SQL* Quick Guide with GRIN-Global

Revision Date
May 30, 2023

Author
Martin Reisinger

This document is a summary of an NPGS Question and Answer session where we focused on the genebank user who is not familiar with SQL basics. Explained here are the basics of running SQL queries in the Public Website and creating custom queries using GRIN-Global table and field names. Tips are also included for joining multiple tables.

Goals

1. Use the Public Website to run SQL statements
2. Review the basics of SQL coding
3. Determine how to locate GG table and column names
4. Determine how to create simple queries, accessing data from multiple tables

* "SQL" – Structured Query Language

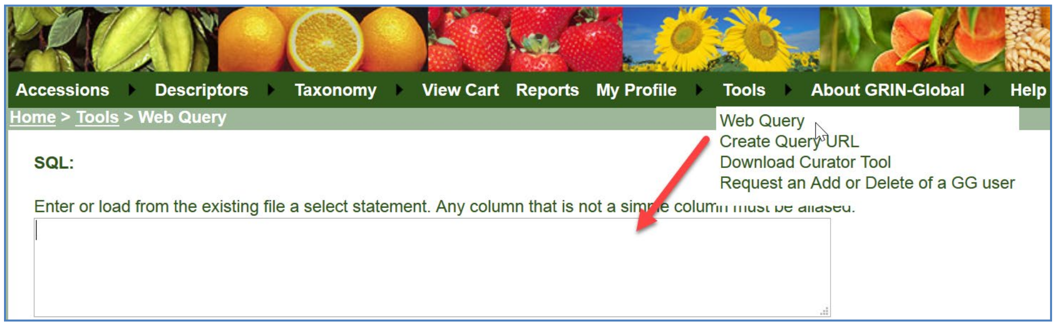
Tip

Refer to the excellent tutorial online if you want additional explanations to any of the SQL reserved words. See https://www.w3schools.com/sql/ Refer to the page: http://www.grin-global.org/sql_examples.htm for additional GRIN-Global SQL examples and resources.

TOC

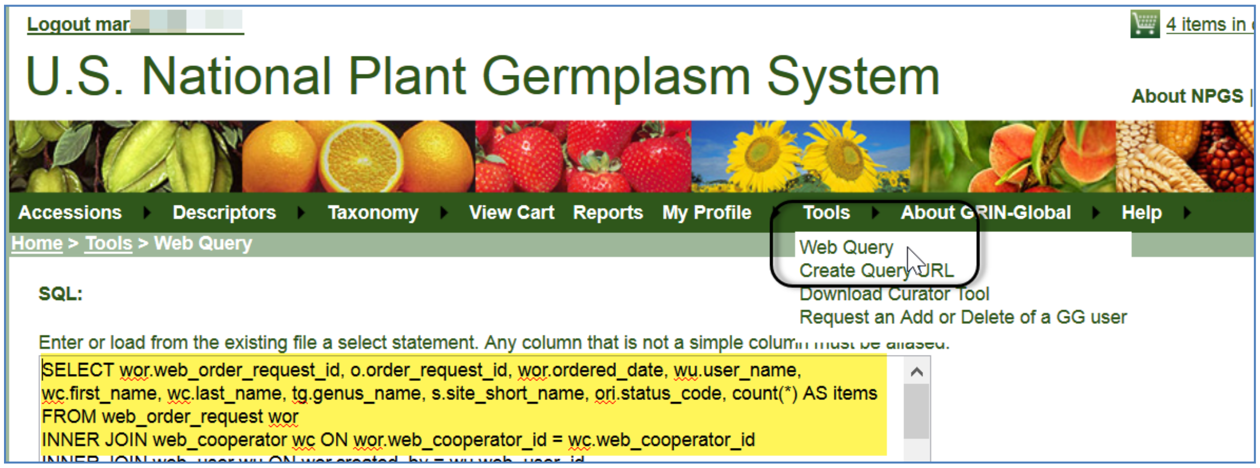
Overview: SQL and the Public Website

Genebank staff who have had their Public Website account connected to their Curator Tool account,* when logged into the Public Website, will have the **Tools** option visible on the menu. From there, select **Web Query** to display the box for inputting SQL:



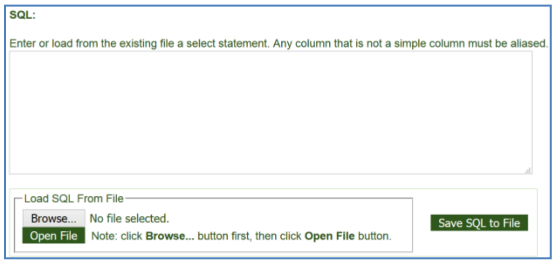
* The organization's GRIN-Global administrator is the only person with the authority to connect the two accounts (via the GG Admin Tool).

Log in; select **Tools** | **Web Query** You can copy or type valid SQL in the box as shown:



```
SELECT wor.web_order_request_id, o.order_request_id, wor.ordered_date, wu.user_name,
wc.first_name, wc.last_name, tg.genus_name, s.site_short_name, ori.status_code, count(*) AS items
FROM web_order_request wor
INNER JOIN web_cooperator wc ON wor.web_cooperator_id = wc.web_cooperator_id
```

In the Public Website, it is possible to open a .txt file in which SQL has been stored. You can also save your SQL for future reuse. The how-to should be fairly intuitive – click **Browse** to find the file on your hard drive or network locations, then click the **Open File** button. When you have a working SQL statement which you may possibly use again, click the **Save SQL to File** button.


