STRUCTURED QUERY LANGUAGE

SUBMITTED BY

ARYA P P

LIBRARY MANAGEMENT SYSTEM & STUDENT MANAGEMENT SYSTEM

LIBRARY MANAGEMENT SYSTEM

INTRODUCTION

This project focuses on creating a simple Library Management System using a relational database. The system manages Books, Members, and Transactions. It allows for efficient management of books, member records, and book transactions (borrow and return). This system also ensures that the available copies of each book are updated in real time based on borrowing and returning actions, using triggers to automate these updates.

OBJECTIVE

The objective of this project is to:

- Implement a relational database schema to manage books, members, and transactions.
- Automate the update of available copies of books when borrowed and returned.
- Provide SQL queries for common library operations like adding books, updating book details, deleting books, and handling transactions.

KEY FEATURES

- Books Table:

Stores information about books like BookID, Title, Author, Genre, PublishedYear, TotalCopies, and AvailableCopies.

- Members Table:

Stores details of library members including MemberID, Name, Email, Phone number, and JoinDate.

- Transactions Table:

Tracks borrowing and returning of books, with details such as TransactionID, MemberID, BookID, BorrowDate, and ReturnDate.

- Triggers:

Decrease_AvailableCopies: Automatically decreases the available copies when a new book is borrowed (when ReturnDate is NULL).

Increase_AvailableCopies: Automatically increases the available copies when a book is returned (when ReturnDate is not NULL).

DATABASE SCHEMA

- Books Table:

```
create table Books(
  BookID integer primary key,
  Title varchar(100) not null,
  Author varchar(100) not null,
  Genre varchar(50),
  PublishedYear integer,
  TotalCopies integer not null,
  AvailableCopies integer not null);
```

- Members Table:

```
create table Members(
  MemberID integer primary key,
  Name varchar(100) not null,
  Email varchar(100) unique not null,
  Phone number unique not null,

  JoinDate date default current_date);
```

- Transactions Table:

```
create table Transactions(

  TransactionID integer primary key,

  MemberID integer not null,

  BookID integer not null,

  BorrowDate date default current_date,

  ReturnDate date,

  foreign key (MemberID) references Members (MemberID),

  foreign key (BookID) references Books (BookID));
```

LIMITATIONS

- Basic Functionality:

The system only manages book borrowing and returning. It does not include features like reservations, overdue fees, or book suggestions.

Error Handling:

The system does not implement comprehensive error handling for cases such as trying to borrow a book with no available copies.

- Scalability:

The system is designed for small to medium-sized libraries and might need optimizations for large-scale operations.

FUTURE IMPROVEMENTS

- Overdue Book Handling:

Implement a feature to track overdue books and impose fines.

- Book Reservation:

Add functionality to allow members to reserve books that are currently unavailable.

- Enhanced User Interface:

Develop a web or mobile interface for easier interaction with the library system.

- Reporting:

Add reporting capabilities, such as the most borrowed books, active members, etc.

SYSTEM STUDY

- System Overview:

This Library Management System (LMS) aims to streamline the management of books, members, and transactions in a library environment. The system uses a relational database to store data related to books, library members, and the borrowing/returning of books. SQL queries and triggers are used to handle various operations like adding, updating, deleting books, and automating the process of managing available copies when books are borrowed or returned.

- System Objectives:
    - To provide an efficient way of managing library data.
    - To automate book borrowing and returning operations using triggers.
    - To ensure the integrity and consistency of available book copies using database triggers.
- System Requirements:

Hardware Requirements:

- A computer system capable of running SQL-based database management systems like Oracle or MySQL.
- Sufficient storage for the database and backup files.

Software Requirements:

- Database Management System (DBMS) such as Oracle or MySQL.
- SQL client tools (e.g., SQL*Plus, MySQL Workbench).

SYSTEM DESIGN

o Database Design:

The database is designed with three main tables: Books, Members, and Transactions.

o Books Table:

Holds the details of books such as book ID, title, author, genre, and available copies.

The AvailableCopies field is automatically updated through triggers when a book is borrowed or returned.

o Members Table:

Stores information about library members, including member ID, name, email, phone number, and the join date.

o Transactions Table:

Tracks the borrowing and returning of books.

It references both the Members and Books tables to establish relationships.

o Data Flow:

A member borrows a book, creating a record in the Transactions table.

The Decrease_AvailableCopies trigger is fired, updating the AvailableCopies of the book in the Books table.

When a book is returned, the Increase_AvailableCopies trigger is fired, incrementing the AvailableCopies field.

MODULES

- ## Books Module:

Create: Allows the addition of new books to the library.

Update: Enables updates to existing books (e.g., change the number of total and available copies).

Delete: Deletes a book from the library when it is no longer needed.

- ## Members Module:

Create: Adds new members to the library system.

Update: Modifies member details (e.g., name, phone number, or email).

Delete: Removes a member from the system when they are no longer a part of the library.

- ## Transactions Module:

Borrow Book: A member borrows a book, which is recorded in the Transactions table and decreases the AvailableCopies.

Return Book: A member returns a book, updating the Transactions table and increasing the AvailableCopies.

- ## Triggers Module:

Decrease_AvailableCopies: Decreases the available copies of a book when a transaction is inserted (book borrowed).

Increase_AvailableCopies: Increases the available copies when a transaction is updated (book returned).

CONCLUSION

The Library Management System developed here is a simple yet effective solution for managing a small to medium-sized library. It leverages the power of relational databases and SQL triggers to automate the update of book availability as transactions (borrow and return) occur.

- Key Takeaways:

- The system ensures data consistency and reduces manual updates by automating the process of tracking available copies of books.
- The modular design allows for easy future enhancements, such as implementing overdue book tracking or adding book reservations.
- The design is scalable for small to medium-sized libraries, and with future improvements, it could support more complex features for larger libraries.

CODING

```
SQL*Plus: Release 21.0.0.0.0 - Production on Tue Jan 14 19:42:13 2025

Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Enter user-name: system

Enter password:

Last Successful login time: Tue Jan 14 2025 19:40:46 +05:30

Connected to:

Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
```

Version 21.3.0.0.0

#CREATING TABLES

(1)BOOKS:

```
SQL> create table Books(
  2  BookID integer primary key,
  3  Title varchar(100) not null,
  4  Author varchar(100) not null,
  5  Genre varchar(50),
  6  PublishedYear integer,
  7  TotalCopies integer not null,
  8  AvailableCopies integer not null);

Table created.
```

(2)MEMBERS:

```
SQL> create table Members(
  2  MemberID integer primary key,
  3  Name varchar(100) not null,
  4  Email varchar(100) unique not null,
  5  Phone number unique not null,
  6  JoinDate date default current_date);

Table created.
```

(3)TRANSACTIONS:

```
SQL> create table Transactions(
  2  TransactionID integer primary key,
  3  MemberID integer not null,
  4  BookID integer not null,
  5  BorrowDate date default current_date,
  6  ReturnDate date,
  7  foreign key (MemberID) references Members (MemberID),
  8  foreign key (BookID) references Books (BookID));

Table created.
```

#DESCRIBING TABLES

(1)BOOKS:

```
SQL> desc Books;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 BOOKID                                    NOT NULL NUMBER(38)
 TITLE                                     NOT NULL VARCHAR2(100)
 AUTHOR                                    NOT NULL VARCHAR2(100)
 GENRE                                              VARCHAR2(50)
 PUBLISHEDYEAR                                      NUMBER(38)
 TOTALCOPIES                               NOT NULL NUMBER(38)
 AVAILABLECOPIES                         NOT NULL NUMBER(38)
```

(2)MEMBERS:

```
SQL> desc Members;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MEMBERID                                  NOT NULL NUMBER(38)
 NAME                                      NOT NULL VARCHAR2(100)
 EMAIL                                     NOT NULL VARCHAR2(100)
 PHONE                                     NOT NULL NUMBER
 JOINDATE                                           DATE
```

