



GATE INSTITUTE OF TECHNOLOGY
& MANAGEMENT SCIENCES

(Approved by AICTE, Affiliated to Sri Venkateswara University, TIRUPATI)

ICET CODE : GTMT



Master of Computer Applications

SEMESTER - II

Hall Ticket No: 122531043

Name of the Student: K. Raja Sree

Subject: Advanced Database Management System

Year: Ist Semester: II Batch 2025-26



GATE EDUCATIONAL INSTITUTIONS

DEPARTMENT OF COMPUTERS

YEAR I SEMESTER II

Reg No. 122531043

CERTIFICATE

Certified that this is the bonafide record of practical work done in the laboratory

by the candidate K. RajaSree *during the*
academic year 2025-27 *subject* Advanced Database management system

No. of Practical's Conducted 15

No. of Practical's Attended 15

LECTURER

HEAD OF THE DEPARTMENT

Submitted for the Practical Examination held on _____

Examiners

1.

2.

INDEX

S.No.	Date	Name of the Experiment	Page No.	Marks Awarded	Remarks
1.	9/3/26	SQL Aggregate functions with example	1-5		
2.	11/3/26	PL/SQL program for electricity bill generation using cursor.	6-11		
3.	13/3/26	PL/SQL program to generate ICET Rank card report with grade Assignment.	12-17		
4.	16/3/26	Exception handling in PL/SQL Examples.	18-23		
5.	18/3/26	Database Trigger to track bank Transaction.	24-29		
6.	23/3/26	SQL commands using Master child Tables.	30-36		
7.	25/3/26	PL/SQL program to find factorial of a number.	35-39		
8.	30/3/26	PL/SQL program to check whether a number is palindrome.	40-44		

INDEX

S.No.	Date	Name of the Experiment	Page No.	Marks Awarded	Remarks
9.	11/4/26	PL/SQL Demonstration of pre-defined exception.	45-49		
10.	6/4/26	PL/SQL program using user-defined exception for divide operation.	50-55		
11.	8/4/26	PL/SQL program for library book issue system with Exception handling.	56-61		
12.	9/4/26	practical usage of predefined Oracle packing in PL/SQL	62-68		
13.	13/4/26	create and manage object types, tables and methods using Oracle SQL.	69-72		
14.	16/4/26	Selection, projection and sorting queries on a simple table.	73-77		
15.	20/4/26	ER diagram for a company database.	78-83		

Date :	GATE Educational Institutions	Page No. : 1
	SQL Aggregate functions with Examples.	Practical No. : 1

Aim :—

To study and demonstrate the various SQL aggregate functions such as COUNT, SUM, AVG, MAX and MIN with practical examples on a relational database table.

Description :—

SQL Aggregate functions are built-in functions that perform a calculation on a set of values and return a single summary value. They are commonly used with the GROUP BY clause to group result sets by one or more columns. These functions are fundamental to data analysis reporting and business intelligence queries.

The main SQL Aggregate functions are :

1. COUNT() - Returns the number of rows that match a specified condition.
2. SUM() - Returns the total sum of a numeric column.
3. AVG() - Returns the average value of a numeric column.
4. MAX() - Returns the largest value in a selected.
5. MIN() - Returns the smallest value in a selected.

Date :	GATE Educational Institutions	Page No. : 2
		Practical No. :

Real-world relevance : Aggregate functions are used in banking, e-commerce, HR, and academic systems.

Algorithm :-

1. State the SQL environment and connect to the database.
2. create a table named EMPLOYEE with columns: EMP_ID, EMP_NAME, DEPARTMENT, SALARY.
3. Insert sample records into the employee table.
4. Use COUNT(*) to count the total number of employees.
5. Use SUM(SALARY) to compute the total salary expenditure.
6. Use AVG(SALARY) to calculate the average salary across all employees.
7. Use MAX(SALARY) to identify the highest salary.
8. Use MIN(SALARY) to identify the lowest salary.
9. Use GROUP BY clause with aggregate functions to get department wise statistics.
10. Display and verify the results.

Date :

GATE Educational Institutions

Page No. : 3

Practical No. :

1. SQL Aggregate Functions with Examples

SQL CODE

```
-- Step 1: Create Table
CREATE TABLE EMPLOYEES (
  EMP_ID NUMBER PRIMARY KEY,
  EMP_NAME VARCHAR2(50),
  DEPARTMENT VARCHAR2(30),
  SALARY NUMBER(10,2)
);

-- Step 2: Insert Records
INSERT INTO EMPLOYEES VALUES (101, 'Ravi Kumar', 'IT', 65000);
INSERT INTO EMPLOYEES VALUES (102, 'Priya Sharma', 'HR', 52000);
INSERT INTO EMPLOYEES VALUES (103, 'Arjun Nair', 'IT', 75000);
INSERT INTO EMPLOYEES VALUES (104, 'Sneha Rao', 'Finance', 80000);
INSERT INTO EMPLOYEES VALUES (105, 'Vikram Singh', 'HR', 48000);
INSERT INTO EMPLOYEES VALUES (106, 'Deepa Menon', 'Finance', 91000);
INSERT INTO EMPLOYEES VALUES (107, 'Kiran Patel', 'IT', 70000);
COMMIT;

-- Step 3: COUNT – Total number of employees
SELECT COUNT(*) AS TOTAL_EMPLOYEES FROM EMPLOYEES;

-- Step 4: SUM – Total salary
SELECT SUM(SALARY) AS TOTAL_SALARY FROM EMPLOYEES;

-- Step 5: AVG – Average salary
SELECT ROUND(AVG(SALARY), 2) AS AVERAGE_SALARY FROM EMPLOYEES;

-- Step 6: MAX – Highest salary
SELECT MAX(SALARY) AS MAX_SALARY FROM EMPLOYEES;

-- Step 7: MIN – Lowest salary
SELECT MIN(SALARY) AS MIN_SALARY FROM EMPLOYEES;

-- Step 8: GROUP BY – Department-wise statistics
SELECT DEPARTMENT,
  COUNT(*) AS EMP_COUNT,
  SUM(SALARY) AS DEPT_TOTAL,
  ROUND(AVG(SALARY), 2) AS DEPT_AVG,
  MAX(SALARY) AS HIGHEST,
  MIN(SALARY) AS LOWEST
FROM EMPLOYEES
GROUP BY DEPARTMENT
ORDER BY DEPARTMENT;
```


Date :

GATE Educational Institutions

Page No. : 4

Practical No. :

OUTPUT

TOTAL_EMPLOYEES

7

TOTAL_SALARY

481000

AVERAGE_SALARY

68714.29

MAX_SALARY

91000

MIN_SALARY

48000

DEPARTMENT EMP_COUNT DEPT_TOTAL DEPT_AVG HIGHEST LOWEST

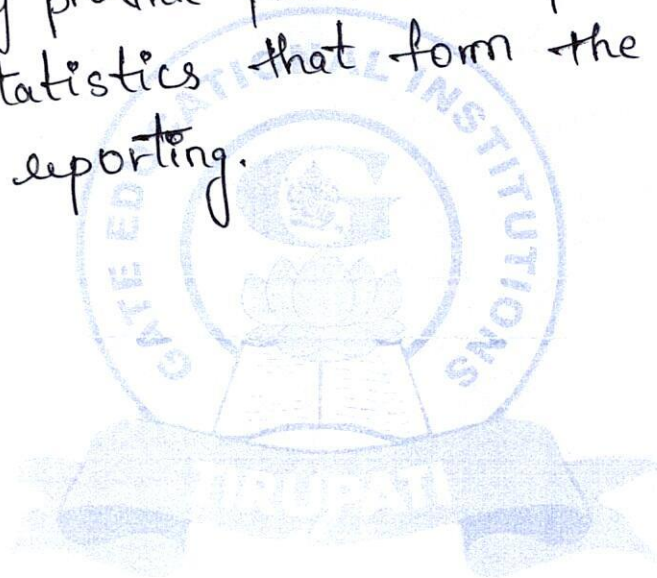
Finance	2	171000	85500.00	91000	80000
HR	2	100000	50000.00	52000	48000
IT	3	210000	70000.00	75000	65000



Date :	GATE Educational Institutions	Page No. : <
		Practical No. :

conclusion :—

SQL Aggregate functions (COUNT, SUM, AVG, SUM, MAX, MIN) were successfully demonstrated. These functions are essential for summarizing and analyzing data in relational databases. When combined with GROUP BY and ORDER BY clauses, they provide powerful departmental and categorical statistics that form the backbone of database reporting.



Aim :—

To develop a PL/SQL program that generates electricity bills for all customers using an explicit cursor and applies multiple slab-based billing conditions to compute the final bill amount.

Description :—

Electricity billing systems use tiered or slab-based pricing, where the cost per unit increases as consumption increases. A PL/SQL cursor is used to process each customer record one by one from the database. For each customer, the consumed units are read and the bill amount is computed based on the applicable slab rates.

Slab rate structure used :

Units 1 - 100 : Rs. 1.50 per unit (Basic slab)

Units 101 - 200 : Rs. 2.50 per unit (medium slab)

Units 201 - 300 : Rs. 4.00 per unit (high slab)

Units > 300 : Rs. 6.00 per unit (premium slab)

A fixed service charge of Rs. 50 is added to every bill. An explicit cursor fetches all consumer records and the IF-ELSIF-ELSE construct applies the slab logic per consumer.

Date :

GATE Educational Institutions

Page No. : 8

Practical No. :

2. PL/SQL Program for Electricity Bill Generation using Cursor**PL/SQL CODE**

```

-- Step 1: Create Table
CREATE TABLE ELECTRICITY (
    CUST_ID NUMBER PRIMARY KEY,
    CUST_NAME VARCHAR2(50),
    UNITS NUMBER,
    BILL_AMT NUMBER(10,2)
);

-- Step 2: Insert Records
INSERT INTO ELECTRICITY VALUES (1, 'Ramesh', 85, NULL);
INSERT INTO ELECTRICITY VALUES (2, 'Suresh', 150, NULL);
INSERT INTO ELECTRICITY VALUES (3, 'Kavitha', 260, NULL);
INSERT INTO ELECTRICITY VALUES (4, 'Anil', 320, NULL);
INSERT INTO ELECTRICITY VALUES (5, 'Meena', 45, NULL);
COMMIT;

-- Step 3: PL/SQL Block with Cursor
DECLARE
    v_cid ELECTRICITY.CUST_ID%TYPE;
    v_name ELECTRICITY.CUST_NAME%TYPE;
    v_units ELECTRICITY.UNITS%TYPE;
    v_bill NUMBER(10,2);

    CURSOR c_elec IS
        SELECT CUST_ID, CUST_NAME, UNITS FROM ELECTRICITY;

BEGIN
    OPEN c_elec;
    DBMS_OUTPUT.PUT_LINE('-----');
    DBMS_OUTPUT.PUT_LINE('      ELECTRICITY BILL REPORT      ');
    DBMS_OUTPUT.PUT_LINE('-----');
    DBMS_OUTPUT.PUT_LINE('ID Name      Units  Bill Amount (Rs.)');
    DBMS_OUTPUT.PUT_LINE('-----');

    LOOP
        FETCH c_elec INTO v_cid, v_name, v_units;
        EXIT WHEN c_elec%NOTFOUND;

        -- Slab-wise bill computation
        IF v_units <= 100 THEN
            v_bill := v_units * 1.50;
        ELSIF v_units <= 200 THEN
            v_bill := (100 * 1.50) + ((v_units - 100) * 2.50);
        ELSIF v_units <= 300 THEN
            v_bill := (100 * 1.50) + (100 * 2.50) + ((v_units - 200) * 4.00);
        ELSE
            v_bill := (100 * 1.50) + (100 * 2.50) + (100 * 4.00)
                + ((v_units - 300) * 6.00);
        END IF;

        -- Add fixed service charge
        v_bill := v_bill + 50;
    
```


Date :

GATE Educational Institutions

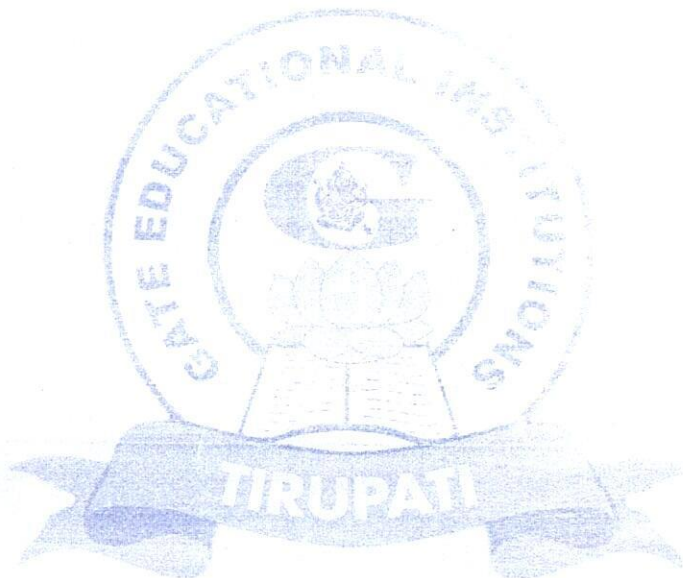
Page No. : 9

Practical No. :

```
-- Update bill in table
UPDATE ELECTRICITY
SET BILL_AMT = v_bill
WHERE CUST_ID = v_cid;

DBMS_OUTPUT.PUT_LINE(
  v_cid || ' ' || RPAD(v_name, 10) ||
  ' ' || LPAD(v_units, 5) ||
  ' Rs.' || v_bill);
END LOOP;

DBMS_OUTPUT.PUT_LINE('-----');
CLOSE c_elec;
COMMIT;
END;
/
```



Date :

GATE Educational Institutions

Page No. : 10

Practical No. :

OUTPUT

ELECTRICITY BILL REPORT

ID	Name	Units	Bill Amount (Rs.)
1	Ramesh	85	Rs. 177.5
2	Suresh	150	Rs. 275
3	Kavitha	260	Rs. 590
4	Anil	320	Rs. 770
5	Meena	45	Rs. 117.5

PL/SQL procedure successfully completed.



