PROJECT

SQL INJECTION ATTACK On Today's Technologies Using MYSQL Database Queries

Prepared By:

Dhanshree Gabhane

Guided By:

Zakir Hussain

TABLE OF CONTENT:

Introduction

SQL Injection (SQLi) is a type of cyberattack where an attacker inserts malicious SQL code into a web application's database.

This can lead to:

1. Data Theft: Extracting sensitive information like usernames, passwords, and credit card details.
2. Data Modification: Altering or deleting data in the database.
3. Unauthorized Access: Gaining higher access levels or executing commands on the server.

SQLi happens when a web application doesn't properly check or clean user inputs, letting attackers insert harmful SQL code.

According to the Open Web Application Security Project (OWASP), SQL Injection has consistently ranked as one of the top vulnerabilities in web applications. Even though more advanced security techniques have emerged, many applications still fail to implement basic protection mechanisms. Hence, understanding and preventing SQLi attacks is crucial for anyone working in web development or cybersecurity.

Common SQLi Attack Vectors:

1. **User Input Forms:**

   Forms that accept user inputs, such as login forms, search boxes, or registration forms, can be vulnerable to SQL Injection if the inputs are not properly sanitized.

2. **URL Parameters:**

   Web applications often use URL parameters to pass data between pages or queries. These parameters can be manipulated to inject malicious SQL code if the application does not properly validate or sanitize them.

3. **Cookies:**

   Cookies are used to store user-specific information, such as session identifiers or preferences. If these cookies are used in SQL queries without proper validation, they can be a vector for SQL Injection attacks.

4. **HTTP Headers:**

   Attackers can exploit HTTP headers, such as User-Agent, Referer, or custom headers, to inject SQL code if the application processes these headers without proper validation.

5. **Error Messages:**

   Detailed error messages returned by the application can provide attackers with clues about the database structure and vulnerabilities.

Types of SQL Injection Attacks:

SQL Injection (SQLi) attacks are a common method used by attackers to exploit vulnerabilities in web applications by injecting malicious SQL queries.

1. Classic SQL Injection (In-Band SQLi)

Classic SQL Injection, also known as In-Band SQL Injection, involves directly injecting SQL code into a query. This type of attack allows the attacker to manipulate the SQL query executed by the database. The attacker typically provides input that alters the query's structure to retrieve or modify data.

- Example: SELECT * FROM users
  WHERE username = 'admin' -- ' AND password = 'password';

2. Blind SQL Injection

Blind SQL Injection occurs when an attacker cannot see the direct results of their injected SQL queries. Instead, the attacker relies on indirect feedback from the application, such as changes in the application's behavior or response times, to infer information about the database.

- Example: SELECT * FROM users
  WHERE id = 1 AND 1=1; SELECT * FROM users
  WHERE id = 1 AND 1=2;

3. Time-Based SQL Injection

Time-Based SQL Injection is a subtype of Blind SQL Injection where the attacker uses time delays to infer information about the database. By injecting SQL code that causes the database to pause or sleep for a specified period, the attacker can deduce information based on the response times.

- Example: SELECT * FROM users
  WHERE IF (username='admin', SLEEP (5), 0);

4. Out-of-Band SQL Injection

Out-of-Band SQL Injection involves sending data to an external server controlled by the attacker. This type of attack is used when in-band or blind SQL Injection techniques are not feasible. It relies on the database's capability to initiate outbound connections.

- Example: The attacker may use a DNS-based approach to send sensitive data to a remote server.

Preventing SQL Injection:

To effectively safeguard against SQL Injection attacks, it's essential to implement best practices that minimize the risk of these vulnerabilities. Here are four key strategies for preventing SQL Injection:

1. Use Prepared Statements

Prepared Statements, also known as Parameterized Queries, are a fundamental defense against SQL Injection. By separating SQL code from user data, prepared statements ensure that user input is treated as data rather than executable code.

Working: In prepared statements, SQL queries are defined with placeholders for parameters. The database engine precompiles the query structure, and user inputs are bound to these placeholders, ensuring that inputs cannot alter the query's logic.

Instead of:

SELECT * FROM users WHERE username = 'user_input' AND password = 'user_password';

Use: SELECT * FROM users WHERE username = ? AND password = ?;

2. Parameterize Queries

Parameterizing queries is a practice closely related to using prepared statements. It involves using parameters to include user input in SQL queries, preventing input from being executed as SQL code. This approach is crucial for preventing SQL Injection attacks.

Working: Parameters in queries are placeholders that are replaced with user input during query execution. This ensures that user data cannot interfere with the SQL command's structure.

Example: SELECT * FROM orders WHERE order_id = :order_id;

The :order_id parameter is replaced with the actual value safely.

3. Validate User Input

Input validation is a proactive measure to ensure that user inputs conform to expected formats and constraints. Proper validation can prevent malicious data from being processed by SQL queries.

Working: Implement validation rules that check the type, format, length, and range of user inputs. Disallow unexpected characters and patterns that may indicate malicious intent.

Example: For a form input expecting a numeric ID, validate that the input contains only digits:

4. Limit Database Privileges

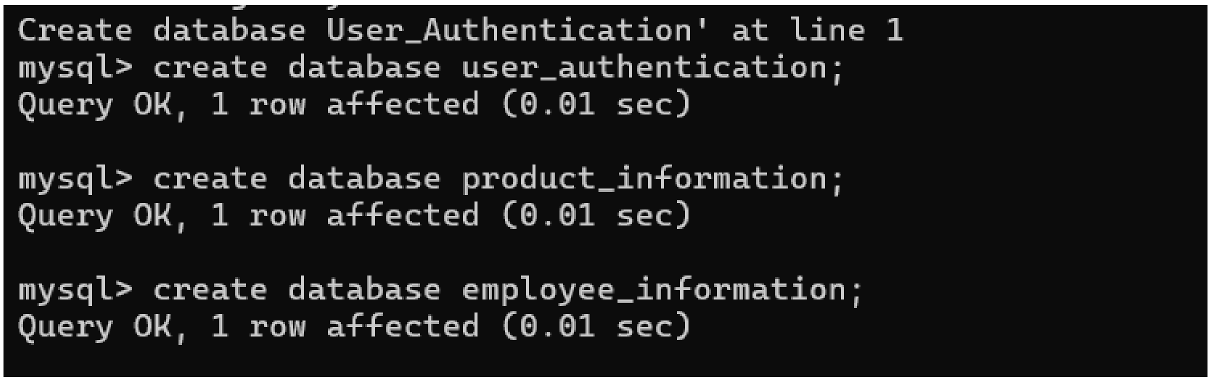
Limiting database privileges reduces the potential impact of SQL Injection attacks by ensuring that the database user account used by the application has only the necessary permissions.

