IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, October 2026
Impact Factor: 8.2

SQL Injection: A Study of Attacks and Prevention Techniques

Prof. Rana Afreen Sheikh[1], Amar S. Ingle[1] , Prathamesh R. Malviya[2]
Assistant Professor, Department of MCA, Vidya Bharati Mahavidyalaya, Amravati, India [1]
Student, Department of MCA, Vidya Bharati Mahavidyalaya, Amravati, India [2]
Student, Department of MCA, Vidya Bharati Mahavidyalaya, Amravati, India [3]

Abstract: *SQL injection is a database security vulnerability in which untrusted application input changes the meaning of an SQL statement. The vulnerability occurs when an application allows user-controlled values to become part of query structure rather than treating them only as data. A successful attack can expose confidential information, alter records, or enable unauthorized database access. This paper reviews the causes, consequences, attack classifications, detection approaches, and prevention techniques associated with SQL injection. The literature indicates that no single detection or prevention mechanism addresses every attack condition. Static analysis, runtime validation, automated testing, and behavior-oriented detection address different stages of the problem, while parameterized queries provide the central application-level defense. Input validation, carefully designed stored procedures, controlled error handling, and least-privilege database accounts provide additional protection*

Keywords: SQL injection, database security, web application security, detection, prevention, prepared statements, parameterized queries

I. INTRODUCTION

Web applications commonly use relational databases to store and retrieve information. Their normal operations include reading records, inserting new data, updating existing information, and deleting information. Because these operations are often controlled through requests received from users, the boundary between application input and database commands must be protected. A taxonomy study of SQL injection describes database-driven web applications as systems that depend on SQL for these operations and identifies SQL injection as a serious threat to them [5].

SQL injection occurs when an attacker supplies input that changes the intended effect of a dynamically generated query. The attack does not require the attacker to obtain direct access to the database first; the vulnerable application can act as the channel through which the database receives the altered request. Research on stored procedures explains that malicious query fragments supplied as input can produce a different database request and can be used to obtain or modify confidential information [6].

The consequences depend on the vulnerable function and the permissions of the database account used by the application. Reported effects include unauthorized access, disclosure of sensitive information, modification or deletion of records, and interference with the confidentiality, integrity, or availability of database information [7], [8]. SQL injection therefore requires attention at more than one layer: secure query construction is the first requirement, while database permissions, error handling, testing, and monitoring reduce residual risk [4].

1.1 Aim

The aim of this paper is to examine SQL injection attacks and the techniques used to detect and prevent them in database-driven web applications.

1.2 Objectives

• To explain how SQL injection vulnerabilities arise.

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/IJARSCT-38858

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 2, October 2026 Impact Factor: 8.2

- To describe the main ways in which SQL injection attacks are classified.
- To examine static, dynamic, runtime, and automated detection approaches.
- To evaluate parameterized queries, input validation, stored procedures, access control, and related safeguards.
- To compare the strengths and limitations of the principal approaches.
- To identify secure development practices that reduce SQL injection risk.

1.3 Scope of the Study

This paper focuses on SQL injection in database-driven web applications. It discusses the problem from a defensive and research perspective and does not provide instructions for attacking real systems. The review emphasizes scholarly surveys and technical studies that examine attack categories, detection, and prevention. It does not claim to provide a systematic review of every publication in the field.

II. METHODOLOGY

This study uses a literature-based review methodology. Scholarly surveys, technical studies, and research papers concerning SQL injection vulnerabilities, attack classifications, detection methods, and prevention techniques were examined and synthesized. The reviewed evidence was organized into four main themes: causes and attack classifications, detection approaches, prevention techniques, and comparative strengths and limitations. The study does not present a new experiment, benchmark, or implementation; therefore, conclusions are based on the findings and limitations reported in the selected literature.

III. LITERATURE REVIEW

The literature describes SQL injection as a continuing security problem in database-driven web applications and presents multiple taxonomies and countermeasures [1], [2], [5]. Research on detection includes static analysis, dynamic testing, runtime validation, pattern matching, syntactic and semantic analysis, and learning-based approaches [2], [6], [14], [17]. Countermeasure studies emphasize parameterized queries, input validation, secure stored procedures, controlled error handling, least-privilege database accounts, and layered controls [3], [4], [18], [19].

The reviewed studies do not establish one universally effective detector or prevention mechanism. Different methods address different stages of the problem, and evaluations may use different datasets, applications, attack sets, and performance measures [3]. This supports a defense-in-depth approach in which secure query construction is reinforced by validation, testing, monitoring, error control, and access restrictions.

IV. BACKGROUND OF SQL INJECTION

SQL injection is a form of code injection that exploits the way an application constructs database queries. When application code concatenates SQL commands with untrusted parameters, a parameter may be interpreted as part of the command instead of as a value [9]. The underlying design error is a failure to keep query structure separate from data. An application may intend to perform one fixed database operation, but unsafe construction allows input to influence operators, conditions, or other elements of the statement [2], [6].

