**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1. What is the primary difference between stored procedures and cursors in DBMS?
   1. Stored procedures are used for data retrieval, while cursors are used for data manipulation
   2. Stored procedures can only be executed by the database administrator, while cursors can be executed by users
   3. **Stored procedures are predefined and stored in the database, while cursors are created and used dynamically**
   4. Stored procedures can return a result set, while cursors cannot
2. Which type of trigger is executed before the modification of data in DBMS?
   1. **Before trigger**
   2. After trigger
   3. Instead of trigger
   4. All of the above
3. Which type of trigger is executed after the modification of data in DBMS?
   1. Before trigger
   2. **After trigger**
   3. Instead of trigger
   4. All of the above
4. Concurrency control in DBMS is used to:
   1. **Ensure data consistency during concurrent transaction execution**
   2. Prevent any transaction from accessing the database
   3. Allow multiple transactions to access and modify the same data simultaneously
   4. Control the order in which transactions are executed
5. The schedule of transactions that preserves the order of all transactions is called:
   1. Conflict-serializable schedule
   2. **Serializable schedule**
   3. Recoverable schedule
   4. Strict schedule
6. Which of the following is NOT an Oracle-supported trigger?
   1. BEFORE
   2. **DURING**
   3. AFTER
   4. INSTEAD OF
7. The recoverability property in a DBMS ensures that:
   1. **Transactions can be undone or redone**
   2. Transactions execute serially
   3. Data integrity is maintained
   4. Deadlocks are prevented
8. Which of the following is a benefit of deferred database modifications?
   1. **Increased concurrency**
   2. Improved data integrity
   3. Reduced disk space usage
   4. Faster transaction processing
9. What are the after triggers?
   1. Triggers generated after a particular operation
   2. **These triggers run after an insert, update or delete on a table**
   3. These triggers run after an insert, views, update or delete on a table
   4. All of the mentioned
10. The variables in the triggers are declared using
    1. –
    2. **@**
    3. /
    4. /@

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

1. What will be printed by the following PL/SQL block?

DECLARE

a number;

b number;

c number;

PROCEDURE findMin(x IN number, y IN number, z OUT number) IS

BEGIN

IF x < y THEN

z:= x;

ELSE

z:= y;

END IF;

END;

BEGIN

a:= 2;

b:= 5;

findMin(a, b, c);

dbms\_output.put\_line(c);

END;

* 1. **2**
  2. 5
  3. 0
  4. Won’t print anything

1. Which of the following is not the FIRING POINT: AFTER?
   1. AFTER INSERT TRIGGER
   2. AFTER UPDATE TRIGGER
   3. **BEFORE DELETE TRIGGER**
   4. AFTER DELETE TRIGGER
2. Which of the following is an optional parameter in Oracle BEFORE INSERT/UPDATE/DELETE trigger statement?
3. Trigger\_name
4. Table\_name
5. **OR REPLACE**
6. None
7. Which of the following procedures of DBMS\_SQL supports bulk DML operations?
8. BIND\_VARIABLE\_CHAR
9. BIND\_VARIABLE\_RAW
10. **BIND\_ARRAY**
11. BIND\_VARIABLE
12. Which is oracle supplied package?
13. **DBMS\_SQL**
14. DB\_ORACLE
15. DB\_SQl
16. DBMS\_OBJ

**SECTION-C(Coding Question) (4x5 marks=20 marks)**

1. Write a PL/SQL program that create Trigger to enforce a unique constraint on the combination of "first\_name" and "last\_name".