Working: Assign minimal permissions to the database user account used by the application. Avoid using accounts with broad privileges, such as administrative roles, for routine operations.

Uses Of SQL Injection :

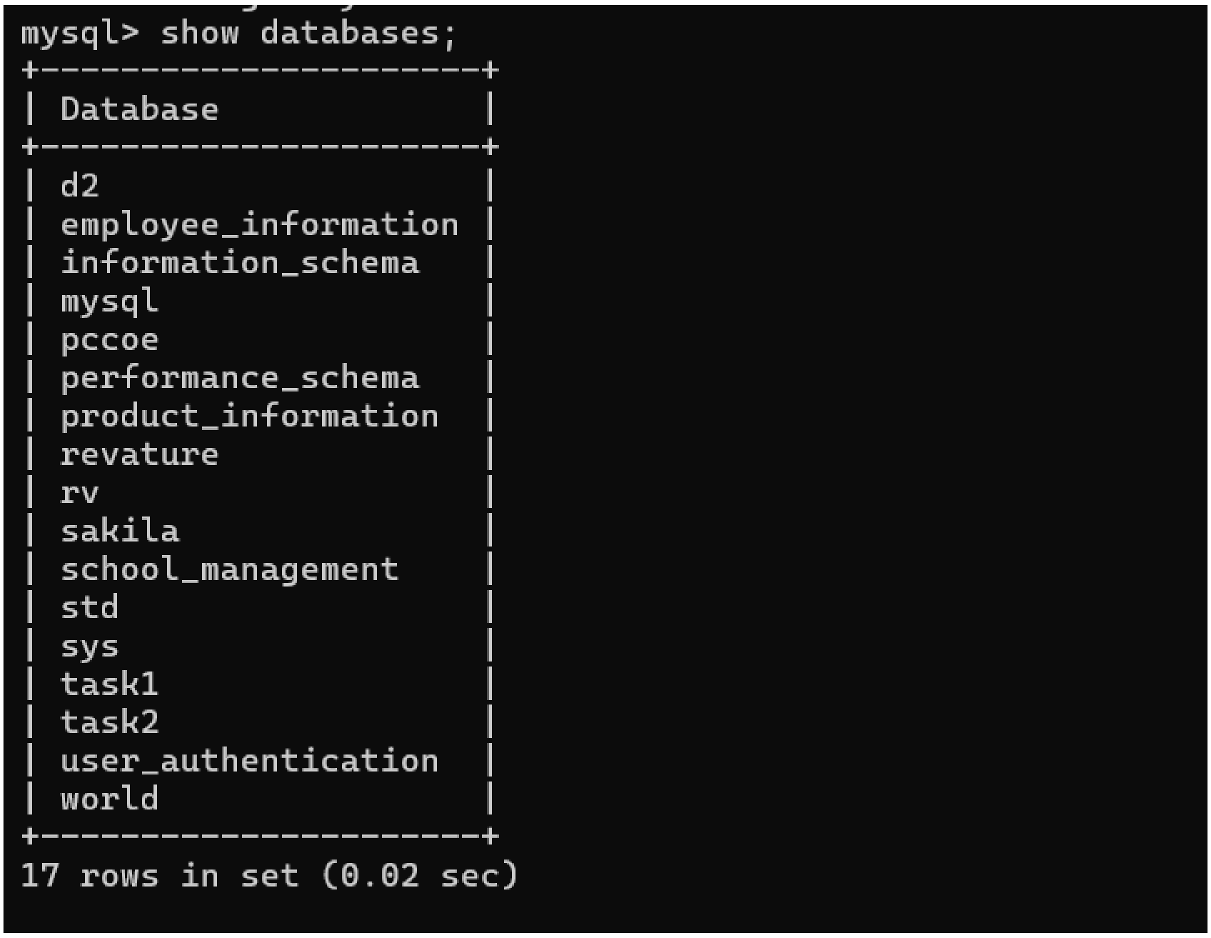
- **Data Extraction:** Retrieve sensitive or confidential data from the database.
- **Data Modification:** Alter or delete existing data in the database.
- **Authentication Bypass:** Gain unauthorized access by bypassing login controls.
- **Privilege Escalation:** Elevate user privileges to access restricted functions or data.
- **Data Exfiltration:** Extract and transfer data out of the database to an external location.
- **System Commands Execution:** Execute arbitrary system commands on the server.
- **Denial of Service (DoS):** Disrupt database operations to make the service unavailable.
- **Backdoor Creation:** Create hidden access points for future unauthorized entry.
- **Information Disclosure:** Reveal internal information about the database structure or application.
- **Corporate Espionage:** Steal proprietary or competitive information for malicious purposes.

