CPAN 152 RELATIONAL DATABASE DESIGN AND SQL

Lecture 10

Chapter 11 Oracle 12c SQL

Group Functions

OBJECTIVES

- Differentiate between single-row and multiple-row functions
- Use the SUM and AVG functions for numeric calculations
- Use the COUNT function to return the number of records containing non-NULL values
- Use the COUNT(*) function to include records containing NULL values
- Use the MIN and MAX functions with non-numeric fields

OBJECTIVES

- Determine when to use the GROUP BY clause to group data
- Identify when the HAVING clause should be used
- List the order of precedence for evaluating WHERE, GROUP BY and HAVING clauses
- State the maximum depth for nesting group functions
- Nest a group function inside a single-row function)

GROUP FUNCTIONS

- Group functions are also referred to as *multiple-row functions* or *aggregate functions*
- They return one result per group of rows processed
- We will only look at the five most common
- The *multiple-row functions* to be discussed include:
  - SUM
  - MAX
  - MIN
  - AVG
  - COUNT

GROUP FUNCTIONS

- We will look at:
  - The GROUP BY clause which is used to identify groups
  - The HAVING clause which is used to restrict groups

Group Function Concepts

```
SELECT *|columnname, columnname…
FROM tablename
[WHERE condition]
[GROUP BY columnname, columnname…]
[HAVING group condition];
```

FIGURE 11-2 Location of multiple-row functions in the SELECT statement

GROUP FUNCTION CONCEPTS

- Rules for working with group functions
  1. Use the DISTINCT keyword to include only unique values. The ALL keyword is the default and it instructs Oracle 12c to include all values except nulls
  2. All group functions ignore NULL values except COUNT(*). To include NULL values nest the NVL function within the group function
     - For example SELECT MAX(NVL(shipdate, SYSDATE) – orderdate) will substitute the system date for the shipping date of any order that has not been shipped

SUM FUNCTION

- The SUM function is used to calculate the total amount stored in a numeric field for a group of records
- The syntax of the SUM function is:
  - SUM([DISTINCT | ALL] n)
    - Optional DISTINCT keyword instructs Oracle 12c to include only *unique numeric values* in its calculations
    - The ALL keyword instructs Oracle 12c to include *multiple occurrences of numeric values* when totaling a field
    - The default is ALL

SUM FUNCTION

Worksheet | Query Builder

```
1  SELECT SUM(msal + comm) "Total Salary"
2  FROM employees
3  WHERE comm IS NOT NULL;
```

Query Result

SQL | All Ro

	Total Salary
1	7800

The SUM function will total or sum up the values for MSAL + COMM and show one total value for all

The query below shows the individual values that give the total

Worksheet | Query Builder

```
1  SELECT msal + comm "Total Salary"
2  FROM employees
3  WHERE comm IS NOT NULL;
```

Query Result

SQL | All Rows Fetched: 4 in 0.047 seconds

	Total Salary
1	1900
2	1750
3	2650
4	1500

SUM FUNCTION

- The query calculates the total salary from the employees table where the value of the COMM IS NOT NULL
- A column alias is also used here
- The query calculates the total salary first, then totals the salary and commission added together for all employees who receive a commission
- The WHERE clause restricts the rows used in the calculation to those who receive commission
- The query produces only a single row by grouping all of the rows and performing the calculation on the group

SUM FUNCTION

- The next slide shows the SUM function being used to calculate the total salary including BONUS

SUM FUNCTION

Worksheet | Query Builder

```
1  SELECT SUM((msal + comm) + bonus) "Total Gross Salary"
2  FROM employees JOIN salgrades
3  ON (msal BETWEEN lowerlimit AND upperlimit)
4  WHERE comm IS NOT NULL;
```

Query Result

SQL | All Rows Fetched: 1 in 0.078 seconds

	Total Gross Salary
1	8100

AVG FUNCTION

- The AVG function calculates the average of the numeric values in a specific column
- The syntax of the AVG function is:
  - AVG([DISTINCT | ALL] n)
    - Where n is a column containing numeric data

AVG FUNCTION

Worksheet | Query Builder

```
1  SELECT AVG(msal) "Average Salary"
2  FROM employees;
```

Query Result

SQL | All Rows Fetched: 1 in 0.046 seconds

	Average Salary
1	2062.5

AVG FUNCTION

- The previous slide determines the average salary for all employees
- The MSAL for each employee is totaled
- The total is then divided by the number of records that contain non NULL values in the specified field

AVG FUNCTION

Worksheet | Query Builder

```
1  SELECT AVG(msal) "Average Salary"
2  FROM employees
3  WHERE deptno = 20;
```

Query Result

SQL | All Rows Fetched: 1 in 0.047 seconds

	Average Salary
1	2175

AVG FUNCTION

- The query on the previous slide shows the average salary for all employees in department 20

AVG FUNCTION

- Managing NULL values when calculating averages might be an issue when calculating averages
- The EMPLOYEES table on the next slide contains monthly salary and current bonus amount for each employee
- The BONUS column is NULL if no bonus has been earned so far

AVG FUNCTION

Worksheet | Query Builder

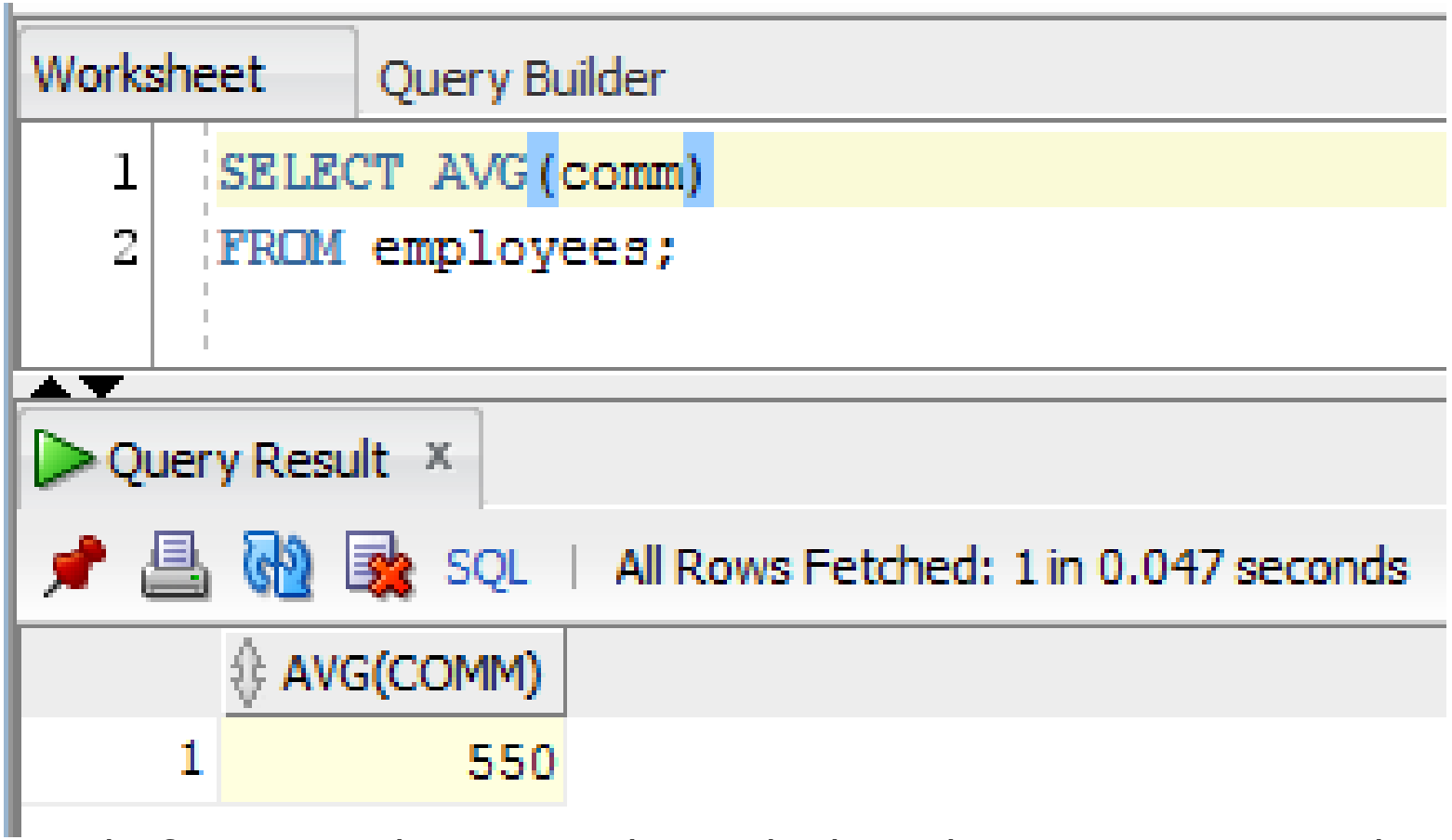
```
1  SELECT empno, ename, msal, comm
2  FROM employees;
```

