

# PIM Training Program

SQL

**Writing your first SQL Query**

# Learning Objective

- At the end of the module learners should be able to run their first query against a single table by filtering with WHERE clause conditions



*Learning Objectives*



# Agenda

## ➤ SQL

- WHERE Clause – SQL's Filter
- Using Comparison Operators
- Using Other Operators
- Handling Cells with No Data – aka NULLs

## ➤ ETL

- Using Explain Plan
- Troubleshooting Errors
- Republishing, Rerunning and Altering Job Runs
- Job Performance History
- Other Ways to Get Help



# WHERE Clause – SQL's Filter

# WHERE Clause – SQL's Filter

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
;
```

Revisiting the query we used in the previous module, it provides us with two columns from the PRODUCT\_GROUPS table – and returns *ALL* the rows in that table – 119 rows.

PRODUCT_GROUP	DESCRIPTION
412	Cloud_Software_Applications
437	Amazon_Points
414	A_Drive
416	Deal_Sourcer
417	Amazon_Sourced
420	Financial_Products
424	Digital_Text_2
425	Digital_Accessories_2
251	Gourmet
241	Watches
236	Misc SDP
234	Travel Store
259	Sports Memorabilia
258	Posters
309	Shoes
23	Electronics
21	Toys

# WHERE Clause – SQL's Filter

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
;
```

Most of the time, you only want a subset of the rows in a table. Some tables have over 1 Billion rows – so effective filtering is key.

PRODUCT_GROUP	DESCRIPTION
412	Cloud_Software_Applications
437	Amazon_Points
414	A_Drive
416	Deal_Sourcer
417	Amazon_Sourced
420	Financial_Products
424	Digital_Text_2
425	Digital_Accessories_2
251	Gourmet
241	Watches
236	Misc SDP
234	Travel Store
259	Sports Memorabilia
258	Posters
309	Shoes
23	Electronics
21	Toys

# WHERE Clause – SQL's Filter

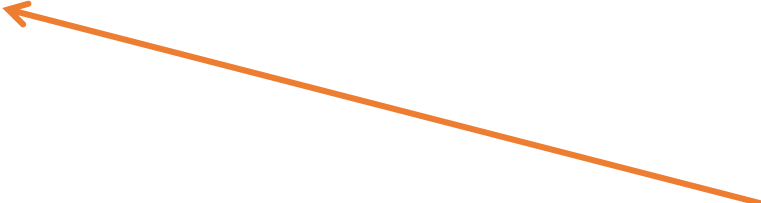
```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
;
```

The main filtering method in SQL is a WHERE clause, which comes after the FROM clause (and before the ORDER BY clause, if there is one).



# WHERE Clause – SQL's Filter

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP = 14  
;
```




To use a filter in Where clause, we enter one or more conditions that must be true for the records to be returned.





# WHERE Clause – SQL's Filter

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP = 14  
;
```



Here we're saying that the value in the PRODUCT\_GROUP column of the table PRODUCT\_GROUPS must be equal to 14 for a row to be returned.



# WHERE Clause – SQL's Filter

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESRIPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP = 14  
;
```

When Oracle runs the SQL query, it checks every row in the table to see if it's TRUE that the value in the PRODUCT\_GROUP column is equal to 14.



# WHERE Clause – SQL's Filter

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP = 14  
;
```

PRODUCT_GROUP	DESCRIPTION
14	Books

The result is a single row (Single record). There was only one record in the table where the PRODUCT\_GROUP column was equal to 14.



# Using Comparison Operators

# Equals

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP = 14  
;
```

The equals sign in this condition is known as the operator.

PRODUCT_GROUP	DESCRIPTION
14	Books



# Not equal

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP != 14  
;
```

The opposite of equals  
(not equals) has it's own  
operator - !=



# Not equal

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP <> 14  
;
```

In fact, it has two. You can also use <> as the 'not equal to' operator.



# Not equal

```
SELECT
pg.PRODUCT_GROUP
, pg.DESCRPTION
FROM PRODUCT_GROUPS pg
WHERE
pg.PRODUCT_GROUP <> 14
;
```

Both produce the same results – all records in the table where it is true that PRODUCT\_GROUP is NOT equal to 14.

PRODUCT_GROUP	DESCRIPTION
424	Digital_Text_2
425	Digital_Accessories_2
437	Amazon_Points
421	Camera
422	Mobile_Electronics
420	Financial_Products
251	Gourmet
241	Watches
236	Misc SDP
234	Travel Store
259	Sports Memorabilia
258	Posters
309	Shoes
23	Electronics
21	Toys
334	Digital_Book_Service
354	Wireless_Prepaid_Phone
264	Art
262	Medical Laboratory Supplies
261	Art and Craft Supplies
260	School Supplies





Using other Operators

# IN Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP IN (14,15,64)  
;
```

Another operator is IN – which checks if a column is equal to any of the values in a list.



