**Module – 5**

**Introduction To DBMS**

1. **Create a new database named school\_db and a table called students with the following columns: student\_id, student\_name, age, class, and address**

**Insert five records into the students table and retrieve all records using the SELECT statement.**

**Ans. CREATE TABLE students**

**(sid INTEGER PRIMARY KEY AUTOINCREMENT, sname TEXT, age INTEGER, class TEXT, address TEXT);**

**INSERT INTO students(sname, age, class, address) VALUES**

**("Bhagirath", 27, "A", "Rajkot"),**

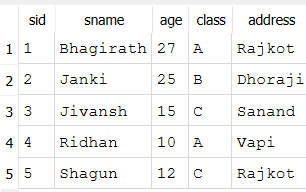
**("Janki", 25, "B", "Dhoraji"),**

**("Jivansh", 15, "C", "Sanand"),**

**("Ridhan", 10, "A", "Vapi"),**

**("Shagun", 12, "C", "Rajkot");**

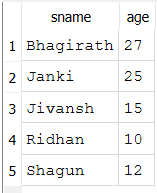
**SELECT \* FROM students;**

**+**

1. **Write SQL queries to retrieve specific columns (student\_name and age) from the students table.**

**Lab 2: Write SQL queries to retrieve all students whose age is greater than 10.**

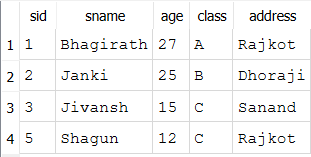
**Ans. SELECT sname, age FROM students;**

****

**SELECT \***

**FROM students**

**WHERE age > 10;**

****

1. **Lab 1: Create a table teachers with the following columns: teacher\_id (Primary Key), teacher\_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).**

**Lab 2: Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table.**

**Ans. CREATE TABLE teachers**

**(tid INTEGER PRIMARY KEY AUTOINCREMENT, tname TEXT, subject TEXT, email UNIQUE);**

**CREATE TABLE students**

**(sid INTEGER PRIMARY KEY AUTOINCREMENT, sname TEXT, city TEXT,**

**tid INTEGER,**

**FOREIGN KEY (tid) REFERENCES teachers (tid));**

**INSERT INTO teachers(tname, subject, email) VALUES**

**('Amit', 'Maths', 'amitaits@gmail.com'),**

**('Yogesh', 'Science', 'yogeshaits@gmail.com'),**

**('Rajdeep', 'Gujarati', 'rajdeepaits@gmail.com'),**

**('Rajesh', 'English', 'rajeshaits@gmail.com'),**

**('Ramesh', 'Hindi', 'rameshits@gmail.com');**

**INSERT INTO students(sname, city, tid) VALUES**

**('Bhagirath', 'Rajkot', 2),**

**('Ridhan', 'Vapi', 1),**

**('Jivansh', 'Sananad', 3),**

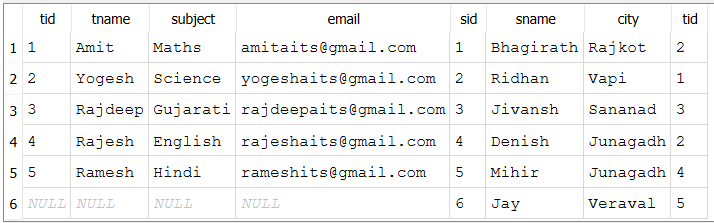
**('Denish', 'Junagadh', 2),**

**('Mihir', 'Junagadh', 4),**

**('Jay', 'Veraval', 5);**

**SELECT \***

**FROM teachers RIGHT JOIN students on teachers.tid=students.sid;**

****

1. **Lab 1: Create a table courses with columns: course\_id, course\_name, and course\_credits. Set the course\_id as the primary key.**

**Ans. use newdb;**

**create table courses**

**(cid integer primary key auto\_increment, cname text, credits integer);**

**select \* from courses;**

****

**Lab 2: Use the CREATE command to create a database university\_db.**

**Ans. create database universitydb;**

1. **Lab 1: Modify the courses table by adding a column course\_duration using the ALTER command.**