SQL:
Enter or load from the existing file a select statement. Any column that is not a simple column must be aliased.
Load SQL From File
Browse...
No file selected.
Open File
Note: click Browse... button first, then click Open File button.
Save SQL to File

SQL – 3 Basic Components

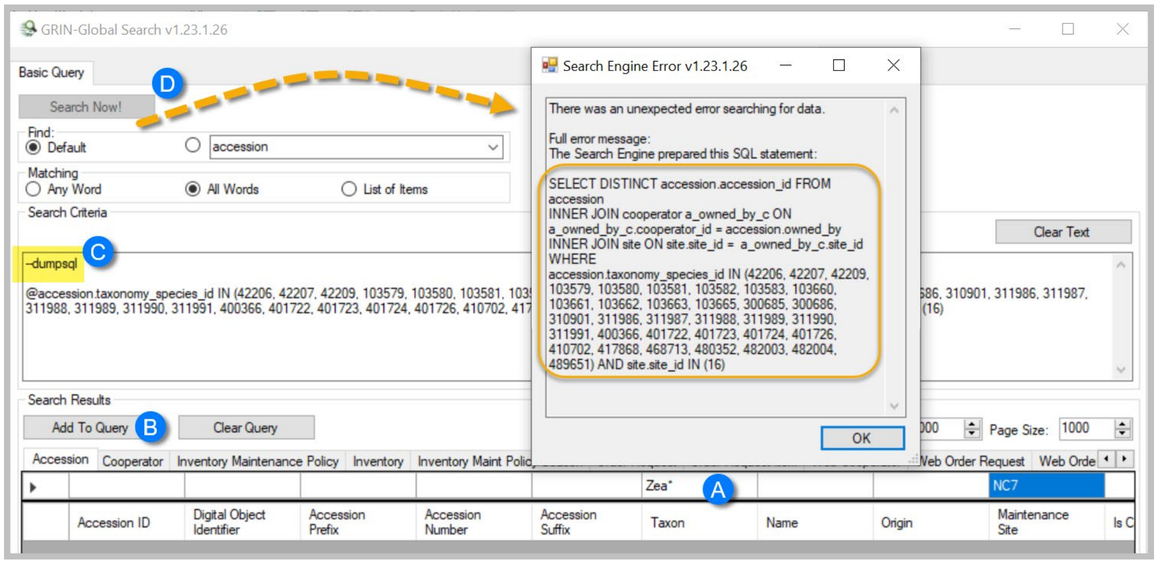
SELECT – what columns to display
FROM – what tables to search
WHERE – what criteria

In general, in GRIN-Global, most SQL statements will use these three words. In a valid command, you indicate what data you want to display and the conditions. In the GRIN-Global Public Website, a user cannot modify data – only read. Statements such as INSERT or DELETE do not work on the PW page.

Syntax

It is often easier to create SQL by using the Search Tool. Set up a search, with the desired dataview, and begin the query with the following statement:

--dumpsql



```
--dumpsql
@accession.taxonomy_species_id IN (42206, 42207, 42209, 103579, 103580, 103581, 103
311988, 311989, 311990, 311991, 400366, 401722, 401723, 401724, 401726, 410702, 417
```

Search Engine Error v1.23.1.26

```
There was an unexpected error searching for data.

Full error message:
The Search Engine prepared this SQL statement:

SELECT DISTINCT accession.accession_id FROM
accession
INNER JOIN cooperator a_owned_by_c ON
a_owned_by_c.cooperator_id = accession.owned_by
INNER JOIN site ON site.site_id =  a_owned_by_c.site_id
WHERE
accession.taxonomy_species_id IN (42206, 42207, 42209,
103579, 103580, 103581, 103582, 103583, 103660,
103661, 103662, 103663, 103665, 300685, 300686,
310901, 311986, 311987, 311988, 311989, 311990,
311991, 400366, 401722, 401723, 401724, 401726,
410702, 417868, 468713, 480352, 482003, 482004,
489651) AND site.site_id IN (16)
```

OK

1. not case sensitive
2. use comments for readability
   a. when you use a double dash -- on a line, anything after the double dash is treated as a comment
   b. to comment multiple lines, start with **/*** and then end your comment with ***/**
3. commas are needed between items in a list
4. use ***** for all
5. the wild cards **%** and _ are valid. % for any number of characters; the underscore for a single character
6. use single quotes, not double, when referring to string literals

In the following examples, items in red can be edited and changed to indicate real data.

Two Simple GG Queries

Find email Address when Web Cooperator Last Name is Known

```
SELECT   last_name, first_name, email
FROM     web_cooperator
WHERE    last_name = 'Reisinger'
```

Find Web Order # when Web Cooperator Email is Known

```
SELECT   web_cooperator_id, first_name, last_name, email, created_date
FROM     web_cooperator
WHERE    email = 'mrducks@rrginc.com'
```

The LIKE Operator & Wildcards

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards used in conjunction with the LIKE operator:

Find email Address when partial spelling of the cooperator's Last Name is known

```
SELECT *
FROM web_cooperator
WHERE last_name LIKE 'Reis%'
```

When Do You Use Quotes?