Query Result

SQL | All Rows Fetched: 14 in 0.062 seconds

	EMPNO	ENAME	MSAL	COMM
1	7369	SMITH	800	(null)
2	7499	ALLEN	1600	300
3	7521	WARD	1250	500
4	7566	JONES	2975	(null)
5	7654	MARTIN	1250	1400
6	7698	BLAKE	2850	(null)
7	7782	CLARK	2450	(null)

AVG FUNCTION



Only four records are used to calculate the average, since the COMM value for remaining employees is a NULL value thy are ignored

AVG FUNCTION

Worksheet | Query Builder

```
1  SELECT ROUND(AVG(NVL(comm, 0)), 2) "Total Average Comm"
2  FROM employees;
```

Query Result

SQL | All Rows Fetched: 1 in 0.046 seconds

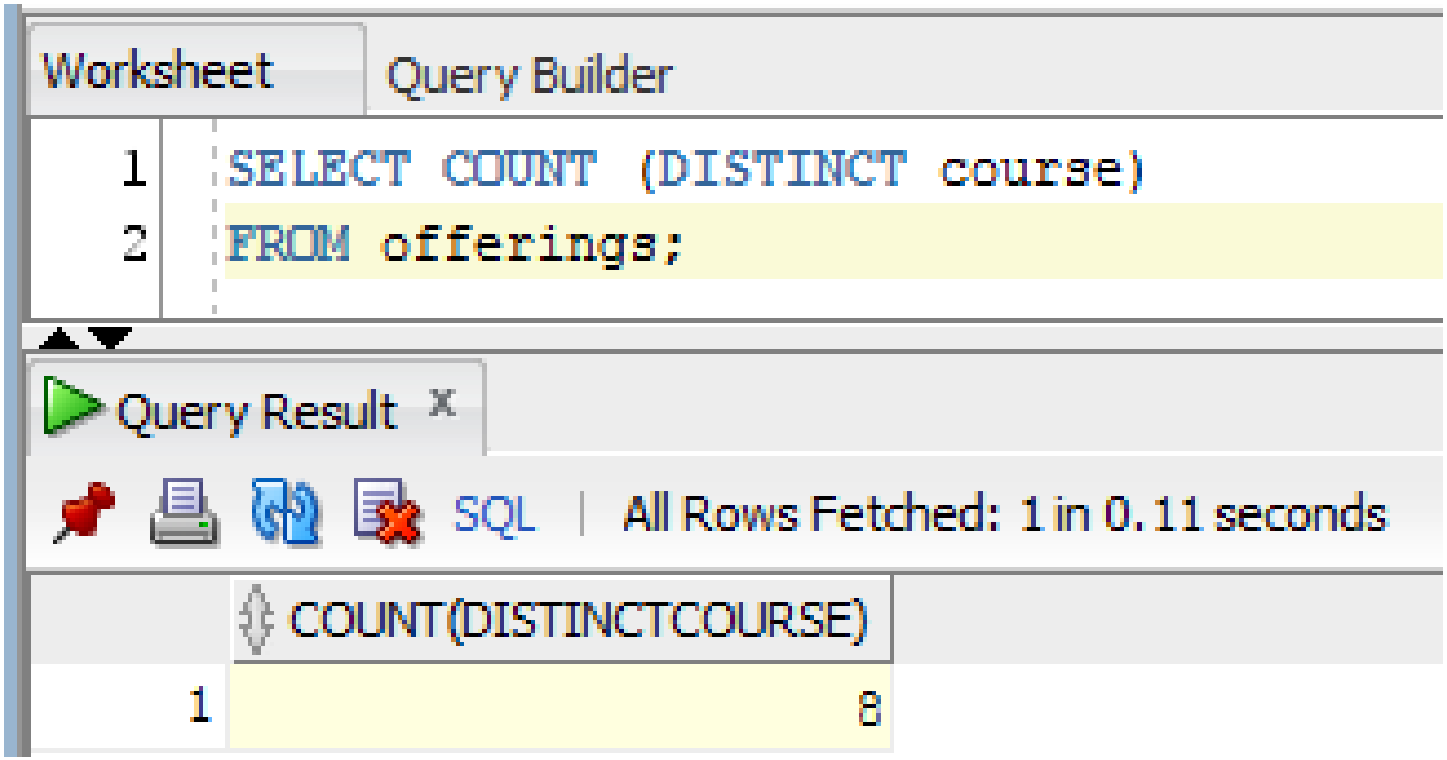
	Total Average Comm
1	157.14

(3000 + 1200 + 1500 + 1900 + 0) / 14 = 157.14
In this case the NVL function is used to convert the NULL value for the remaining employees to a 0, there are now 14 values calculated for the AVG function, the ROUND is used to format the output

COUNT FUNCTION

- Depending on the argument used, the COUNT function can either:
  - Count the records that have non-NULL values in a specified field
  - Count the total records that meet a specific condition, including those containing NULL values
  - The syntax of the COUNT function is:
    - COUNT(* | [DISTINCT|ALL] c)
      - Where c represents the any type of column, numeric or non-numeric

COUNT FUNCTION



A column alias could be used to make the heading more attractive

COUNT FUNCTION

- Notice the DISTINCT keyword precedes the column name in the argument of the COUNT function
- This instructs Oracle to count each different value in the COURSE column
- If it had appeared directly after the SELECT keyword in the statement, it would apply to the entire COUNT function and would have been interpreted to mean that only duplicate rows should be suppressed, rather than duplicate COURSE values

COUNT FUNCTION

Worksheet | Query Builder

```
1  SELECT DISTINCT COUNT(course)
2  FROM offerings;
```

