

# Spring Boot web application with JDBC – Delete Product

## Overview

This tutorial will show how to delete existing products from the database. The process is similar to update but the form is not required. The steps are:

1. Select the product to be deleted from the product list
  - a. Pass the product id as a parameter, e.g. <http://localhost:8080/deleteProduct/?id=1>
2. Delete the product by id from the database.
3. Re-load the products list.

## 1. Show delete links in the product list

The goal is to add an delete button for each link. Clicking the button should call and pass the product id to **deleteProduct()** in the controller. The result should look like this:

| Id | Name                       | Description            | Stock | Price     |                      |                        |
|----|----------------------------|------------------------|-------|-----------|----------------------|------------------------|
| 1  | Kettle                     | Steel Electric Kettle  | 97    | € 35.60   | <a href="#">edit</a> | <a href="#">delete</a> |
| 2  | Fridge freezer             | Fridge + freezer large | 45    | € 799.00  | <a href="#">edit</a> | <a href="#">delete</a> |
| 3  | Microsoft Surface Laptop 2 | 8GB ram, 512GB ssd     | 5     | € 1299.00 | <a href="#">edit</a> | <a href="#">delete</a> |

Add a new table column (**<td>** element) to the product row for the delete button and link:

```
<td><a th:href="@{/updateProduct/?id=' + ${product.ProductId}}'" class="btn-sm btn-danger" role="button">edit</a></td>
<td><a th:href="@{/deleteProduct/?id=' + ${product.ProductId}}'" onclick="return confirmDelete();" class="btn-sm btn-danger" role="button">delete</span></a></td>
</tr>
</table> <!-- End table -->
```

Text version for copying:

```
<td><a th:href="@{/deleteProduct/?id=' + ${product.ProductId}}'" onclick="return confirmDelete();"
class="btn-sm btn-danger" role="button">delete</span></a></td>
```

## Confirming delete

It would not be a good idea to delete without first confirming with the user – in case the button was clicked accidentally. JavaScript can be used to confirm the action before calling the delete link. Note the **onclick even handler** in the link:

```
onclick="return confirmDelete();"

```

This calls the following JavaScript function which should be added to the end of the page body:

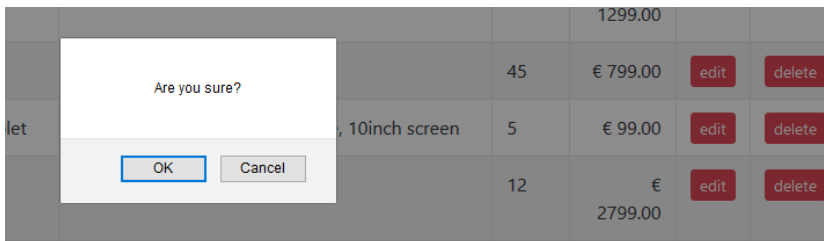
```

</div> <!-- End container -->
<script>
    // JavaScript function returns true if user clicks yes, otherwise, false
    // true - proceed (open link)
    // false - cancel
    function confirmDelete() {
        return confirm('Are you sure?');
    }
</script>

<!--*/ <th:block th:include="fragments/footer :: footer"></th:block> /*/-->
</body>
</html>

```

Now whenever a delete button is clicked the user will be asked to confirm:



Ok will return true, allowing the link to be loaded, cancel returns false which stops `/deleteProduct/?id=1` from being called.

## 2. Product DAO update method

Add a method to **ProductDao** and its implementation to perform the update

First **ProductDao** (the interface)

The create method accepts an existing product object parameter and returns the number of rows affected by the update (usually 1).

```

ProductDao.java x ProductDaoImpl.java ApplicationContro
22 public Product create(final Product product);
23
24 // update an exiting product - return the number
25 public int update(final Product product);
26
27 // Delete product with matching id
28 public int delete(int id);
29
30

```

The product implementation, **ProductDaoImpl.java**

The SQL statement required for delete. This is an SQL statement with value placeholders (?)

```
private final String DELETE_SQL_BY_ID = "DELETE FROM dbo.Product WHERE ProductId = ?";
```

The **delete** method which will fill in the values and execute the query (see code comments)

```
// Delete a product by id
public int delete(int id) {

    // Use the delete sql, setting the id paramater
    // This method returns the number of rows affected
    return jdbcTemplate.update(DELETE_SQL_BY_ID, new Object[] {id});
}
```

### 3. A Controller Method to execute the delete

This method will call the delete and redirect back to the products page. Also use **RedirectAttributes** (flash attribute) to confirm delete or report error.

