# Preventing SQL Injection Vulnerabilities

## Input Validation / Sanitization

SQL injection happens because developers create dynamic queries using user input that isn't properly sanitized. You (as a developer) should always sanitize user input, and if it is expected to match a certain form (e.g. email) then validate. The best mindset is to treat all user input as if it were dangerous.

## Parameterized Queries

Using parameterized queries is a very good way to avoid SQLi vulnerabilities, because you pass the query and variables separately allowing the server to understand what is code and what is data, regardless of user input.

Here is an example of a vulnerable SQL query that concatenates user input into the query.

Code: php

...  
$sql = "SELECT email FROM accounts WHERE username = '" . $\_POST['username'] . "'";  
$stmt = sqlsrv\_query($conn, $sql);  
$row = sqlsrv\_fetch\_array($stmt, SQLSRV\_FETCH\_ASSOC);  
...  
sqlsrv\_free\_stmt($stmt);   
...

This is how the same query would look like if it were parameterized. It's a small change, but it's the difference between vulnerable and secure code.

Code: php

$sql = "SELECT email FROM accounts WHERE username = ?";   
$stmt = sqlsrv\_query($conn, $sql, array($\_POST['username']));   
$row = sqlsrv\_fetch\_array($stmt, SQLSRV\_FETCH\_ASSOC);   
...  
sqlsrv\_free\_stmt($stmt);

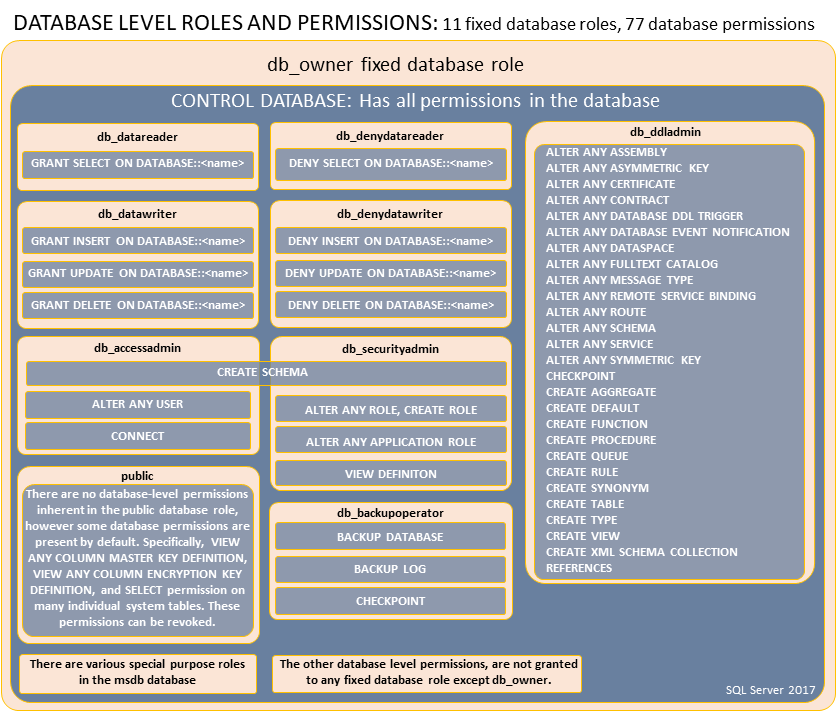
**Note:** Even after all of this, we should still not completely trust all user-data stored in the db, as we may always miss something and the user may be able to store something malicious in the db. This is why it is also recommended to also apply sanitization/filtering on data output, escpecially when outputting user-generated data. This way, we prevent 2nd-level SQL attacks, which execute upon data output instead of data input.

## MSSQL-Specific Precautions

Regarding MSSQL specifically, there are a couple of things you may want to do to prevent MSSQL-specific attacks.

#### Don't Run Queries as Sysadmin!

First and foremost, don't use sa to run your queries. More concretely, use an account with [as few privileges as possible](https://www.paloaltonetworks.com/cyberpedia/what-is-the-principle-of-least-privilege). Any extra privileges can and will be exploited by attackers who identify an SQL injection.

This graphic ([source](https://learn.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver16)) highlights the built-in database roles in MSSQL. The public role is the default role and anything else is extra (although the roles db\_denydatareader and db\_denydatawriter actually take away privileges). 

#### Disable Dangerous Functions

You may want to disable dangerous functions for users who do not need them. For example, attackers can use xp\_dirtree to leak NetNTLM hashes, and it's likely your website doesn't use this function, so you may want to disable it for the specific user your website uses to query the database.

For example, to revoke execution privileges on xp\_dirtree for all users with the public role, we would run this command:

Code: sql

REVOKE EXECUTE ON xp\_dirtree TO public

Note: It is possible to completely disable functions like xp\_dirtree, but this is not something you'd want to do, as the server itself uses this function.