

Assignment 1, Section A Worth 20% Out of 100

Due by Wednesday, February 24th by
9pm

Group 6

Name : Leonaire George

Student ID = 012067153

Name : Long Nguyen

Student ID: 155176183

Name : Rujal Tuladhar

Student ID: 154594188

Submission

Your submission will be a Document file containing all 10 Select statements Plus their Outputs. Also you will attach a single text-based SQL file (.sql) with appropriate header and commenting. Please ensure your file runs when the entire file is executed in SQL Developer or SQL Plus.

Save these files as **Asgn1_Gr6.docx** and **Asgn1_Gr6.sql**, here is X your Group Number.
Only one e-mail submission per group please.

Submission Checklist

Use the following checklist, to make sure you have completed the assignment successfully.

Tasks to be completed	Yes	No
You have read the assignment group submission and completion policies and all instructions provided in the assignment document and have not missed a word.	YES	
Student information and the assignment information have been added to the header of the submission. (Same as the template provided in the assignment documents)	YES	
All questions are answered in a text file and are saved as a .sql file.		
Comments are included. (questions definition or any additional explanation)	YES	
All SQL statements are executed successfully without errors. (Use "Run Script" or @scriptname to execute all statements together.)	YES	
The result of your SQL statements contains the given sample row and the header in the assignment document.	YES	

Group Work

This assignment is to be completed in groups of 3 or 4. Please only one submission per group. The comment header **MUST** have all students' name and student number.

It is suggested that you **ALL do it individually** and then meet to compare answers. Those not doing the work may be barred from your group resulting in a zero and incomplete on the assignment.

Tasks

For each question, the Header and the First Sample Row (or Rows) must match the sample row or rows given in that question. If you are using SQL Plus, then you may format the width of certain items (columns) output with SUBSTR(item,1,length)

1. Display the employee number, full employee name, job title, and hire date of all employees hired in September, excluding employees with Administrative jobs (their title starts with **Admin**), with the most recently hired employees displayed first, followed by their last name ascending.

```
SELECT employee_id AS "Emp Id", (last_name || ', ' || first_name)
AS "Full Name", job_title AS "Job",
to_char(hire_date, 'fmMonth ddth "of "YYYY') AS "Start Date" FROM employees
WHERE UPPER(job_title) NOT LIKE UPPER('Admin%')
AND extract(month from hire_date)=extract(month from DATE '2020-09-01')
ORDER BY hire_date DESC, last_name ASC;
```

OUTPUT:

	Emp Id	Full Name	Job	Start Date
1	12	James, Elliott	Accountant	September 30th of 2016
2	27	Long, Kai	Stock Clerk	September 28th of 2016
3	11	Ramirez, Tyler	Accountant	September 28th of 2016

Emp Id Full Name Job Start Date

12 James, Elliott Accountant [September 30th of 2016]

- The company wants to see the total sale amount per sales person (salesman) for all orders. Assume that some online orders do not have any sales representative. For online orders (orders with no salesman ID), consider the salesman ID as 0. Display the salesman ID and the total sale amount for the employee for each employee.

Sort the result according to employee number.

```
SELECT COALESCE(employee_id, 0) AS "Employee Id",  
TO_CHAR(SUM(unit_price*quantity),'$999,999,999.99') AS "Total Sale"  
From (employees RIGHT JOIN orders ON (employees.employee_id=orders.salesman_id))  
LEFT JOIN order_items USING (order_id)  
GROUP BY employee_id  
ORDER BY "Employee Id" ASC;
```

OUTPUT:

	Employee Id	Total Sale
1	0	\$18,245,463.50
2	54	\$1,884,295.40
3	55	\$3,525,462.19
4	56	\$2,754,951.05
5	57	\$3,522,704.53
6	59	\$3,900,172.99
7	60	\$3,233,737.31
8	61	\$3,252,131.23
9	62	\$8,081,332.30
10	64	\$4,341,842.14