Databases Structure And Setup-



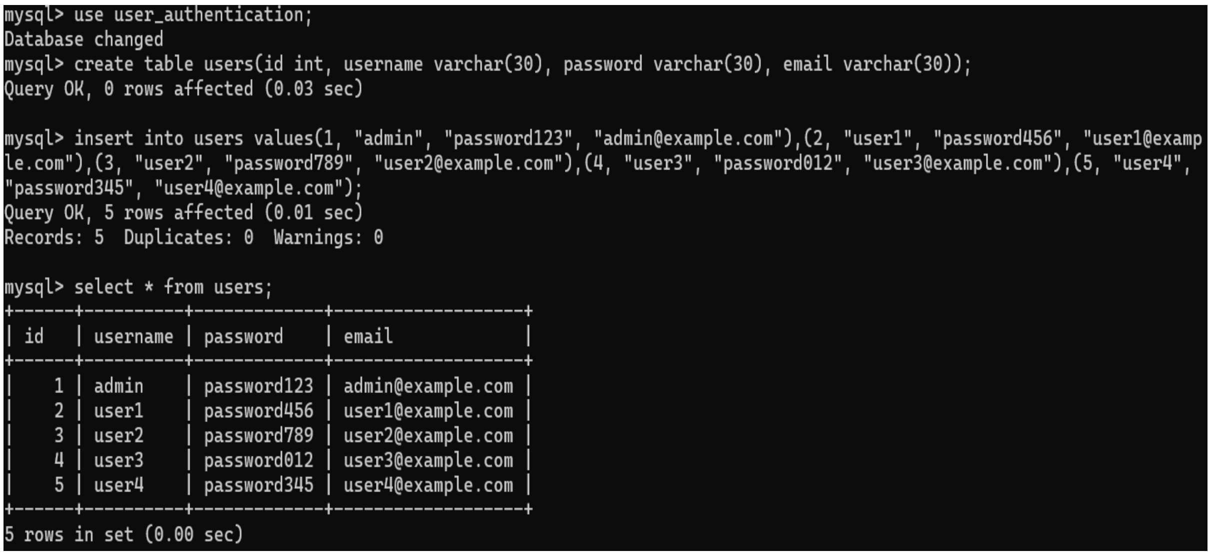
```
Create database User_Authentication' at line 1
mysql> create database user_authentication;
Query OK, 1 row affected (0.01 sec)

mysql> create database product_information;
Query OK, 1 row affected (0.01 sec)

mysql> create database employee_information;
Query OK, 1 row affected (0.01 sec)
```

To Show Databases-



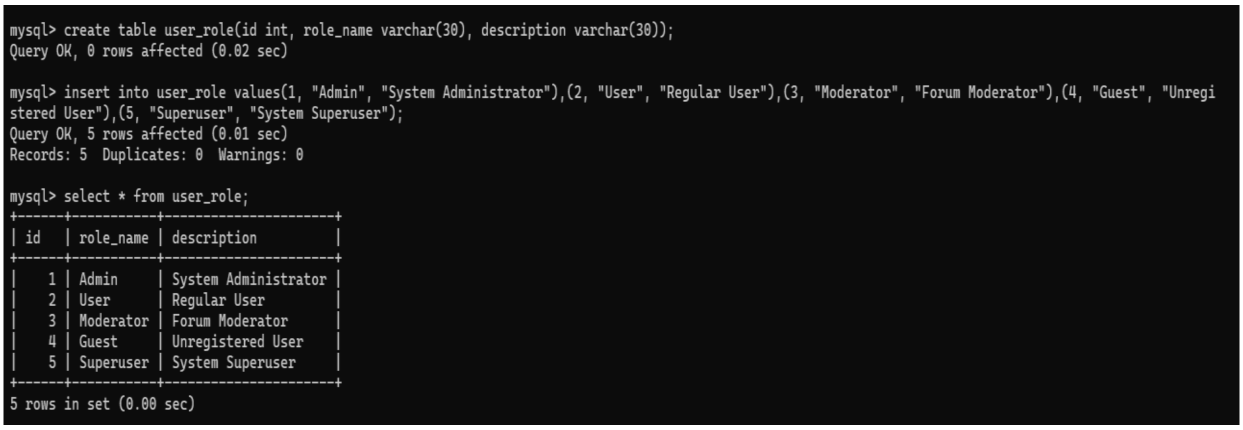
```
mysql> show databases;
+----------------------+
| Database             |
+----------------------+
| d2                   |
| employee_information |
| information_schema   |
| mysql                |
| pccoe                |
| performance_schema   |
| product_information  |
| revature             |
| rv                   |
| sakila               |
| school_management    |
| std                  |
| sys                  |
| task1                |
| task2                |
| user_authentication  |
| world                |
+----------------------+
17 rows in set (0.02 sec)
```

Creating table user for user_authentication-



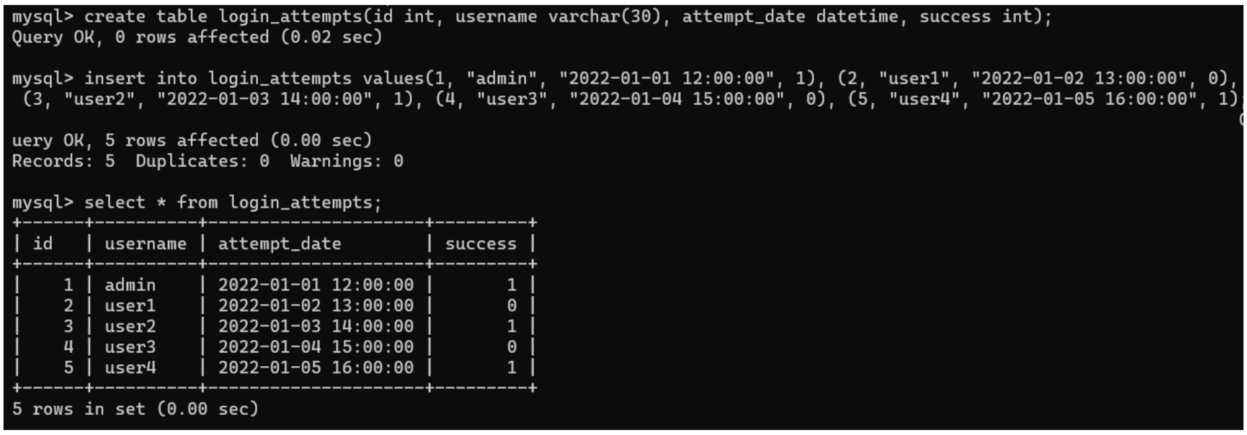
```
mysql> use user_authentication;
Database changed
mysql> create table users(id int, username varchar(30), password varchar(30), email varchar(30));
Query OK, 0 rows affected (0.03 sec)

mysql> insert into users values(1, "admin", "password123", "admin@example.com"),(2, "user1", "password456", "user1@examp
le.com"),(3, "user2", "password789", "user2@example.com"),(4, "user3", "password012", "user3@example.com"),(5, "user4",
"password345", "user4@example.com");
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from users;
+------+----------+-------------+-------------------+
| id   | username | password    | email             |
+------+----------+-------------+-------------------+
|    1 | admin    | password123 | admin@example.com |
|    2 | user1    | password456 | user1@example.com |
|    3 | user2    | password789 | user2@example.com |
|    4 | user3    | password012 | user3@example.com |
|    5 | user4    | password345 | user4@example.com |
+------+----------+-------------+-------------------+
5 rows in set (0.00 sec)
```