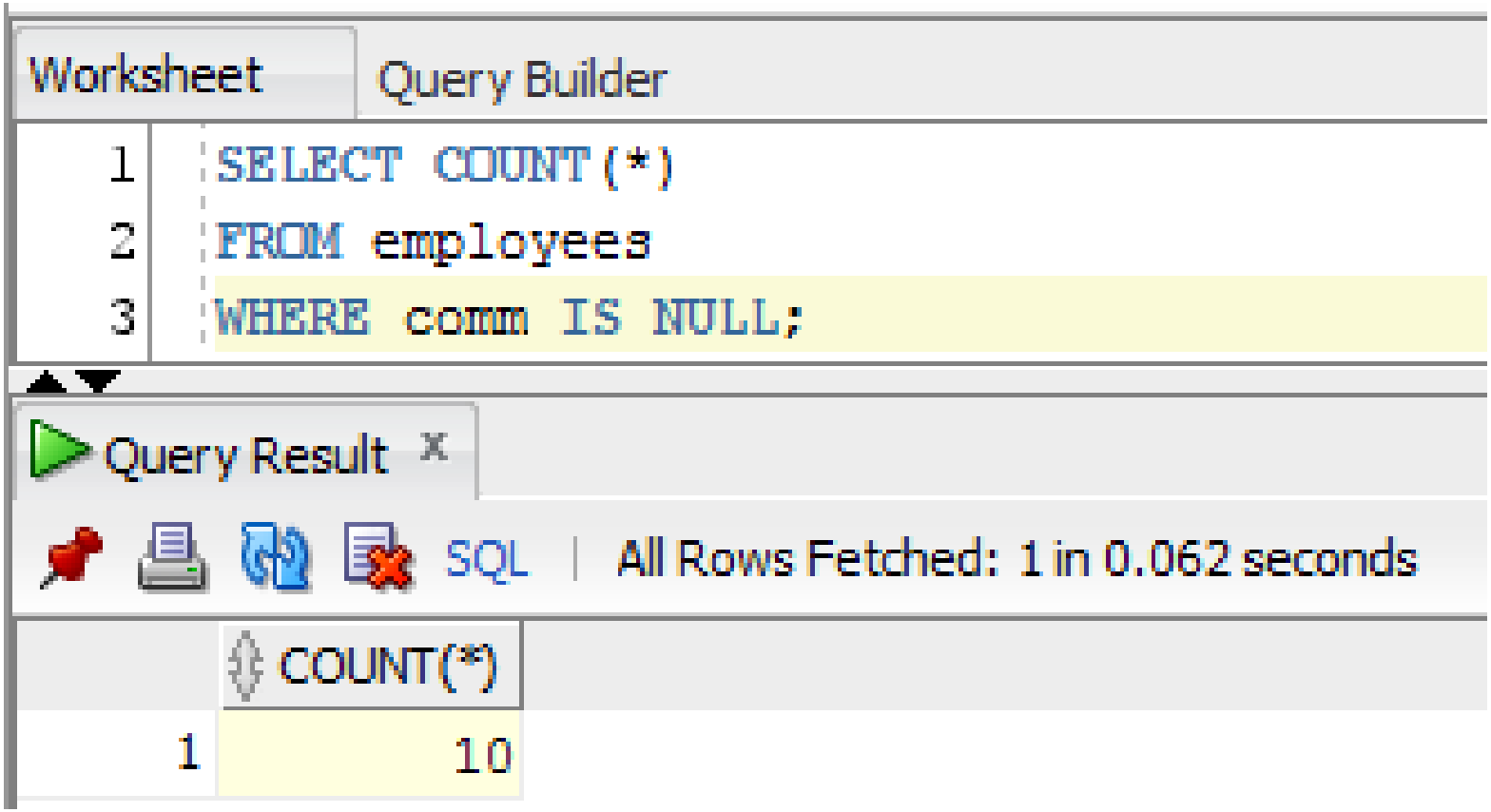
Query Result

SQL | All Rows Fetched: 1 in 0.047 seconds

	COUNT(COURSE)
1	13

Here, the DISTINCT keyword is included as part of the SELECT clause. Oracle 12c returns an actual count of the rows in the OFFERINGS table

COUNT FUNCTION

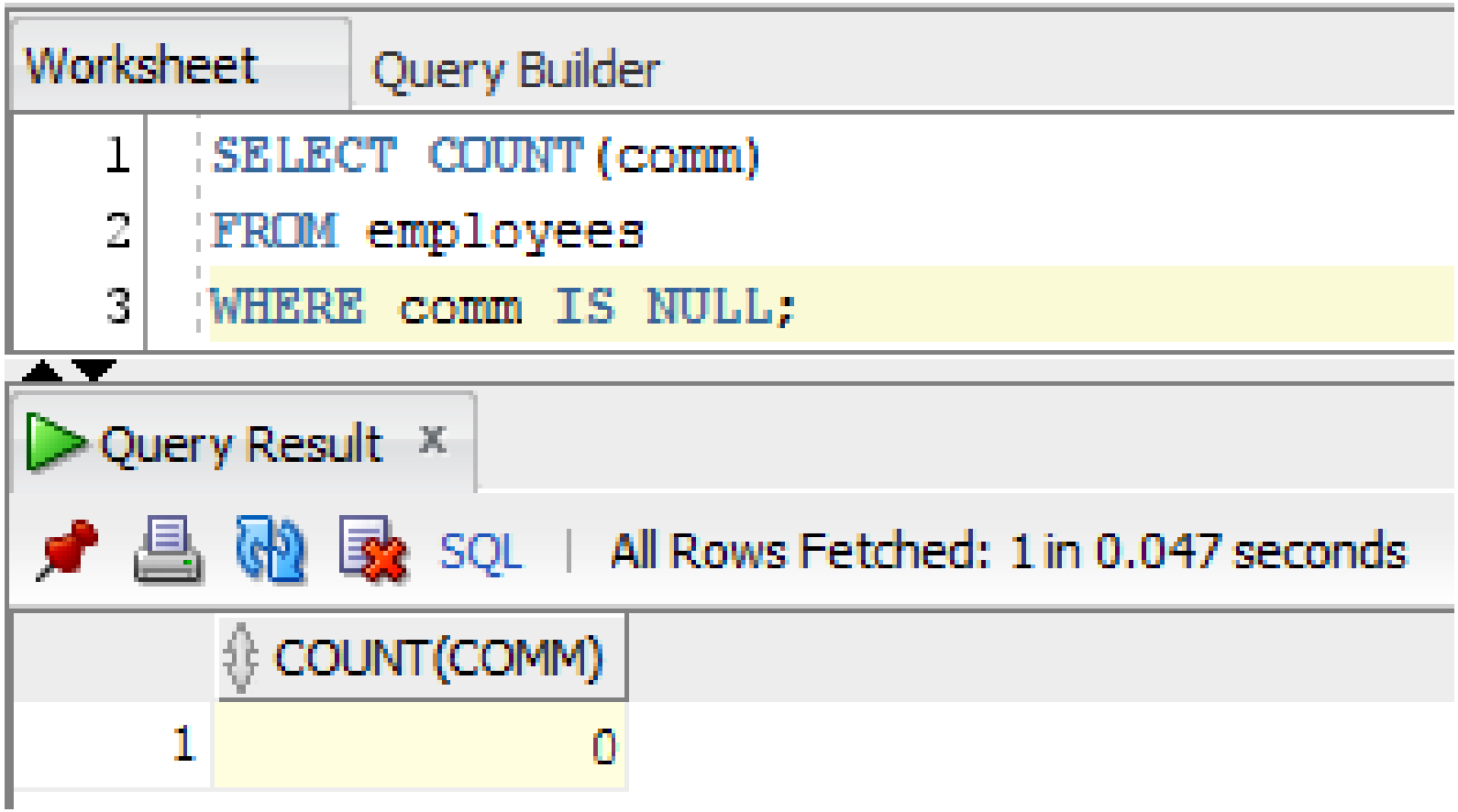


The number of employees that have NULL value for the comm

COUNT FUNCTION

- When the argument supplied in the COUNT function is an asterisk, the existence of the entire record is counted
- When this happens, NULLs are not discarded by the COUNT function
- The WHERE clause restricts the rows to be counted, in this case only count the rows where the comm is NULL
- We will try this using the other variation for the COUNT function

COUNT FUNCTION



COUNT FUNCTION

- The asterisk was replaced with the comm column in the COUNT argument
- The WHERE clause restricts the records to only those where the comm is NULL
- Since the comm column is NULL, the COUNT function can only count rows where the comm is NULL
- But COUNT won't count NULL records, so the result is zero
- *Whenever NULL values may affect the COUNT function, you should use an asterisk as the argument rather than the column name*