**Ans. use newdb;**

**alter table courses add column courses\_duration text;**

**select \* from courses;**

****

**Lab 2: Drop the course\_credits column from the courses table.**

**Ans. alter table courses drop column credits;**

****

1. **Lab 1: Drop the teachers table from the school\_db database.**

**Ans. Drop table teachers;**

**Lab 2: Drop the students table from the school\_db database and verify that the table has been removed.**

**Ans. Drop table students;**

1. **Lab 1: Insert three records into the courses table using the INSERT command.**

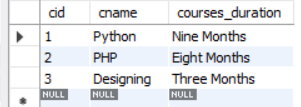
**Ans.**

**insert into courses(cname, courses\_duration)values**

**('Python', 'Nine Months'),**

**('PHP', 'Eight Months'),**

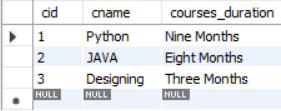
**('Designing', 'Three Months');**

****

**Lab 2: Update the course duration of a specific course using the UPDATE command.**

**Ans.**

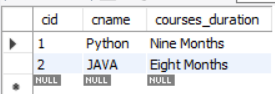
**update courses set cname='JAVA' where cid=2;.**

****

**Lab 3: Delete a course with a specific course\_id from the courses table using the DELETE command.**

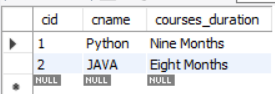
**Ans.**

**delete from courses where cid=3;**

****

1. **Lab 1: Retrieve all courses from the courses table using the SELECT statement.**

**Ans.**

****

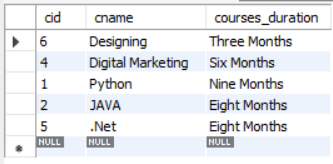
**Lab 2: Sort the courses based on course\_duration in descending order using ORDER BY.**

**Ans.**

**SELECT \***

**FROM courses**

**ORDER BY courses\_duration DESC;**

****

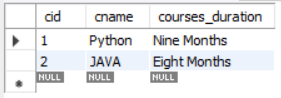
**Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT.**

**Ans.**

**SELECT \***

**FROM Courses**

**LIMIT 2;**

****

1. **Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.**

**Ans.**

**create user user1@127.0.0.1;**

**create user user2@127.0.0.1;**

**grant select on newdb.courses to 'user1'@127.0.0.1;**

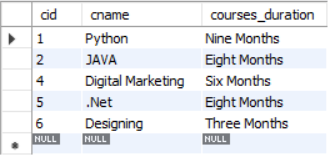
**create user user1@127.0.0.1;**

**create user user2@127.0.0.1;**

**grant select on newdb.courses to 'user1'@127.0.0.1;**

**use newdb;**

**select \* from courses;**

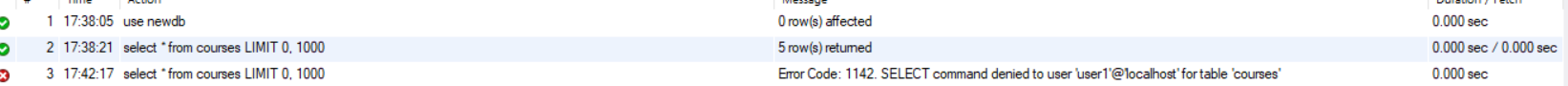
****

**Lab 2: Revoke the INSERT permission from user1 and give it to user2.**

**Ans.**

**revoke select on newdb.courses from 'user1'@127.0.0.1;**

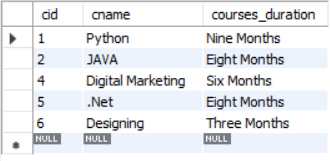
**select \* from courses;**

****

**grant select on newdb.courses to 'user2'@127.0.0.1;**

**use newdb;**

**select \* from courses;**

****

1. **Insert a few rows into the courses table and use COMMIT to save the changes.**

**Ans.**

**insert into courses(cname, courses\_duration)values**

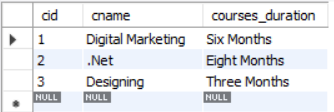
**('Digital Marketing', 'Six Months'),**

