Zachary Campbell / Student Number: 07382936 / Lab 5

COMP400725
Lab Book 5

- ➔ Create a folder called ***Lab5*** in your ***COMP40725*** folder. You may wish to have this inside a folder that is being synced by Google Drive.
- ➔ Use Notepad++ or TextWrangler to write SQL queries. Test these queries in your ***Oracle 11g XE Database*** system. You can either use ***SQL Command Line*** or ***Oracle Web Interface*** to test your queries. Do not use ***MySQL*** as the commands will not work.
- ➔ Use comments in your sql queries: -- is for a single line, and /*... */ for multiline.
- ➔ Submit this document with the following name **"Lab5_LastName_FirstName_StudentNumber".**

NB. Web links in PDF version of document may be corrupt.

Please complete this practical using the web interface for Oracle. See "How to Launch Oracle and Execute SQL Commands" document on Moodle.

This lab requires you to create and test the following queries.

1. Run the example SQL DDL/DML code provided in ***Customer.sql*** available on Moodle. Please note you must cut and paste this into the SQL Command editor in the online Interface of Oracle. It will create 3 tables and populate them with data. Read the code so you understand the tables, columns, and their relationships. Enter the create table commands one at a time and then use the following to save time

```
BEGIN
<All your insert statements>
END
```

[There is no requirement to paste information here]

2. Create a query that lists each order number (CUST_ORDER table) with the first name and surname of the employee associated with the order.

[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

3. Create a self-join on the employee table, where the results should show each employee's name and their manager's name. Rename the columns in the results to be more appropriate.

[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

4. Use an outer join to list all employees' first name and surname with the customer order's (ORDER_NBR) they are associated with (i.e. sales_emp_id in the CUST_ORDER table). As it is an outer join, it should also list employees that have never had a sale.

[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

5. Create a query to display the total sales price of all orders (i.e. 1 value).

[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

6. Create a query to display the average sale_price of an order by each employee.

[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

7. Create a query to display the total sales price of all orders from each customer.

[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

8. Create a query to display the total sales price of all orders from each customer - where only customers who spent more than 1000 are considered.

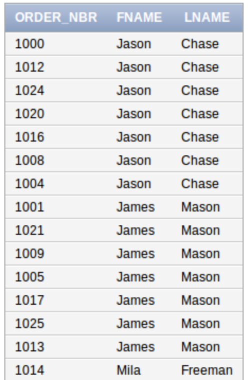
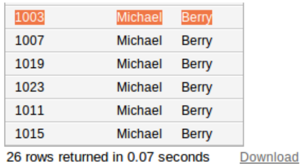
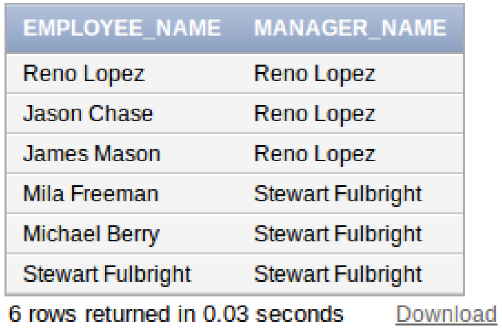
[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

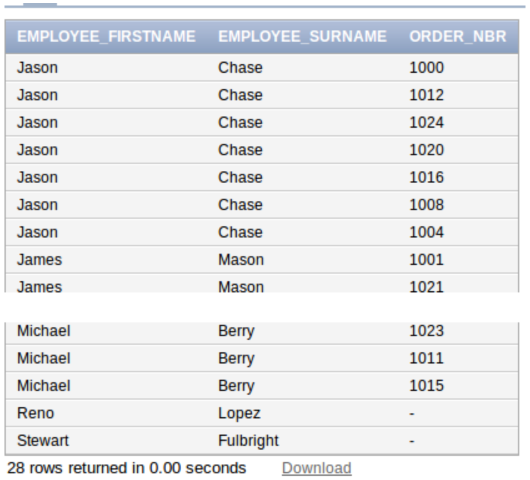
9. Demonstrate the use of a CUBE query on two columns (from any 2 tables in the sample database -can be same as lecture example if you add a category column in the part table).