MAX FUNCTION

- The MAX function returns the largest value stored in a specified column
- The syntax for the MAX function is:
  - MAX([DISTINCT | ALL] c)
    - Where **c** can represent any numeric, character, or date field

MAX FUNCTION

Worksheet | Query Builder

```
1  SELECT MAX(msal + NVL(comm, 0)) "Highest Salary"
2  FROM employees;
```

Query Result

SQL | All Rows Fetched: 1 in 0.046 seconds

	Highest Salary
1	5000

MAX FUNCTION

- The query returns the highest salary for an employee, but it does not tell us which employee it is
- The column name cannot be added directly to the query since SQL will think you want to group items based on other information

MAX FUNCTION

Worksheet | Query Builder

```
1  SELECT ename, MAX(msal + NVL(comm, 0)) "Highest Salary"
2  FROM employees;
```

Query Result

SQL | Executing:SELECT ename, MAX(msal + NVL(comm, 0)) "Highest Salary"FROM employees in 0 seconds

```
ORA-00937: not a single-group group function
00937. 00000 -  "not a single-group group function"
*Cause:
*Action:
Error at Line: 1 Column: 8
```

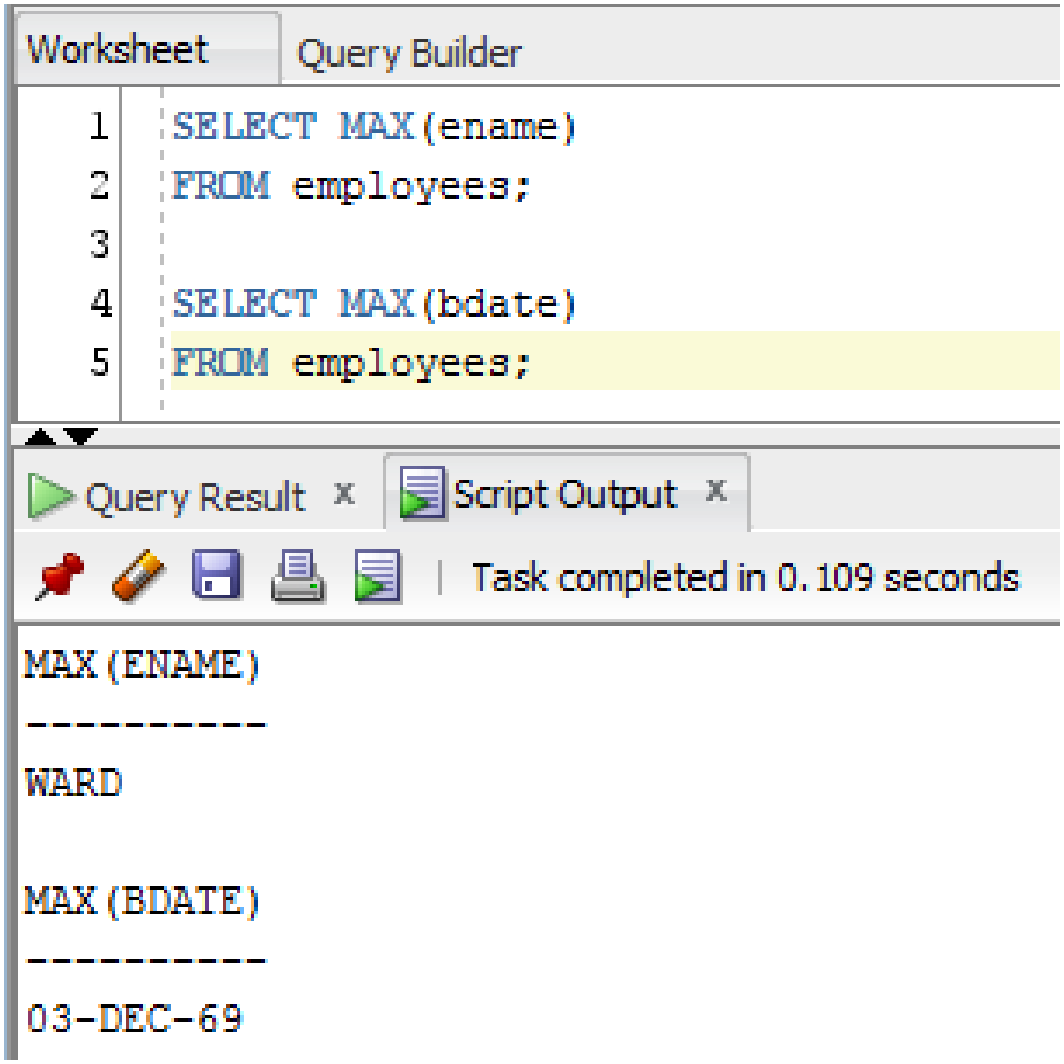
This returns an error since it is expecting to be grouped on title but the query is not set up to perform grouping, more on this later, the query contains an aggregate MAX, and a non-aggregate column title, a GROUP BY needs to be included if this is the case

The automatic code suggestion may automatically add the GROUP BY for you

MAX FUNCTION

- The MAX function can also be used with non-numeric data
- The output shows the first value that occurs when sorted in descending order
- Similarly, the output shows the most recent date, which is considered to have the highest value
- If the MAX function is applied to character data, a Z is considered to be higher than an A

MAX FUNCTION



The MAX function applied to a character field ENAME, and then to a date field BDATE

MIN FUNCTION

- The MIN function is the opposite to the MAX function, in that it returns the smallest value in a specified column
- The syntax for the MIN function is:
  - MIN([DISTINCT | ALL] c)
    - Where **c** can represent any numeric, character, or date field

MIN FUNCTION

Worksheet | Query Builder

```
1  SELECT MIN(msal +NVL(comm, 0))
2  FROM employees;
3
4  SELECT MAX(ename)
5  FROM employees;
6
7  SELECT MAX(bdate)
8  FROM employees;
```

Query Result | Script Output

Task completed in 0.141 seconds

```
MIN(MSAL+NVL(COMM,0))
---------------------
                  800

MAX(ENAME)
----------
WARD

MAX(BDATE)
----------
03-DEC-69
```

GROUP BY CLAUSE

- The query on the following slide displays the average salary for all employees in department 30
- What if you wanted to return the average salary for each department? You could rewrite the query used previously and change the WHERE clause to be each of the various departments

GROUP BY CLAUSE

Worksheet | Query Builder

```
1  SELECT ROUND(AVG(msal), 2)
2  FROM employees
3  WHERE deptno = 30;
4
5  SELECT ROUND(AVG(msal), 2)
6  FROM employees
7  WHERE deptno = 20;
```