Use quotes when the fields have text (non-numeric) data.

```
…WHERE accession_id = 1927546

…WHERE  s.site_short_name = 'S9'

BETWEEN  '10-01-2014' and '9-30-2015'
```

Most of the examples in this document can be copied directly onto the Public Website page and then be executed. However, the ‘ used by Word is invalid in SQL. You will often need to edit the apostrophes to ensure that the SQL is valid and replace with **'**

ORDER BY

ORDER BY is used to sort the results in ascending or descending order. By default, in ascending order; use **ORDER BY DESC** to sort the records in descending order.

Find Site Information

```
SELECT site_id, site_short_name, fao_institute_number
FROM site
ORDER BY site_id
```

Determining Table and Field names?

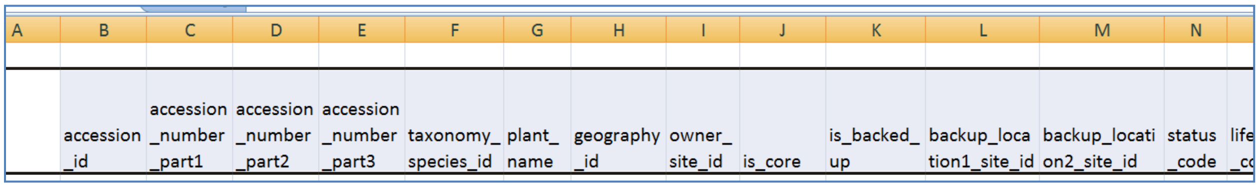
The **INFORMATION_SCHEMA.COLUMNS** view

```
SELECT table_name, column_name, ordinal_position, data_type, character_maximum_length
FROM information_schema.columns

SELECT table_name, column_name
FROM information_schema.columns
WHERE table_name LIKE  'accession%'
```

Use the Curator Tool to Determine Field Names

A Curator Tool dataview often has fields from more than one table; in fact, when editing in a dataview, the CT user should be aware that the gray fields are not editable for various reasons – often because that column is a derived (calculated) field or comes from another table. Remember that users of the Curator Tool work with dataviews, not directly with tables. However, in the CT, when using the CTRL key when you drag and drop a row to an Excel sheet, you can determine the actual database fieldnames:

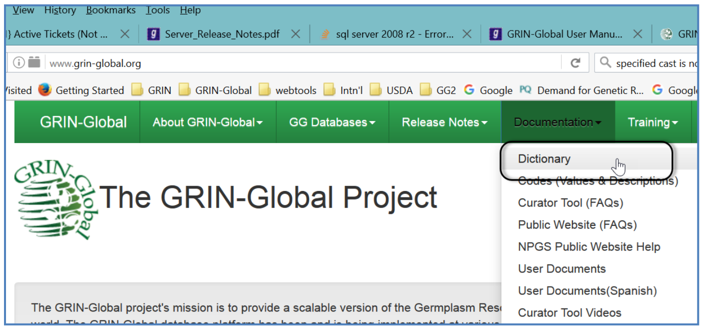


A	B	C	D	E	F	G	H	I	J	K	L	M	N	
	accession_id	accession_number_part1	accession_number_part2	accession_number_part3	taxonomy_species_id	plant_name	geography_id	owner_site_id	is_core	is_backed_up	backup_location1_site_id	backup_location2_site_id	status_code	life_c

Even when using the CTRL drag & drop method to display field names, the tables names are not displayed, so some deduction is in order. In the example here for the Accession dataview, the taxonomy_species_id field is a good example. We don't know for sure what table this came from, but the name gives us a good idea. The naming convention used throughout GRIN-Global was to name the primary key field with "_id" – preceded by the table name. In this case, the table is **taxonomy_species**. When you cannot determine the table by deduction, familiarity, or reviewing the data dictionary, contact your GG administrator who can use additional tools, such as the GG Admin Tool.

Data Dictionary is also a Source for Table and Field Names

Also, the online data dictionary is another alternative which can be used to display column names.



shortened URL direct to the dictionary: https://goo.gl/z2y1gh

COUNT

The COUNT() function returns the number of rows that matches a specified criteria.

Two GRIN-Global Examples

```
SELECT
  COUNT(*) AS Order_Items
FROM    order_request_item   ori

SELECT
  COUNT(*) AS Active_Accessions
FROM    accession a
WHERE  status_code = 'ACTIVE'
```

DISTINCT

The SELECT DISTINCT statement is used to return only distinct (different) values.