**('.Net', 'Eight Months'),**

**('Designing', 'Three Months');**

**commit;**

**select \* from courses;**

****

**Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.**

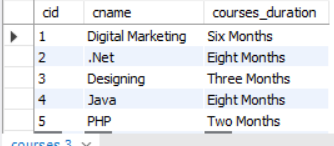
**Ans.**

**insert into courses(cname, courses\_duration)values**

**('Java', 'Eight Months'),**

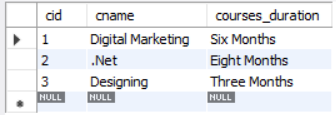
**('PHP', 'Two Months');**

**select \* from courses;**

****

**rollback;**

**select \* from courses;**

****

**Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.**

1. **Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.**

**Ans.**

**CREATE TABLE departments**

**(did integer primary key auto\_increment,dname VARCHAR(20));**

**CREATE TABLE employees**

**(eid integer primary key auto\_increment,**

**ename varchar(20), salary real);**

**insert into departments(dname) values**

**('Human Resources'),**

**('Engineering'),**

**('Marketing'),**

**('Sales');**

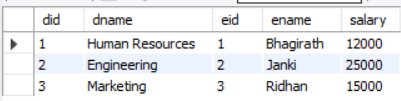
**insert into employees(ename, salary) values**

**('Bhagirath', 12000),**

**('Janki', 25000),**

**('Ridhan', 15000);**

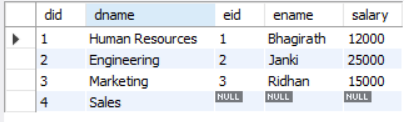
**select \* from departments join employees on departments.did=employees.eid;**

****

**Lab 2: Use a LEFT JOIN to show all departments, even those without employees.**

**Ans.**

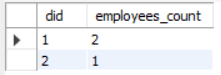
**select \* from departments left join employees on departments.did=employees.eid;**

****

1. **Lab 1: Group employees by department and count the number of employees in each department using GROUP BY.**

**Ans.**

**select did, count(\*) as employees\_count from employees group by did;**

****

**Lab 2: Use the AVG aggregate function to find the average salary of employees in each department.**

**Ans.**

**select avg(salary) from employees;**

****

1. **Write a stored procedure to retrieve all employees from the employees table based on department.**