Script Output | Query Result

Task completed in 0.094 seconds

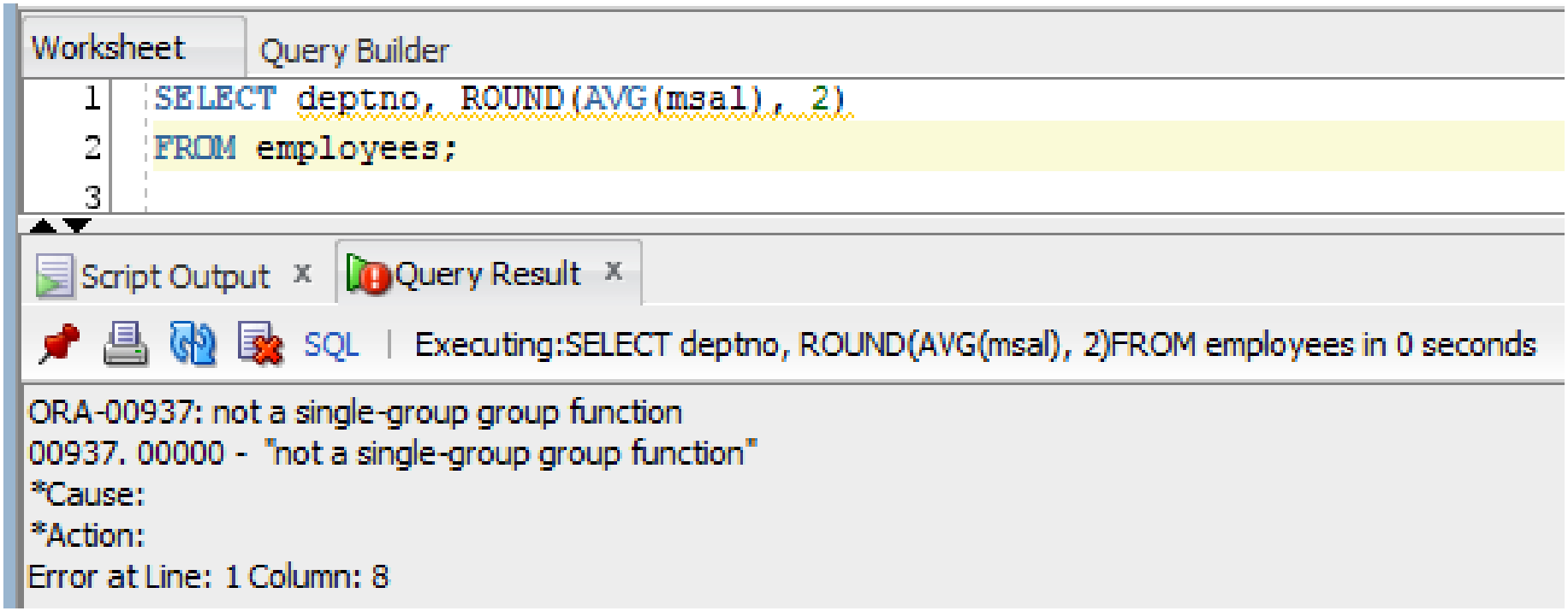
```
ROUND(AVG(MSAL),2)
------------------
           1541.67

ROUND(AVG(MSAL),2)
------------------
              2175
```

Average calculated for two different departments, one SELECT statement for each department

GROUP BY CLAUSE

- An alternative is to divide the records in the EMPLOYEES table into groups, then calculate the average for each group
- In order to improve on the previous attempt, it would have to be done in a single query
- We can use the GROUP BY clause to accomplish this
- The syntax for the GROUP BY clause is:
  - GROUP BY column_name [, column_name, …],
    - Where the column_name is the column(s) to be used to create the *groups or sets of data*

GROUP BY CLAUSE



This SELECT statement includes a single column named DEPTNO, and an AVG group function. Since a GROUP BY clause was not included, the error is returned

GROUP BY CLAUSE

- To specify which groups should be created, add the GROUP BY clause to the SELECT statement
- When using the GROUP BY clause, remember the following:
  - If a group function is used in the SELECT clause, then any individual columns listed in the SELECT clause must also be listed in the GROUP BY clause
  - Columns used to group data in the GROUP BY clause do not have to be listed in the SELECT clause. They are only included in the SELECT clause to have the groups identified in the output
  - *Column aliases* cannot be used in the GROUP BY clause

GROUP BY CLAUSE

- Results returned from a SELECT statement that include a GROUP BY clause are not listed in any order. To present the results in a particular sequence, use the ORDER BY clause.
- The SELECT clause should display *one record for each department of employees.* The deptno should display first, then its average salary
- Let us now perform the same query, this time with the GROUP BY clause

GROUP BY CLAUSE

Worksheet | Query Builder

```
1  SELECT deptno, ROUND(AVG(msal), 2)
2  FROM employees
3  GROUP BY deptno;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.045 seconds

	DEPTNO	ROUND(AVG(MSAL),2)
1	30	1541.67
2	20	2175
3	10	2916.67

The single column name listed in the SELECT clause is included in the GROUP BY clause

When the query executes, the records in the EMPLOYEES table are first grouped by DEPTNO, then the average salary for each department is calculated

Since the deptno is listed in the SELECT clause as well, it is displayed along with the average salary for each department

GROUP BY CLAUSE

- Consider the next example
- It shows an inappropriate use of the GROUP BY clause
- The query is syntactically correct but you do not need the GROUP BY to produce the output that is given
- Have a look can you see why?

GROUP BY CLAUSE

Worksheet | Query Builder

```
1  SELECT ename, MAX(msal)
2  FROM employees
3  GROUP BY ename;
```

Script Output | Query Result

SQL | All Rows Fetched: 13 in 0.047 seconds

	ENAME	MAX(MSAL)
1	ALLEN	1600
2	JONES	2975
3	FORD	3000
4	CLARK	2450
5	MILLER	1300
6	SMITH	800
7	WARD	1250
8	MARTIN	1250
9	SCOTT	3000
10	TURNER	1500
11	ADAMS	1100
12	BLAKE	2850
13	KING	5000

Can you see why this is an inappropriate use of a GROUP BY clause?

Could you obtain the same results with another query?

What would that query be?

GROUP BY CLAUSE

Worksheet | Query Builder

```
1  SELECT ename, msal
2  FROM employees;
```

Script Output | Query Result

SQL | All Rows Fetched: 14 in 0.046 seconds

	ENAME	MSAL
1	SMITH	800
2	ALLEN	1600
3	WARD	1250
4	JONES	2975
5	MARTIN	1250
6	BLAKE	2850
7	CLARK	2450
8	SCOTT	3000
9	KING	5000
10	TURNER	1500
11	ADAMS	1100
12	JONES	800
13	FORD	3000
14	MILLER	1300

The GROUP BY was not needed since each title only occurs once, you were creating a situation where each group was the individual title for the book

This query gives you the same output without grouping the data

GROUP BY CLAUSE

- Let us look at another example of a GROUP BY clause, this time using the SUM function
- The billing department of a company requests a list of the amount due from each customer for each order they have placed
- This will require the use of the GROUP BY function to group the rows together
- We will need to see the order number, customer number, and SUM for each order by the customer with the calculation to determine the amount due

GROUP BY CLAUSE

