Introduction to Running SQL Queries
Against the DAACS PostgreSQL backend using Navicat

2: JOINS and GROUP BY

This document describes how to use Navicat to build SQL queries that will return a list of ceramic counts, categorized by ware type, in their contexts for a particular site. It then covers how to add additional information on ceramics to the query results (e.g. Genre and Stylistic Elements). Finally, it shows how to aggregate ceramic counts for features within a site.

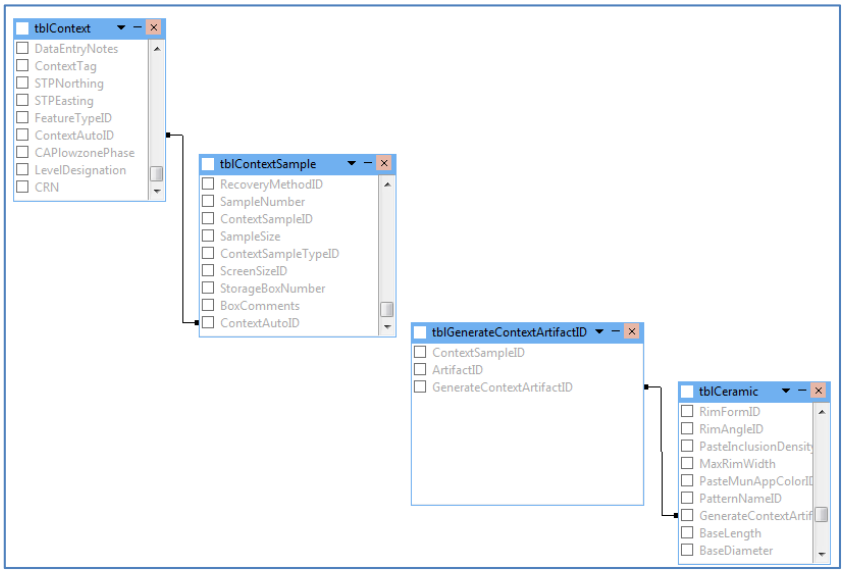
We will use the ceramic table (tblCeramic) as a learning example. The same query logic can be applied to the other artifact tables.

1. Linking the Context Table to Artifact Tables

Reporting on artifacts in their contexts requires linking context data with artifact data. In DAACS this is achieved by joining the following tables:

tblContext
tblContextSample
tblGenerateContextArtifactID
tblCeramic

To see how this works in practice, launch Navicat and double click on the *daacs-production* database to establish a connection. Choose *Query*, *New Query*. *<Click>* the *Query Builder* tab.



Locate the four tables listed above and select drag each of them into the upper left pane. The result will look like this. Note that *tblContext* is linked to *tblContextSample* on the *ContextAutoID* field. *tblGenerateContextArtifactID* is linked to *tblCeramic* on the *generateContextArtifactID* field.

We need to add a link between *tblContextSample* and *tblGenerateContextArifactID*. To do this, select the *ContextSampleID* field in the former table and drag it onto the same field in the latter table. You will see a link appear between them.

This sequence of linkages -- or *JOINS* -- between *tblContext* and the various artifact tables is fundamental to most reporting on artifacts in DAACS.

2. Getting a list of Ware-Type Counts for all Contexts at Single Site.

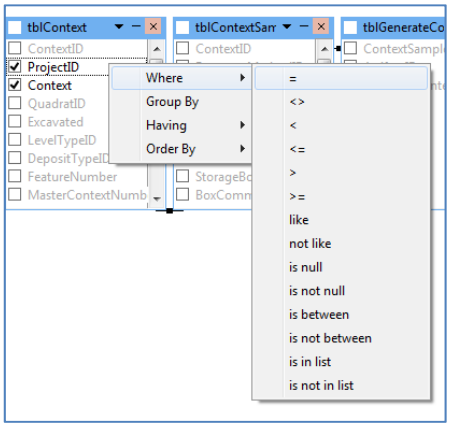
We have already completed the first step: linking *tblConext* to *tblCeramic* via the two intermediate tables.

