Engineer

SQL injection

an attack that turns input into part of a database query

What it is

SQL injection, often shortened to SQLi, is the attack in which an application builds an SQL command out of text that someone else supplied, and the database ends up reading part of that text as SQL rather than as a value. The Common Weakness Enumeration lists it as CWE-89 and describes the weakness as a command built wholly or partly from outside input, without first neutralising the characters that carry special meaning in SQL. OWASP's prevention cheat sheet puts the same idea more plainly: the attack works when untrusted data becomes part of the syntax of a query.

The root of the trouble is that SQL draws no firm line between the instructions in a command and the data inside it. A single quotation mark ends a string literal, and whatever the attacker types after it is parsed as more SQL. Two hyphens start a comment, which lets the attacker discard the rest of the developer's statement. A semicolon, where the database and driver allow it, ends one statement and starts another. None of them is exotic: the same marks turn up in surnames and free text.

OWASP's Web Security Testing Guide sorts the attacks into three classes by how the results reach the attacker. In-band attacks read the answer through the same channel they used to send it, typically on the application's own page. Out-of-band attacks have the database deliver the results somewhere else, for instance over a separate HTTP connection. Inferential or blind attacks receive no data at all, and rebuild it one question at a time from how the application behaves. The guide names five common techniques across those classes: a `UNION` operator that joins a second `SELECT` onto the first, Boolean conditions, errors provoked on purpose, out-of-band requests, and deliberate time delays.

SQL injection belongs to a larger family. In CWE it sits under the general injection weakness, CWE-74, alongside cross-site scripting, and the OWASP Top Ten for 2025 places both in its A05 Injection category. The common thread is data crossing a boundary into a place where it is read as instructions. Here the interpreter is the database's SQL parser, and the cure is to keep every value out of the SQL text altogether.

Use the concept in a decision

Put "What is SQL injection?" to work in a real decision. We can help relate the concept to the system you are building or buying. Accounts, passwords and permissions arranged so people hold exactly the access their job needs, and leavers lose it that day.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

The damage depends on what the application's database account may do, and that is often a great deal. CWE-89 lists reading data the attacker should never see, getting past authentication without knowing a password, changing the data that decides who may do what, modifying or deleting records, and in some set-ups running operating-system commands. OWASP's attack page adds administrative operations such as shutting the database down, reading files from the database server, and taking over the database server as its administrator.

Both sources rate it severe. CWE gives the likelihood of exploit as High, and OWASP's attack page describes the flaw as both easy to find and easy to exploit. The weakness appears in every edition of the CWE Top 25 Most Dangerous Software Weaknesses from 2019 to 2025. The CWE entry also records a 2008 wave in which one SQL injection string, working against many different programs, compromised a large number of web servers and made them serve malicious code.

The defence, by contrast, is well understood and cheap. OWASP's cheat sheet makes prepared statements with bound parameters its first defence, and CWE recommends them too: the SQL and the values travel separately, so the database cannot mistake one for the other. Everything else, from least privilege to careful error handling, is a second layer that limits the damage when a mistake slips through.

What it looks like in practice

Take a shop whose order history page looks up orders by the email address typed into a form. The developer builds the SQL with a Python f-string:

```
cur.execute(f"SELECT id, total FROM orders WHERE email = '{email}'")
```

A visitor types `x' OR '1'='1` into the field. The program pastes it in, and the database receives this:

```
SELECT id, total FROM orders WHERE email = 'x' OR '1'='1'
```

The quotation mark after `x` closes the string the developer opened, and the `OR` clause after it is always true, so the database returns every order in the table. CWE-89 and the testing guide use the same always-true trick, which can also let a login form accept a visitor who knows neither the username nor the password.

A worse input ends the statement and starts another. The CWE entry's example enters `name'; DELETE FROM items; --` as an item name, so that a second statement follows the first and the hyphens comment out the rest. Whether that runs depends on the database and the driver, and several current interfaces refuse it. PostgreSQL's `PQexecParams` takes at most one SQL command per string, which its documentation counts as of some use against SQL injection. Python's `sqlite3` module raises an error whenever `execute()` receives more than one statement, and psycopg refuses several statements in one call whenever parameters are passed. None of that stops the always-true condition above, which needs only one statement.

The fix is to pass the value separately. PEP 249, Python's database interface standard, has each driver declare its placeholder style in a module constant called `paramstyle`, and lists five styles: `qmark` (`?`), `numeric` (`:1`), `named` (`:name`), `format` (`%s`) and `pyformat` (`%(name)s`). The `sqlite3` module is fixed to `qmark` and also accepts `named`. Psycopg, the PostgreSQL driver, takes `%s` and `%(name)s`:

```
cur.execute(
    "SELECT id, total FROM orders WHERE email = %s",
    (email,),
)
```

The `%s` looks like Python string formatting, but nothing is formatted: the SQL and the tuple of values reach the driver as two separate arguments. Psycopg 3 keeps them apart all the way, sending the command and its parameters to the server separately rather than merging them on the client. OWASP's cheat sheet makes the point with its own sample input, `tom' or '1'='1` : bound as a parameter, it is a strange user name looked up literally, and it matches nobody.