(3)TRANSACTIONS:

```
SQL> desc Transactions;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TRANSACTIONID                             NOT NULL NUMBER(38)
 MEMBERID                                  NOT NULL NUMBER(38)
 BOOKID                                    NOT NULL NUMBER(38)
 BORROWDATE                                         DATE
 RETURNDATE                                         DATE
```

#INSERTING VALUES

(1)BOOKS:

```
SQL> insert into Books Values(1,'The Great Gatsby','F.Scott Fitzgerald','Fiction',1925,5,5);

1 row created.

SQL> insert into Books Values(2,'1984','George Orwell','Dystopian',1949,3,3);

1 row created.

SQL> insert into Books Values(3,'To Kill a Mockingbird','Harper Lee','Fiction',1960,4,4);

1 row created.
```

(2)MEMBERS:

```
SQL> insert into Members values(10,'Alice Johnson','alice@gmail.com',8921829654,to_date('2024-
06-24','yy-mm-dd'));

1 row created.

SQL> insert into Members values(20,'Bob Smith','bob@gmail.com',9495696542,to_date('2024-08-
12','yy-mm-dd'));

1 row created.
```

(3)TRANSACTIONS:

```
SQL> insert into Transactions values(11,10,1,to_date('2024-06-28','yy-mm-dd'),to_date('2024-07-
28','yy-mm-dd'));
```

```
1 row created.

SQL> insert into Transactions values(22,20,3,to_date('2024-09-05','yy-mm-dd'),to_date('2024-10-
05','yy-mm-dd'));

1 row created.
```

#SQL QUERIES

(1)INSERT A NEW BOOK:

```
SQL> insert into Books values(4,'Pride and Prejudice','Jane Austen','Classic',1813,2,2);

1 row created.
```

(2)UPDATE BOOK DETAILS:

```
SQL> update Books set TotalCopies=6,AvailableCopies=6 where BookID=1;

1 row updated.
```

(3)DELETE A BOOK:

```
delete from Books where Title='1948';

1 rows deleted.
```

(4)RETRIEVING COLUMNS FROM TABLE:

```
(.)SQL> select Title,Genre from Books;


TITLE

--------------------------------------------------------------------------------

GENRE

--------------------------------------------------

The Great Gatsby

Fiction


1984

Dystopian


To Kill a Mockingbird

Fiction


Pride and Prejudice

Classic


(.)SQL> select * from Members;


MEMBERID

----------

NAME

--------------------------------------------------------------------------------
```

```
EMAIL

--------------------------------------------------------------------------------

PHONE

----------

JOINDATE

---------

10

Alice Johnson

alice@gmail.com

8921829654

24-JUN-24



MEMBERID

----------

NAME

--------------------------------------------------------------------------------

EMAIL

--------------------------------------------------------------------------------

PHONE

----------

JOINDATE

---------

20

Bob Smith
```

```
bob@gmail.com

9495696542

12-AUG-24
```

```
(.)SQL> select BorrowDate,ReturnDate from Transactions;

BORROWDAT RETURNDATE
--------- ---------
28-JUN-24 28-JUL-24
05-SEP-24 05-OCT-24
```

#ADVANCED FEATURES

(1)ADD A TRIGGER TO UPDATE AVAILABLE COPIES ON BORROW

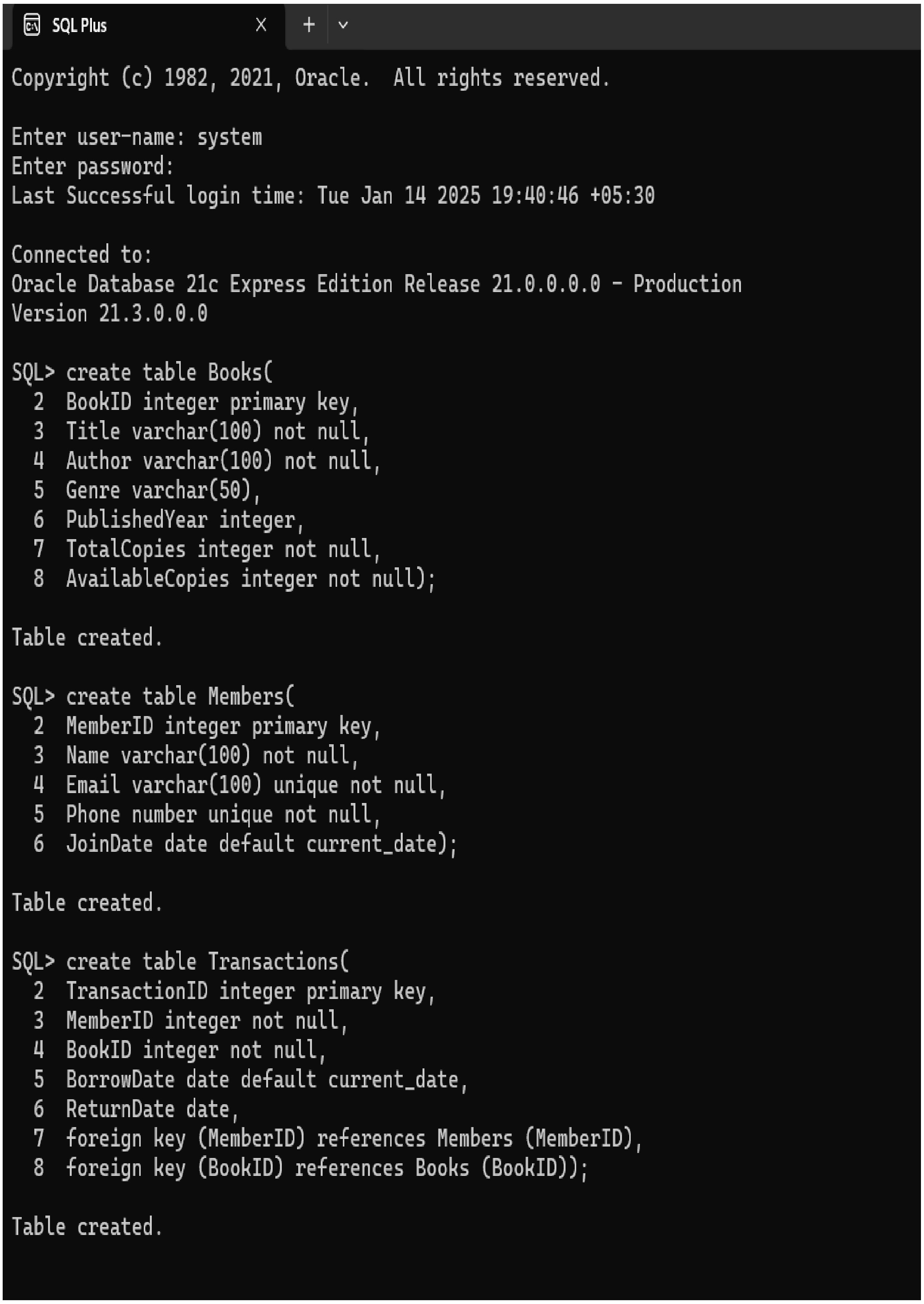
```
SQL> CREATE OR REPLACE TRIGGER Decrease_AvailableCopies
  2  AFTER INSERT ON Transactions
  3  FOR EACH ROW
  4  BEGIN
  5  IF :NEW.ReturnDate IS NULL THEN
  6  UPDATE Books
  7  SET AvailableCopies = AvailableCopies - 1
  8  WHERE BookID = :NEW.BookID;
  9  END IF;
 10  END;
 11  /
```

```
Trigger created.
```