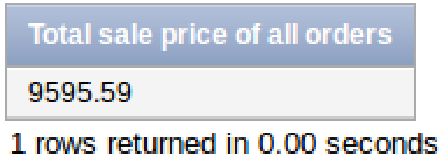
[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

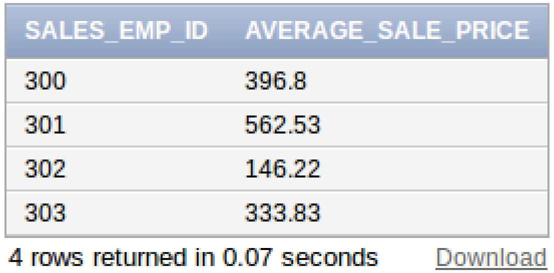
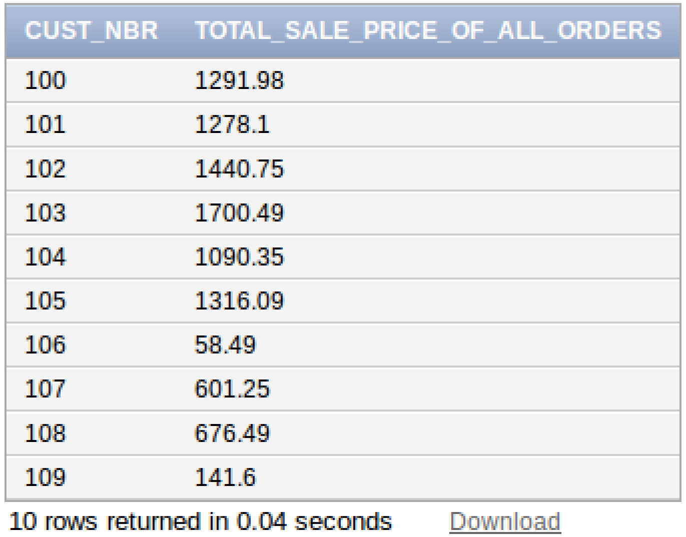
10. Demonstrate the use of the ROLLUP query on two columns different to those in the CUBE query above.

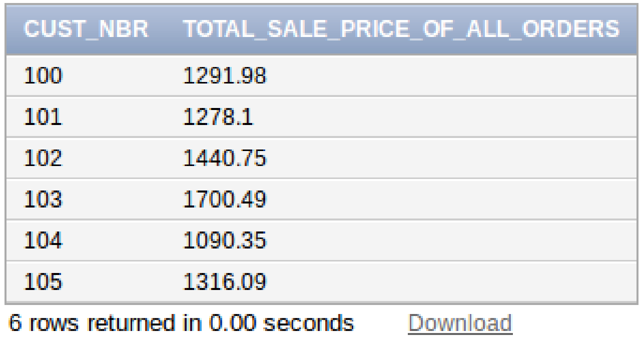
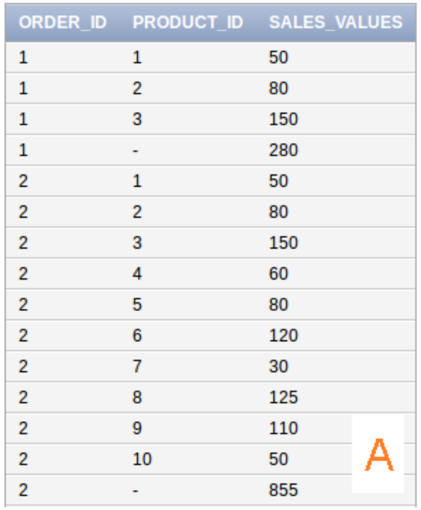
[Paste you SQL Command here including comments]
[Paste a screen shot of the output here]

