Building Dynamic SQL Statements the Right Way

The following examples of implementing common use cases of dynamic SQL statements in OutSystems can help you prevent SQL injection vulnerabilities.

Also, check the SQL Injection Warning page for more information on the warnings you might get when the OutSystems platform detects a known bad practice that might lead to vulnerabilities.

Switching a filter condition on or off

Use an extra variable that switches a condition on/off without using parameters with the Expand Inline property enabled.

For example, if you wanted to provide an optional filter by surname, you could define your SQL query in the following way:

```
SELECT {Users}.[Username], {Users}.[Surname], {Users}.[Firstname]
FROM {Users}
WHERE {Users}.[IsActive] = 1
AND (@filterBySurname = 0 OR {Users}.[Surname] LIKE '%' + @surnameFilter +'%')
```

The Query Parameters `filterBySurname` and `surnameFilter` — configured with Boolean and Text data type, respectively — would have the following values:

```
filterBySurname = surnameFilter <> ""
surnameFilter = surnameFilter
```

This way you could use this SQL statement to list the username of all users or of users having a specific surname while avoiding having the Expand Inline property enabled and using the EncodeSql function.
Dynamically building a "WHERE ... IN (...)" clause

It's not possible to use a prepared statement for the values in a WHERE <column> IN (@valuelist) clause because you can't replace a query parameter (valuelist) with an array of values. Therefore, in this case, you must enable the Expand Inline property for the valuelist Query Parameter.

To properly build the values for the "IN" clause you should always use one of BuildSafe_InClauseIntegerList and BuildSafe_InClauseTextList functions available in the Sanitization extension.

Example 1:

```sql
SELECT {Users}.[Username], {Users}.[Surname], {Users}.[Firstname]
FROM {Users}
WHERE {Users}.[IsActive] = 1 AND {Users}.[Id] IN (@idlist)
```

In this example, the idlist query parameter is built using the BuildSafe_InClauseIntegerList function available in the Sanitization extension.

This example assumes that you have filled in a List of "Integer" Records — where "Integer" is a structure defined in the Sanitization extension with a single attribute for holding a long integer value — with the user identifiers for the "IN (@idlist)" filter. For example:

```plaintext
userids[0] = 4
userids[1] = 45
userids[2] = 76
```

Define the value of the idlist SQL Query Parameter using the BuildSafe_InClauseIntegerList function to build the content of the IN clause:

```sql
idlist = BuildSafe_InClauseIntegerList(userids)
```

idlist will contain the text "4,45,76".

Example 2:

```sql
SELECT {Users}.[Username], {Users}.[Surname], {Users}.[Firstname]
FROM {Users}
WHERE {Users}.[IsActive] = 1 AND {Users}.[Surname] IN (@namelist)
```

In this example, the namelist query parameter is built using the BuildSafe_InClauseTextList function available in the Sanitization extension.
This example assumes that you have filled in a List of "Text" Records — where "Text" is a structure defined in the Sanitization extension with a single attribute for holding a text value — with the surnames for the "IN (@namelist)" filter. For example:

```java
surnames[0] = "Smith"
surnames[1] = "Johnson"
surnames[2] = "Martinez"
```

Define the value of the `namelist` SQL Query Parameter using the `BuildSafe_InClauseTextList` function to build the IN clause:

```java
namelist = BuildSafe_InClauseTextList(surnames)
```

`namelist` will contain the text "'Smith', 'Johnson', 'Martinez'".

### Implementing custom sort orders in SQL queries

Query parameters in prepared statements can only be used for data replacements; they can't be used as parameters for specifying table names, table fields, operators, or SQL syntax like "ORDER BY" clauses. Therefore, to customize the sort order of the results returned by a SQL query you need to enable the `Expand Inline` property for the parameter that defines the custom sort order.

You must not use values provided by end users as part of SQL statements — this is also valid for the sort order of an SQL statement. The `EncodeSql` built-in won't protect you in this case since it's designed to encode string literals and not parts of a SQL statement.

Learn how to implement custom sort orders in your SQL statement in [How to enable dynamic sorting in a table fed by a SQL query](https://success.outsystems.com/Documentation/Best_Practices/Development/Building_Dynamic_SQL_Statements_the_Right Way). If you must provide complex sorting abilities in your application provided by end users, you should provide them with a UI where they can select their desired sorting options without having to enter any column/attribute names. All column names and sort order should be determined by your application from the options selected by end users.