Solution:

**-- Create the "employee" table**

**CREATE TABLE employee (**

**employee\_id NUMBER PRIMARY KEY,**

**first\_name VARCHAR2(50),**

**last\_name VARCHAR2(50),**

**department VARCHAR2(50),**

**salary NUMBER**

**);**

**-- Insert sample records into the "employee" table**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (1, 'John', 'Doe', 'HR', 50000);**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (2, 'Jane', 'Smith', 'Finance', 60000);**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (3, 'Michael', 'Johnson', 'IT', 70000);**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (4, 'Merry', 'Agarwal', 'IT', 20000);**

**CREATE OR REPLACE TRIGGER trg\_unique\_name**

**BEFORE INSERT OR UPDATE OF first\_name, last\_name ON employee**

**FOR EACH ROW**

**DECLARE**

**cnt NUMBER;**

**BEGIN**

**SELECT COUNT(\*) INTO cnt**

**FROM employee**

**WHERE first\_name = :NEW.first\_name AND last\_name = :NEW.last\_name;**

**IF cnt > 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20002, 'Employee with the same name already exists.');**

**END IF;**

**END;**

**/**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (6, 'John', 'Doe', 'HR', 60000);**

1. Write a PL/SQL program to create Package with a function to find the factorial of a number.

Solution:

**CREATE OR REPLACE PACKAGE math\_operations AS**

**FUNCTION calculate\_factorial(number\_in NUMBER) RETURN NUMBER;**

**END math\_operations;**

**CREATE OR REPLACE PACKAGE BODY math\_operations AS**

**FUNCTION calculate\_factorial(number\_in NUMBER) RETURN NUMBER IS**

**result NUMBER := 1;**

**BEGIN**

**IF number\_in = 0 THEN**

**RETURN 1;**

**ELSE**

**FOR i IN 1..number\_in LOOP**

**result := result \* i;**

**END LOOP;**

**RETURN result;**

**END IF;**

**END;**

**END math\_operations;**

**BEGIN**

**math\_operations.calculate\_factorial(5);**

**END;**

**/**

1. Write a PL/SQL procedure that calculates the average salary of all employees and returns it as an output

Solution:

**-- Create the "employee" table**

**CREATE TABLE employee (**

**employee\_id NUMBER PRIMARY KEY,**

**first\_name VARCHAR2(50),**

**last\_name VARCHAR2(50),**

**department VARCHAR2(50),**

**salary NUMBER**

**);**

**-- Insert sample records into the "employee" table**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (1, 'John', 'Doe', 'HR', 50000);**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (2, 'Jane', 'Smith', 'Finance', 60000);**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (3, 'Michael', 'Johnson', 'IT', 70000);**

**INSERT INTO employee (employee\_id, first\_name, last\_name, department, salary)**

**VALUES (4, 'Merry', 'Agarwal', 'IT', 20000);**

**CREATE OR REPLACE PROCEDURE calculate\_average\_salary(p\_avg\_salary OUT NUMBER) AS**

**BEGIN**

**SELECT AVG(salary)**

**INTO p\_avg\_salary**

**FROM employee;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**p\_avg\_salary := 0;**

**END;**

**/**

1. Write a bodiless package to check if a given string is a palindrome or not.

Solution:

**CREATE OR REPLACE PACKAGE palindrome\_check\_pkg AS**

**END palindrome\_check\_pkg;**

**/**

**CREATE OR REPLACE PACKAGE BODY palindrome\_check\_pkg AS**

**FUNCTION is\_palindrome(input\_string VARCHAR2) RETURN BOOLEAN IS**

**reversed\_string VARCHAR2(100);**

**BEGIN**

**reversed\_string := REVERSE(input\_string);**

**RETURN input\_string = reversed\_string;**

**END is\_palindrome;**

**PROCEDURE display\_palindrome\_result(input\_string VARCHAR2) IS**

**result BOOLEAN;**

**BEGIN**

**result := is\_palindrome(input\_string);**

**IF result THEN**

**DBMS\_OUTPUT.PUT\_LINE(input\_string || ' is a palindrome.');**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE(input\_string || ' is not a palindrome.');**

**END IF;**

**END display\_palindrome\_result;**

**END palindrome\_check\_pkg;**

**/**