# IN Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP IN (14,15,64)  
;
```

The list is enclosed in parentheses, and the values are separated by commas.



# IN Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP IN (14,15,64)  
;
```

PRODUCT_GROUP	DESCRIPTION
15	Music
14	Books

This time, the results include two rows – those matching any of the values in the list.

Since there is no PRODUCT\_GROUP equal to 64 in the table, no record was returned for that value.



# NOT IN Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP NOT IN (14,15,64)  
;
```

Opposite of IN operator is NOT IN.



## < Less Than operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP < 14  
;
```

You can also use greater than and less than symbols as operators...

PRODUCT_GROUP	DESCRIPTION
-1	Mixed
0	Unassigned



## < = and >= operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP <= 14  
;
```

PRODUCT_GROUP	DESCRIPTION
-1	Mixed
0	Unassigned
14	Books

As well as greater than or equal to, and less than or equal to



# Between operator

```
SELECT
pg.PRODUCT_GROUP
, pg.DESCRPTION
FROM PRODUCT_GROUPS pg
WHERE
pg.PRODUCT_GROUP BETWEEN 20 AND 27
;
```

BETWEEN is a handy operator for ranges of numbers, values or (especially) dates.

PRODUCT_GROUP	DESCRIPTION
23	Electronics
21	Toys
22	Games
20	Gifts
27	Video
26	Unknown

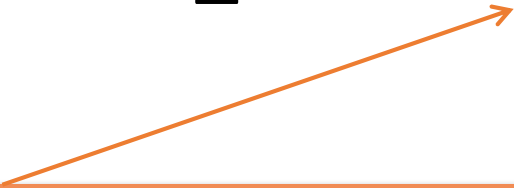
BETWEEN is inclusive of both values listed, so we get results for PRODUCT\_GROUP 20, PRODUCT\_GROUP 27, and everything in between.





# Null Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESRIPTION  
, pg.CREATION_DATE  
FROM PRODUCT_GROUPS pg  
WHERE pg.CREATION_DATE IS NULL  
;
```



There's also an operator for finding blanks in a column: IS NULL

PRODUCT_GROUP	DESCRIPTION	CREATION_DATE
23	Electronics	
21	Toys	
18	Unknown	
22	Games	
20	Gifts	
15	Music	
14	Books	
-1	Mixed	
27	Video	
60	Home Improvement	
0	Unassigned	
75	Baby	



# Not Null Operator

```
SELECT
pg.PRODUCT_GROUP
, pg.DESCRPTION
, pg.CREATION_DATE
FROM PRODUCT_GROUPS pg
WHERE pg.CREATION_DATE IS NOT NULL
;
```

And it's opposite – IS NOT NULL – which returns all rows where a column is not blank.

PRODUCT_GROUP	DESCRIPTION	CREATION_DATE
416	Deal_Sourcer	31-Mar-11
417	Amazon_Sourced	31-Mar-11
424	Digital_Text_2	15-Jul-11
425	Digital_Accessories_2	15-Jul-11
414	A_Drive	20-Oct-10
251	Gourmet	4-Aug-03
241	Watches	4-Jun-03
236	Misc SDP	20-Dec-02
234	Travel Store	9-Sep-02
259	Sports Memorabilia	31-Oct-03
258	Posters	31-Oct-03
309	Shoes	18-Aug-05



# LIKE Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
FROM PRODUCT_GROUPS pg  
WHERE pg.SHORT_DESC LIKE '%To%'  
;
```

Another handy operator is LIKE. It allows you to search for a value in a column that contains a certain text string.

It's combined with a wildcard that takes the place of the rest of the string.



# LIKE Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
FROM PRODUCT_GROUPS pg  
WHERE pg.SHORT_DESC LIKE '%To%'  
;
```

For example, here we're using the LIKE operator to look for any value in the SHORT\_DESC column that has the letters 'To' in it. We use the % wildcard to take the place of any characters on either side of the To.