Date :

GATE Educational Institutions

Page No. : 1)

Practical No. :

conclusion :-

A PL/SQL program for electricity bill generation was successfully implemented using an explicit cursor and multiple IF - ELSIF - ELSE conditions. The program correctly applied slab - based billing rates for each customer, added the fixed services charge, updated the database and displayed a formatted bill reported. This demonstrates the practical use of PL/SQL cursors in real world billing systems.

Date :	GATE Educational Institutions	Page No. : 12 ✓
	PL/SQL program to generate ICET Rank card report With Grade Assignment.	Practical No. : 3

Aim :—

To develop a PL/SQL program Using a cursor and IF-ELSE conditions to generate an ICET Rank card. Report the fetches student details and assign grades based on their scores.

Description :—

ICET [Integrated common Entrance test] is a state-level entrance examination for admission into MBA and MCA programs. The rank card report displays the candidate's hall ticket number, name, subject-wise marks, total score, and assigned grade.

Grade criteria Used.

- Score ≥ 90 Grade O (Outstanding)
- Score ≥ 75 Grade A⁺ (Excellent)
- Score ≥ 60 Grade A (Very Good)
- Score ≥ 50 Grade B (Good)
- Score ≥ 40 Grade C (Average)
- Score ≥ 40 Grade F (Fail)

An explicit cursor iterates through all student records, computes the total score, assigns a grade via IF-ELSIF-ELSE logic, and prints the rankcard.

Algorithm :-

1. create the ICET-students table with hall ticket number, name and subject scores.
2. Insert sample student records with marks for three subjects.
3. Declare cursor c-let to select all students and their score.
4. Declare Variables for each field and computed values.
5. OPEN the cursor and begin the loop.
6. FETCH each student record into the declared variable.
7. EXIT when cursor % NOT FOUND.
8. compute total = maths + reasoning + communication.
9. Apply IF - ELIF - ELSE to assign grade based on total score.
10. print the formatted rank card entry for each student.
11. CLOSE the cursor after all records are processed.

Date :

GATE Educational Institutions

Page No. : 14

Practical No. :

3. PL/SQL Program to Generate ICET Rank Card Report with Grade Assignment**PL/SQL CODE**

```

-- Step 1: Create Table
CREATE TABLE ICET_STUDENTS (
  HALL_TICKET VARCHAR2(15) PRIMARY KEY,
  STUDENT_NAME VARCHAR2(50),
  MATHS NUMBER(3),
  REASONING NUMBER(3),
  COMM NUMBER(3)
);

-- Step 2: Insert Sample Data
INSERT INTO ICET_STUDENTS VALUES ('ICET2024001','Arjun Reddy', 92, 88, 90);
INSERT INTO ICET_STUDENTS VALUES ('ICET2024002','Priya Lakshmi', 76, 80, 74);
INSERT INTO ICET_STUDENTS VALUES ('ICET2024003','Kiran Babu', 55, 62, 58);
INSERT INTO ICET_STUDENTS VALUES ('ICET2024004','Divya Nair', 42, 38, 45);
INSERT INTO ICET_STUDENTS VALUES ('ICET2024005','Suresh Goud', 30, 28, 35);
COMMIT;

-- Step 3: PL/SQL Block
DECLARE
  v_hall ICET_STUDENTS.HALL_TICKET%TYPE;
  v_name ICET_STUDENTS.STUDENT_NAME%TYPE;
  v_math ICET_STUDENTS.MATHS%TYPE;
  v_res ICET_STUDENTS.REASONING%TYPE;
  v_comm ICET_STUDENTS.COMM%TYPE;
  v_total NUMBER;
  v_grade VARCHAR2(5);

  CURSOR c_icet IS
    SELECT HALL_TICKET, STUDENT_NAME, MATHS, REASONING, COMM
    FROM ICET_STUDENTS
    ORDER BY HALL_TICKET;
BEGIN
  OPEN c_icet;
  DBMS_OUTPUT.PUT_LINE('=====');
  DBMS_OUTPUT.PUT_LINE('      ICET 2024 RANK CARD REPORT      ');
  DBMS_OUTPUT.PUT_LINE('=====');
  DBMS_OUTPUT.PUT_LINE('HallTicket      Name      Math Reas Comm Total Grade');
  DBMS_OUTPUT.PUT_LINE('-----');

  LOOP
    FETCH c_icet INTO v_hall, v_name, v_math, v_res, v_comm;
    EXIT WHEN c_icet%NOTFOUND;

    v_total := v_math + v_res + v_comm;

    IF v_total >= 270 THEN v_grade := 'O';
    ELSIF v_total >= 225 THEN v_grade := 'A+';
    ELSIF v_total >= 180 THEN v_grade := 'A';
    ELSIF v_total >= 150 THEN v_grade := 'B';
    ELSIF v_total >= 120 THEN v_grade := 'C';
    ELSE v_grade := 'F';
    END IF;

    DBMS_OUTPUT.PUT_LINE(
      RPAD(v_hall, 15) || RPAD(v_name, 18) ||
      LPAD(v_math, 4) || LPAD(v_res, 5) ||
      LPAD(v_comm, 5) || LPAD(v_total, 6) ||
      ' ' || v_grade);
  END LOOP;

```

Date :

GATE Educational Institutions

Page No. : 15

Practical No. :

```
DBMS_OUTPUT.PUT_LINE('-----');  
CLOSE c_jcet;  
END;  
/
```



Date :

GATE Educational Institutions

Page No. : 16

Practical No. :

OUTPUT

ICET 2024 RANK CARD REPORT

HallTicket	Name	Math	Reas	Comm	Total	Grade
ICET2024001	Arjun Reddy	92	88	90	270	O
ICET2024002	Priya Lakshmi	76	80	74	230	A+
ICET2024003	Kiran Babu	55	62	58	175	B
ICET2024004	Divya Nair	42	38	45	125	C
ICET2024005	Suresh Goud	30	28	35	93	F

PL/SQL procedure successfully completed.



Date :

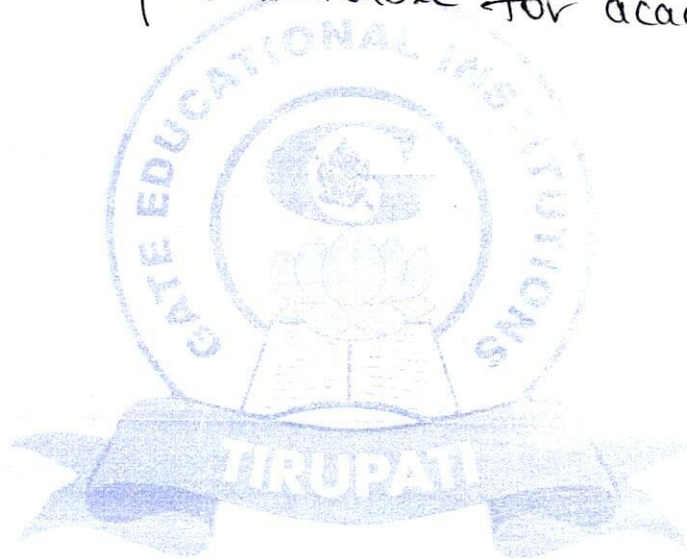
GATE Educational Institutions

Page No. : 12

Practical No. :

conclusion :-

The PL/SQL program for ICET Rank card generation was successfully implemented. The program used an explicit cursor to iterate through all student records, computed the total score, and applied appropriate grades. The formatted output simulates a real-world rank card report suitable for academic system.



Aim :-

To implement a database trigger in Oracle PL/SQL that automatically records every deposit and withdraw transaction into an audit / log table whenever the account balance is updated.

Description :-

A database trigger is a stored PL/SQL block that automatically executed in response to specified DML events on a table. Triggers are used to auditing enforcing business rules, maintaining referential integrity and generating derived data.

In a banking system, it is critical to maintain a complete audit trail of all balance changes. An After Update trigger on the BANK-ACCOUNTS table captures every balance modification and logs it into the TRANSACTION-LOG table with transaction type, amount and time stamp.

Types of Triggers : Before / After, Row - LEVEL, STATEMENT - LEVEL. This program uses an AFTER UPDATE, FOR EACH Row trigger.

Algorithm :-

1. CREATE the BANK-ACCOUNTS table with account number, holder name, and balance.
2. create the TRANSACTION-LOG table with ID, account number, transaction type, amount and time stamp.
3. Create a Sequence TXN-SEQ to generate unique log IDs.
4. Insert sample account records.
5. create a AFTER UPDATE FOR EACH ROW triggers on BANK-ACCOUNTS.
6. Inside the trigger, compare : New Balance with : Old Balance.
7. IF : NEW. BALANCE > OLD BALANCE, record 'DEPOSIT' with difference amount.
8. IF : NEW. BALANCE < OLD BALANCE, record 'WITHDRAWAL' with difference amount.
9. Insert the transaction details into TRANSACTION-LOG.
10. perform test UPDATE on BANK-ACCOUNTS to fire the trigger.
11. Query TRANSACTION-LOG to verify trigger fired correctly.

Date :

GATE Educational Institutions

Page No. : 20

Practical No. :

4. Database Trigger to Track Bank Transactions**PL/SQL CODE**

```

-- Step 1: Create Bank Accounts Table
CREATE TABLE BANK_ACCOUNTS (
  ACC_NO VARCHAR2(15) PRIMARY KEY,
  ACC_NAME VARCHAR2(50),
  BALANCE NUMBER(12,2)
);

-- Step 2: Create Transaction Log Table
CREATE TABLE TRANSACTION_LOG (
  LOG_ID NUMBER PRIMARY KEY,
  ACC_NO VARCHAR2(15),
  TXN_TYPE VARCHAR2(15),
  AMOUNT NUMBER(12,2),
  OLD_BAL NUMBER(12,2),
  NEW_BAL NUMBER(12,2),
  TXN_TIME TIMESTAMP DEFAULT SYSTIMESTAMP
);

-- Step 3: Create Sequence
CREATE SEQUENCE TXN_SEQ START WITH 1 INCREMENT BY 1;

-- Step 4: Insert Sample Accounts
INSERT INTO BANK_ACCOUNTS VALUES ('SB100001', 'Ravi Kumar', 50000);
INSERT INTO BANK_ACCOUNTS VALUES ('SB100002', 'Priya Sharma', 30000);
INSERT INTO BANK_ACCOUNTS VALUES ('SB100003', 'Arjun Nair', 75000);
COMMIT;

-- Step 5: Create Trigger
CREATE OR REPLACE TRIGGER trg_bank_txn
AFTER UPDATE OF BALANCE ON BANK_ACCOUNTS
FOR EACH ROW
DECLARE
  v_txn_type VARCHAR2(15);
  v_amount NUMBER(12,2);
BEGIN
  IF :NEW.BALANCE > :OLD.BALANCE THEN
    v_txn_type := 'DEPOSIT';
    v_amount := :NEW.BALANCE - :OLD.BALANCE;
  ELSEIF :NEW.BALANCE < :OLD.BALANCE THEN
    v_txn_type := 'WITHDRAWAL';
    v_amount := :OLD.BALANCE - :NEW.BALANCE;
  ELSE
    v_txn_type := 'NO CHANGE';
    v_amount := 0;
  END IF;

  INSERT INTO TRANSACTION_LOG
  (LOG_ID, ACC_NO, TXN_TYPE, AMOUNT, OLD_BAL, NEW_BAL)
  VALUES
  (TXN_SEQ.NEXTVAL, :NEW.ACC_NO,
   v_txn_type, v_amount, :OLD.BALANCE, :NEW.BALANCE);
END;
/

-- Step 6: Test the Trigger
UPDATE BANK_ACCOUNTS SET BALANCE = 60000 WHERE ACC_NO = 'SB100001'; -- Deposit
UPDATE BANK_ACCOUNTS SET BALANCE = 20000 WHERE ACC_NO = 'SB100002'; -- Withdrawal
UPDATE BANK_ACCOUNTS SET BALANCE = 95000 WHERE ACC_NO = 'SB100003'; -- Deposit
COMMIT;

```

Date :	GATE Educational Institutions	Page No. : 21
		Practical No. :

```
-- Step 7: Verify Log
SELECT LOG_ID, ACC_NO, TXN_TYPE, AMOUNT, OLD_BAL, NEW_BAL,
       TO_CHAR(TXN_TIME, 'DD-MON-YYYY HH24:MI:SS') AS TXN_TIME
FROM TRANSACTION_LOG
ORDER BY LOG_ID;
```



Date :

GATE Educational Institutions

Page No. : 22

Practical No. :

OUTPUT

Trigger TRG_BANK_TXN created.

3 rows updated.

