

Module 6.2 Database Project Assignment (Revised)

Task: For questions 1-6, building on the tables, constraints, and data developed through Module 6.2, and using the updated EERD provided at the end of this assignment, complete each of the following problems listed below. For questions 7-10, use the materials provided in the assignment

Background: Upon reviewing the requirements for this assignment after completing assignment 6.1, it was determined that additional records would be required in the task, work log, and client documents tables; with the intent that each query developed for Module 6.2 query would return meaningful results. No functional changes were made to the database design in this iteration.

To prepare the database for this module, all previous tables were dropped. The script for this schema was loaded into Oracle 12c and is titled **2023_04_07_Risk_Insights_Build.sql**.

With the schema in place, a second script was developed to populate the tables using filename **2023_04_21_Risk_Insights_Data_Load_62.sql**. Data populated within the database continues to be drawn from entities from the cartoon, 'The Flintstones.'¹

The SQL code to complete assignment for Module 6.1 is captured in total within the file titled **2023_04_23_M62_Functions.sql**.

All three SQL scripts are submitted as text files. Code snippets and screenshots for each table structure and inserts are submitted below in accordance with assignment specifications.

¹ Characters Database, [List Of Flintstones Characters](#)

1. Write an SQL query that uses a single-row subquery in a WHERE clause. Explain what the query is intended to do.
Explanation: Identify the tasks, by id number, that were completed faster than the average completion time.

<pre>SELECT task_id, (task_compl_date - task_start_date) "Days to Complete Task" FROM task WHERE 1=1 AND (task_compl_date - task_start_date) < (SELECT AVG(task_compl_date-task_start_date) FROM task);</pre>	<pre>19 SELECT sysdate, 'KMALY2' from dual; 20 21 SELECT task_id, 22 (task_compl_date - task_start_date) "Days to Complete Task" 23 FROM task 24 WHERE 1=1 25 AND (task_compl_date - task_start_date) < 26 (SELECT AVG(task_compl_date-task_start_date) FROM task); 27</pre> <p>Query Result x Query Result 1 x</p> <p>SQL All Rows Fetched: 1 in 0.018 seconds</p> <table border="1"> <thead> <tr> <th>SYSDATE</th> <th>'KMALY2'</th> </tr> </thead> <tbody> <tr> <td>1 28-APR-23</td> <td>KMALY2</td> </tr> </tbody> </table>	SYSDATE	'KMALY2'	1 28-APR-23	KMALY2	<pre>19 SELECT sysdate, 'KMALY2' from dual; 20 21 SELECT task_id, 22 (task_compl_date - task_start_date) "Days to Complete Task" 23 FROM task 24 WHERE 1=1 25 AND (task_compl_date - task_start_date) < 26 (SELECT AVG(task_compl_date-task_start_date) FROM task); 27</pre> <p>Query Result x Query Result 1 x</p> <p>SQL All Rows Fetched: 7 in 0.012 seconds</p> <table border="1"> <thead> <tr> <th>TASK_ID</th> <th>Days to Complete Task</th> </tr> </thead> <tbody> <tr><td>1 3</td><td>3</td></tr> <tr><td>2 5</td><td>2</td></tr> <tr><td>3 6</td><td>1</td></tr> <tr><td>4 7</td><td>1</td></tr> <tr><td>5 8</td><td>2</td></tr> <tr><td>6 9</td><td>2</td></tr> <tr><td>7 11</td><td>2</td></tr> </tbody> </table>	TASK_ID	Days to Complete Task	1 3	3	2 5	2	3 6	1	4 7	1	5 8	2	6 9	2	7 11	2
SYSDATE	'KMALY2'																					
1 28-APR-23	KMALY2																					
TASK_ID	Days to Complete Task																					
1 3	3																					
2 5	2																					
3 6	1																					
4 7	1																					
5 8	2																					
6 9	2																					
7 11	2																					

2. Write an SQL query that uses a multiple-column subquery in a FROM clause. Explain what the query is intended to do.
Explanation: List the client tasks, by their id number, that do not have associated documents in the document library

