RDBMS Practical-1

1. **create table account**
   (ano varchar2(3), balance number (9), bname varchar2(10));
2. **create table employee**
   (eid varchar2(4), ename varchar2(10), birthdate DATE, salary number (7),
   city varchar2(10));
3. **create table products**
   (id number (10), name varchar2(100), price number (10,2), description varchar2(500));
4. **create table orders**
   (id number (10), user_id number (10), product_id number (10), quantity number (10),
   price number (5,2));
5. **ALTER TABLE Account**
   MODIFY Bname varchar2(30);
6. **ALTER TABLE Account**
   ADD COLUMN birthdate DATE;
7. **ALTER TABLE Account**
   DROP COLUMN birthdate;
8. **ALTER TABLE Account**
   RENAME COLUMN bname to branch_name;
9. **RENAME TABLE Employee to Emp;**
10. **CREATE TABLE Acc2 AS**
    SELECT * FROM Account WHERE 1=0;
11. **CREATE TABLE Emp_Manager AS**
    SELECT *FROM Employee WHERE job_title = 'Manager';
12. **INSERT INTO Acc2**
    SELECT * FROM Account;
13. **DROP TABLE Emp_Manager;**
14. **TRUNCATE TABLE Acc2;**

RDBMS Practical-2

1.
   ```
   INSERT INTO Account VALUES (‘A01’,5000,’vvn’);
   INSERT INTO Account VALUES (‘A02’,6000,’ksad’);
   INSERT INTO Account VALUES (‘A03’,7000,’anand’);
   INSERT INTO Account VALUES (‘A04’,8000,’ksad’);
   INSERT INTO Account VALUES (‘A05’,6000,’vvn’);
   ```

2.
   ```
   INSERT INTO Employee VALUES (‘E01’,’Tulsi’,’26-JAN-82’,12000,’Ahmedabad’,’E02’);
   INSERT INTO Employee VALUES (‘E02’,’Gopi’,’15-AUG-83’,15000,’anand’, NULL);
   INSERT INTO Employee VALUES (‘E03’,’Rajashree’,’31-OCT-84’,20000,’vadodara’, NULL);
   INSERT INTO Employee VALUES (‘E04’,’vaishali’,’23-MAR-85’,25000,’surat’,‘E03’);
   INSERT INTO Employee VALUES (‘E05’,’laxmi’,’14-FEB-83’,18000,’anand’,‘E03’);
   INSERT INTO Employee VALUES (‘E06’,’shivali’,’05-SEP-84’,20000,’surat’,‘E02’);
   ```

3.
   ```
   DELETE FROM Account WHERE bname = 'ksad';
   ```

4.
   ```
   UPDATE Account SET balance = balance * 1.10;
   ```

5.
   ```
   SELECT DISTINCT bname FROM Account;
   ```

6.
   ```
   SELECT name, salary FROM Employee WHERE salary < 8000;
   ```

7.
   ```
   SELECT * FROM Employee WHERE city = 'Anand' AND salary < 17000;
   ```

8.
   ```
   SELECT * FROM Employee WHERE city = 'Anand' OR city = 'Surat';
   ```

9.
   ```
   SELECT * FROM Employee WHERE city NOT IN ('Ahmedabad', 'Vadodara', 'Surat');
   ```

10.
    ```
    SELECT name, salary FROM Employee WHERE salary BETWEEN 5000 AND 15000;
    ```

11.
    ```
    SELECT name || ' lives in ' || city AS employee_info FROM Employee;
    ```

12.
    ```
    SELECT * FROM Employee WHERE name LIKE 'S%';
    ```

13.
    ```
    SELECT * FROM Employee WHERE name LIKE '%A%A%';
    ```

14.
    ```
    SELECT * FROM Employee WHERE LENGTH (name) = 5;
    ```

RDBMS Practical-3

1. **UPDATE employees SET salary = 5000 WHERE employee_id = 'E01';**
   COMMIT;
   DELETE FROM employees WHERE city = 'Anand';
   SELECT * FROM employees;
   ROLLBACK;

2. **Create User YAH identified by 123;**
   Grant Select on employee to YAH;
   Grant Update on employee to YAH;
   Revoke Update on employee from YAH;
   GRANT ALL PRIVILEGES ON TABLE account TO PUBLIC;