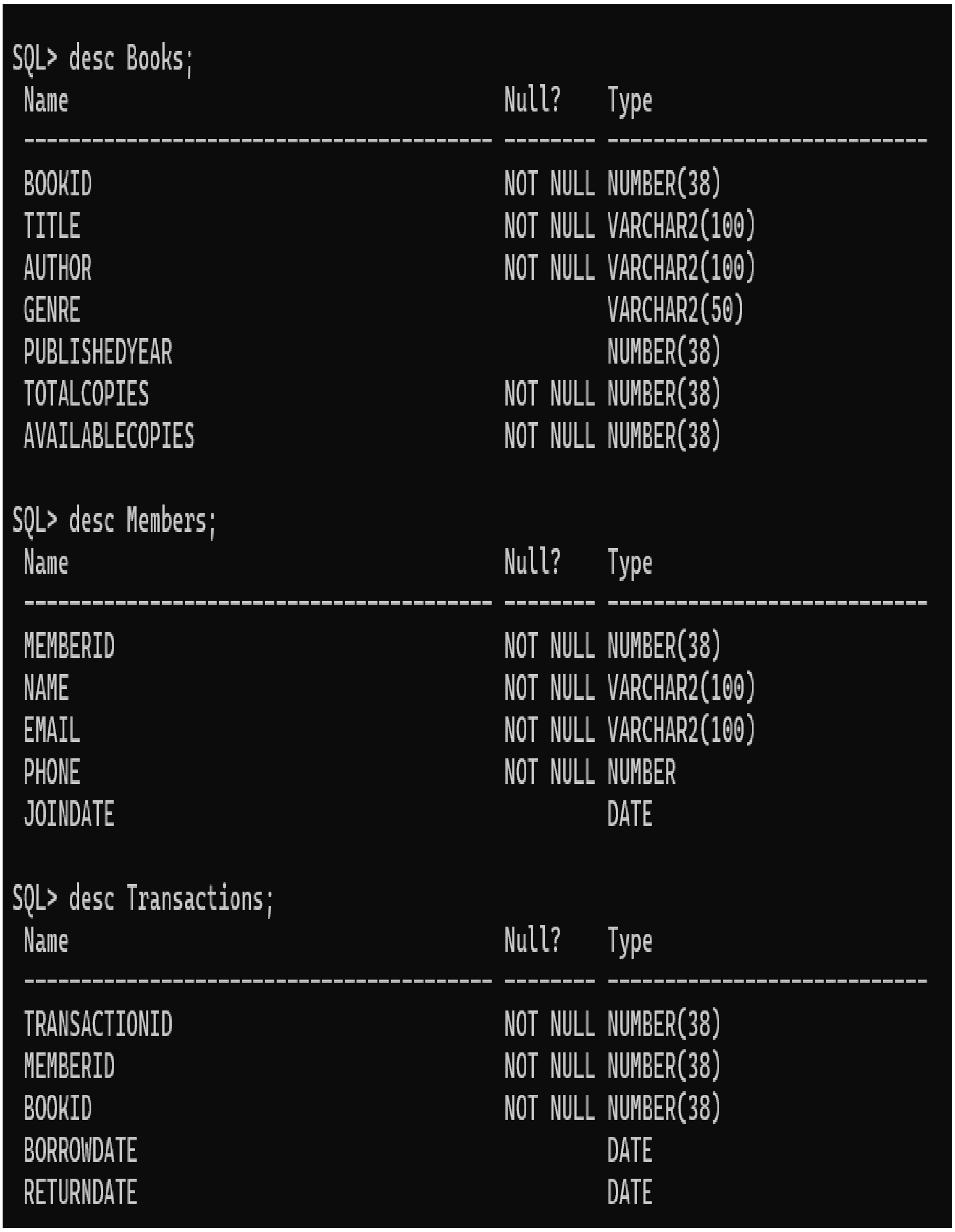
(2)ADD A TRIGGER TO UPDATE AVAILABLE COPIES ON RETURN

```
SQL> CREATE OR REPLACE TRIGGER Increase_AvailableCopies
  2  AFTER UPDATE OF ReturnDate ON Transactions
  3  FOR EACH ROW
  4  BEGIN
  5  IF :NEW.ReturnDate IS NOT NULL THEN
  6  UPDATE Books
  7  SET AvailableCopies = AvailableCopies + 1
  8  WHERE BookID = :NEW.BookID;
  9  END IF;
 10  END;
 11  /

Trigger created.
```

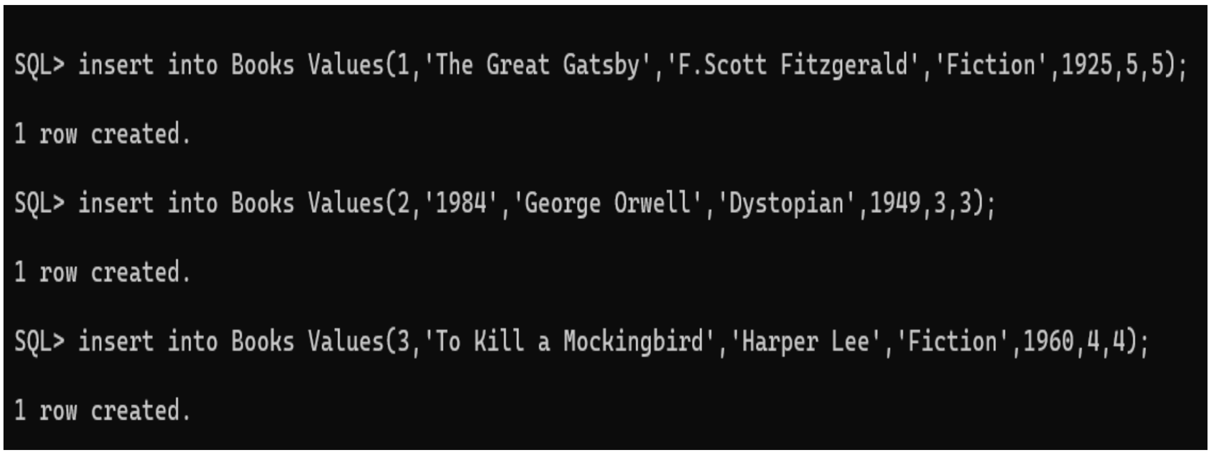
OUTPUT SCREENSHOT



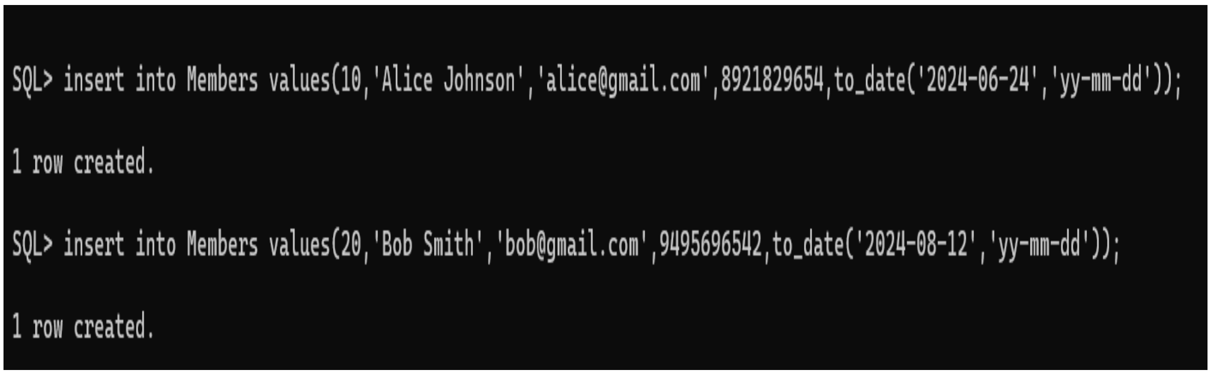
```
Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Enter user-name: system
Enter password:
Last Successful login time: Tue Jan 14 2025 19:40:46 +05:30

Connected to:
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> create table Books(
BookID integer primary key,
Title varchar(100) not null,
Author varchar(100) not null,
Genre varchar(50),
PublishedYear integer,
TotalCopies integer not null,
AvailableCopies integer not null);

Table created.

SQL> create table Members(
MemberID integer primary key,
Name varchar(100) not null,
Email varchar(100) unique not null,
Phone number unique not null,
JoinDate date default current_date);

Table created.

SQL> create table Transactions(
TransactionID integer primary key,
MemberID integer not null,
BookID integer not null,
BorrowDate date default current_date,
ReturnDate date,
foreign key (MemberID) references Members (MemberID),
foreign key (BookID) references Books (BookID));

Table created.
```



