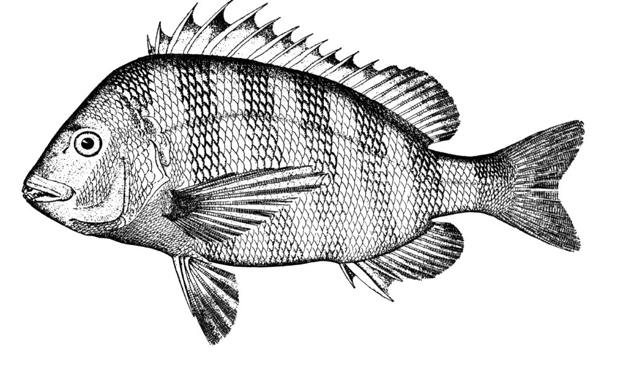
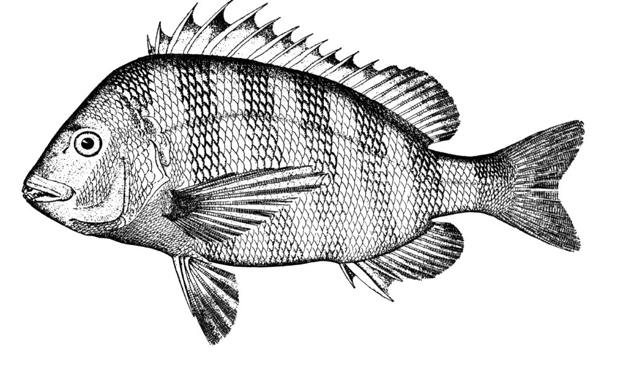
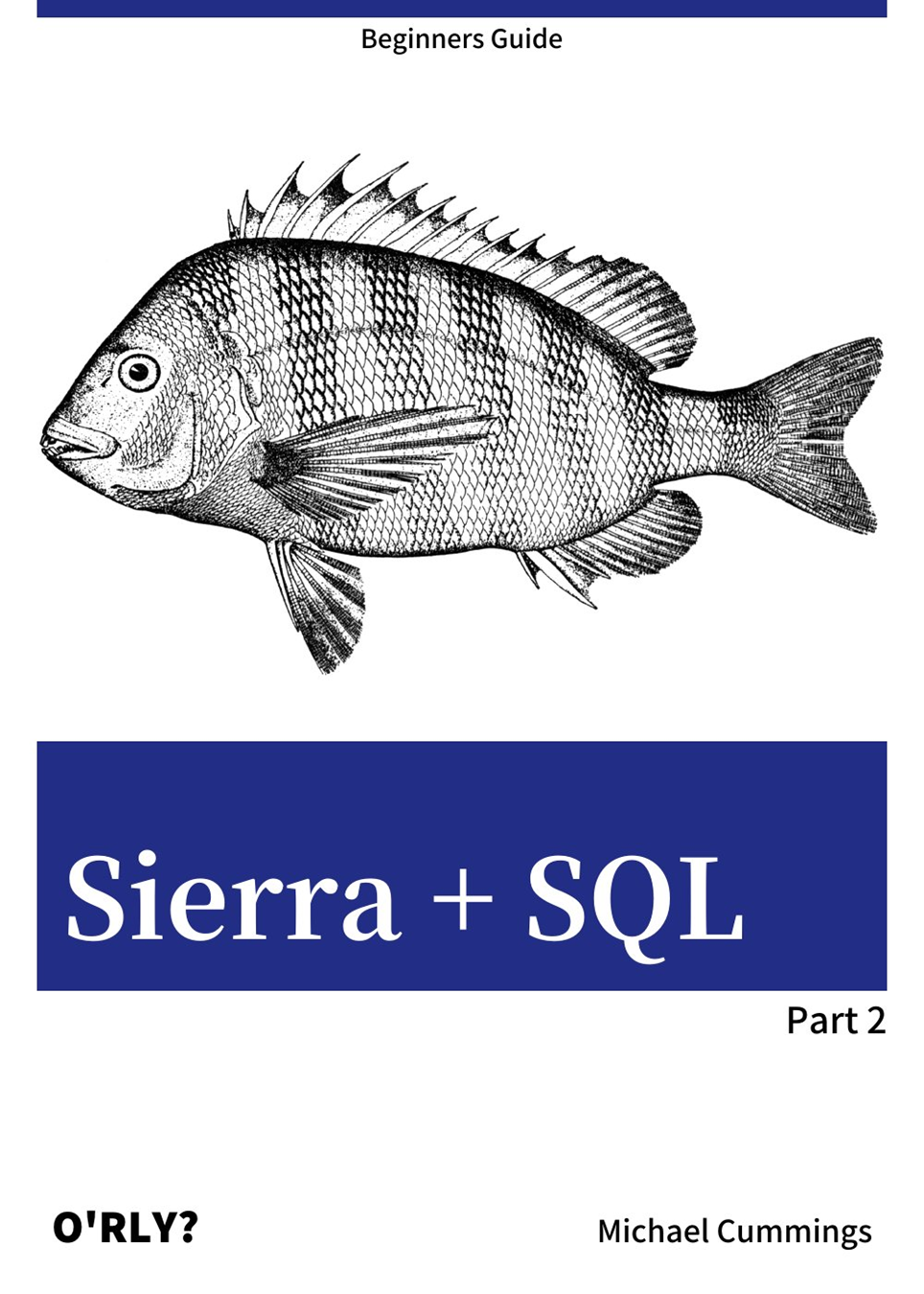
[[1]](#footnote-1)



