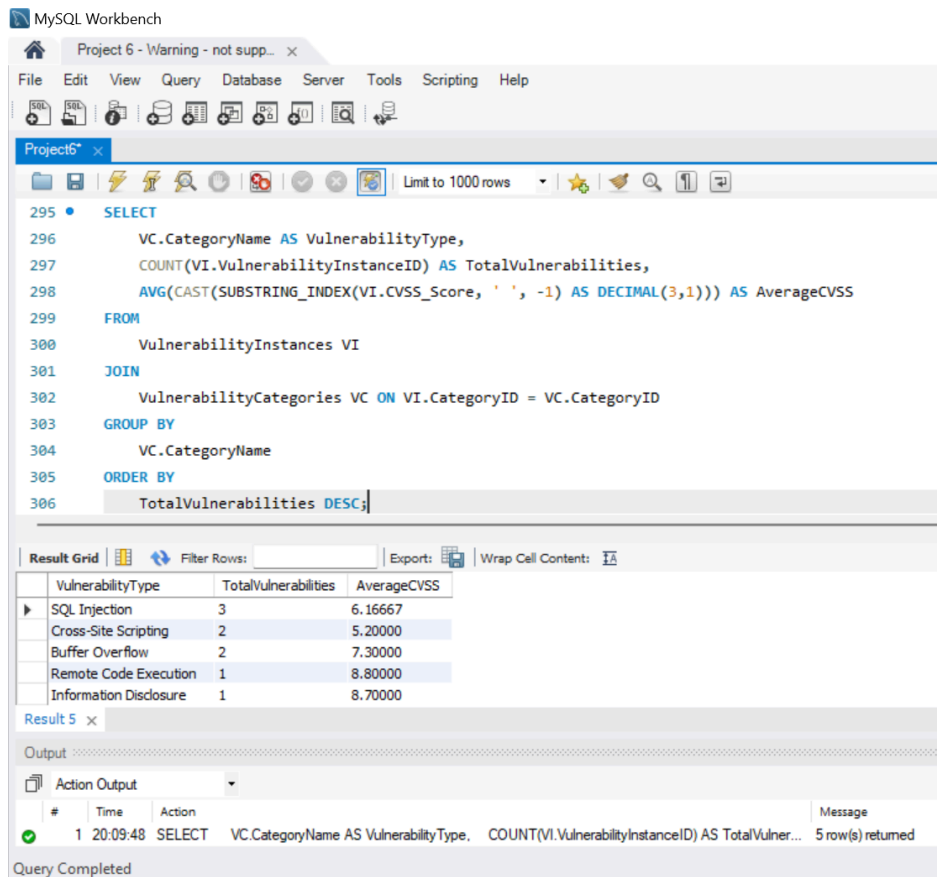
- Query 7:** Retrieve aggregated vulnerability data: for each vulnerability type, display the total number of vulnerabilities and the average CVSS score, sorted by the total count in descending order.

```

SELECT
    VC.CategoryName AS VulnerabilityType,
    COUNT(VI.VulnerabilityInstanceID) AS TotalVulnerabilities,
    AVG(CAST(SUBSTRING_INDEX(VI.CVSS_Score, ' ', -1) AS DECIMAL(3,1))) AS AverageCVSS
FROM
    VulnerabilityInstances VI
JOIN
    VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
GROUP BY
    VC.CategoryName
ORDER BY
    TotalVulnerabilities DESC;

```

This query aggregates vulnerability data by vulnerability type. For each type (e.g., 'SQL Injection', 'Buffer Overflow', etc.), it counts the total number of vulnerability instances. Calculates the average CVSS score. Sorting by the total count in descending order helps identify the vulnerability types with the highest number of occurrences, which can be useful for prioritization and risk assessment.



MySQL Workbench

Project 6 - Warning - not supp... x

File Edit View Query Database Server Tools Scripting Help

Project6 x

Limit to 1000 rows

```

295 • SELECT
296     VC.CategoryName AS VulnerabilityType,
297     COUNT(VI.VulnerabilityInstanceID) AS TotalVulnerabilities,
298     AVG(CAST(SUBSTRING_INDEX(VI.CVSS_Score, ' ', -1) AS DECIMAL(3,1))) AS AverageCVSS
299 FROM
300     VulnerabilityInstances VI
301 JOIN
302     VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
303 GROUP BY
304     VC.CategoryName
305 ORDER BY
306     TotalVulnerabilities DESC;

```

Result Grid

VulnerabilityType	TotalVulnerabilities	AverageCVSS
SQL Injection	3	6.16667
Cross-Site Scripting	2	5.20000
Buffer Overflow	2	7.30000
Remote Code Execution	1	8.80000
Information Disclosure	1	8.70000