RDBMS Practical-4

1. **select add_months('01-JUN-19',4) from dual;**
2. **select months_between('01-MAY-19','01-JUN-19') from dual;**
3. **select last_day('01-FEB-19') from dual;**
4. **select next_day('18-JUN-19','Tuesday') from dual;**
5. **select round (TO_DATE('17-JUN-19 12:35:00 PM', 'DD-MON-YY HH:MI:SS PM'), 'DAY') from dual;**
6. **select trunc (TO_DATE('17-JUN-19 12:35:00 PM', 'DD-MON-YY HH:MI:SS PM'), 'DAY') from dual;**

RDBMS Practical-5

- **Numeric Functions**
    1. **select abs (-15) from dual;**
    2. **select sqrt (81) from dual;**
    3. **select power (3,4) from dual;**
    4. **select mod (16,5) from dual;**
    5. **select floor (-27.2) from dual;**
    6. **select round (182.284, -2) from dual;**
    7. **select trunc (182.284 ,1) from dual;**
- **Character Functions**
    1. **select LENGTH ('Computer Engineering') from dual;**
    2. **select UPPER (name) from employees;**
    3. **select INITCAP ('character function') from dual;**
    4. **select SUBSTR ('Computer Engineering', 12, 11) from dual;**
    5. **select RPAD (last_name, 20, '#') from employees;**
    6. **select LTRIM ('greatest', 'gbrsea') from dual;**
    7. **select REPLACE ('government', 'govern','suppli') from dual;**
    8. **select ASCII('s'), ASCII('A'), ASCII('a') from dual;**
    9. **select INSTR('Database', 'base') from dual;**

RDBMS Practical-6

- **Conversion Functions**
  1. **select TO_NUMBER ('+01234.78', '9999999.99') from dual;**
  2. **select TO_CHAR (123789, '9,99,999') from dual;**
  3. **select TO_CHAR (birth_date, 'Day, DDth Mon, YYYY') from employees;**
  4. **SELECT DECODE ('MAX',**

```
           'MAX', 'this is maximum',
           'MIN', 'this is minimum',
           'This is equal') AS Message
FROM dual;
```

- **Group Functions**
  1. **SELECT COUNT (*) AS TotalEmployees,**

```
MAX (salary) AS MaximumSalary,
MIN (salary) AS MinimumSalary,
AVG (salary) AS AverageSalary
FROM employee;
```

  2. **SELECT COUNT (*) AS TotalEmployees,**

```
SUM (salary) AS TotalSalary
FROM employees WHERE city = 'Surat';
```

RDBMS Practical-7

1. SELECT branch,

   SUM (balance) AS Total Balance

   FROM Account

   GROUP BY branch;

2. SELECT branch,

   SUM (balance) AS TotalBalance

   FROM Account

   WHERE branch = 'vvn'

   GROUP BY branch;

3. SELECT branch,

   SUM (balance) AS TotalBalance

   FROM Account

   GROUP BY branch

   HAVING SUM (balance) > 12000;

4. SELECT city,

   SUM (salary) AS TotalSalary

   FROM Employee

   GROUP BY city

   ORDER BY TotalSalary DESC;

RDBMS Practical-8

1.
```
SELECT employee_id, name, salary,
        CASE
                WHEN salary < 15000 THEN 'Low'
                WHEN salary BETWEEN 15000 AND 20000 THEN 'Medium'
                ELSE 'High'
                END AS SalaryCategory
FROM Employee;
```

2.
```
SELECT
        CASE
                WHEN salary < 15000 THEN 'Low'
                WHEN salary BETWEEN 15000 AND 20000 THEN 'Medium'
                ELSE 'High'
                END AS SalaryCategory,
SUM (salary) AS TotalSalary
FROM Employee
GROUP BY
        CASE
                WHEN salary < 15000 THEN 'Low'
                WHEN salary BETWEEN 15000 AND 20000 THEN 'Medium'
                ELSE 'High'
END;
```

RDBMS Practical-9

1. **SELECT Name FROM Customer**

   UNION

   SELECT Name FROM Employee;

2. **SELECT Name FROM Customer**

   INTERSECT

   SELECT Name FROM Employee;