PostgreSQL offers the same split in plain SQL. `PREPARE` fixes the statement with numbered parameters, and `EXECUTE` supplies the values later:

```
PREPARE order_lookup (text) AS
    SELECT id, total FROM orders WHERE email = $1;
EXECUTE order_lookup('x'' OR ''1''=''1');
```

The hostile input, written here as an SQL literal, arrives in `$1` as one odd-looking email address and finds no orders.

On a platform holding professional and organisational data, input is validated before anything touches the database, and then the API connects as a least-privilege role that may only execute the app's functions, with everything parameterised. Binding keeps data from turning into SQL, and the narrow role limits what a mistake could reach.

Where it goes wrong

Building the string anyway. An f-string, the `%` operator, `+` or `.format()` merges the value into the SQL before the driver sees it, and so does a placeholder written inside quotation marks, such as `'%s'` . OWASP notes that concatenating untrusted values before preparing a statement defeats the whole protection, and psycopg's documentation tells developers never to use `%` or `+` to merge values into queries. Pass every value as the second argument of `execute()` , with the placeholder bare.

Escaping by hand. Doubling quotation marks or adding backslashes looks equivalent, but the rules depend on the SQL dialect and its settings. OWASP's example is MySQL's `NO_BACKSLASH_ESCAPES` mode, which switches backslash escaping off, and the cheat sheet marks escaping as strongly discouraged. Replace hand-escaped strings with bound parameters.

Validation or a WAF treated as the fix. Validation catches mistakes, but free text has to allow the characters an attacker needs: the CWE entry's example is the surname O'Reilly, which would likely pass validation and still contains a quotation mark. Denylists of dangerous words have loopholes, and CWE rates an application firewall's effectiveness as only moderate. As OWASP puts it, "Validation of a value does not make it safe to concatenate into SQL." Keep validating for correctness, and bind for safety.

Table names, column names or sort order taken from input. Placeholders stand for values only, so a developer who needs a user to choose a column or the direction of `ORDER BY` is tempted to paste the text in. OWASP's answer is to map each choice to a small set of names the application owns and to reject anything else, for example turning a requested direction into the literal `ASC` or `DESC`. When a name really must be dynamic, psycopg's `psycopg.sql` module composes identifiers, though OWASP notes that quoting a name does not limit which tables the application can be steered to.

Stored procedures assumed to be safe. Calling a procedure with bound arguments protects the call, not the SQL the procedure builds inside itself. CWE notes that using a stored procedure rules out only some injection paths and leaves others possible. Inside the database, use static SQL, or bind values in dynamic SQL too; in PostgreSQL that means `EXECUTE ... USING` for values and `format('%I', ...)` for identifiers.

An ORM with strings stitched in. An object-relational mapper binds values when it is used as designed, and CWE credits persistence layers such as Hibernate with significant protection when used properly. OWASP adds that Hibernate queries can still be injected when developers concatenate input into the query text. Use the mapper's parameters, not string building.

An application account with administrator rights. If the account can do everything, so can the attacker. OWASP asks for read-only accounts where only reads are needed, a separate database user for each web application, views that expose only the columns an account needs, and no administrator rights for

application accounts. When an application works only through stored procedures, it should be allowed to execute those and nothing else. Least privilege limits the damage; it does not prevent the injection.

Detailed database errors shown to users. An error message that reveals the statement's structure helps an attacker tailor the next attempt, and the testing guide's error-based technique relies on exactly that. CWE recommends minimal error messages. Show the user a generic failure and keep the detail on the server for the team.

Trusting data already in the database. A value that was stored safely can still be dangerous when a later piece of code pastes it into SQL. OWASP says its controls apply wherever the data originates, be it an HTTP request, another service or an earlier write to storage, and CWE warns that later processes may forget to escape what was saved. Bind values read from your own tables just as you bind input from a form.

Further reading

- CWE-89 in the Common Weakness Enumeration — the weakness entry: consequences, examples, and the mitigations with their limits.
- OWASP SQL Injection Prevention Cheat Sheet — parameterised queries, stored procedures, dynamic identifiers and least privilege.
- OWASP Web Security Testing Guide: Testing for SQL Injection — the attack classes and techniques, as a tester meets them.
- PEP 249: Python Database API Specification v2.0 — the interface Python drivers share, including the five placeholder styles.
- Psycopg 3: Passing parameters to SQL queries — the rules for placeholders and the right way to build dynamic SQL.
- PostgreSQL: Command Execution Functions — how libpq sends parameters apart from the command string.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is SQL injection?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/sql-injection/