**Prevention of SQL injection.**

The idea is very simple - the query and the data are sent to the database server *separately*.

That's all.

The root of the SQL injection problem is **mixing of the code and the data.**

In fact, our SQL query is **a legitimate program**. And we are creating such a program dynamically, by adding some data on the fly. Thus, this data may interfere with the *program code* and even alter it, as every SQL injection example shows it (all examples in PHP/Mysql):

$expected\_data = 1;

$query = "SELECT \* FROM users where id=$expected\_data";

will produce a regular query

SELECT \* FROM users where id=1

while this code

$spoiled\_data = "1; DROP TABLE users;"

$query = "SELECT \* FROM users where id=$spoiled\_data";

will produce a malicious sequence

SELECT \* FROM users where id=1; DROP TABLE users;

It works because we are adding the data directly to the program body and it become a part of the program, so the data may alter the program and depending on the data passed, we will have either a regular output or a table users deleted.

While **in case of prepared statements we don't alter our program, it remains intact**

That's the point.

We are sending a *program* to the server first

$db->prepare("SELECT \* FROM users where id=?");

where the data is substituted by some *variable* called a parameter or a placeholder.

Note that the very same query being sent to the server, without any data in it! And then we're sending the data with the *second* request, essentially separated from the query itself:

$db->execute($data);

so, it can't alter our program and do any harm.

--------------------------------------------------------------------------------------------------

In [SQL Server](http://en.wikipedia.org/wiki/Microsoft_SQL_Server), using a prepared statement is definitely injection-proof because the input parameters don't form the query. It means that the executed query is not a dynamic query.

Example of an SQL injection vulnerable statement.

string sqlquery = "select \* from table where username='" + inputusername +"' and password='" + pass + "'";

Now if the value in the inputusername variable is something like a' or 1=1 --, this query now becomes:

select \* from table where username='a' or 1=1 -- and password=asda

And the rest is commented after --, so it never gets executed and bypassed as using the prepared statement example as below.

Sqlcommand command = new sqlcommand("select \* from table where username = @userinput and password=@pass");

command.Parameters.Add(new SqlParameter("@userinput", 100));

command.Parameters.Add(new SqlParameter("@pass", 100));

command.prepare();

So in effect you cannot send another parameter in, thus avoiding SQL injection...