```
1  SELECT customer#, o.order#, SUM(quantity * paideach) "Order Total"
2  FROM orders o JOIN orderitems oi ON (o.order# = oi.order#)
3  JOIN books b ON (oi.isbn = b.isbn)
4  GROUP BY customer#, o.order#
5  ORDER BY customer#, o.order#;
```

Script Output | Query Result

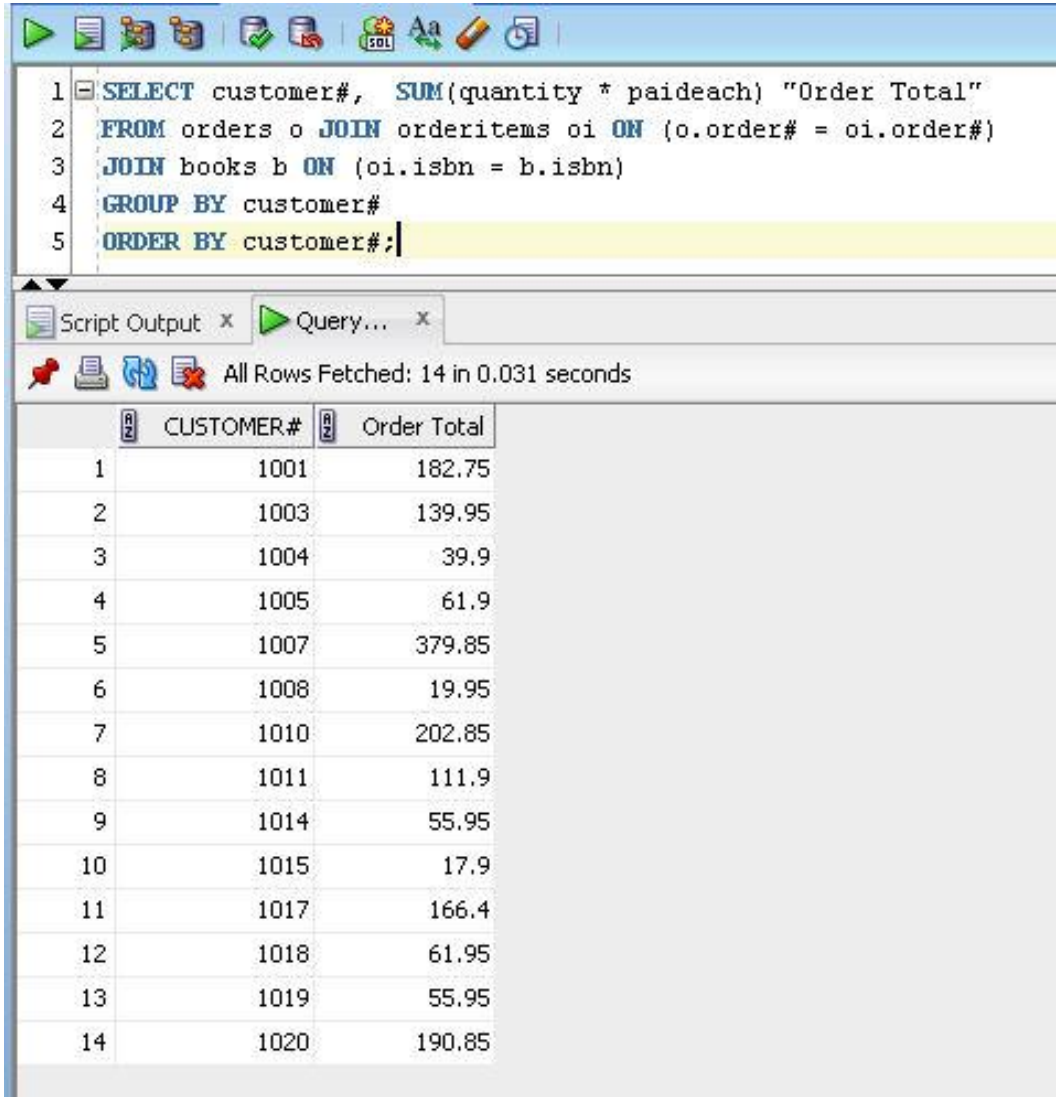
All Rows Fetched: 21 in 0.047 seconds

	CUSTOMER#	ORDER#	Order Total
1	1001	1003	106.85
2	1001	1018	75.9
3	1003	1006	54.5
4	1003	1016	85.45
5	1004	1008	39.9
6	1005	1000	19.95
7	1005	1009	41.95
8	1007	1007	335.85
9	1007	1014	44
10	1008	1020	19.95
11	1010	1001	117.4
12	1010	1011	85.45
13	1011	1002	111.9
14	1014	1013	55.95
15	1015	1017	17.9
16	1017	1012	166.4
17	1018	1005	39.95
18	1018	1019	22
19	1019	1010	55.95
20	1020	1004	170.9
21	1020	1015	19.95

Since the SELECT statement includes both customer# and order#, both of these columns must be listed in the GROUP BY clause

If the order# were omitted, the query would have returned only the amount due for each customer, not the amount due by customer for each order

GROUP BY CLAUSE



```
1  SELECT customer#,  SUM(quantity * paideach) "Order Total"
2  FROM orders o JOIN orderitems oi ON (o.order# = oi.order#)
3  JOIN books b ON (oi.isbn = b.isbn)
4  GROUP BY customer#
5  ORDER BY customer#;
```

Script Output | Query...

All Rows Fetched: 14 in 0.031 seconds

	CUSTOMER#	Order Total
1	1001	182.75
2	1003	139.95
3	1004	39.9
4	1005	61.9
5	1007	379.85
6	1008	19.95
7	1010	202.85
8	1011	111.9
9	1014	55.95
10	1015	17.9
11	1017	166.4
12	1018	61.95
13	1019	55.95
14	1020	190.85

This shows the result with the order# omitted from the query

Can you see the difference in how the data is grouped?

The result displays the total for each customer but no longer broken down by the order number

HAVING CLAUSE

- The HAVING clause is used to restrict groups returned by a query
- You need to use the HAVING clause to restrict groups, since the WHERE clause *cannot contain any grouping functions*
- The WHERE clause restricts the records that will appear in the query. The HAVING clause will specify which groups will be displayed in the results
- The HAVING clause is a WHERE clause for groups

HAVING CLAUSE

- The syntax for the HAVING clause is:
  - HAVING group_function comparison_operator value
- In addition, the logical operators AND, OR and NOT *can be used to join conditions* in the HAVING clause
- In the next slide, the HAVING clause is used to limit the groups displayed to those *categories with an average salary of 2000.00 or more*

HAVING CLAUSE

Worksheet | Query Builder

```
1  SELECT deptno, TO_CHAR(ROUND(AVG(msal + NVL(comm, 0)), 2), '99999.99')
2  FROM employees
3  GROUP BY deptno
4  HAVING  AVG(msal + NVL(comm, 0)) > 2000;
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0.047 seconds

