**2.1 Extracting Data from a Database**

A Relational Database Management System (RDBMS) handles the way data in a database is stored, maintained, and retrieved. In the case of Cerner Millennium, Oracle SQL is the RDBMS. With that said, you will likely never have to write true SQL (although CCL and SQL are astonishingly similar).

All Cerner Millennium applications use CCL to select from, insert into, update into, and delete data from the Millennium database. Any time you are looking at information in a front-end Millennium solution (e.g. PowerChart, PHA Med Manager, or Order Result Viewer), CCL was used to initially write that information to the database and used to display what you are seeing in the application.

When you write and execute CCL, the query is converted and passed to the RDBMS (Oracle) in a form that is understood. This will be an important concept when we get to trace level logging.

There are two main ways to execute CCL. The first is with a CCL utility run from the back-end and the second is with an application you should already be slightly familiar with after the last chapter called Discern Visual Developer (DVDev). Our primary focus over the remainder of the book will be on DVDev. However, several chapters will cover the CCL utility in the backend.

**2.2 Our First Query**

DVDev is a front-end integrated development environment (IDE) that is used to create and edit CCL. It combines text editor functionality with Cerner's data dictionary and efficiency tools. It allows you to do table and code set lookups, review fields that exist on tables, build prompts, forms and layouts for reports, and more.

In your non-production environment, open up the application *DiscernVisualDeveloper.exe*.

Machine generated alternative text:
File Edit 
Code View 
Code View 
Output 
Discern Visual Developer 
View Build Tools Reports Window Help 
NUM 
Request/RepIy Tables/FieIds 
Macros 
MYO 8134316 Lno, colo 

Once in the application, click on the while paper icon and then select *New (or just type Ctrl + N)*. Ensure Blank is selected as the *File Type* and click *OK (or just press ENTER)*.

Machine generated alternative text:
File Ed- 
c 

Your gray background should turn white with a curser.

<label diagram>

Machine generated alternative text:
File Edit 
Code View 
Code View 
Output 
Discern Visual Developer - 18134316 DVDI) 
View Build Tools Reports Window Help 
NUM 
Request/ Reply Tables/ Fields 
B134316 DVDI 
Macros 
MYO 8134316 Lnl, coli 

There are a number of built-in tools in DVDev. We will be using Query Builder quite frequently (you should already be familiar with it from the previous exercises) in the beginning, but by the end of this book you shouldn't need to use it. It is much quicker to free hand your queries (this comes with experience, of course).

Machine generated alternative text:
Tools Reports Window 
Prompt Builder... 
Query Builder... 
Layout Builder... 
Record Builder... 
PowerNote Query Builder... 
Transfer Objects... 
Code Lookup... 
Add Variable... 
Add Subroutine... 
Add Code Values... 
Import Items... 
Record Quick Macro... 
Play Quick Macro 
Save Quick Macro 
Help 
Ctrl+Shift+H 
Ctrl+Shift+Q 
Ctrl+Shift+L 
Ctrl+Shift+ I 
Ctrl+Shift+2 
Ctrl*F1 
Ctrl*D 
Ctrl+8 
Ctrl* 1 
Ctrl+2 
Ctrl*Shift+R 
Ctrl+Shift+P 

If you have ever learned a programming language, you know that one of the first applications you ever write, traditionally, is a "Hello World" application. There is no "beginner" way to do this in CCL, so we will suffice with a different introductory query.

Type the following into your text window:

*select \* from person with maxrec=1*

Machine generated alternative text:
B134316 DVD4* 
select 
from person with maxrec—l 

Now, type *CTRL + Q (or you can click on BUILD -> Run Ad Hoc Query).* You should see a Query Ouput - Ad Hoc Query window appear with a single record from the PERSON table! Congratulations on your first query!

Machine generated alternative text:
Edit Query 
PERSON ID 
UPDT CNT 
UPDT ID 
590623 . oo 
UPDT 
TASK 
15301 
UPDT APPLCTX 
2075552874.00 
ACTIVE IND 

Let's break down this basic query and analyze it.

This is our basic select statement. There are three bolded keywords (select, from, with) and three additional items (\*, person, maxrec=1).