<pre>SELECT t1.task_id FROM task t1 LEFT JOIN (SELECT t.task_id FROM task t JOIN cli_doc cl ON t.task_id = cl.task_id) t2 ON t1.task_id = t2.task_id WHERE t2.task_id IS NULL;</pre>	<pre>32 SELECT sysdate, 'KMALY2' from dual; 33 34 SELECT t1.task_id 35 FROM task t1 36 LEFT JOIN (SELECT t.task_id 37 FROM task t 38 JOIN cli_doc cl 39 ON t.task_id = cl.task_id) t2 40 ON t1.task_id = t2.task_id 41 WHERE t2.task_id IS NULL; 42</pre> <p>Query Result x Query Result 1 x</p> <p>SQL All Rows Fetched: 1 in 0.0</p> <table border="1"> <thead> <tr> <th>SYSDATE</th> <th>'KMALY2'</th> </tr> </thead> <tbody> <tr> <td>1 28-APR-23</td> <td>KMALY2</td> </tr> </tbody> </table>	SYSDATE	'KMALY2'	1 28-APR-23	KMALY2	<pre>32 SELECT sysdate, 'KMALY2' from dual; 33 34 SELECT t1.task_id 35 FROM task t1 36 LEFT JOIN (SELECT t.task_id 37 FROM task t 38 JOIN cli_doc cl 39 ON t.task_id = cl.task_id) t2 40 ON t1.task_id = t2.task_id 41 WHERE t2.task_id IS NULL; 42</pre> <p>Query Result x Query Result 1 x</p> <p>SQL All Rows Fetched: 4 in 0.0</p> <table border="1"> <thead> <tr> <th>TASK_ID</th> </tr> </thead> <tbody> <tr><td>1 6</td></tr> <tr><td>2 5</td></tr> <tr><td>3 10</td></tr> <tr><td>4 11</td></tr> </tbody> </table>	TASK_ID	1 6	2 5	3 10	4 11
SYSDATE	'KMALY2'										
1 28-APR-23	KMALY2										
TASK_ID											
1 6											
2 5											
3 10											
4 11											

4. Write an SQL query that is based on multiple tables and uses a multiple-row subquery in a WHERE clause. The subquery will include the GROUP BY statement and another multiple-row subquery in a HAVING clause. Explain what the query is intended to do.

Query:

[illegible]

5. Write an SQL query that joins three tables and uses any type of a subquery. Explain what the query is intended to do.
 Explanation: Prepare a report breaking down the work accomplished or in progress for the client, which includes the company name, number of plans, number of tasks completed, and documents prepared. Do not include clients that have no prepared documents.

```
WITH custActRpt AS (SELECT cust_dba_name clientName,
                        count(*) tasks,
                        sum(hours_worked) hours,
                        count(doc_id) docs
                     FROM customer c
                        JOIN riplan p ON c.cust_id = p.cust_id
                        LEFT JOIN task t ON p.plan_id = t.plan_id
                        LEFT JOIN work_log wl ON t.task_id = wl.task_id
                        LEFT JOIN cli_doc d ON t.task_id = d.task_id
                     GROUP BY cust_dba_name)
SELECT clientName, tasks, hours, docs
FROM custActRpt;
```

```

81 SELECT sysdate, 'KMALY2' from dual;
82
83 WITH custActRpt AS (SELECT cust_dba_name clientName,
84                        count(*) tasks,
85                        sum(hours_worked) hours,
86                        count(doc_id) docs
87                     FROM customer c
88                        JOIN riplan p ON c.cust_id = p.cust_id
89                        LEFT JOIN task t ON p.plan_id = t.plan_id
90                        LEFT JOIN work_log wl ON t.task_id = wl.task_id
91                        LEFT JOIN cli_doc d ON t.task_id = d.task_id
92                     GROUP BY cust_dba_name)
93 SELECT clientName, tasks, hours, docs
94 FROM custActRpt;
95

```

Query Result 1 x	
SQL All Rows Fetched: 1 in 0.02 seconds	
SYSDATE	'KMALY2'
1 29-APR-23	KMALY2

```

81 SELECT sysdate, 'KMALY2' from dual;
82
83 WITH custActRpt AS (SELECT cust_dba_name clientName,
84                          count(*) tasks,
85                          sum(hours_worked) hours,
86                          count(doc_id) docs
87                     FROM customer c
88                     JOIN riplan p ON c.cust_id = p.cust_id
89                     LEFT JOIN task t ON p.plan_id = t.plan_id
90                     LEFT JOIN work_log wl ON t.task_id = wl.task_id
91                     LEFT JOIN cli_doc d ON t.task_id = d.task_id
92                     GROUP BY cust_dba_name)
93 SELECT clientName, tasks, hours, docs
94 FROM custActRpt;
95

```

Query Result x Query Result 1 x

SQL | All Rows Fetched: 4 in 0.029 seconds

CLIENTNAME	TASKS	HOURS	DOCS
1 Loyal Order of Water Buffalos	3	13	2
2 Bedrock Newspaper	3	13	2
3 Gravel Hotel	2	10	1
4 Alien Enterprises	4	14	3

6. Write an SQL query that is based on multiple tables and uses the DECODE function. Explain what the query is intended to do.