The resulting query may return information that the user was not authorized to see or perform an operation that the application did not intend. The problem is not limited to one programming language or database product. It can appear wherever an application builds SQL dynamically and fails to control the boundary between commands and values. It can also occur inside stored procedures when those procedures construct dynamic SQL from user input [6].

V. TYPES AND CLASSIFICATIONS OF SQL INJECTION ATTACKS

5.1 Query-Manipulation Attacks

In query-manipulation attacks, supplied input changes the logic or structure of the statement generated by the application. The purpose may be to bypass an intended condition, change which records are returned, or cause an

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, October 2026
Impact Factor: 8.2

unauthorized operation. The defining feature is that the input is processed as SQL syntax instead of remaining a data value [9], [12].

5.2 Data-Extraction Attacks

Some attacks attempt to make the application return information from database structures or records that were not part of the original user request. The literature describes SQL injection as a means of retrieving sensitive data and obtaining unauthorized access to the underlying database [2], [13].

5.3 Error- and Response-Based Attacks

An attacker may learn about the database through differences in application responses. Error messages can reveal implementation details, while changes in response content or behavior can provide information even when the application does not display database records directly. The research literature consequently includes response behavior, query semantics, and error handling in discussions of detection and defense [4], [14].

5.4 Blind and Advanced Variants

When direct database output is unavailable, attacks can rely on indirect observations such as application behavior or response timing. Research on advanced SQL injection includes deep-blind variants and examines classical as well as newer detection and prevention methods [15]. These variants are difficult because the application may appear to function normally while small response differences reveal information.

5.5 Impact-Based Classification

SQL injection can also be classified by the security property affected. A successful attack may compromise confidentiality by exposing data, integrity by changing or deleting records, or availability by disrupting database or application operations. This classification is useful for risk assessment because it connects the technical weakness to the potential business impact [4], [8].

VI. DETECTION OF SQL INJECTION

Detection methods seek to identify vulnerable code before deployment, malicious input during testing, or abnormal queries during operation. A systematic review examined a pool of 60 papers on SQL injection detection in web applications, showing that the research area contains several distinct technical approaches rather than one standard detector [2].

6.1 Static Analysis

Static analysis examines application or database code without relying on a live attack. It can identify unsafe construction patterns and determine where user input reaches SQL statements. Wei, Muthuprasanna, and Kothari combine static application analysis with a stored-procedure parser and compare intended query structure with the structure produced after input is included [6]. Static analysis is useful early in development because it can identify weaknesses before deployment. Its limitation is that it may not fully capture behavior created at runtime.

6.2 Dynamic Testing

Dynamic testing evaluates a running application by sending test inputs and observing its responses. It can reveal whether an input point changes query behavior, produces unexpected database results, or exposes an error. Its effectiveness depends on test coverage; an input point, application branch, or attack variant that is not exercised may remain undetected [11], [16].

6.3 Runtime Validation and Monitoring

Runtime approaches inspect queries or application behavior while the system operates. A query can be compared with an expected structure, or the system can monitor syntactic, semantic, and behavioral characteristics for anomalies. Runtime protection can observe activity that static analysis or pre-deployment testing misses, but it may introduce processing cost and must distinguish legitimate variation from malicious behavior [3], [6].

6.4 Pattern Matching and Learning-Based Detection

Research surveys group SQL injection detection methods into categories such as data analysis, feature training, syntactic and semantic analysis, and pattern matching [14]. Machine-learning-based studies attempt to identify

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, October 2026
Impact Factor: 8.2

suspicious query or input patterns from examples of legitimate and malicious behavior [17]. Learning-based detection may identify patterns that are difficult to describe with fixed signatures, but it depends on representative training data and appropriate evaluation.

VII. PREVENTION TECHNIQUES

7.1 Parameterized Queries and Prepared Statements

Parameterized queries separate the SQL statement from the values supplied by a user. The application defines the query structure and passes input as a parameter, reducing the possibility that the database will interpret the value as SQL syntax [18], [19]. Parameterized queries should be the default method for database access wherever the application framework supports them. They address the main cause of injection, although authorization, secure database permissions, and testing remain necessary.

7.2 Input Validation

Input validation checks whether a value matches the format and range expected for a particular field. Validation is an additional control, not a substitute for parameterization. A blacklist may miss an unanticipated representation of malicious input, while an allowlist is practical when the valid input domain is known [4], [19].

7.3 Stored Procedures

Stored procedures can centralize database operations, but their security depends on implementation. A procedure that uses parameters safely can support a strong design; a procedure that constructs dynamic SQL from untrusted values can remain vulnerable [6]. Stored procedures should therefore be parameterized and tested rather than assumed to be secure automatically.