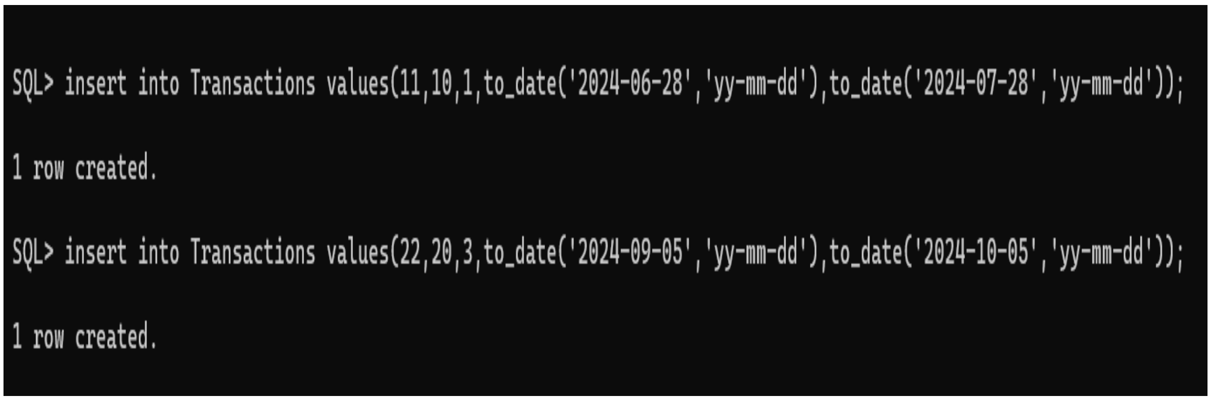
```
SQL> desc Books;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 BOOKID                                    NOT NULL NUMBER(38)
 TITLE                                     NOT NULL VARCHAR2(100)
 AUTHOR                                    NOT NULL VARCHAR2(100)
 GENRE                                              VARCHAR2(50)
 PUBLISHEDYEAR                                      NUMBER(38)
 TOTALCOPIES                               NOT NULL NUMBER(38)
 AVAILABLECOPIES                           NOT NULL NUMBER(38)

SQL> desc Members;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MEMBERID                                  NOT NULL NUMBER(38)
 NAME                                      NOT NULL VARCHAR2(100)
 EMAIL                                     NOT NULL VARCHAR2(100)
 PHONE                                     NOT NULL NUMBER
 JOINDATE                                           DATE

SQL> desc Transactions;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TRANSACTIONID                             NOT NULL NUMBER(38)
 MEMBERID                                  NOT NULL NUMBER(38)
 BOOKID                                    NOT NULL NUMBER(38)
 BORROWDATE                                         DATE
 RETURNDATE                                         DATE
```



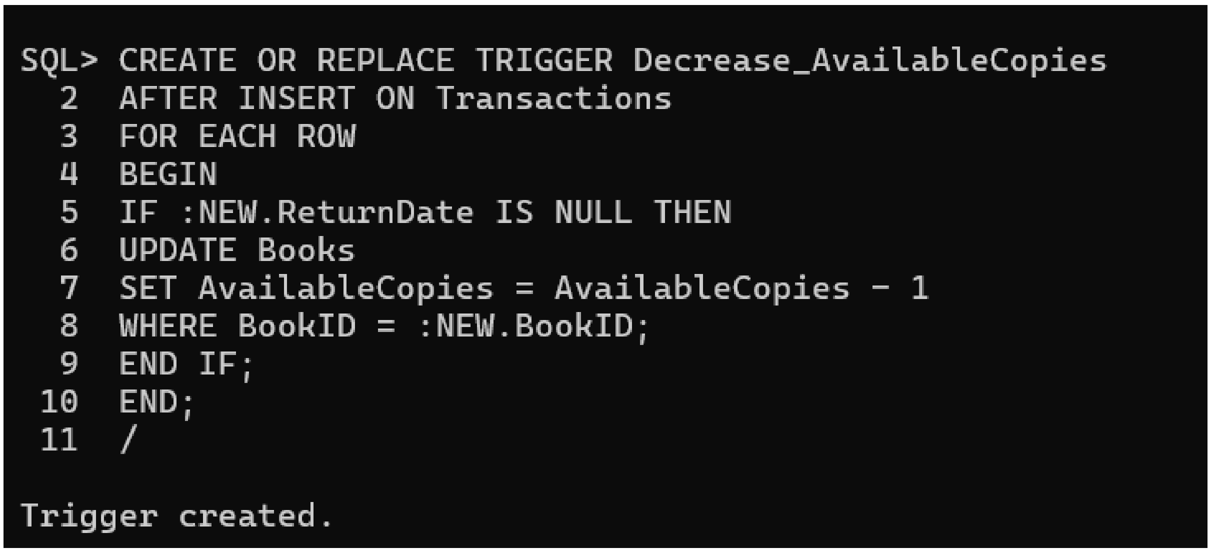
```
SQL> insert into Books Values(1,'The Great Gatsby','F.Scott Fitzgerald','Fiction',1925,5,5);

1 row created.

SQL> insert into Books Values(2,'1984','George Orwell','Dystopian',1949,3,3);

1 row created.

SQL> insert into Books Values(3,'To Kill a Mockingbird','Harper Lee','Fiction',1960,4,4);

1 row created.
```