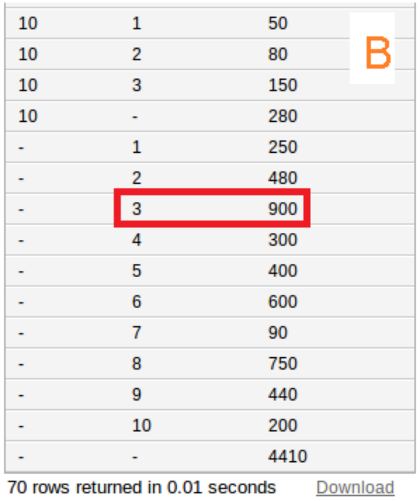
<table>
<tr><th></th><th>Create Table Statements</th><th>Screenshot</th><th>Comments</th></tr>
<tr>
<td><b>Q.2</b></td>
<td><b>SELECT ORDER_NBR,FNAME,LNAME FROM CUST_ORDER JOIN EMPLOYEE
ON SALES_EMP_ID = EMP_ID;</b></td>
<td>

<table>
<tr><th>ORDER_NBR</th><th>FNAME</th><th>LNAME</th></tr>
<tr><td>1000</td><td>Jason</td><td>Chase</td></tr>
<tr><td>1012</td><td>Jason</td><td>Chase</td></tr>
<tr><td>1024</td><td>Jason</td><td>Chase</td></tr>
<tr><td>1020</td><td>Jason</td><td>Chase</td></tr>
<tr><td>1016</td><td>Jason</td><td>Chase</td></tr>
<tr><td>1008</td><td>Jason</td><td>Chase</td></tr>
<tr><td>1004</td><td>Jason</td><td>Chase</td></tr>
<tr><td>1001</td><td>James</td><td>Mason</td></tr>
<tr><td>1021</td><td>James</td><td>Mason</td></tr>
<tr><td>1009</td><td>James</td><td>Mason</td></tr>
<tr><td>1005</td><td>James</td><td>Mason</td></tr>
<tr><td>1017</td><td>James</td><td>Mason</td></tr>
<tr><td>1025</td><td>James</td><td>Mason</td></tr>
<tr><td>1013</td><td>James</td><td>Mason</td></tr>
<tr><td>1014</td><td>Mila</td><td>Freeman</td></tr>
</table>

<table>
<tr><td>1003</td><td>Michael</td><td>Berry</td></tr>
<tr><td>1007</td><td>Michael</td><td>Berry</td></tr>
<tr><td>1019</td><td>Michael</td><td>Berry</td></tr>
<tr><td>1023</td><td>Michael</td><td>Berry</td></tr>
<tr><td>1011</td><td>Michael</td><td>Berry</td></tr>
<tr><td>1015</td><td>Michael</td><td>Berry</td></tr>
</table>
26 rows returned in 0.07 seconds Download
Please note not all columns could be shown in a screenshot.
</td>
<td>This query lists each of the order numbers in the CUST_ORDER table where the first name and surname of the employee associated with the order.
This was made possible by joining the SALES_EMP_ID in the CUST_ORDER table with that of the Employee on EMP_ID</td>
</tr>
<tr>
<td><b>Q.3</b></td>
<td><b>SELECT
a.FNAME || ' ' || a.LNAME AS Employee_Name,
b.FNAME || ' ' || b.LNAME AS Manager_Name
FROM employee a join employee b
ON
a.MANAGER_EMP_ID=b.EMP_ID;</b></td>
<td>