	DEPTNO	TO_CHAR(ROUND(AVG(MSAL+NVL(COMM,0)),2),'99999.99')
1	20	2175.00
2	10	2916.67

HAVING CLAUSE

Worksheet | Query Builder

```
1  SELECT deptno, TO_CHAR(ROUND(AVG(msal + NVL(comm, 0)), 2), '99999.99')
2  FROM employees
3  GROUP BY deptno
4  -- HAVING  AVG(msal + NVL(comm, 0)) > 2000;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.047 seconds

	DEPTNO	TO_CHAR(ROUND(AVG(MSAL+NVL(COMM,0)),2),'99999.99')
1	30	1908.33
2	20	2175.00
3	10	2916.67

Without the use of the HAVING clause, all 3 departments show

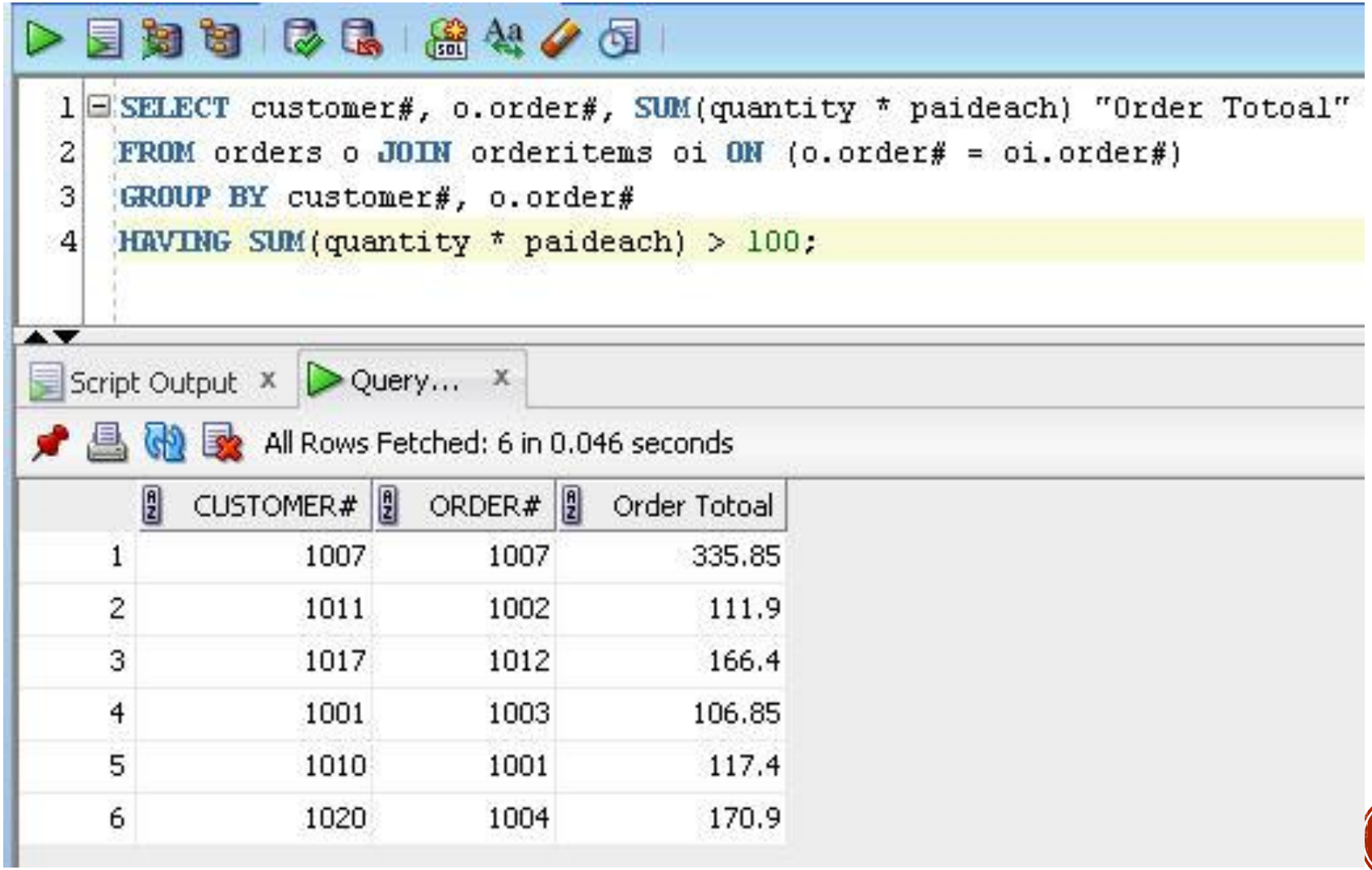
HAVING CLAUSE

- Whenever a SELECT statement includes all three clauses, the order in which they are evaluated is as follows:
  1. WHERE clause
  2. GROUP BY clause
  3. HAVING clause
- The WHERE clause filters the data *before* grouping, whereas the HAVING clause filters the groups *after* the grouping occurs

HAVING CLAUSE

- Let us now revisit the query that produced the list of totals on each order by a customer from slide 41
- Management now would like this query modified so it will only display customer orders where the total on the order is greater than $100.00

HAVING CLAUSE



