11_02 Implementing <cfqueryparam />

In this exercise, you will update the queries of your application to make them use `<cfqueryparam />` for each parameters. Remember that this is an important security feature that will help you protect your data against the SQL injection attack. It will also increase the performance of your application.

Update the necessary queries to use <cfqueryparam />

Since the '`newsService.cfc`' component is still open from the video (if not, just reopen it), you will start this exercise by updating the '`getNewsForYear()`' method of that component.

1. In the '`newsService.cfc`' component, locate the '`getNewsForYear()`' method.
2. Update the `WHERE` clause to pass the '`arguments.year`' parameter using a `<cfqueryparam />` tag.
3. Use the '`cf_sql_integer`' data type.

The final code is.

```
<cfquery  name="rsNewsOfYear">
  SELECT FLD_NEWSTITLE, FLD_NEWSCREATIONDATE, FLD_NEWSID
  FROM TBL_NEWS
  WHERE year(FLD_NEWSCREATIONDATE) = <cfqueryparam
      value="#arguments.year#" cfsqltype="cf_sql_integer" />
  ORDER BY FLD_NEWSCREATIONDATE DESC
</cfquery>
```

Next, you will update the query that is in the '`authenticationService.cfc`'

4. Open the '`components/authenticationService.cfc`'.
5. In the '`doLogin()`' method, locate the `<cfquery>` that retrieves the user from the database.
6. Use a `<cfqueryparam />` tag for both the '`fld_userEmail`' and the '`fld_userPassword`' parameters. Use '`cf_sql_varchar`' data type for both parameters.

Your code should be

```
<!---Get the user data from the database--->
<cfquery name="rsLoginUser">
  SELECT TBL_USERS.FLD_USERFIRSTNAME, TBL_USERS.FLD_USERLASTNAME,
    TBL_USERS.FLD_USERID, TBL_USERS.FLD_USEREMAIL,
    TBL_USERS.FLD_USERPASSWORD, TBL_ROLES.FLD_ROLENAME
  FROM TBL_USERS INNER JOIN TBL_ROLES ON TBL_USERS.FLD_USERROLE =
    TBL_ROLES.FLD_ROLEID
  WHERE FLD_USEREMAIL = <cfqueryparam
    value="#arguments.userEmail#" cfsqltype="cf_sql_varchar" />
    AND FLD_USERPASSWORD = <cfqueryparam
    value="#arguments.userPassword#" cfsqltype="cf_sql_varchar" />
    AND FLD_USERISACTIVE = 1
</cfquery>
```

Notice the value of the '`cfsqltype`' attribute. The term '`varchar`' is, more or less, a synonym of '`string`', used in the database world. In this example, both the eMail and the password of the user are strings of text.

7. Save and close the '`authenticationService.cfc`' component.
8. Open the '`eventService.cfc`' component.
9. Locate the In the '`getEventByID()`'.
10. Update the `<cfquery />` of the '`getEventByID()`' method so it uses `<cfqueryparam />`. Can you define the proper data type for this parameter?

After the update, the code is

```
<cfquery name="rsSingleEvent">
  SELECT FLD_EVENTID, FLD_EVENTNAME, FLD_EVENTDATETIME,
    FLD_EVENTLOCATION, FLD_EVENTVENUE, FLD_EVENTDESCRIPTION
  FROM TBL_EVENTS
  WHERE FLD_EVENTID = <cfqueryparam value="#arguments.eventID#"
    cfsqltype="cf_sql_integer" />
</cfquery>
```

11. Save and close the '`eventService.cfc`' component.
12. Open the '`userService.cfc`' component.

In this last component, there is a little more work to do.

13. First, you will update the '`getUserByID()`' method so the '`fld_userID`' parameter is passed to the database using a `<cfqueryparam />` tag.

After the update, the code is.

```
<cfquery name="rsSingleUser" >
    SELECT FLD_USERID, FLD_USERFIRSTNAME, FLD_USERLASTNAME,
        FLD_USERCOMMENT, FLD_USEREMAIL, FLD_USERROLE,
        FLD_USERPASSWORD, FLD_USERINSTRUMENT, FLD_USERAPPROVED,
        FLD_USERISACTIVE
    FROM TBL_USERS
    WHERE FLD_USERID = <cfqueryparam value="#arguments.userID#"
    cfsqltype="cf_sql_integer" />
</cfquery>
```

14. Now, you will modify the '`updateUser()`' method.

This method is called from the '`profile.cfm`' page when a user wants to edit his profile. The main difference between this query and the previous ones you have updated is that this query uses seven parameters. You need one `<cfqueryparam />` tag for each of the even parameters of the query. Refer to the following table for the proper data type to use.