<table>
<tr><th>EMPLOYEE_NAME</th><th>MANAGER_NAME</th></tr>
<tr><td>Reno Lopez</td><td>Reno Lopez</td></tr>
<tr><td>Jason Chase</td><td>Reno Lopez</td></tr>
<tr><td>James Mason</td><td>Reno Lopez</td></tr>
<tr><td>Mila Freeman</td><td>Stewart Fulbright</td></tr>
<tr><td>Michael Berry</td><td>Stewart Fulbright</td></tr>
<tr><td>Stewart Fulbright</td><td>Stewart Fulbright</td></tr>
</table>
6 rows returned in 0.03 seconds Download
</td>
<td>This query creates a self-join on the employee table. The result of this shows each employee name and their manager's name.
Note the column names for first name and last name have been concatenated. The same process was also applied for manager. This creates a more appropriate table.</td>
</tr>
</table>

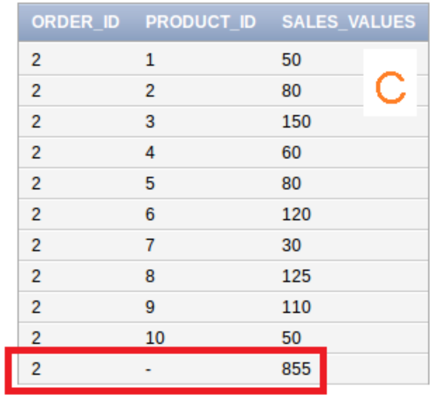
<table>
<tr>
<td><b>Q.4</b></td>
<td><b>SELECT
a.FNAME AS
Employee_FirstName,
a.LNAME AS
Employee_Surname,
b.ORDER_NBR
FROM EMPLOYEE a LEFT
OUTER JOIN CUST_ORDER b
ON a.EMP_ID=b.SALES_EMP_ID;</b></td>
<td>

<table>
<tr><th>EMPLOYEE_FIRSTNAME</th><th>EMPLOYEE_SURNAME</th><th>ORDER_NBR</th></tr>
<tr><td>Jason</td><td>Chase</td><td>1000</td></tr>
<tr><td>Jason</td><td>Chase</td><td>1012</td></tr>
<tr><td>Jason</td><td>Chase</td><td>1024</td></tr>
<tr><td>Jason</td><td>Chase</td><td>1020</td></tr>
<tr><td>Jason</td><td>Chase</td><td>1016</td></tr>
<tr><td>Jason</td><td>Chase</td><td>1008</td></tr>
<tr><td>Jason</td><td>Chase</td><td>1004</td></tr>
<tr><td>James</td><td>Mason</td><td>1001</td></tr>
<tr><td>James</td><td>Mason</td><td>1021</td></tr>
<tr><td>Michael</td><td>Berry</td><td>1023</td></tr>
<tr><td>Michael</td><td>Berry</td><td>1011</td></tr>
<tr><td>Michael</td><td>Berry</td><td>1015</td></tr>
<tr><td>Reno</td><td>Lopez</td><td>-</td></tr>
<tr><td>Stewart</td><td>Fulbright</td><td>-</td></tr>
</table>
28 rows returned in 0.00 seconds Download
</td>
<td>This query uses an outer join to list all the employees' first name and surnames with the customer order's table. This is possible when the employee table was outer joined with the cust_order table using the employeeid with that of the sales_emp_id.

The benefit of the outer join is that we can see employees that have never had a sale.</td>
</tr>
<tr>
<td><b>Q.5</b></td>
<td><b>SELECT SUM(sale_price)
AS "Total sale price of all
orders"
FROM CUST_ORDER;</b></td>
<td>

<table>
<tr><th>Total sale price of all orders</th></tr>
<tr><td>9595.59</td></tr>
</table>
1 rows returned in 0.00 seconds
</td>
<td>This query selects the sum of the sale price column and displays the total price of all orders. This value has been calculated from the CUST_ORDER table.</td>
</tr>
</table>