Creating table user_role for user_authentication-



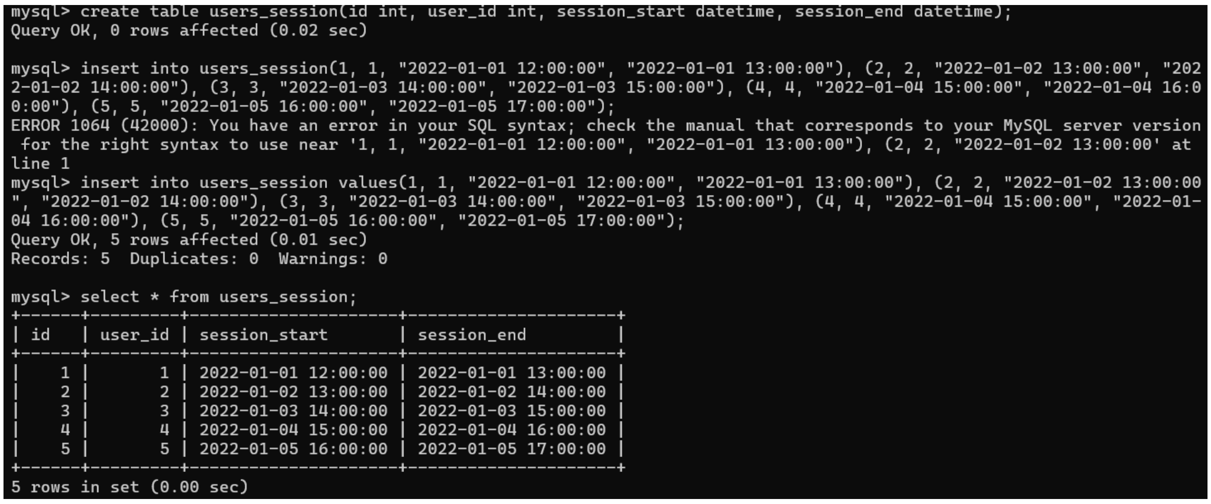
```
mysql> create table user_role(id int, role_name varchar(30), description varchar(30));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into user_role values(1, "Admin", "System Administrator"),(2, "User", "Regular User"),(3, "Moderator", "Forum Moderator"),(4, "Guest", "Unregi
stered User"),(5, "Superuser", "System Superuser");
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from user_role;
+------+-----------+----------------------+
| id   | role_name | description          |
+------+-----------+----------------------+
|    1 | Admin     | System Administrator |
|    2 | User      | Regular User         |
|    3 | Moderator | Forum Moderator      |
|    4 | Guest     | Unregistered User    |
|    5 | Superuser | System Superuser     |
+------+-----------+----------------------+
5 rows in set (0.00 sec)
```

Creating table login_attempts for user_authentication-



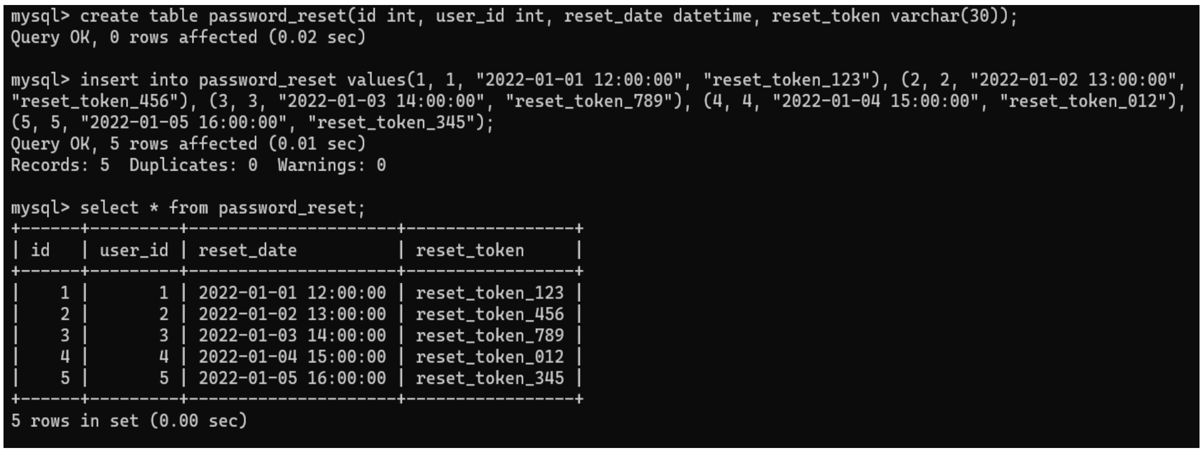
```
mysql> create table login_attempts(id int, username varchar(30), attempt_date datetime, success int);
Query OK, 0 rows affected (0.02 sec)

mysql> insert into login_attempts values(1, "admin", "2022-01-01 12:00:00", 1), (2, "user1", "2022-01-02 13:00:00", 0),
 (3, "user2", "2022-01-03 14:00:00", 1), (4, "user3", "2022-01-04 15:00:00", 0), (5, "user4", "2022-01-05 16:00:00", 1)

uery OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from login_attempts;
+------+----------+---------------------+---------+
| id   | username | attempt_date        | success |
+------+----------+---------------------+---------+
|    1 | admin    | 2022-01-01 12:00:00 |       1 |
|    2 | user1    | 2022-01-02 13:00:00 |       0 |
|    3 | user2    | 2022-01-03 14:00:00 |       1 |
|    4 | user3    | 2022-01-04 15:00:00 |       0 |
|    5 | user4    | 2022-01-05 16:00:00 |       1 |
+------+----------+---------------------+---------+
5 rows in set (0.00 sec)
```

