## 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

|  |  |
| --- | --- |
| **image2449.png** | 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 3](#_Toc136352943)

[SQL – 3 Basic Components 4](#_Toc136352944)

[Syntax 4](#_Toc136352945)

[Two Simple GG Queries 5](#_Toc136352946)

[The LIKE Operator & Wildcards 5](#_Toc136352947)

[When Do You Use Quotes? 5](#_Toc136352948)

[ORDER BY 5](#_Toc136352949)

[Determining Table and Field names? 6](#_Toc136352950)

[COUNT 7](#_Toc136352951)

[DISTINCT 8](#_Toc136352952)

[The IN operator allows you to specify multiple values in a WHERE clause. 8](#_Toc136352953)

[NOT IN is also valid 8](#_Toc136352954)

[Multiple Tables 9](#_Toc136352955)

[ALIASES 10](#_Toc136352956)

[JOINs: Relating Tables to Obtain Data 10](#_Toc136352957)

[JOIN Example: Table Code Value and Code Value Lang 12](#_Toc136352958)

[JOIN Example: Web Cooperator 13](#_Toc136352959)

[JOIN Query for Crops with Observations at a Site 14](#_Toc136352960)

[Source and Source Cooperator Example 14](#_Toc136352961)

[EXISTS Operator (and Subqueries) 15](#_Toc136352962)

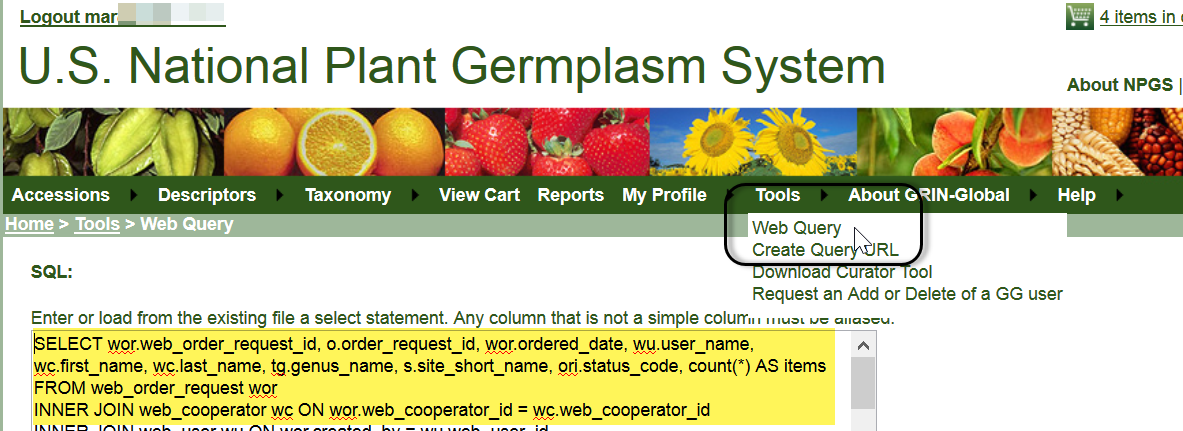
[Appendix A: Frequently Used JOIN Statements 16](#_Toc136352963)

[Appendix B: Document Change Notes 18](#_Toc136352964)

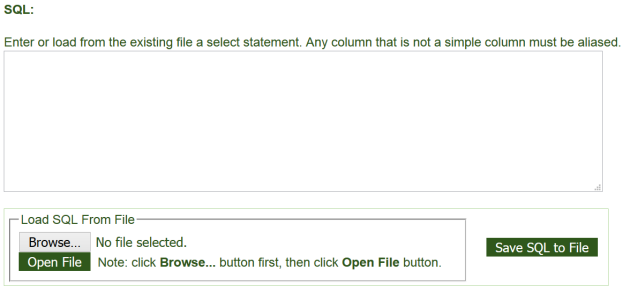
#### Overview: SQL and the Public Website