# LIKE Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
FROM PRODUCT_GROUPS pg  
WHERE pg.SHORT_DESC LIKE '%To%'  
;
```

For example, here we're using the LIKE operator to look for any value in the SHORT\_DESC column that has the letters 'To' in it. We use the % wildcard to take the place of 0-many characters on either side of the To.

It all gets put in single quotes, because it's a text string.


This would match to 'Tools', 'Toys', 'Big Tops', or another other text string with the letters 'To'.



# LIKE Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
FROM PRODUCT_GROUPS pg  
WHERE pg.SHORT_DESC LIKE '%To%'  
;
```

In our case, it matched to  
'Toys' and 'Tools'



PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
21	Toys	Toys
60	Home Improvement	Tools



# LIKE Operator

```
SELECT
pg.PRODUCT_GROUP
, pg.DESCRPTION
, pg.SHORT_DESC
FROM PRODUCT_GROUPS pg
WHERE pg.SHORT_DESC LIKE '%To%'
;
```

Notice that it didn't, however, match to 'Auto' or 'Stones', even though those two values exist in rows of this table.

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
21	Toys	Toys
60	Home Improvement	Tools

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
263	Automotive	Auto
246	Loose Stones	Stones



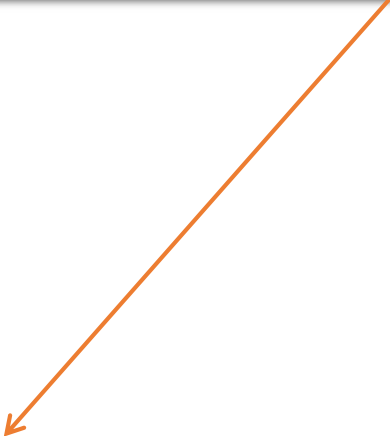
# LIKE Operator

```
SELECT
pg.PRODUCT_GROUP
, pg.DESCRPTION
, pg.SHORT_DESC
FROM PRODUCT_GROUPS pg
WHERE pg.SHORT_DESC LIKE '%To%'
;
```

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
21	Toys	Toys
60	Home Improvement	Tools

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
263	Automotive	Auto
246	Loose Stones	Stones

It's because anything in single-quotes is CASE sensitive in SQL, and the T in 'Auto' and 'Stones' is not capitalized.





# LIKE Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
FROM PRODUCT_GROUPS pg  
WHERE UPPER(pg.SHORT_DESC) LIKE '%TO%'  
;
```

One way around that case-sensitive issue is to wrap our column name with the UPPER() function.

As redshift evaluates each row in the table, it converts the value in the SHORT\_DESC column into all capital letters. As long as we use all capitals in our string, it will match 'TO', 'to', 'To', or 'tO'.



# LIKE Operator

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
FROM PRODUCT_GROUPS pg  
WHERE UPPER(pg.SHORT_DESC) LIKE '%TO%'  
;
```

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
21	Toys	Toys
60	Home Improvement	Tools
263	Automotive	Auto
246	Loose Stones	Stones



# LIKE Operator – Not Like

```
SELECT
pg.PRODUCT_GROUP
, pg.DESRIPTION
, pg.SHORT_DESC
FROM PRODUCT_GROUPS pg
WHERE UPPER(pg.SHORT_DESC) NOT LIKE
'%TO%'
;
```

Probably not a surprise at this point, but LIKE has an opposite, too - NOT LIKE – that returns any row where the value in the specified field doesn't include the string.

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
251	Gourmet	Gourmt
241	Watches	Watches
309	Shoes	Shoes
23	Electronics	CE
267	Musical Instruments	Mus Instr
65	Software	SW
15	Music	Music
14	Books	Books
27	Video	Video
74	Video DVD	DVD
63	Video Games	VG
75	Baby	Baby
293	Tires	Tires
147	PC	PC
351	Digital_Ebook_Purcha se	Digital Ebooks
111	Ebook	Ebook
86	Lawn and Garden	H&G



# LIKE Operator – Not Like

```
SELECT
pg.PRODUCT_GROUP
, pg.DESRIPTION
, pg.SHORT_DESC
FROM PRODUCT_GROUPS pg
WHERE UPPER(pg.SHORT_DESC) NOT LIKE
'%TO%'
;
```

Note: Usage of NOT LIKE, NOT IN, and != do not return nulls. For example, By using NOT LIKE in the query, we didn't get any null value(row) in the SHORT\_DESC table, in spite of having many null values available in the column.