***SELECT***

*\**

***FROM***

*person*

***WITH*** *maxrec=1*

A select statement (our entire query) is how we retrieve data from the database. Our statement starts with a SELECT clause to denote this. There is an asterisk (\*) in the SELECT clause. Everything we list here (we will see a list in a second) is data that will be returned from the query. If you use a wildcard (\*), then every field on the table is returned.

Following the SELECT clause is the FROM clause. The FROM clause contains all of the tables we are selecting from. We are selecting data from the PERSON table..

Lastly, we have a WITH clause. The WITH keyword denotes control options - optional parameters we can apply to our query. One of these parameters is "maxrec" which controls the number of records our query returns. It is set to 1 because otherwise you would see every patient in your environment.

**You always want to add control parameters, especially in a production environment, when you run a query for the first couple of times until you know it is working.** There is a 100% chance that you will eventually write a query that takes 10 minutes to run. It is inevitable. I still do it.

Depending on how your environment is set up, this can just be annoying and a waste of time, or it could prevent processing on server 56/58, which can affect other users.

If you happen to write a query that takes a very long time, the *Query Output* window won't show any data and you will see *Executing Ad Hoc Query* in the bottom left corner of the screen.

Machine generated alternative text:
Task Edit View Help 
Edit Query 
Executin Ad Hoc Que 

If this happens, click the X in the top right corner of the screen.

Machine generated alternative text:
Cannot close window while repot execution is in progress. 

Click OK. Repeat 9 more times (10 total) and you will see the following window. Click Yes.

Machine generated alternative text:
Discern Output Viewer 
Execution is taking a long time. Would you like to close this window? 
(Process will continue running until completion.) 

Close the Query Output window so we can modify our query a little bit. Often times you will not want to return every single field for a table. So, let's take a couple of the fields from the PERSON table and explicitly write them. Notice how after each field there is a comma. Also note, if there is more than one table name listed, you need a comma after each as well.

*select*

*person\_id*

*, name\_full\_formatted*

*, birth\_dt\_tm*

*from*

*person*

*with maxrec=1*

Machine generated alternative text:
Edit Query 
PERSON ID 
DT TM 
27310766. 
NAME FULL FORMATTED 
BIRTH 
Grue t ze 
S t ephanie 

Since we explicitly listed three fields our query only returned three fields. Now, you wrote those fields because I told you to. Until you become familiar with the PERSON table, you aren't going to be able to remember all of the fields on this table. Therefore, either you can use the Query Builder, or we can learn about aliases.

**2.3 Adding Aliases**

Close the Query Output window so we can modify our query a little bit. Add the letter "p" after person. The letter(s) following a table name is called an **alias**. Any alphanumeric sequence can be used as an alias for a table. Aliases are useful for a number of reasons and required when two or more tables are listed (we will see this in Chapter 3).

*select*

*person\_id*

*, name\_full\_formatted*

*, birth\_dt\_tm*

*from*

*person p*

*with maxrec=1*

Add a new line after BIRTH\_DT\_TM to add a new field (don't forget your comma!) and type "p." As soon as you type the period you should see a window pop-up telling you the table fields have not been loaded. Click *Yes*.

Machine generated alternative text:
Tables fields not loaded 
T able fields have not been loaded 
Would you like to do this now? 
Don't ask me again 

Two things should have happened. The first is that you should see a scroll bar with a list of fields. You can navigate through this list by using either your mouse or the Up/Down arrow keys on your keyboard. Double clicking on a field name or pressing *Enter* while it is selected will add it to your query.

Machine generated alternative text:
from 
per 
abs blith dt tm 
active ind 
active status cd 
active status dt tm 
active_status_prsnl id 
age_at death 
prec_mod flag 
unit cd 
archive env id 

If you click away from your query you might lose your drop down. No need to fret - simply press *CTRL + Spacebar* or simply delete the period and retype it and your dropdown will reappear.

Machine generated alternative text:
select 
person id 
, name full formatted 
, birth dt tm 
from 
person p 
i th maxrec—l 

The second thing that should have happened is the *Tables/Fields* tab to the left of your text window should be populated with data.