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

GRIN-Global Example

```
… AND ori.status_code IN ('INSPECT','PSHIP','SHIPPED')

…
JOIN site s ON s.site_id =  c.site_id
WHERE s.site_short_name  IN  ('NR6', 'S9')
```

NOT IN is also valid

```
SELECT accession_number_part1, accession_number_part2, accession_number_part3, c.last_name,
c.first_name, s.site_short_name
FROM accession a
JOIN cooperator c ON a.owned_by = c.cooperator_id
JOIN site s ON s.site_id =  c.site_id
WHERE s.site_short_name  NOT IN  ('NR6', 'S9')
```

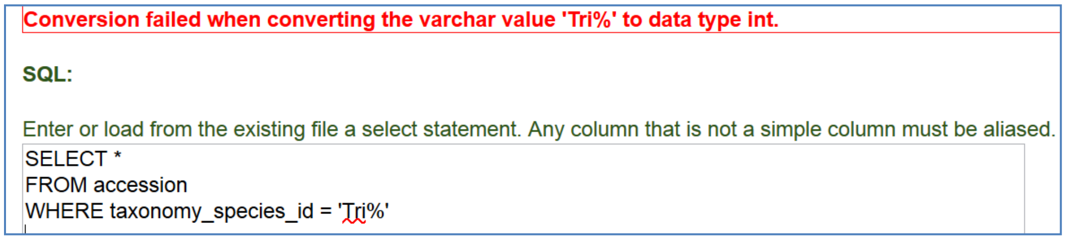
Multiple Tables

GRIN-Global has many tables by design. Database designers do this for multiple reasons, generally, by doing so, they make the database more flexible and capable of handling future data needs. But having the data spread across multiple tables requires more finesse when writing your SQL. You will frequently find that in order to display data that you want, your SQL statements will include JOIN clauses.

For example, if you were interested in searching for accessions with a certain Taxon, such as Triticum%, at first glance the following may appear valid:

```
SELECT *
FROM accession
WHERE taxonomy_species_id = ‘Tri%’
```

But the system will respond:



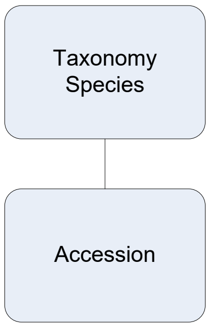
Conversion failed when converting the varchar value 'Tri%' to data type int.

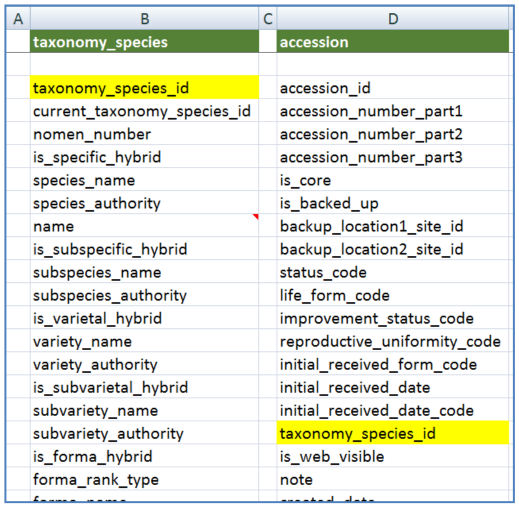
SQL:

Enter or load from the existing file a select statement. Any column that is not a simple column must be aliased.

```
SELECT *
FROM accession
WHERE taxonomy_species_id = 'Tri%'
```

The taxonomy_species_id field is numeric (data type Integer). In fact, all of the GG _id fields are numeric. If we want to specify the species name (or partial name) in our WHERE criterion clause, we need to have SQL use two tables, the **accession**, and the **taxonomy_species**.





taxonomy_species	accession
taxonomy_species_id	accession_id
current_taxonomy_species_id	accession_number_part1
nomen_number	accession_number_part2
is_specific_hybrid	accession_number_part3
species_name	is_core
species_authority	is_backed_up
name	backup_location1_site_id
is_subspecific_hybrid	backup_location2_site_id
subspecies_name	status_code
subspecies_authority	life_form_code
is_varietal_hybrid	improvement_status_code
variety_name	reproductive_uniformity_code
variety_authority	initial_received_form_code
is_subvarietal_hybrid	initial_received_date
subvariety_name	initial_received_date_code
subvariety_authority	taxonomy_species_id
is_forma_hybrid	is_web_visible
forma_rank_type	note

(see the tax-acc spreadsheet on file join_examples.xlsx)

The field that is common to both tables is the **taxonomy_species_id** field. It is the primary_key field for the **taxonomy_species** table; each record in that table has a unique **taxonomy_species_id**. The WHERE clause needs to point to the **name** field in the **taxonomy_species** table.

ALIASES

An alias is simply an alternative name for either a table or a field. In the following example, aliases will be created for the two tables, **accession**, and **taxonomy_species**. An alias is typically a shorter name, making it easier to code, and also making the code clearer because you can quickly see which table the field is in. The renaming is temporary; the actual table names do not change.

In the following SELECT clause, **a** is the alias for **accession**, and **ts** is the alias for **taxonomy_species.** These aliases are actually defined in the FROM and JOIN clauses, which follow the SELECT clause. (Aliases typically use letters from the original table name, but they are not required to do so.)

```
SELECT
a.accession_number_part1, a.accession_number_part2,
a.accession_number_part3,
ts.name

FROM taxonomy_species ts
JOIN accession  a  ON ts.taxonomy_species_id = a.taxonomy_species_id

WHERE ts.name LIKE 'Trit%'
  AND  a.status_code = 'ACTIVE'
```