**Ans.**

**DELIMITER //**

**CREATE PROCEDURE getemployees(IN dept\_id INT)**

**BEGIN**

**SELECT e.eid, e.ename, e.salary, d.dname**

**FROM employees e**

**JOIN departments d ON e.did = d.did**

**WHERE e.did = dept\_id;**

**END //**

**DELIMITER ;**

**call getemployees(2);**

****

**Lab 2: Write a stored procedure that accepts course\_id as input and returns the course details.**

1. **Create a view to show all employees along with their department names.**

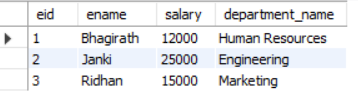
**Ans.**

**CREATE VIEW EmployeeDepartmentView AS**

**SELECT e.eid, e.ename, e.salary, d.dname AS department\_name**

**FROM employees e JOIN departments d ON e.did = d.did;**

**SELECT \* FROM EmployeeDepartmentView;**

****

**Lab 2: Modify the view to exclude employees whose salaries are below $50,000.**

**DROP VIEW IF EXISTS EmployeeDepartmentView;**

**CREATE VIEW EmployeeDepartmentView AS**

**SELECT**

**e.eid,**

**e.ename,**

**e.salary,**

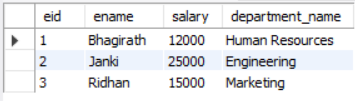
**d.dname AS department\_name**

**FROM employees e**

**JOIN departments d ON e.did = d.did**

**WHERE e.salary <= 50000;**

**SELECT \* FROM EmployeeDepartmentView;**

****

1. **Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.**

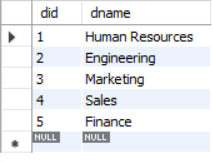
**Ans.**

**START TRANSACTION;**

**INSERT INTO departments (dname) VALUES ('Finance');**

**select \* from departments;**

**SAVEPOINT sp1;**

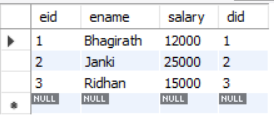
****

**INSERT INTO employees (ename, salary, did) VALUES ('Anjali', 60000, 1);**

**INSERT INTO employees (ename, salary, did) VALUES ('Ravi', 55000, 2);**

**select \* from employees;**

**rollback to sp1;**

****

**Lab 2: Commit part of a transaction after using a savepoint and then rollback the remaining changes.**

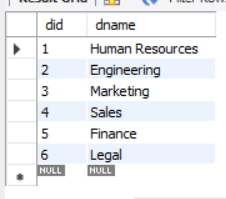
**Ans.**

**START TRANSACTION;**

**insert into departments(dname) values ('Legal');**

**savepoint sp2;**

**select \* from departments;**

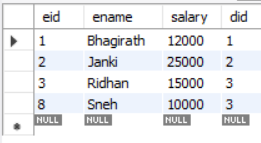
****

**insert into employees (ename, salary, did) values**

**('Sneh', 10000, 3);**

**RELEASE SAVEPOINT sp2;**

**select \* from employees;**

****

**commit;**

**START TRANSACTION;**

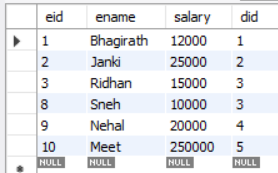
**insert into employees (ename, salary, did) values**

**('Nehal', 20000, 4),**

**('Meet', 250000, 5);**

**select \* from employees;**

**rollback;**

****

1. **Create a database called library\_db and a table books with columns: book\_id, title, author, publisher, year\_of\_publication, and price. Insert five records into the table.**

**Ans.**

**create database libarary\_db;**

**use libarary\_db;**

**create table books**

**(book\_id int auto\_increment primary key,**

**title varchar(100),**

**author varchar(100),**

**publisher varchar(100),**

**year\_of\_publication year,**

**price decimal(8, 2));**

**insert into books (title, author, publisher, year\_of\_publication, price) values**

**('the white tiger', 'aravind adiga', 'harpercollins india', 2008, 14.99),**

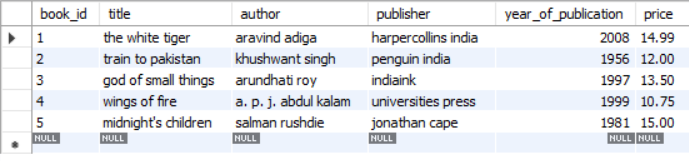
**('train to pakistan', 'khushwant singh', 'penguin india', 1956, 12.00),**

**('god of small things', 'arundhati roy', 'indiaink', 1997, 13.50),**

**('wings of fire', 'a. p. j. abdul kalam', 'universities press', 1999, 10.75),**

**('midnight''s children', 'salman rushdie', 'jonathan cape', 1981, 15.00);**

**select \* from books;**

****

**Create a table members in library\_db with columns: member\_id, member\_name, date\_of\_membership, and email. Insert five records into this table.**

**Ans.**

**create table members**

**(member\_id int auto\_increment primary key,**

**member\_name varchar(100),**

**date\_of\_membership date,**

**email varchar(100));**

**insert into members (member\_name, date\_of\_membership, email) values**

**('rahul sharma', '2023-01-15', 'rahul.sharma@example.com'),**

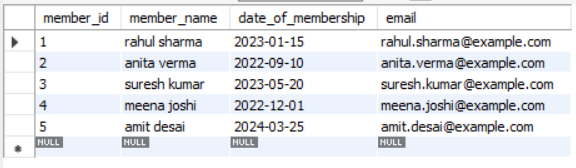
**('anita verma', '2022-09-10', 'anita.verma@example.com'),**

**('suresh kumar', '2023-05-20', 'suresh.kumar@example.com'),**

**('meena joshi', '2022-12-01', 'meena.joshi@example.com'),**

**('amit desai', '2024-03-25', 'amit.desai@example.com');**

**select \* from members;**

****

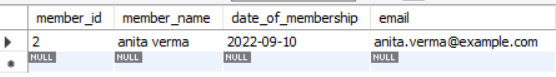
1. **Retrieve all members who joined the library before 2022. Use appropriate SQL syntax with WHERE and ORDER BY.**