Machine generated alternative text:
Tables/ Fields 
v person 
Fields 
9 
abs birth dt tm 
active ind 
active status cd 
active status 
dt tm 
active status 
prsnl id 
age_at_death 
prec_mod flag 
age_at_death 
unit cd 
archive env id 
archive status cd 
archive status dt tm 
autopsy_cd 
beg_effective dt tm 
birth dt cd 
Type 
d q8 
Macros 
Code View Request/RepIy Tables/FieIds 

If you double click a field name its properties appear.

Machine generated alternative text:
T able Name: 
PERSON 
Description: 
-eld Properti 
Field Name: 
ACTIVE STATUS DT TM 
active status date and time 
D efinition: 
the date and time that the was set 
Close 

The alias you use and the table name should be listed above all of the fields.

Machine generated alternative text:
Tables/FieIds 
v person 

As you start to become more proficient at writing CCL and start using more tables, you can easily switch back and forth with the dropdown.

Machine generated alternative text:
person 
accession order r 
ce result set link 
clinical event 
code value outbound 
container accession 
container event 
cqm fsieso_que 
cqm_oeninterface_que 
cqm_oeninterface tr I 
credential 
eem moh detail 
eem trans reltn 

Going forward, all of the examples in this book will use aliases and the Tables/Fields window to write queries because it is much faster. The sooner you get comfortable writing queries this way, the sooner you will become proficient at writing CCL. If you have any difficulty, you can always write queries in the Query Builder.

**Exercise 2.1**

Take our existing query (shown below) and modify it to use the alias p for the PERSON table.

*select*

*person\_id*

*, name\_full\_formatted*

*, birth\_dt\_tm*

*from*

*person p*

*with maxrec=1*

The easiest way to write queries is to start with the SELECT keyword and skip to the FROM section to list your tables.

*select*

*from*

*person p*

Then add your control option.

*select*

*from*

*person p*

*with maxrec=1*

Now, jump back up to the select and type out the fields. Writing the query in this order allows you to use the drop down generated when you type *"alias."* to easily select your field names. If you write your field names first, you will not get a drop down.

Add the PERSON\_ID and the NAME\_FULL\_FORMATTED by selecting them from the drop down list after typing the alias.

Lastly, find the BIRTH\_DT\_TM in the Tables/Fields tab and click and drag it into your query. This is another way to add fields to your query instead of typing them.

Once you have your query complete, run it to ensure it works (CTRL + Q).

**Exercise 2.2**

It's impossible not to make mistakes writing queries. Therefore, it is important to know what common errors mean when you see them.

Type the following query in DVDev exactly as you read it and then run it (CTRL + Q).

*select*

*o.order\_id*

*, o.order\_mnemonic*

*o.orig\_order\_dt\_tm*

*from*

*orders o*

*with maxrec = 1*

If you forget a comma separating fields in the SELECT clause, you will receive this error message.

Machine generated alternative text:
Failed to execute report! Discern Explorer program: Ad Hoc Query 
%CCL-E-g-VCCL RUN 
symbol found. 

**2.4 Qualifications**

So far, we have written a very basic select statement returning a single row on the *PERSON* table. While this example is great to illustrate a basic query, it's not very useful. When you write a query, you write it with a purpose. You might be looking for a specific person or all people with a certain name. We can add these constraints to a query to return the data we want or to limit the data that is returned. These constraints are called **qualifications**.

Let's say we want to gather a list of all people whose last name is "Smith". We need to add a qualification to our query to limit the people returned to those whose last name *equals* "Smith". Qualifications are written in a WHERE clause and follow tables listed in the FROM clause. In our example, we only want people whose *last name equals* "Smith".

Therefore, we can write the following qualification where "=" is an equality operator. There are several operators you can use in qualifications and we will see several more later in the chapter.