LOG_ID	ACC_NO	TXN_TYPE	AMOUNT	OLD_BAL	NEW_BAL	TXN_TIME
--------	--------	----------	--------	---------	---------	----------

1	SB100001	DEPOSIT	10000	50000	60000	01-JAN-2025 10:15:32
2	SB100002	WITHDRAWAL	10000	30000	20000	01-JAN-2025 10:15:32
3	SB100003	DEPOSIT	20000	75000	95000	01-JAN-2025 10:15:32



Date :

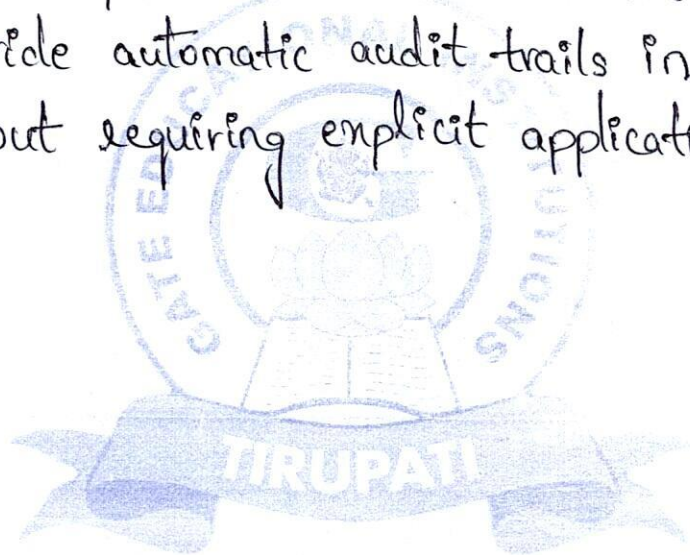
GATE Educational Institutions

Page No. : 23

Practical No. :

conclusion :-

A database trigger was successfully created and tested to track all bank account balance changes. The AFTER UPDATE FOR EACH ROW Trigger automatically classified each change as DEPOSIT or WITHDRAWAL and inserted a time stamped record into the TRANSACTION-LOG table. This demonstrates how Oracle triggers provide automatic audit trails in financial system without requiring explicit application level logging.



Aim :—

To understand and demonstrate Exception handling in PL/SQL Using predefined Exceptions. User defined exceptions, and the OTHERS Exception handler with practical examples.

Description :—

Exception handling is a critical feature of PL/SQL that allow programming to gracefully manage runtime errors without abruptly terminating the program. An exception is an error or warning condition that disrupts the normal execution flow of a PL/SQL block.

PL/SQL Exception categories :

1. predefined Exceptions - Built-in Oracle Exceptions.
 2. Non-predefined Exceptions - Oracle errors without name : declared using PRAGMA EXCEPTION-INIT
 3. User-defined Exceptions - custom exceptions declared and raised by the programmer using RAISE.
- structure of Exception handling Block ;

```

DECLARE ..... BEGIN ..... EXCEPTION
WHEN <Exception>
THEN ..... END ;
  
```

Algorithm :-

1. create a demo table students with ID, name , and marks.
2. Insert a few sample records.
3. Write Eg:- 1 : Handle NO - DATA - FOUND when SELECT returns NO - ROW.
4. Write eg:- 2 : Handle TOO - MANY - ROWS when SELECT INTO fetches multiple rows.
5. Write eg:- 3 : Handle ZERO - DIVIDE exception.
6. Write eg:- 4 : Declare and raise a user - defined exception.
7. Write eg:- 5 : Use WHEN OTHERS to each Unexpected exception.
8. Use SQLCODE and SQLERRM in others handler for error diagnostics.
9. Execute each block and observe the exception messages.

Date :

GATE Educational Institutions

Page No. : 26

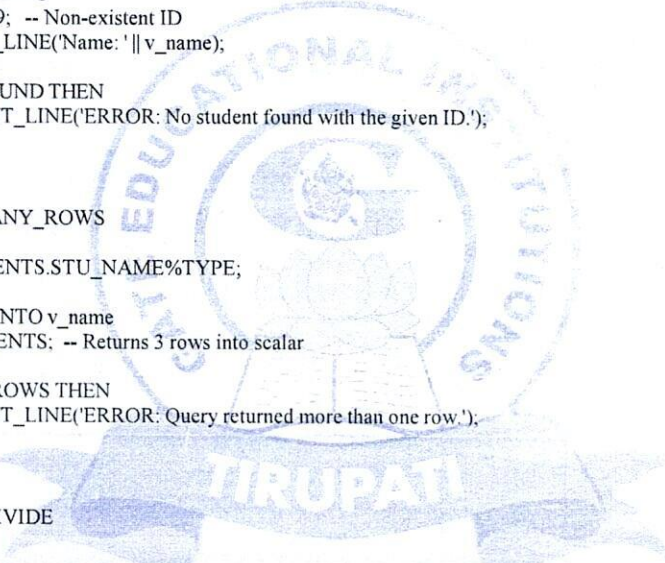
Practical No. :

5. Exception Handling in PL/SQL with Examples**PL/SQL CODE**

```

-- Setup Table
CREATE TABLE DEMO_STUDENTS (
  STU_ID NUMBER PRIMARY KEY,
  STU_NAME VARCHAR2(40),
  MARKS NUMBER(3)
);
INSERT INTO DEMO_STUDENTS VALUES (1, 'Ankit', 85);
INSERT INTO DEMO_STUDENTS VALUES (2, 'Bhavana', 72);
INSERT INTO DEMO_STUDENTS VALUES (3, 'Charan', 90);
COMMIT;

-- EXAMPLE 1: NO_DATA_FOUND
DECLARE
  v_name DEMO_STUDENTS.STU_NAME%TYPE;
BEGIN
  SELECT STU_NAME INTO v_name
  FROM DEMO_STUDENTS
  WHERE STU_ID = 999; -- Non-existent ID
  DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('ERROR: No student found with the given ID.');
```



```

END;
/

-- EXAMPLE 2: TOO_MANY_ROWS
DECLARE
  v_name DEMO_STUDENTS.STU_NAME%TYPE;
BEGIN
  SELECT STU_NAME INTO v_name
  FROM DEMO_STUDENTS; -- Returns 3 rows into scalar
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('ERROR: Query returned more than one row.');
```

TIRUPATI

```

END;
/

-- EXAMPLE 3: ZERO_DIVIDE
DECLARE
  v_result NUMBER;
BEGIN
  v_result := 100 / 0;
  DBMS_OUTPUT.PUT_LINE('Result: ' || v_result);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('ERROR: Division by zero is not allowed.');
```

TIRUPATI

```

END;
/

-- EXAMPLE 4: User-Defined Exception
DECLARE
  e_low_marks EXCEPTION;
  v_marks NUMBER := 28;
BEGIN
  IF v_marks < 35 THEN
    RAISE e_low_marks;
  END IF;
  DBMS_OUTPUT.PUT_LINE('Student Passed.');
```

TIRUPATI

```

EXCEPTION
  WHEN e_low_marks THEN
```

Date :

GATE Educational Institutions

Page No. : 27

Practical No. :

DBMS_OUTPUT.PUT_LINE('ALERT: Student has scored below minimum pass marks (35).');
END;

```
-- EXAMPLE 5: WHEN OTHERS with SQLCODE / SQLERRM  
DECLARE  
  v_num NUMBER := 'ABC'; -- Triggers VALUE_ERROR  
BEGIN  
  DBMS_OUTPUT.PUT_LINE('Value: ' || v_num);  
EXCEPTION  
  WHEN OTHERS THEN  
    DBMS_OUTPUT.PUT_LINE('Unexpected Error!');  
    DBMS_OUTPUT.PUT_LINE('Error Code : ' || SQLCODE);  
    DBMS_OUTPUT.PUT_LINE('Error Msg : ' || SQLERRM);  
END;  
/
```



Date :

GATE Educational Institutions

Page No. : 28

Practical No. :

OUTPUT

-- Example 1:

ERROR: No student found with the given ID.

-- Example 2:

ERROR: Query returned more than one row.

-- Example 3:

ERROR: Division by zero is not allowed.

-- Example 4:

ALERT: Student has scored below minimum pass marks (35).

-- Example 5:

Unexpected Error!

Error Code : -6502

Error Msg : ORA-06502: PL/SQL: numeric or value error: character to number conversion error



Date :

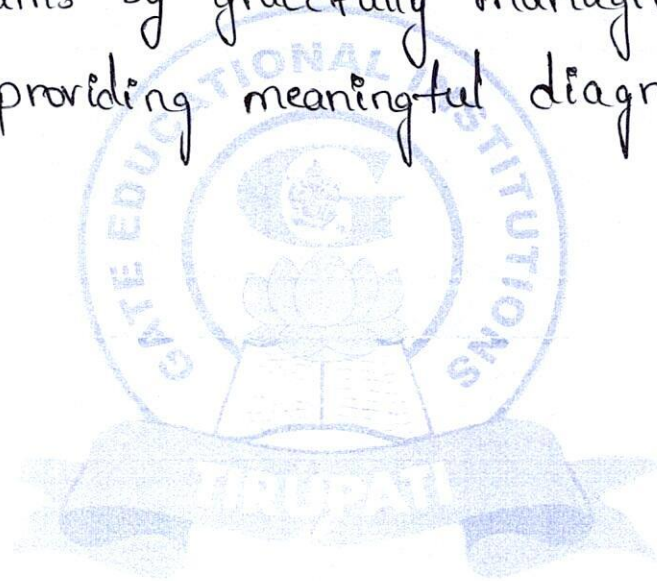
GATE Educational Institutions

Page No. : 29

Practical No. :

conclusion :—

Exception handling in PL/SQL was successfully demonstrated using five practical examples covering predefined exception, user defined exceptions, and the WHEN OTHERS generic handler. proper exception handling ensures robust, fault-tolerant PL/SQL programs by gracefully managing runtime errors and providing meaningful diagnostic messages.



Date :	GATE Educational Institutions	Page No. : 30
	SQL commands Using Master child tables.	Practical No. : 6

Aim :-

To demonstrate SQL commands using Master-child table relationships, including creation of foreign key constraints, cascading operations and JOIN queries.

Description :-

In relational databases a master-child relationship exists when one table contains primary key data that is referenced by another table via a foreign key. This enforces referential integrity, ensuring that child records always have a corresponding parent record.

Example : A DEPARTMENT table and an EMPLOYEE table. Each exist before an employee can be assigned to it. Key SQL concepts demonstrated PRIMARY KEY, FOREIGN KEY, ON DELETE CASCADE, INNER JOIN, LEFT JOIN, cascading deletes, DML operations on related tables.

Algorithm :—

1. create the DEPARTMENTS master table with DEPT-ID as primary key.
2. create the EMPLOYEES child table with DEPT-ID as foreign key referencing DEPARTMENTS.
3. Insert records into DEPARTMENTS first.
4. Insert records into EMPLOYEES refing valid DEPT-ID values.
5. Demonstrate INSERT isolation: try inserting an employee with invalid DEPT-ID.
6. Demonstrate DELETE restriction: try deleting a department that has employees.
7. perform INNER JOIN to display employees with department names.
8. perform LEFT JOIN to show departments even without employees.
9. Apply ON DELETE CASCADE: recreate with cascade and verify.
10. Display final results of all queries.

Date :

GATE Educational Institutions

Page No. : 32

Practical No. :

6. SQL Commands using Master-Child Tables

SQL CODE

-- Step 1: Create Master Table

```
CREATE TABLE DEPARTMENTS (  
    DEPT_ID NUMBER PRIMARY KEY,  
    DEPT_NAME VARCHAR2(40),  
    LOCATION VARCHAR2(40)  
);
```

-- Step 2: Create Child Table with Foreign Key

```
CREATE TABLE EMP_DETAIL (  
    EMP_ID NUMBER PRIMARY KEY,  
    EMP_NAME VARCHAR2(50),  
    SALARY NUMBER(10,2),  
    DEPT_ID NUMBER,  
    CONSTRAINT fk_dept FOREIGN KEY (DEPT_ID)  
        REFERENCES DEPARTMENTS(DEPT_ID)  
        ON DELETE CASCADE  
);
```

-- Step 3: Insert into Master Table

```
INSERT INTO DEPARTMENTS VALUES (10, 'Information Technology', 'Hyderabad');  
INSERT INTO DEPARTMENTS VALUES (20, 'Human Resources', 'Bangalore');  
INSERT INTO DEPARTMENTS VALUES (30, 'Finance', 'Mumbai');  
INSERT INTO DEPARTMENTS VALUES (40, 'Research & Dev', 'Pune');  
COMMIT;
```

-- Step 4: Insert into Child Table

```
INSERT INTO EMP_DETAIL VALUES (101, 'Ravi Kumar', 65000, 10);  
INSERT INTO EMP_DETAIL VALUES (102, 'Priya Sharma', 52000, 20);  
INSERT INTO EMP_DETAIL VALUES (103, 'Arjun Nair', 75000, 10);  
INSERT INTO EMP_DETAIL VALUES (104, 'Sneha Rao', 80000, 30);  
INSERT INTO EMP_DETAIL VALUES (105, 'Kiran Patel', 70000, 10);  
COMMIT;
```

-- Step 5: INNER JOIN - Employees with Department

```
SELECT E.EMP_ID, E.EMP_NAME, E.SALARY,  
       D.DEPT_NAME, D.LOCATION  
FROM EMP_DETAIL E  
INNER JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID  
ORDER BY E.EMP_ID;
```

-- Step 6: LEFT JOIN - All Departments (even without employees)

```
SELECT D.DEPT_ID, D.DEPT_NAME,  
       NVL(E.EMP_NAME, 'No Employees') AS EMP_NAME  
FROM DEPARTMENTS D  
LEFT JOIN EMP_DETAIL E ON D.DEPT_ID = E.DEPT_ID  
ORDER BY D.DEPT_ID;
```