3. **SELECT Name FROM Customer**

   MINUS

   SELECT Name FROM Employee;

RDBMS Practical-10

1.
```
SELECT a.AccountNumber, a. AccountHolderName, b.BranchName, b.Address
FROM Account a JOIN Branch b
ON a.BranchID = b.BranchID;
```
2.
```
SELECT b.BranchName, b.Address
FROM Account a JOIN Branch b
ON a.BranchID = b.BranchID
WHERE a.AccountNumber = 'A01';
```
3.
```
SELECT e.EmployeeID, e.EmployeeName, m.EmployeeName AS ManagerName
FROM Employee e LEFT JOIN Employee m
ON e.ManagerID = m.EmployeeID;
```
4.
```
SELECT a.AccountNumber, b.BranchName
FROM Account a CROSS JOIN Branch b;
```
5.
```
SELECT e.EmployeeName, c.CustomerName
FROM Employee e LEFT JOIN Customer c
ON e.EmployeeName = c.CustomerName;
```
6.
```
SELECT e.EmployeeName, c.CustomerName
FROM Employee e RIGHT JOIN Customer c
ON e.EmployeeName = c.CustomerName;
```
7.
```
SELECT e.EmployeeName, c.CustomerName
FROM Employee e FULL OUTER JOIN Customer c
ON e.EmployeeName = c.CustomerName;
```

RDBMS Practical-11

1.
   ```sql
   SELECT Balance FROM Account
   WHERE AccountNumber = (SELECT AccountNumber FROM Customer WHERE
   CustomerID = 'C01');
   ```

2.
   ```sql
   SELECT Balance FROM Account
   WHERE AccountNumber IN (SELECT AccountNumber FROM Customer WHERE
   CustomerName = 'Tulsi');
   ```

RDBMS Practical-12

1. **CREATE VIEW employee_info AS SELECT Name, Birthdate, Salary FROM Employee;**
2. **Drop the existing view**

   DROP VIEW IF EXISTS employee_info;

   -- Recreate the view with the updated condition

   CREATE VIEW employee_info AS SELECT Name, Birthdate, Salary FROM Employee WHERE Salary > 10000;
3. **DROP VIEW IF EXISTS employee_info;**

RDBMS Practical-13

1. **CREATE TABLE users (id INTEGER PRIMARY KEY, name VARCHAR2(50), email VARCHAR2(100), password VARCHAR2(100));**
2. **CREATE TABLE products (id INTEGER PRIMARY KEY, name VARCHAR2(100), price DECIMAL (10, 2) CHECK (price > 100), description TEXT);**
3. **CREATE TABLE orders (id INTEGER PRIMARY KEY, user_id INTEGER, product_id INTEGER, quantity INTEGER NOT NULL, FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (product_id) REFERENCES products(id));**
4. **ALTER TABLE Employee ADD CONSTRAINT pk_employee_number PRIMARY KEY (employee_number);**

   ALTER TABLE Employee MODIFY employee_name VARCHAR2(50) NOT NULL;

RDBMS Practical-14

1. CREATE SYNONYM emp_synonym FOR employees;
2. CREATE SEQUENCE seq_emp_id

   START WITH 1

   INCREMENT BY 1;
3. CREATE INDEX idx_emp_name

   ON employees(ename);

RDBMS Practical-15

1.
```
BEGIN
        FOR i IN REVERSE 1..10 LOOP
                DBMS_OUTPUT.PUT_LINE(i);
        END LOOP;
END;
/
```

2.
```
DECLARE
        A NUMBER := &A;          -- Accept input value for A
        B NUMBER := &B;          -- Accept input value for B
        C NUMBER := &C;          -- Accept input value for C
        max_value NUMBER;
BEGIN
        -- Find the maximum value
        max_value := A;
        IF B > max_value THEN
                max_value := B;
        END IF;
        IF C > max_value THEN
                max_value := C;
        END IF;
        -- Display the maximum value
                DBMS_OUTPUT.PUT_LINE('The maximum value is: ' || max_value);
END;
/
```