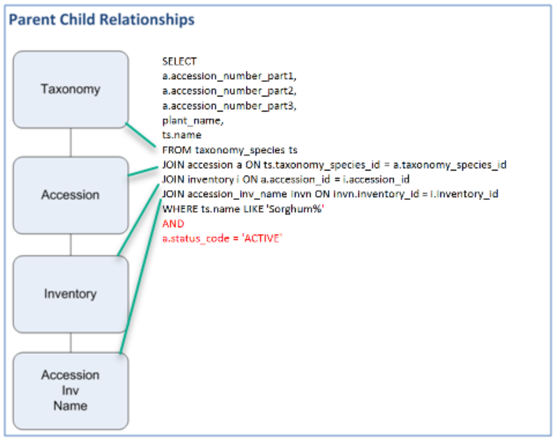
In this case, it did not matter which table's taxonomy_species_id field was listed first. We could have written

```
JOIN accession  a  ON a.taxonomy_species_id = ts.taxonomy_species_id
```

JOINs: Relating Tables to Obtain Data

A "JOIN" in SQL returns rows where there is at least one match on both tables. Assume we want to search for accession records whose name is SORGHUM… Let's assume that we have the following tables:

(see spreadsheet: [tax-acc-inv-name]())



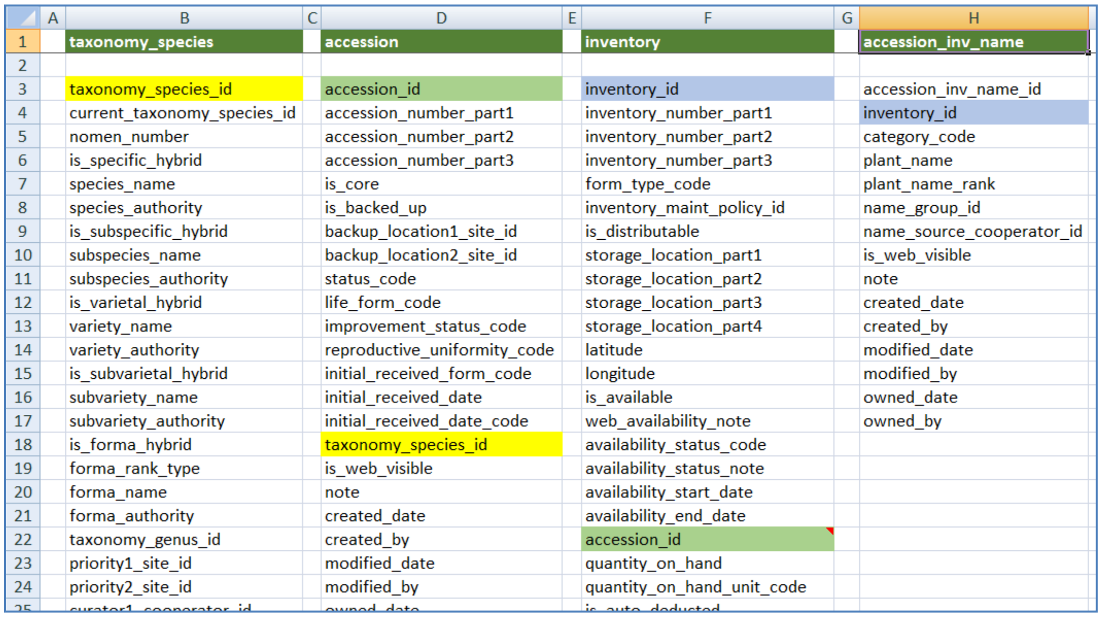
```
SELECT
a.accession_number_part1,
a.accession_number_part2,
a.accession_number_part3,
plant_name,
ts.name
FROM taxonomy_species ts
JOIN accession a ON ts.taxonomy_species_id = a.taxonomy_species_id
JOIN inventory i ON a.accession_id = i.accession_id
JOIN accession_inv_name invn ON invn.inventory_id = i.inventory_id
WHERE ts.name LIKE 'Sorghum%'
AND
a.status_code = 'ACTIVE'
```

```
SELECT
a.accession_number_part1, a.accession_number_part2,
a.accession_number_part3, plant_name,  ts.name

FROM  taxonomy_species ts
JOIN   accession a ON ts.taxonomy_species_id = a.taxonomy_species_id
JOIN   inventory i ON a.accession_id = i.accession_id
JOIN   accession_inv_name invn ON invn.inventory_id = i.inventory_id

WHERE ts.name LIKE 'Sorghum%'   AND   a.status_code = 'ACTIVE'
```

You may find it very helpful to first list the fields from each table into a spreadsheet, similar to the following:



	A	B	C	D	E	F	G	H
1		**taxonomy_species**		**accession**		**inventory**		**accession_inv_name**
2								
3		taxonomy_species_id		accession_id		inventory_id		accession_inv_name_id
4		current_taxonomy_species_id		accession_number_part1		inventory_number_part1		inventory_id
5		nomen_number		accession_number_part2		inventory_number_part2		category_code
6		is_specific_hybrid		accession_number_part3		inventory_number_part3		plant_name
7		species_name		is_core		form_type_code		plant_name_rank
8		species_authority		is_backed_up		inventory_maint_policy_id		name_group_id
9		is_subspecific_hybrid		backup_location1_site_id		is_distributable		name_source_cooperator_id
10		subspecies_name		backup_location2_site_id		storage_location_part1		is_web_visible
11		subspecies_authority		status_code		storage_location_part2		note
12		is_varietal_hybrid		life_form_code		storage_location_part3		created_date
13		variety_name		improvement_status_code		storage_location_part4		created_by
14		variety_authority		reproductive_uniformity_code		latitude		modified_date
15		is_subvarietal_hybrid		initial_received_form_code		longitude		modified_by
16		subvariety_name		initial_received_date		is_available		owned_date
17		subvariety_authority		initial_received_date_code		web_availability_note		owned_by
18		is_forma_hybrid		taxonomy_species_id		availability_status_code		
19		forma_rank_type		is_web_visible		availability_status_note		
20		forma_name		note		availability_start_date		
21		forma_authority		created_date		availability_end_date		
22		taxonomy_genus_id		created_by		accession_id		
23		priority1_site_id		modified_date		quantity_on_hand		
24		priority2_site_id		modified_by		quantity_on_hand_unit_code		
25		curator1_cooperator_id		owned_date		is_auto_deducted		