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
251	Gourmet	Gourmt
241	Watches	Watches
309	Shoes	Shoes
23	Electronics	CE
267	Musical Instruments	Mus Instr
65	Software	SW
15	Music	Music
14	Books	Books
27	Video	Video
74	Video DVD	DVD
63	Video Games	VG
75	Baby	Baby
293	Tires	Tires
147	PC	PC
351	Digital_Ebook_Purcha se	Digital Ebooks
111	Ebook	Ebook
86	Lawn and Garden	H&G



Handling Cells with No Data – aka NULLs

# Handling Cells with No Data – aka NULLs

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
FROM PRODUCT_GROUPS pg  
WHERE UPPER(pg.SHORT_DESC) IS NULL  
;
```

A null value in a column (aka a blank) can't be equal or not equal to anything, so must be handled via IS NULL or IS NOT NULL...

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
416	Deal_Sourcer	
417	Amazon_Sourced	
421	Camera	
422	Mobile_Electronics	
437	Amazon_Points	
236	Misc SDP	
234	Travel Store	
259	Sports Memorabilia	
258	Posters	



# Handling Cells with No Data – aka NULLs

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
FROM PRODUCT_GROUPS pg  
WHERE NVL(pg.SHORT_DESC, 'Unknown') != 'CE'  
;
```

We can also use NVL()  
function instead of  
NULL

When redshift evaluates each row of the table, it translates any nulls found in the SHORT\_DESC column into the text string 'Unknown'. Then it evaluates that result for whether it's not equal to 'CE'. Since 'Unknown' isn't equal to 'CE', it will return all the null rows, as well as non-null rows that aren't equal to 'CE'.



# Handling Cells with No Data – aka NULLs

```
SELECT
pg.PRODUCT_GROUP
, pg.DESCRPTION
, pg.SHORT_DESC
FROM PRODUCT_GROUPS pg
WHERE NVL(pg.SHORT_DESC, 'Unknown') != 'CE'
;
```

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC
416	Deal_Sourcer	
417	Amazon_Sourced	
251	Gourmet	Gourmt
241	Watches	Watches
236	Misc SDP	
234	Travel Store	
259	Sports Memorabilia	
258	Posters	
309	Shoes	Shoes
21	Toys	Toys





# Handling Cells with No Data – aka NULLs

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP BETWEEN 20 AND 27  
AND pg.CREATION_DATE IS NOT NULL  
;
```


You can filter multiple conditions, too

If BOTH conditions must be true, separate them with AND.



# Handling Cells with No Data – aka NULLs

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP BETWEEN 20 AND 27  
OR pg.DESCRPTION <> 'Electronics'  
;
```



If EITHER condition being true is sufficient, separate them with OR.



# Handling Cells with No Data – aka NULLs

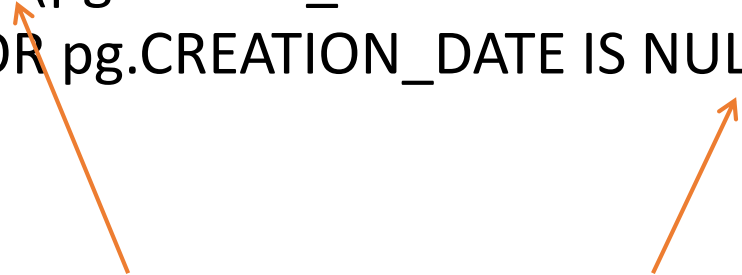
```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP BETWEEN 20 AND 27  
OR pg.DESCRPTION <> 'Electronics'  
;
```

This would return all records where PRODUCT\_GROUP is between 20 and 27, PLUS any record where the DESCRIPTION is not equal to 'Electronics'



# Handling Cells with No Data – aka NULLs

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
, pg.CREATION_DATE  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP BETWEEN 20 AND 27  
AND (pg.SHORT_DESC IS NULL  
    OR pg.CREATION_DATE IS NULL)  
;
```



You can also use parentheses to control how the conditions are applied



# Handling Cells with No Data – aka NULLs