RDBMS Practical-16

1.

```
DECLARE
        v_account_number  Account.AccountNumber%TYPE;
        v_balance       Account.Balance%TYPE;
        v_min_balance    NUMBER := 5000;
BEGIN
      -- Accept the account number from the user
      v_account_number := &v_account_number;

      -- Fetch the balance for the given account number
      SELECT Balance INTO v_balance FROM Account
      WHERE AccountNumber = v_account_number;

      -- Check if the balance is less than the minimum balance
      IF v_balance < v_min_balance THEN
                -- Deduct Rs.100 from the balance
                UPDATE Account
                SET Balance = Balance - 100
                WHERE AccountNumber = v_account_number;
                DBMS_OUTPUT.PUT_LINE('Rs.100 deducted from the account. New balance: ' ||
    (v_balance - 100));
      ELSE
                DBMS_OUTPUT.PUT_LINE('Balance is sufficient, no deduction needed.');
      END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('Account number not found.');
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

2.

```
DECLARE
        v_branch_name    VARCHAR2(100);
        v_rows_affected   NUMBER;
BEGIN
        -- Accept the branch name from the user
        v_branch_name := '&v_branch_name';

        -- Update the branch names to uppercase
        UPDATE Account
        SET BranchName = UPPER(BranchName)
        WHERE BranchName = v_branch_name;

        -- Get the number of affected rows
        v_rows_affected := SQL%ROWCOUNT;

        -- Display the number of accounts affected
        DBMS_OUTPUT.PUT_LINE(v_rows_affected || ' accounts updated.');
EXCEPTION
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

RDBMS Practical-17

1.

```
DECLARE
        v_employee_id    Employee.EmployeeID%TYPE := &employee_id; -- Accept employee ID
        from user
        v_employee_name  Employee.EmployeeName%TYPE;
BEGIN
        -- Query the employees table for a non-existing employee
         SELECT EmployeeName INTO v_employee_name FROM Employee
          WHERE EmployeeID = v_employee_id;

         -- If found, display the employee name
          DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_employee_name);
EXCEPTION
          WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('No employee found with the given ID.');
          WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

2.

```
DECLARE
        v_employee_id    Employee.EmployeeID%TYPE := &employee_id;
        -- Accept employee ID from user
        v_employee_name  Employee.EmployeeName%TYPE := '&employee_name';
        -- Accept employee name from user
        v_employee_dob   Employee.Birthdate%TYPE := TO_DATE('&employee_dob', 'YYYY-MM-DD');
        -- Accept DOB from user
        v_employee_salary Employee.Salary%TYPE := &employee_salary;
        -- Accept salary from user

        -- Named exception for duplicate value on unique index
        DUP_VAL_ON_INDEX EXCEPTION;

        -- Pragma for associating named exception with Oracle error number
        PRAGMA EXCEPTION_INIT(DUP_VAL_ON_INDEX, -00001);
BEGIN
        -- Attempt to insert data into the Employee table
        INSERT INTO Employee (EmployeeID, EmployeeName, Birthdate, Salary)
        VALUES (v_employee_id, v_employee_name, v_employee_dob, v_employee_salary);

        -- If successful, display a message
        DBMS_OUTPUT.PUT_LINE('Employee inserted successfully.');
EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
                DBMS_OUTPUT.PUT_LINE('Error: An employee with this ID already exists.');
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/
```

RDBMS Practical-18

1.

```
DECLARE
        -- Variables to store user input for length and width
        v_length NUMBER := &length;
        v_width  NUMBER := &width;
        v_area   NUMBER;

        -- Function to calculate the area of a rectangle
        FUNCTION calculate_area(p_length NUMBER, p_width NUMBER) RETURN NUMBER IS
        BEGIN
                RETURN p_length * p_width;
        END calculate_area;
BEGIN
        -- Call the function and store the result
        v_area := calculate_area(v_length, v_width);

        -- Display the result
        DBMS_OUTPUT.PUT_LINE('The area of the rectangle is: ' || v_area);
END;
/
```

2.

- First, create the procedure:

```
CREATE OR REPLACE PROCEDURE get_employee_name
(
        p_emp_no IN Employee.EmployeeID%TYPE,
        p_emp_name OUT Employee.EmployeeName%TYPE
) IS
BEGIN
        -- Query to get the employee name by employee number
        SELECT EmployeeName INTO p_emp_name
        FROM Employee
        WHERE EmployeeID = p_emp_no;