Employee Id Total Sale

0 \$18,245,463.50
54 \$1,884,295.40

3. Display customer Id, customer name and total number of orders for customers with their Id less than 200 and with name starting on F or J, but only if their total number of orders is less than 3. Include the customers with no orders in your report as well. Sort the result by the value of total orders ascending, followed by name ascending.

```
SELECT customer_id AS "CustId", name AS "Name", COUNT(order_id) AS "Total Orders"
FROM customers LEFT JOIN orders USING(customer_id)
WHERE (UPPER(name) LIKE UPPER('F%') OR UPPER(name) LIKE UPPER('J%'))
GROUP BY customer_id, name HAVING (customer_id < 200
AND COUNT(order_id) < 3)
ORDER BY "Total Orders" ASC, customers.name ASC;
```

OUTPUT:

	⚡ CustId	⚡ Name	⚡ Total Orders
1	103	Fannie Mae	0
2	150	FedEx	0
3	77	FirstEnergy	0
4	40	Fluor	0
5	96	Ford Motor	0
6	135	Freddie Mac	0
7	110	J.P. Morgan Chase	0
8	131	Johnson & Johnson	0
9	165	Johnson Controls	0
10	43	Facebook	1
11	61	Freeport-McMoRan	1

CustId	Name	TotalOrders
103	Fannie Mae	0
150	FedEx	0

4. Display customer Id, customer name, and the order id and the order date of all orders for customer whose ID is 44.
 - a. Show also the total number of items ordered and the total amount of each customer's order.
 - b. Exclude Orders with the Total Amount exceeding 1 million dollars
 - c. Sort the result from the highest to lowest total order amount.

```

SELECT customer_id AS "Cust#", name AS "Name", order_id AS "Order Id",
TO_CHAR(order_date, 'DD-MON-YY') AS "Order Date",
SUM(quantity) AS "Total Items",
TO_CHAR(SUM(unit_price*quantity), '$999,999,999.99') AS "Total Amount"
FROM (orders LEFT JOIN order_items USING (order_id))
JOIN customers USING(customer_id)
WHERE order_id IN(SELECT order_id FROM orders WHERE customer_id = 44)
GROUP BY order_id, order_date, name, customer_id
HAVING(SUM(unit_price*quantity) < 1000000)
ORDER BY "Total Items";

```

OUTPUT:

	Cust#	Name	Order Id	Order Dat	Total Items	Total Amount
1	44	Jabil Circuit	69	17-MAR-17	581	\$755,093.92
2	44	Jabil Circuit	82	03-DEC-16	687	\$398,636.25
3	44	Jabil Circuit	29	14-AUG-17	831	\$508,588.59
4	44	Jabil Circuit	10	24-JAN-17	883	\$620,962.99

Cust#	Name	Order Id	Order Dat	Total Items	Total Amount
44	Jabil Circuit	69	17-MAR-17	581	\$755,093.92

5. Display customer Id, name, total number of orders, the total number of items ordered, and the total order amount for customers who have more than 30 orders. Sort the result based on the total number of orders.

SOLUTION:

```
select customer_id as "Cust#", name as "Name", count(order_id) as "#of Orders",  
sum(quantity) as "Total Items",  
to_char(sum(quantity*unit_price),'$9,999,999.99') as "Total Amount"  
from orders  
join customers using (customer_id)  
join order_items using (order_id)  
group by customer_id,name  
having count(order_id) > 30  
order by 3;
```

OUTPUT:

	⚡ Cust#	⚡ Name	⚡ #of Orders	⚡ Total Items	⚡ Total Amount	
1	47	General Mills	33	3116	\$3,725,138.14	
2	8	International Paper	35	3281	\$2,642,238.04	
3	49	NextEra Energy	37	3351	\$2,452,508.95	
4	9	Emerson Electric	37	3301	\$2,893,564.97	
5	44	Jabil Circuit	45	3772	\$3,334,221.72	

