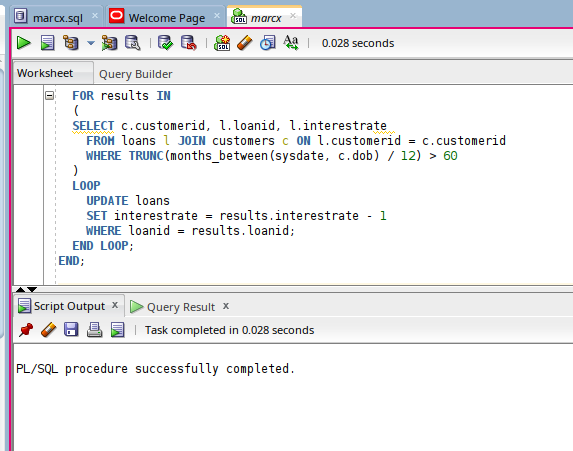
| *Exercise 1: Control Structures* |
| --- |

Scenario 1 :

Since the bank wants to apply a discount of 1% to customers over 60 years of age, we can use an anonymous block to run a control structured logic for once.

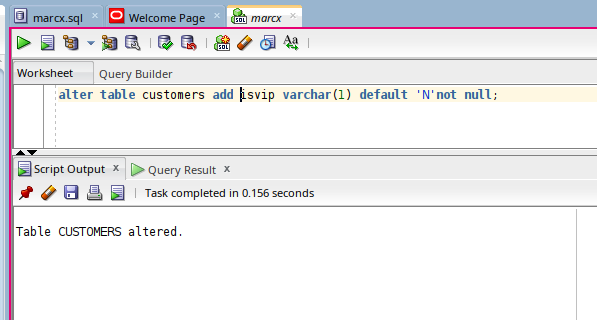


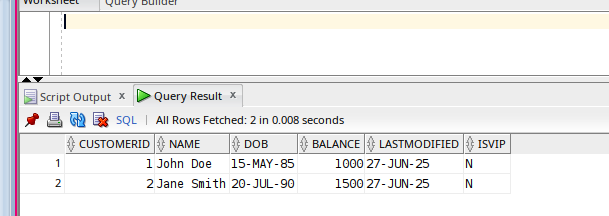
The logic works as follows :

1. The begin keyword tells that the block has started. If this were a named block, the block would have been started from create or replace procedure proc\_name.
2. The logic of pl sql starts after the begin keyword, since we have to iterate over all customers having 60 or higher age, we’ll use a loop. Now since the number of records are always fixed, under a condition use of for each kind of loop is preferable under resultset of **JOIN customers c ON l.customerid = c.customerid WHERE TRUNC(MONTHS\_BETWEEN(SYSDATE, c.dob) / 12) > 60.**
3. We update the interest rate in each iteration, using a simple DML inside the loop (see image).
4. The end keyword marks the end of the anonymous block and tells the Query Execution Engine to stop.

Scenario 2:

First of all, **there is no isVIP field in the schema of the customers table** of the database. Being a privileged user, one can alter table schema by default value of ‘No’ or ‘N’. The point here to be noted is there is no boolean type in sql, so we can’t simply save space by assigning a boolean type to isVIP attribute, we'll have to go through the hard way making it a character or varchar.

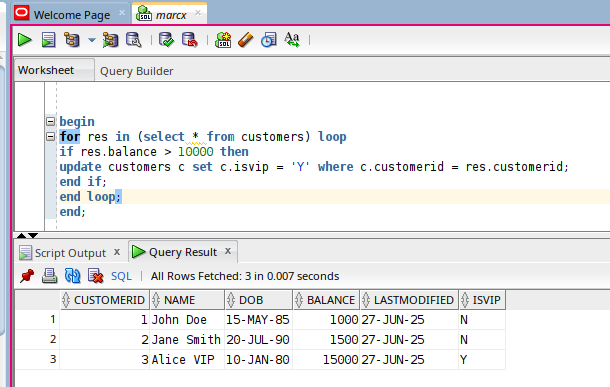




Then we can simply move forward with our pl sql block using the following control structure(s): for loop, if statement.

Since a customer can be upgraded on the basis of their balance, we iterate over all the

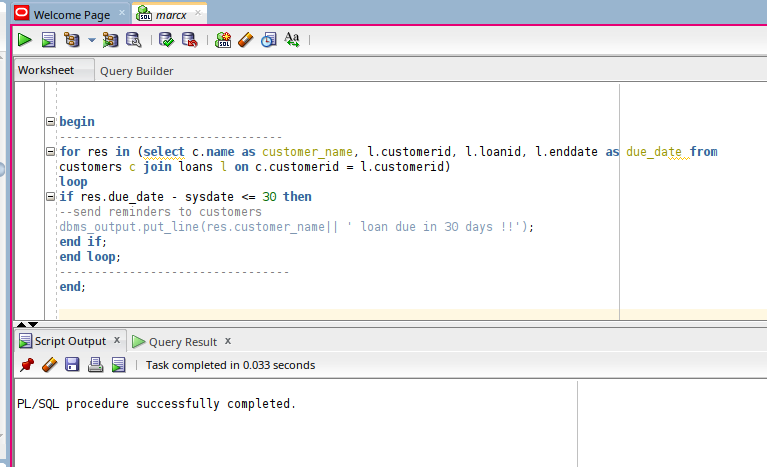
customers and if we find that his/her balance is > 10000, set his status as vip.



The logic works as follows:

1. The block begins with the begin keyword, indicating the start of an anonymous PL/SQL block. Since it's not a named block like a procedure or function, we don't use CREATE OR REPLACE; instead, the logic starts directly with BEGIN.
2. Next, we declare a FOR loop using a cursor that iterates over the result of SELECT \* FROM customers. This means the block will go through each row in the customers table one by one. Since the number of rows is predetermined and relatively small, using a cursor FOR loop is both simple and efficient.
3. Inside the loop, we apply a condition using the IF statement. For every customer record retrieved, we check if the balance is greater than 10,000. If the condition is true (i.e., the customer’s balance is above 10,000), we perform an UPDATE operation. The statement sets the isvip column to 'Y' for that specific customer. The WHERE clause ensures only the matched record is updated based on the unique customerid.
4. Finally, the block ends with the END; keyword, indicating that the PL/SQL execution logic is complete. At this point, Oracle’s Query Execution Engine finishes processing.

Scenario 3:



The logic works as follows:

1. The main logic begins with a FOR loop, which iterates over the result set of a JOIN query between the customers and loans tables. The query selects each customer's name, customer ID, loan ID, and loan end date, which is aliased as due\_date. This cursor loop is efficient for fixed-size result sets and allows us to process each record individually.
2. Inside the loop, a condition is checked using an IF statement. It calculates the difference in days between the current date (SYSDATE) and the loan's due date (res.due\_date). If the due date is within the next 30 days (i.e., res.due\_date - SYSDATE <= 30), the condition evaluates to true. This ensures that only loans due soon (in the next month) are considered.
3. When the condition is satisfied, the program uses DBMS\_OUTPUT.PUT\_LINE to print a message on the screen, reminding the customer by name that their loan is due in 30 days. This serves as a placeholder for any future notification logic, such as sending emails or SMS alerts.
4. Finally, the END LOOP marks the end of the loop, and END; closes the PL/SQL block, signaling to Oracle's Query Execution Engine that execution is complete.