Creating table Users_Session for user_authentication-



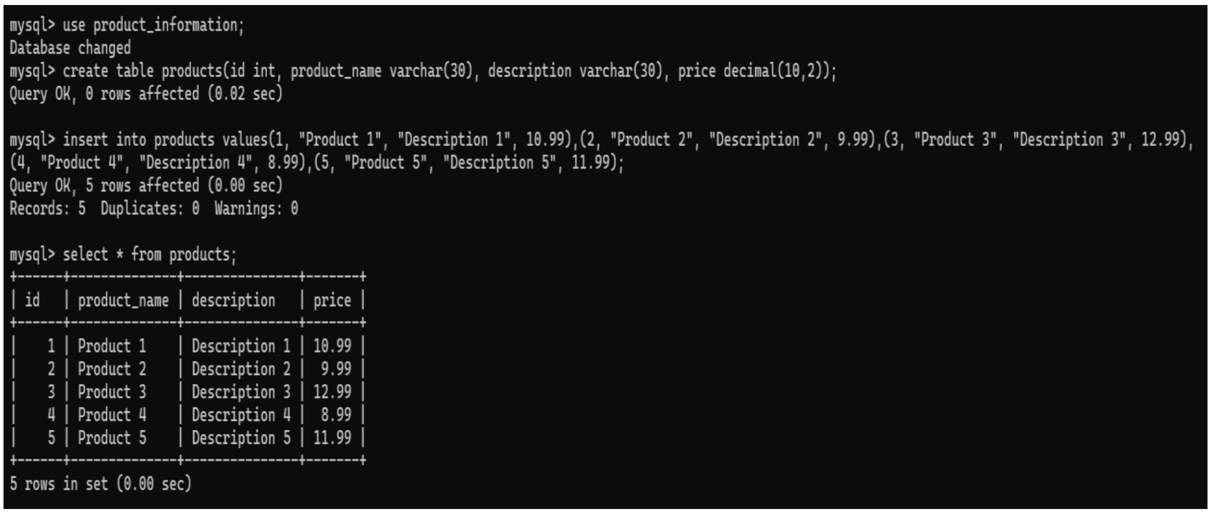
```
mysql> create table users_session(id int, user_id int, session_start datetime, session_end datetime);
Query OK, 0 rows affected (0.02 sec)

mysql> insert into users_session(1, 1, "2022-01-01 12:00:00", "2022-01-01 13:00:00"), (2, 2, "2022-01-02 13:00:00", "202
2-01-02 14:00:00"), (3, 3, "2022-01-03 14:00:00", "2022-01-03 15:00:00"), (4, 4, "2022-01-04 15:00:00", "2022-01-04 16:0
0:00"), (5, 5, "2022-01-05 16:00:00", "2022-01-05 17:00:00");
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
 for the right syntax to use near '1, 1, "2022-01-01 12:00:00", "2022-01-01 13:00:00"), (2, 2, "2022-01-02 13:00:00' at
line 1
mysql> insert into users_session values(1, 1, "2022-01-01 12:00:00", "2022-01-01 13:00:00"), (2, 2, "2022-01-02 13:00:00
", "2022-01-02 14:00:00"), (3, 3, "2022-01-03 14:00:00", "2022-01-03 15:00:00"), (4, 4, "2022-01-04 15:00:00", "2022-01-
04 16:00:00"), (5, 5, "2022-01-05 16:00:00", "2022-01-05 17:00:00");
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from users_session;
+------+---------+---------------------+---------------------+
| id   | user_id | session_start       | session_end         |
+------+---------+---------------------+---------------------+
|    1 |       1 | 2022-01-01 12:00:00 | 2022-01-01 13:00:00 |
|    2 |       2 | 2022-01-02 13:00:00 | 2022-01-02 14:00:00 |
|    3 |       3 | 2022-01-03 14:00:00 | 2022-01-03 15:00:00 |
|    4 |       4 | 2022-01-04 15:00:00 | 2022-01-04 16:00:00 |
|    5 |       5 | 2022-01-05 16:00:00 | 2022-01-05 17:00:00 |
+------+---------+---------------------+---------------------+
5 rows in set (0.00 sec)
```