-- Step 7: ON DELETE CASCADE Demo

```
DELETE FROM DEPARTMENTS WHERE DEPT_ID = 10;  
COMMIT;
```

SELECT * FROM EMP_DETAIL; -- Dept 10 employees auto-deleted

Date :

GATE Educational Institutions

Page No. : 33

OUTPUT

Practical No. :

-- INNER JOIN Result:

EMP_ID EMP_NAME SALARY DEPT_NAME LOCATION

101	Ravi Kumar	65000	Information Technology	Hyderabad
102	Priya Sharma	52000	Human Resources	Bangalore
103	Arjun Nair	75000	Information Technology	Hyderabad
104	Sneha Rao	80000	Finance	Mumbai
105	Kiran Patel	70000	Information Technology	Hyderabad

-- LEFT JOIN Result:

DEPT_ID DEPT_NAME EMP_NAME

10	Information Technology	Ravi Kumar
10	Information Technology	Arjun Nair
10	Information Technology	Kiran Patel
20	Human Resources	Priya Sharma
30	Finance	Sneha Rao
40	Research & Dev	No Employees

-- After DELETE CASCADE (DEPT 10 deleted):

EMP_ID EMP_NAME SALARY DEPT_ID

102	Priya Sharma	52000	20
104	Sneha Rao	80000	30



Date :

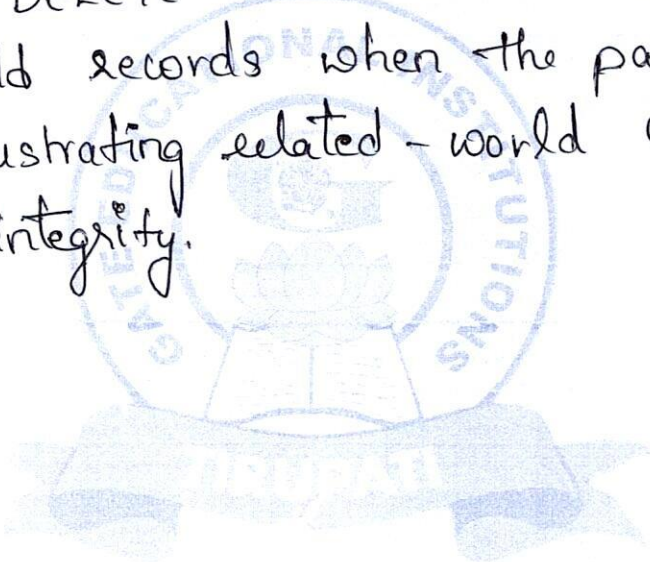
GATE Educational Institutions

Page No. : 36

Practical No. :

conclusion :—

Master-child table relationships were successfully demonstrated using DEPARTMENTS and EMP-DETAIL tables. SQL foreign key constraints enforced referential integrity. INNER JOIN and LEFT JOIN queries retrieved combined data across related tables. ON DELETE CASCADE automatically removed child records when the parent was deleted, illustrating related-world cascading referential integrity.



Date :	GATE Educational Institutions	Page No. : 35
	PL/SQL program to find factorial of a number	Practical No. : 7

Aim :—

To write a PL/SQL program that accepts a number from the user and computes its factorial using a loop construct, demonstrating basic PL/SQL programming with variables, loops and output.

Description :—

The factorial of a non-negative integer n is the product of all positive integers from 1 to n . By definition, $0! = 1$.

Formula : $n! = n \times (n-1) \times (n-2) \dots \times 2 \times 1$

Example : $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$ | $0! = 1$ | $1! = 1$

In PL/SQL, factorial is computed using a for loop or while loop. The result variable is initialized to 1 and multiplied iteratively.

The program also handles the special case of 0 and validates that negative numbers do not have a factorial.

Algorithm :-

1. start the PL/SQL block :
2. Declare Variables : $v-n$ (input) $v-fact$, $v-i$
3. Assign the input value to $v-n$.
4. check if $v-n$ is negative. If so print error and exit.
5. If $v-n = 0$, factorial is 1.
6. Use a FOR loop from 1 to $v-n$.
7. Inside the loop, multiply $v-fact$ by loop counter :
$$v-fact := v-fact * i$$
8. After the loop ends, print : 'factorial of $v-n$ is $v-fact$ '
9. End the PL/SQL block.

Date :

GATE Educational Institutions

Page No. : 38

7. PL/SQL Program to Find Factorial of a Number

Practical No. :

PL/SQL CODE

```
DECLARE
  v_n NUMBER := 6; -- Input: Change this value to test
  v_fact NUMBER := 1;
  i NUMBER;
BEGIN
  IF v_n < 0 THEN
    DBMS_OUTPUT.PUT_LINE('ERROR: Factorial is not defined for negative numbers.');
```

```
  ELSIF v_n = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Factorial of 0 = 1');
  ELSE
    FOR i IN 1..v_n LOOP
      v_fact := v_fact * i;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Factorial of ' || v_n || ' = ' || v_fact);
  END IF;
END;
/

-- Test with 0
DECLARE
  v_n NUMBER := 0;
  v_fact NUMBER := 1;
BEGIN
  IF v_n = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Factorial of 0 = 1 (Special Case)');
```

```
  ELSE
    FOR i IN 1..v_n LOOP
      v_fact := v_fact * i;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('Factorial of ' || v_n || ' = ' || v_fact);
  END IF;
END;
/
```


Date :

GATE Educational Institutions

Page No. : 38

Practical No. :

OUTPUT

Factorial of 6 = 720

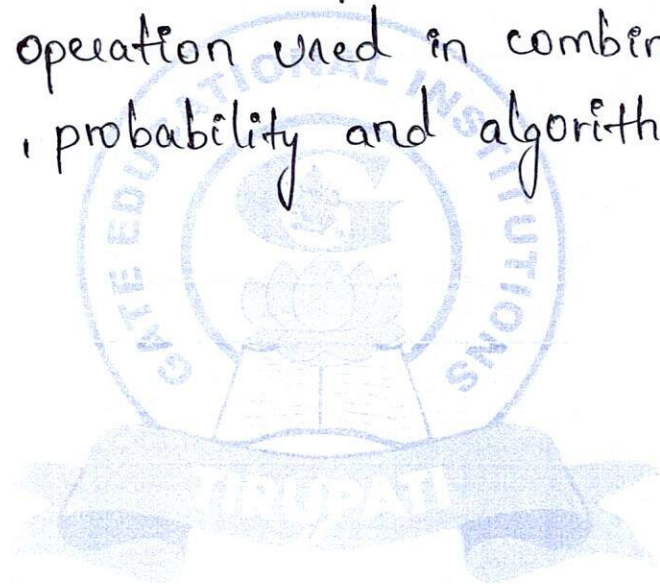
Factorial of 0 = 1 (Special Case)

PL/SQL procedure successfully completed.



conclusion :-

The PL/SQL program to compute the factorial of a number was successfully implemented using a for loop. The program handles all edge cases including $0! = 1$ and negative number validation. Factorial computation is a fundamental mathematical operation used in combinatorial permutations, probability and algorithm analysis.



Date :	GATE Educational Institutions	Page No. 40
	PL/SQL program to check whether is palindrome	Practical No. : 8.

Aim :-

To write a PL/SQL program that checks whether a given number is a palindrome - i.e; reads the same forwards and backwards.

Description :-

A palindrome number is a number that remains the same when its digits are reversed. eg; 121, 1331, 12321 are palindromes; while 123456 are not.

Logic : Extract the last digit of number using $\text{MOD}(n, 10)$. Build the reversed number by multiplying the current reversed value by 10 and adding the digit. Divide the original by 10.

Repeat until $n=0$ compare the reversed number with original.

Additional : The program also uses the built-in `TO_CHAR` and `REVERSE` function as an alternative string-based approach for verification.

Algorithm :—

1. Declare Variables : V_num , V_temp , V_rev , V_digit
2. Initialize V_rev to 0 and V_temp to V_num
3. Enter a WHILE loop : continue while $V_temp > 0$.
4. Inside the loop : extract last digit with MOD.
5. Build reversed number : $V_rev := V_rev * 10 + V_digit$.
6. Remove last digit : $V_temp := TRUNC(V_temp / 10)$
7. Exit loop when $V_temp = 0$
8. compare V_rev with V_num .
9. IF equal, print 'palindrome'; otherwise print 'not a palindrome'.

Date :

GATE Educational Institutions

Page No. : 42 ✓

Practical No. :

8. PL/SQL Program to Check Whether a Number is Palindrome

PL/SQL CODE

```
DECLARE
v_num NUMBER := 12321; -- Test number (change to test others)
v_temp NUMBER;
v_rev NUMBER := 0;
v_digit NUMBER;
BEGIN
v_temp := v_num;

-- Reverse the number
WHILE v_temp > 0 LOOP
v_digit := MOD(v_temp, 10);
v_rev := v_rev * 10 + v_digit;
v_temp := TRUNC(v_temp / 10);
END LOOP;

DBMS_OUTPUT.PUT_LINE('Original Number : ' || v_num);
DBMS_OUTPUT.PUT_LINE('Reversed Number : ' || v_rev);

-- Check palindrome
IF v_rev = v_num THEN
DBMS_OUTPUT.PUT_LINE(v_num || ' is a PALINDROME number. ');
ELSE
DBMS_OUTPUT.PUT_LINE(v_num || ' is NOT a PALINDROME number. ');
END IF;
END;
/

-- Test with non-palindrome
DECLARE
v_num NUMBER := 12345;
v_temp NUMBER;
v_rev NUMBER := 0;
v_digit NUMBER;
BEGIN
v_temp := v_num;
WHILE v_temp > 0 LOOP
v_digit := MOD(v_temp, 10);
v_rev := v_rev * 10 + v_digit;
v_temp := TRUNC(v_temp / 10);
END LOOP;
IF v_rev = v_num THEN
DBMS_OUTPUT.PUT_LINE(v_num || ' is a PALINDROME number. ');
ELSE
DBMS_OUTPUT.PUT_LINE(v_num || ' is NOT a PALINDROME number. ');
END IF;
END;
/
```

Date :

GATE Educational Institutions

Page No. : 43

Practical No. :

OUTPUT

Original Number : 12321
Reversed Number : 12321
12321 is a PALINDROME number.

12345 is NOT a PALINDROME number.

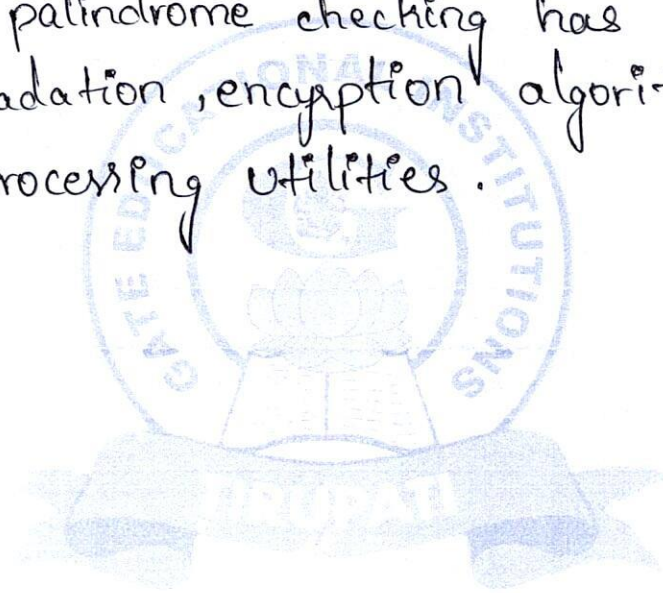
PL/SQL procedure successfully completed.



Date :	GATE Educational Institutions	Page No. : 64
		Practical No. :

conclusion :-

The PL/SQL program to check palindrome numbers was successfully implemented using a WHILE LOOP and digit extraction with MOD and TRUNC functions. The program correctly identified 12321 as a palindrome and 12345 as a non-palindrome. Palindrome checking has applications in data validation, encryption algorithms, and string processing utilities.



Aim :-

To demonstrate the various predefined exceptions available in Oracle PL/SQL by writing separate examples for each commonly used exception type.

Description :-

Oracle PL/SQL has a set of predefined exceptions that are automatically raised by the Oracle engine when specific runtime error occurs. These exceptions are pre-named and do not need to be declared. They can be directly referenced in the EXCEPTION section.

Common predefined Exceptions :

NO-DATA-FOUND - SELECT INTO returns no row

TOO-MANY-ROWS - SELECT INTO returns more than one row.

ZERO-DIVIDE - Division by zero attempt.

VALUE-ERROR - Arithmetic, conversion, or truncation error.

INVALID-CURSOR - Illegal cursor operation.

CURSOR-ALREADY-OPEN - Attempt to open an already open.

DUP-VAL-ON-INDEX - Duplicate value in unique index.

LOGIN-DENIED - Invalid user name / password.

Algorithm :-

1. create a test table SAMPLE - DATA with a unique constraint.
2. Insert test records.
3. Eg:-1 : Trigger NO - DATA - FOUND with SELECT for a missing row.
4. Eg:-2 : Trigger TOO - MANY - ROWS with SELECT INTO returning multiple rows.
5. Eg:-3 : Trigger ZERO - DIVIDE with an integer division by zero.
6. Eg:-4 : Trigger DUP - VAL - ON - INDEX with a duplicate INSERT.
7. Eg:-5 : Trigger VALUE - ERROR by assigning a large value to a small variable.
8. In each example, catch the exception and display a descriptive message.