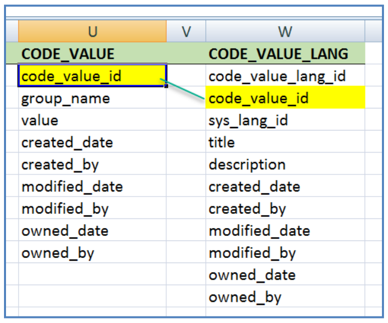
(see the tax-acc-inv-name spreadsheet on file join_examples.xlsx)

The fields linking the tables were highlighted to show how the tables relate to each other. The four tables were required for this SQL statement because the user wanted to display the data in the **plant_name** field in the **accession_inv_name** table. Since that table relates indirectly to the **accession** table via the **inventory** table, we needed the four tables. (We saw in the previous example why we needed the taxonomy_species and the accession tables.)

JOIN Example: Table Code Value and Code Value Lang

Another example when JOINING tables is necessary is the **Code Value** table. IN GG we don't store the titles and descriptions for the Codes used in dropdowns because it is possible to use different languages in GG. The codes that display in drop downs in the CT display in the user's preferred language. For example, in the U.S. NPGS, all users have their languages set to English when they are given a CT account.

The following spreadsheet graphic shows how the Code Value and the Code Value Lang tables relate to each other, via the common **code_value_id** field. Following the illustration is sample SQL code.

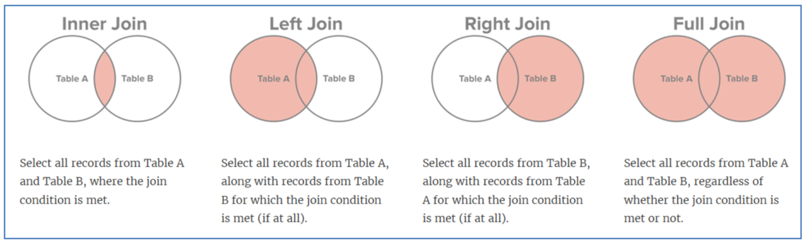


U	V	W
CODE_VALUE		CODE_VALUE_LANG
code_value_id		code_value_lang_id
group_name		code_value_id
value		sys_lang_id
created_date		title
created_by		description
modified_date		created_date
modified_by		created_by
owned_date		modified_date
owned_by		modified_by
		owned_date
		owned_by

```
SELECT cv.code_value_id, group_name, value, title
FROM code_value_lang cvl    -- language table
JOIN code_value cv  ON  cv.code_value_id = cvl.code_value_id  -- join w/ the code_value table
WHERE  cvl.sys_lang_id = 1
/* AND group_name = 'IMPROVEMENT_LEVEL'  */
```

(see the Codes spreadsheet on file join_examples.xlsx)

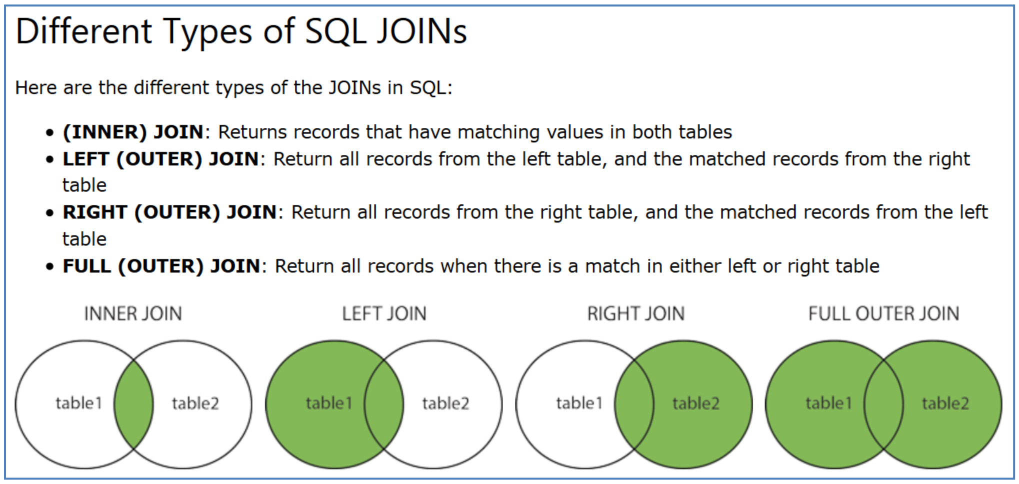
In many of the GG databases, default languages were installed; English happened to be the first language, hence **cvl.sys_lang_id = 1** is indicating the English language.