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:  


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 – 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

|  |  |
| --- | --- |
| A picture containing light, drawing  Description automatically generated | 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** |

1. not case sensitive
2. use comments for readability
   1. when you use a double dash -- on a line, anything after the double dash is treated as a comment
   2. 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'

|  |  |
| --- | --- |
| image2449.png | 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

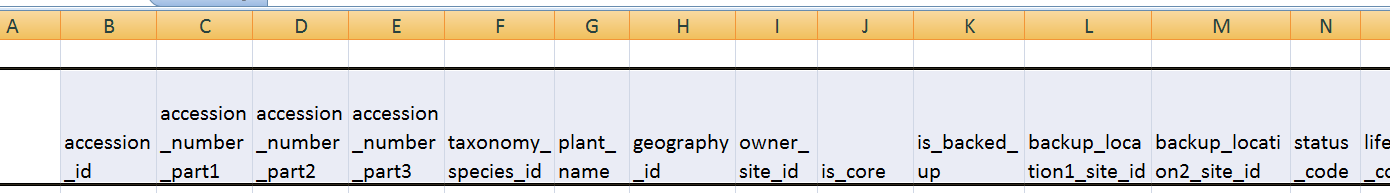
SELECT table\_name, column\_name, ordinal\_position, data\_type, character\_maximum\_length  
FROM information\_schema.columns

SELECT table\_name, column\_name

FROM information\_schema.columns

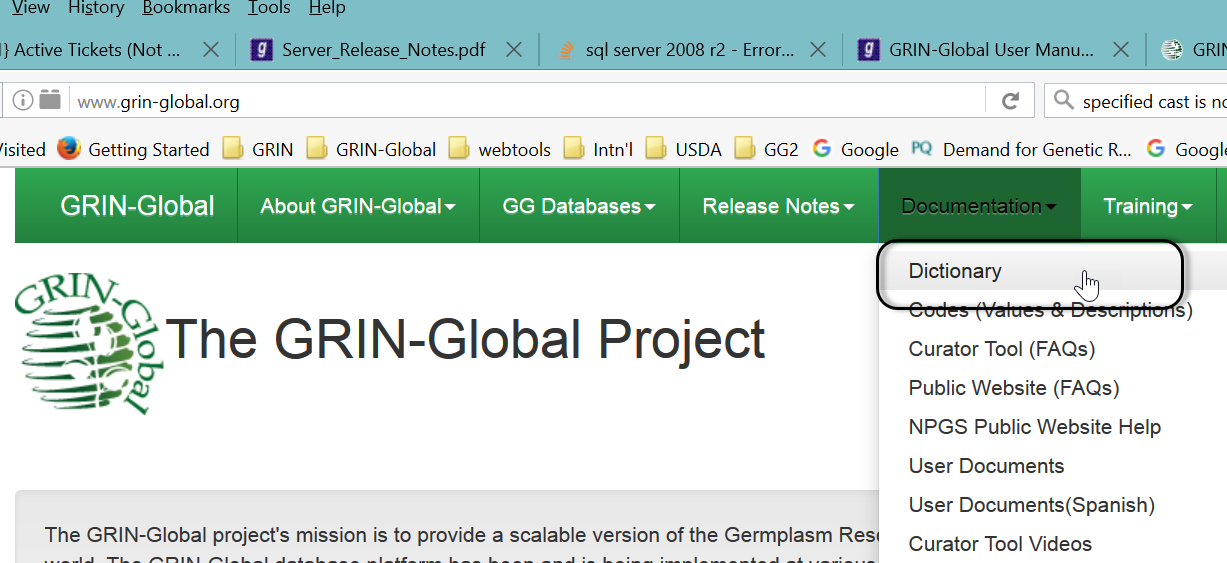
WHERE table\_name LIKE '**accession**%'

##### 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:   


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](https://goo.gl/z2y1gh) 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