Date :

GATE Educational Institutions

Page No. : 42

9. Demonstration of Predefined Exceptions in PL/SQL

Practical No. :

PL/SQL CODE

```

-- Setup
CREATE TABLE SAMPLE_DATA (
  S_ID  NUMBER PRIMARY KEY,
  S_NAME VARCHAR2(30)
);
INSERT INTO SAMPLE_DATA VALUES (1, 'Alpha');
INSERT INTO SAMPLE_DATA VALUES (2, 'Beta');
INSERT INTO SAMPLE_DATA VALUES (3, 'Gamma');
COMMIT;

-- 1. NO_DATA_FOUND
DECLARE
  v_name SAMPLE_DATA.S_NAME%TYPE;
BEGIN
  SELECT S_NAME INTO v_name FROM SAMPLE_DATA WHERE S_ID = 99;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('NO_DATA_FOUND] No record exists for S_ID = 99. ');
END;
/

-- 2. TOO_MANY_ROWS
DECLARE
  v_name SAMPLE_DATA.S_NAME%TYPE;
BEGIN
  SELECT S_NAME INTO v_name FROM SAMPLE_DATA; -- Multiple rows
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('TOO_MANY_ROWS] Query returned multiple rows unexpectedly. ');
END;
/

-- 3. ZERO_DIVIDE
DECLARE
  v_a NUMBER := 50;
  v_b NUMBER := 0;
  v_r NUMBER;
BEGIN
  v_r := v_a / v_b;
  DBMS_OUTPUT.PUT_LINE('Result: ' || v_r);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('ZERO_DIVIDE] Cannot divide ' || v_a || ' by zero. ');
END;
/

-- 4. DUP_VAL_ON_INDEX
BEGIN
  INSERT INTO SAMPLE_DATA VALUES (1, 'Delta'); -- S_ID=1 already exists
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE('DUP_VAL_ON_INDEX] Primary key 1 already exists. ');
END;
/

-- 5. VALUE_ERROR
DECLARE
  v_x VARCHAR2(3);
BEGIN
  v_x := 'ABCDEFGG'; -- Too large for VARCHAR2(3)
EXCEPTION
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE('VALUE_ERROR] Value is too large for the variable. ');
END;
/

```


Date :	GATE Educational Institutions	Page No. : 48
OUTPUT		Practical No. :

[NO_DATA_FOUND] No record exists for S_ID = 99.

[TOO_MANY_ROWS] Query returned multiple rows unexpectedly.

[ZERO_DIVIDE] Cannot divide 50 by zero.

[DUP_VAL_ON_INDEX] Primary key 1 already exists.

[VALUE_ERROR] Value is too large for the variable.

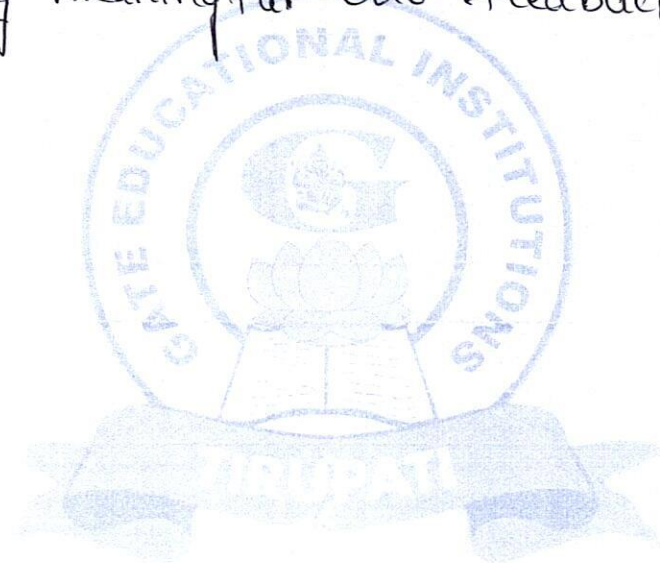
PL/SQL procedure successfully completed.



Date :	GATE Educational Institutions	Page No. : 49
		Practical No. :

conclusion :—

ALL five commonly used predefined PL/SQL Exceptions were successfully demonstrated with clear, executable examples. These built-in Exceptions allow PL/SQL programs to handle common runtime error gracefully preventing abnormal termination and providing meaningful error feedback to the User.



Date :	GATE Educational Institutions	Page No. : 50
	PL/SQL program using user-defined Exception for divide operation.	Practical No. : 10.

Aim :-

To write a PL/SQL program that performs a division operation and uses a user-defined Exception to handle division by zero along with additional business - logic validations.

Description :-

User-defined Exceptions allow PL/SQL programmers to define and raise their own application specific error conditions. Unlike predefined exceptions that are raised automatically by Oracle using the RAISE statement must be :

1) Declare in the DECLARE section, 2) Explicitly raised using RAISE statement 3) handled in the EXCEPTION section.

e - divide - by - zero - Raised when divisor is 0.

e - negative - input - Raised when any operand is negative.

e - overflow - Raised when result exceeds a defined limit.

Date :	GATE Educational Institutions	Page No. : 51
		Practical No. :

Algorithm :-

1. start the DECLARE Section.
2. Declare V-numerator and V-denominator as NUMBER Input Variables.
3. Declare V-result as NUMBER to store the output.
4. Declare three user-defined Exceptions : e-divide-by zero, e-negative-input, e-overflow.
5. In the BEGIN section, check if the denominator is 0, if yes, RAISE e-divide-by zero
6. check if either operand is negative ; if yes e-negative-input.
7. perform the division : $V\text{-result} = V\text{-numerator} / V\text{-denominator}$.
8. check if result exceeds limit ; if yes, RAISE e-overflow.
9. print result
10. In the EXCEPTION Section, handle each user-defined exception with WHEN THEN.

Date :

GATE Educational Institutions

Page No. : 52

Practical No. :

10. PL/SQL Program using User-Defined Exception for Divide Operation**PL/SQL CODE**

```

-- Full Program with User-Defined Exceptions
DECLARE
    v_numerator NUMBER := 100;
    v_denominator NUMBER := 0; -- Change to test different scenarios
    v_result NUMBER;

    -- User-defined exception declarations
    e_divide_by_zero EXCEPTION;
    e_negative_input EXCEPTION;
    e_overflow EXCEPTION;
    c_max_result CONSTANT NUMBER := 10000;

BEGIN
    -- Validation: Check for negative inputs
    IF v_numerator < 0 OR v_denominator < 0 THEN
        RAISE e_negative_input;
    END IF;

    -- Validation: Check for division by zero
    IF v_denominator = 0 THEN
        RAISE e_divide_by_zero;
    END IF;

    -- Perform Division
    v_result := v_numerator / v_denominator;

    -- Validation: Check overflow
    IF v_result > c_max_result THEN
        RAISE e_overflow;
    END IF;

    DBMS_OUTPUT.PUT_LINE('Division Result: ' || v_numerator ||
        '/' || v_denominator || '=' || v_result);
EXCEPTION
    WHEN e_divide_by_zero THEN
        DBMS_OUTPUT.PUT_LINE('USER EXCEPTION: Division by zero is not allowed!');
        DBMS_OUTPUT.PUT_LINE('Denominator must be a non-zero value.');

```

Date :

GATE Educational Institutions

Page No. : 53

OUTPUT

Practical No. :

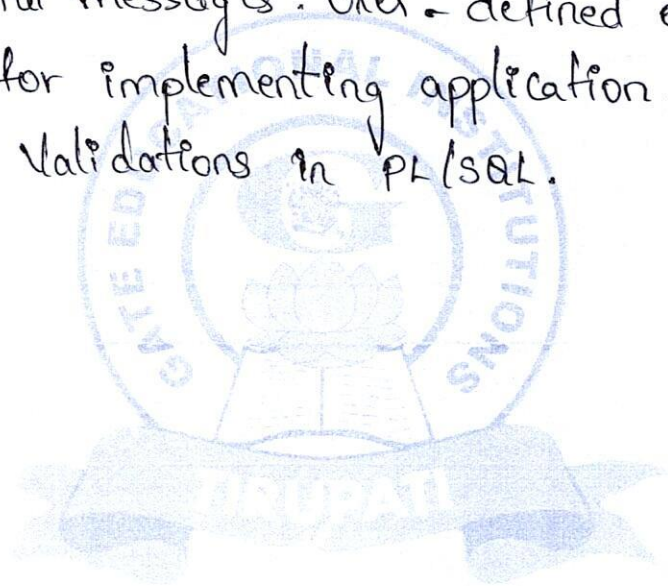
```
-- Test 1: v_numerator=100, v_denominator=0
USER EXCEPTION: Division by zero is not allowed!
Denominator must be a non-zero value.
-- Test 2: v_numerator=100, v_denominator=4
Division Result: 100 / 4 = 25
-- Test 3: v_numerator=-50, v_denominator=5
USER EXCEPTION: Negative inputs are not accepted!
Both numerator and denominator must be positive.
PL/SQL procedure successfully completed.
```



Date :	GATE Educational Institutions	Page No. : 84
		Practical No. :

conclusion :-

A PL/SQL program using user-defined exceptions for a division operation was successfully implemented. The program declared three custom exception, raised them explicitly using RAISE when corresponding conditions were met, and handled them in the EXCEPTION section with meaningful messages. User-defined exceptions are essential for implementing application-specific business rule validations in PL/SQL.



Date :	GATE Educational Institutions	Page No. : 55
	PL/SQL program for library book issue system with Exception handling.	Practical No. : 11

Aim :-

To develop a PL/SQL program simulating a library book issue system that uses cursors to process book issue requests and incorporates comprehensive exception handling for cases like book unavailability, member not found, and already issued books.

Description :-

A library management system is a classic real-world application of database programming involves :
 Verifying that the member exists, checking that the book is available, recording the issue transaction, and updating the book availability status. This program uses : EXPLICIT CURSOR to process pending issue requests, predefined exceptions and user-defined exceptions.

Date :	GATE Educational Institutions	Page No. : 56
		Practical No. :

Algorithm :-

1. Create LIBRARY_MEMBERS table.
2. Create LIBRARY_books table.
3. Create BOOK_ISSUES table.
4. Insert sample members and books.
5. create a PL/SQL block with a cursor on pending issue requests.
6. for each issue request, use SELECT INTO to fetch member and book details.
7. Handle NO-DATA-FOUND for non-existent member (or) book.
8. check available copies : if 0, raise e-book-not-available.
9. check if member already has the book issued : raise 'e-already-issued'.
10. check member issue limit : raise e-max-limit if exceeded.
11. if all checks pass : INSERT into Book-ISSUES and UPDATE book stock.
12. print success or error message for each transaction.

Date :

GATE Educational Institutions

Page No. 57

Practical No. :

11. PL/SQL Program for Library Book Issue System with Exception Handling**PL/SQL CODE**

```

-- Table Definitions
CREATE TABLE LIB_MEMBERS (
  MEM_ID  NUMBER PRIMARY KEY,
  MEM_NAME VARCHAR2(50),
  ISSUED_CNT NUMBER DEFAULT 0
);

CREATE TABLE LIB_BOOKS (
  BOOK_ID  NUMBER PRIMARY KEY,
  TITLE    VARCHAR2(100),
  AUTHOR   VARCHAR2(50),
  AVAIL_COPY NUMBER
);

CREATE TABLE BOOK_ISSUES (
  ISSUE_ID  NUMBER PRIMARY KEY,
  MEM_ID    NUMBER,
  BOOK_ID   NUMBER,
  ISSUE_DATE DATE DEFAULT SYSDATE,
  DUE_DATE  DATE DEFAULT SYSDATE + 14
);

-- Issue Requests Table (simulates incoming requests)
CREATE TABLE ISSUE_REQUESTS (
  REQ_ID NUMBER PRIMARY KEY,
  MEM_ID NUMBER,
  BOOK_ID NUMBER
);

CREATE SEQUENCE iss_seq START WITH 101 INCREMENT BY 1;

-- Insert Sample Data
INSERT INTO LIB_MEMBERS VALUES (1, 'Ravi Kumar', 0);
INSERT INTO LIB_MEMBERS VALUES (2, 'Priya Devi', 0);
INSERT INTO LIB_MEMBERS VALUES (3, 'Arjun Singh', 3); -- Already at limit

INSERT INTO LIB_BOOKS VALUES (201, 'Database Systems', 'Korth', 2);
INSERT INTO LIB_BOOKS VALUES (202, 'Operating Systems', 'Silberschatz', 0); -- No copies
INSERT INTO LIB_BOOKS VALUES (203, 'Data Structures', 'Cormen', 5);

INSERT INTO ISSUE_REQUESTS VALUES (1, 1, 201); -- Valid request
INSERT INTO ISSUE_REQUESTS VALUES (2, 2, 202); -- Book unavailable
INSERT INTO ISSUE_REQUESTS VALUES (3, 3, 203); -- Member at max limit
INSERT INTO ISSUE_REQUESTS VALUES (4, 9, 201); -- Invalid member
COMMIT;

-- Main PL/SQL Block
DECLARE
  v_mem_id LIB_MEMBERS.MEM_ID%TYPE;
  v_book_id LIB_BOOKS.BOOK_ID%TYPE;
  v_mem_name LIB_MEMBERS.MEM_NAME%TYPE;
  v_title LIB_BOOKS.TITLE%TYPE;
  v_avail LIB_BOOKS.AVAIL_COPY%TYPE;
  v_cnt LIB_MEMBERS.ISSUED_CNT%TYPE;

  e_book_not_available EXCEPTION;
  e_max_limit EXCEPTION;

  CURSOR c_req IS
    SELECT REQ_ID, MEM_ID, BOOK_ID FROM ISSUE_REQUESTS;