Result 5 x

Output

Action Output

#	Time	Action	Message
1	20:09:48	SELECT VC.CategoryName AS VulnerabilityType, COUNT(VI.VulnerabilityInstanceID) AS TotalVulnerabilities, AVG(CAST(SUBSTRING_INDEX(VI.CVSS_Score, ' ', -1) AS DECIMAL(3,1))) AS AverageCVSS	5 row(s) returned

Query Completed

Figure 10: Execution Output for Query 7 – Aggregated Vulnerability Data

8. **Query 8:** Retrieve all vulnerabilities with a severity level of 'High' and display their CVE IDs, vulnerability types, and discovery dates, along with the names of the products affected by them. Sort the results by CVE ID.

```

SELECT
    VI.CVE_ID,
    VC.CategoryName AS VulnerabilityType,
    VI.DiscoveryDate,
    GROUP_CONCAT(P.ProductName SEPARATOR ', ') AS AffectedProducts
FROM
    VulnerabilityInstances VI
JOIN
    VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID


```

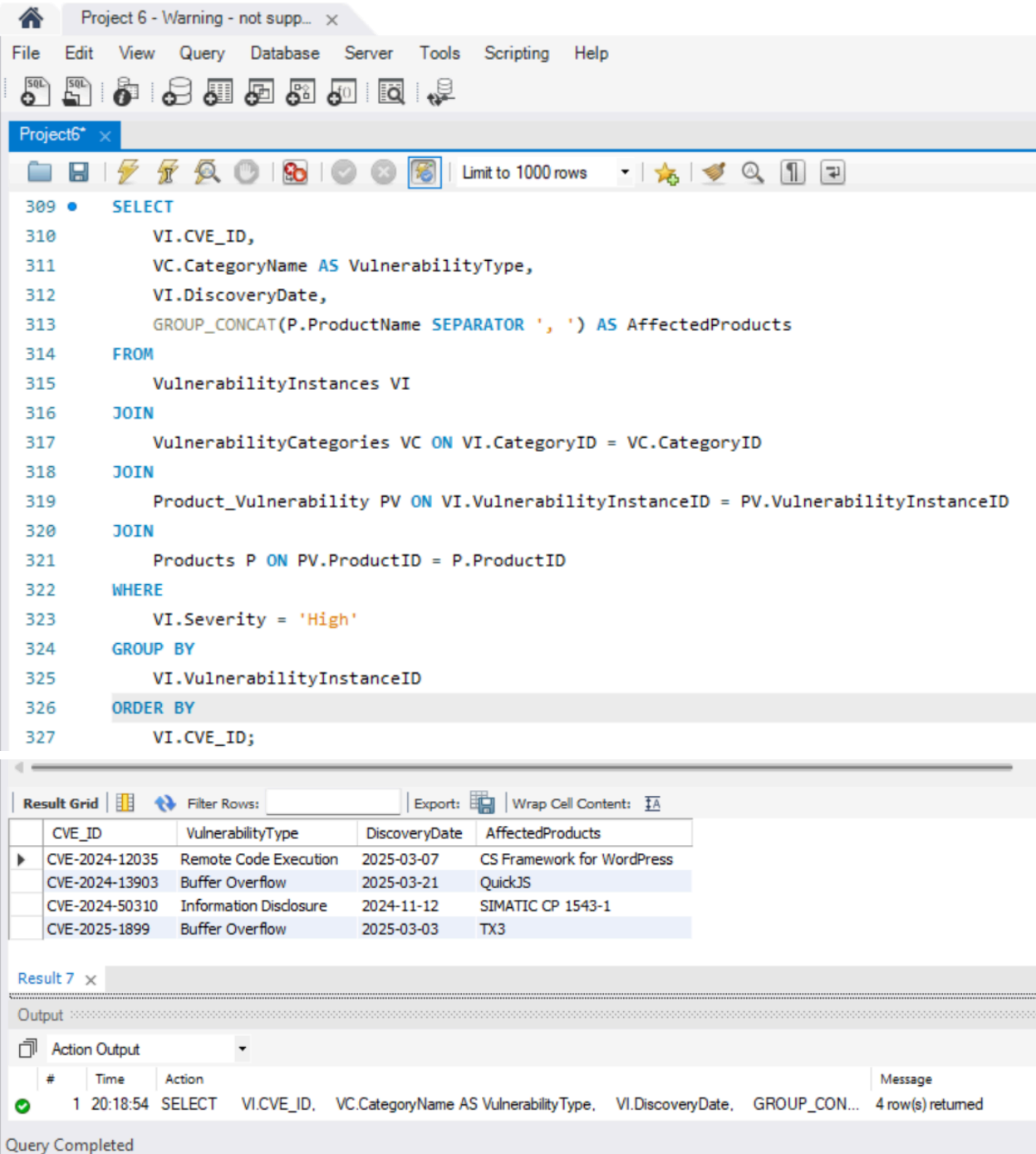
```

JOIN
    Product_Vulnerability PV ON VI.VulnerabilityInstanceID = PV.VulnerabilityInstanceID
JOIN
    Products P ON PV.ProductID = P.ProductID
WHERE
    VI.Severity = 'High'
GROUP BY
    VI.VulnerabilityInstanceID
ORDER BY
    VI.CVE_ID;