Cust# Name # of Orders Total Items Total Amount

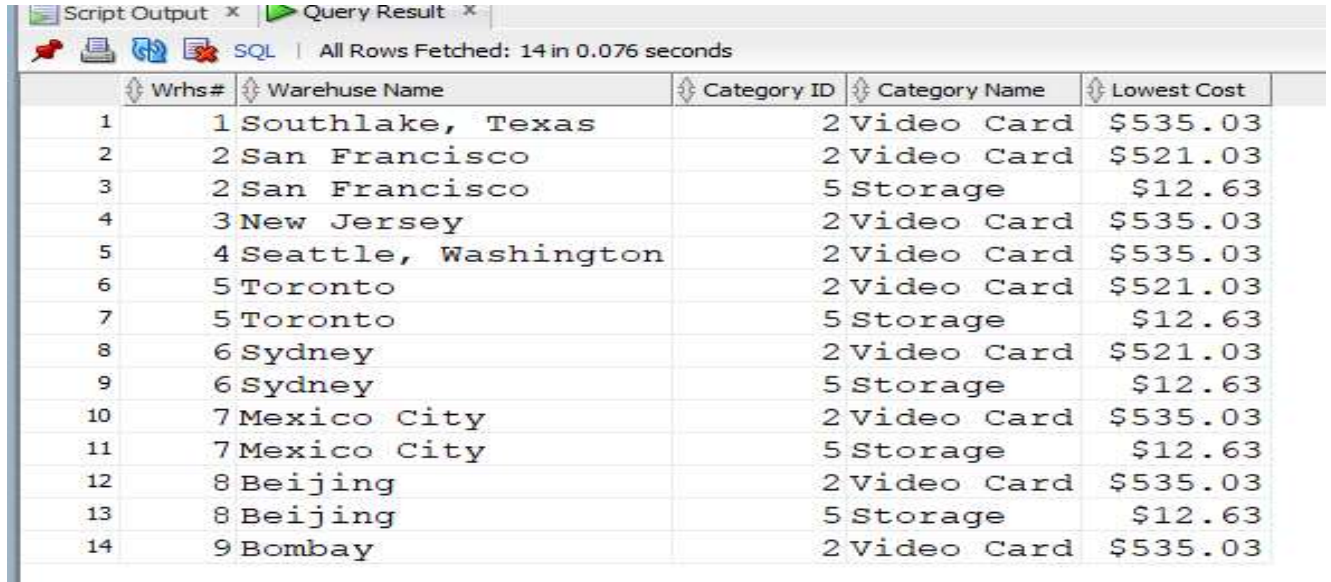
47 General Mills 33 3116 \$3,725,138.14

6. Display Warehouse Id, warehouse name, product category Id, product category name, and the lowest product standard cost for this combination.
- In your result, include the rows that the lowest standard cost is less then \$200.
 - Also, include the rows that the lowest cost is more than \$500.
 - Sort the output according to Warehouse Id, warehouse name and then product category Id, and product category name.

```
SELECT
w.warehouse_id AS "Wrhs#",
warehouse_name AS "Warehuse Name",
p.category_id AS "Category ID",
category_name AS "Category Name",
MIN (TO_CHAR(standard_cost, '$999.99')) AS "Lowest Cost"
FROM
inventories i
JOIN
warehouses w
ON i.warehouse_id = w.warehouse_id
JOIN products p
ON i.product_id = p.product_id
JOIN product_categories pc
ON p.category_id = pc.category_id
GROUP BY w.warehouse_id,
warehouse_name,
p.category_id,
category_name
HAVING
MIN(standard_cost) < 200
OR MIN(standard_cost) > 500
ORDER BY
w.warehouse_id,
```


warehouse_name,
p.category_id,
category_name;

OUTPUT:



	Wrhs#	Warehouse Name	Category ID	Category Name	Lowest Cost
1	1	Southlake, Texas	2	Video Card	\$535.03
2	2	San Francisco	2	Video Card	\$521.03
3	2	San Francisco	5	Storage	\$12.63
4	3	New Jersey	2	Video Card	\$535.03
5	4	Seattle, Washington	2	Video Card	\$535.03
6	5	Toronto	2	Video Card	\$521.03
7	5	Toronto	5	Storage	\$12.63
8	6	Sydney	2	Video Card	\$521.03
9	6	Sydney	5	Storage	\$12.63
10	7	Mexico City	2	Video Card	\$535.03
11	7	Mexico City	5	Storage	\$12.63
12	8	Beijing	2	Video Card	\$535.03
13	8	Beijing	5	Storage	\$12.63
14	9	Bombay	2	Video Card	\$535.03

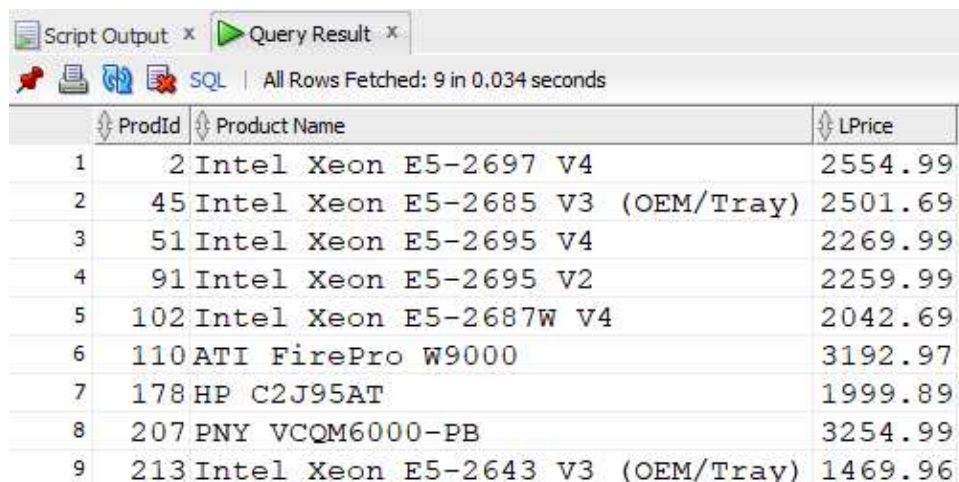
Wrhs#	Warehouse Name	Category ID	Category Name	Lowest Cost
1	Southlake, Texas	2	Video Card	\$535.03
2	San Francisco	2	Video Card	\$521.03
2	San Francisco	5	Storage	\$12

7. Display product Id, name, and list Price for products that were purchased in orders handled by salesman Marshall and with list price greater than all average list prices per each category. Sort the output by Id ascending.

Solution:

```
select product_id AS "ProdId", product_name AS "Product Name", list_price AS "LPrice"
from order_items
join products using (product_id)
join orders using (order_id)
where salesman_id =( select employee_id
                     from employees
                     where upper(last_name) like '%MARSHALL%')
and list_price >all (select avg(list_price)
                   from products
                   group by category_id)
order by 1 asc ;
```

OUTPUT:



The screenshot shows a SQL query result window with two tabs: 'Script Output' and 'Query Result'. The 'Query Result' tab is active, displaying a table with 9 rows and 3 columns: 'ProdId', 'Product Name', and 'LPrice'. The table is sorted by 'ProdId' in ascending order. The data is as follows:

	ProdId	Product Name	LPrice
1	2	Intel Xeon E5-2697 V4	2554.99
2	45	Intel Xeon E5-2685 V3 (OEM/Tray)	2501.69
3	51	Intel Xeon E5-2695 V4	2269.99
4	91	Intel Xeon E5-2695 V2	2259.99
5	102	Intel Xeon E5-2687W V4	2042.69
6	110	ATI FirePro W9000	3192.97
7	178	HP C2J95AT	1999.89
8	207	PNY VCQM6000-PB	3254.99
9	213	Intel Xeon E5-2643 V3 (OEM/Tray)	1469.96