```
SELECT
pg.PRODUCT_GROUP
, pg.DESCRPTION
, pg.SHORT_DESC
, pg.CREATION_DATE
FROM PRODUCT_GROUPS pg
WHERE
pg.PRODUCT_GROUP BETWEEN 20 AND 27
AND (pg.SHORT_DESC IS NULL
      OR pg.CREATION_DATE IS NULL)
;
```

In this case, the PRODUCT\_GROUP must be between 20 and 27 AND the EITHER the SHORT\_DESC must be null OR the CREATION\_DATE must be null.

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC	CREATION_DATE
23	Electronics	CE	
21	Toys	Toys	
22	Games		
20	Gifts		
27	Video	Video	
26	Unknown		9-Mar-00



# Handling Cells with No Data – aka NULLs

```
SELECT  
pg.PRODUCT_GROUP  
, pg.DESCRPTION  
, pg.SHORT_DESC  
, pg.CREATION_DATE  
FROM PRODUCT_GROUPS pg  
WHERE  
pg.PRODUCT_GROUP BETWEEN 20 AND 27  
AND pg.SHORT_DESC IS NULL  
    OR pg.CREATION_DATE IS NULL  
;
```

Without the parentheses, the results could be very different.

Removing parentheses means the condition after the OR is sufficient to return results, so any row where CREATION\_DATE is null will be returned, regardless of the other two conditions.



# Handling Cells with No Data – aka NULLs

```
SELECT
pg.PRODUCT_GROUP
, pg.DESCRPTION
, pg.SHORT_DESC
, pg.CREATION_DATE
FROM PRODUCT_GROUPS pg
WHERE
pg.PRODUCT_GROUP BETWEEN 20 AND 27
AND pg.SHORT_DESC IS NULL
    OR pg.CREATION_DATE IS NULL
;
```

So rows where the CREATION\_DATE is null but the PRODUCT\_GROUP is NOT between 20 and 27 are now returned.

PRODUCT_GROUP	DESCRIPTION	SHORT_DESC	CREATION_DATE
23	Electronics	CE	
21	Toys	Toys	
18	Unknown		
22	Games		
20	Gifts		
15	Music	Music	
14	Books	Books	
-1	Mixed		
27	Video	Video	
60	Home Improvement	Tools	
0	Unassigned		
26	Unknown		9-Mar-00
75	Baby	Baby	



# ETL Topics

- 1) Explain Plan
- 2) Troubleshooting
- 3) Altering Job Runs
- 4) Job Performance History
- 5) Help

Reference Link: <https://w.amazon.com/index.php/DanGSQLClass/IntroToSqlEtl/Lesson2#HTheWHEREClause2013SQL2019sFilter>





# Lesson 2 Assignment

1. Create a query that pulls a list of warehouses that are in North America (Region 1) and have Amazon inventory from the D\_WAREHOUSES table. Be sure to run an Explain Plan on the query before running it.
2. Edit the query to add the FC Name as an element called 'FC Name', and include only FCs with the word 'Logistics' in their name. Remember to run an Explain Plan first.
3. Check out the table PRODUCT\_GROUPS in Hoot. How many rows does it have? How many columns? Which columns might have NULLs in them? What type of information is in the PRODUCT\_GROUP column? What type of table is it?
4. Create a query that pulls a list of GL Product Group Codes, in numerical order, from the PRODUCT\_GROUPS table. Include the column SHORT\_DESC in your results, and replace any null values in that column with the word 'Unknown'. (Although there isn't a column explicitly named GL\_PRODUCT\_GROUP in the table, one of the columns contains this information. Use BI Metadata and look at the Data Types of the columns, and make an educated guess about which column to pull.
5. Edit the query to include the DESCRIPTION column, to only return results with a GL Product Group value of at least 14, and only return results with a DESCRIPTION in the following list: Books, Universal, Shops, Advertising, or Art.
6. If you haven't studied logic, or are having difficulty wrapping your head around the difference between the results you'd get from WHERE A AND B OR C and WHERE A AND (B OR C), do a little Googling on logic. Concepts like Modus Ponens and Modus Tollens will aid you greatly in writing and understanding SQL.
7. Run Explain plans on the following queries, but DO NOT RUN THEM. These are good examples of bad queries:



# Lesson 2 Homework

## Query 1

```
SELECT ddo.order_id
FROM d_distributor_orders ddo
, d_warehouses fcs
WHERE ddo.warehouse_id = fcs.warehouse_id;
```

## Query 2

```
SELECT ddo.order_id
FROM d_distributor_orders ddo
, d_distributor_order_items doi;
```



END