```


Date :

GATE Educational Institutions

Page No. 58

Practical No. :

v_req_id NUMBER;
BEGIN

```

OPEN c_req;
DBMS_OUTPUT.PUT_LINE('===== LIBRARY BOOK ISSUE PROCESSING =====');
LOOP
  FETCH c_req INTO v_req_id, v_mem_id, v_book_id;
  EXIT WHEN c_req%NOTFOUND;
  BEGIN
    -- Fetch Member
    SELECT MEM_NAME, ISSUED_CNT
    INTO v_mem_name, v_cnt
    FROM LIB_MEMBERS WHERE MEM_ID = v_mem_id;

    -- Fetch Book
    SELECT TITLE, AVAIL_COPY
    INTO v_title, v_avail
    FROM LIB_BOOKS WHERE BOOK_ID = v_book_id;

    IF v_avail = 0 THEN
      RAISE e_book_not_available;
    END IF;
    IF v_cnt >= 3 THEN
      RAISE e_max_limit;
    END IF;

    -- Issue the book
    INSERT INTO BOOK_ISSUES(ISSUE_ID, MEM_ID, BOOK_ID)
    VALUES (iss_seq.NEXTVAL, v_mem_id, v_book_id);

    UPDATE LIB_BOOKS SET AVAIL_COPY = AVAIL_COPY - 1
    WHERE BOOK_ID = v_book_id;

    UPDATE LIB_MEMBERS SET ISSUED_CNT = ISSUED_CNT + 1
    WHERE MEM_ID = v_mem_id;

    DBMS_OUTPUT.PUT_LINE('[SUCCESS] Book "' || v_title ||
      '" issued to ' || v_mem_name);
    COMMIT;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('[ERROR] Req# ' || v_req_id ||
        ': Member or Book not found. ');
    WHEN e_book_not_available THEN
      DBMS_OUTPUT.PUT_LINE('[ERROR] Req# ' || v_req_id ||
        ': Book "' || v_title || '" not available. ');
    WHEN e_max_limit THEN
      DBMS_OUTPUT.PUT_LINE('[ERROR] Req# ' || v_req_id ||
        ': ' || v_mem_name || ' has reached maximum issue limit (3). ');
  END;
END LOOP;
CLOSE c_req;
DBMS_OUTPUT.PUT_LINE('===== PROCESSING COMPLETE =====');
END;
/

```

Date :

GATE Educational Institutions

Page No. : 55

Practical No. :

OUTPUT

===== LIBRARY BOOK ISSUE PROCESSING =====
[SUCCESS] Book "Database Systems" issued to Ravi Kumar
[ERROR] Req#2: Book "Operating Systems" not available.
[ERROR] Req#3: Arjun Singh has reached maximum issue limit (3).
[ERROR] Req#4: Member or Book not found.
===== PROCESSING COMPLETE =====

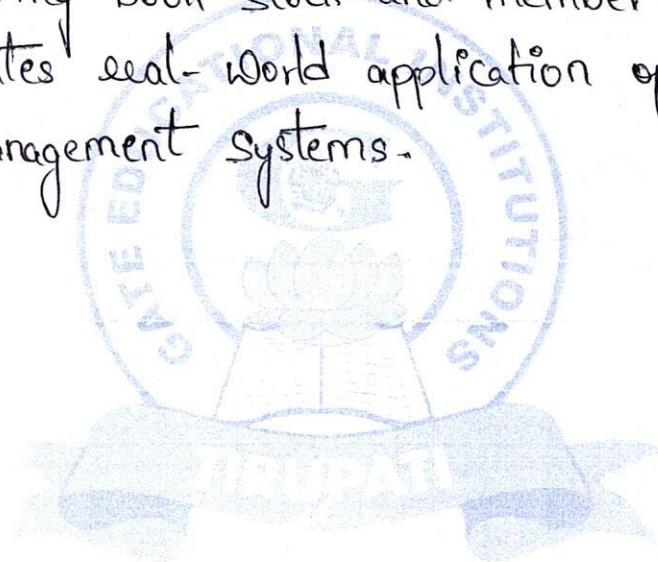
PL/SQL procedure successfully completed.



Date :	GATE Educational Institutions	Page No. : 60
		Practical No. :

conclusion :-

A comprehensive library book issue system was successfully implemented in PL/SQL. The program used as explicit cursor to process multiple issue requests, combined predefined and user-defined exceptions for robust error handling and performed transactional updates including book stock and member issue count. This demonstrates real-world application of PL/SQL in library management systems.



Date :	GATE Educational Institutions	Page No. : 61
	practical usage of pre-defined oracle package in PL/SQL.	Practical No. : 12

Aim :-

To demonstrate the practical usage of commonly used predefined oracle PL/SQL packages : DBMS - OUTPUT, DBMS - DATE, UTL - FILE and DBMS RANDOM with examples.

Description :-

Oracle provides a rich library of built-in packages that extend PL/SQL's functionality. These packages contain procedures, functions, and variables that can be called from any PL/SQL block.

Key predefined packages covered :

DBMS - OUTPUT - Displays output from PL/SQL blocks to screens.

DBMS - RANDOM - Generates random numbers and strings.

DBMS - UTILITY - provides miscellaneous utility procedure.

UTL - FILE - Enables file I/O operations.

TO - CHAR / TO - DATE - Date / number formatting functions.

Date :	GATE Educational Institutions	Page No. : 62 ✓
		Practical No. :

Algorithm :-

1. Demonstrate DBMS - OUTPUT package : PUT, PUTLINE, NEW-LINE ENABLE.
2. Demonstrate DBMS - RANDOM : VALUE (Float), VALUE (low, high) (integer range), STRING
3. Demonstrate data functions with TO-CHAR, SYSDATE, ADD-MONTHS, MONTHS - BETWEEN.
4. Demonstrate DBMS - UTILITY. FORMAT - CALL - STACT to view execution content.
5. show DBMS - UTILITY. GET-TIME for performance measurement.
6. Execute each package example and display result.

Date :

GATE Educational Institutions

Page No. : 63

Practical No. :

12. Practical Usage of Predefined Oracle Packages in PL/SQL**PL/SQL CODE**

```

-- 1. DBMS_OUTPUT Package
BEGIN
  DBMS_OUTPUT.ENABLE(1000000); -- Enable buffer (1 MB)
  DBMS_OUTPUT.PUT_LINE('=== DBMS_OUTPUT DEMO ===');
  DBMS_OUTPUT.PUT('Hello '); -- No newline
  DBMS_OUTPUT.PUT_LINE('World!');
  DBMS_OUTPUT.NEW_LINE; -- Blank line
  DBMS_OUTPUT.PUT_LINE('Buffer size: 1,000,000 bytes');
END;
/

-- 2. DBMS_RANDOM Package
DECLARE
  v_float NUMBER;
  v_int NUMBER;
  v_str VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('=== DBMS_RANDOM DEMO ===');

  -- Random float between 0 and 1
  v_float := DBMS_RANDOM.VALUE;
  DBMS_OUTPUT.PUT_LINE('Random Float (0-1) : ' || ROUND(v_float, 6));

  -- Random integer between 1 and 100
  v_int := TRUNC(DBMS_RANDOM.VALUE(1, 100));
  DBMS_OUTPUT.PUT_LINE('Random Integer (1-100): ' || v_int);

  -- Random uppercase string of length 8
  v_str := DBMS_RANDOM.STRING('U', 8);
  DBMS_OUTPUT.PUT_LINE('Random String (U,8) : ' || v_str);

  -- Random lowercase string of length 6
  v_str := DBMS_RANDOM.STRING('L', 6);
  DBMS_OUTPUT.PUT_LINE('Random String (L,6) : ' || v_str);

  DBMS_RANDOM.TERMINATE;
END;
/

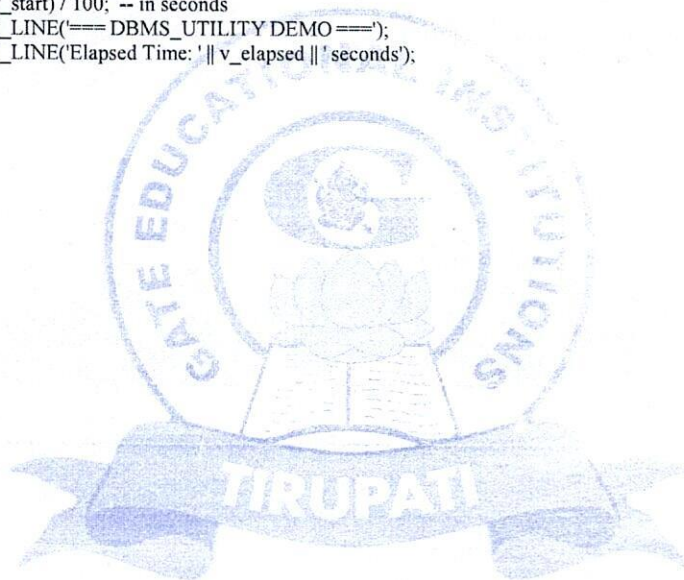
-- 3. Date Functions and SYSDATE
DECLARE
  v_today DATE := SYSDATE;
  v_future DATE;
  v_months NUMBER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('=== DATE FUNCTION DEMO ===');
  DBMS_OUTPUT.PUT_LINE('Today : ' ||
    TO_CHAR(v_today, 'DD-MON-YYYY HH24:MI:SS'));
  DBMS_OUTPUT.PUT_LINE('Day of Week : ' ||
    TO_CHAR(v_today, 'DAY'));
  DBMS_OUTPUT.PUT_LINE('Day of Year : ' ||
    TO_CHAR(v_today, 'DDD'));

  -- Add 3 months
  v_future := ADD_MONTHS(v_today, 3);
  DBMS_OUTPUT.PUT_LINE('+3 Months : ' ||
    TO_CHAR(v_future, 'DD-MON-YYYY'));

  -- Months between two dates
  v_months := MONTHS_BETWEEN(v_future, v_today);

```


Date :	GATE Educational Institutions	Page No. : 64
DBMS_OUTPUT.PUT_LINE('Months Between : ' v_months);		Practical No. :
<pre> -- Last day of month DBMS_OUTPUT.PUT_LINE('Last of Month : ' TO_CHAR(LAST_DAY(v_today), 'DD-MON-YYYY')); END; / -- 4. DBMS_UTILITY – Performance Timing DECLARE v_start NUMBER; v_end NUMBER; v_elapsed NUMBER; BEGIN v_start := DBMS_UTILITY.GET_TIME; -- Simulate workload FOR i IN 1..100000 LOOP NULL; END LOOP; v_end := DBMS_UTILITY.GET_TIME; v_elapsed := (v_end - v_start) / 100; -- in seconds DBMS_OUTPUT.PUT_LINE('=== DBMS_UTILITY DEMO ==='); DBMS_OUTPUT.PUT_LINE('Elapsed Time: ' v_elapsed ' seconds'); END; / </pre>		



Date :	GATE Educational Institutions	Page No. : 68
		Practical No. :

OUTPUT

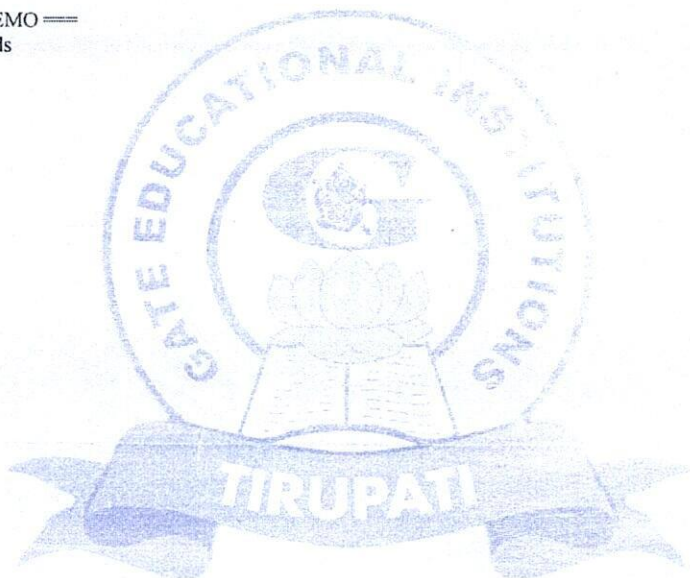
=== DBMS_OUTPUT DEMO ===
Hello World!