<table>
<tr>
<td><b>Q.6</b></td>
<td><b>SELECT SALES_EMP_ID,
ROUND(AVG(CAST(SALE_PRICE AS FLOAT)), 2)
AS AVERAGE_SALE_PRICE
FROM CUST_ORDER
GROUP BY SALES_EMP_ID
ORDER BY SALES_EMP_ID;</b></td>
<td>
<table>
<tr><th>SALES_EMP_ID</th><th>AVERAGE_SALE_PRICE</th></tr>
<tr><td>300</td><td>396.8</td></tr>
<tr><td>301</td><td>562.53</td></tr>
<tr><td>302</td><td>146.22</td></tr>
<tr><td>303</td><td>333.83</td></tr>
</table>
4 rows returned in 0.07 seconds Download</td>
<td>This query displays the average sale price of an order by each employee.</td>
</tr>
<tr>
<td><b>Q.7</b></td>
<td><b>SELECT CUST_NBR,
SUM(SALE_PRICE)
AS
TOTAL_SALE_PRICE_OF_ALL_ORDERS
FROM CUST_ORDER
GROUP BY CUST_NBR
ORDER BY CUST_NBR;</b></td>
<td>
<table>
<tr><th>CUST_NBR</th><th>TOTAL_SALE_PRICE_OF_ALL_ORDERS</th></tr>
<tr><td>100</td><td>1291.98</td></tr>
<tr><td>101</td><td>1278.1</td></tr>
<tr><td>102</td><td>1440.75</td></tr>
<tr><td>103</td><td>1700.49</td></tr>
<tr><td>104</td><td>1090.35</td></tr>
<tr><td>105</td><td>1316.09</td></tr>
<tr><td>106</td><td>58.49</td></tr>
<tr><td>107</td><td>601.25</td></tr>
<tr><td>108</td><td>676.49</td></tr>
<tr><td>109</td><td>141.6</td></tr>
</table>
10 rows returned in 0.04 seconds Download</td>
<td>This query displays the average total sales price of all orders from each customer. The total sale price of all orders column was renamed from sale price. This is because it is more efficient to have columns representing what the data pertains to. The customer numbers were also put in order.</td>
</tr>
</table>

<table>
<tr>
<td><b>Q.8</b></td>
<td><b>SELECT CUST_NBR,
SUM(SALE_PRICE)
AS
TOTAL_SALE_PRICE_OF_ALL_ORDERS
FROM CUST_ORDER
HAVING SUM(SALE_PRICE) > 1000
GROUP BY CUST_NBR
ORDER BY CUST_NBR;</b></td>
<td>

<table>
<tr><th>CUST_NBR</th><th>TOTAL_SALE_PRICE_OF_ALL_ORDERS</th></tr>
<tr><td>100</td><td>1291.98</td></tr>
<tr><td>101</td><td>1278.1</td></tr>
<tr><td>102</td><td>1440.75</td></tr>
<tr><td>103</td><td>1700.49</td></tr>
<tr><td>104</td><td>1090.35</td></tr>
<tr><td>105</td><td>1316.09</td></tr>
</table>
6 rows returned in 0.00 seconds Download
</td>
<td>This creates a query to display the total sales price of all orders from each customer. It will only show customers who spent more than 1000.</td>
</tr>
<tr>
<td><b>Q.9</b></td>
<td><b>SELECT ORDER_ID,
PRODUCT_ID,
SUM(UNIT_PRICE) AS "SALES_VALUES"
FROM DEMO_ORDER_ITEMS
GROUP BY CUBE (ORDER_ID, PRODUCT_ID)
ORDER BY ORDER_ID, PRODUCT_ID;</b></td>
<td>


<table>
<tr><th>ORDER_ID</th><th>PRODUCT_ID</th><th>SALES_VALUES</th></tr>
<tr><td>1</td><td>1</td><td>50</td></tr>
<tr><td>1</td><td>2</td><td>80</td></tr>
<tr><td>1</td><td>3</td><td>150</td></tr>
<tr><td>1</td><td>-</td><td>280</td></tr>
<tr><td>2</td><td>1</td><td>50</td></tr>
<tr><td>2</td><td>2</td><td>80</td></tr>
<tr><td>2</td><td>3</td><td>150</td></tr>
<tr><td>2</td><td>4</td><td>60</td></tr>
<tr><td>2</td><td>5</td><td>80</td></tr>
<tr><td>2</td><td>6</td><td>120</td></tr>
<tr><td>2</td><td>7</td><td>30</td></tr>
<tr><td>2</td><td>8</td><td>125</td></tr>
<tr><td>2</td><td>9</td><td>110</td></tr>
<tr><td>2</td><td>10</td><td>50</td></tr>
<tr><td>2</td><td>-</td><td>855</td></tr>
</table>