**Ans.**

**select \* from members**

**where date\_of\_membership < '2022-10-10'**

**order by date\_of\_membership;**

****

**Lab : Write SQL queries to display the titles of books published by a specific author. Sort the results by year\_of\_publication in descending order.**

**Ans.**

**select title from books**

**where author = 'arundhati roy'**

**order by year\_of\_publication desc;**

****

1. **Add a CHECK constraint to ensure that the price of books in the books table is greater than 0.**

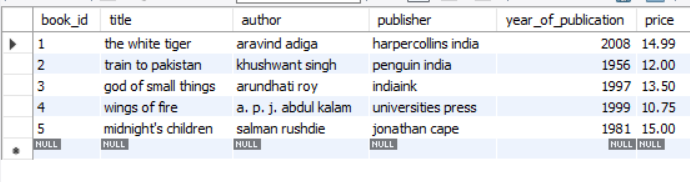
**Ans.**

**alter table books**

**add constraint check\_price\_positive**

**check (price > 0);**

**select \* from books;**

****

**Lab : Modify the members table to add a UNIQUE constraint on the email column, ensuring that each member has a unique email address.**

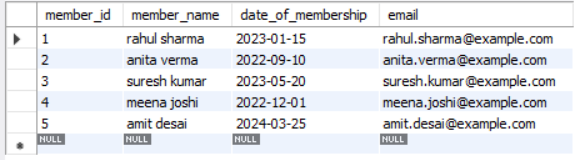
**Ans.**

**alter table members**

**add constraint unique\_email**

**unique (email);**

**select \* from members;**

****

1. **Create a table authors with the following columns: author\_id, first\_name, last\_name, and country. Set author\_id as the primary key.**

**Ans.**

**create table authors**

**(author\_id int auto\_increment primary key,**

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

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

**country varchar(50));**

**select \* from authors;**

****

**Lab : Create a table publishers with columns: publisher\_id, publisher\_name, contact\_number, and address. Set publisher\_id as the primary key and contact\_number as unique.**

**Ans.**

**create table publishers**

**(publisher\_id int auto\_increment primary key,**

**publisher\_name varchar(100),**

**contact\_number varchar(20) unique,**

**address varchar(255));**

**select \* from publishers;**

****

1. **Add a new column genre to the books table. Update the genre for all existing records.**

**Ans.**

**update books**

**set genre = 'literary fiction'**

**where title = 'the white tiger';**

**update books**

**set genre = 'historical fiction'**

**where title = 'train to pakistan';**

**update books**

**set genre = 'contemporary fiction'**

**where title = 'god of small things';**

**update books**

**set genre = 'autobiography'**

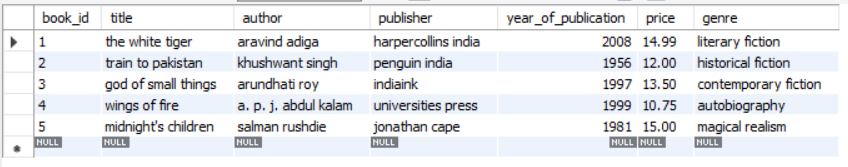
**where title = 'wings of fire';**

**update books**

**set genre = 'magical realism'**

**where title = 'midnight''s children';**

**select \* from books;**

****

**Lab : Modify the members table to increase the length of the email column to 100 characters.**

**Ans.**

**alter table members**

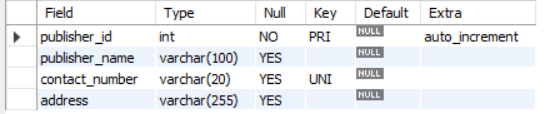
**modify column email varchar(100);**

1. **Drop the publishers table from the database after verifying its structure.**