Buffer size: 1,000,000 bytes

=== DBMS_RANDOM DEMO ===
Random Float (0-1) : 0.748213
Random Integer (1-100): 63
Random String (U,8) : KXRPMNBQ
Random String (L,6) : fvzjqr

=== DATE FUNCTION DEMO ===
Today : 01-JAN-2025 10:30:45
Day of Week : WEDNESDAY
Day of Year : 001
+3 Months : 01-APR-2025
Months Between : 3
Last of Month : 31-JAN-2025

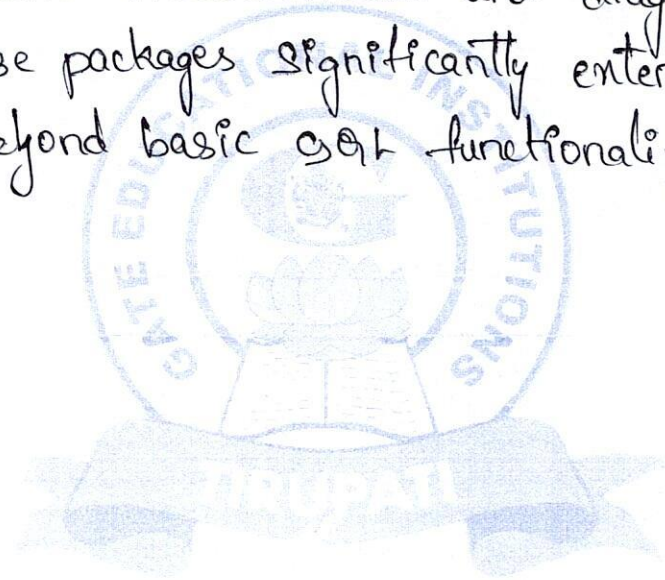
=== DBMS_UTILITY DEMO ===
Elapsed Time: 0.02 seconds



Date :	GATE Educational Institutions	Page No. : 66
		Practical No. :

conclusion :-

preddefined Oracle pl/sql packages were Successfully demonstrated. DBMS - OUTPUT provides screen I/O capabilities ; DBMS - RANDOM generates random values for testing and simulations ; Date built-in-functions support comprehensive data arithmetic ; DBMS - UTILITY offers performance measurement and diagnostic utilities. These packages significantly extend pl/sql capabilities beyond basic SQL functionality.



Date :	GATE Educational Institutions	Page No. : 67
	create and manage object types, Tables, and methods using Oracle SQL.	Practical No. : 13

Aim :—

To create and manage Oracle object types, object tables, and member methods in Oracle SQL to demonstrate the object-Relational feature of Oracle database.

Description :—

Oracle supports object-relational programming through user-defined data types called object types. An object type is a Schema Object that encapsulates a data structure and behaviour into a single unit, similar to classes in object oriented programming.

Key concepts :

object type - Template defining attributes and methods.

Object table - A table where each row is an instance of an object type.

member methods - functions/procedures defined inside an object type body.

constructors - Automatically generated by Oracle to create object instances.

SELF - keyword inside methods referring to the current object instance.

Algorithm :-

1. create an object type STUDENT-TYPE with attributes.
2. Define a member function GET-GRADE in the type specification.
3. create the object type body to implement the GET-GRADE function logic.
4. create an object table STUDENT-OBJ-TABLE of type STUDENT-TYPE.
5. Query the object table to retrieve all attribute values.
6. Insert student object table instances using the constructor STUDENT-TYPE.
7. call the member method GET-GRADE using dot notation.
8. create a second type ADDRESS-TYPE for demonstrating type composition.
9. create an EMPLOYEE-OBJ type for demonstrating type composition.
10. Insert and query nested object instances.

Date :

GATE Educational Institutions

Page No. : 69

Practical No. :

13. Create and Manage Object Types, Tables, and Methods using Oracle SQL

SQL / PL-SQL CODE

```
-- Step 1: Create Object Type Specification
CREATE OR REPLACE TYPE STUDENT_TYPE AS OBJECT (
    ROLL_NO NUMBER,
    STU_NAME VARCHAR2(50),
    MARKS NUMBER(5,2),
    -- Member Function Declaration
    MEMBER FUNCTION GET_GRADE RETURN VARCHAR2,
    MEMBER FUNCTION GET_DETAILS RETURN VARCHAR2
);
/

-- Step 2: Create Object Type Body (Implementation)
CREATE OR REPLACE TYPE BODY STUDENT_TYPE AS

    MEMBER FUNCTION GET_GRADE RETURN VARCHAR2 IS
        v_grade VARCHAR2(5);
    BEGIN
        IF SELF.MARKS >= 90 THEN v_grade := 'O';
        ELSIF SELF.MARKS >= 75 THEN v_grade := 'A+';
        ELSIF SELF.MARKS >= 60 THEN v_grade := 'A';
        ELSIF SELF.MARKS >= 50 THEN v_grade := 'B';
        ELSIF SELF.MARKS >= 40 THEN v_grade := 'C';
        ELSE v_grade := 'F';
        END IF;
        RETURN v_grade;
    END GET_GRADE;

    MEMBER FUNCTION GET_DETAILS RETURN VARCHAR2 IS
    BEGIN
        RETURN 'Roll: ' || SELF.ROLL_NO ||
            ' | Name: ' || SELF.STU_NAME ||
            ' | Marks: ' || SELF.MARKS ||
            ' | Grade: ' || SELF.GET_GRADE();
    END GET_DETAILS;

END;
/

-- Step 3: Create Object Table
CREATE TABLE STUDENT_OBJ_TABLE OF STUDENT_TYPE (
    PRIMARY KEY (ROLL_NO)
);

-- Step 4: Insert Object Instances
INSERT INTO STUDENT_OBJ_TABLE VALUES (STUDENT_TYPE(101, 'Arjun Reddy', 92.5));
INSERT INTO STUDENT_OBJ_TABLE VALUES (STUDENT_TYPE(102, 'Priya Devi', 78.0));
INSERT INTO STUDENT_OBJ_TABLE VALUES (STUDENT_TYPE(103, 'Kiran Babu', 55.5));
INSERT INTO STUDENT_OBJ_TABLE VALUES (STUDENT_TYPE(104, 'Divya Nair', 38.0));
COMMIT;

-- Step 5: Query Object Table
SELECT S.ROLL_NO, S.STU_NAME, S.MARKS, S.GET_GRADE() AS GRADE
FROM STUDENT_OBJ_TABLE S
ORDER BY S.ROLL_NO;

-- Step 6: Call Member Method GET_DETAILS
DECLARE
    v_stu STUDENT_TYPE;
BEGIN
    SELECT VALUE(S) INTO v_stu
```


Date :

GATE Educational Institutions

Page No. : 20

Practical No. :

```
FROM STUDENT_OBJ_TABLES  
WHERE S.ROLL_NO = 101;
```

```
DBMS_OUTPUT.PUT_LINE(v_stu.GET_DETAILS());  
END;  
/
```

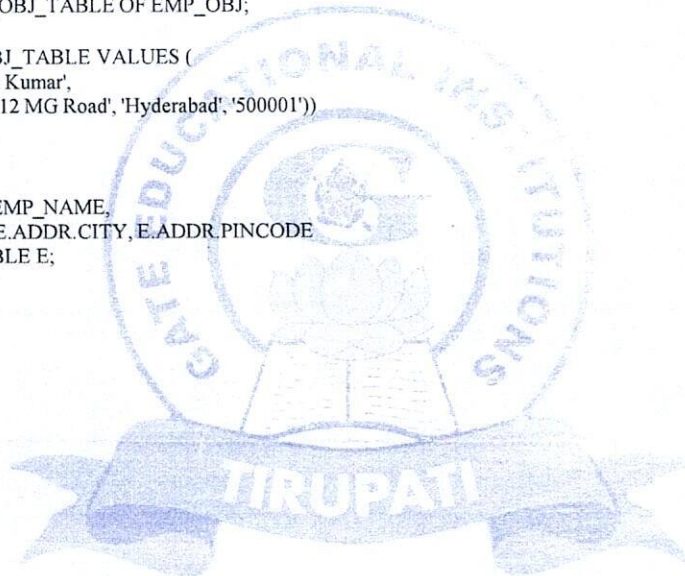
```
-- Step 7: Nested Object Type - ADDRESS_TYPE  
CREATE OR REPLACE TYPE ADDRESS_TYPE AS OBJECT (  
    STREET VARCHAR2(50),  
    CITY VARCHAR2(30),  
    PINCODE VARCHAR2(10)  
);  
/
```

```
CREATE OR REPLACE TYPE EMP_OBJ AS OBJECT (  
    EMP_ID NUMBER,  
    EMP_NAME VARCHAR2(50),  
    ADDR ADDRESS_TYPE -- Nested Object  
);  
/
```

```
CREATE TABLE EMP_OBJ_TABLE OF EMP_OBJ;
```

```
INSERT INTO EMP_OBJ_TABLE VALUES (  
    EMP_OBJ(1001, 'Ravi Kumar',  
        ADDRESS_TYPE('12 MG Road', 'Hyderabad', '500001'))  
);  
COMMIT;
```

```
SELECT E.EMP_ID, E.EMP_NAME,  
    E.ADDR.STREET, E.ADDR.CITY, E.ADDR.PINCODE  
FROM EMP_OBJ_TABLE E;
```



Date :	GATE Educational Institutions	Page No. : 21
		Practical No. :

OUTPUT

Object type STUDENT_TYPE created.
Object type body STUDENT_TYPE created.
Table STUDENT_OBJ_TABLE created.
4 rows inserted.

ROLL_NO	STU_NAME	MARKS	GRADE
101	Arjun Reddy	92.5	O
102	Priya Devi	78.0	A+
103	Kiran Babu	55.5	B
104	Divya Nair	38.0	F

Roll: 101 | Name: Arjun Reddy | Marks: 92.5 | Grade: O

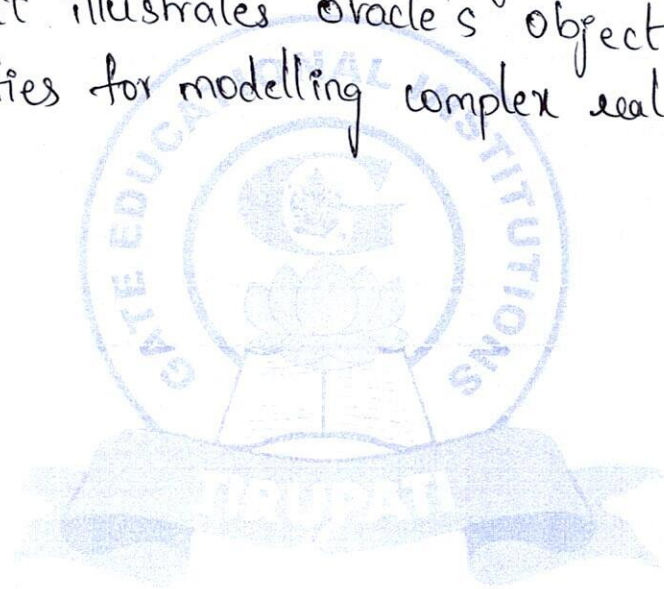
EMP_ID	EMP_NAME	STREET	CITY	PINCODE
1001	Ravi Kumar	12 MG Road	Hyderabad	500001



Date :	GATE Educational Institutions	Page No. : <i>27</i>
		Practical No. :

conclusion :-

Oracle object type were Successfully created , implemented and queried. The STUDENT-TYPE object type encapsulated attributes and methods in a single unit. Object tables stored typed object instances. Nested object types demonstrated type composition. This experiment illustrates Oracle's object-relational model capabilities for modelling complex real-world entities.



Date :	GATE Educational Institutions	Page No. : 73
	Selection, projection and Sorting queries on a simple Table [Case study - I].	Practical No. : 14

Aim :—

To perform and demonstrate fundamental SQL operations Selection projection and Sorting (Ordering results - on a simple relational table representing a student database.

Description :—

Three fundamental relational algebra operations are directly mapped to SQL.

1. SELECTION - Retrieves specific rows that satisfy a condition. Reduces the number of rows in the result.
2. PROJECTION - Retrieves specific rows columns from a table. Reduces the number of columns in the result.
3. SORTING - Arranges result rows in ascending (or) descending order.

These operations are the building blocks of all SQL queries and form the foundation of relational algebra applied to database management.

Case study : A STUDENT table is created with RollNo, Name, Branch, City and CGPA. Various combinations of selection, projection and sorting are performed.

Date :	GATE Educational Institutions	Page No. : 24
		Practical No. :

Algorithm :-

1. create the STUDENT table with appropriate columns and datatypes.
2. Insert at least 10 student records covering different branches and cities.
3. perform PROJECTION : select only specific columns.
4. perform SELECTION : select rows where BRANCH='cse'
5. combine selection and projection : select name and CGPA.
6. perform SORTING : ORDER BY CGPA ASC and CGPA DESC.
7. combine all three : project specific columns , apply WHERE filter and sort.
8. Demonstrate DISTINCT projection to eliminate duplicate values.
9. use BETWEEN , IN , LIKE , ISNULL in selection conditions.
10. Display and verify all results.