EXCEPTION
        WHEN NO_DATA_FOUND THEN
                p_emp_name := 'No employee found with the given number.';
        WHEN OTHERS THEN
                p_emp_name := 'An error occurred: ' || SQLERRM;
END get_employee_name;
/
```

- **Next, create an anonymous block to call the procedure:**

```
DECLARE
        v_emp_no Employee.EmployeeID%TYPE := &emp_no;
        -- Accept employee number from user
        v_emp_name Employee.EmployeeName%TYPE;
BEGIN
         -- Call the procedure
        get_employee_name(v_emp_no, v_emp_name);

         -- Display the result
        DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_emp_name);
END;
/
```

RDBMS Practical-19

- **Package Specification**

```
CREATE OR REPLACE PACKAGE Employee_Info AS
FUNCTION get_salary(p_emp_id IN Employee.EmployeeID%TYPE)
RETURN Employee.Salary%TYPE;
FUNCTION get_city(p_emp_id IN Employee.EmployeeID%TYPE)
RETURN Employee.City%TYPE;
END Employee_Info;
/
```

- **Package Body**

```
CREATE OR REPLACE PACKAGE BODY Employee_Info AS
        FUNCTION get_salary(p_emp_id IN Employee.EmployeeID%TYPE)
                RETURN Employee.Salary%TYPE
                IS v_salary Employee.Salary%TYPE;
        BEGIN
                -- Query to get the employee's salary by employee ID
                SELECT Salary INTO v_salary
                FROM Employee
                WHERE EmployeeID = p_emp_id;
                RETURN v_salary;
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                RETURN NULL; -- Return NULL if no employee is found with the given ID
                WHEN OTHERS THEN
                RAISE; -- Raise other exceptions
  END get_salary;

        FUNCTION get_city(p_emp_id IN Employee.EmployeeID%TYPE) RETURN
        Employee.City%TYPE IS v_city Employee.City%TYPE;
        BEGIN
                -- Query to get the employee's city by employee ID
                SELECT City INTO v_city FROM Employee
```

```
                        WHERE EmployeeID =  p_emp_id;
                        RETURN v_city;
            EXCEPTION
                        WHEN NO_DATA_FOUND THEN
                        RETURN NULL; -- Return NULL if no employee is found with the given ID
                        WHEN OTHERS THEN
                        RAISE; -- Raise other exceptions
            END get_city;
END Employee_Info;
/
```

RDBMS Practical-20

1.

```
CREATE OR REPLACE TRIGGER prevent_salary_change_surat
BEFORE UPDATE OF Salary ON Employee
FOR EACH ROW
BEGIN
        IF :OLD.City = 'Surat' THEN
        RAISE_APPLICATION_ERROR(-20001, 'Cannot change the salary for employees in
Surat.');
          END IF;
END;
/
```

Example:

```
UPDATE Employee
SET Salary = 60000
WHERE EmployeeID = 1 AND City = 'Surat';
-- This will raise an error if the city of the employee with EmployeeID 1 is 'Surat'
```

2.

```
CREATE OR REPLACE TRIGGER account_operations
BEFORE INSERT OR UPDATE OR DELETE ON Account
BEGIN
  CASE
        WHEN INSERTING THEN
              DBMS_OUTPUT.PUT_LINE('Insert operation is performed on Account table.');
        WHEN UPDATING THEN
              DBMS_OUTPUT.PUT_LINE('Update operation is performed on Account table.');
        WHEN DELETING THEN
              DBMS_OUTPUT.PUT_LINE('Delete operation is performed on Account table.');
  END CASE;
END;
/
```

Example:

```
INSERT INTO Account (AccountNumber, AccountHolderName, BranchID, Balance)
VALUES (101, 'John Doe', 1, 1000);
-- This will display: 'Insert operation is performed on Account table.'

UPDATE Account
SET Balance = 2000
WHERE AccountNumber = 101;
-- This will display: 'Update operation is performed on Account table.'