The tasks that remain are:

-Use the text authority terms for ware type, not the numeric codes that appear in the *WareID* field in *tblCeramic.*
-Tell SQL which fields we want listed in the final output and the order in which to sort them -- say by context.
-Choose the site. We'll use the *projectID* field to do this. Let's use Site 8 at Monticello. As we can see from the results of our first queries with *tblProject*, the *ProjectID* for Site 8 is *108*.

We can accomplish the first of these tasks by linking the authority table that contains the ware-type names. The table is called *tblCeramicWare.* Drag it into the *Query Builder* window. Navicat will automatically make a link between the *WareID* field in *tblCeramic* and the same field in *tblCeramicWare.*

Next we need to tell SQL which fields we want the query to return. Do that by checking the boxes next to the field names. Let's check two field in TblContext: ProjectID and Context, two fields in tblCeramic: ArtifactID and Quantity, and one field in tblCeramicWare: Ware.



OK. Only two more steps to go!

First, select the *projectID* field in *tblContext*, *<right click>* and choose "=". This is shown in the box to the right. Then type *'108'* into the Edit Box that appears. '108' is the *projectID* for Site 8 at Monticello.

Second, select the *Context* field in *tblContext, <right click>* and choose *Order By > Asc.* This will sort the records in ascending order.

Now click *Run.* The query should return 5521 records. Running the query will place you in the *Query Editor* window, where you will see the SQL code that your dragging and dropping have generated. The code should look like this:

```
SELECT
"public"."tblContext"."ProjectID",
"public"."tblCeramic"."ArtifactID",
"public"."tblCeramic"."Quantity",
"public"."tblContext"."Context",
"public"."tblCeramicWare"."Ware"
FROM
"public"."tblContext"
INNER JOIN "public"."tblContextSample" ON "public"."tblContextSample"."ContextAutoID" =
"public"."tblContext"."ContextAutoID"
```

```
INNER JOIN "public"."tblGenerateContextArtifactID" ON
"public"."tblContextSample"."ContextSampleID" =
"public"."tblGenerateContextArtifactID"."ContextSampleID"
INNER JOIN "public"."tblCeramic" ON "public"."tblCeramic"."GenerateContextArtifactID" =
"public"."tblGenerateContextArtifactID"."GenerateContextArtifactID"
INNER JOIN "public"."tblCeramicWare" ON "public"."tblCeramic"."WareID" =
"public"."tblCeramicWare"."WareID"
WHERE
"public"."tblContext"."ProjectID" = '108'
ORDER BY
"public"."tblContext"."Context" ASC
```

Now that we have done this hard work, don't forget to save the query!

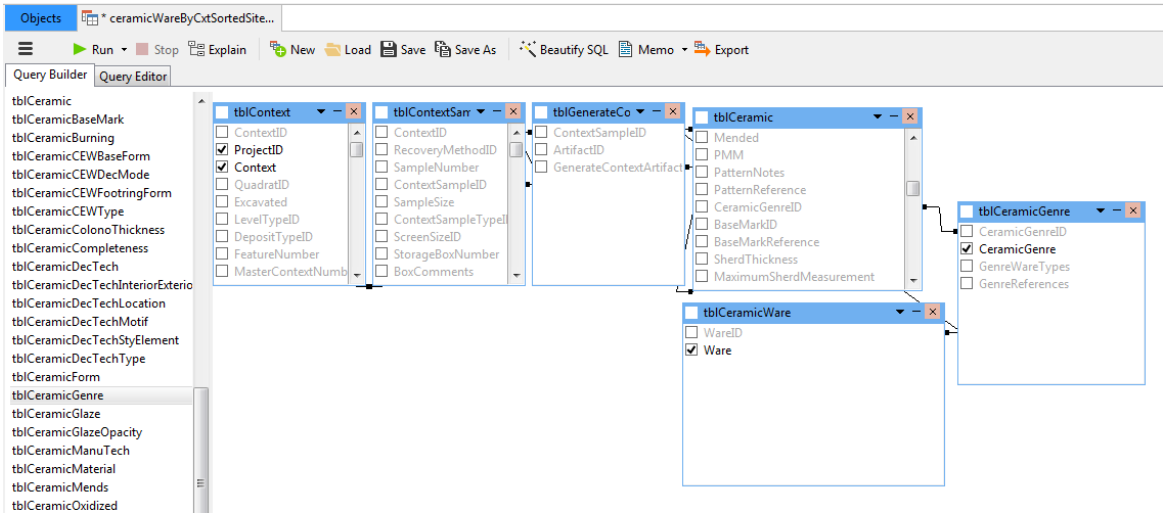
3. Adding Details on Ceramics

Obviously there is a LOT more information in DAACS on ceramics than just ware type! We can explore two additional kinds of information.