*where p.name\_last = "Smith"*

This query works and returns the data we want, but it doesn't work very well.

select

p.person\_id

, p.name\_full\_formatted

, p.birth\_dt\_tm

from

person p

where p.name\_last = "Smith"

Machine generated alternative text:
Task Edit 
V iew 
uery Outpu 
Help 
d Hoc Que 
TM 
NAME LAST 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
eg.g27 
NAME FULL FORMATTED 
BIRTH 
Edit Query 
PERSON ID 
60416B 
20777311 
20777313. 
20735044. 
23340705. 
24048795. 
24592774. 
24592779. 
26127417 
26131009 .00 
26128503 . oo 
26128513 . oo 
Read 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th 
Smi th, 
Smi th , 
Smi th 
Smi th 
Smi th , 
Allan MD 
Zakila 
Z akila 
Eric C 
Amanda B 
S t ephanie 
S t ephanie 
Ki rby 
— Smi th, 
— Smi th , 
— Smi th , 
MSOE0497 
MSOE2097 
MSOE0006 
Anne Marie 
Ad r ana 
Steven W 
13 
Records: 
Execute time (secs): 

**Note:** if no data returns from your query it means there is no test patients with the last name "Smith". Find a test patient and use their last name instead or try the NAME\_LAST\_KEY = "TEST" so you can follow along.

Does anything look amiss with this screen shot?

Notice in the bottom right corner that it took roughly 90 seconds to run this query. That's a really long time. If you design your queries correctly, there should be very few scenarios where it takes more than three seconds to run a query.

So, what's wrong with the query?

The answer has to do with indexes. Remember that indexes are "vehicles to improve lookup efficiency." The data within an index is stored in a different way than a row within a database and this allows the lookup of the information to increase greatly.

The problem with our query is that our qualification looks at a non-indexed field NAME\_LAST. When you execute your query, it looks through every single row on the PERSON table for people whose last name is "Smith". In my non-production environment, there happens to be over 24 million people. That's why it took so long.

**Always use indexes to write qualifications if possible.**

In our example, there is a very simple fix to improve the efficiency of our query. Use the field NAME\_LAST\_KEY, which is an index.

Machine generated alternative text:
Tables/FieIds 
v 
last key 
person 
Type 
vc1DO 
Fields 
name 
9 
name 
last 

To use the index, change the field name in the where clause. Do you remember what we have to do with the actual value? Key fields hold all caps, alphanumeric (no special characters like dashes) strings. So, we can't qualify on "Smith" we have to qualify on "SMITH".

*select*

*p.person\_id*

*, p.name\_full\_formatted*

*, p.birth\_dt\_tm*

*from*

*person p*

*where p.name\_last\_key = "SMITH"*

By simply changing the qualification to use an indexed field, my query ran in .2 seconds. That's 45,000% faster.

**2.5 Relational Operators & Additional Qualifications**

In our previous example, we already discussed the equality operator ( = ). There are a number of other operators that allow us to return the data we want. This is not a comprehensive list of operators; however, they are the most common.

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| = | Equal to | PERSON.name\_last\_key = "SMITH"  PERSON.person\_id = 26125362.00 |
| != | Not equal to | PERSON.name\_last\_key != "SMITH"  PERSON.person\_id != 26125362.00  PERSON.birth\_dt\_tm != null |
| > | Greater than | PERSON.birth\_dt\_tm > cnvtdatetime("01-JAN-2000")  CLINICAL\_EVENT.result\_val > "15"  PERFORM\_RESULT.normal\_high > 200 |
| >= | Greater than or equal to | PERSON.birth\_dt\_tm >= cnvtdatetime("01-JAN-2000")  CLINICAL\_EVENT.result\_val >= "15"  PERFORM\_RESULT.normal\_high >= 200 |
| < | Less than | PERSON.birth\_dt\_tm < cnvtdatetime("01-JAN-2000")  CLINICAL\_EVENT.result\_val < "15"  PERFORM\_RESULT.normal\_high < 200 |
| <= | Less than or equal to | PERSON.birth\_dt\_tm <= cnvtdatetime("01-JAN-2000")  CLINICAL\_EVENT.result\_val <= "15"  PERFORM\_RESULT.normal\_high <= 200 |
| Between | Inclusive range | Returns data where the value is between two expressions.    PERSON.birth\_dt\_tm between cnvtdatetime("01-JAN-2000") and cnvtdatetime("31-JAN-2000")  PERFORM\_RESULT.normal\_high between 200 and 400 |
| In | Multiple OR qualification | Returns data for any value within the given list.    PERSON.person\_id in (26131489, 26126938, 26122207, 26120748, 26118866) |
| Not | Negate an operator | Commonly combined with In or Exists to create Not In or Not Exists.    PERSON.person\_id not in (26131489, 26126938, 26122207, 26120748, 26118866) |
| Exists | here | This will be covered in Nested Select section. |

