Lesson 23: Executing Other SQL Queries From the DSL

Created by Jacques Marais, last modified on Jul 22, 2019

> *As a System Admin I would like to resolve all current service tickets or mark all current service tickets as spam.*

Lesson Outcomes

By the end of this lesson you should know how to make use of the `sql:execute` built-in function to execute PostgreSQL queries that are not select queries.

App Use Case Scenario

The support ticket menu item for the `"System Admin"` user role presents a data wall of service tickets. Each ticket can be individually resolved or marked as spam. The feature described in this lesson allows the user to resolve all current service tickets or mark all current service tickets as spam. A typical way to achieve this in a DSL presenter is to query the database for service tickets using a selector and then looping through the resulting service ticket collection one by one to resolve or mark as spam. A much simpler implementation is to make use of a single update query. This is demonstrated in the remainder of this lesson.

New & Modified App Files

`./web-app/views/user_management/Support.vxml`

`./web-app/presenters/user_management/Support.mez`

`./web-app/lang/en.lang`

View & Presenter Additions

The only additions needed for this lesson are two new buttons on the `Support` view and two new action functions for these buttons on the `Support` unit in the **`Support.mez`** presenter file. The view additions and skeleton code for our newly added presenter functions are shown below:

```
1  <button label="button.resolve_all_tickets" action="resc
2  <button label="button.mark_all_tickets_as_spam" action=
```

```
1
2  DSL_VIEWS resolveAllTickets() {
3      .
```

```
    .
    .
    return DSL_VIEWS.Support;
}

DSL_VIEWS markAllTicketsAsSpam() {
    .
    .
    .
    return DSL_VIEWS.Support;
}
```

Query Execution

To update the service tickets we execute the update queries using the `sql:execute` built-in function. Note that the `sql:query` function is specifically for select queries while the `sql:execute` built-in function is specifically for insert, update and delete queries. Only update is demonstrated in this lesson but the others work similarly.

Once again the function requires the query as first argument followed by query parameters values where applicable. As mentioned in Lesson 22, these parameters are denoted by **?** in the query itself and replaced at runtime with the specified values.

The `sql:execute` built-in function returns an integer representing the number of records that were modified. The syntax requires the result of the function to be assigned to a variable and does not allow execution of the `sql:execute` function without this assignment.

Below are the action functions responsible for query execution in their entirety:

```
// Resolve tickets that have not been resolved and not been
DSL_VIEWS resolveAllTickets() {
    string query = "update supportticket set resolved=true w
    int updates = sql:execute(query);
    Mez:alert("alert.tickets_resolved");
    return DSL_VIEWS.Support;
}

// Mark tickets as spam that have not been marked as such an
DSL_VIEWS markAllTicketsAsSpam() {
    string query = "update supportticket set spam=true where
    int updates = sql:execute(query);
    Mez:alert("alert.tickets_marked_as_spam");
    return DSL_VIEWS.Support;
}
```

A Note on Temp Tables

Avoid SQL using

It's important to note that SQL making use of temporary tables are, in most cases, not supported in Helium and in general the use of temporary tables in DSL apps is discouraged. Please see additional notes on this here.

temporary tables in DSL apps.

Lesson Source Code

Lesson 23.zip

No labels