We'll start simple. There is an addition field called *Genre* that provides a finer-grained breakdown based selected aspects of decoration and manufacturing technique. To bring this into the query, we need to link the relevant authority table (*tblCeramicGenre*) and check the authority term (*Genre*).

Go back to the *Query Builder* tab that contains the query that we just ran. Scroll the left pane that contains the list of tables to find *tblCeramicGenre.* Drag it into the upper right pane of the *Query Builder*. Find the *CeramicGenreID* field in *tblCeramic* and *tblCeramicGenre.* Click on one of the fields and drag it to the other to establish a link between them. This will generate the following line of code in the Query Editor:

```
INNER JOIN "public"."tblCeramicGenre" ON "public"."tblCeramicGenre"."CeramicGenreID" =
"public"."tblCeramic"."CeramicGenreID"
```



Finally, don't forget to tell SQL that you want to retrieve the *CeramicGenre* text field. Do this by clicking the box next to the field name in *tbl.CeramicGenre*.

Run the query and you will get back the same 5521 records you got in the previous query. But these records now include the values for the Genre field.

4. Adding More Details on Ceramics: Stylistic Elements and the LEFT JOIN

In DAACS the data on individual stylistic elements are stored in *tblCeramicDecTech.* Browse the table in Navicat. We'll be using two fields in this table in our next query. *CeramicDecTechTypeID* contains information on the decorative technique used to make a particular stylistic element on a sherd. *CeramicDecTechStyElemID* codes the particular element used. The relevant authority terms are to be found -- no surprise -- in the tables called *tblCeramicDecTechType* and *tblCeramicDecTechStyElement.*

Start by returning to the *Query Builder* tab for the genre query that we just ran (see 3. above). Drag *tblCeramicDecTech* and the two authority tables in to the upper right pane of the *Query Builder*. Navicat will link the table automatically and add the following lines of code to the query

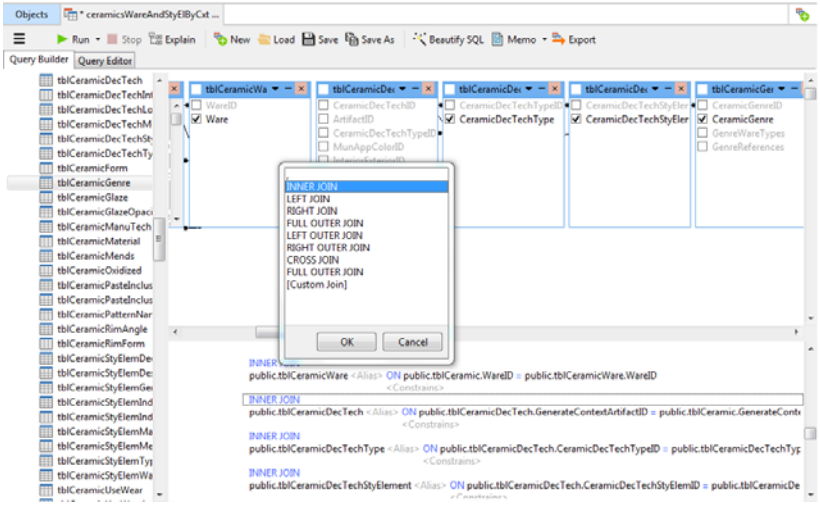
```
INNER JOIN "public"."tblCeramicDecTech" ON
"public"."tblCeramicDecTech"."GenerateContextArtifactID" =
"public"."tblCeramic"."GenerateContextArtifactID"
INNER JOIN "public"."tblCeramicDecTechStyElement" ON
"public"."tblCeramicDecTech"."CeramicDecTechStyElemID" =
"public"."tblCeramicDecTechStyElement"."CeramicDecTechStyElemID"
INNER JOIN "public"."tblCeramicDecTechType" ON
"public"."tblCeramicDecTech"."CeramicDecTechTypeID" =
"public"."tblCeramicDecTechType"."CeramicDecTechTypeID"
```

Don't forget to check the boxes next to the field names in the two authority tables, so that SQL includes them in the results.