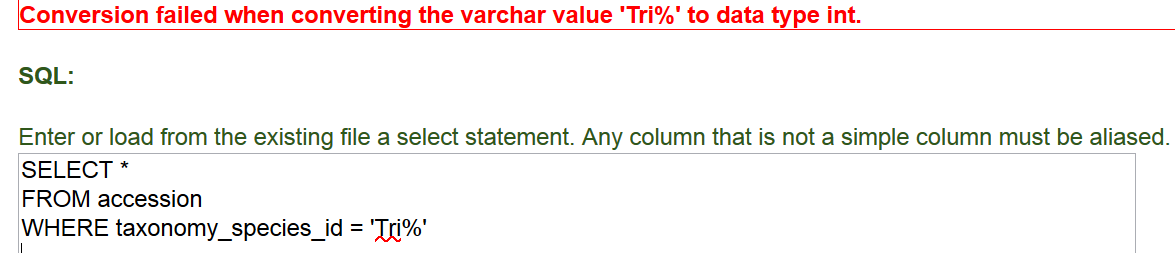
SELECT accession\_number\_part1, accession\_number\_part2, accession\_number\_part3, c.last\_name, c.first\_name, s.site\_short\_name   
FROM accession a  
JOIN cooperator c ON a.owned\_by = c.cooperator\_id   
JOIN site s ON s.site\_id = c.site\_id   
WHERE s.site\_short\_name NOT IN ('NR6', 'S9')

#### Multiple Tables

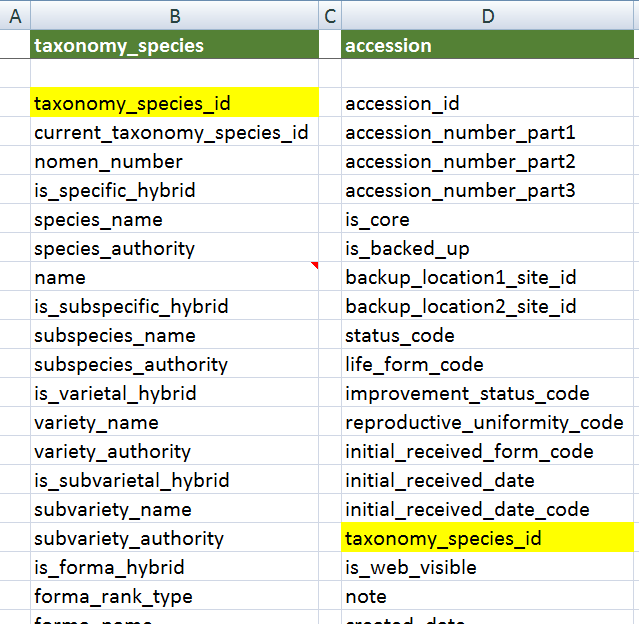
GRIN-Global has many tables by design. Database designers do this for multiple reasons, generally, by doing so, they make the database more flexible and capable of handling future data needs. But having the data spread across multiple tables requires more finesse when writing your SQL. You will frequently 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:  


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**.

(see the tax-acc spreadsheet on file [join\_examples.xlsx](http://www.grin-global.org/docs/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'

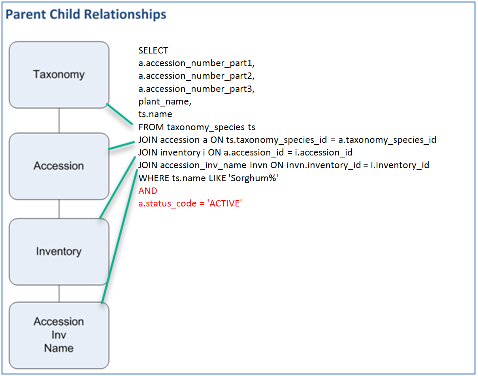
In this case, it did not matter which table’s taxonomy\_species\_id field was listed first. We could have written