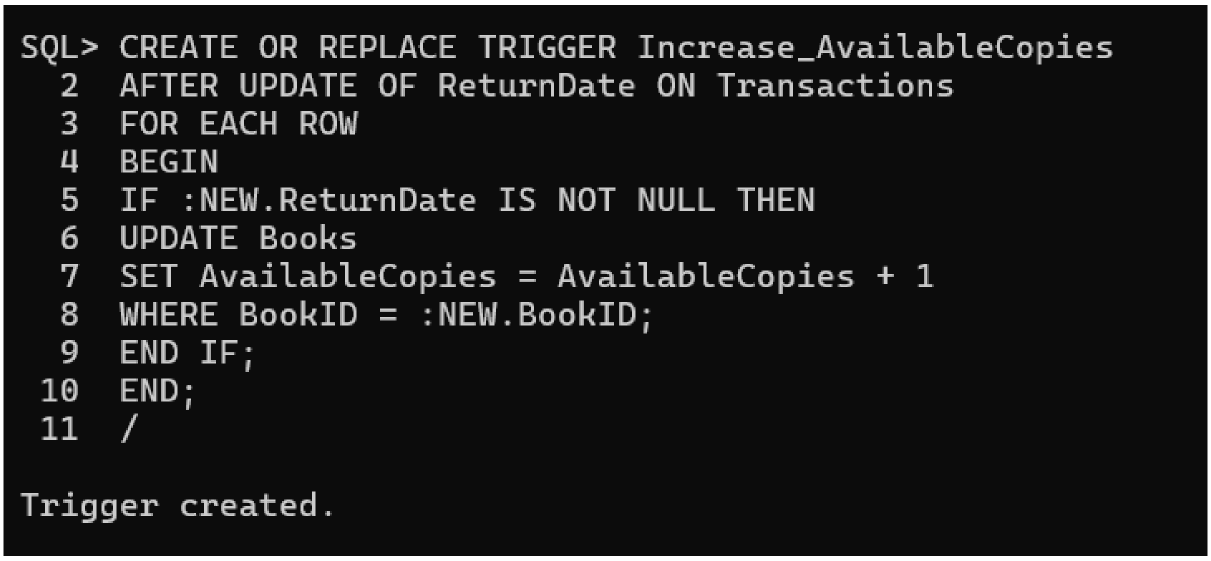
```
SQL> insert into Members values(10,'Alice Johnson','alice@gmail.com',8921829654,to_date('2024-06-24','yy-mm-dd'));

1 row created.

SQL> insert into Members values(20,'Bob Smith','bob@gmail.com',9495696542,to_date('2024-08-12','yy-mm-dd'));

1 row created.
```



```
SQL> insert into Transactions values(11,10,1,to_date('2024-06-28','yy-mm-dd'),to_date('2024-07-28','yy-mm-dd'));

1 row created.

SQL> insert into Transactions values(22,20,3,to_date('2024-09-05','yy-mm-dd'),to_date('2024-10-05','yy-mm-dd'));

1 row created.
```



```
SQL> CREATE OR REPLACE TRIGGER Decrease_AvailableCopies
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
IF :NEW.ReturnDate IS NULL THEN
UPDATE Books
SET AvailableCopies = AvailableCopies - 1
WHERE BookID = :NEW.BookID;
END IF;
END;
/

Trigger created.
```



```
SQL> CREATE OR REPLACE TRIGGER Increase_AvailableCopies
AFTER UPDATE OF ReturnDate ON Transactions
FOR EACH ROW
BEGIN
IF :NEW.ReturnDate IS NOT NULL THEN
UPDATE Books
SET AvailableCopies = AvailableCopies + 1
WHERE BookID = :NEW.BookID;
END IF;
END;
/

Trigger created.
```

STUDENT MANAGEMENT SYSTEM

INTRODUCTION

The Student Management System is a database-driven application designed to efficiently manage student-related data, course information, and enrollment details. The system provides a centralized platform for educational institutions to store, retrieve, and analyze student and course data with ease. By leveraging relational database technology, the system ensures data consistency, integrity, and accessibility.

OBJECTIVE

The primary objective of this system is to:

1. Maintain detailed records of students, courses, and enrollments.
2. Facilitate the management of academic information in a structured manner.
3. Provide tools for querying and analyzing data to enhance decision-making processes.
4. Ensure data security and integrity through proper database constraints and relationships.

KEY FEATURES

1. **Student Records Management**: Stores personal details like name, date of birth, email, and phone number.
2. **Course Management**: Maintains course details such as course name, description, and credits.
3. **Enrollment Tracking**: Links students with courses and records their grades.
4. **Data Retrieval**: Supports SQL-based queries for fetching and analyzing data.
5. **Relational Integrity**: Ensures proper relationships between entities using foreign keys.

6. **Scalability**: Can be expanded to include additional features such as attendance tracking and fee management.

DATABASE SCHEMA

1. **Students Table**:
   - StudentID: Primary key, unique identifier for each student.
   - FirstName: First name of the student.
   - LastName: Last name of the student.
   - DOB: Date of birth of the student.
   - Email: Email address of the student.
   - PhoneNumber: Contact number of the student.
2. **Courses Table**:
   - CourseID: Primary key, unique identifier for each course.
   - CourseName: Name of the course.
   - CourseDescription: Brief description of the course.
   - Credits: Number of credits assigned to the course.
3. **Enrollments Table**:
   - EnrollmentID: Primary key, unique identifier for each enrollment.
   - StudentID: Foreign key referencing Students.StudentID.
   - CourseID: Foreign key referencing Courses.CourseID.
   - Grade: Grade achieved by the student in the course.

LIMITATIONS

1. Lack of user interface for non-technical users to interact with the database.

2. Limited to academic records; does not include features like attendance or financial data.
3. Manual SQL query execution required for data retrieval and analysis.
4. No automated notification or alert system for updates or deadlines.

FUTURE IMPROVEMENTS

1. Development of a user-friendly web or mobile application interface.
2. Integration of attendance and fee management modules.
3. Implementation of data analytics dashboards for performance tracking.
4. Inclusion of automated email and SMS notifications.
5. Support for multiple languages and localization.

SYSTEM STUDY

The system is based on a relational database model, ensuring that data is organized in interconnected tables. It employs SQL for data manipulation and retrieval, providing a robust and scalable solution. The study reveals that most academic institutions face challenges in managing student data due to scattered and unstructured storage. This system addresses these issues by consolidating data into a single platform.

SYSTEM DESIGN

1. **Entity-Relationship (ER) Diagram**: Represents the relationships between Students, Courses, and Enrollments.
2. **Normalization**: Ensures data is stored in a structured manner without redundancy.
3. **Database Constraints**: Includes primary keys, foreign keys, and data type restrictions to maintain integrity.
4. **Logical Flow**:

- Input: Data entry into Students and Courses tables.
- Processing: Enrollments linking students to courses.
- Output: Query results for analysis and reporting.

MODULES

1. **Student Module**:
   - Add, update, and delete student records.
   - View student details.
2. **Course Module**:
   - Add, update, and delete course information.
   - View course details.
3. **Enrollment Module**:
   - Register students for courses.
   - Assign and update grades.
4. **Reporting Module**:
   - Generate reports on student performance.
   - Count and list students enrolled in each course.

CONCLUSION

The Student Management System provides an efficient and reliable method for managing academic records. By leveraging relational databases, it ensures data integrity and accessibility. While the current system meets fundamental requirements, future enhancements will make it more comprehensive and user-friendly. This system is a stepping stone toward fully digitalized academic management.

CODING

```
SQL*Plus: Release 21.0.0.0.0 - Production on Wed Jan 15 19:06:47 2025

Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Enter user-name: system

Enter password:

Last Successful login time: Wed Jan 15 2025 19:00:14 +05:30

Connected to:

Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production

Version 21.3.0.0.0
```

#CREATING TABLES

(1)STUDENTS:

```
SQL> create table Students(
  2  StudentID int primary key,
  3  FirstName varchar(50),
  4  LastName varchar(50),
  5  DOB date,
  6  Email varchar(100),
  7  PhoneNumber integer);

Table created.
```

(2)COURSES:

```
SQL> create table Courses(
  2  CourseID int primary key,
  3  CourseName varchar(100),
  4  CourseDescription varchar(50),
  5  Credits integer);

Table created.
```

(3)ENROLLMENTS:

```
SQL> create table Enrollments(
  2  EnrollmentID int primary key,
  3  StudentID int,
  4  CourseID int,
  5  Grade char(20),
  6  foreign key (StudentID) references Students (StudentID),
  7  foreign key (CourseID) references Courses (CourseID));

Table created.
```

#DESCRIBING TABLES

(1)STUDENTS:

```
SQL> desc Students;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STUDENTID                                 NOT NULL NUMBER(38)
 FIRSTNAME                                          VARCHAR2(50)
 LASTNAME                                           VARCHAR2(50)
 DOB                                                DATE
 EMAIL                                              VARCHAR2(100)
 PHONENUMBER                          NUMBER(38)
```

(2)COURSES:

```
SQL> desc Courses;
 Name                          Null?    Type
 ----------------------------------------- -------- ----------------------------
 COURSEID                      NOT NULL NUMBER(38)
 COURSENAME                             VARCHAR2(100)
 COURSEDESCRIPTION                      VARCHAR2(50)
 CREDITS                                NUMBER(38)
```

(3)ENROLLMENTS:

```
SQL> desc Enrollments;
 Name                          Null?    Type
 ----------------------------------------- -------- ----------------------------
 ENROLLMENTID                  NOT NULL NUMBER(38)
 STUDENTID                              NUMBER(38)
 COURSEID                               NUMBER(38)
 GRADE                                  CHAR(20)
```