Run the query. Examine the results carefully. Check the Artifact ID field. Notice that there are now multiple records for the same artifact, identified by ArtifactID. But despite this, there are only 1308 records in the dataset that SQL returned, fewer than the 5521 records we got in the previous query. The *Quantity* field is now a count of stylistic elements, NOT sherds.

What has happened? We have lost all the ceramics that are undecorated and therefore have no corresponding records in *tblCeramicDecTech.* We lost these records because we used INNER JOINS in the query. The INNER JOIN command only returns records in the table named in the FROM statement for which there are matching entries in the table named in the INNER JOIN statement.

To force SQL to return records for both decorated ceramics AND undecorated ceramics, we need to change the INNER JOINS for *tblCeramicDecTech* and its two related tables to LEFT JOINS. The LEFT JOIN statement tells SQL to include all the records in the table named in the FROM statement (the "left" table), regardless of whether they have matching records in the table named in the JOIN statement.



To change in INNER JOINS to LEFT JOINS in Navicat, start in the *Query Builder* tab. In the lower right pane, find the first instance of the INNER JOIN that needs to be changed – it's the line where you bring in *tblCeramicDecTech* for the first time. Click on the INNER JOIN statement and choose LEFT JOIN. Do the same for the next two JOIN statements which reference the two authority tables: *tblCeramicDecTechType* and *tblCeramicDecTechStyElement.*

The lines of code that you altered should now read:

```
LEFT JOIN "public"."tblCeramicDecTech" ON
"public"."tblCeramicDecTech"."GenerateContextArtifactID" =
"public"."tblCeramic"."GenerateContextArtifactID"
LEFT JOIN "public"."tblCeramicDecTechType" ON "public"."tblCeramicDecTech"."CeramicDecTechTypeID"
= "public"."tblCeramicDecTechType"."CeramicDecTechTypeID"
LEFT JOIN "public"."tblCeramicDecTechStyElement" ON
"public"."tblCeramicDecTech"."CeramicDecTechStyElemID" =
"public"."tblCeramicDecTechStyElement"."CeramicDecTechStyElemID"
```

Run the query again. It will now return 5942 records. There will be one record for each stylistic element on decorated shreds and one record for each undecorated sherd.

5. Counting Ceramics in Features using GROUP BY

Listing records for shreds and the stylistic elements on them is great. But so is producing summaries of sherd or element frequencies in contexts or larger units of aggregation. The next example shows how to do this for sherd counts. In this case the unit of aggregation is the archaeological feature, which may contain multiple contexts. We will build on our work so far and start with the ceramic ware-by-context query we did earlier (Section 2).

Since you saved that query, you will be able to load it into the Query Builder suing the *Load* button on the top menu. Once the query is loaded, we can edit it – we'll start by adding tables to it. Let's start with the table that contains feature data: *tblContextFeature.* Locate it in the left pane and drag it into the upper right pane of the *Query Builder*. In addition to the *FeatureNumber,* this table contains a field called *FeatureTypeID* that says what kind of feature we are dealing with. The associated authority terms are in *tblContextfeatureType.* So bring that in as well. Navicat should automatically link it to *tblContextFeature.* Check the *FeatureType* field to include it in the query.

We now need to think about linking the context and feature tables. We will use two fields: *ProjectID* and *FeatureNumber*. We need both because we cannot assume that feature numbers are unique across projects – in fact we know they are not! Select the *ProjectID* field from *tblContext* and drag it to the *ProjectID* field in *tblContextFeature*. Do the same with the *FeatureNumber* fields in the two tables. This will add two lines of code that joins these tables:

```
INNER JOIN "public"."tblContextFeature" ON "public"."tblContextFeature"."ProjectID" =
"public"."tblContext"."ProjectID" AND "public"."tblContextFeature"."FeatureNumber" =
"public"."tblContext"."FeatureNumber"
```

Running this code will return 1389 records.