More often than not, you are going to have more than one qualification needed to return the data you want. The relationship between the qualifications is important - you might want to return people whose last name is "Smith" AND who has a birth date after (greater than) the year 2000.

Type the following query out in your environment using the AND keyword and see if any data returns. If not, change the qualification until it does.

*select*

*p.person\_id*

*, p.name\_full\_formatted*

*, p.birth\_dt\_tm*

*from*

*person p*

*where p.name\_last\_key = "SMITH"*

***and*** *p.birth\_dt\_tm > cnvtdatetime("01-JAN-2000")*

What if you want to return people whose birth date is greater than 2000 OR less than 1935?

*select*

*p.person\_id*

*, p.name\_full\_formatted*

*, p.birth\_dt\_tm*

*from*

*person p*

*where p.name\_last\_key = "SMITH"*

***and*** *p.birth\_dt\_tm > cnvtdatetime("01-JAN-2000")*

***or*** *p.birth\_dt\_tm < cnvtdatetime("01-JAN-1935")*

Notice how in the screen shot there are several people born in 2016/2017 as well as people from 1930!

Machine generated alternative text:
Edit Query 
PERSON ID 
22222461 
2235003.00 
4939432.00 
24803681 
22886335 . oo 
1413994.00 
26121532 . oo 
7496611 
4112432.00 
24045224.00 
25153233.00 
12677421 
NAME FULL FORMATTED 
BIRTH DT TM 
07/28/16 
09/14/32 
06/15/26 
07/18/17 
08/06/16 
11/02/27 
06/22/19 
07/01/30 
09/06/30 
03/09/17 
05/09/17 
08/31/02 
02/02/08 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
ÄDÄLINE 
ANNE M 
BRADY O 
ccR_Ä 
GABRIEL 
HENRY L 
JANE 
JUANITA 
MARY K 
NEIL W 
REMINGTON 
SAVANNAH R 
ZARA D 

The keywords AND and OR allow you to conditionally control what data returns from the query. You will use the AND keyword much more often than the OR keyword, but it's important to know it is available.

We also used the function cnvtdatetime() in the query. We will be talking much more about what functions are, how they are used, and specifically what cnvtdatetime() does in Chapter 5.

Let's look at another example so we can take a look at the IN operator.

The following query returns a list of lab orderables from the ORDER\_CATALOG table. It assumes the name of the orderable, so if nothing returns when running it, try changing the names.

Type in the following query and run it.

*select*

*oc.primary\_mnemonic*

*from*

*order\_catalog oc*

*where oc.primary\_mnemonic in ("BMP\*", "CBC\*", "BUN\*")*

Machine generated alternative text:
Edit Query 
PRIMARY MNEMONIC 
CBC wi Chou t Differential 
CBC with Differential 

Let's try adding Billirubin.

*select*

*oc.primary\_mnemonic*

*from*

*order\_catalog oc*

*where oc.primary\_mnemonic in ("BMP\*", "CBC\*", "BUN\*", "Bilirubin\*")*

Nothing returns.

Machine generated alternative text:
Edit Query 
NO RESULTS 

There are Bilirubin orderables built in my environment, though.

Machine generated alternative text:
Edit Query 
PRIMARY MNEMONIC 
Fluid 
(SENDOUT) 
Bf I rubin 
Bf I rubin 
Bf I rubin 
Bf I rubin 
Bf I rubin 
Bi I rubin, 
Total 
Total and Direct 
Urine 
Body Fluid 
Cord Blood 
Total 

The reason nothing returns is that when using the IN operator, you can't list more than 1000 strings and numbers or three wildcard items. "Bilirubin\*" was our fourth wildcard in the IN operator, so nothing returned. Notice how we didn't get an error either!