```

This query focuses on vulnerabilities with a high severity level. For each vulnerability we collect CVE\_ID, VulnerabilityType, DiscoveryDate, AffectedProducts Sorting by CVE\_ID provides an organized, easily navigable list of high-severity vulnerabilities, enabling security teams to quickly identify and address the most critical issues.

 MySQL Workbench



The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. Below the menu is a toolbar with various icons. The main editor area displays a SQL query (lines 309-327) that selects CVE\_ID, VulnerabilityType, DiscoveryDate, and AffectedProducts from the VulnerabilityInstances, VulnerabilityCategories, Product\_Vulnerability, and Products tables, filtered by Severity = 'High' and ordered by CVE\_ID.

Below the query editor, the "Result Grid" tab is active, showing the results of the query. The results are displayed in a table with 5 columns: CVE\_ID, VulnerabilityType, DiscoveryDate, and AffectedProducts. The table contains 4 rows of data.

CVE_ID	VulnerabilityType	DiscoveryDate	AffectedProducts
CVE-2024-12035	Remote Code Execution	2025-03-07	CS Framework for WordPress
CVE-2024-13903	Buffer Overflow	2025-03-21	QuickJS
CVE-2024-50310	Information Disclosure	2024-11-12	SIMATIC CP 1543-1
CVE-2025-1899	Buffer Overflow	2025-03-03	TX3

Below the result grid, the "Output" tab is active, showing the "Action Output" section. It displays a message: "4 row(s) returned".

At the bottom of the interface, a status bar indicates "Query Completed".

Figure 11: Execution Output for Query 8 – High Severity Vulnerabilities

9. **Query 9:** Retrieve a list of all vulnerabilities along with their CVE IDs, vulnerability types, detailed descriptions, impact, and corresponding mitigation strategies, sorted by the CVE ID.

```
SELECT
    VI.CVE_ID,
    VC.CategoryName AS VulnerabilityType,
    VI.Description,
    VI.Impact,
    VI.Mitigation
FROM
    VulnerabilityInstances VI
JOIN
    VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
ORDER BY
    VI.CVE_ID;
```

This query is particularly useful for providing a detailed overview of all vulnerabilities, which helps security analysts, auditors, and management in understanding the nature of each threat and planning appropriate mitigation strategies.

MySQL Workbench

Project 6 - Warning - not supp... x

File Edit View Query Database Server Tools Scripting Help

Project6 x

Limit to 1000 rows

```
331 SELECT
332     VI.CVE_ID,
333     VC.CategoryName AS VulnerabilityType,
334     VI.Description,
335     VI.Impact,
336     VI.Mitigation
337 FROM
338     VulnerabilityInstances VI
339 JOIN
340     VulnerabilityCategories VC ON VI.CategoryID = VC.CategoryID
341 ORDER BY
342     VI.CVE_ID;
```

Result Grid

CVE_ID	VulnerabilityType	Description
CVE-2024-12035	Remote Code Execution	The CS Framework plugin for WordPress is vulnerable to arbitrary file deletion due to insufficient file path validation in the <code>cs_widget_file_delete()</code> function.
CVE-2024-13903	Buffer Overflow	Affected by this vulnerability is the function <code>JS_GetRuntime</code> of the file <code>quickjs.c</code> of the component <code>qjs</code> . The manipulation leads to stack-based buffer overflow.
CVE-2024-50310	Information Disclosure	Affected devices do not properly handle authorization.
CVE-2025-1899	Buffer Overflow	Affected by this vulnerability is an unknown functionality of the file <code>/goform/setPtpUserList</code> . The manipulation of the argument list leads to buffer overflow.