#INSERTING VALUES

(1)STUDENTS:

```
SQL> insert into Students values(001,'John','Doe',to_date('2000-01-01','yy-mm-
dd'),'john@gmail.com',9674376345);

1 row created.
```

```
SQL> insert into Students values(002,'Jane','smith',to_date('1999-05-15','yy-mm-
dd'),'smith@gmail.com',9865443223);

1 row created.
```

(2)COURSES:

```
SQL> insert into Courses values(1,'Database Management','Learn about relational databases',4);

1 row created.

SQL> insert into Courses values(2,'Web Development','Introduction to front-end and back-end',3);

1 row created.
```

(3)ENROLLMENTS:

```
SQL> insert into Enrollments values(11,001,1,'A');

1 row created.

SQL> insert into Enrollments values(22,002,2,'B');

1 row created.
```

#QUERYING DATA

(1)FETCH ALL STUDENTS AND THEIR COURSES:

```
SQL> select Students.FirstName,Students.LastName,Courses.CourseName,Enrollments.Grade from
Enrollments join
  2  Students on Enrollments.StudentID=Students.StudentID join
  3  Courses on Enrollments.CourseID=Courses.CourseID;

FIRSTNAME
--------------------------------------------------
LASTNAME
--------------------------------------------------
COURSENAME
----------------------------------------------------------------------------------
GRADE
--------------------
John
Doe
Database Management
A


FIRSTNAME
--------------------------------------------------
LASTNAME
--------------------------------------------------
COURSENAME
----------------------------------------------------------------------------------
GRADE
--------------------
Jane
smith
Web Development
B
```

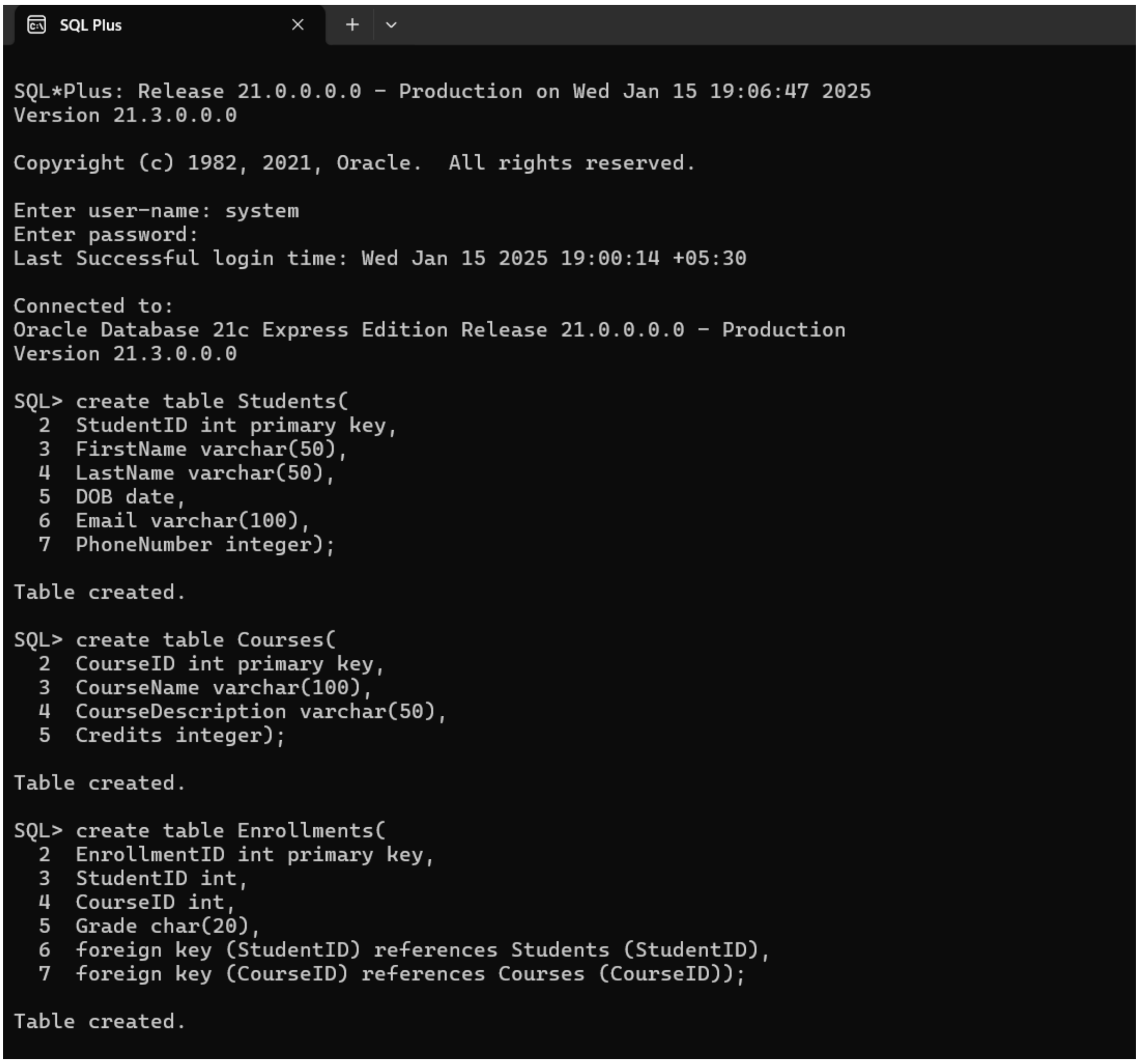
(2)COUNT THE NUMBER OF STUDENTS IN EACH COURSE:

```
SQL> select Courses.CourseName,count(Enrollments.StudentID)as TotalStudents from Enrollments
  2  join Courses on Enrollments.CourseID=Courses.CourseID group by Courses.CourseName;

COURSENAME
--------------------------------------------------------------------------------
TOTALSTUDENTS
-------------
Database Management
            1

Web Development
            1
```

(3)GET STUDENT DETAILS WHO SCORED 'A' IN ANY COURSE:

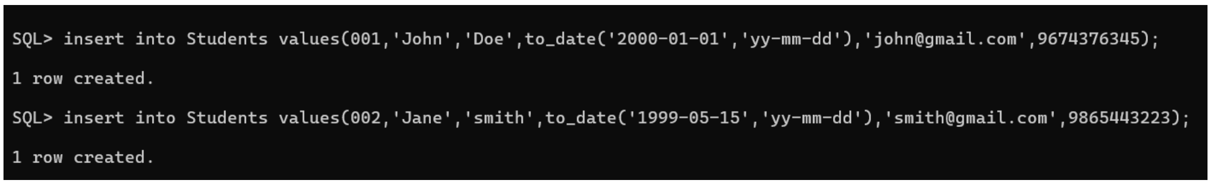
```
SQL> select Students.FirstName,Students.LastName,Courses.CourseName from Enrollments
  2  join Students on Enrollments.StudentID=Students.StudentID
  3  join Courses on Enrollments.CourseID=Courses.CourseID
  4  where Enrollments.Grade='A';

FIRSTNAME
--------------------------------------------------
LASTNAME
--------------------------------------------------
COURSENAME
--------------------------------------------------------------------------------
John
Doe
Database Management
```