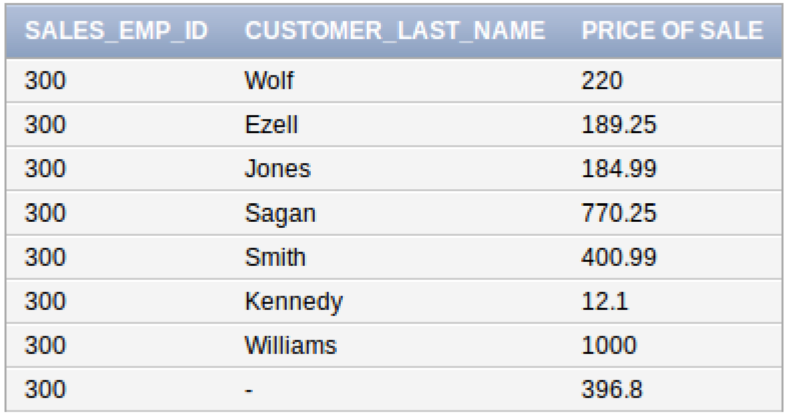
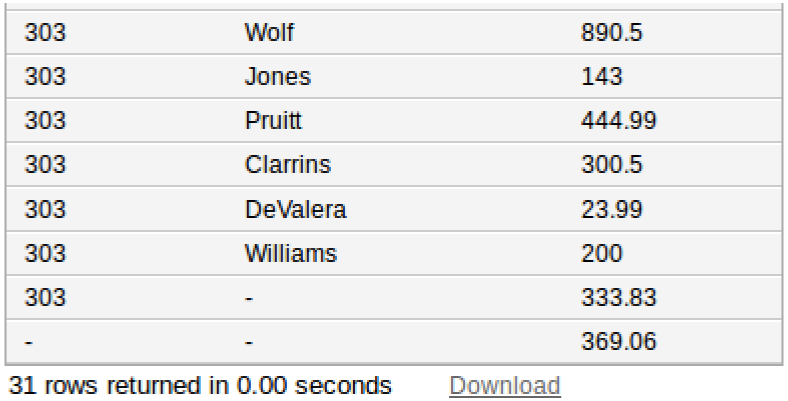
<table>
<tr><td>10</td><td>1</td><td>50</td></tr>
<tr><td>10</td><td>2</td><td>80</td></tr>
<tr><td>10</td><td>3</td><td>150</td></tr>
<tr><td>10</td><td>-</td><td>280</td></tr>
<tr><td>-</td><td>1</td><td>250</td></tr>
<tr><td>-</td><td>2</td><td>480</td></tr>
<tr><td>-</td><td>3</td><td>900</td></tr>
<tr><td>-</td><td>4</td><td>300</td></tr>
<tr><td>-</td><td>5</td><td>400</td></tr>
<tr><td>-</td><td>6</td><td>600</td></tr>
<tr><td>-</td><td>7</td><td>90</td></tr>
<tr><td>-</td><td>8</td><td>750</td></tr>
<tr><td>-</td><td>9</td><td>440</td></tr>
<tr><td>-</td><td>10</td><td>200</td></tr>
<tr><td>-</td><td>-</td><td>4410</td></tr>
</table>
70 rows returned in 0.01 seconds Download