```
1  SELECT customer#, o.order#, SUM(quantity * paideach) "Order Totoal"
2  FROM orders o JOIN orderitems oi ON (o.order# = oi.order#)
3  GROUP BY customer#, o.order#
4  HAVING SUM(quantity * paideach) > 100;
```

Script Output | Query...

All Rows Fetched: 6 in 0.046 seconds

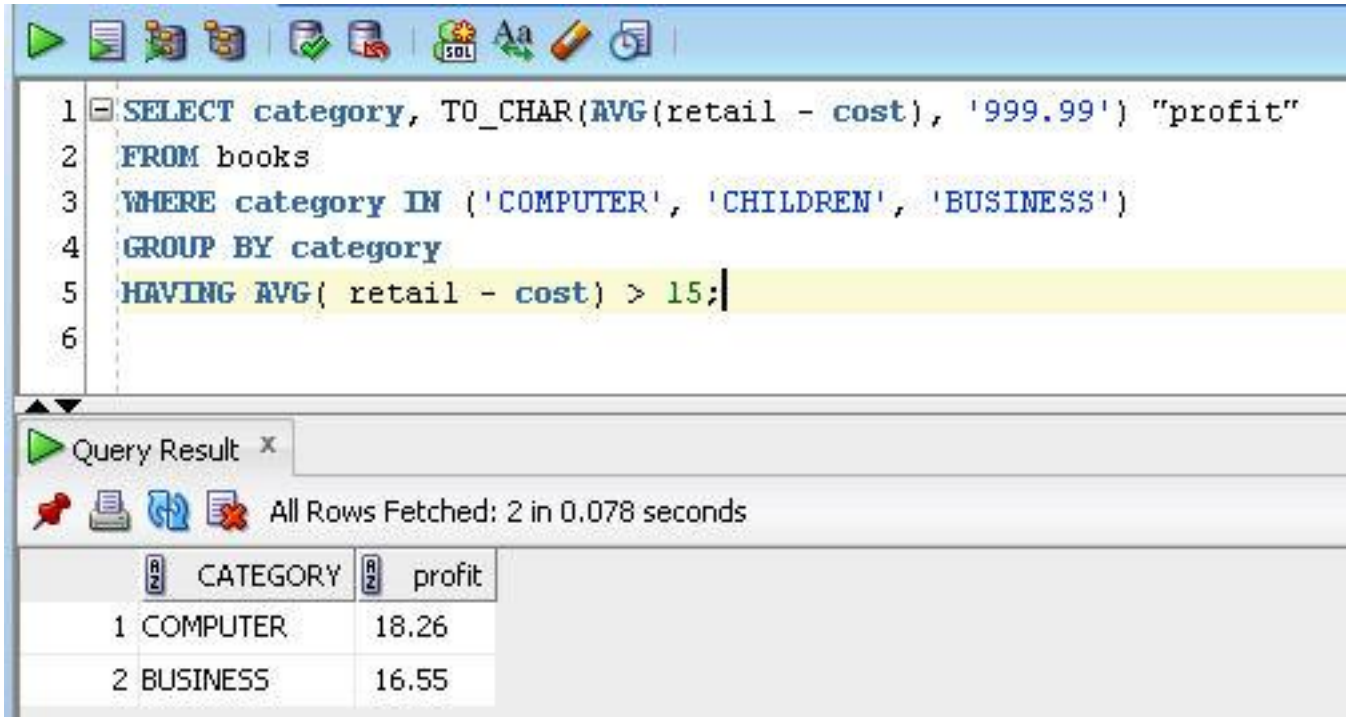
	CUSTOMER#	ORDER#	Order Totoal
1	1007	1007	335.85
2	1011	1002	111.9
3	1017	1012	166.4
4	1001	1003	106.85
5	1010	1001	117.4
6	1020	1004	170.9

HAVING CLAUSE

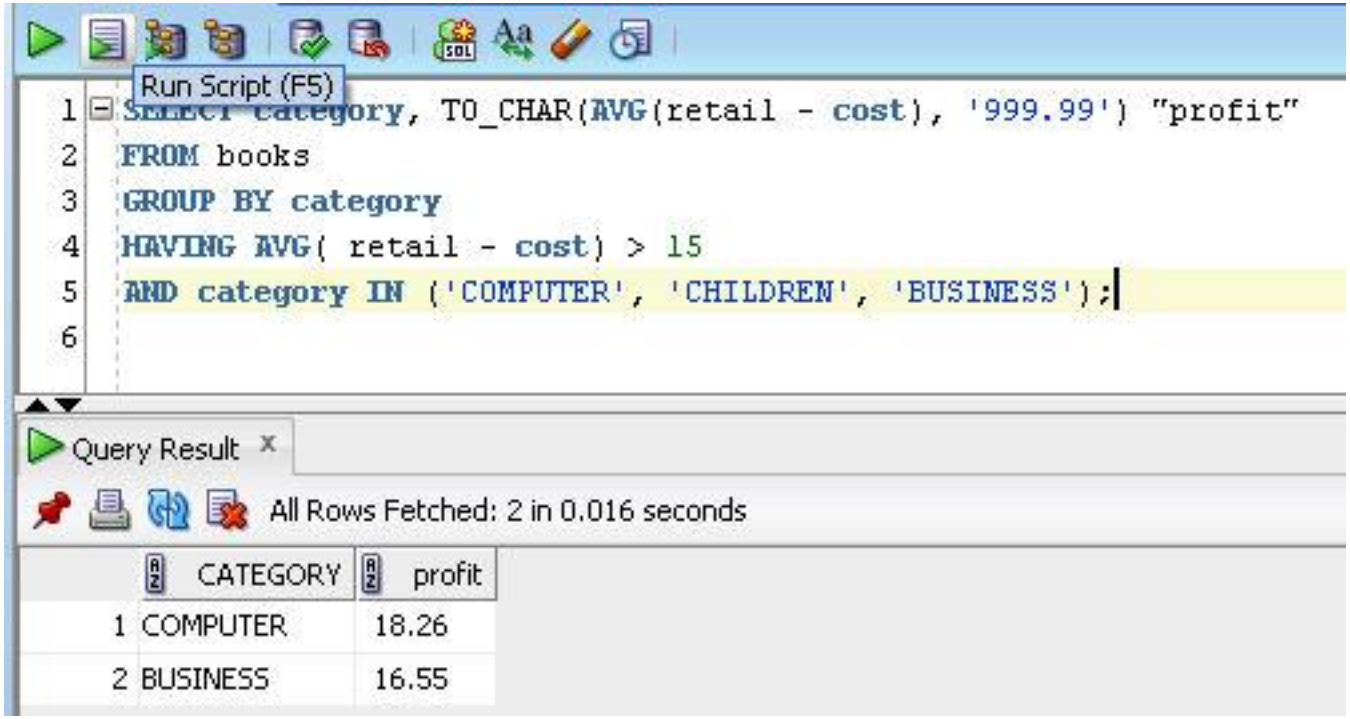
- If we were to modify look at a query where only certain categories were to appear in the output, where would the filtering task take place?
- The query needs to include the following data filters:
  - Show only book categories with an average profit greater than $15.00
  - Include only the categories Computer, Children and Business
- The first filtering task should be done with the HAVING clause because it output is depended on an aggregated value
- The second filtering task should be handled with a WHERE clause at the row level to exclude these categories before the aggregation in performed

HAVING CLAUSE

This is the proper way to resolve the query, filter the rows you need in your query then group the remaining rows



HAVING CLAUSE



HAVING CLAUSE

- Although the results are the same, the query produced the filtering in the HAVING clause after the aggregation was performed so all the categories were aggregated first.
- This is inefficient and considered poor programming practice.
- The statement must process all the rows in the BOOKS table with the aggregate calculation then eliminate categories
- Using the WHERE clause instead the rows that are not required are eliminated at the beginning of the query process before the aggregation takes place

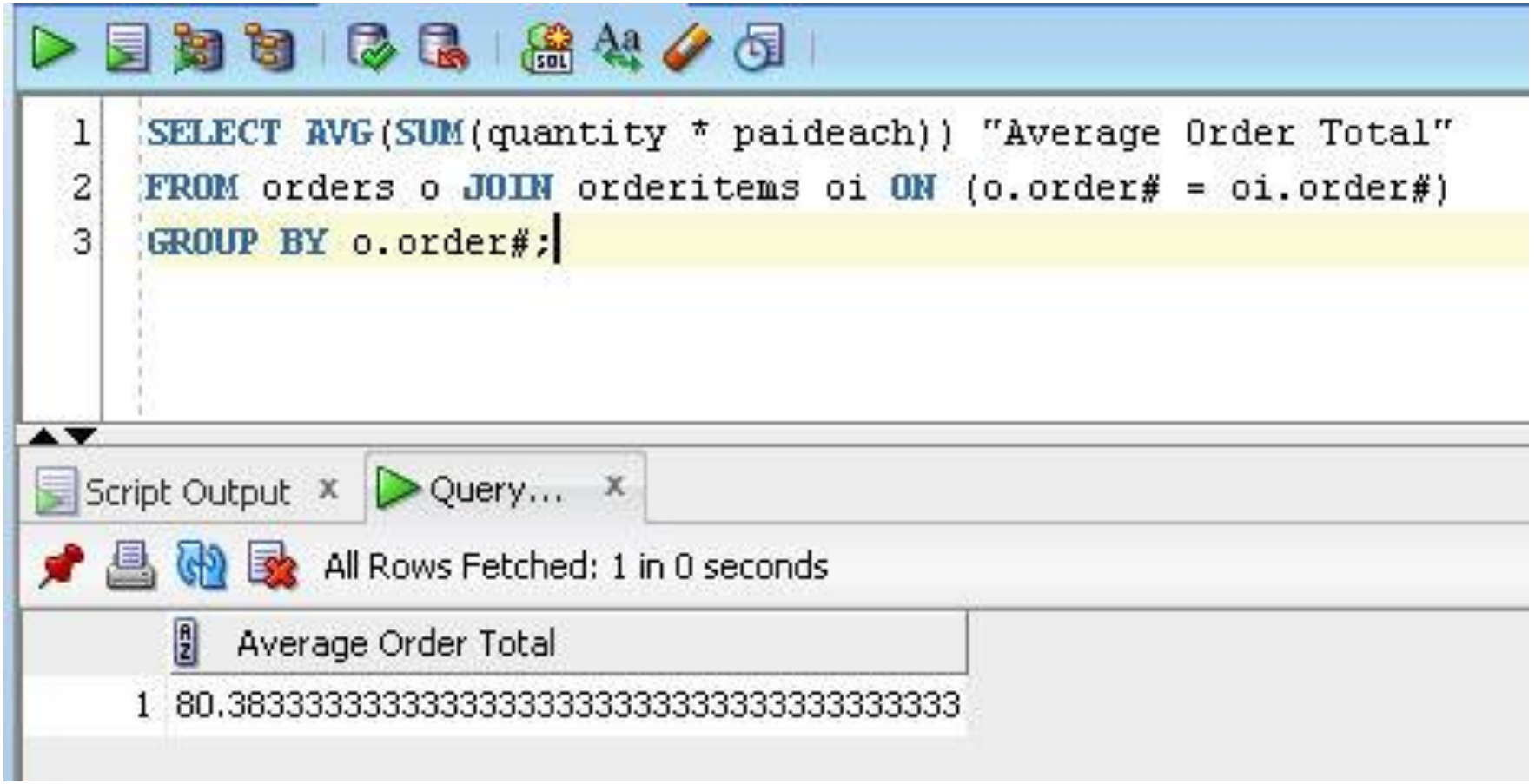
NESTING GROUP FUNCTIONS

- As with single-row functions, when group functions are nested, the *inner function is resolved first*
- The result of the inner function is *passed back as input to the outer function*
- Unlike single-row functions that have *no restriction on how many nesting levels can occur,* group functions can only be *nested to a depth of two*
- We have already seen that a group function can be *nested inside* a single-row function
- A group function can also be *nested inside* a group function

NESTING GROUP FUNCTIONS

- We would like to determine the average total amount for each order
- To accomplish this, the SUM function is nested inside the AVG function
- The GROUP BY clause first groups all of the records based on the order# column
- The total order is then calculated for each order by the SUM function
- The AVG function is then used to calculate the average of the total order amounts calculated by the SUM function
- The result is one record displaying the average total amount due for the orders currently in the ORDERS table

NESTING GROUP FUNCTIONS



NESTING GROUP FUNCTIONS

- Keep in mind that group functions can only be nested to only two levels
- The query must include a GROUP BY clause
- The inner group function creates the aggregated result fro each group
- The outer group function performs an aggregation on the grouped results