ProdId Product Name LPrice

2 Intel Xeon E5-2697 V 2554.99
 45 Intel Xeon E5-2685 V 2501.69

8. Display customer Id, name, and total number of orders, for orders handled by salesman Marshall, but only if customer name begins on General or ends on Electric. Exclude customers who placed a single order, but include customers without orders as well. Sort the result based on the total number of orders descending and then by name ascending. Do not use LIKE operator and do not join 3 tables.

```
select a.customer_id AS "CUSTOMER_ID" , a.name AS "Customer Name" ,count(b.order_id)
AS "# of Orders"
```

```
from customers a left
```

```
join orders b on a.customer_id = b.customer_id
```

```
and b.salesman_id in (select b.salesman_id
```

```
from employees
```

```
where last_name = 'Marshall')
```

```
where (substr(a.name,1,7) = 'General'
```

```
OR substr(a.name,-8,8) = 'Electric')
```

```
group by a.customer_id, a.name
```

```
having count(b.order_id)<>1
```

```
order by count(b.order_id) desc, a.name;
```

OUTPUT:

	CUSTOMER_ID	Customer Name	# of Orders
1	47	General Mills	3
2	9	Emerson Electric	2
3	185	General Dynamics	0
4	98	General Electric	0
5	95	General Motors	0

```
CUSTOMER_ID  Customer Name      # of Orders
-----
47 General Mills      3
```

9	Emerson Electric	2
185	General Dynamics	0

9. Display product Id, name, and list Price for products that their list price is more than any highest product standard cost per warehouse outside Americas regions.

(You need to find the highest standard cost for each warehouse that is located outside the Americas regions. Then you need to return all products that their list price is higher than any highest standard cost of those warehouses.)

Sort the result according to list price.

```

SELECT
product_id AS "Product iD",
product_name AS "Product Name",
to_char(list_price, '$9,999.99') AS "List Price"
FROM
products
WHERE
list_price > ANY (
SELECT
MAX(standard_cost)
FROM
locations l
JOIN
countries c
ON l.country_id = c.country_id
JOIN
regions r
ON r.region_id = c.region_id
JOIN

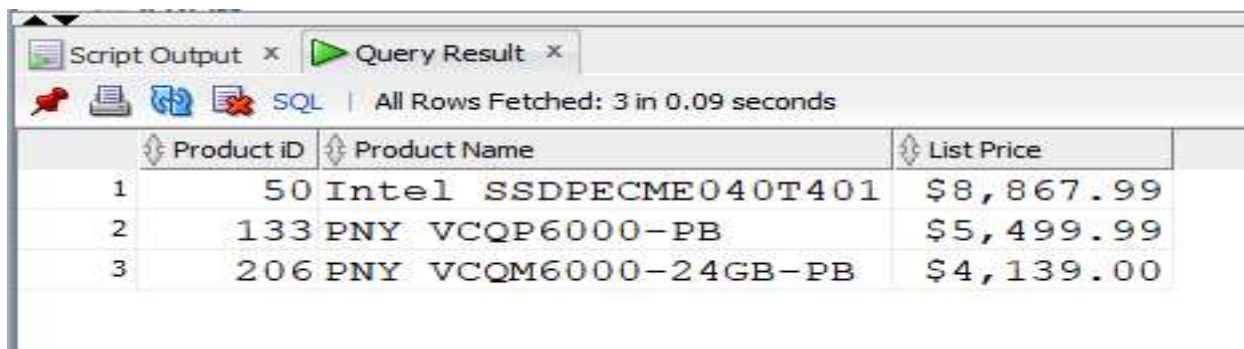
```

```

warehouses w
ON w.location_id = l.location_id
JOIN
inventories i
ON i.warehouse_id = w.warehouse_id
JOIN
products p
ON p.product_id = i.product_id
WHERE
region_name NOT LIKE 'Americas'
GROUP BY
w.warehouse_id)
ORDER BY
list_price DESC;