1. **RedirectAttributes** allows a confirmation message to be sent to the next page via redirect.
  - a. Useful for confirming actions
  - b. Requires **import org.springframework.web.servlet.mvc.support.RedirectAttributes;**
  - c. (1 in code) Inject **RedirectAttributes** **redirAttrs** as a parameter to the **deleteProduct** method
2. (2 in code) Add a flash attribute named **message** (value shows rows successfully deleted)
3. (3,4 in code) Add a flash attribute named **error** (value shows error condition – delete failed)

```
226 // Delete Product
227 @RequestMapping(value = "/deleteProduct", method = RequestMethod.GET)
228 public String deleteProduct(@RequestParam(name = "id", required = true) String pId, RedirectAttributes redirAttrs) {
229
230     // Initialise id (default value used to get all products)
231     int id = 0;
232
233     // The parameter may be not be valid - which could crash the application
234     // This tries to parse the string converting it to an int
235     // If successful id will be assigned the int value
236     // Otherwise - catch any exception
237     // If it fails (i.e an exception occurs) id value will not be changed (from 0).
238     try {
239         id = Integer.parseInt(pId);
240     }
241     catch(NumberFormatException e) {
242         System.out.println("Bad input for id: " + e);
243     }
244
245     // If id value is greater than 0 then delete - otherwise error
246     if (id != 0) {
247         int rows = productData.delete(id);
248
249         // Verify that something was deleted (rows affected > 1)
250         if (rows >= 1) {
251             // Set a flash message confirming the delete
252             redirAttrs.addFlashAttribute("message", rows + " rows deleted");
253         }
254         else {
255             // Nothing deleted - set error flash message
256             redirAttrs.addFlashAttribute("error", "Error: Product delete failed");
257         }
258     }
259     else {
260         // can't delete id = 0 - show error
261         redirAttrs.addFlashAttribute("error", "Nothing to delete");
262     }
263
264     // Return to products page
265     return "redirect:/products";
266 }
267 }
```

Annotations and Callouts:

- 1. **RedirectAttributes redirAttrs** (indicated by a red arrow)
- Parse the parameter value and handle exceptions (callout for the try-catch block)
- id > 0, proceed with delete (callout for the if (id != 0) block)
- Set message if delete succeeded, otherwise and error message (callout for the if (rows >= 1) block)
- id == 0, nothing to delete (callout for the else block)

Copy code:

```
// Delete Product
@RequestMapping(value = "/deleteProduct", method = RequestMethod.GET)
public String deleteProduct(@RequestParam(name = "id", required = true) String pId, RedirectAttributes redirAttrs) {

    // Initialise id (default value used to get all products)
    int id = 0;

    // The parameter may be not be valid - which could crash the application
    // This trys to parse the string converting it to an int
    // If successfull id will be assigned the cat value
    // Otherwise - catch any exception
    // If it fails (i.e an exception occurs) id value will not be changed (from 0).
    try {
        id = Integer.parseInt(pId);
    }
    catch(NumberFormatException e) {
        System.out.println("Bad input for id: " + e);
    }
    // If id value is greater than 0 then delete - otherwise error
    if (id != 0) {
        int rows = productData.delete(id);

        // Verify that something was deleted (rows affected > 1)
        if (rows >= 1) {
            // Set a flash message confirming the delete
            redirAttrs.addFlashAttribute("message", rows + " rows deleted");
        }
        else {
            // Nothing deleted - set error flash message
            redirAttrs.addFlashAttribute("error", "Error: Product delete failed");
        }
    }
    else {
        // can't delete id = 0 - show error
        redirAttrs.addFlashAttribute("error", "Nothing to delete");
    }

    // Return to products page
    return "redirect:/products";
}
```

#### 4. Display the results (after delete)

After delete, the flash message should be displayed in the products page. These are added in **products.html** – if they exist (i.e. they were set in the controller before redirecting to this page).

```
27 <!-- Column 2 - Product List -->
28 <div class="col-sm-10">
29
30 <h3>Products</h3>
31
32 <th:block th:if="${message} != null">
33     <h6 class="alert alert-success mb-2" th:text="${message}"></h6>
34 </th:block>
35
36 <th:block th:if="${error} != null">
37     <h6 class="alert alert-danger mb-2" th:text="${error}"></h6>
38 </th:block>
39
40
41 <!-- Add table if the products list contains products-->
42 <!-- Bootstrap: https://getbootstrap.com/docs/4.0/content/tables/-->
43 <table class="table table-bordered table-striped" th:if="${not #lists.isEmpty(products)}">
44     <!-- Table header row -->
45     <tr>
```

The result after deleting a product:

### Products

1 rows deleted

| Id | Name              | Description            | Stock | Price    |      |        |
|----|-------------------|------------------------|-------|----------|------|--------|
| 1  | Kettle            | Steel Electric Kettle  | 97    | € 35.60  | edit | delete |
| 2  | Fridge freezer    | Fridge + freezer large | 45    | € 799.00 | edit | delete |
| 3  | Microsoft Surface | RGR ram 512GB ssd      | 5     | €        | edit | delete |