Explanation: Create a report status of all entities that Risk Insights has engaged since incorporating and the key dates associated with each.

```
SELECT cust_dba_name, DECODE(c.cust_code, 1, 'CONTACT', 2, 'LEAD', 3, 'CLIENT', 'CLOSED') "Status",  
       date_created, contract_start_date, closed_date  
FROM customer c  
LEFT JOIN ri_client ric ON c.cust_code = ric.cust_code AND c.cust_id = ric.cust_id  
LEFT JOIN closed cl ON c.cust_code = cl.cust_code AND c.cust_id = cl.cust_id  
ORDER BY c.cust_code;
```

The screenshot displays two SQL query execution results side-by-side. Both queries are identical, using the DECODE function to map cust_code values to status names.

Left Panel Query Result:

	SYSDATE	'KMALY2'
1	29-APR-23	KMALY2

Right Panel Query Result:

CUST_DBA_NAME	Status	DATE_CREATED	CONTRACT_START_DATE	CLOSED_DATE
1 Boulder's Rules	CONTACT	03-APR-23	(null)	(null)
2 bedrock babysitters	LEAD	01-APR-23	(null)	(null)
3 PG investigators	LEAD	02-APR-23	(null)	(null)
4 Rockland Club	LEAD	01-APR-23	(null)	(null)
5 bedrock volunteer firefighters	LEAD	01-APR-23	(null)	(null)
6 Bedrock Newspaper	CLIENT	01-APR-23	02-APR-23	(null)
7 Alien Enterprises	CLIENT	01-APR-23	02-APR-23	(null)
8 Gravel Hotel	CLIENT	02-APR-23	03-APR-23	(null)
9 Loyal Order of Water Buffalos	CLIENT	01-APR-23	02-APR-23	(null)
10 Hollyrock Film Studios	CLOSED	01-APR-23	(null)	03-APR-23
11 mother-in-laws, inc	CLOSED	01-APR-23	(null)	04-APR-23
12 Bedrock Quarrel and Gravel	CLOSED	01-APR-23	(null)	06-APR-23

7. Create the Faculty table and populate it with data using the script below: Check the result using the query: select * from faculty;

```
CREATE TABLE faculty (  
    f_id NUMBER(6), f_last VARCHAR2(30), f_first VARCHAR2(30),  
    f_mi CHAR(1), loc_id NUMBER(5), f_phone VARCHAR2(10), f_rank VARCHAR2(9),  
    f_super NUMBER(6),  
    CONSTRAINT faculty_f_id_pk PRIMARY KEY(f_id)  
);  
  
INSERT INTO faculty VALUES (1, 'Marx', 'Teresa', 'J', 9, '4075921695', 'Associate', 4);  
INSERT INTO faculty VALUES (2, 'Zhulin', 'Mark', 'M', 10, '4073875682', 'Full', NULL);  
INSERT INTO faculty VALUES (3, 'Langley', 'Colin', 'A', 12, '4075928719', 'Assistant', 4);  
INSERT INTO faculty VALUES (4, 'Brown', 'Jonnel', 'D', 11, '4078101155', 'Full', NULL);
```

SELECT * FROM faculty;

The image displays three sequential screenshots of the SQL Developer interface, showing the execution of a script to create and populate a table named 'faculty'.

Left Screenshot: The script is executed, and the output shows the table creation and data insertion. The table 'FACULTY' is created, and four rows are inserted. The output is as follows:

```
Table FACULTY created.  
  
1 row inserted.  
  
1 row inserted.  
  
1 row inserted.  
  
1 row inserted.  
  
>>>Query Run In:Query Result 1  
Commit complete.
```

Middle Screenshot: The script is executed, and the output shows the table creation and data insertion. The table 'FACULTY' is created, and four rows are inserted. The output is as follows:

```
Table FACULTY created.  
  
1 row inserted.  
  
1 row inserted.  
  
1 row inserted.  
  
1 row inserted.  
  
>>>Query Run In:Query Result 1  
Commit complete.
```

Right Screenshot: The script is executed, and the output shows the table creation and data insertion. The table 'FACULTY' is created, and four rows are inserted. The output is as follows:

```
Table FACULTY created.  
  
1 row inserted.  
  
1 row inserted.  
  
1 row inserted.  
  
1 row inserted.  
  
>>>Query Run In:Query Result 1  
Commit complete.
```