Next we need to get SQL to add up the ceramic ware type counts within each feature. To do this requires two steps. First we tell SQL what field contains the counts to add up. Navigate to the lower right pane in the *Query Builder* – where the SQL code is displayed find the line in the SELECT statement that names the *Quantity* field. <Click> on the "<Func>" text string and choose "Sum". Then <click> on the "<Alias>" text string and type "TotalCount" into the editor box that appears. This will generate the following line of code in the SELECT statement:

```
Sum("public"."tblCeramic"."Quantity") AS "TotalCount",
```

Now we need to delete the lines in the SELECT statement that we will not need. Since we want total counts of ware types by feature numbers, we have no use for information on individual contexts or artifact IDs. To delete the two lines that mention the *ArtifactID* and *Context* fields in the SELECT statement, find them on the lower right pane of the Query Builder. Select them. Then <right click> and choose "delete". Since we just told SQL not to return data on the Context field, we also need to delete the statement that asks that the results be sorted on that field: the ORDER BY statement.

The next step is to tell SQL that we want it to sum the quantities for ware types by feature numbers. In other words, we need to specify the levels of aggregation. To do this, we will use a GROUP BY statement. We want total counts for wares by features by projects. So the *Ware,* the *FeatureNumber,* the *FeatureType* and the *ProjectID* fields will be used to form the groups within which the quantities will be added or summarized. We need to include *ProjectID* and *FeatureType* in the GROUP BY if we also want that field returned in the results. More generally, when you use a GROUP BY statement, all fields in the SELECT statement must either appear in the GROUP BY or as arguments to summary functions in the SELECT statement (in our case *SUM(Quantity)*). Finally, let's sort the results by FeatureType, Feature Number, and Ware.

Run the query. It should return 73 records. The final code will look like this:

```
SELECT
"public"."tblContext"."ProjectID",
Sum("public"."tblCeramic"."Quantity") AS "TotalCount",
"public"."tblCeramicWare"."Ware",
"public"."tblContextFeature"."FeatureNumber",
"public"."tblContextFeatureType"."FeatureType"
FROM
"public"."tblContext"
INNER JOIN "public"."tblContextSample" ON "public"."tblContextSample"."ContextAutoID" =
"public"."tblContext"."ContextAutoID"
INNER JOIN "public"."tblGenerateContextArtifactID" ON
"public"."tblContextSample"."ContextSampleID" =
"public"."tblGenerateContextArtifactID"."ContextSampleID"
INNER JOIN "public"."tblCeramic" ON "public"."tblCeramic"."GenerateContextArtifactID" =
"public"."tblGenerateContextArtifactID"."GenerateContextArtifactID"
INNER JOIN "public"."tblCeramicWare" ON "public"."tblCeramic"."WareID" =
"public"."tblCeramicWare"."WareID"
INNER JOIN "public"."tblContextFeature" ON "public"."tblContextFeature"."ProjectID" =
"public"."tblContext"."ProjectID" AND "public"."tblContextFeature"."FeatureNumber" =
"public"."tblContext"."FeatureNumber"
INNER JOIN "public"."tblContextFeatureType" ON "public"."tblContextFeature"."FeatureTypeID" =
"public"."tblContextFeatureType"."FeatureTypeID"
WHERE
"public"."tblContext"."ProjectID" = '108'
GROUP BY
"public"."tblContextFeature"."FeatureNumber",
"public"."tblCeramicWare"."Ware",
```

```
"public"."tblContext"."ProjectID",
"public"."tblContextFeatureType"."FeatureType"
ORDER BY
"public"."tblContextFeatureType"."FeatureType" ASC,
"public"."tblContextFeature"."FeatureNumber" ASC,
"public"."tblCeramicWare"."Ware" ASC
```

FDN
6/5/2014