Creating table Password Reset for user_authentication-



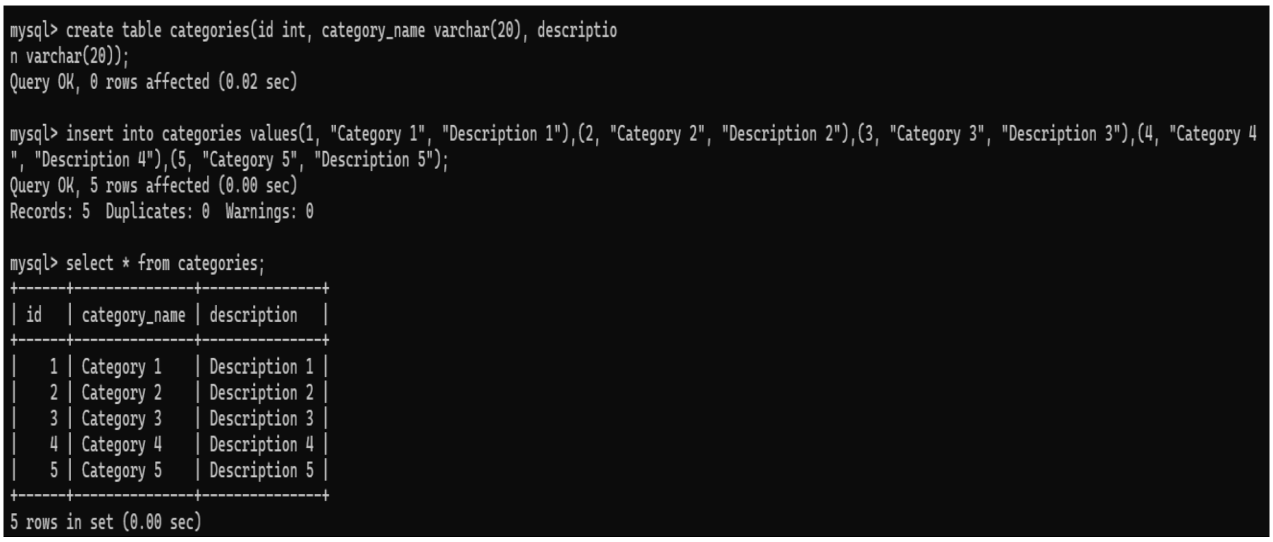
```
mysql> create table password_reset(id int, user_id int, reset_date datetime, reset_token varchar(30));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into password_reset values(1, 1, "2022-01-01 12:00:00", "reset_token_123"), (2, 2, "2022-01-02 13:00:00",
"reset_token_456"), (3, 3, "2022-01-03 14:00:00", "reset_token_789"), (4, 4, "2022-01-04 15:00:00", "reset_token_012"),
(5, 5, "2022-01-05 16:00:00", "reset_token_345");
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from password_reset;
+------+---------+---------------------+-----------------+
| id   | user_id | reset_date          | reset_token     |
+------+---------+---------------------+-----------------+
|    1 |       1 | 2022-01-01 12:00:00 | reset_token_123 |
|    2 |       2 | 2022-01-02 13:00:00 | reset_token_456 |
|    3 |       3 | 2022-01-03 14:00:00 | reset_token_789 |
|    4 |       4 | 2022-01-04 15:00:00 | reset_token_012 |
|    5 |       5 | 2022-01-05 16:00:00 | reset_token_345 |
+------+---------+---------------------+-----------------+
5 rows in set (0.00 sec)
```

Using second database and creating table products for product_information:



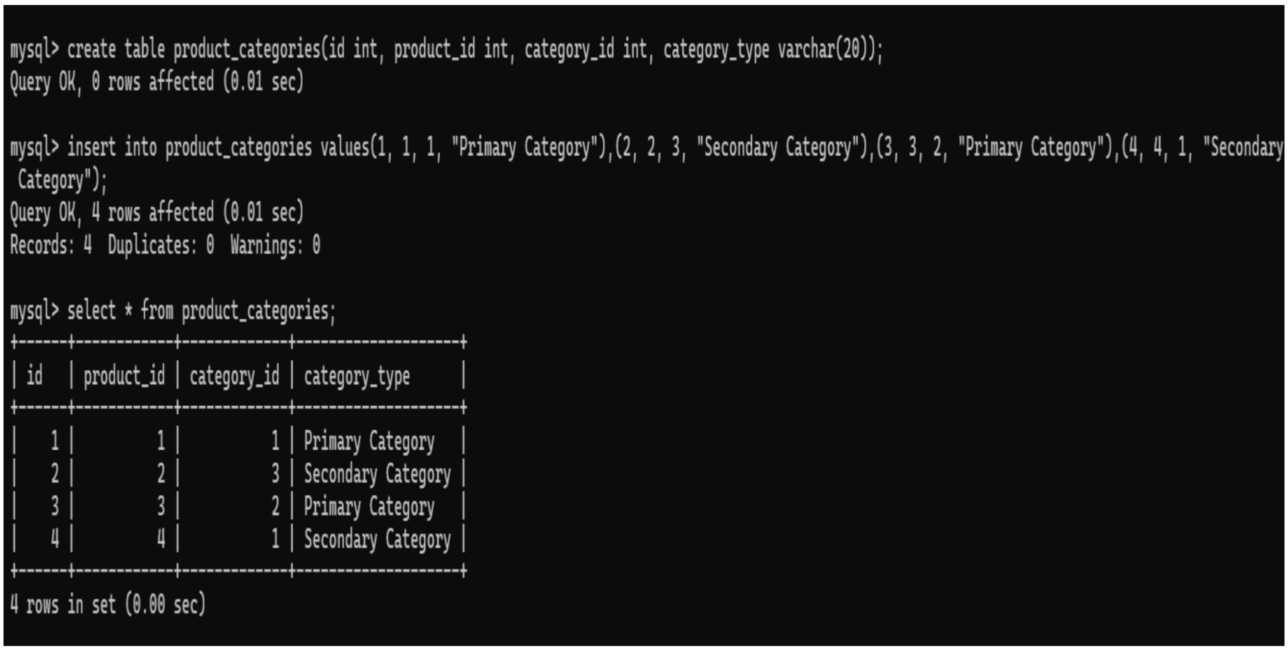
```
mysql> use product_information;
Database changed
mysql> create table products(id int, product_name varchar(30), description varchar(30), price decimal(10,2));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into products values(1, "Product 1", "Description 1", 10.99),(2, "Product 2", "Description 2", 9.99),(3, "Product 3", "Description 3", 12.99),
(4, "Product 4", "Description 4", 8.99),(5, "Product 5", "Description 5", 11.99);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from products;
+------+--------------+---------------+-------+
| id   | product_name | description   | price |
+------+--------------+---------------+-------+
|    1 | Product 1    | Description 1 | 10.99 |
|    2 | Product 2    | Description 2 |  9.99 |
|    3 | Product 3    | Description 3 | 12.99 |
|    4 | Product 4    | Description 4 |  8.99 |
|    5 | Product 5    | Description 5 | 11.99 |
+------+--------------+---------------+-------+
5 rows in set (0.00 sec)
```