8. Create the Bonus table that consists of two columns: f_id (PK) and bonus. For the f_id column, use the same description as in the Faculty table. For the bonus column, use the NUMBER data type and the DEFAULT constraint to set the values for the bonus column to 1000 (bonus amount). Next, use a subquery to copy ids of mentors given in the Faculty table into the Bonus table. Check the result using the select * from bonus; command.

```
CREATE TABLE bonus (b_id NUMBER(6), b_bonus NUMBER(7) DEFAULT ON NULL 1000,  
CONSTRAINT bonus_b_id_pk PRIMARY KEY (b_id), CONSTRAINT bonus_b_id_fk FOREIGN KEY (b_id)  
REFERENCES faculty(f_id));
```

```
DESC bonus;
```

```
INSERT INTO bonus
```

```
VALUES ((SELECT DISTINCT f_super FROM faculty WHERE f_super IS NOT NULL), NULL);
```

```
SELECT * FROM bonus;
```

The first screenshot shows the SQL script execution in SQL Developer. The script includes:
1. A SELECT statement to check the current date and user: `SELECT sysdate, 'KMALY2' from dual;`
2. The CREATE TABLE statement for the BONUS table with columns b_id (NUMBER(6)) and b_bonus (NUMBER(7) DEFAULT ON NULL 1000), and a primary key constraint on b_id.
3. A DESC command to view the table structure.
4. An INSERT INTO statement using a subquery to insert distinct f_super values from the faculty table where f_super is not null.
5. A SELECT * FROM bonus; command to verify the data.
The status bar indicates "Task completed in 4.389 seconds".
Below the script, the "Table BONUS created." message is shown, followed by a table structure summary:
Name Null? Type
B_ID NOT NULL NUMBER(6)
B_BONUS NOT NULL NUMBER(7)
It also states "1 row inserted." and shows the command prompt ">>Query Run In:Query Result 1".

The second screenshot shows the same SQL script, but the execution time is faster: "All Rows Fetched: 1 in 0.027 seconds". The "Query Result" window displays the output of the first query:
SYSDATE KMALY2
1 30-APR-23 KMALY2

The third screenshot shows the same SQL script, with the execution time being "All Rows Fetched: 1 in 0.019 seconds". The "Query Result" window displays the output of the SELECT * FROM bonus; query:
B_ID B_BONUS
1 4 1000

9. Add two new records to the Faculty table using the command below. These records represent new faculty who came to the university this year. Using the command `SELECT * from faculty`, check the result.

```
INSERT INTO faculty VALUES (5, 'Sealy', 'James', 'L', 13, '4079817153', 'Associate', 1);
INSERT INTO faculty VALUES (6, 'Smith', 'John', 'D', 10, '4238102345', 'Full', NULL);
SELECT * FROM faculty;
```

```
162 SELECT sysdate, 'KMALY2' from dual;
163
164 INSERT INTO faculty VALUES (5, 'Sealy', 'James', 'L', 13, '4079817153', 'Associate', 1);
165 INSERT INTO faculty VALUES (6, 'Smith', 'John', 'D', 10, '4238102345', 'Full', NULL);
166
167 SELECT * FROM faculty;
168
```

Query Result x | Script Output x | Query Result 1 x

Task completed in 0.572 seconds

1 row inserted.

1 row inserted.

>>Query Run In:Query Result 1

1 row merged.

```
162 SELECT sysdate, 'KMALY2' from dual;
163
164 INSERT INTO faculty VALUES (5, 'Sealy', 'James', 'L', 13, '4079817153', 'Associate', 1);
165 INSERT INTO faculty VALUES (6, 'Smith', 'John', 'D', 10, '4238102345', 'Full', NULL);
166
167 SELECT * FROM faculty;
168
```

Query Result x | Script Output x | Query Result 1 x

All Rows Fetched: 1 in 0.04 seconds

SYSDATE	'KMALY2'
1 30-APR-23	KMALY2

```
162 SELECT sysdate, 'KMALY2' from dual;
163
164 INSERT INTO faculty VALUES (5, 'Sealy', 'James', 'L', 13, '4079817153', 'Associate', 1);
165 INSERT INTO faculty VALUES (6, 'Smith', 'John', 'D', 10, '4238102345', 'Full', NULL);
166
167 SELECT * FROM faculty;
168
```

Query Result x | Script Output x | Query Result 1 x

All Rows Fetched: 6 in 0.03 seconds

F_ID	F_LAST	F_FIRST	F_MI	LOC_ID	F_PHONE	F_RANK	F_SUPER
1	1Marx	Teresa	J		9 4075921695	Associate	4
2	2 ZhuLin	Mark	M		10 4073875682	Full	(null)
3	3 Langley	Colin	A		12 4075920719	Assistant	4
4	4 Brown	Jonnel	D		11 4078101155	Full	(null)
5	5 Sealy	James	L		13 4079817153	Associate	1
6	6 Smith	John	D		10 4238102345	Full	(null)

10. Assume that the same Bonus table is used next year to assign and update bonuses. Use the MERGE statement to modify the Bonus table as follows: - if a mentor already exists in the Bonus table, increase the bonus by 1% - If there is a new mentor in the Faculty table, add him/her to the BONUS table Check the result using the select * from bonus; command.

MERGE INTO bonus

USING (SELECT DISTINCT f_super FROM faculty WHERE f_super IS NOT NULL) temp

ON (bonus.b_id = temp.f_super)

WHEN MATCHED THEN

UPDATE SET b_bonus = b_bonus * 1.01

WHEN NOT MATCHED THEN

INSERT (b_id) VALUES (temp.f_super);

<pre> 175 SELECT sysdate, 'KMALY2' from dual; 176 177 MERGE INTO bonus 178 USING (SELECT DISTINCT f_super 179 FROM faculty 180 WHERE f_super IS NOT NULL) temp 181 ON (bonus.b_id = temp.f_super) 182 WHEN MATCHED THEN 183 UPDATE SET b_bonus = b_bonus * 1.01 184 WHEN NOT MATCHED THEN 185 INSERT (b_id) 186 VALUES (temp.f_super); 187 188 SELECT * from BONUS; </pre> <p>Script Output x Query Result x Query Result</p> <p>Task completed in 0.401 seconds</p> <p>2 rows merged.</p> <p>>>Query Run In:Query Result</p>	<pre> 175 SELECT sysdate, 'KMALY2' from dual; 176 177 MERGE INTO bonus 178 USING (SELECT DISTINCT f_super 179 FROM faculty 180 WHERE f_super IS NOT NULL) temp 181 ON (bonus.b_id = temp.f_super) 182 WHEN MATCHED THEN 183 UPDATE SET b_bonus = b_bonus * 1.01 184 WHEN NOT MATCHED THEN 185 INSERT (b_id) 186 VALUES (temp.f_super); 187 188 SELECT * from BONUS; </pre> <p>Script Output x Query Result x Query Result</p> <p>SQL All Rows Fetched: 1 in 0.012 seco</p> <table border="1"> <thead> <tr> <th></th> <th>SYSDATE</th> <th>'KMALY2'</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>30-APR-23</td> <td>KMALY2</td> </tr> </tbody> </table>		SYSDATE	'KMALY2'	1	30-APR-23	KMALY2	<pre> 175 SELECT sysdate, 'KMALY2' from dual; 176 177 MERGE INTO bonus 178 USING (SELECT DISTINCT f_super 179 FROM faculty 180 WHERE f_super IS NOT NULL) temp 181 ON (bonus.b_id = temp.f_super) 182 WHEN MATCHED THEN 183 UPDATE SET b_bonus = b_bonus * 1.01 184 WHEN NOT MATCHED THEN 185 INSERT (b_id) 186 VALUES (temp.f_super); 187 188 SELECT * from BONUS; </pre> <p>Script Output x Query Result x Query Result</p> <p>SQL All Rows Fetched: 2 in 0.015 seco</p> <table border="1"> <thead> <tr> <th></th> <th>B_ID</th> <th>B_BONUS</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>4</td> <td>1020</td> </tr> <tr> <td>2</td> <td>1</td> <td>1000</td> </tr> </tbody> </table>		B_ID	B_BONUS	1	4	1020	2	1	1000
	SYSDATE	'KMALY2'															
1	30-APR-23	KMALY2															
	B_ID	B_BONUS															
1	4	1020															
2	1	1000															

AIT-524 / DL1: Revised EERD (Module 6.2)
 Keith Maly (kmaly2@gmu.edu)
 April 21, 2023

CHECK Constraints: ending dates within tables must be after starting dates; customer codes may only be 00, 01, 02, or 03.
 UNIQUE The only anticipated 'unique' data element outside of primary keys is the clients EIN
 Per Oracle Docs, all attributes in sub/supertype design must be NOT NULL