7.4 Error Handling

Detailed database errors should not be exposed to ordinary users. Error responses can disclose information about database structure and query processing. Applications should record diagnostic information securely for administrators while returning controlled messages to users [4].

7.5 Least-Privilege Database Accounts

The application should connect to the database with only the permissions required for its normal functions. Limiting permissions cannot repair unsafe query construction, but it can reduce the consequences of a successful exploit by preventing unnecessary reads, writes, or administrative operations [4].

7.6 Web Application Firewalls and Layered Controls

A web application firewall or request-filtering mechanism may identify suspicious requests before they reach application logic. However, network and filtering controls alone do not address unsafe query construction. Effective prevention requires controls at application, database, and network levels [4].

VIII. RESULTS AND DISCUSSION

The literature comparison shows that SQL injection prevention should be treated as a layered engineering problem. Parameterized queries provide the most direct protection against the central data-versus-code confusion. Validation, secure procedures, error control, monitoring, and least privilege address different stages or consequences of the risk [3], [4], [18].

Approach	Primary Strength	Main Limitation
Static analysis	Identifies unsafe input-to-query flows before deployment.	May miss runtime or dynamic query behavior.
Dynamic testing	Exposes vulnerabilities through observed application behavior.	Coverage depends on inputs, paths, and variants tested.
Runtime validation	Can detect activity missed by code review or pre-deployment testing.	May add overhead and must distinguish legitimate variation.
Parameterized queries	Directly separates query structure from	Does not replace authorization, validation,

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, October 2026
Impact Factor: 8.2

	user-supplied values.	access control, or testing.
Input validation	Reduces unexpected input and supports application rules.	Should not be the only SQL injection defense.
Secure stored procedures	Can centralize database logic with safe parameter handling.	Dynamic SQL inside a procedure can remain vulnerable.
Least privilege	Reduces potential impact if another control fails.	Does not prevent the query-construction vulnerability itself.
Layered controls	Provides defense in depth across failure points.	Network-level controls alone are insufficient.

The literature agrees on the basic cause of SQL injection: untrusted input reaches a query in a form that permits the input to influence SQL structure. Detection research has expanded from signature and rule-based techniques to static analysis, runtime validation, syntactic and semantic analysis, behavior-oriented methods, and machine learning [1], [17]. At the same time, the evidence should be interpreted carefully because studies do not always use the same datasets, test applications, attack sets, or performance measures [3].

For practical development, prevention should begin with parameterized queries and safe database APIs. Validation should enforce expected business and data rules, while stored procedures should be reviewed for unsafe dynamic SQL. Database permissions should be limited, errors should be handled without disclosing internal details, and testing should cover the application's input paths [4], [6].

IX. ADVANTAGES OF SECURE SQL INJECTION PREVENTION

A secure prevention strategy can protect the confidentiality and integrity of database records by reducing the chance that input will be interpreted as executable SQL [8]. It can also reduce unauthorized database operations and limit the impact of a successful compromise when least-privilege permissions are used [4]. Secure query construction improves the consistency of application development because database access follows a defined interface rather than ad hoc string concatenation. These benefits are strongest when secure coding, testing, permissions, and monitoring are applied together.

X. LIMITATIONS

This paper is a literature-based study and does not present a new experiment, benchmark, or implementation. The conclusions therefore depend on the designs, terminology, test environments, and evaluation criteria used by the selected studies. The countermeasure literature itself reports that many techniques have been tested on limited scales and that no single approach provides flawless protection [3].

The reviewed evidence does not support a single numerical ranking of all prevention methods. Parameterized queries, validation, runtime monitoring, stored procedures, and access controls serve different purposes, so their results cannot be combined into one common performance score without a controlled comparison.

XI. FUTURE SCOPE

Future work can evaluate detection and prevention methods on common benchmark applications with the same attack corpus and reporting criteria. Such a design would make it easier to compare detection accuracy, false positives, computational overhead, coverage of blind or obfuscated variants, and resistance to evasion.

Machine-learning-based detection remains an active research direction, but future studies should report the composition of training and test data, the balance between legitimate and malicious examples, and the ability of the model to generalize beyond the development data [14], [17]. Future studies should also examine secure stored procedures and dynamic SQL separately because a stored procedure should not be classified as secure merely because it runs in the database layer [6].

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, October 2026
Impact Factor: 8.2

XII. CONCLUSION

SQL injection is caused by an unsafe relationship between application input and SQL query structure. It can allow unauthorized access, disclosure, modification, or deletion of database information and can affect confidentiality, integrity, and availability [2], [8]. The literature contains several taxonomies and detection strategies. Static analysis examines code, dynamic testing examines running behavior, runtime validation compares executed behavior with expected query structure, and newer approaches use syntax, semantics, patterns, behavior, or machine learning. These methods are useful for different purposes, but research reviews indicate that no single approach guarantees complete protection [3], [6], [14].