OUTPUT SCREENSHOT

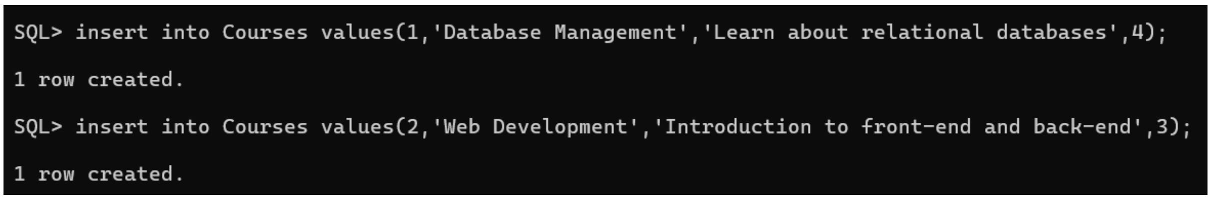


SQL Plus

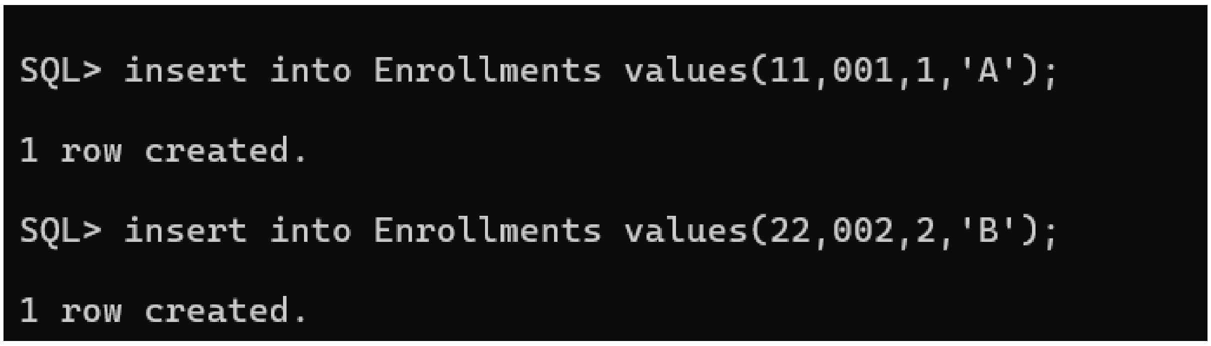
```
SQL*Plus: Release 21.0.0.0.0 - Production on Wed Jan 15 19:06:47 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Enter user-name: system
Enter password:
Last Successful login time: Wed Jan 15 2025 19:00:14 +05:30

Connected to:
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL> create table Students(
  2  StudentID int primary key,
  3  FirstName varchar(50),
  4  LastName varchar(50),
  5  DOB date,
  6  Email varchar(100),
  7  PhoneNumber integer);

Table created.

SQL> create table Courses(
  2  CourseID int primary key,
  3  CourseName varchar(100),
  4  CourseDescription varchar(50),
  5  Credits integer);

Table created.

SQL> create table Enrollments(
  2  EnrollmentID int primary key,
  3  StudentID int,
  4  CourseID int,
  5  Grade char(20),
  6  foreign key (StudentID) references Students (StudentID),
  7  foreign key (CourseID) references Courses (CourseID));

Table created.
```



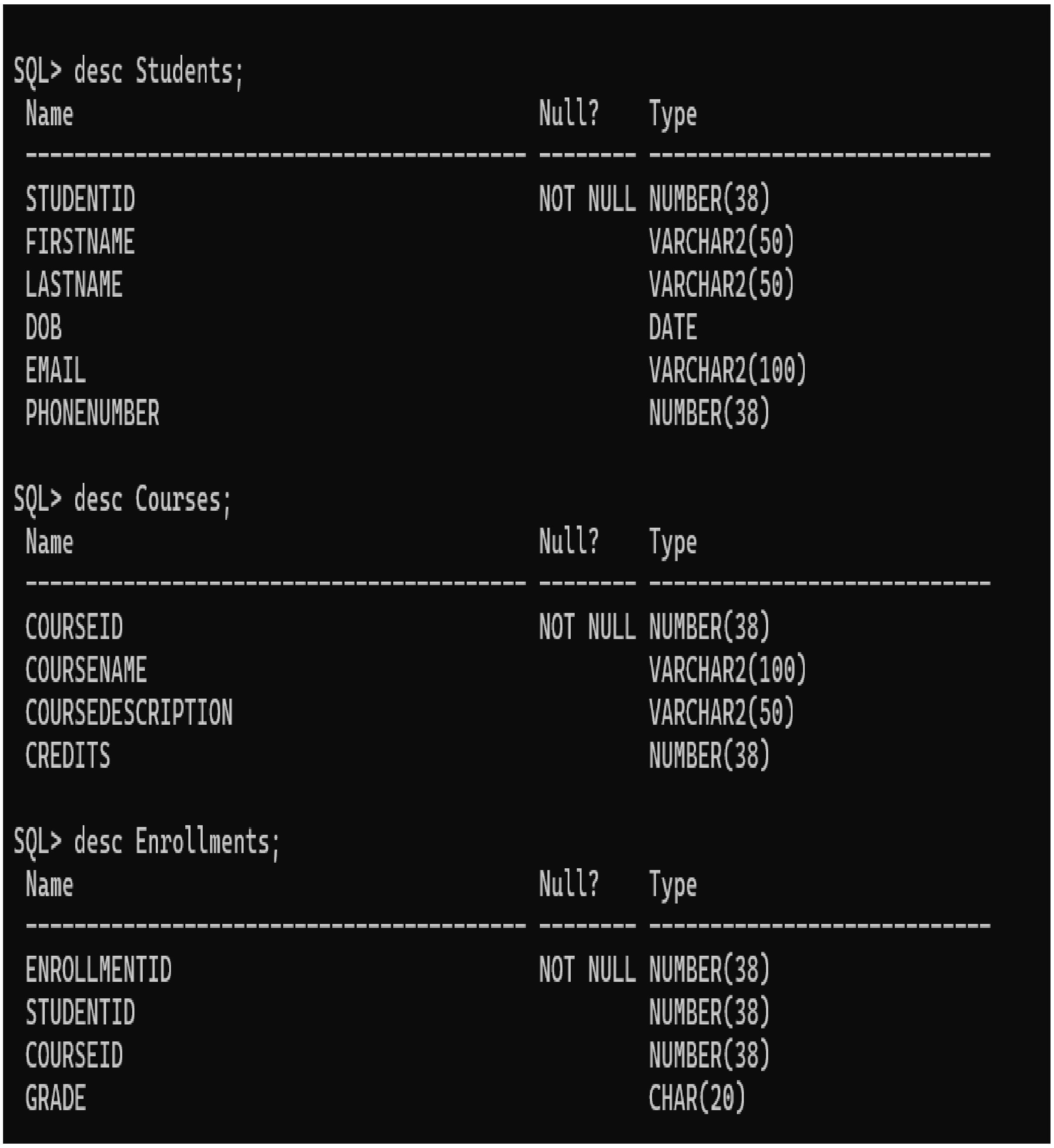
```
SQL> insert into Students values(001,'John','Doe',to_date('2000-01-01','yy-mm-dd'),'john@gmail.com',9674376345);

1 row created.

SQL> insert into Students values(002,'Jane','smith',to_date('1999-05-15','yy-mm-dd'),'smith@gmail.com',9865443223);

1 row created.
```



