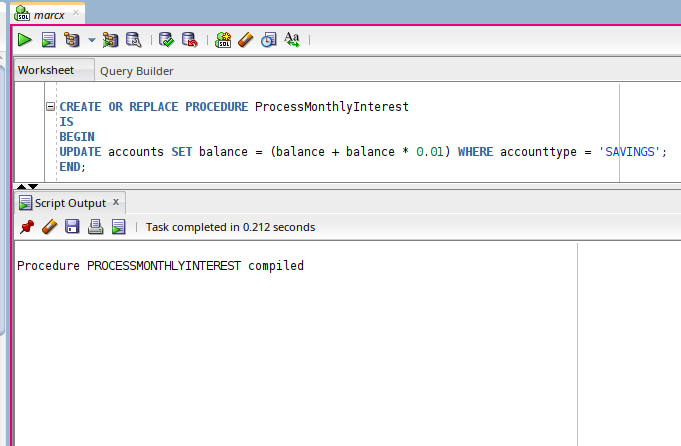
| *Exercise 3: Stored Procedures* |
| --- |

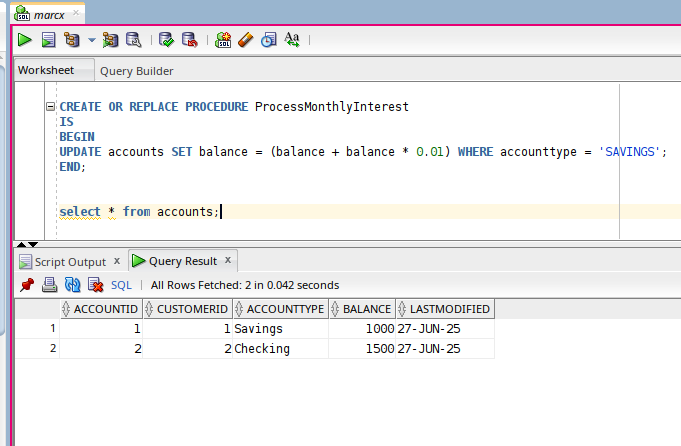
Scenario 1:

The problem states to create a stored procedure to calculate + update the balance by applying an interest rate. The problem can be easily solved by creating a stored procedure, and since the interest rate is already known, there is no need for specific parameters in or out of the block.

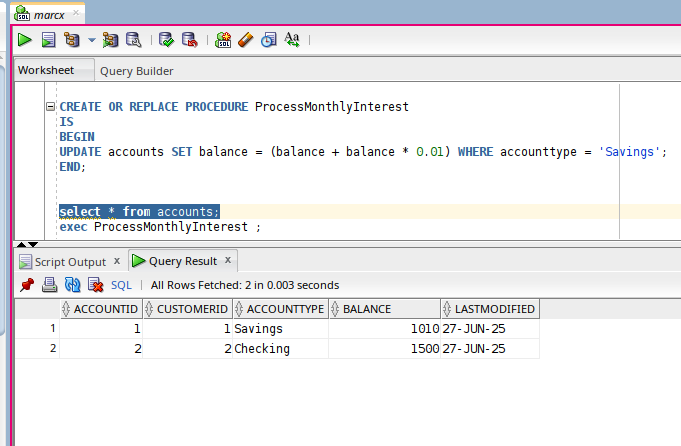


To execute the procedure, exec keyword followed by the procedure name is used, i.e.

before executing :



After executing :

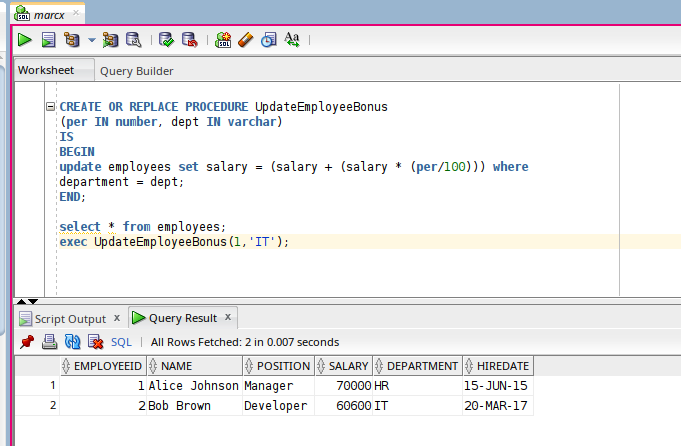


The block works as follows :