Edited by Carol Choi

Contents

[Introduction 3](#_Toc134536631)

[Query Development 3](#_Toc134536632)

[Retrieving descriptive text for codes, an example 4](#_Toc134536633)

[Sierra DNA database documentation 5](#_Toc134536634)

[Joining tables 12](#_Toc134536635)

[Rules for joining tables 16](#_Toc134536636)

[Joining tables with a WHERE clause 16](#_Toc134536637)

[Joining tables with the JOIN statement 20](#_Toc134536638)

[Examples 21](#_Toc134536639)

[Joining Data Tables 23](#_Toc134536640)

[bib\_view and varfield\_view 24](#_Toc134536641)

[bib\_view, item\_view, and bib\_record\_item\_record\_link 25](#_Toc134536642)

[hold, patron\_record\_fullname, item\_view, and bib\_record\_item\_record\_link 27](#_Toc134536643)

[Tables often used in queries 28](#_Toc134536644)

[Left Joins 29](#_Toc134536645)

[left join syntax 33](#_Toc134536646)

[Left join examples 35](#_Toc134536647)

[Patrons and patron\_addresses 35](#_Toc134536648)

[Bibs and items 36](#_Toc134536649)

[Further Study 37](#_Toc134536650)

[Appendix 38](#_Toc134536651)

[Some SQL Queries used at Watson Library 38](#_Toc134536652)

[Address checker 38](#_Toc134536653)

[Items checked out from book cage 40](#_Toc134536654)

[All patrons 41](#_Toc134536655)

# Introduction

This is Part 2 of a two-part course called, “Sierra + SQL.” This material focuses on using pgADMIN and Sierra documentation to develop queries that involve more than one table or, in SQL terms, multiple table JOINs. These complex queries are demonstrated through different examples.

Completion of Part 1 is prerequisite to the material in this document. The reader should:

* have installed pgAdmin
* have configured a connection to Sierra via pgAdmin
* be able to navigate the PgAdmin views
* know how to list the columns/fields in a table
* know the basic SQL syntax for SELECT, FROM, and WHERE clauses

It will be helpful if you can access pgAdmin and the online SierraDNA documentation while you are working your way through the course material.

# Query Development

It is common practice in database design to store brief **codes** on data records instead of descriptive text. The descriptive text is typically stored in a separate table. This is a good starting point for learning about multi-table queries. Let’s look at a couple of examples involving code lookups in Sierra.

## Retrieving descriptive text for codes, an example

A table that stores many fields with information about bibs is named **sierra\_view.bib\_view**. Recall from Part 1 of the course that you get a list of fields in a table by expanding Columns in PgAdmin as shown below, left.

Based on the field names on the bib\_view table, several fields look like they store a short character abbreviation or number code: ‘language\_code’, ‘bcode1’, bcode2’, and ‘country\_code’ fields.

This screen shot from PgAdmin shows the fields on the bib\_view table, an SQL query statement that selects the code fields, and the data output when the query ran.

Table

Description automatically generated with low confidence

Looking at the data output, it should be obvious that the language\_code ‘eng’ means English and ‘ger’ means German. The meaning of other codes (‘m’, ‘a’, ‘gw’, ‘au’) are not very obvious. Another problem is that the field names “bcode1” and “bcode2” are not meaningful.

We will learn how to find the descriptions of those codes using the database documentation.

### Sierra DNA database documentation

Online documentation about Sierra tables, views, and relationships between tables is known as “Sierra DNA.” We will go through an exercise to research fields and table relationships using this documentation.

* Open a browser window to <https://techdocs.iii.com>
* Login with your institution account
* Choose **Sierra DNA** on the left

Graphical user interface

Description automatically generated

* select **Entries**

Graphical user interface, text, application, email

Description automatically generated

* select **Bib**. We are looking for information about codes on the bib\_view table
* select **Detailed View**
* **scroll down** to find **bib\_view**

Graphical user interface, application, Teams

Description automatically generated

As far as the codes we are looking for are concerned, the detailed view about Bibs tells us that:

1. the **language\_code** is based on the MARC 21 Code List for Languages;
2. the **country\_code** is based on the MARC 21 Code List of Countries;
3. for fields **bcode1** and **bcode2** it says   
   “The library determines the name and purpose of this code and the code's definition.”

* **scroll down** to the field **marc**\_**type**\_code field. This not related to our query, but worth mentioning about the documentation.

A very limited number of fields provide the possible values for a field in the detailed view.   
The documentation will list valid values in a little chart embedded in the page like this:

Graphical user interface, text, application

Description automatically generated

When you see presentations like this, the codes are *not* found in a database table.

* **Scroll further down the list of tables** related to Bib. They’re in alphabetical order. Eventually you should see two tables, **user\_defined\_bcode1\_myuser** and **user\_defined\_bcode2\_myuser**.

The so-called “user” in the table name refers to the Sierra customer, which in our case means Watson Library! Here are the descriptions as provided in the documentation.

Graphical user interface, application

Description automatically generated

There are really only two fields we are concerned about -- the field that has the code value and the field that spells out the full code description.

It appears that in each table “name” field contains the description. Also, the “code” fields are the ones used to represent the field.

These codes are defined by the library to use as fields describing bib records. The documentation doesn’t tell us how Watson uses the fields. However, we have some sample data from our query that gives us some clues.

Let’s continue with our research...

Review what we know so far based on the sierraDNA documentation about Bibs:

|  |  |  |
| --- | --- | --- |
| bib\_view field | Characteristic | Defined in |
| **language\_code** | standard MARC 21 language  e.g., ‘ger’ | ?? |
| **country\_code** | standard MARC 21 country  e.g., ‘gw’ | ?? |
| **bcode1** | library defined code  e.g., ‘m’ | The **table user\_defined\_bcode1\_myuser**  where  the “code” field repeats the code on bcode1  the “name” field has the meaning / description |
| **bcode2** | library defined code  e.g., ‘a’ | The table **user\_defined\_bcode2\_myuser**  where  the “code” field repeats the code on bcode2  the “name” field has the meaning /description |

Although we might assume there is a table for countries and languages, we’re still not sure about where to get descriptions for the language\_code and country\_code based on what we found so far.

* Scroll down to the bottom of the Bib detail to find a link to “**ERD view**.”
* Click the **ERD View** link.

Graphical user interface, application

Description automatically generated

* The **ERD View** provides a high-level diagram showing the related tables.

Diagram

Description automatically generated

COUNTRY CODE

Following the lines leading off the bib\_view table country\_code field we learn that

this diagram tells us that the details about the country\_code may be found in the table named “**country\_property\_myuser**”

LANGUAGE CODE

Similarly, the lines leading off the bib\_view table language\_code field,

we find it tells us that the details may be found in the tables named “**language\_property\_myuser**”

Let’s update what we know so far about the situation.

We found four fields on the bib\_view table store codes. The sierraDNA documentation tells us characteristics about the fields and the diagram illustrates field to table relationship(s).

|  |  |  |
| --- | --- | --- |
| bib\_view field | Characteristic | Related table and comment |
| **language\_code** | standard MARC 21 language  e.g., ‘ger’ | **language\_property\_myuser**  the “code” field on this table matches the code on bib\_view.language\_code  the “name” field has the meaning / description |
| **country\_code** | standard MARC 21 country  e.g., ‘gw’ | **country\_property\_myuser**  the “code” field on this table matches the code on bib\_view.country\_code  the “name” field has the meaning / description |
| **bcode1** | library defined code  e.g., ‘m’ | The **table user\_defined\_bcode1\_myuser**  the “code” field matches the code on bcode1  the “name” field has the meaning / description |
| **bcode2** | library defined code  e.g., ‘a’ | The **user\_defined\_bcode2\_myuser**  the “code” field matches the code on bcode2  the “name” field has the meaning /description |

Alright, now we know the connection between the bib\_view table and four other tables.

MAKE A DIAGRAM

It helps to make your own diagram of the related tables, including relevant fields, and to include sample data in the diagram. You might want to create a sketch for yourself.

As an example, on the next page we made a variation of the ERD View where we added some of the field names in each table and sample data from a known record we got from our original query.

On the left is the bib\_view table, several fields and their values are listed.

On the right we have the four tables we determined are related, and their values.

The ‘>>’ symbols in the middle of the diagram show exactly which fields on the bib\_view are related to particular fields on the related tables.

In this case all the bib\_view code fields match up with a field named “code” on the related table. In addition, all of the fields that provide the descriptive text about the codes are named “name.”  
  
 **Related tables**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **bib\_view** | |  | language\_property\_myuser | |
| language\_code: | ger | >> | code: | ger |
|  |  |  | name: | German |
| record\_type\_code: | b |  |  |  |
|  |  |  | country\_property\_myuser | |
| country\_code: | gw | >> | code: | gw |
|  |  |  | name: | Germany |
| record\_num: | 1142992 |  |  |  |
|  |  |  | user\_defined\_bcode1\_myuser | |
| bcode1: | m | >> | code: | m |
|  |  |  | name: | MONOGRAPH |
| cataloging\_date\_gmt: | 1992-06-22 00:00:00-04 |  |  |  |
|  |  |  | user\_defined\_bcode2\_myuser | |
| bcode2: | a | >> | code: | a |
|  |  |  | name: | Text |

If you are familiar with cataloging:

* the value ‘MONOGRAPH’ (code ‘m’) should be recognizable as the bibliographic level
* and the value ‘Text’ (code ‘a’) should be recognizable as the material type.

You don’t necessarily need to make a detailed diagram for yourself. It is a good idea, however, to have a sample record in mind. That way you can verify you are looking at the correct information.

Many tables in Sierra have the same sort of code fields, whether it is the patron information, orders, vendors, etcetera. You can follow the same process that was just described to research fields and table relationships for any situation.

Now that we know the related fields and tables, we are ready to write SQL queries that bring all the pieces together. Next, we will gradually build up a complete, five table query.

## Joining tables

First, let’s write a SQL query to retrieve the full name of the country for the sample record with

record\_num = 1142992.

Below, we select the fields “record\_num” and “name” FROM two tables, “bib\_view” and “country\_property\_myuser”. The criteria in the WHERE statement identifies the

bib record number. In addition the WHERE clause needs to tell the system how to match the tables based on the values in the “code” fields being equal.  
  
CREATE THE QUERY JOINING TWO TABLES

SELECT

record\_num,

name

FROM

sierra\_view.bib\_view,

sierra\_view.country\_property\_myuser

WHERE

record\_num=1142992

AND sierra\_view.bib\_view.**country\_code** = sierra\_view.country\_property\_myuser.**code**

The last line of the query joins the tables on the fields that we determined they have in common.

EXPAND THE QUERY TO INCLUDE LANGUAGE (THREE TABLES)

You may expand on what we’ve developed so far by bringing in the second related table. You can do that by including the table name in the FROM clause and another line in the WHERE statement that tells the system how to join the new table.

Since the terms ‘code’ and ‘name’ are used in all these tables, you have to help the system out by removing any ambiguity. Remove ambiguity simply by adding the table name as the prefix every time the field names are mentioned in the query.

Another thing you ought to do is assign meaningful aliases to the fields instead of the vague label ‘name.’ As a reminder, you use the syntax **fieldname AS alias,** as shown below to create an alias.

Here is the previous query expanded to include the language. The fields are given appropriate aliases in the SELECT statement. Revised or additional lines have been highlighted.

SELECT

record\_num,

**sierra\_view.country\_property\_myuser.name as country,**

**sierra\_view.language\_property\_myuser.name as language**

FROM

sierra\_view.bib\_view,

sierra\_view.country\_property\_myuser,

**sierra\_view.language\_property\_myuser**

WHERE

record\_num=1142992

**AND sierra\_view.bib\_view.country\_code = sierra\_view.country\_property\_myuser.code**

**AND sierra\_view.bib\_view.language\_code = sierra\_view.language\_property\_myuser.code**

EXPAND THE QUERY TO INCLUDE BCODE1 (FOUR TABLES)

Continuing with our same example, we add another table. Here is the previous query expanded to include the bcode1 (bib level). The fields are given appropriate aliases in the SELECT statement. Revised or additional lines have been highlighted.

SELECT

record\_num,

sierra\_view.country\_property\_myuser.**name** **as** country,

sierra\_view.language\_property\_myuser.**name** **as** **language,**

**sierra\_view.user\_defined\_bcode1\_myuser.name as biblevel**

FROM

sierra\_view.bib\_view,

sierra\_view.country\_property\_myuser,

sierra\_view.language\_property\_myuser**,**

**sierra\_view.user\_defined\_bcode1\_myuser**

WHERE

record\_num=1142992

AND sierra\_view.bib\_view.country\_code = sierra\_view.country\_property\_myuser.code

AND sierra\_view.bib\_view.language\_code = sierra\_view.language\_property\_myuser.code

**AND sierra\_view.bib\_view.bcode1 = sierra\_view.user\_defined\_bcode1\_myuser.code**

EXPAND THE QUERY TO INCLUDE BCODE2 (FIVE TABLES)  
Here we add the fifth table to our query. Revised or additional lines have been highlighted.

SELECT

record\_num,

sierra\_view.country\_property\_myuser.name as country,

sierra\_view.language\_property\_myuser.name as language,

sierra\_view.user\_defined\_bcode1\_myuser.name as biblevel**,**

**sierra\_view.user\_defined\_bcode2\_myuser.name as mattype,**

FROM

sierra\_view.bib\_view,sierra\_view.country\_property\_myuser,sierra\_view.language\_property\_myuser,

sierra\_view.user\_defined\_bcode1\_myuser**,**

**sierra\_view.user\_defined\_bcode2\_myuser**

WHERE

record\_num=1142992

AND sierra\_view.bib\_view.country\_code = sierra\_view.country\_property\_myuser.code

AND sierra\_view.bib\_view.language\_code = sierra\_view.language\_property\_myuser.code

AND sierra\_view.bib\_view.bcode1 = sierra\_view.user\_defined\_bcode1\_myuser.code

**AND sierra\_view.bib\_view.bcode2 = sierra\_view.user\_defined\_bcode2\_myuser.code**

MODIFY THE QUERY TO INCLUDE ALIASES FOR FIELDS AND TABLES (*OPTIONAL*)

Some programmers prefer to shorten the verbose field names by assigning aliases to the table names. Here is a revised script in which all tables are assigned a single letter alias in the FROM clause. Then each field is referenced by the table alias.

-- fields and tables with alias version

SELECT record\_num,  
l.name as language,  
v.name as biblevel,  
m.name as mattype,

c.name as pub\_place

FROM

sierra\_view.bib\_view b,

sierra\_view.language\_property\_myuser l ,

sierra\_view.user\_defined\_bcode1\_myuser v,

sierra\_view.user\_defined\_bcode2\_myuser m ,

sierra\_view.country\_property\_myuser c

WHERE

record\_num=1142992 AND

b.language\_code = l.code AND

b.bcode1 = v.code AND

b.bcode2 = m.code AND

b.country\_code = c.code

The alias may be more than one character if you prefer.

This approach of using table and field aliases is optional. If you find it confusing, use the table name prefix method in the query on the previous page.

The query output from our five table query provides the labels we chose (instead of language\_code, bcode1, bcode2, and country\_code) and the full text for each record rather than ‘ger’,’m’,’a’, and ‘gw’.

This is the query output:

A picture containing text

Description automatically generated

The descriptive text improves readability of the outut. Retrieving descriptive text, either in addition to or instead of code values, is a very typical use of joining tables.

## Rules for joining tables

We saw that we matched the bib\_view and related tables based on a common value between fields on the bib\_view table and fields on the related tables. In general, the following rules apply when joining tables in SQL:

* The value in the field of one table must match the value in the field of the other table. This commonality links the tables.
* The data type in the columns on both tables must be the same. Examples of data type are integer, date, and character. If the data type is character, both fields must be the same length.
* You do not need to include the common field(s) that you use to join the tables in the SELECT statement of your query.
* The name of the column from one table can be different from the name of the matching column on the other table.
* If the name of a field on both tables is the same, it is important to prefix the field name with the table name or table alias to avoid ambiguity.

## Joining tables with a WHERE clause

As we have seen, you may join tables by including criteria statements in a WHERE clause that specify which fields to match.

Let’s try a different example, this time relating to patrons.

**NEW SCENARIO**

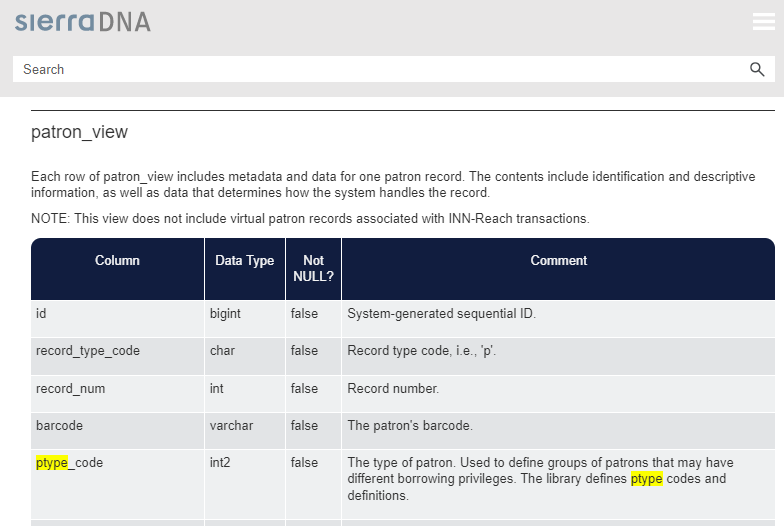
You can look up patron p102999a, and you would find they have a Patron Type of 20. The question is, what does the code 20 mean? How do you output the descriptive text in addition to, or in place of the code?

You can approach this question using the same steps we followed in the previous example with Bib related fields and tables.

1. **IDENTIFY THE RELATED TABLES AND THEIR COMMON FIELDS**

You can use sierraDNA to view the fields on any table. There you will see that the **patron\_view** table and **ptype\_property\_myuser** table have a certain field in common.

* go to the sierraDNA site
* click **Entities**
* click **Patron**
* click **Detailed** **View**
* **scroll down to patron\_view**



The list of fields on patron\_view includes a field named “ptype\_code” which the comment indicates is

the type of patron.

Next find where ptype\_codes are defined.

* scroll down the list of other patron related tables
* find **ptype\_property\_myuser**
* notice the field named “**value**” has a comment that says it is the patron type code
* the field named “**name**” contains the patron type name/description

Table

Description automatically generated

1. **ADD THE FIELD YOU WANT IN THE SELECT STATEMENT AND BOTH TABLE NAMES   
   TO THE QUERY’S FROM CLAUSE**

When you are joining tables using a WHERE clause, you must include both table names in the FROM  
 clause so the query knows where/how it will JOIN the tables.

SELECT  
record\_num, ptype\_code, **name**  
FROM

**sierra\_view.patron\_view, sierra\_view.ptype\_property\_myuser**

1. **ASSIGN TABLE ALIASES**

Optionally, give the tables short alias names to make referring to them easier.

FROM

sierra\_view.patron\_view **P**, sierra\_view.ptype\_property\_myuser **N**

1. **JOIN THE TABLES ON THEIR COMMON FIELD**

WHERE

P.ptype\_code = N.value

Here is a complete script that retrieves the name that corresponds to the patron type code.

The query joins the ptype\_property\_myuser lookup table and the patron\_view table.

Table

Description automatically generated with medium confidence

The following statements mean the same thing. The latter takes advantage of the table alias:

* sierra\_view.patron\_view.**ptype\_code** = sierra\_view.ptype\_property\_myuser.**value**
* P.**ptype\_code** = N.**value**

Optionally, edit the query by assigning an alias for the field “name” such as “patron\_type.”

## Joining tables with the JOIN statement

The second technique for combining tables is to use a “join” statement INSTEAD OF THE WHERE CLAUSE. Here is the syntax pattern.

SELECT field1, field2 ... fieldN FROM table1

**JOIN** table2 **on** table1.fieldname = table2.fieldname

When using this technique, you do not list both tables in the FROM clause.   
You can assign an alias and use the alias in the JOIN statement.

Here is a revision of our patron\_type query using the JOIN statement syntax.

Table

Description automatically generated

The results are the same whether you use a WHERE clause or JOIN statement. [[2]](#footnote-2)

**Using the JOIN syntax is the preferred method.**

### Examples

Here are more examples of joining pairs of tables.

**Exampe 1a: using JOIN statement**Watson uses the field pcode1 on the patron\_view table to identify the Institution.

SELECT

record\_num, barcode, pcode1, name as Institution

FROM

sierra\_view.patron\_view P

JOIN sierra\_view.user\_defined\_pcode1\_myuser C on P.pcode1 = C.code

WHERE

C.name like '%Museum%' and barcode like '2062%14'

**Example 1b: using WHERE clause**

SELECT

record\_num, barcode, pcode1, name as Institution

FROM

sierra\_view.patron\_view P,sierra\_view.user\_defined\_pcode1\_myuser C

WHERE

**P.pcode1 = C.code**

and

C.name like '%Museum%' and barcode like '2062%14'

Table

Description automatically generated

*After considering this output, you might consider enhancing the query by replacing pcode1 with the descriptive text.*

**Example 2 using JOIN statement**

Watson uses the field pcode3 on the patron\_view table to identify Departments in the Museum.

SELECT

record\_num,

barcode,

pcode3,

name as Department

FROM

sierra\_view.patron\_view P

JOIN sierra\_view.user\_defined\_pcode3\_myuser C on P.pcode3 =

cast((C.code) as int)

WHERE

barcode not like '2062%'and pcode3 != 1 and barcode like '%321'

*This is rare, but the joined fields are different data types. One is an integer and the other is a character. The statement “cast((C.code) as int)” means treat the code as an integer.*

Query output showing the department description that corresponds to pcode3:

Table

Description automatically generated

## Joining Data Tables

We saw that many tables provide descriptive text for codes. The purpose of those tables is not to store information about a particular thing, like a bib, item, hold, order, etc. Other tables store that data, and so we will refer to them as “data tables.” Data tables often have a lot of fields and the information from several data tables can be combined to get a complete picture of, let’s say a patron. Joining these data tables typically involves using the record number or a system generated id number.

### bib\_view and varfield\_view

The table **varfield\_view** stores the MARC tags, indicators, and subfields related to bib records. Each row in the table represents a tag, and the value is stored in a field named “field\_content.” Here is an example of a two-table join.

Table

Description automatically generated

It is normal for some values in the field\_content to contain the subfield delimiters (|a, |b, |c, etc.) as shown in the query output above.

### bib\_view, item\_view, and bib\_record\_item\_record\_link

If we want to list all the items associated with a bib record one problem is that the item record does not contain the bib id number. The bib record itself does not have a list of associated items.

The database has a table that serves to link the bib and item tables. That table is named

“**bib\_record\_item\_record\_link**.”

You might picture the relationship between these three tables as illustrated in the diagram below. The diagram is similar to what you will find in sierraDNA ERD View. We added sample values for some fields which makes it a little more understandable.

The linking table is in the middle. The “item\_record\_id” field on the linking table matches the “id” field on the item\_view (at left; note the content of the fields are the same, but the field names are different), and the “bib\_record\_id” on the linking table matches the “id” field on the bib\_view (at right; again, the content of the fields match, but the field names are different).

|  |  |  |
| --- | --- | --- |
| **item\_view** | **bib\_record\_item\_record\_link** | **bib\_view** |
| id: **1221474** < - - - - - - - - - - | item\_record\_id: **1221474** |  |
| item\_loc: s  barcode: 30620001142719 | bib\_record\_id: **1115022** - - - - - - - - - - > | id: **1115022**  Title: Art in America |

*Relationship diagram*

NOTE: If there aren’t any items linked to a bib record, there will not be a bib\_record\_item\_record\_link.

The following query demonstrates three table joins, and outputs fields from item\_view and bib\_view. It identifies bibs with items.

Table

Description automatically generated

A specific bib record was used for the query. That bib record is linked to ten items.

### hold, patron\_record\_fullname, item\_view, and bib\_record\_item\_record\_link

We looked at a query for holds in Part One, Exercise 10. This is the output from that Exercise:

Table

Description automatically generated

The hold query may be enhanced by retrieving the item barcode, patron name and bib\_id from other tables. A four-table join is needed. Here is the revised query and output.

**SELECT**

barcode,h.id **AS** hold\_id,id2reckey(h.record\_id) **AS** item\_id,

placed\_gmt **AS** request\_datetime,id2reckey(h.patron\_record\_id) **AS** patron\_id,

last\_name || ', ' || left(first\_name,1) **AS** patronname,

id2reckey(k.bib\_record\_id) **AS** bib\_id

**FROM**

sierra\_view.hold h

**JOIN** sierra\_view.patron\_record\_fullname f **on**

h.patron\_record\_id = f.patron\_record\_id

**JOIN** sierra\_view.item\_view i **on** h.record\_id = i.id

**JOIN** sierra\_view.bib\_record\_item\_record\_link k **on** i.id = k.item\_record\_id

**WHERE** i.location\_code = 'off' **and** placed\_gmt > '2022-12-22'

**ORDER BY** hold\_id

*Off-site holds. Detail query using JOIN statements to connect four tables*

Revised data output:

Table

Description automatically generated

## Tables often used in queries

At this point you’ve been introduced to several tables and the type of data they contain. You know how to use pgAdmin to explore the contents of the tables. Here is a reminder of some of the more useful tables.

|  |  |
| --- | --- |
| **bib\_view** | **item\_view** |
| **varfield\_view** | **phrase\_entry** |
| **bib\_record\_location** | **location** |
| **bib\_record\_property** | **patron\_record\_fullname** |
| **bib\_record\_item\_record\_link** | **user\_defined\_<x>\_myuser** |
| **patron\_view** | **hold** |

# Left Joins

There are scenarios when using the standard **Join** statement results in **excluding some data**.

Let’s look at an example when you need a different join statement, a “left join” statement.

First we will look at how to get the standard join figured out in the first place.

Below you see a simple query that retrieves three fields from one table, the item\_view table.

Graphical user interface

Description automatically generated with medium confidence

The data output provides an id of the last patron who checked out each item.

In the first two rows, the value of “last\_patron\_record\_metadata\_id” is **[null]** indicating those items were never checked out.

The last two items in the data output display a “last\_patron\_record\_metadata\_id” value.

Let’s suppose we want to retrieve the names of patrons who last checked out the items.

STEPS TO FINDING THE PATRON’S NAME

If you look at the **item\_view** information in sierraDNA, it tells you the table **record\_metadata** connects to the system generated id, “last\_patron\_record\_metadata\_id.”

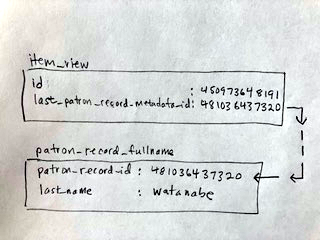
1. The item\_view description in sierraDNA says the last\_patron\_record\_metadata\_id is a “Foreign key to **record\_metadata** (for the last patron record).”
2. The **record\_metadata** table stores ids of bibs, items, patrons etc. By the name of the field “last\_patron\_record\_metatdata\_id” we can be confident that the value 481036437320 is a patron’s system generated id.
3. If you use sierraDNA to browse the patron tables, you discover the patron’s name is included in a table named **patron\_record\_fullname**.
4. You could run a quick SQL query in pgAdmin for patron\_record\_fullname with an id from the preceding example, 481036437320:

Graphical user interface, text, application, email

Description automatically generated

We know the last\_patron\_record\_metadata\_id is a patron id, so we should be able to matchup the tables based on that id. A diagram to illustrate the relationship between the tables may help.

You might sketch the relationship with sample data like this:



Now we know our standard join statement, not including table aliases. It should say:

...

FROM  
sierra\_view.item\_view  
join sierra\_view.patron\_record\_fullname on

sierra\_view.item\_view.last\_patron\_record\_metadata\_id =

sierra\_view.patron\_record\_fullname.patron\_record\_id

Here then is a query that uses a standard join between these two tables on their matching fields. The query assigns the tables aliases (‘i’ and ‘f’) for convenience. In the SELECT clause, the field named ‘id’ is given the table prefix ‘i.’ to avoid ambiguity.

Graphical user interface, text, application

Description automatically generated

Analyze the result

The barcodes in rows 1 and 2 of this result are the same ones in rows 3 and 4 of the original query output (30620008141474 and 30620007519175). The query returned the patron name for rows Watanabe and Kruglov.

The problem is that in the original query’s output the barcodes in rows 1 and 2 **are now missing from the output** of the joined table query (30620000000025 and 30620008643271).

Instead, the system returned two other rows that it found which have items with a previous checkout.

The item\_view table had [null] values for last\_patron\_record\_metadata\_ids. There wasn’t any patron record to match. Therefore, the first two lines got dropped from the results!

Assume we *intended to get all the items from item\_view*. Also, we want to retrieve the names of the last patrons to checkout the item AND if there is no checkout, then we want to keep the item\_view record anyway.

Solution: left join

In this example you might literally picture the item\_view table (the one you want all records from) as being on the left side of the equal sign in an equation. Whereas the patron\_record\_fullname table, which may or may not be matched, is on the right of the equal sign of a WHERE statement:

The SQL term we use to indicate we want all records on the left is “**LEFT JOIN.”**

### left join syntax

As a reminder, we use a “join” statement INSTEAD OF THE WHERE CLAUSE. Here is the syntax pattern again, this time with the term “LEFT” preceding the term JOIN.

SELECT field1, field2 ... fieldN FROM table1

**LEFT JOIN** table2 **on** table1.fieldname = table2.fieldname

When using this technique, you do not list both tables in the FROM clause.   
You can assign an alias and use the alias in the JOIN statement.

Here is the revised query using **left join** (line 6 and 7) instead of the standard join:

Graphical user interface, text, application

Description automatically generated

Success! Now we have all four of the original rows from the item\_view table. The first two rows in the output show the last\_patron\_record\_metadata\_id and last\_name values as [null], which is correct.

The original rows 1 & 2 were not dropped from the output.

Lines 3 and 4 include the last name of the patron which checked out the item.

## Left join examples

### Patrons and patron\_addresses

Here is an example of using a LEFT JOIN between patrons and patron\_addresses.

SELECT 'p' || record\_num || 'a' as patronid,addr1

FROM

sierra\_view.patron\_view p

LEFT JOIN sierra\_view.patron\_record\_address a

ON p.id = a.patron\_record\_id

WHERE

ptype\_code = 6

LIMIT 10

Table

Description automatically generated

Notes: The patron on line 3 does not have an address, but because the ‘left join’ statement is used the record was not excluded from the output.

Patrons on line 1 and 8 have the city, state, and zip code all in the addr1 field. This should be corrected. There are separate fields for city, state, and zip code.

### Bibs and items

Here is another typical example of using a LEFT JOIN connecting bib\_view, bib\_record\_item\_record\_link and item\_view.

SELECT

b.record\_num as bibid,title,i.record\_num as itemid, barcode

FROM

sierra\_view.bib\_view b

LEFT JOIN sierra\_view.bib\_record\_item\_record\_link k

ON b.id = k.bib\_record\_id

LEFT JOIN sierra\_view.item\_view i

ON i.id = k.item\_record\_id

WHERE

title like 'Burea%and%'

Graphical user interface, text, application, email

Description automatically generated

Notes: The bib records on lines 2 and 4 do not have items attached, but because the ‘left join’ statement is used the records were not excluded from the output.

If you do not include a left join between item\_view and bib\_record\_item\_view link lines 2 and 4 would be excluded from the output.

Electronic Resources do not normally have an item with a barcode, so this is correct.

Bib id 1048954 on lines 5 and 6 is listed twice, one row for each item.

# Further Study

You have completed a basic, introductory level course on SQL queries with the Sierra database using pgAdmin.

Here is a list of several resources you can use if you find you require something more sophisticated than a basic query.

* See the Appendix of this document for a few examples of slightly more complex SQL queries we use in Watson.
* If you have access to the Systems folders on the shared drive, you can find examples of complex SQL queries in the course materials from the Innovative SQL Workshop and from IUG Conferences.
  + SY\_Systems > SY\_Hardware and Software > Innovative > Sierra > SierraDNA SQL
    - Adv\_SQL\_Workshop\_Queries.txt
    - Sierra\_SQL\_Workshop\_2.pdf
    - SierraAdvancedSQLcourse.docx
    - IUG 2017 – Sierra Direct SQL Access – Davidson and Matta.pptx
    - IUG 2021 – SQL – Jeremy Goldstein.pptx
    - WILIUG\_Sierra\_Direct\_SQL\_Access\_101\_PPT\_061214.pdf
* If you are able to attend a follow up “SQL Jam” hands-on workshop, you will be able to try some multi-table join exercises.

# Appendix

## Some SQL Queries used at Watson Library

### Address checker

SELECT DISTINCT

v.id as system\_id,

checkout\_total,

'out',

checkout\_count,

to\_char(activity\_gmt, 'YYYY') as recency,

-- searchable id, where a is a wildcard replacing the check digit

'.p' || v.record\_num || 'a' as patron\_rec\_id,

UPPER(first\_name) as first\_name,

UPPER(last\_name) as last\_name,

CASE

WHEN (addr2 ISNULL AND city ISNULL AND region ISNULL and

postal\_code ISNULL)

THEN 'ADDRESS NEEDS UPDATE'

WHEN (LENGTH(addr2) > 2 AND LENGTH(city)=2) THEN UPPER(addr2)

WHEN (city = 'NY') THEN 'NEW YORK'

WHEN (city ISNULL) THEN 'ADDRESS NEEDS CITY'

ELSE

UPPER(city)

END as city,

CASE

WHEN region > '' THEN regexp\_replace(region, '\.', '', 'g')

WHEN (LENGTH(city)=2 AND addr2 >'') THEN UPPER(city)

WHEN city = 'New York' THEN 'NY'

ELSE

UPPER(region)

END AS region,

CASE

WHEN (country ISNULL) THEN 'United States'

ELSE

country

END AS country,

postal\_code,

-- calculate the status

CASE

WHEN expiration\_date\_gmt isnull THEN 'ACTIVE'

WHEN expiration\_date\_gmt > NOW() THEN 'ACTIVE'

ELSE

'EXPIRED'

END as status

FROM

sierra\_view.patron\_view v

JOIN sierra\_view.patron\_record\_fullname n

ON v.id = n.patron\_record\_id

JOIN sierra\_view.user\_defined\_pcode1\_myuser q

ON v.pcode1 = q.code

JOIN sierra\_view.user\_defined\_pcode2\_myuser r

ON v.pcode2 = r.code

JOIN sierra\_view.record\_metadata m

ON v.id = m.id

-- some records, might not have a patron address

LEFT JOIN sierra\_view.patron\_record\_address a

ON v.id = a.patron\_record\_id

WHERE

v.ptype\_code in ('6')

AND m.record\_type\_code='p'

ORDER BY recency

limit 30000

Table

Description automatically generated …

…Table

Description automatically generated…

…Table

Description automatically generated

### Items checked out from book cage

SELECT

i.record\_num as item\_record\_num,

i.barcode as item\_barcode,

i.record\_creation\_date\_gmt as item\_created,

b.record\_num as b\_record\_num,

b.country\_code,

b.language\_code,

b.title,

checkout\_gmt,

p.record\_num as patron\_record\_num,

p.id as patron\_metadata\_id,

i.last\_checkout\_gmt,

i.last\_patron\_record\_metadata\_id

FROM

sierra\_view.checkout c,

sierra\_view.patron\_view p,

sierra\_view.item\_view i,

sierra\_view.bib\_view b,

sierra\_view.bib\_record\_item\_record\_link k

WHERE

c.item\_record\_id=i.id

and

c.patron\_record\_id=p.id

and

i.id=k.item\_record\_id

and

b.id=k.bib\_record\_id

and

i.location\_code='bc'

and

checkout\_gmt > '2022-12-20'

Table

Description automatically generated

### All patrons

SELECT

v.field\_content as barcode,

v.record\_num as recno,

CASE

WHEN COALESCE(to\_char(expiration\_date\_gmt, 'YYYY-MM-DD'), '')

> '2022-11-04'

THEN 'A'

WHEN COALESCE(to\_char(expiration\_date\_gmt, 'YYYY-MM-DD'), '')

= ''

THEN 'A'

ELSE 'XX'

END as stat,

CASE

WHEN ptype\_code = 6

THEN 'VR'

ELSE 'S'

END as ptype,

pcode1 as aff,

pcode2 as grp,

pcode3 as dept

FROM

sierra\_view.patron\_view p,

sierra\_view.varfield\_view v

WHERE

p.record\_num = v.record\_num

and v.record\_type\_code= 'p'

and v.varfield\_type\_code ='b'

Table

Description automatically generated

1. Cover generated by <https://orly.nanmu.me/> [↑](#footnote-ref-1)
2. There are situations where a so-called “LEFT JOIN” statement is required. This will be explained later. [↑](#footnote-ref-2)