```

OUTPUT:



The screenshot shows a SQL query result window with a toolbar and a table of results. The toolbar includes icons for script output, query result, and a status bar indicating 'All Rows Fetched: 3 in 0.09 seconds'. The table has three columns: Product ID, Product Name, and List Price. The data is as follows:

	Product ID	Product Name	List Price
1	50	Intel SSDPECME040T401	\$8,867.99
2	133	PNY VCQP6000-PB	\$5,499.99
3	206	PNY VCQM6000-24GB-PB	\$4,139.00

```

Product ID Product Name          List Price
-----
          50 Intel SSDPECME040T40  $8,867.99

```

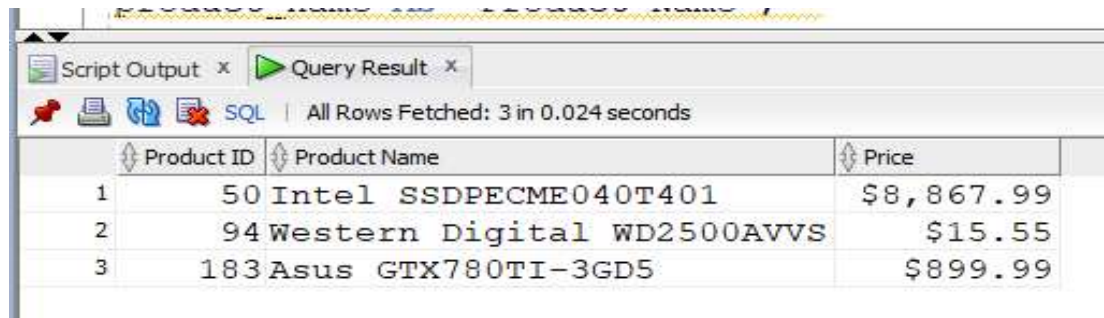
10. Display product Id, name, and list Price for the most expensive product, then the cheapest product and also for the product with the price closest to the average product price (rounded to the nearer ten). For the third row exclude products with name that starts on Intel. Here is the exact output.

```
SELECT product_id AS "Product ID",
product_name AS "Product Name",
TO_CHAR(list_price, '$9,999.99') AS "Price"
FROM products
WHERE product_id = 50
UNION
SELECT product_id AS "Product ID",
product_name AS "Product Name",
TO_CHAR(list_price, '$9,999.99') AS "Price"
FROM products
WHERE list_price = (
    SELECT MIN(list_price)
    FROM products)
GROUP BY product_id, product_name, list_price
UNION
SELECT product_id AS "Product ID",
product_name AS "Product Name",
TO_CHAR(list_price, '$9,999.99') AS "Price"
FROM products
WHERE ROUND(list_price, -1) = (
    SELECT ROUND(AVG(list_price), -1)
    FROM products)
```

AND product_name NOT LIKE 'Intel%'

GROUP BY product_id, product_name, list_price;

OUTPUT:



The screenshot shows a SQL query result window with two tabs: 'Script Output' and 'Query Result'. The 'Query Result' tab is active, displaying a table with three columns: 'Product ID', 'Product Name', and 'Price'. The table contains three rows of data. The status bar at the top indicates 'All Rows Fetched: 3 in 0.024 seconds'.

	Product ID	Product Name	Price
1	50	Intel SSDPECME040T401	\$8,867.99
2	94	Western Digital WD2500AVVS	\$15.55
3	183	Asus GTX780TI-3GD5	\$899.99

Product ID	Product Name	Price
50	Intel SSDPECME040T40	\$8,867.99
94	Western Digital WD25	\$15.55
183	Asus GTX780TI-3GD5	\$899.99