1. The block begins with CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest, which defines a **named PL/SQL stored procedure**. The OR REPLACE clause allows you to recreate the procedure without manually dropping it if it already exists.
2. Following the procedure declaration, the IS keyword indicates the beginning of the procedure body. Since this procedure doesn’t take any input parameters or declare local variables, it directly proceeds to the BEGIN keyword, which marks the start of the executable section.
3. Within the BEGIN ... END block, the logic performs a single SQL operation: an UPDATE on the accounts table. This update targets only those rows where the accounttype is 'Savings', implying that interest is being calculated **only for savings accounts**.
4. The balance of each matching account is increased by 1% of its current value. This is done by the expression balance + balance \* 0.01, which calculates and applies the interest in one atomic step, ensuring data consistency.
5. Finally, the END; statement marks the end of the procedure's logic. Once compiled, this procedure can be invoked at any time using a simple EXEC ProcessMonthlyInterest; command, allowing the interest update to be triggered manually or scheduled automatically via a job.

Scenario 2:

Now this problem asks to implement an UpdateEmployeeBonus procedure that would be parametrised with the bonus percentage and given department name. The problem if looked closely resembles the previous one except it’s going to be parametrised. The procedure then can be called using exec prec\_name param1, param2 syntax; or exec prc\_name (param1,param2); depending on the version of the oracle db used.



The block works as follows:

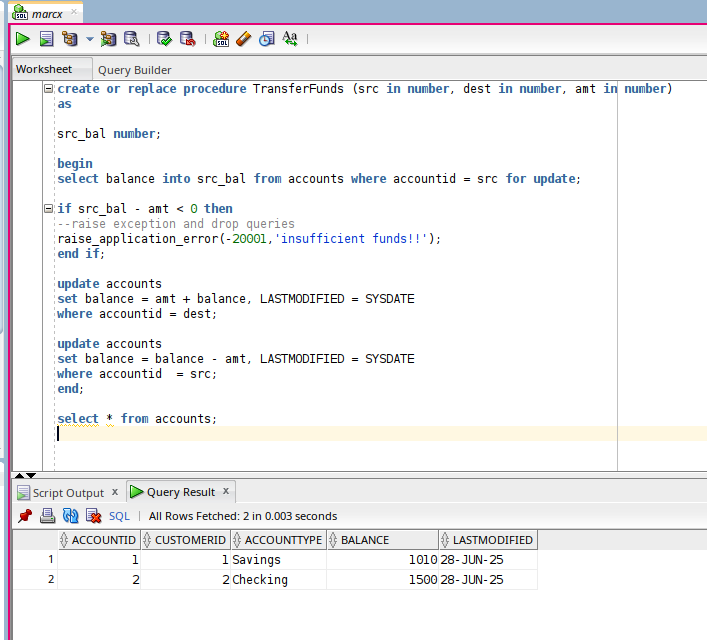
1. The procedure is defined using the CREATE OR REPLACE PROCEDURE statement with the name UpdateEmployeeBonus. The OR REPLACE part ensures that if a procedure with the same name already exists, it will be replaced without needing to drop it manually.
2. The procedure accepts **two parameters**: per (a NUMBER type) representing the **percentage of bonus** to apply, and dept (a VARCHAR) which identifies the **target department** whose employees should receive the bonus.
3. The IS keyword marks the beginning of the procedure body, followed by the BEGIN keyword which starts the executable section.
4. Within the body, the procedure executes an UPDATE statement on the employees table. It calculates the new salary for each employee in the given department by adding a bonus. The expression (salary + (salary \* (per / 100))) increases each employee’s current salary by the specified percentage.
5. The WHERE clause ensures that only employees belonging to the department specified by the dept parameter are updated. This makes the bonus update **targeted and dynamic**, based on the inputs provided when calling the procedure.
6. The block ends with the END; statement, completing the procedure definition. Once compiled, this procedure can be called using: **EXEC UpdateEmployeeBonus(10, 'HR');**
7. This would give a **10% bonus** to all employees in the **HR department**.