Creating table categories for product_information:



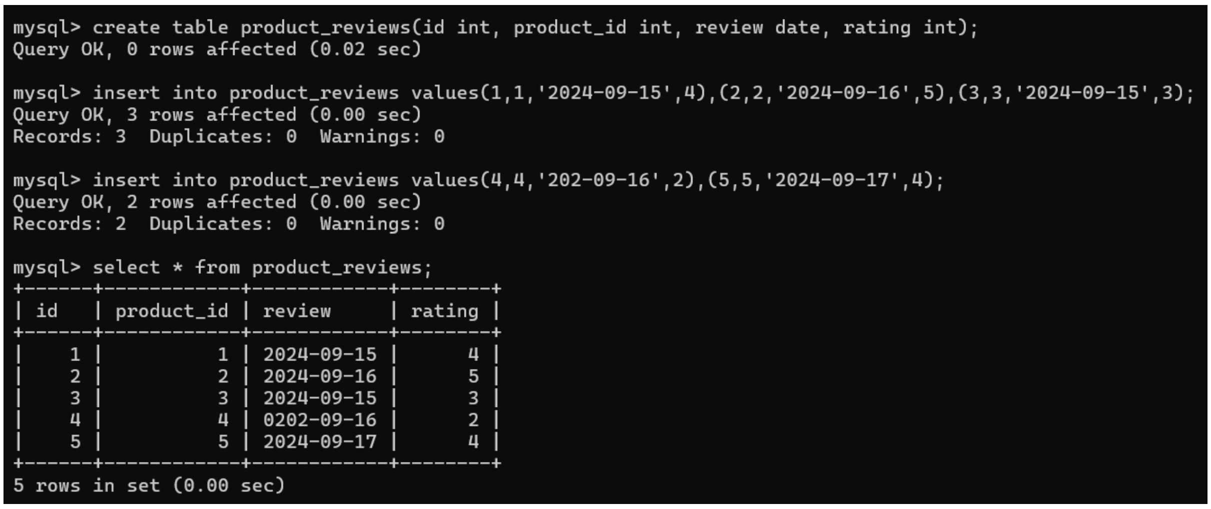
```
mysql> create table categories(id int, category_name varchar(20), descriptio
n varchar(20));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into categories values(1, "Category 1", "Description 1"),(2, "Category 2", "Description 2"),(3, "Category 3", "Description 3"),(4, "Category 4
", "Description 4"),(5, "Category 5", "Description 5");
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from categories;
+------+---------------+---------------+
| id   | category_name | description   |
+------+---------------+---------------+
|    1 | Category 1    | Description 1 |
|    2 | Category 2    | Description 2 |
|    3 | Category 3    | Description 3 |
|    4 | Category 4    | Description 4 |
|    5 | Category 5    | Description 5 |
+------+---------------+---------------+
5 rows in set (0.00 sec)
```

Creating table product_categories for product_information:



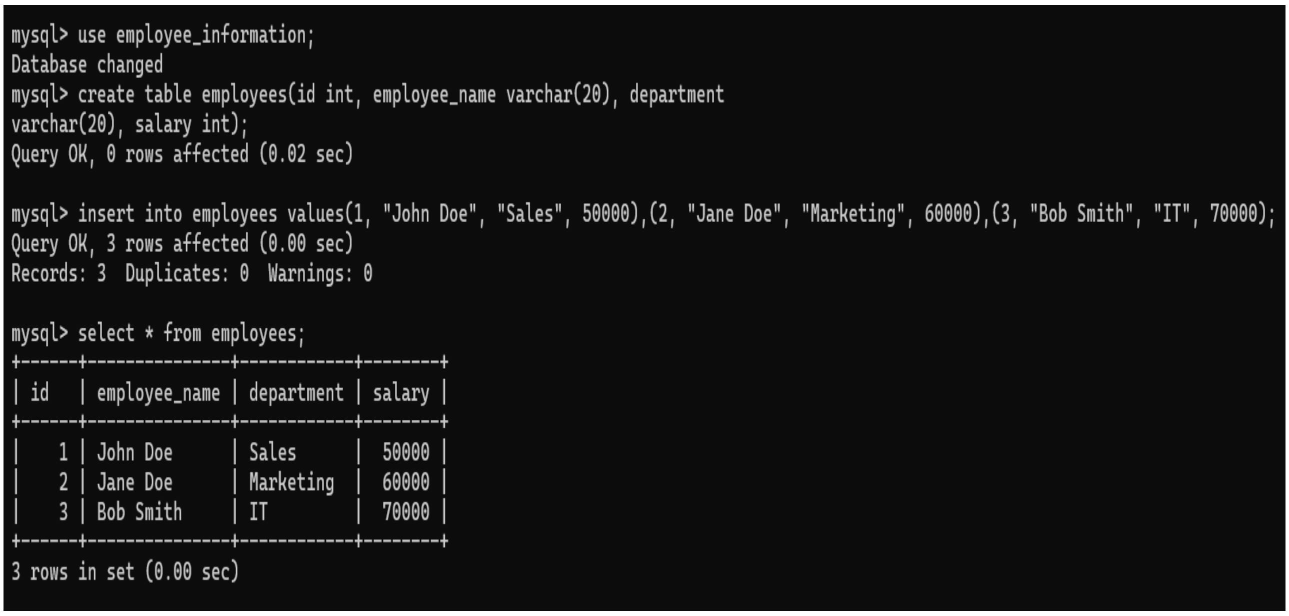
```
mysql> create table product_categories(id int, product_id int, category_id int, category_type varchar(20));
Query OK, 0 rows affected (0.01 sec)

mysql> insert into product_categories values(1, 1, 1, "Primary Category"),(2, 2, 3, "Secondary Category"),(3, 3, 2, "Primary Category"),(4, 4, 1, "Secondary
 Category");
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> select * from product_categories;
+------+------------+-------------+--------------------+
| id   | product_id | category_id | category_type      |
+------+------------+-------------+--------------------+
|    1 |          1 |           1 | Primary Category   |
|    2 |          2 |           3 | Secondary Category |
|    3 |          3 |           2 | Primary Category   |
|    4 |          4 |           1 | Secondary Category |
+------+------------+-------------+--------------------+
4 rows in set (0.00 sec)
```