The most important development practice is to separate SQL commands from user-provided values through parameterized queries or safely implemented parameterized procedures. Input validation, controlled error handling, least-privilege database accounts, testing, monitoring, and additional filtering controls should reinforce that foundation. SQL injection prevention is therefore best understood as defense in depth: prevent unsafe interpretation at the application boundary, detect abnormal behavior, and limit the damage if another control fails [4], [18].

ACKNOWLEDGMENT

The authors express their sincere gratitude to Prof. Rana Afreen Sheikh for guidance and support during the preparation of this study.

REFERENCES

[1] D. A. Kindy and A. K. Pathan, “A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques,” 2011 IEEE 15th International Symposium on Consumer Electronics, 2011, doi: 10.1109/isce.2011.5973873.

[2] M. Nasereddin, A. ALKhamaiseh, M. Qasaimeh, and R. S. Al-Qassas, “A systematic review of detection and prevention techniques of SQL injection attacks,” Information Security Journal, 2021, doi: 10.1080/19393555.2021.1995537.

[3] M. Alenezi, M. Nadeem, and R. Asif, “SQL injection attacks countermeasures assessments,” Indonesian Journal of Electrical Engineering and Computer Science, 2021, doi: 10.11591/ijeecs.v21.i2.pp1121-1131.

[4] K. Ahmad and M. Karim, “A method to prevent SQL injection attack using an improved parameterized stored procedure,” 2021, doi: 10.14569/ijacsa.2021.0120636.

[5] A. Sadeghian, M. Zamani, and S. M. Abdullah, “A taxonomy of SQL injection attacks,” 2013 International Conference on Informatics and Creative Multimedia, 2013, doi: 10.1109/icicm.2013.53.

[6] K. Wei, M. Muthuprasanna, and S. C. Kothari, “Preventing SQL injection attacks in stored procedures,” Australian Software Engineering Conference, 2006, doi: 10.1109/aswec.2006.40.

[7] G. Yiğit and M. Arnavutoğlu, “SQL injection attacks detection & prevention techniques,” International Journal of Computer Theory and Engineering, 2017, doi: 10.7763/ijcte.2017.v9.1165.

[8] A. Sadeghian, M. Zamani, and A. A. Manaf, “A taxonomy of SQL injection detection and prevention techniques,” 2013, doi: 10.1109/icicm.2013.18.

[9] M. Garcia, “SQL injection attacks and prevention techniques,” 2006 Annual Conference & Exposition Proceedings, 2020, doi: 10.18260/1-2--195.

[10] S. M. S. Sajjadi and B. T. Pour, “Study of SQL injection attacks and countermeasures,” International Journal of Computer and Communication Engineering, 2013, doi: 10.7763/ijcce.2013.v2.244.

[11] B. Nagpal, N. Chauhan, and N. Singh, “A survey on the detection of SQL injection attacks and their countermeasures,” Journal of Information Processing Systems, 2015, doi: 10.3745/jips.03.0024.

[12] K. Umar, A. B. M. Sultan, H. Zulzalil, N. Admodisastro, and M. T. Abdullah, “SQL injection attack roadmap and fusion,” Indian Journal of Science and Technology, 2016, doi: 10.17485/ijst/2016/v9i28/97810.

[13] J. Majumder and G. Saha, “Analysis of SQL injection attack,” International Journal of Computer Science and Informatics, 2013, doi: 10.47893/ijcsi.2013.1102.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 2, October 2026
Impact Factor: 8.2

[14] J. Xu, M. Ni, D. Zhu, and X. Yu, "Overview of SQL injection attack detection techniques," CNML, 2023, doi: 10.1145/3640912.3640956.
[15] J. P. Singh, "Analysis of SQL injection detection techniques," Theoretical and Applied Informatics, 2017, doi: 10.20904/281-2037.
[16] N. Joshi, T. Sheth, V. Shah, J. Gupta, and S. F. Mujawar, "A detailed evaluation of SQL injection attacks, detection and prevention techniques," International Conference on Awareness Science and Technology, 2022, doi: 10.1109/ICAST55766.2022.10039662.
[17] J. Hu, W. Zhao, and Y. Cui, "A survey on SQL injection attacks, detection and prevention," Proceedings of the 2020 12th International Conference on Machine Learning and Computing, 2020, doi: 10.1145/3383972.3384028.
[18] H. Zhang, "A method to defend against SQL injection attack based on parameter query," Dalian Minzu Xueyuan xuebao, 2012.
[19] B. Kumar and P. P. Anaswara, "Vulnerability detection and prevention of SQL injection," International Journal of Engineering & Technology, 2018, doi: 10.14419/ijet.v7i2.31.13388.