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](http://www.grin-global.org/docs/join_examples.xlsx))



SELECT

a.accession\_number\_part1, a.accession\_number\_part2,

a.accession\_number\_part3, plant\_name, ts.name

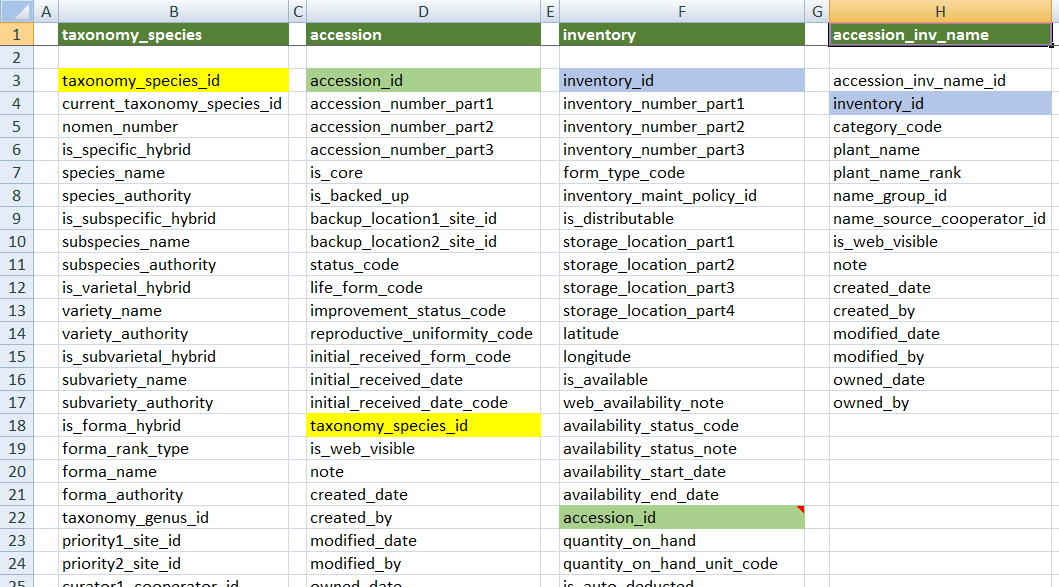
FROM taxonomy\_species ts

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:  


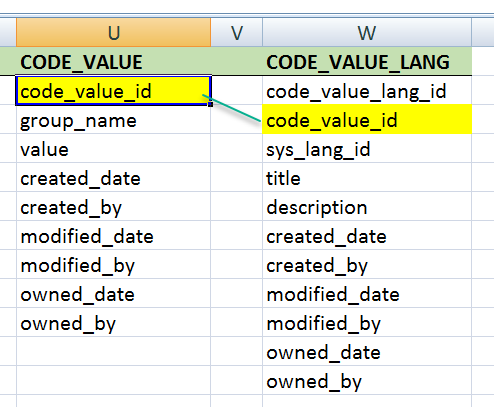
(see the tax-acc-inv-name spreadsheet on file [join\_examples.xlsx](http://www.grin-global.org/docs/join_examples.xlsx) )

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.