**Exercise 2.2**

Write two different queries in your non-production environment on the ORDERS table, one using the BETWEEN operator and one using the IN operator.

1. Write a query to return all orders for the previous day. Use the BETWEEN operator to qualify against the ORIG\_ORDER\_DT\_TM field on the ORDERS table. Return the ORDER\_ID, the name of the orderable, and the order date.

Hint: use the cnvtdatetime() function with two reserved variables cnvtdatetime(curdate, curtime3). To look back a day, subtract the number of days from curdate: *curdate - 1*. You can also specify the time cnvtdatetime(curdate, 0800) (8am in the morning).

select

o.order\_id

, o.order\_mnemonic

, o.orig\_order\_dt\_tm

from

orders o

where o.orig\_order\_dt\_tm between cnvtdatetime(curdate - 1,0)

and o.orig\_order\_dt\_tm cnvtdatetime(curdate - 1,2359)

1. Write a query using the IN operator to return the first five orders returned from query # 1. Return every field on the table.

select

\*

from

orders o

where o.order\_id in (

14091708825

, 13938612569

, 10706964237

, 13941701081

, 13938039941

)

**Exercise 2.3**

Query the ENCOUNTER table for the number of patients who arrived at any facility in your system today who have not been discharged. For the sake of this example, ignore the fact that outpatients may not ever be discharged in the system and a number of other complexities. The point of this example is simply to practice chaining qualifications together.

1. You will need to use the cnvtdatetime() function just like in exercise 2.2.
2. How do you determine whether a patient hasn't been discharged yet? If you are stumped, look back at Chapter 1 to review NULL.
3. Add a third qualification on the ACTIVE\_IND field to ensure the encounter is active.
4. Use count(\*) in the SELECT clause to return the total number of patients.

select

count(\*)

from

encounter e

where e.arrive\_dt\_tm > cnvtdatetime(curdate,0)

and e.disch\_dt\_tm = null

and e.active\_ind = 1

**2.6 User-Defined Variables**

When you run a query, the name of the columns in the output window are the same as the names of the fields in your SELECT clause. However, you can control the names displayed in the output by using **user-defined variables**. User-defined variables not only change the name of a column header, but they can be used later in the program to reference a field name. Sometimes this is nice to do if a field name has a really long name.

Run the following query and compare the names of the fields in the SELECT clause to those in the output window.

*select*

*p.person\_id*

*, p.name\_full\_formatted*

*, p.birth\_dt\_tm*

*from*

*person p*

*where p.name\_last\_key = "SMITH"*

Modify the query with user-defined variables to change the output.

*select*

***unique\_id =*** *p.person\_id*

*,* ***full\_name =*** *p.name\_full\_formatted*

*,* ***birth\_date =*** *p.birth\_dt\_tm*

*from*

*person p*

*where p.name\_last\_key = "SMITH"*

After you run the query above, the column names should have changed.

Machine generated alternative text:
Edit Query 
UNIQUE_ID 
24803681 
25153233.00 
24045224.00 
22886335 . oo 
FULL NAME 
BIRTH DATE 
07/18/17 
05/09/17 
03/09/17 
SMITH, 
SMITH, 
SMITH, 
SMITH, 
CORA 
REMINGTON 
NEIL W 
GABRIEL R 

An important note is that these user-defined variables cannot be used in a WHERE clause, they can only be used in an ORDER clause and REPORT WRITER clauses, which we will cover in much greater detail in the coming chapters.

**Chapter 2 Recap**

In this chapter, we talked about the basic building blocks of a query and how to put one together. We discussed the *SELECT, FROM, and WHERE* clauses and how we can use each of them to return the information we want.

Multiple qualifications in the WHERE clause can be chained together using the AND and OR keywords to control data. In addition to AND and OR, the relational operators: =, !=, >, >=, <, <=, NOT, BETWEEN, and IN can be used to further control the data.

When using a qualification, always use an index to qualify against if one is available.