Parameter	Data type
arguments.userFirstName	cf_sql_varchar
arguments.userLastName	cf_sql_varchar
arguments.userEmail	cf_sql_varchar
arguments.userPassword	cf_sql_varchar
arguments.userInstrument	cf_sql_integer
arguments.userComment	cf_sql_longvarchar
arguments.userID	cf_sql_integer

After the update, the code should be

```
<cfquery >
  UPDATE TBL_USERS
  SET
  FLD_USERFIRSTNAME = <cfqueryparam value="#form.fld_userFirstName#"
      cfsqltype="cf_sql_varchar" />,
  FLD_USERLASTNAME = <cfqueryparam value="#form.fld_userLastName#"
      cfsqltype="cf_sql_varchar" />,
  FLD_USEREMAIL = <cfqueryparam value="#form.fld_userEmail#"
      cfsqltype="cf_sql_varchar" />,
  FLD_USERPASSWORD = <cfqueryparam value="#form.fld_userPassword#"
      cfsqltype="cf_sql_varchar" />,
  FLD_USERINSTRUMENT = <cfqueryparam value="#form.fld_userInstrument#"
      cfsqltype="cf_sql_integer" />,
  FLD_USERCOMMENT = <cfqueryparam value="#form.fld_userComment#"
      cfsqltype="cf_sql_longvarchar" />
  WHERE FLD_USERID = <cfqueryparam value="#form.fld_userID#"
      cfsqltype="cf_sql_integer" />
</cfquery>
```

In the above code, there are a few things to notice :

- Every parameter uses the '`cf_sql_varchar`' type, except the '`fld_userInstrument`' that uses the '`cf_sql_integer`' type and the '`fld_userComment`' that uses the '`cf_sql_longvarcher`' type.
- There is a comma (,) at the end of every `<cfqueryparam />` tag (except for the last one) in order to separate the parameters. This comma is required by the SQL syntax, not by ColdFusion.
- The quotes (' ') that used to surround the values are gone! It is part of `<cfqueryparam />`'s job to take care of them!

Finally, you will apply the same update to the query that is in the '`addUser()`' method. The data types are the same as for the '`updateUser()`' method. The data type for the additional parameters can be found in the table below.

Parameter	Data type
arguments.userRole	cf_sql_integer
arguments.userIsApproved	cf_sql_integer
arguments.userIsActive	cf_sql_integer

Your updated code should be

```
<cfquery >
  INSERT INTO TBL_USERS
  (FLD_USERFIRSTNAME, FLD_USERLASTNAME, FLD_USEREMAIL, FLD_USERPASSWORD,
  FLD_USERROLE, FLD_USERINSTRUMENT, FLD_USERCOMMENT, FLD_USERAPPROVED,
  FLD_USERISACTIVE)
  VALUES
  (
    <cfqueryparam value="#arguments.userFirstName#"cfsqltype="cf_sql_varchar" />,
    <cfqueryparam value="#arguments.userLastName#" cfsqltype="cf_sql_varchar" />,
    <cfqueryparam value="#arguments.userEmail#" cfsqltype="cf_sql_varchar" />,
    <cfqueryparam value="#arguments.userPassword#" cfsqltype="cf_sql_varchar" />,
    <cfqueryparam value="#arguments.userRole#" cfsqltype="cf_sql_integer" />,
    <cfqueryparam value="#arguments.userInstrument#" cfsqltype="cf_sql_integer" />,
    <cfqueryparam value="#arguments.userComment#" cfsqltype="cf_sql_longvarchar" />,
    <cfqueryparam value="#arguments.userIsApproved#" cfsqltype="cf_sql_integer" />,
    <cfqueryparam value="#arguments.userIsActive#"cfsqltype="cf_sql_integer" />
  )
</cfquery>
```

WARNING : double check, or even triple check, your typing. Did you use the right types? Did you place the right commas? Did you close the parenthesis?
To waive out the most common issues, try to save and run the component. If you can

access the automatic documentation, you know the syntax is correct and you can move on.

Should you use <cfqueryparam /> for every query?

And the answer is : *probably not*. Take the '`history.cfm`' page for instance. It calls the '`getPageByID()`' method of the '`pageService.cfc`' component, but the argument passed to the method does not come from some kind of user input variable, like the '`newsID`' came from the URL scope. Instead, the value of the '`pageID`' argument is *hard coded*, which means, written directly in the code. In this case, the value is 2.

```
<!---Get page content--->
<cfset variables.rsPage = application.pageService.getPageByID(2) />
```

So there is no way for any attacker to tamper with that value and inject some SQL in the database. In other words, the query in the '`getPageByID()`' method does not absolutely need to be updated with a `<cfqueryparam />` tag.
That being said, it is considered best practice to use `<cfqueryparam />` for every dynamic parameter of every query. Personally, when I write an application, I never ask myself, '*Do I have to use* `<cfqueryparam />` *for this particular query or not?*' I always use `<cfqueryparam />` in any case.

Time for Some Testing

Make sure every file is saved, and then, run the '`index.cfm`' page. Add the '`restartApp`' parameter to the URL and reload the page to cache the new version of the components in the application scope. Once done, these are the tests to be performed.

1. Browser every page of the site.
2. Submit a musician on the '`comePlayWithUs.cfm`' page.
3. Log on to the site as an administrator.
4. Access the administrator's profile and change some data (If you change the password, make sure you remember it for the rest of the book).

Confirm every feature of the site still works before moving on to the next step.