DELETE FROM Account
WHERE AccountNumber = 101;
-- This will display: 'Delete operation is performed on Account table.'
```

RDBMS Practical-21

Designing the E-R Diagram for a University Database Management System

The E-R diagram should capture the entities and their relationships to manage students, courses, faculty members, course schedules, student enrolment, faculty assignments, student grades, and academic records.

Entities and Attributes:

1. **Student**
    - **StudentID (Primary Key)**
    - **FirstName**
    - **LastName**
    - **DateOfBirth**
    - **Gender**
    - **Address**
    - **Email**
    - **Phone**
2. **Course**
    - **CourseID (Primary Key)**
    - **CourseName**
    - **CourseDescription**
    - **Credits**
3. **Faculty**
    - **FacultyID (Primary Key)**
    - **FirstName**
    - **LastName**
    - **Email**
    - **Phone**
    - **Department**
4. **Schedule**
    - **ScheduleID (Primary Key)**
    - **CourseID (Foreign Key)**
    - **FacultyID (Foreign Key)**
    - **Semester**
    - **Year**
    - **Days**
    - **Time**

5. **Enrollment**
    - **EnrollmentID (Primary Key)**
    - **StudentID (Foreign Key)**
    - **ScheduleID (Foreign Key)**
6. **Grade**
    - **GradeID (Primary Key)**
    - **EnrollmentID (Foreign Key)**
    - **Grade**

E-R Diagram:

Below is a textual description of the E-R diagram as I can't create visual diagrams directly here:

- **Student (1, N) --- (N, 1) Enrollment**
- **Course (1, N) --- (N, 1) Schedule**
- **Faculty (1, N) --- (N, 1) Schedule**
- **Schedule (1, N) --- (N, 1) Enrollment**
- **Enrollment (1, 1) --- (1, N) Grade**

Database Schema in Third Normal Form (3NF)

1. Student Table (1NF, 2NF, 3NF)

```
CREATE TABLE Student (
        StudentID INT PRIMARY KEY,
        FirstName VARCHAR(50),
        LastName VARCHAR(50),
        DateOfBirth DATE,
        Gender CHAR(1),
        Address VARCHAR(255),
        Email VARCHAR(100),
        Phone VARCHAR(15)
);
```

2. Course Table (1NF, 2NF, 3NF)

```
CREATE TABLE Course (
        CourseID INT PRIMARY KEY,
        CourseName VARCHAR(100),
        CourseDescription TEXT,
        Credits INT);
```

3. Faculty Table (1NF, 2NF, 3NF)

```
CREATE TABLE Faculty (
        FacultyID INT PRIMARY KEY,
        FirstName VARCHAR(50),
        LastName VARCHAR(50),
        Email VARCHAR(100),
        Phone VARCHAR(15),
        Department VARCHAR(100)
);
```

4. Schedule Table (1NF, 2NF, 3NF)

```
CREATE TABLE Schedule (
        ScheduleID INT PRIMARY KEY,
        CourseID INT,
        FacultyID INT,
        Semester VARCHAR(10),
        Year INT,
        Days VARCHAR(50),
        Time VARCHAR(50),
        FOREIGN KEY (CourseID) REFERENCES Course(CourseID),
        FOREIGN KEY (FacultyID) REFERENCES Faculty(FacultyID)
);
```

5. Enrollment Table (1NF, 2NF, 3NF)

```
CREATE TABLE Enrollment (
        EnrollmentID INT PRIMARY KEY,
        StudentID INT,
        ScheduleID INT,
        FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
        FOREIGN KEY (ScheduleID) REFERENCES Schedule(ScheduleID)
);
```

6. Grade Table (1NF, 2NF, 3NF)

```
CREATE TABLE Grade (
        GradeID INT PRIMARY KEY,
        EnrollmentID INT,
        Grade CHAR(2),
        FOREIGN KEY (EnrollmentID) REFERENCES Enrollment(EnrollmentID));
```

Explanation of Normalization:

1. **First Normal Form (1NF):**
    - **Each table has a primary key.**
    - **Each column contains atomic values.**
    - **Each column contains values of a single type.**
2. **Second Normal Form (2NF):**
    - **The table is in 1NF.**
    - **All non-key attributes are fully functional dependent on the primary key.**
3. **Third Normal Form (3NF):**
    - **The table is in 2NF.**
    - **All attributes are functionally dependent only on the primary key.**