Date :

GATE Educational Institutions

Page No. : 26

Practical No. :

Case Study-1

14. Selection, Projection, and Sorting Queries on a Simple Table

SQL CODE

-- Create Table

```
CREATE TABLE STUDENT (  
  ROLL_NO NUMBER PRIMARY KEY,  
  S_NAME VARCHAR2(50),  
  BRANCH VARCHAR2(20),  
  CITY VARCHAR2(30),  
  CGPA NUMBER(3,1)  
);
```

-- Insert Records

```
INSERT INTO STUDENT VALUES (1, 'Arjun Reddy', 'CSE', 'Hyderabad', 9.2);  
INSERT INTO STUDENT VALUES (2, 'Priya Sharma', 'ECE', 'Bangalore', 8.5);  
INSERT INTO STUDENT VALUES (3, 'Kiran Babu', 'CSE', 'Chennai', 7.8);  
INSERT INTO STUDENT VALUES (4, 'Divya Nair', 'IT', 'Hyderabad', 8.1);  
INSERT INTO STUDENT VALUES (5, 'Suresh Goud', 'CSE', 'Mumbai', 6.5);  
INSERT INTO STUDENT VALUES (6, 'Meena Kumari', 'ECE', 'Pune', 9.0);  
INSERT INTO STUDENT VALUES (7, 'Vikram Singh', 'IT', 'Delhi', 7.2);  
INSERT INTO STUDENT VALUES (8, 'Anjali Patel', 'CSE', 'Hyderabad', 8.9);  
INSERT INTO STUDENT VALUES (9, 'Ravi Kumar', 'MECH', 'Chennai', 7.5);  
INSERT INTO STUDENT VALUES (10, 'Deepa Menon', 'IT', 'Bangalore', 9.1);  
COMMIT;
```

-- 1. PROJECTION: Select only Name and CGPA
SELECT S_NAME, CGPA FROM STUDENT;

-- 2. SELECTION: Filter CSE students
SELECT * FROM STUDENT WHERE BRANCH = 'CSE';

-- 3. Selection + Projection: Name & CGPA where CGPA > 8
SELECT S_NAME, BRANCH, CGPA
FROM STUDENT
WHERE CGPA > 8.0;

-- 4. SORTING: Order by CGPA Descending
SELECT ROLL_NO, S_NAME, BRANCH, CGPA
FROM STUDENT
ORDER BY CGPA DESC;

-- 5. Selection + Projection + Sorting (Combined)
SELECT ROLL_NO, S_NAME, CITY, CGPA
FROM STUDENT
WHERE BRANCH IN ('CSE', 'IT')
ORDER BY CGPA DESC;

-- 6. DISTINCT – Unique cities
SELECT DISTINCT CITY FROM STUDENT ORDER BY CITY;

-- 7. BETWEEN – Students with CGPA between 7.5 and 8.5
SELECT S_NAME, CGPA FROM STUDENT
WHERE CGPA BETWEEN 7.5 AND 8.5;

-- 8. LIKE – Students whose name starts with 'A'
SELECT S_NAME, BRANCH FROM STUDENT
WHERE S_NAME LIKE 'A%';

Date :

GATE Educational Institutions

Page No. : 86

OUTPUT

Practical No. :

-- 3. Selection + Projection (CGPA > 8).

S_NAME BRANCH CGPA

Arjun Reddy CSE 9.2
Priya Sharma ECE 8.5
Divya Nair IT 8.1
Meena Kumari ECE 9.0
Anjali Patel CSE 8.9
Deepa Menon IT 9.1

-- 4. Sorting by CGPA DESC:

ROLL_NO S_NAME BRANCH CGPA

1 Arjun Reddy CSE 9.2
10 Deepa Menon IT 9.1
6 Meena Kumari ECE 9.0
8 Anjali Patel CSE 8.9
...

-- 6. DISTINCT Cities:

CITY

Bangalore
Chennai
Delhi
Hyderabad
Mumbai
Pune



Date :

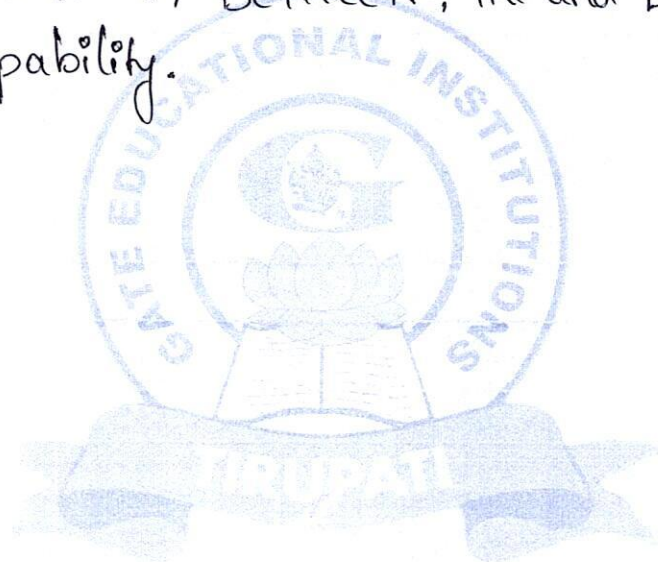
GATE Educational Institutions

Page No. : 77

Practical No. :

conclusion :-

The three fundamental SQL operations - selection, projection and sorting - were successfully demonstrated on the STUDENT table. These operations correspond directly to relational Algebra operations and form the core building blocks of all SQL queries. Additional clause like DISTINCT, BETWEEN, IN and LIKE extended the querying capability.



Aim :-

To design an entity - relationship . Diagram for a company database representing employee, Departments and projects and to implement the schema is SQL with appropriate constraints .

Description :-

An Entity - relationship diagram is a graphical representation of the logical structure of a database . It is used in the database design phase to model real-world entities , their attributes and the relationship between them before implementation .

ER diagram Symbols :

Rectangle - Entity .

Ellipse - Attribute

Double Ellipse - Multi valued attribute

Dashed Ellipse - Derived attribute

Diamond - relationship

Double rectangle - weak entity .

Line - connecting entities and relationships .

company database ER entities :

EMPLOYEE - EMP-ID, EMP-NAME, DOB, SALARY,
DEPT-NO. DEPARTMENT- DEPT-NO, DEPT-NAME,
LOCATION, MGR - EMP-ID. PROJECT - PROJ-ID, PROJ-
NAME, BUDGET, DEPT-NO. WORKS-ON - EMP-ID + PROJ-
ID, HOURS

Date :	GATE Educational Institutions	Page No. : 79
		Practical No. :

DEPENDENT - weak entity with EMP-ID, DEP-NAME, RELATION.

ER Relationships :

EMPLOYEE WORKS - IN DEPARTMENT

DEPARTMENT MANAGES EMPLOYEE

EMPLOYEE WORKS - ON PROJECT

EMPLOYEE HAS DEPENDENT

DEPARTMENT CONTROLS PROJECT.

Algorithm :-

1. Identify entities : EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT
2. Identify attributes : for each entity and mark prime keys.
3. Identify relationships : WORKS - IN, MANAGES, WORKS - ON, CONTROLS, HAS.
4. Determine cardinality (1:1, 1:N, M:N) for each relationship.
5. Identify weak entities and their identifying relationship.
6. Translate ER diagram to relational schema.
7. create SQL table with appropriate primary keys and foreign keys.
8. create WORKS - ON junction table to resolve M:N relationship.
9. Insert sample data and verify with JOIN queries.
10. write a query to display each employee's department and projects.

Date :

GATE Educational Institutions

Page No. : 80

Practical No. :

Case Study-2

15. ER Diagram for a Company Database

SQL CODE

-- ER Diagram Translation to SQL Tables

-- 1. DEPARTMENT Table (Created first for FK reference)

```
CREATE TABLE COMPANY_DEPT (  
    DEPT_NO NUMBER PRIMARY KEY,  
    DEPT_NAME VARCHAR2(40) NOT NULL,  
    LOCATION VARCHAR2(30),  
    MGR_ID NUMBER -- FK to COMPANY_EMP (added after)  
);
```

-- 2. EMPLOYEE Table

```
CREATE TABLE COMPANY_EMP (  
    EMP_ID NUMBER PRIMARY KEY,  
    EMP_NAME VARCHAR2(50) NOT NULL,  
    DOB DATE,  
    SALARY NUMBER(10,2),  
    DEPT_NO NUMBER,  
    CONSTRAINT fk_emp_dept FOREIGN KEY (DEPT_NO)  
        REFERENCES COMPANY_DEPT(DEPT_NO)  
);
```

-- Add Manager FK to DEPARTMENT

```
ALTER TABLE COMPANY_DEPT  
    ADD CONSTRAINT fk_dept_mgr FOREIGN KEY (MGR_ID)  
        REFERENCES COMPANY_EMP(EMP_ID);
```

-- 3. PROJECT Table

```
CREATE TABLE COMPANY_PROJ (  
    PROJ_ID NUMBER PRIMARY KEY,  
    PROJ_NAME VARCHAR2(60) NOT NULL,  
    BUDGET NUMBER(12,2),  
    DEPT_NO NUMBER,  
    CONSTRAINT fk_proj_dept FOREIGN KEY (DEPT_NO)  
        REFERENCES COMPANY_DEPT(DEPT_NO)  
);
```

-- 4. WORKS_ON Table (M:N Relationship)

```
CREATE TABLE WORKS_ON (  
    EMP_ID NUMBER,  
    PROJ_ID NUMBER,  
    HOURS NUMBER(5,2),  
    CONSTRAINT pk_works_on PRIMARY KEY (EMP_ID, PROJ_ID),  
    CONSTRAINT fk_wo_emp FOREIGN KEY (EMP_ID) REFERENCES COMPANY_EMP(EMP_ID),  
    CONSTRAINT fk_wo_proj FOREIGN KEY (PROJ_ID) REFERENCES COMPANY_PROJ(PROJ_ID)  
);
```

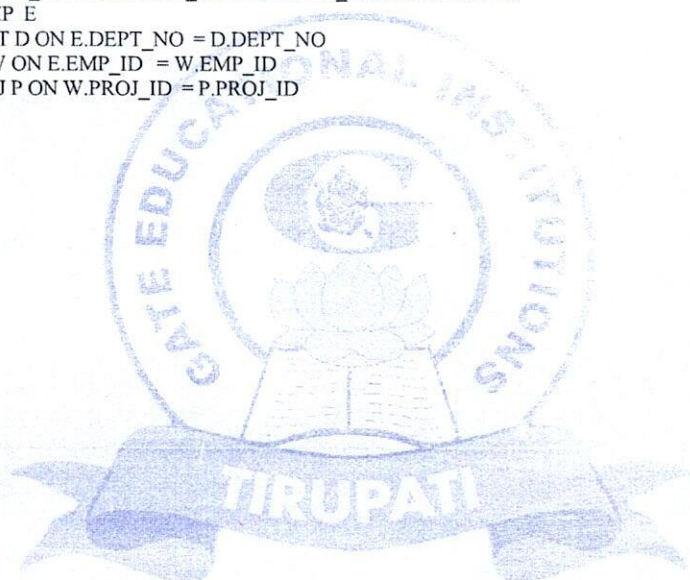
-- 5. DEPENDENT (Weak Entity)

```
CREATE TABLE DEPENDENT (  
    EMP_ID NUMBER,  
    DEP_NAME VARCHAR2(40),  
    RELATION VARCHAR2(20),  
    CONSTRAINT pk_dep PRIMARY KEY (EMP_ID, DEP_NAME),  
    CONSTRAINT fk_dep_emp FOREIGN KEY (EMP_ID) REFERENCES COMPANY_EMP(EMP_ID)  
);
```

-- Insert Sample Data

```
INSERT INTO COMPANY_DEPT(DEPT_NO,DEPT_NAME,LOCATION) VALUES (10,'IT','Hyderabad');
```


Date :	GATE Educational Institutions	Page No. : 81
	INSERT INTO COMPANY_DEPT(DEPT_NO,DEPT_NAME,LOCATION) VALUES (20,'HR','Bangalore');	Practical No. :
<pre> INSERT INTO COMPANY_EMP VALUES (101,'Ravi Kumar', DATE'1990-05-15', 70000, 10); INSERT INTO COMPANY_EMP VALUES (102,'Priya Devi', DATE'1992-08-20', 55000, 20); INSERT INTO COMPANY_EMP VALUES (103,'Arjun Nair', DATE'1988-03-10', 80000, 10); UPDATE COMPANY_DEPT SET MGR_ID = 101 WHERE DEPT_NO = 10; UPDATE COMPANY_DEPT SET MGR_ID = 102 WHERE DEPT_NO = 20; INSERT INTO COMPANY_PROJ VALUES (501,'ERP System', 500000, 10); INSERT INTO COMPANY_PROJ VALUES (502,'HR Portal', 250000, 20); INSERT INTO WORKS_ON VALUES (101, 501, 30.0); INSERT INTO WORKS_ON VALUES (103, 501, 25.5); INSERT INTO WORKS_ON VALUES (101, 502, 10.0); INSERT INTO DEPENDENT VALUES (101, 'Kavya', 'Spouse'); INSERT INTO DEPENDENT VALUES (101, 'Rahul', 'Son'); COMMIT; -- Comprehensive Query: Employee, Department, Project SELECT E.EMP_ID, E.EMP_NAME, D.DEPT_NAME, P.PROJ_NAME, W.HOURS FROM COMPANY_EMP E JOIN COMPANY_DEPT D ON E.DEPT_NO = D.DEPT_NO JOIN WORKS_ON W ON E.EMP_ID = W.EMP_ID JOIN COMPANY_PROJ P ON W.PROJ_ID = P.PROJ_ID ORDER BY E.EMP_ID; </pre>		



Date :	GATE Educational Institutions	Page No. : 87
OUTPUT		Practical No. :

Tables created successfully.
Data inserted and committed.

-- Final JOIN Query Output:

EMP_ID EMP_NAME DEPT_NAME PROJ_NAME HOURS

101	Ravi Kumar	IT	ERP System	30.0
101	Ravi Kumar	IT	HR Portal	10.0
103	Arjun Nair	IT	ERP System	25.5

-- ER Diagram Summary:

Entities : EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT

Relationships: WORKS_IN (M:1), MANAGES (1:1),
WORKS_ON (M:N), HAS DEPENDENT (1:N),
CONTROLS (1:N)

Cardinalities enforced via FK constraints.



Date :	GATE Educational Institutions	Page No. : 83
		Practical No. :

conclusion :—

An Entity - relationship diagram for a company database was designed and successfully translated into a relational SQL schema. The ER model captured entities their attributes and five different relationship with appropriate cardinalities . The M:N relationship between EMPLOYEE and PROJECT was resolved using the WORKS-ON junction table . All constraints including primary key , foreign key , and NOT NULL were implemented , demonstrating the complete ER - to - relational mapping process.