Scenario 3:

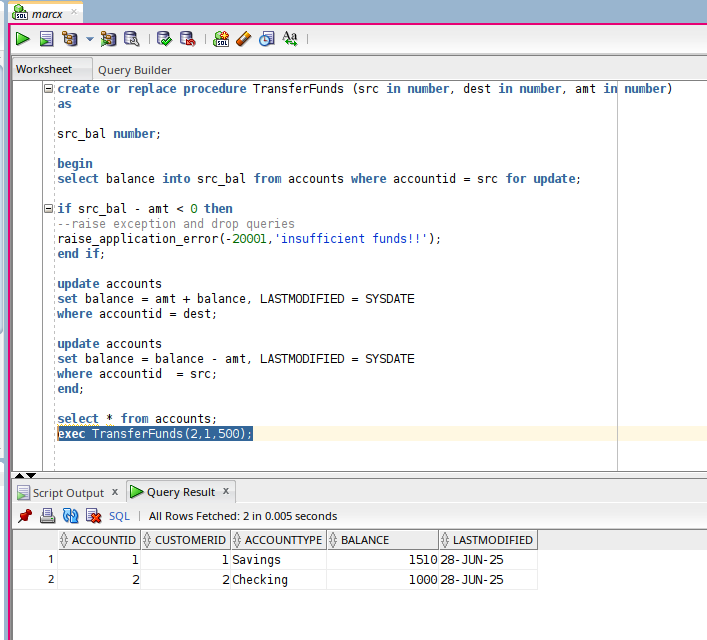
Now this problem is interesting, it asks to perform a classical transfer of funds from one account to another. This procedure requires a specialized “for update” keyword in the sql block, to let the engine know that we need to deal with concurrency here, as quoted by cockroach labs : *“SELECT FOR UPDATE is a SQL command that’s useful in the context of transactional workloads. It allows you to “lock” the rows returned by a SELECT query until the entire transaction that query is part of has been committed.”*

It also uses exceptional handling when the transfer funds are not sufficient in the source account, and stops execution of query.

Before transfer :



After execution of transfer funds procedure:



The procedure works as follows:

1. The procedure TransferFunds is created using the CREATE OR REPLACE PROCEDURE statement, which allows Oracle to overwrite the existing procedure if it already exists. It accepts three input parameters: src (source account ID), dest (destination account ID), and amt (the amount to be transferred), all of type NUMBER.
2. Inside the declaration section (AS), a local variable src\_bal of type NUMBER is declared. This variable will temporarily hold the balance of the source account for validation purposes.
3. The BEGIN keyword marks the start of the executable block. The procedure first retrieves the current balance of the source account by executing a SELECT ... INTO query, and uses the FOR UPDATE clause to lock the corresponding row. This ensures that no other transaction can modify the source account’s balance while the transfer is being processed.
4. Next, an IF condition checks whether the source account has enough funds to complete the transfer. This is done by comparing src\_bal - amt. If the result is less than zero, a custom exception is raised using RAISE\_APPLICATION\_ERROR with error code -20001 and the message 'insufficient funds!!'. This prevents the transaction from proceeding and automatically rolls back any changes made so far.
5. If there are sufficient funds, the procedure proceeds to update the destination account. It increases the balance by amt and updates the LASTMODIFIED column to the current date and time using SYSDATE.
6. Finally, it updates the source account by subtracting the transfer amount from its balance and again updating the LASTMODIFIED column.
7. The block ends with the END; statement, which marks the completion of the procedure. Once compiled, the procedure can be executed using EXEC TransferFunds(2, 1, 500);, which will attempt to transfer ₹500 from account ID 2 to account ID 1, only if account 2 has sufficient balance.