<table>
<tr><th>ORDER_ID</th><th>PRODUCT_ID</th><th>SALES_VALUES</th></tr>
<tr><td>2</td><td>1</td><td>50</td></tr>
<tr><td>2</td><td>2</td><td>80</td></tr>
<tr><td>2</td><td>3</td><td>150</td></tr>
<tr><td>2</td><td>4</td><td>60</td></tr>
<tr><td>2</td><td>5</td><td>80</td></tr>
<tr><td>2</td><td>6</td><td>120</td></tr>
<tr><td>2</td><td>7</td><td>30</td></tr>
<tr><td>2</td><td>8</td><td>125</td></tr>
<tr><td>2</td><td>9</td><td>110</td></tr>
<tr><td>2</td><td>10</td><td>50</td></tr>
<tr><td>2</td><td>-</td><td>855</td></tr>
</table>
</td>
<td>A cube is an expansion of data records for individual events. It can be used to provide more expansive analyse of columns.

It aggregates all combinations of values in the selected columns.

<b><i>Please note all Figures pertain to one table – segmented here to demonstrate what's happening with the query.</i></b>

For example in this query we can see in <b>Figure.A</b> what products were sold in what order i.e 4 orders amounted to 280 dollars in sale. Only 3 products were used in that order. In <b>Figure.B</b> we can see the product that was ordered the most which was product 3 that earned the company 900euro. <b>Figure C</b> showed that Order 2 was the biggest order with multiple sales accumulating to 855euro.</td>
</tr>
</table>

<table>
<tr>
<td><b>Q.10</b></td>
<td><b>SELECT a.SALES_EMP_ID,
b.LNAME as
CUSTOMER_LAST_NAME,
ROUND(AVG(a.SALE_PRICE),2)
AS "PRICE OF SALE"
FROM CUST_ORDER a JOIN
CUSTOMER b
ON a.CUST_NBR = b.CUST_NBR
GROUP BY ROLLUP
(a.SALES_EMP_ID, b.LNAME);</b></td>
<td>

<table>
<tr><th>SALES_EMP_ID</th><th>CUSTOMER_LAST_NAME</th><th>PRICE OF SALE</th></tr>
<tr><td>300</td><td>Wolf</td><td>220</td></tr>
<tr><td>300</td><td>Ezell</td><td>189.25</td></tr>
<tr><td>300</td><td>Jones</td><td>184.99</td></tr>
<tr><td>300</td><td>Sagan</td><td>770.25</td></tr>
<tr><td>300</td><td>Smith</td><td>400.99</td></tr>
<tr><td>300</td><td>Kennedy</td><td>12.1</td></tr>
<tr><td>300</td><td>Williams</td><td>1000</td></tr>
<tr><td>300</td><td>-</td><td>396.8</td></tr>
</table>

<table>
<tr><td>303</td><td>Wolf</td><td>890.5</td></tr>
<tr><td>303</td><td>Jones</td><td>143</td></tr>
<tr><td>303</td><td>Pruitt</td><td>444.99</td></tr>
<tr><td>303</td><td>Clarrins</td><td>300.5</td></tr>
<tr><td>303</td><td>DeValera</td><td>23.99</td></tr>
<tr><td>303</td><td>Williams</td><td>200</td></tr>
<tr><td>303</td><td>-</td><td>333.83</td></tr>
<tr><td>-</td><td>-</td><td>369.06</td></tr>
</table>
31 rows returned in 0.00 seconds Download
</td>
<td><b>Price of Sale shows the average of the total sum per employee ID.</b>

ROLLUP generates a result set that shows aggregates for a hierarchy of values in the selected columns

The rollup query therefore allows multiple groupings to take place at the one time, as seen in this query.

