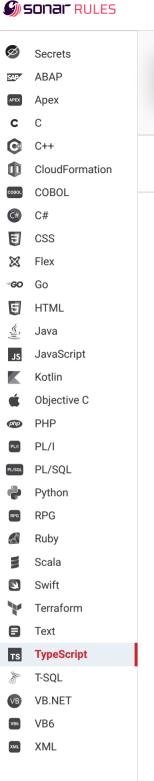
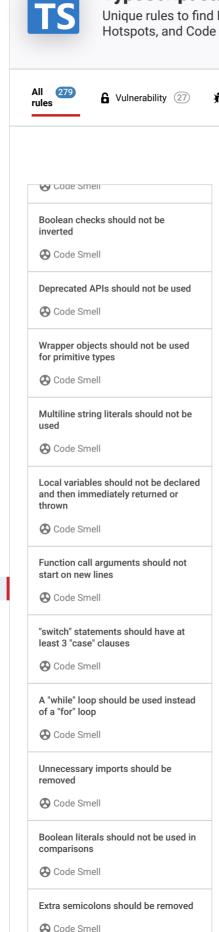
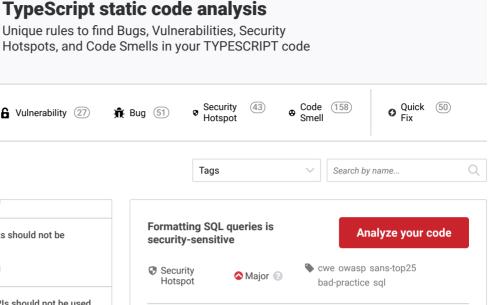


Products >





Class names should comply with a



Formatted SQL queries can be difficult to maintain, debug and can increase the risk of SQL injection when concatenating untrusted values into the query. However, this rule doesn't detect SQL injections (unlike rule {rule:javascript:S3649}), the goal is only to highlight complex/formatted queries

Ask Yourself Whether

- Some parts of the query come from untrusted values (like user inputs).
- The query is repeated/duplicated in other parts of the code.
- The application must support different types of relational databases.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

- Use parameterized queries, prepared statements, or stored procedures and bind variables to SQL query parameters.
- . Consider using ORM frameworks if there is a need to have an abstract layer to access data

Sensitive Code Example

```
// === MySQL ===
const mysql = require('mysql');
const mycon = mysql.createConnection({ host: host, user: use
mycon.connect(function(err) {
  mycon.query('SELECT * FROM users WHERE id = ' + userinput,
// === PostgreSQL ===
const pg = require('pg');
const pgcon = new pg.Client({ host: host, user: user, passwo
pgcon.connect();
pgcon.query('SELECT * FROM users WHERE id = ' + userinput, (
```

Compliant Solution

```
// === MySQL ===
const mysql = require('mysql');
const mycon = mysql.createConnection({ host: host, user: use
mycon.connect(function(err) {
  mycon.query('SELECT name FROM users WHERE id = ?', [userin
});
// === PostgreSQL ===
const pg = require('pg');
const pgcon = new pg.Client({ host: host, user: user, passwo
```

TypeScript static code naming convention

Code Smell

Track uses of "TODO" tags

Code Smell

Web SQL databases should not be used

Vulnerability

Variables declared with "var" should be declared before they are used

Code Smell

Track lack of copyright and license headers

pgcon.connect();
pgcon.query('SELECT name FROM users WHERE id = \$1', [userinp

Exceptions

This rule's current implementation does not follow variables. It will only detect SQL queries which are formatted directly in the function call.

const sql = 'SELECT * FROM users WHERE id = ' + userinput; mycon.query(sql, (err, res) => $\{\}$); // Sensitive but no issu

See

- OWASP Top 10 2021 Category A3 Injection
- OWASP Top 10 2017 Category A1 Injection
- <u>MITRE, CWE-89</u> Improper Neutralization of Special Elements used in an SQL Command
- MITRE, CWE-564 SQL Injection: Hibernate
- MITRE, CWE-20 Improper Input Validation
- <u>MITRE, CWE-943</u> Improper Neutralization of Special Elements in Data Query Logic
- SANS Top 25 Insecure Interaction Between Components
- Derived from FindSecBugs rules Potential SQL/JPQL Injection (JPA), Potential SQL/JDQQL Injection (JDO), Potential SQL/HQL Injection (Hibernate)

Available In:

sonarcloud 🙆 | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy