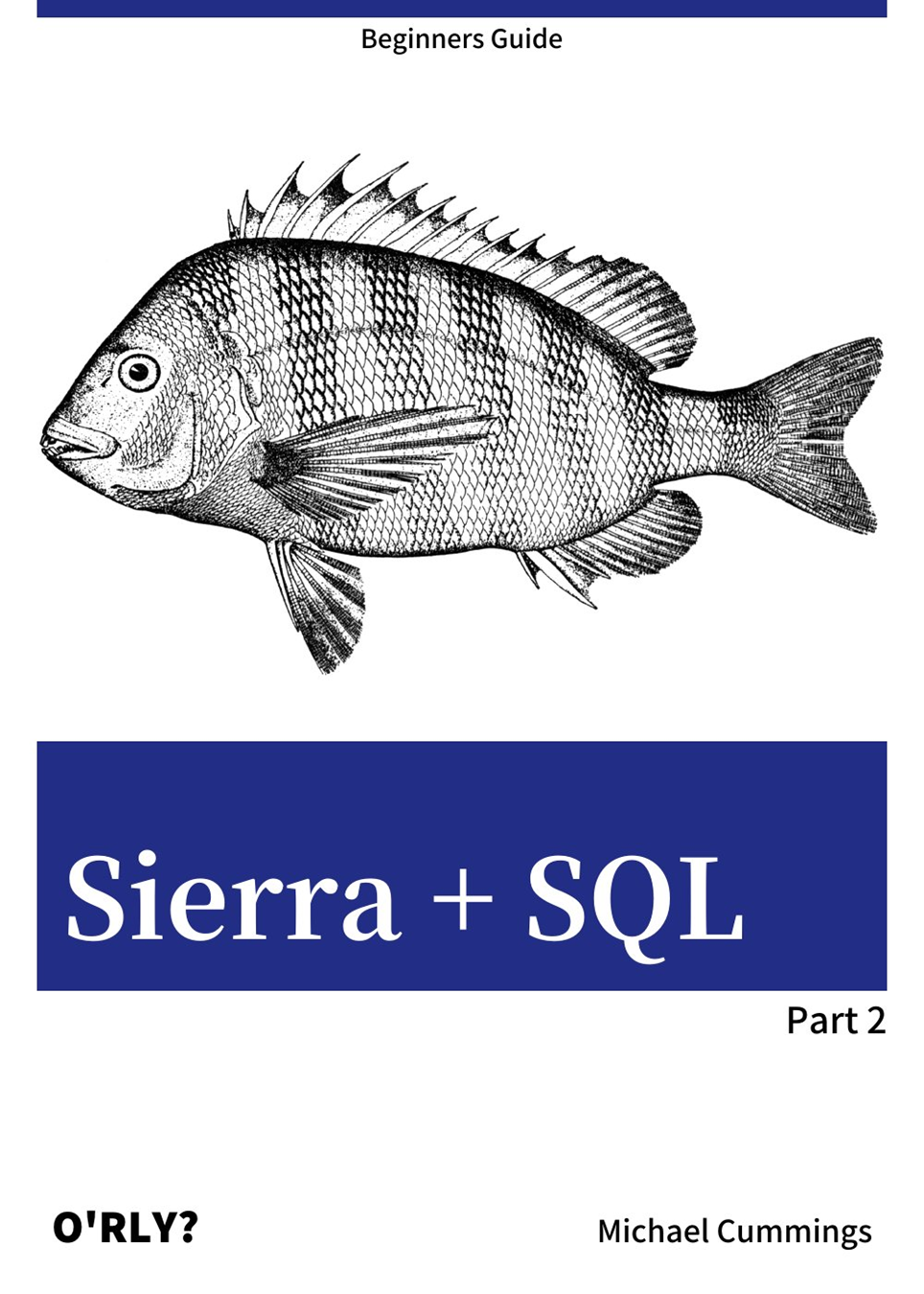
[[1]](#footnote-1)

**Part 1**

Edited by Carol Choi

Contents

[SQL course Introduction 6](#_Toc122609104)

[Course Objectives: Part One 6](#_Toc122609105)

[Acknowledgement 6](#_Toc122609106)

[Important Database Terminology 7](#_Toc122609107)

[Sierra database tables / views 8](#_Toc122609108)

[Views 8](#_Toc122609109)

[pgAdmin 9](#_Toc122609110)

[Install pgAdmin 9](#_Toc122609111)

[Navigate the pgAdmin tree down to the views 10](#_Toc122609112)

[List fields on a table 13](#_Toc122609113)

[View / Edit Data 14](#_Toc122609114)

[The Query Tool 15](#_Toc122609115)

[Query scripts and output 16](#_Toc122609116)

[Save a query 16](#_Toc122609117)

[Open a saved query 16](#_Toc122609118)

[Save / copy output 16](#_Toc122609119)

[Download output 16](#_Toc122609120)

[Copy query statement from MS Word to pgAdmin 16](#_Toc122609121)

[Basic SQL functions 17](#_Toc122609122)

[AS alias 18](#_Toc122609123)

[left(string,number) 18](#_Toc122609124)

[table alias 18](#_Toc122609125)

[like % 18](#_Toc122609126)

[Greater than operator 18](#_Toc122609127)

[not equal 18](#_Toc122609128)

[ORDER BY title 19](#_Toc122609129)

[LIMIT 19](#_Toc122609130)

[concat() function 20](#_Toc122609131)

[Comments in SQL 20](#_Toc122609132)

[Counting results 20](#_Toc122609133)

[id2reckey() function 21](#_Toc122609134)

[Handling errors 22](#_Toc122609135)

[Close completed queries 23](#_Toc122609136)

[Simultaneous Query windows warning 24](#_Toc122609137)

[The limit command 24](#_Toc122609138)

[Working with Date and Time fields 25](#_Toc122609139)

[Workshop 1 26](#_Toc122609140)

[SQL query exercises using a single table 26](#_Toc122609141)

[Exercise 1: bib\_view 27](#_Toc122609142)

[Exercise 2: varfield\_view 28](#_Toc122609143)

[Exercise 3: subfield\_view 29](#_Toc122609144)

[Exercise 4: phrase\_entry 30](#_Toc122609145)

[Convert id field to a record number id2reckey() 31](#_Toc122609146)

[Exercise 5: bib\_record\_location 33](#_Toc122609147)

[Exercise 6: item\_record\_property 34](#_Toc122609148)

[Exercise 7: patron\_view 35](#_Toc122609149)

[Exercise 8: user\_defined\_pcode1\_myuser 36](#_Toc122609150)

[Exercise 9: patron\_record\_address 37](#_Toc122609151)

[Exercise 10: hold 38](#_Toc122609152)

[Exercise 11: challenge 39](#_Toc122609153)

[What’s Next 40](#_Toc122609154)

[Use the Appendix as a resource 40](#_Toc122609155)

[Part Two of Sierra & SQL 40](#_Toc122609156)

[Appendix 41](#_Toc122609157)

[Intermediate / Advanced functions 41](#_Toc122609158)

[Regular expressions in SQL 41](#_Toc122609159)

[Convert a field type using cast() 41](#_Toc122609160)

[Find the aggregate value max() 41](#_Toc122609161)

[Group by 42](#_Toc122609162)

[Group by rollup 43](#_Toc122609163)

[filter() 44](#_Toc122609164)

[The HAVING clause 44](#_Toc122609165)

[Aggregate functions 45](#_Toc122609166)

[CASE statements / conditional statements 46](#_Toc122609167)

[array() and array\_to\_string() 47](#_Toc122609168)

[More functions 48](#_Toc122609169)

[Types of data tables 48](#_Toc122609170)

[List of values tables 48](#_Toc122609171)

[Descriptive data tables 48](#_Toc122609172)

[Transaction data tables 49](#_Toc122609173)

[Other data tables 49](#_Toc122609174)

[Trouble shooting common SQL errors 50](#_Toc122609175)

[Column does not exist 50](#_Toc122609176)

[Syntax error 51](#_Toc122609177)

[Missing table prefix 52](#_Toc122609178)

[Missing parameters in a function 52](#_Toc122609179)

[Incorrect or missing wildcard 53](#_Toc122609180)

[Solutions to Exercises 54](#_Toc122609181)

List of Figures

[Figure 1 Navigation pane open to iii 10](#_Toc122447143)

[Figure 2 Navigation pane open at Schemas 10](#_Toc122447144)

[Figure 3 Navigation pane open at sierra\_view 11](#_Toc122447145)

[Figure 4 Navigation pane open at Views 12](#_Toc122447146)

[Figure 5 Navigation pane open to bib\_view columns 13](#_Toc122447147)

[Figure 6 Table bib\_view context menu 14](#_Toc122447148)

[Figure 7 Output from View / Edit, last 100 rows of bib\_view 14](#_Toc122447149)

[Figure 8 Output from Query Tool query of bib\_view 15](#_Toc122447150)

[Figure 9 Output of a bib\_view query with functions 19](#_Toc122447151)

[Figure 10 id2reckey() output Example 1 21](#_Toc122447152)

[Figure 11 id2reckey() output Example 2 22](#_Toc122447153)

[Figure 12 Messages tab showing an error 23](#_Toc122447154)

[Figure 13 Query pane with Close icon highlighted 23](#_Toc122447155)

[Figure 14 Exercise 1 query output 27](#_Toc122447156)

[Figure 15 Exercise 2 query output 28](#_Toc122447157)

[Figure 16 Exercise 3 query output 29](#_Toc122447158)

[Figure 17 Exercise 4 query output 30](#_Toc122447159)

[Figure 18 Example of id2reckey() function 31](#_Toc122447160)

[Figure 19 id2reckey() output example2 32](#_Toc122447161)

[Figure 20 Exercise 5 query output 33](#_Toc122447162)

[Figure 21 Exercise 6 query output 34](#_Toc122447163)

[Figure 22 Exercise 7 query output 35](#_Toc122447164)

[Figure 23 Exercise 8 query output 36](#_Toc122447165)

[Figure 24 Exercise 9 query output 37](#_Toc122447166)

[Figure 25 Exercise 10 query output 38](#_Toc122447167)

[Figure 26 Output from a GROUP BY clause 42](#_Toc122447168)

[Figure 27 Group output with rollup total 43](#_Toc122447169)

[Figure 28 Output from a query with filter() 44](#_Toc122447170)

[Figure 29 Output after using a CASE condition 46](#_Toc122447171)

[Figure 30 Output using a string array for bib loc(s) 47](#_Toc122447172)

[Figure 31 Error: column does not exist 50](#_Toc122447173)

[Figure 32 Error: trailing comma in SELECT 51](#_Toc122447174)

[Figure 33 Error: missing commas between fields 51](#_Toc122447175)

[Figure 34 Error: missing table prefix sierra\_view. 52](#_Toc122447176)

[Figure 35 Error: function not recognized. Incomplete parameters 52](#_Toc122447177)

[Figure 36 Error: incorrect or missing wildcard syntax 53](#_Toc122447178)

# SQL course Introduction

The acronym SQL stands for Structured Query Language. Library staff may use SQL query as an alternate the Sierra staff client program or the Create Lists function. The results of an SQL query may be exported very similar to the data that library staff are able to produce using the Sierra Create Lists function. In fact, most of the Web Management Reports and Decision Center reports can be replicated with SQL.

There are several advantages SQL provides compared to Create Lists. These advantages include:

* More control over the level of detail in the data is exported
* More options for sorting the output and group the results
* The ability to apply aggregate functions and custom fields
  + Sum, average, minimum, maximum, top N percent, etc.
* No limit to the number of records exported
* The SQL query data may be incorporated into an automated workflow

Automating workflows makes SQL especially useful. A few examples of automated workflows we have set up at Watson that incorporate SQL queries are:

* Requests for off-site items are automatically identified and transferred to Clancy daily.
* Patrons who are issued new library cards are identified and their demographic profiles are transmitted to Mavin daily.
* Scripts are run periodically that list identifiers on bibliographic records (e.g., OCLC, ISBN) and then search external providers for digital content. The providers include Hathi Trust, Internet Archive, Google Books, and others.

## Course Objectives: Part One

This two-part course is intended for staff that do not have prior programming or scripting experience. The course provides a basic introduction to SQL.

Part One goals:

* Use the pgAdmin program to explore data and run queries
* Learn where different categories of Sierra data are stored
* Become comfortable with the SQLquery clauses (SELECT, FROM, WHERE, ORDER BY)
* Learn to use built-in SQL functions when searching for information in Sierra
* Know how to format the create and save SQL queries
* Know how to customize the output of the query and export the results

Part Two will build upon the basic skills the reader acquired. Examples of actual SQL queries used at Watson will be reviewed and resources for further study will be provided.

## Acknowledgement

Thank you for your interest in SQL. The Library Systems team is ready to help staff with any custom SQL queries. This course is merely being offered to provide an opportunity for people who have heard about SQL to learn something about it.

# Important Database Terminology

As mentioned earlier, the acronym “SQL” stands for structured query language. These are the terms or concepts describing the “structure” of our database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| addresses | | | | |
| address\_id (PK) | street | city | state | zip |
| **400192504** | 1000 Fifth Avenue | New York | NY | 10028 |
| **400192505** | 1 East 70th St | New York | NY | 10021 |
| **400192506** | 1071 Fifth Avenue | New York | NY | 10028 |
| **400192507** | 1220 Fifth Avenue | New York | NY | 10029 |
| **400186548** | 6th and Constitution Ave., NW | Washington | DC | 20565 |

* **Table**The main concept is “table” which contains several rows of information. An example of a table is “addresses.” Usually, table names are the plural form of a word.

Table 1 Sample table

* **Row / record**Each “row” in a table describes a particular “record”, for example a building address. The terms “row” and “record” are synonymous.
* **Field / column**The records are broken down into “fields” or “columns”, such as street address, city, state, zip.  
  The terms “field name” and “column” are synonymous.  
  Generally, a field name does not include spaces (underscores to separate words).
* **Primary key / id**  
  Each row in a table must be unique. A common way of ensuring uniqueness is to include a field containing a system-generated number. Typically, this field is named “id”, “record\_id” or something similar. We refer to the field as the “id”, the “key field”, or the “primary key.” The value of a primary key is never null, nor may it be modified.

The “(PK)” shown in the first column heading above (next to address\_id) is not part of that field’s name.It is there to indicate that this field is the primary key for the records.

More about this later. For now, just beware there are these system-generated id fields in almost every table you encounter in Sierra.

* **tablename.fieldname syntax**

When referring to a field, it is common to use the syntax tablename dot fieldname. For example, the full name of the zip field on the addresses table would be called “addresses.zip”

# Sierra database tables / views

For the purposes of SQL queries, we use the data in Sierra which has been stored in tables.

Although data in Sierra naturally lends itself to being represented in tables, no single table contains all the data that describes a patron, an item, a holding, or other concept. The preferred database term for concept is “entity.”

**Non-MARC data**Many of the tables contain records originating from non-MARC sources. Examples include electronic resource data, patron, item, and holding data, order, invoice, and vendor data, tables that store lists of values. These are represented across several database tables.

**MARC data**Our bibliographic catalog records in the MARC data format are stored in Sierra. This data is also extracted from the MARC format and stored across several database tables.

### Views

To represent entities like a bibliographic record, it is necessary to combine the data from multiple database tables. Sierra has several **pre-defined combinations of data tables** which are commonly referred to as a “view.” Therefore, rather than referring to a table name, sometimes you will refer to the name of the view.

Commonly used views are “bib\_view”, “patron\_view”, “item\_view.”

To explore the data that is stored in some of these tables or views, you will need to install a database query program.

Sierra is a **PostgreSQL** database[[2]](#footnote-2). We use a program called **pgAdmin** to create and run SQL queries.

# pgAdmin

After installing the pgAdmin program, you will use that program to explore data in Watson’s Sierra database. You will be using the version of SQL that works with both pgAdmin and a postGres database.[[3]](#footnote-3)

Note: The system requires a strong password of at least 8 characters. If you have an old, shorter password you must change your Sierra password.

## Install pgAdmin

Download pgAdmin from <https://www.pgadmin.org/download/pgadmin-4-windows/> by clicking any one of the package links on the page. Download the file with the extension ‘exe’

* Once downloaded, double-click the exe program file to begin the install. You may be prompted for a port number first. If so, enter 1032. You may have to restart pgAdmin to continue.
* Configure a **connection** for PostgreSQL with the following Connection Settings (under New Server/Register Server, or **Add New**). You can name the Connection Sierra.
* If you set a master password, use your Sierra password

**Connection**

* + Host: **your library db.iii.com**
  + Port: **1032**
  + Maintenance  
    database: **postgres**
  + User \* : **your Sierra user name**
  + Password \* : **your Sierra password**
  + **If there is a field for SSL mode, choose ‘Require’**

**\* Please note:**

* The user identified in the connection settings must have the Sierra assigned application “Sierra SQL Access” in Sierra Web Admin.
* You must either be on a VPN connection to query Sierra or run your query from a “white-listed address” – an address that has been pre-arranged with Innovative.

## Navigate the pgAdmin tree down to the views

Having installed pgAdmin you should now follow these instructions to familiarize yourself with the program and data.

* In the menu tree, double-click the name or click the expand icons next to **Servers** then **Databases** then **iii**

Graphical user interface, application, Word

Description automatically generated

Figure 1 Navigation pane open to iii

Ignore everything except Servers / Sierra / Databases(2) / **iii** / **Schemas(12)**

* Double-click the name or click the expand icon next to **Schemas (12)**

Graphical user interface, application, Word

Description automatically generated

Figure 2 Navigation pane open at Schemas

Ignore everything except the last entry under Schemas(12) / **sierra\_view**

* Double-click the name or click the expand icon next to **sierra\_view**

Graphical user interface

Description automatically generated with medium confidence

Figure 3 Navigation pane open at sierra\_view

Ignore everything except the last one, sierra\_view / **Views**

* Double click the name or click the expand icon next to **Views (360)**

Graphical user interface, text, application

Description automatically generated

Figure 4 Navigation pane open at Views

There are 360 tables and views you can use to run SQL queries. The ones with the suffix “\_view” are pre-defined combinations of fields from more than one table. We will focus on single table queries for now. The next time you open pgAdmin it may automatically open the navigation pane to this point.

## List fields on a table

Double-click the name or click the expand icon next to **bib\_view,** then click the expand icon next to **Columns (18)**

Graphical user interface, application, Word

Description automatically generated

Figure 5 Navigation pane open to bib\_view columns

With the bib\_view columns expanded, we can see there are 18 “fields” or “columns” on the bib\_view table.

Every table column has an associated type of allowed data. The types of data include integer, big integer, character, date or timestamp, numeric.

## View / Edit Data

We will use the “View / Edit” function to explore the data. The name of this selection is a misnomer since **you cannot edit the data**. Ignore menu options that indicate otherwise.

* Right-click on **bib\_view** to see a context menu.

Graphical user interface

Description automatically generated with medium confidence

Figure 6 Table bib\_view context menu

* From the context menu,select **View / Edit Data**
* Select **Last 100 Rows**

The right-hand side of the window has a Query pane above, and a pane at the bottom for Data output.

* Rearrange the size of the panes for a better view by clicking and dragging the divider between columns. Table

  Description automatically generated

Figure 7 Output from View / Edit, last 100 rows of bib\_view

* Notice the **Query** statement on the top pane.The asterisk character in the Query is a “wildcard” character meaning ALL fields.
* Click the **“x”** on the top, right corner of the query to close it

## The Query Tool

Next, we will try out the option named “Query Tool” instead of “View / Edit Data”

* Right-click **bib\_view** in the navigation pane again
* This time select **Query Tool**
* Copy / paste or type the following text into the Query pane:

SELECT

record\_num,

language\_code,

title,

cataloging\_date\_gmt

FROM

sierra\_view.bib\_view

limit 100

This SQL query will pull the first 100 records, similar to the View/Edit Data function.

Most SQL developers find that breaking up statements over multiple lines makes them easier to read and debug. You do not need to capitalize the words SELECT, FROM, WHERE, ORDER BY. However, it helps readability.

* Execute the query.   
  You can press **F5** to execute the query or click the run icon  above the query pane.

Graphical user interface, text, application

Description automatically generated

Figure 8 Output from Query Tool query of bib\_view

You may want to resize the width of the fields in the Data output tab.

## Query scripts and output

Those were two commonly used functions/ways to explore Sierra data. Now let’s take a look at how to save and retrieve your work.

### Save a query

Save a SQL query statement by clicking the diskette icon (located above the Query workspace, it says “Save File” when you hover over it). Navigate to a directory and save the file. The file extension should be **.sql**

### Open a saved query

You should be able to retrieve any of your previously saved SQL queries by clicking the folder icon above the Query pane (it says “Open File” when you hover over it).

### Save / copy output

You can click the top left corner of the Data output tab to select all the output cells. Click the selector/down arrow next to the copy icon to choose whether to copy with headers.

Then type CTRL/c to copy. Open an Excel spreadsheet and paste (CTRL/v) the text.

### Download output

You can download the output as a .csv file by clicking the download icon on the Data output tab (it says “Save results to file” when you hover over it) and then import the .csv file into Excel.

Copy query statement from MS Word to pgAdmin  
[TIP] You can copy query statements from this Word document and paste the text into pgAdmin. However, beware of the smart quotes characters in Word; In pgAdmin, smart quotes cause syntax errors. You must change the smart quotes to single quote characters.

# Basic SQL functions

Here are some useful terms related to the syntax of SQL:

* Function – one or a series of statements that perform a specific task; functions typically act upon a field named in parenthesis after the function name.  
  An example of a function is left(title, 40). The function name is “left.” Its task is to act on the title field, returning the specified number of characters. In this example, 40 characters.
* Clause – these add conditions to the function; they can be required or optional,  
   such as ORDER BY
* Operator – reserved word(s) or character(s) that define a condition in WHERE functions such as greater than “>” or equal to “=”
* Alias – this is exactly what you may think, an alternate way of referring to something. For example, the statement “cataloging\_date\_gmt as cat\_date” establishes “cat\_date” as an alternative to using the longer form.

Now let’s try a new query:

* Edit the original SQL query statement to look like the following revision.
* Take care to include punctuation and make sure the quotes are single quotes   
  (not smart quotes).

**SELECT**

id,

record\_num,

language\_code AS lang,

left(title,40),

cataloging\_date\_gmt AS cat\_date

**FROM**

sierra\_view.bib\_view b

**WHERE**

language\_code!='eng' and

title like 'A%' and

cataloging\_date\_gmt > '01-01-2022 00:00'

**ORDER BY** title

limit 20

* Execute the query (press F5) after making the revisions.

The statements introduced in the revised query are described below.

**id**

Nearly every table has a system-generated field named id or record\_id of type “big int” - something like 420908827361. This id is the **primary key** of the record.

**record\_num**

The record number does not include the last digit, nor does it include the prefix (such as “.b”, for bib numbers). The last digit, or “check digit” is a value the system calculates based on the other digits.

### AS alias

**language\_code AS lang**

The syntax “**AS** *alias*” is a way to change the field name displayed in the results to an alias

### left(string,number)

**left(title,40)**

“left()” is a function that returns only the first number of characters of the field in parenthesis, starting from the left; the number of characters is the number in parenthesis.

### table alias

**FROM sierra\_view.bib\_view b**

This statement establishes a very short alias for the table! We only need the alias, “b” if we refer to that table again.

The statements in the **WHERE clause** act as filters.

### like %

**title like ‘A%’**

The “like” operator is used for approximate matches. In this example, it means the title begins with capital A and the % is a wildcard meaning match any others that follow.

### Greater than operator

**cataloging\_date\_gmt > ‘01-01-2022 00:00’**

The operator “>” means greater than. The operator “<” means less than.

### not equal

**language\_code!=’eng’**

The operator “**!=**” means not equal. Use a single “**=**” to match equal.

### ORDER BY title

“**ORDER BY**” means sort. By default, sorting is from lowest value to highest. Alternatively, you could say “**ORDER BY** title asc”. To sort in descending order, you would say “**ORDER BY** title desc.” If you use an “**ORDER BY**” clause it should always be the last clause in the **SELECT** statement.

It is possible to sort data by a column that is not retrieved in the **SELECT** statement. You can sort by multiple columns by specifying the column names separated by commas.

### LIMIT

The **LIMIT** command allows you to control the number of rows a query will return.

Limits are important when you are developing queries. Include a limit number using a **WHERE** clause or you could inadvertently return thousands or even millions of rows of data!

A variant of your query may look something like the following. The **concat** function on line 3 in this example outputs a format of the bib that is searchable in Sierra.

Graphical user interface, text, application

Description automatically generated

Figure 9 Output of a bib\_view query with functions

You may need to adjust the column widths in the Data output.

### concat() function

Concatenate is a function that combines two or more strings together. You could concatenate the appropriate prefix, the record number, and the wildcard check digit character “a” using the SQL concat function. Alternatively, you may use the SQL operator ”||“ to combine strings. Both techniques retrieve a value that looks like a complete record number such as .b1000202a. Here is the syntax you would use:

**concat(**‘.b’,record\_num,’a’**) as** bib\_id**,**

**or**

‘.b’ || record\_num || ‘a’ as bib\_id,

**Alias**

Notice an alias for the title was not provided in the example statement. The statement just says “left(title,40)” and the output shows the field name is “left.” It could have written “**left(title,40) AS short\_title**” as an alias. The alias name of fields will appear in the output column heading.

### Comments in SQL

You can include comment lines in your query statement. A comment is notation that is not part of the code. Prefix the comment line with two hyphens.

--filename: varfield.sql  
--author : James Hendershaw  
SELECT \* from sierra\_view.varfield\_view   
WHERE record\_num = 2043861

### 

### Counting results

You can include a basic counter in your query statement.

**SELECT** count(\*) from sierra\_view.bib\_view   
**WHERE** lang= hun

### id2reckey() function

The **id2reckey()** function is a handy tool for converting those long id numbers into the relevant, more familiar record number. This works for bib, item, patron, order, etc. Here is an example:

SELECT

**id,record\_id,id2reckey(record\_id) as “bibid”**

FROM

sierra\_view.bib\_record

WHERE

cataloging\_date\_gmt > ‘2022-10-13’

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 10 id2reckey() output Example 1

* The value returned from the id2reckey() function has a letter prefix.
* The check digit is not included.

Another example of **id2reckey()**

SELECT

id2reckey(v.record\_id) as “bib\_id”,

marc\_tag, field\_content

FROM

sierra\_view.varfield\_view v

WHERE

record\_id=’420908909341’

order by marc\_tag

The query results are shown on the next page.

Table

Description automatically generated

Figure 11 id2reckey() output Example 2

### Handling errors

It’s certain that your query will have some kind of error as you develop your queries. Let’s edit the sample script to force it to produce an error.

* Edit the query. Change the statement that reads ‘order by title’ to **‘SORT BY** title**’**
* Press F5 to execute the revised query

The program will not recognize the command. You should see an error in the message tab:

Text

Description automatically generated

Figure 12 Messages tab showing an error

SQL will let you know what parts of a statement aren’t working. You can use the Error messages it provides to troubleshoot.

* Let’s go back to ‘**ORDER BY** title’
* Fix any other typos that may be present
* Press F5 to execute the query

[TIP] Additional tips for handling errors are provided in the Appendix of this document.

### 

### Close completed queries

It is important to close completed pgAdmin queries. This means you should close any query panes you have active once you finish the query and you are ready to move on to something else.

Graphical user interface, text, application, email

Description automatically generated

Figure 13 Query pane with Close icon highlighted

Click the “x” icon to close active query panes.

### Simultaneous Query windows warning

pgAdmin allows only a few simultaneous SQL query connections Each SQL query you open in the query tool uses a connection.

If you exceed the number of allowed connections, you will see an error message and you won’t be able to run any queries.

If this happens, close pgAdmin and relaunch the program; it should allow you to continue.

### The limit command

Earlier the Limit command was summarized. This subject is worth repeating. The Limit command allows you to control the number of rows a query will return.

Limits are important when you are developing queries. Include a limit number using a WHERE clause or you could inadvertently return thousands or even millions of rows of data!

Examples of limiting statements using ‘**limit**’, ‘**in**’, ‘**between**’, or ‘**NULL**’ that will help to contain the query results:

limit 500

where location\_code **=** ’off’

where checkout\_total **>** 1000

where expiration\_date\_gmt **>** ’2026-12-31 00:00’ and ptype\_code = 10

where expenditure **between** 1000 and 1500

where location\_code **in** (’off’,’s’,’ref’)

where location\_code = ’off’ **or** ’www’

where location\_code **is NULL**

You do not need to enclose numeric values in quotes.

### Working with Date and Time fields

There are functions you can use to manipulate columns that represent date and time. Here are three functions you will find useful.

#### Date\_part(), extract(), and to\_date().

The examples below assume we are given the date time value **2022-09-07 08:30:00-04**

**Query Statement Returns**

date\_part(‘year’,cataloging\_date\_gmt) 2022

extract(year from cataloging\_date\_gmt) 2022

extract(month from cataloging\_date\_gmt) 9

# Workshop 1

Now that you’ve had a chance to see different types of tables and what data they store. it’s time for you to write some SQL queries using the Query Tool. The following exercises will both use the functions you’ll recognize from above and familiarize you with some of the most used/main tables and entities used.

[TIP] Note: if you get stuck, check the [Trouble shooting common SQL errors](#_Trouble_shooting_common).

## SQL query exercises using a single table

* Close any query windows you may left open from the above exploration of tables
* Select **Tools > Query Tool** from the menu bar
* Just a reminder:
  + You can expand a table in the views to reveal the column names
  + Save each of your queries for later.
  + **Be sure to close the query when you’re finished before going on the next one.**
  + It would be helpful to navigate to the view in pgAdmin to see what fields are in each view and ensure you’re using the right name of them in your queries

### Exercise 1: bib\_view

|  |  |
| --- | --- |
| The bib\_view table has quite a few fields.  You can see the list of fields or ‘columns’ by expanding the table name **bib\_view** and the **Columns** in the navigation pane. | Graphical user interface, application, table  Description automatically generated |

Write a simple query. The SELECT clause should specify the fields listed below. The FROM clause should specify the Table named sierra\_view.bib\_view. The WHERE clause specifies we want records where the catalog date is after October 11, 2022 and the language code is equal to ‘spa’ (Spanish). The WHERE clause you should use is shown below.

[Tip]: Figure 9 is a similar query.

Fields: id, record\_num, cataloging\_date\_gmt as cat\_date, left(title,40) as title  
Table: sierra\_view.bib\_view  
Where: cataloging\_date\_gmt > ‘2022-10-01’ and language\_code = ’spa’  
Limit: 10

Text, table

Description automatically generated with medium confidence

Figure 14 Exercise 1 query output

*Your data output should appear like this screenshot. (You may need to widen the title column.)*

### Exercise 2: varfield\_view

The varfield\_view table contains all the MARC tags from bib records and variable fields from other records like patrons.

The column named “field\_content” has the variable field text. The field\_content includes subfield delimiters.

Save and Close your Exercise 1 query.

Write a new query to retrieve the variable fields for the given record number. As before, you will need to put the following specifications into the appropriate SQL clauses: SELECT, FROM, WHERE. This exercise also sorts the results. You add an ORDER BY clause at the end.

Fields: record\_id, record\_num, marc\_tag, field\_content   
Table: sierra\_view.varfield\_view  
Where: record\_num = 2043861  
Order by: marc\_tag

The query specifies a single record number, so you do not need to include a limit statement.

Graphical user interface, text, application, email

Description automatically generated

Figure 15 Exercise 2 query output

*Your data output should appear like this screenshot*

The delimiter markers (|a, |b, |c, etc) that appear in the field\_content look very strange but that is not an error.

### Exercise 3: subfield\_view

The subfield\_view table removes the delimiter markers from varfield\_view.field\_content, itemizing every subfield as a separate line.

Write a query to retrieve some of the MARC tags and subfields as specified below.

Fields: record\_num, marc\_tag, tag, content   
Table: sierra\_view.subfield\_view  
Where: record\_num = 2043861 and marc\_tag between ‘245’ and ‘299’  
Order by: marc\_tag, tag, display\_order

Notes:

* The subfield\_view table has a field named “display order” which can be used to sort subfields within the tag.
* The field named “tag” is actually the subfield.
* The marc\_tag field is defined as a character field, so you need to put quotes around in the WHERE clause even though the tag looks like a number.

Table

Description automatically generated

Figure 16 Exercise 3 query output

*Your data output should appear like this screenshot*

### Exercise 4: phrase\_entry

The phrase\_entry table stores all the indexed values.

Entries in this table having record\_type\_code ‘b’ are bib index values; the index\_tag indicates the type of index (e.g., a=author, i=isbn, c=call number, t=title, etc.)

Write a query to display the index entries for the bib with the system-generated record\_id provided below.

Fields: record\_id, index\_tag, varfield\_type\_code, index\_entry  
Table: sierra\_view.phrase\_entry   
Where: record\_id = 420908838869  
Order by: index\_tag, varfield\_type\_code  
Limit: 20

Table

Description automatically generated

Figure 17 Exercise 4 query output

*Your data output should appear like this screenshot*

### Convert id field to a record number id2reckey()

The **id2reckey()** function is a handy tool for converting those long id numbers into the relevant, more familiar record number. This works for bib, item, patron, order, etc. Here is an example:

SELECT

record\_id, id2reckey(record\_id) as bibid, language\_code

FROM

sierra\_view.bib\_record

WHERE

cataloging\_date\_gmt > '2022-10-13' and

language\_code not in ('eng', 'fre', 'ger', '')

limit 10

Table

Description automatically generated

Figure 18 Example of id2reckey() function

Notes:

* The value the id2reckey() function returns is a character with a letter prefix.
* The “check digit” is not included at the end of the bibid returned by the id2reckey() function.
* The script above shows an example of using the **in()** or **not in()** function. That function will match a comma separated list, or exclude a comma separated list.
* The empty element at the end of the list for language\_codes means exclude records where there is no language\_code

Another example of id2reckey()

SELECT

id2reckey(v.record\_id) as “bib\_id”,

marc\_tag, field\_content

FROM

sierra\_view.varfield\_view v

WHERE

record\_id='420908909341'

order by marc\_tag

Table

Description automatically generated

Figure 19 id2reckey() output example2

### Exercise 5: bib\_record\_location

Write a simple query to retrieve the locations for a specific bib using the system’s bib\_record\_id   
given in the exercise below.

Fields: bib\_record\_id, id2reckey(bib\_record\_id),location\_code   
Table: sierra\_view.bib\_record\_location  
Where: bib\_record\_id=420908719549

Graphical user interface, application

Description automatically generated

Figure 20 Exercise 5 query output

*Your data output should appear like this screenshot*

### Exercise 6: item\_record\_property

There are a lot of fields on the table item\_view, but it doesn’t include the item call number. You’ll find the item call number on item\_record\_property records. In fact, the table has the value entered and a ‘normalized’ version of the call number. The normalized version strips out delimiters and converts everything to lower case.

Use the following specifications to write a query against the item\_record\_property table.

Fields: id2reckey(item\_record\_id) as item\_no, call\_number, call\_number\_norm, barcode  
Table: sierra\_view.item\_record\_property  
Where: call\_number\_norm like ‘nd20%’  
Limit: 10

Graphical user interface, application

Description automatically generated

Figure 21 Exercise 6 query output

*Your data output should appear like this screenshot*

### Exercise 7: patron\_view

Let’s look at some non-MARC data. Write a query to list patrons and their total checkout count.

To make this a little more challenging, the conditions for the where clause given below need to be put in the proper SQL syntax. That is, you need to use the correct field/column name, the appropriate operator symbol, and enclose character field/columns in single quotes.

Fields: record\_num as patronID, barcode, checkout\_total   
Table: patron\_view  
Where: the barcode starts with 2062000 and

the ptype code is 6 and

the checkout total is greater than 49  
Limit: 10

*Hint: Postgres SQL does not use ‘starts with…’ Instead use the ‘%’ wildcard character.   
The condition like ‘2062000%’ means the value starts with 2062000*

Table

Description automatically generated

Figure 22 Exercise 7 query output

*Your data output should appear like this screenshot*

### Exercise 8: user\_defined\_pcode1\_myuser

Write a simple query to display the institutions displayed in the patron record.

The patron institution code is stored on the patron\_view table. The descriptions for Watson’s institution codes are in the lookup table named ‘user\_defined\_pcode1\_myuser’. This query will get the description from the user\_defined\_pcode1\_myuser table. Other examples of lookup tables are in the Appendix.

There is a column on this table named ‘display\_order’. You don’t have to select that field, but sort the results by that field.

Fields: code, name as institution   
Table: sierra\_view.user\_defined\_pcode1\_myuser  
Order by: display\_order  
Limit: 15

Table

Description automatically generated

Figure 23 Exercise 8 query output

*Your data output should appear like this screenshot*

### Exercise 9: patron\_record\_address

It’s a constant battle to enforce data entry rules. Since patrons can self-register, they often input the wrong thing in the registration form. A common error is they input “NY” for the city and leave the state blank.

Write a query to pull the following fields from the patron address table. Use the functions and column aliases shown.

Fields: id2reckey(patron\_record\_id) as patronID, addr1, city, region AS state, postal\_code as zip, country

Table: sierra\_view.patron\_record\_address

Where: city = 'NY'

Limit: 10

Results will contain addresses where the data has a blank state and ‘NY’ as the city.

Table

Description automatically generated

Figure 24 Exercise 9 query output

*Your data output should appear like this screenshot*

Go edit those records (just kidding!).

### Exercise 10: hold

The hold table stores hold requests. It includes the ids of the item and patron. The item barcode is not included in this table, but this table provides the basic information about the old.

Write a query to select the following fields with the alias names shown. You should use a recent date rather than the one shown below.

Fields: id as hold\_id, id2reckey(patron\_record\_id) AS patron, id2reckey(record\_id) AS item,  
 placed\_gmt

Table: sierra\_view.hold  
Where: placed\_gmt > ‘2022-12-22 00:00’  
Order by: hold\_id  
Limit: 10

Table

Description automatically generated

Figure 25 Exercise 10 query output

*Your data output should appear like this screenshot*

In the Part 2 of the course you will learn how enhance this hold script to retrieve the associated barcode for the item from another table.

### Exercise 11: challenge

Up to now you were given the specific fields, table name, and where clause necessary to complete an exercise. Hopefully in doing the exercise you didn’t run into too many errors – and if you did you were able to fix them.

For this last exercise, please see the screenshot of a query’s output below.

Your challenge is to find the table in the database that contains the column names shown in the screenshot.

After you determine the table you need, write an SQL query to replicate the following results.

Select only the left-most 50 characters of the fields named “best\_title” and “best\_author”.

Limit results to 15 rows.

Graphical user interface, text, application, email

Description automatically generated

Figure 26 Exercise 11 challenge output

# What’s Next

If you feel comfortable with the course material up to this point and did not have much trouble with Exercises 1-10, you are ready for Part 2 of the course.

You have completed the basic introductory skills using pgAdmin to explore Sierra data. There is still a great deal more to learn. The Appendix which follows contains information for future reference.

### Use the Appendix as a resource

**Intermediate / Advanced Functions**

A few examples of somewhat complicated functions are described in the Appendix section of this document, under the heading “Intermediate / Advanced Functions.”

**Categories of Tables**

The Appendix provides summaries of general categories of the tables in Sierra with examples and suggestions how to browse these tables. You should find this list useful when you need to choose which table to query.

**Troubleshooting Common SQL Errors**

Refer to this section for help when you are developing your SQL queries. You will almost certainly run into some errors along the way, so hopefully the tips provided here will help you correct them.

**Solutions to Exercise Problems**

The solutions to Exercises 1 – 10 are provided in this section of the Appendix.

### Part Two of Sierra & SQL

Part Two of this course will build upon what you have learned so far. We mentioned that no single table fully describes a concept like a patron or book. Instead, we will need to retrieve information from two or more tables to get a complete description of something. Part Two focuses specifically on using SQL to retrieve data from multiple related tables.

# Appendix

## Intermediate / Advanced functions

Here are a few more functions you can use in your SQL queries for future reference.

### Regular expressions in SQL

If you need to use a regular expression (or a pattern of characters that is recognizably standardized like a social security number or a 7-digit phone number), this is supported in SQL. For example, this looks for a 7-digit phone number:

-- 7-digit phone pattern

**WHERE** patron\_record\_phone.phone\_number ~ '**^[0-9]{3}\-[0-9]{4}$'**

### Convert a field type using cast()

Sierra stores currency as regular numbers. You can use the **cast()** function to display the output of this type as money. Here is an example: **cast**((amt.start\_charge) as money) as paid

**SELECT**

voucher\_num,

paid\_amount,

**cast**((paid\_amount) as money) **as** paid

**FROM**

sierra\_view.order\_record\_paid

**WHERE**

voucher\_num=32323

### Find the aggregate value max()

You can use the **max()** function to find the maximum value of something. For example, this finds the highest patron ID assigned to date:

**SELECT**

max(record\_num)

**FROM**

sierra\_view.patron\_view

### Group by

You can use the ‘**GROUP BY**’ clause to get totals or subtotals over a group of records. For example, this query groups bib records by their material type code and counts how many there are for each group.

**SELECT**

material\_code,count(\*)

**FROM**

sierra\_view.bib\_record\_property

**WHERE**

material\_code in ('3','s','a')

**GROUP BY** material\_code

Graphical user interface, text, application, table

Description automatically generated

Figure 27 Output from a GROUP BY clause

Notes

* Every column listed in the group by clause must be included in the select clause.
* **GROUP BY** clause must come after the **WHERE** clause and before any **ORDER BY** clause.

### Group by rollup

The output of the previous example shows the number of Text and Auction bib records. It begs the question, “What is the total?” The rollup function provides the total of the group.

**SELECT**

material\_code,count(\*)

**FROM**

sierra\_view.bib\_record\_property

**WHERE**

material\_code in ('s','a')

**Group by rollup**(material\_code)

The new output will contain the total of the groups:

Table

Description automatically generated

Figure 28 Group output with rollup total

Suggestion: Rather than overcomplicating the SQL query, just retrieve ungrouped data. Then export the data and bring it into Excel or Google Refine where it is easier to group data using a pivot table.

### filter()

Another function that may be applied to aggregate values is the **filter()** function. As you can see in the example below, the **WHERE** clause is performed on a field by field basis:

**SELECT**

location\_code, **COUNT**(id) **AS** total\_items,

**COUNT**(id) **FILTER**(WHERE item\_status\_code = '-') **AS** total\_available,

**COUNT**(id) **FILTER**(WHERE item\_status\_code = 'm') **AS** total\_missing

**FROM**

sierra\_view.item\_record

**GROUP BY** 1

**ORDER BY** 1;

Table

Description automatically generated

Figure 29 Output from a query with filter()

### 

### The HAVING clause

When you create a group, you may apply a filter on the group by using a ‘**HAVING**’ clause.

For example, if you add the following line at the end of the above script, the output will only is material codes a and s:

**HAVING count(\*) > 100000**

### Aggregate functions

SQL supports several functions when you have a group, including

* **SUM()**
* **AVG()**
* **Min()**
* **MAX()**
* **COUNT()**

### CASE statements / conditional statements

You may want to control the output based on a condition. For example, given the SQL query that counts bib records by material type code, you might output a description instead of a code. Take this script that outputs a single character code:

**SELECT**

material\_code,count(\*)

**FROM**

sierra\_view.bib\_record\_property

**WHERE**

material\_code in ('s','a')

**GROUP BY** material\_code

We can convert the default material\_code using a conditional statement. We give the result of the **CASE** statement the alias mat\_type:

**SELECT**

**CASE**

**WHEN** material\_code = **'a'**

**THEN** 'Text**'**

**WHEN** material\_code ='s'

**THEN** 'Auctions'

**END** mat\_type,

**count**(\*) **as** qty

**FROM**

sierra\_view.bib\_record\_property

**WHERE**

material\_code **in** (‘s’,'a')

**GROUP BY** material\_code

**HAVING** count(\*) > 100000

Graphical user interface, application, table

Description automatically generated

Figure 30 Output after using a CASE condition

In the above queries you could use the position of the field in the **SELECT** statement instead of the field name in the **GROUP BY** clause. That is, you could say ‘**GROUP BY** 1’

### array() and array\_to\_string()

An array is basically a list. This function may be useful when you have a repeatable field in a record. As an example, consider this query that retrieves the repeatable field ‘location’ from a bib record.

Notice the array element is based on a **SELECT** query within the first **SELECT** query. That query retrieves all location codes on a bib and separates each location by a comma. It doesn’t bother to include the location code ‘multi’ since that’s included when there is more than one location

The results of the array of location names obtained by the second ‘**SELECT**’ clause passed to the other function, array\_to\_string. That function simply does just that – it makes a single character string out of the list of locations and gives the string the alias name ‘loc\_codes.

**SELECT**

ID2RECKEY(id),

**array\_to\_string**(

**array**(**SELECT** location\_code

**from**

sierra\_view.bib\_record\_location **as** bibloc

**WHERE**

sierra\_view.bib\_view.id = bibloc.bib\_record\_id

-- we don't need to see the word multi

**and** location\_code != 'multi'

**order** **by** 1),', ')

**as** loc\_codes

**FROM**

sierra\_view.bib\_view

**LIMIT** 25

A picture containing table

Description automatically generated

Figure 31 Output using a string array for bib loc(s)

The previous example uses a second **SELECT** statement, known as a sub-query. It also joined two tables, bib\_view and bib\_record\_location. An explanation of those features will be provided in Part Two of the SQL course.

### More functions

There are many more postGres SQL functions available besides the ones described in this document. Here is a list of examples:

**exists not exists lag() lead() over() rank()**

**partition() split\_part() replace() substring() lower() upper()**

**length() string\_agg() age() now() to\_number() reverse()**

distinct join union left join right join

## Types of data tables

### List of values tables

Some tables simply provide a list of values for the system. Usually, they have an id field and a description field. Here are some examples of tables that have that structure:

|  |  |  |
| --- | --- | --- |
| acq\_type\_property\_name | foreign\_currency | item\_status\_property\_name |
| location\_name | ptype\_property\_name | record\_type\_name |
| scat\_category\_name | user\_defined\_pcode1\_myuser | user\_defined\_pcode2\_myuser |

Table 2 List of some value tables

#### Explore list of values tables with View / Edit

Take a few minutes to examine the values in some of the list of value tables. Follow these steps:

* expand open **sierra\_view > Views**
* right-click the table name you want to explore
* select **View / Edit**
* you can select **All Rows** since these tables don’t have many records
* peruse the data output, **close the query**

### Descriptive data tables

Several tables store what you would consider data. It may be descriptive data (e.g., bib, item, order, vendor, etc.) or transaction data (e.g, check outs, holds, orders).

Here are a few tables that provide descriptive data:

|  |  |  |
| --- | --- | --- |
| bib\_record | bib\_record\_location | bib\_view |
| varfield\_view | bib\_record\_property | item\_record |
| item\_view | order\_record | order\_view |
| patron\_record | patron\_view | patron\_record\_address |
| phrase\_entry | resource\_record | resource\_view |

Table 3 Some descriptive data tables

#### Explore descriptive data tables with View / Edit

Take a few minutes to examine the values in some of the descriptive tables. Use the following steps

* expand open **sierra\_view > Views**
* right-click the table name you want to explore
* select **View / Edit**
* select **Last 100 rows** because these tables have many thousands of records
* peruse the Data output, **close the query**

### Transaction data tables

A transaction table stores a record of activity such as hold placed on an item. Here are the names of a few transaction tables.

|  |  |  |
| --- | --- | --- |
| checkout | hold | invoice\_record |
| invoice\_view | order\_record | record\_metadata |

Table 4 Some transaction data tables

#### Explore transaction data tables with View / Edit

Take a few minutes to examine the values in some of the transaction tables. Use the following steps

* expand open **sierra\_view > Views**
* right-click the table name you want to explore
* select **View / Edit**
* select **Last 100 rows** because these tables have many thousands of records
* peruse the Data output, **close the query**

### Other data tables

There are other tables which are primarily used to show the relationship between multiple tables. Tose so-called **linking tables are discussed** in Part Two of this course.

## Trouble shooting common SQL errors

You will undoubtedly make mistakes as you type your SQL query. To show you some of the most common mistakes let’s start with the correct syntax of this query:

1 SELECT  
2 record\_num,  
3 marc\_tag,  
4 field\_content  
5 FROM  
6 sierra\_view.varfield\_view  
7 limit 50

Below are some of the most common error messages you will see and the query that elicited them. You can compare them to the above if you’re not clear about how to correct your query.

### Column does not exist

A field in the query does not exist. It may be spelled incorrectly, or the name is not valid. View the list of column names to find the problem.

Graphical user interface, application

Description automatically generated

Figure 32 Error: column does not exist

In this example the field name ‘content’ is wrong – the correct field name is “field\_content” so you need to edit line 4.

### Syntax error

There are many sorts of syntax errors you might come across. Here is the error you will see if you have a comma following the last field in the **SELECT** clause (also known as a trailing comma):

Graphical user interface, text, application

Description automatically generated

Figure 33 Error: trailing comma in SELECT

Below, we are missing the commas that should follow record\_num and marc\_tag

Graphical user interface, text, application

Description automatically generated

Figure 34 Error: missing commas between fields

### Missing table prefix

In some SQL environments you cannot just give the table name in the **FROM** clause. In pgAdmin, you must prefix table names with “sierra\_view.” So in the example below line 6 should say: sierra\_view.varfield\_view

Graphical user interface, text, application

Description automatically generated

Figure 35 Error: missing table prefix sierra\_view.

### Missing parameters in a function

Sometimes you will use a function but forget the appropriate parameters that should go in the parenthesis. Here, we intended to ask for the first 40 characters of a title. We used the **left()** function. That function needs two parameters, the field name and length to return. Line 2 should say **left(title,40**):

Graphical user interface, text, application, email

Description automatically generated

Figure 36 Error: function not recognized. Incomplete parameters

### Incorrect or missing wildcard

When you are using wildcard characters as a filter, you may run into problems if you get the syntax wrong. Here, we meant to retrieve titles that begin with ‘A’ but forgot the wildcard for “**like**” and returned zero results. Line 7 should include the character wildcard symbol.

Graphical user interface, text, application

Description automatically generated

Figure 37 Error: incorrect or missing wildcard syntax

It should say:  
title **like ‘**A%**’ and**

## Solutions to Exercises

**Exercise 1 Solution**

-- SQL4LIB

-- 1. bib\_view

SELECT

id, record\_num, cataloging\_date\_gmt as cat\_date,

left(title,50) as title

FROM

sierra\_view.bib\_view

WHERE

cataloging\_date\_gmt > '2022-10-01' and language\_code = 'spa'

LIMIT 10

**Exercise 2 Solution**

-- SQL4LIB

-- 2. varfield\_view

SELECT

record\_id, record\_num, marc\_tag, field\_content

FROM

sierra\_view.varfield\_view

WHERE

record\_num = 2043861

ORDER BY marc\_tag

**Exercise 3 Solution**

-- SQL4LIB

-- 3. subfield\_view

SELECT

record\_num, marc\_tag, tag, content

FROM

sierra\_view.subfield\_view

WHERE

record\_num = 2043861 and marc\_tag between '245' and '299'

ORDER BY marc\_tag, display\_order

**Exercise 4 Solution**

-- SQL4LIB

-- 4. phrase\_entry

SELECT

record\_id, index\_tag, varfield\_type\_code, index\_entry

FROM

sierra\_view.phrase\_entry

WHERE

record\_id = 420908838869

ORDER BY index\_tag, varfield\_type\_code

LIMIT 20

**Exercise 5 Solution**

-- SQL4LIB

-- 5. bib\_record\_location

SELECT

bib\_record\_id, id2reckey(bib\_record\_id, location\_code

FROM

sierra\_view.bib\_record\_location

WHERE

bib\_record\_id = 420908719549

**Exercise 6 Solution**

-- SQL4LIB

-- 6. item\_record\_property

SELECT

id2reckey(item\_record\_id) as item\_no, call\_number,

call\_number\_norm, barcode

FROM

sierra\_view.item\_record\_property

WHERE

call\_number\_norm like 'nd20%'

LIMIT 10

**Exercise 7 Solution**

-- SQL4LIB

-- 7. patron\_view

SELECT

record\_num as patronID, barcode, checkout\_total

FROM

sierra\_view.patron\_view

WHERE

barcode LIKE '2062000%' and

ptype\_code = 6 and

checkout\_total > 49

LIMIT 10

**Exercise 8 Solution**

-- SQL4LIB

-- 8. user\_defined\_pcode1\_myuser

SELECT

code, name as institution

FROM

sierra\_view.user\_defined\_pcode1\_myuser

ORDER BY

display\_order

LIMIT 15

**Exercise 9 Solution**

-- SQL4LIB

-- 9. patron\_record\_address

SELECT

id2reckey(patron\_record\_id) as patronID,

addr1, city,

region as state,

postal\_code as zip, country

FROM

sierra\_view.patron\_record\_address

WHERE

city = 'NY'

limit 10

**Exercise 10 Solution**

-- SQL4LIB

-- 10. hold

SELECT

id as hold\_id,

id2reckey(patron\_record\_id) as patron,

id2reckey(record\_id) as item,

placed\_gmt

FROM

sierra\_view.hold

WHERE

--

-- change the placed\_gmt date to a recent date

--

placed\_gmt > '2022-12-20'

ORDER BY hold\_id

limit 10

**Exercise 11 Solution**

-- SQL4LIB

-- 11. Bib\_record\_property

SELECT

publish\_year,

left(best\_title,50) as best\_title,

left(best\_author,50) as best\_author

FROM

sierra\_view.bib\_record\_property

WHERE

publish\_year = 1700

ORDER BY publish\_year, best\_title

LIMIT 15

1. Cover generated by <https://orly.nanmu.me/> [↑](#footnote-ref-1)
2. For general information about **PostgreSQL** visit <https://www.postgresql.org/> [↑](#footnote-ref-2)
3. There are variants of SQL like MySQL, Oracle, SQL server. Minor differences in syntax exists between these. If you do any research outside the scope of this document, be sure to use the official documentation found at <https://www.postgresql.org/docs/current/>

   [↑](#footnote-ref-3)