Inner Join
Table A
Table B
Select all records from Table A and Table B, where the join condition is met.
Left Join
Table A
Table B
Select all records from Table A, along with records from Table B for which the join condition is met (if at all).
Right Join
Table A
Table B
Select all records from Table B, along with records from Table A for which the join condition is met (if at all).
Full Join
Table A
Table B
Select all records from Table A and Table B, regardless of whether the join condition is met or not.

http://www.sql-join.com/sql-join-types/ (INNER JOIN is the same JOIN)

"The first table mentioned is the left side and the second table is the right. When you're joining from parent to child (FROM parent JOIN child ON…), the parent is the left side. If you don't want to see childless parents use an (INNER) JOIN. If you do want to see childless parents, then you need a LEFT JOIN. Whenever I'm joining in the reverse direction from parent to child, I'm usually focusing on the children so an INNER JOIN is fine because GG doesn't have parentless children."

-- a SQL guru



Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

https://www.w3schools.com/sql/sql_join.asp

JOIN Example: Web Cooperator

```
SELECT c.last_name, c.first_name, c.email, c.address_line1, c.address_line2,  c.address_line3,
c.city, c.postal_index, g.country_code, c.web_cooperator_id
FROM cooperator c
JOIN geography g ON c.geography_id = g.geography_id
JOIN web_cooperator wc ON wc.web_cooperator_id = c.web_cooperator_id
WHERE
/* substitute name */
c.last_name LIKE 'Reisinger'   AND   c.first_name LIKE 'Mar%'
```

ON vs. WHERE

Regarding ON "I quickly came to appreciate how they closely associated the conditions for joining each table. Previously I would often find myself untangling all the conditions in the WHERE section trying to determine which were used to join the tables and which were about getting the right data. With the JOIN and ON, those conditions are arranged in an orderly fashion."

For in-depth comparison of ON and WHERE, see: http://stackoverflow.com/questions/2722795/in-sql-mysql-what-is-the-difference-between-on-and-where-in-a-join-statem.

"...The ON clause defines the relationship between the tables. The WHERE clause describes which rows you are interested in (the criteria). Many times you can swap them and still get the same result, however this is not always the case with a left outer join.

- If the ON clause fails you still get a row with columns from the left table but with nulls in the columns from the right table.
- If the WHERE clause fails you won't get that row at all."

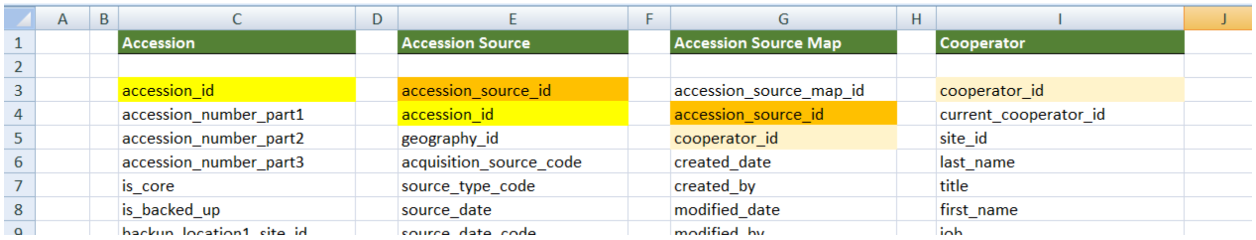
JOIN Query for Crops with Observations at a Site

The following query will display a count of the observations under each crop in a specified site. It relates five tables to get the results:

```
SELECT crop.name AS Crop,
    COUNT(*) AS Total_obs
FROM   crop
   JOIN crop_trait ct
        ON crop.crop_id = ct.crop_id
   JOIN crop_trait_observation cto
        ON ct.crop_trait_id = cto.crop_trait_id
   JOIN cooperator c
        ON ct.owned_by = c.cooperator_id
   JOIN site s
        ON c.site_id = s.site_id
WHERE  s.site_short_name = 'S9'
GROUP  BY crop.name
ORDER  BY crop.name
```

(see the SiteCropObs spreadsheet on file join_examples.xlsx)

Source and Source Cooperator Example



	A	B	C	D	E	F	G	H	I	J
1			Accession		Accession Source		Accession Source Map		Cooperator	
2										
3			accession_id		accession_source_id		accession_source_map_id		cooperator_id	
4			accession_number_part1		accession_id		accession_source_id		current_cooperator_id	
5			accession_number_part2		geography_id		cooperator_id		site_id	
6			accession_number_part3		acquisition_source_code		created_date		last_name	
7			is_core		source_type_code		created_by		title	
8			is_backed_up		source_date		modified_date		first_name	
9			backup_location1_site_id		source_date_code		modified_by		job	

(see the Source spreadsheet on file join_examples.xlsx)

EXISTS Operator (and Subqueries)

The EXISTS condition is used in combination with a subquery. The EXISTS operator returns true if the subquery returns one or more records.

To answer the question: "How do I query for unavailable accessions?" you must look at the related inventory records. (There isn't an availability flag field at the accession level. Accessions are considered unavailable when none of their related inventory records are both distributable and available.) By using a subquery, the SQL first searches for that condition and then uses the results to resolve the main query.