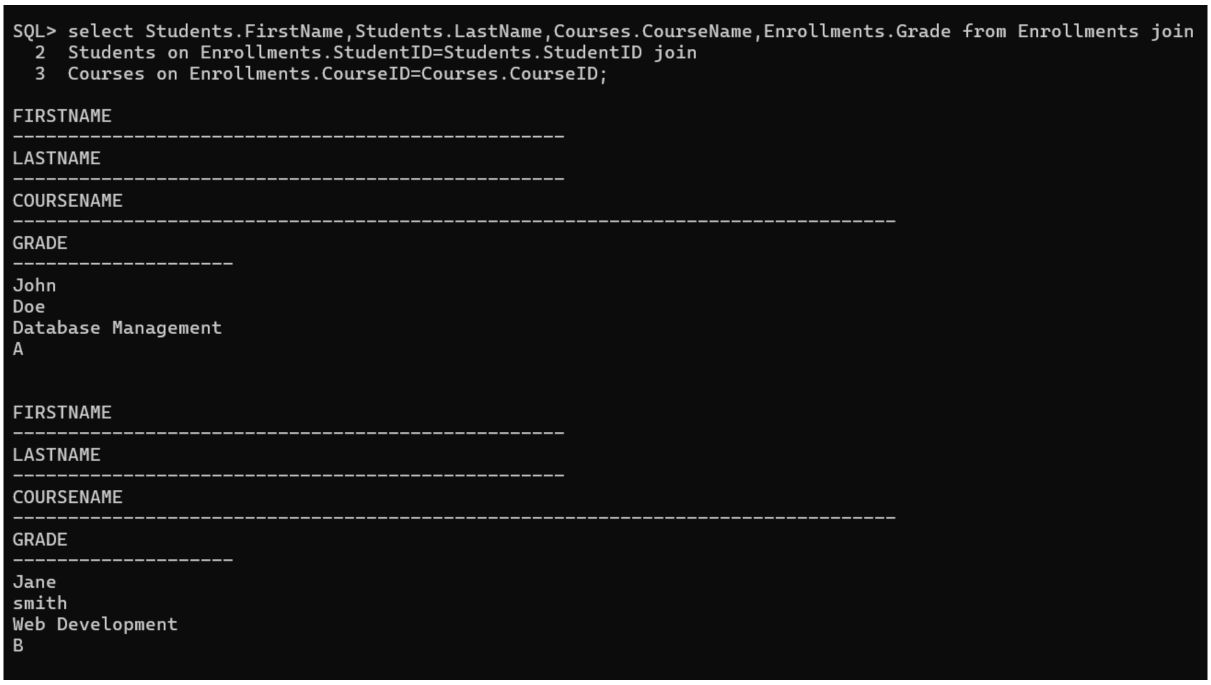
```
SQL> insert into Courses values(1,'Database Management','Learn about relational databases',4);

1 row created.

SQL> insert into Courses values(2,'Web Development','Introduction to front-end and back-end',3);

1 row created.
```



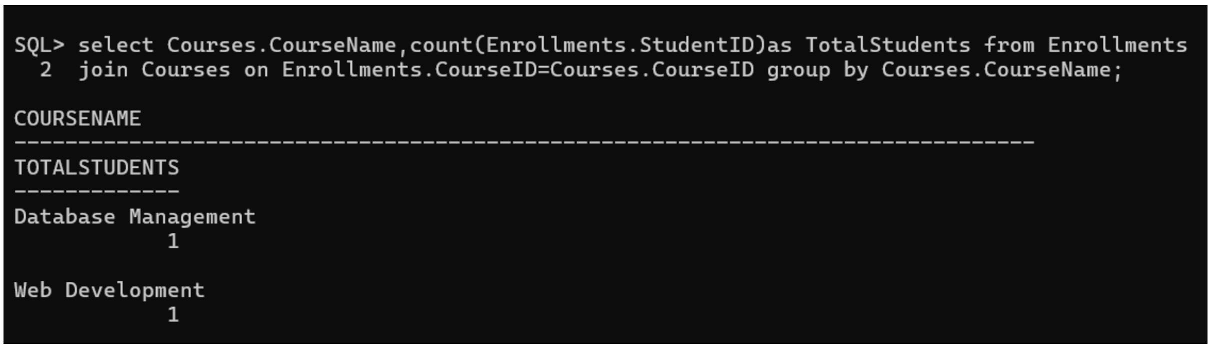
```
SQL> insert into Enrollments values(11,001,1,'A');

1 row created.

SQL> insert into Enrollments values(22,002,2,'B');

1 row created.
```



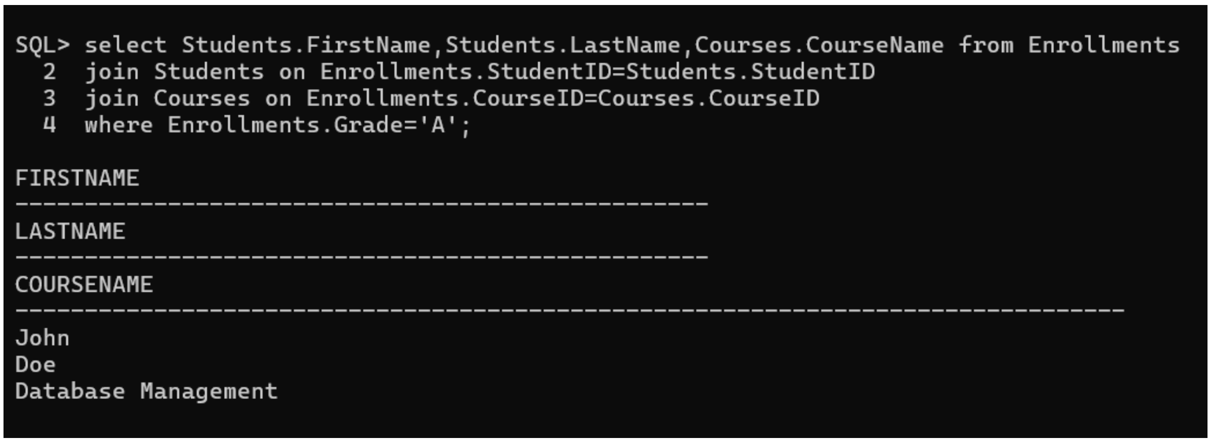
```
SQL> desc Students;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STUDENTID                                 NOT NULL NUMBER(38)
 FIRSTNAME                                          VARCHAR2(50)
 LASTNAME                                           VARCHAR2(50)
 DOB                                                DATE
 EMAIL                                              VARCHAR2(100)
 PHONENUMBER                                        NUMBER(38)

SQL> desc Courses;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COURSEID                                  NOT NULL NUMBER(38)
 COURSENAME                                         VARCHAR2(100)
 COURSEDESCRIPTION                                  VARCHAR2(50)
 CREDITS                                            NUMBER(38)

SQL> desc Enrollments;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ENROLLMENTID                              NOT NULL NUMBER(38)
 STUDENTID                                          NUMBER(38)
 COURSEID                                           NUMBER(38)
 GRADE                                              CHAR(20)
```



```
SQL> select Students.FirstName,Students.LastName,Courses.CourseName,Enrollments.Grade from Enrollments join
  2  Students on Enrollments.StudentID=Students.StudentID join
  3  Courses on Enrollments.CourseID=Courses.CourseID;

FIRSTNAME
--------------------------------------------------
LASTNAME
--------------------------------------------------
COURSENAME
--------------------------------------------------------------------------------
GRADE
--------------------
John
Doe
Database Management
A


FIRSTNAME
--------------------------------------------------
LASTNAME
--------------------------------------------------
COURSENAME
--------------------------------------------------------------------------------
GRADE
--------------------
Jane
smith
Web Development
B
```



```
SQL> select Courses.CourseName,count(Enrollments.StudentID)as TotalStudents from Enrollments
  2  join Courses on Enrollments.CourseID=Courses.CourseID group by Courses.CourseName;

COURSENAME
--------------------------------------------------------------------------------
TOTALSTUDENTS
-------------
Database Management
            1

Web Development
            1
```



```
SQL> select Students.FirstName,Students.LastName,Courses.CourseName from Enrollments
  2  join Students on Enrollments.StudentID=Students.StudentID
  3  join Courses on Enrollments.CourseID=Courses.CourseID
  4  where Enrollments.Grade='A';

FIRSTNAME
--------------------------------------------------
LASTNAME
--------------------------------------------------
COURSENAME
--------------------------------------------------------------------------------
John
Doe
Database Management
```

THANK

YOU