Creating table product_reviews for product_information:



```
mysql> create table product_reviews(id int, product_id int, review date, rating int);
Query OK, 0 rows affected (0.02 sec)

mysql> insert into product_reviews values(1,1,'2024-09-15',4),(2,2,'2024-09-16',5),(3,3,'2024-09-15',3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> insert into product_reviews values(4,4,'202-09-16',2),(5,5,'2024-09-17',4);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from product_reviews;
+------+------------+------------+--------+
| id   | product_id | review     | rating |
+------+------------+------------+--------+
|    1 |          1 | 2024-09-15 |      4 |
|    2 |          2 | 2024-09-16 |      5 |
|    3 |          3 | 2024-09-15 |      3 |
|    4 |          4 | 0202-09-16 |      2 |
|    5 |          5 | 2024-09-17 |      4 |
+------+------------+------------+--------+
5 rows in set (0.00 sec)
```

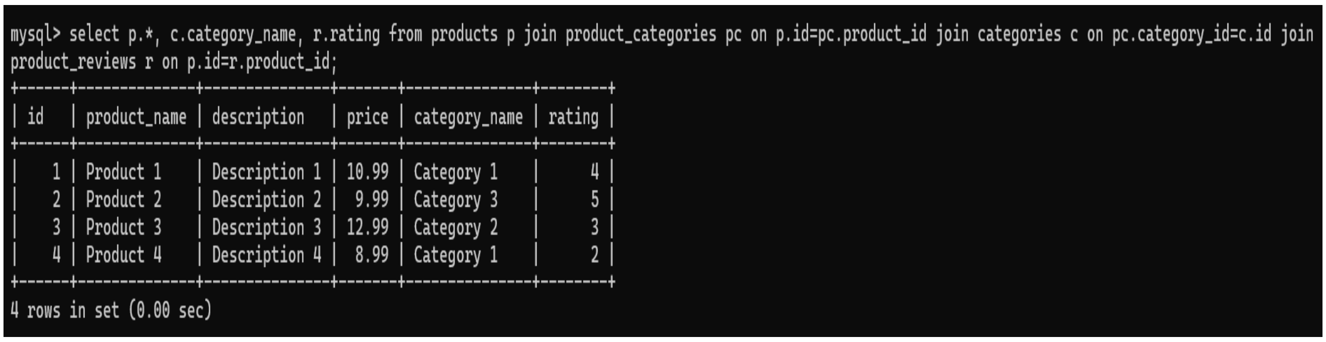
Using third database and creating table employees for employee_information:



```
mysql> use employee_information;
Database changed
mysql> create table employees(id int, employee_name varchar(20), department
varchar(20), salary int);
Query OK, 0 rows affected (0.02 sec)

mysql> insert into employees values(1, "John Doe", "Sales", 50000),(2, "Jane Doe", "Marketing", 60000),(3, "Bob Smith", "IT", 70000);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from employees;
+------+---------------+------------+--------+
| id   | employee_name | department | salary |
+------+---------------+------------+--------+
|    1 | John Doe      | Sales      |  50000 |
|    2 | Jane Doe      | Marketing  |  60000 |
|    3 | Bob Smith     | IT         |  70000 |
+------+---------------+------------+--------+
3 rows in set (0.00 sec)
```

Execution and Results

Retrieve all products with corresponding category and review information



```
mysql> select p.*, c.category_name, r.rating from products p join product_categories pc on p.id=pc.product_id join categories c on pc.category_id=c.id join
product_reviews r on p.id=r.product_id;
+------+--------------+---------------+-------+---------------+--------+
| id   | product_name | description   | price | category_name | rating |
+------+--------------+---------------+-------+---------------+--------+
|    1 | Product 1    | Description 1 | 10.99 | Category 1    |      4 |
|    2 | Product 2    | Description 2 |  9.99 | Category 3    |      5 |
|    3 | Product 3    | Description 3 | 12.99 | Category 2    |      3 |
|    4 | Product 4    | Description 4 |  8.99 | Category 1    |      2 |
+------+--------------+---------------+-------+---------------+--------+
4 rows in set (0.00 sec)
```

Conclusion:

SQL Injection (SQLi) is a major security risk where attackers insert malicious SQL code into a database through web applications.
Common Attack Vectors:
SQLi can occur through user input forms, URL parameters, cookies, HTTP headers, and detailed error messages.

Types of Attacks:

1. Classic SQL Injection: Directly manipulates SQL queries.
2. Blind SQL Injection: Infers data from application responses.
3. Time-Based SQL Injection: Uses delays to extract information.
4. Out-of-Band SQL Injection: Sends data to an external server.

Prevention:
To prevent SQLi, use prepared statements, parameterize queries, validate user inputs, and limit database privileges.

Uses of SQL Injection:
SQLi can be used for data extraction, modification, authentication bypass, privilege escalation, and more.

Database Structure:
The project involves three databases: User_Authentication, Product_Information, and Employee_Information, with relevant tables for each.

Execution:
Run queries in Command Line Interface to test and identify SQL Injection vulnerabilities.

Conclusion:
SQL Injection remains a critical threat. Implementing proper defenses like prepared statements and input validation is essential for web application security.