The main difference between Rollup and Cube is that Cube returns the totals for all possible combinations. So in this query it shows the total price of all averages per sales employee with a total at the end.</td>
</tr>
</table>

Appendix 1. Queries to create tables

```
CREATE TABLE CUSTOMER (
 CUST_NBR NUMBER(10) NOT NULL ,
 FNAME NVARCHAR2(20) NULL,
 LNAME NVARCHAR2(20) NULL,
 PRIMARY KEY(CUST_NBR)
 );

 CREATE TABLE EMPLOYEE (
 EMP_ID NUMBER(10) NOT NULL ,
 FNAME NVARCHAR2(20) NULL,
 LNAME NVARCHAR2(20) NULL,
 MANAGER_EMP_ID NUMBER(10) NULL,
 PRIMARY KEY(EMP_ID),
 FOREIGN KEY(MANAGER_EMP_ID)
 REFERENCES EMPLOYEE(EMP_ID)
 );

 CREATE TABLE CUST_ORDER (
 ORDER_NBR NUMBER(10) NOT NULL ,
 CUST_NBR NUMBER(10) NOT NULL,
 SALES_EMP_ID NUMBER(10) NOT NULL,
 SALE_PRICE NUMBER(10, 2) NULL,
 PRIMARY KEY(ORDER_NBR),
 FOREIGN KEY(SALES_EMP_ID)
 REFERENCES EMPLOYEE(EMP_ID),
 FOREIGN KEY(CUST_NBR)
 REFERENCES CUSTOMER(CUST_NBR)
 );

 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 100, 'John', 'Smith');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 101, 'David','Williams');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 102, 'Angelina','Wolf');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 103, 'Natalie','Clarrins');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 104, 'Carl', 'Sagan');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 105, 'Renata', 'Jones');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 106, 'Julie','DeValera');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 107, 'Bruce', 'Ezell');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 108, 'Mark', 'Pruitt');
 INSERT INTO CUSTOMER (CUST_NBR, FNAME, LNAME) VALUES ( 109, 'Nigel','Kennedy');

 INSERT INTO EMPLOYEE (EMP_ID, FNAME, LNAME, MANAGER_EMP_ID) VALUES( 304,'Reno', 'Lopez', 304);
 INSERT INTO EMPLOYEE (EMP_ID, FNAME, LNAME, MANAGER_EMP_ID) VALUES( 305,'Stewart', 'Fulbright',
 305);

 INSERT INTO EMPLOYEE (EMP_ID, FNAME, LNAME, MANAGER_EMP_ID) VALUES( 300,'Jason', 'Chase', 304);
 INSERT INTO EMPLOYEE (EMP_ID, FNAME, LNAME, MANAGER_EMP_ID) VALUES( 301,'James', 'Mason', 304);
INSERT INTO EMPLOYEE (EMP_ID, FNAME, LNAME, MANAGER_EMP_ID) VALUES( 302,'Mila', 'Freeman', 305);
INSERT INTO EMPLOYEE (EMP_ID, FNAME, LNAME, MANAGER_EMP_ID) VALUES( 303,'Michael', 'Berry', 305);

INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1000, 100, 300,
400.99);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1001, 100, 301,
800.00);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1002, 100, 302,
90.99);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1003, 101, 303,
200.00);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1004, 101, 300,
1000.00);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1005, 101, 301,
78.10);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1006, 102, 302,
330.25);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1007, 102, 303,
890.50);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1008, 102, 300,
220.00);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1009, 103, 301,
1300.00);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1010, 103, 302,
99.99);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1011, 103, 303,
300.50);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1012, 104, 300,
770.25);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1013, 104, 301,
230.00);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1014, 104, 302,
90.10);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1015, 105, 303,
143.00);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1016, 105, 300,
184.99);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1017, 105, 301,
988.10);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1018, 106, 302,
34.50);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1019, 106, 303,
23.99);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1020, 107, 300,
189.25);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1021, 107, 301,
412.00);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1022, 108, 302,
 231.50);
 INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1023, 108, 303,
 444.99);
 INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1024, 109, 300,
 12.10);
INSERT INTO CUST_ORDER (ORDER_NBR, CUST_NBR, SALES_EMP_ID, SALE_PRICE) VALUES(1025, 109, 301,
129.50);
```