Result 8 x

Output

Action Output

#	Time	Action	Message
1	20:28:34	SELECT VI.CVE_ID, VC.CategoryName AS VulnerabilityType, VI.Description, VI.Impact, VI.Mitigation	9 row(s) returned

Query Completed

Figure 12: Execution Output for Query 9 – Detailed Vulnerability Information

Impact
Exploitation of this vulnerability enables authenticated attackers (with Subscriber-level access or higher) to delete arbitrary files on the server. By targeting critical files such as wp-config.php, attackers can achieve remote code execution and fully ...
Exploitation of this vulnerability allows remote attackers to trigger a stack-based buffer overflow within the JS_GetRuntime function, potentially leading to memory corruption, application crashes, and in the worst case, arbitrary code execution.
Exploitation of this vulnerability could allow unauthenticated remote attackers to bypass authorization mechanisms, gaining access to the filesystem. This may result in unauthorized data exposure, modification, or further system compromise.
Exploitation of this vulnerability may allow remote attackers to trigger a buffer overflow by manipulating the "list" parameter. This could result in memory corruption, system crashes, or potentially enable remote code execution, thereby compromising the.

Figure 12: Execution Output for Query 9 – Detailed Vulnerability Information (2)

Mitigation
Enforce strict file path validation and restrict the file deletion functionality to only the intended and authorized directories. Additionally, ensure that file deletion operations are limited to users with the appropriate privileges.
Review and enforce secure coding practices, such as proper input validation and bounds checking, to prevent similar vulnerabilities.
Enforce network segmentation and restrict remote access to critical systems to mitigate the risk of unauthorized access.
Implement strict bounds checking and input validation on the "list" parameter in the /goform/setPtpUserList functionality. Ensure secure memory handling.

Figure 12: Execution Output for Query 9 – Detailed Vulnerability Information (3)

10. **Query 10:** Retrieve a distinct list of mitigation strategies used for vulnerabilities, along with the count of vulnerabilities that use each mitigation strategy, sorted by the count in descending order.

```
SELECT
    Mitigation,
    COUNT(*) AS VulnerabilityCount
FROM
    VulnerabilityInstances
GROUP BY
    Mitigation
ORDER BY
    VulnerabilityCount DESC;
```

This query retrieves a distinct list of mitigation strategies that are employed to address vulnerabilities across the system and counts the number of occurrences for each strategy. By grouping the vulnerabilities by their mitigation field, the query shows how frequently each mitigation approach is used. This insight can help in understanding prevalent remedial practices and may assist in further optimizing security measures by highlighting the most relied-upon strategies.

MySQL Workbench

Project 6 - Warning - not supp... x

File Edit View Query Database Server Tools Scripting Help

Project6 x

Limit to 1000 rows

```
346 SELECT
347     Mitigation,
348     COUNT(*) AS VulnerabilityCount
349 FROM
350     VulnerabilityInstances
351 GROUP BY
352     Mitigation
353 ORDER BY
354     VulnerabilityCount DESC;
```

Result Grid

Mitigation	VulnerabilityCount
Implement proper input validation and output encoding on the affected file (/passenger.php). Sanitize all user-supplied data to ensure special characters are handled safely, and apply secure coding practices to prevent script inject...	1
Apply strict input validation and output encoding for the onload attribute in the Image Handler component. Additionally, review and patch the affected component in GetmeUK ContentTools.	1
Implement robust input validation and sanitization for the parameters (fullname, emailid, mobileNumber). Use parameterized queries or prepared statements to prevent SQL injection attacks.	1
Implement strict input validation and sanitization for the "Search" parameter in the /api/data.php file. Use parameterized queries or prepared statements to prevent SQL injection attacks, and restrict database privileges to the mini...	1
Implement robust input sanitization and parameterized queries to securely handle the "sSearch" parameter. Ensure that any user-supplied data is properly escaped and validated before use in SQL queries.	1
Implement strict bounds checking and input validation on the "list" parameter in the /goform/setPtpUserList functionality. Ensure secure memory handling.	1
Review and enforce secure coding practices, such as proper input validation and bounds checking, to prevent similar vulnerabilities.	1
Enforce strict file path validation and restrict the file deletion functionality to only the intended and authorized directories. Additionally, ensure that file deletion operations are limited to users with the appropriate privileges.	1
Enforce network segmentation and restrict remote access to critical systems to mitigate the risk of unauthorized access.	1

Result 9 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	20:28:34	SELECT	VI.CVE_ID, VC.CategoryName AS VulnerabilityType, VI.Description, VI.Impact, VI.M...	9 row(s) returned 0.000 sec / 0.000 sec

Query Completed

Figure 13: Execution Output for Query 10 – Mitigation Strategies and Vulnerability Counts