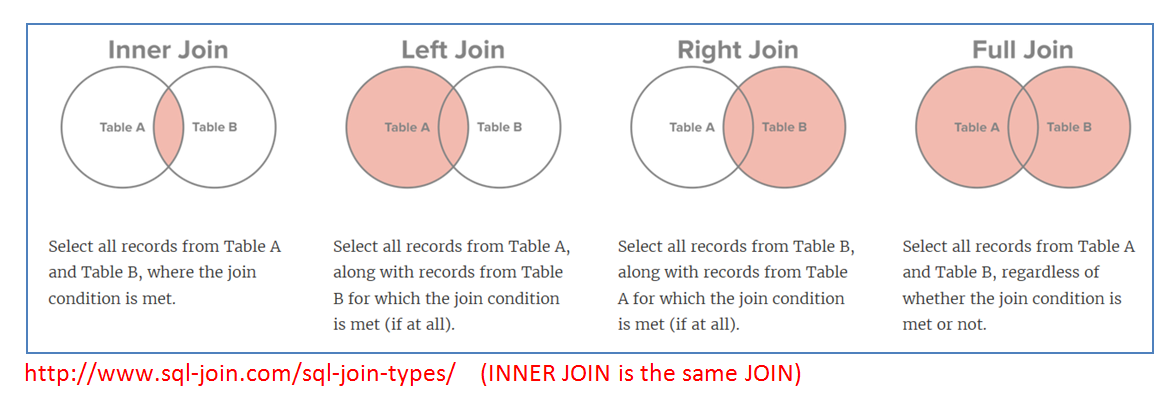
The following spreadsheet graphic shows how the Code Value and the Code Value Lang tables relate to each other, via the common **code\_value\_id** field. Following the illustration is sample SQL code.



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' \*/

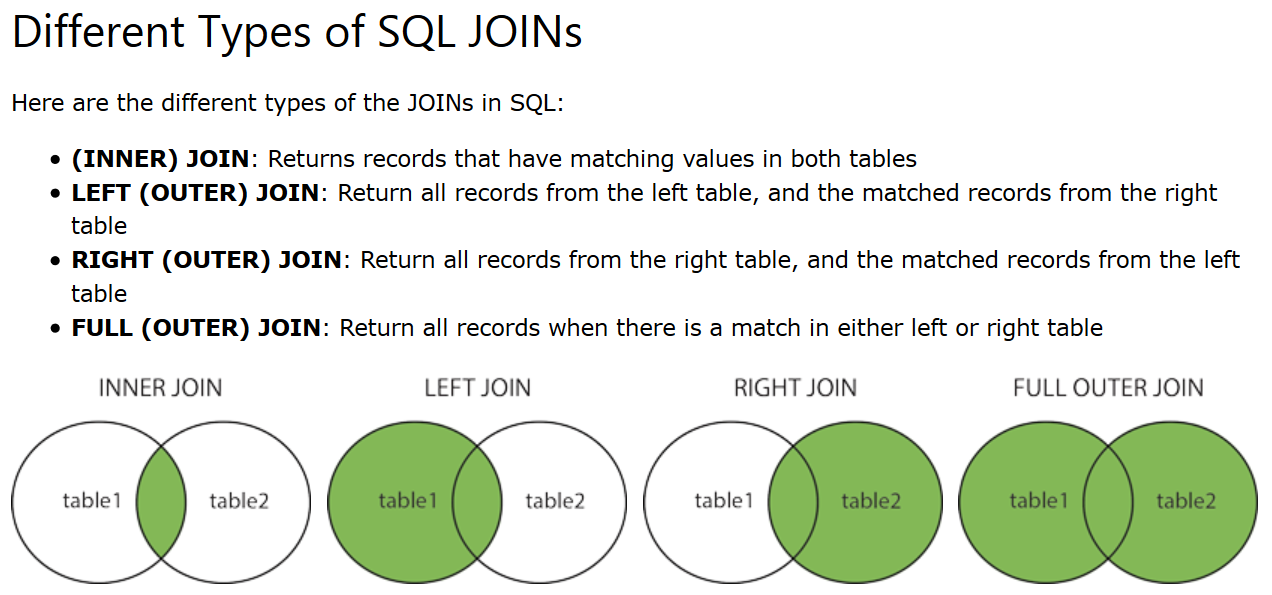
(see the Codes spreadsheet on file [join\_examples.xlsx](http://www.grin-global.org/docs/join_examples.xlsx) )

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.

****

“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

  
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](http://www.grin-global.org/docs/join_examples.xlsx) )

#### Source and Source Cooperator Example

##### 

(see the Source spreadsheet on file [join\_examples.xlsx](http://www.grin-global.org/docs/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