In the following example:

```
SELECT a.*
FROM accession a
JOIN taxonomy_species ts ON ts.taxonomy_species_id = a.taxonomy_species_id
WHERE ts.name like 'Glycine%'
  AND NOT EXISTS (SELECT * FROM inventory I
WHERE i.accession_id = a.accession_id
AND is_distributable = 'Y' AND is_available = 'Y')
```

the subquery is

```
(SELECT * FROM inventory I
WHERE i.accession_id = a.accession_id
AND is_distributable = 'Y' AND is_available = 'Y')
```

using NOT EXISTS (SELECT *condition*) excludes the records found in the subquery condition. In this example, when the inventory records have two fields both equal to "Y" , the condition is met -- the accession records would be available. But the question was asking for those accessions that are not available, hence the SQL uses *NOT* EXISTS (subquery).

Appendix A: Frequently Used JOIN Statements

The following SQL can be used to generate JOIN statements for common child tables:

```
SELECT pt.table_name Parent, ct.table_name Child, ' JOIN ' + ct.table_name +' ON '
+ct.table_name +'.'+ cf.field_name +' = '+ pt.table_name +'.'+ pf.field_name AS join_clause

FROM sys_table_relationship str

JOIN sys_table_field pf ON pf.sys_table_field_id = str.other_table_field_id
JOIN sys_table pt ON pt.sys_table_id = pf.sys_table_id
JOIN sys_table_field cf ON cf.sys_table_field_id = str.sys_table_field_id
JOIN sys_table ct ON ct.sys_table_id = cf.sys_table_id

WHERE relationship_type_tag = 'OWNER_PARENT'
ORDER BY 1,2
```

Parent	**Child**	**join_clause**
accession	accession_action	JOIN accession_action ON accession_action.accession_id = accession.accession_id
accession	accession_ipr	JOIN accession_ipr ON accession_ipr.accession_id = accession.accession_id
accession	accession_pedigree	JOIN accession_pedigree ON accession_pedigree.accession_id = accession.accession_id
accession	accession_quarantine	JOIN accession_quarantine ON accession_quarantine.accession_id = accession.accession_id
accession	accession_source	JOIN accession_source ON accession_source.accession_id = accession.accession_id
crop	genetic_marker	JOIN genetic_marker ON genetic_marker.crop_id = crop.crop_id
crop_trait	crop_trait_code	JOIN crop_trait_code ON crop_trait_code.crop_trait_id = crop_trait.crop_trait_id
inventory	accession_inv_annotation	JOIN accession_inv_annotation ON accession_inv_annotation.inventory_id = inventory.inventory_id
inventory	accession_inv_attach	JOIN accession_inv_attach ON accession_inv_attach.inventory_id = inventory.inventory_id
inventory	accession_inv_name	JOIN accession_inv_name ON accession_inv_name.inventory_id = inventory.inventory_id
inventory	accession_inv_voucher	JOIN accession_inv_voucher ON accession_inv_voucher.inventory_id = inventory.inventory_id
inventory	crop_trait_observation	JOIN crop_trait_observation ON crop_trait_observation.inventory_id = inventory.inventory_id
inventory	genetic_observation	JOIN genetic_observation ON genetic_observation.inventory_id = inventory.inventory_id
inventory	geneva_site_inventory	JOIN geneva_site_inventory ON geneva_site_inventory.inventory_id = inventory.inventory_id
inventory	inventory_action	JOIN inventory_action ON inventory_action.inventory_id = inventory.inventory_id
inventory	inventory_quality_status	JOIN inventory_quality_status ON inventory_quality_status.inventory_id = inventory.inventory_id
inventory	inventory_viability	JOIN inventory_viability ON inventory_viability.inventory_id = inventory.inventory_id

inventory	nc7_site_inventory	JOIN nc7_site_inventory ON nc7_site_inventory.inventory_id = inventory.inventory_id
inventory	ne9_site_inventory	JOIN ne9_site_inventory ON ne9_site_inventory.inventory_id = inventory.inventory_id
inventory	nssl_site_inventory	JOIN nssl_site_inventory ON nssl_site_inventory.inventory_id = inventory.inventory_id
inventory	opgc_site_inventory	JOIN opgc_site_inventory ON opgc_site_inventory.inventory_id = inventory.inventory_id
inventory	parl_site_inventory	JOIN parl_site_inventory ON parl_site_inventory.inventory_id = inventory.inventory_id
inventory	s9_site_inventory	JOIN s9_site_inventory ON s9_site_inventory.inventory_id = inventory.inventory_id
inventory	w6_site_inventory	JOIN w6_site_inventory ON w6_site_inventory.inventory_id = inventory.inventory_id
inventory_maint_policy	inventory	JOIN inventory ON inventory.inventory_maint_policy_id = inventory_maint_policy.inventory_maint_policy_id
order_request	order_request_action	JOIN order_request_action ON order_request_action.order_request_id = order_request.order_request_id
order_request	order_request_item	JOIN order_request_item ON order_request_item.order_request_id = order_request.order_request_id
taxonomy_family	taxonomy_genus	JOIN taxonomy_genus ON taxonomy_genus.taxonomy_family_id = taxonomy_family.taxonomy_family_id

Appendix B: Document Change Notes

– May 30, 2023

- added notes and image re --dumpsql