**Ans.**

**desc publishers;**

**show columns from publishers;**

****

**drop table publishers;**

**Lab : Create a backup of the members table and then drop the original members table.**

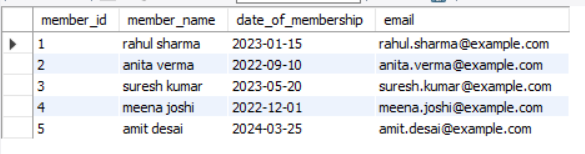
**Ans.**

**create table members\_backup as**

**select \* from members;**

**drop table members;**

**select \* from members\_backup;**

****

1. **Insert three new authors into the authors table, then update the last name of one of the authors.**

**Ans.**

**insert into authors (first\_name, last\_name, country) values**

**('chetan', 'bhagat', 'india'),**

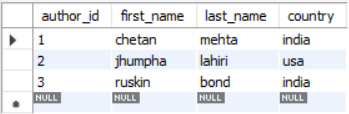
**('jhumpha', 'lahiri', 'usa'),**

**('ruskin', 'bond', 'india');**

**update authors set last\_name = 'mehta'**

**where first\_name = 'chetan' and last\_name = 'bhagat';**

**select \* from authors;**

****

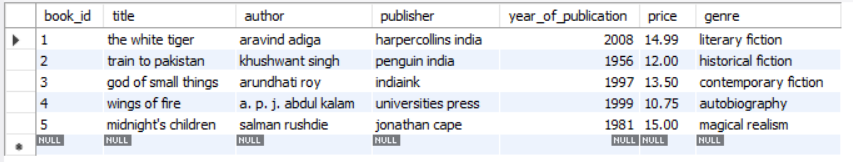
**Lab : Delete a book from the books table where the price is higher than $100.**

**Ans.**

**delete from books**

**where price > 100;**

**select \* from BOOKS;**

****

1. **Update the year\_of\_publication of a book with a specific book\_id.**

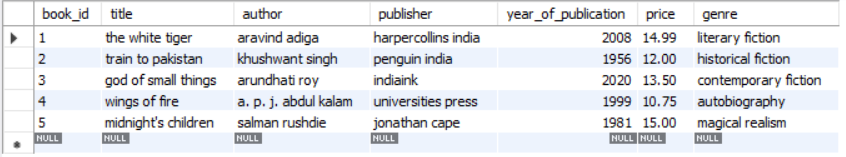
**Ans.**

**update books**

**set year\_of\_publication = 2020**

**where book\_id = 3;**

**select \* from BOOKS;**

****

**Lab : Increase the price of all books published before 2015 by 10%.**

**Ans.**

**update books**

**set price = price \* 1.10**

**where year\_of\_publication < 2015;**

**select \* from BOOKS;**

****

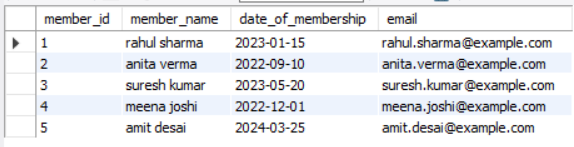
1. **Remove all members who joined before 2020 from the members table.**

**Ans.**

**delete from members\_backup**

**where date\_of\_membership < '2020-01-01';**

**select \* from members\_backup;**

****

**Lab: Delete all books that have a NULL value in the author column.**

**Ans.**

**delete from books**

**where author is null;**

**select \* from books;**

****

1. **Write a query to retrieve all books with price between $16 and $100.**

**Ans.**

**select \* from books**

**where price between 50 and 100;**

****

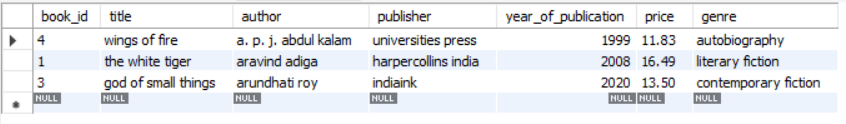
**Lab : Retrieve the list of books sorted by author in ascending order and limit the results to the top 3 entries.**

**Ans.**

**select \* from books**

**order by author asc**

**limit 3;**

****

1. **Use COMMIT after inserting multiple records into the books table, then make another insertion and perform a ROLLBACK.**

**Ans.**

**start transaction;**

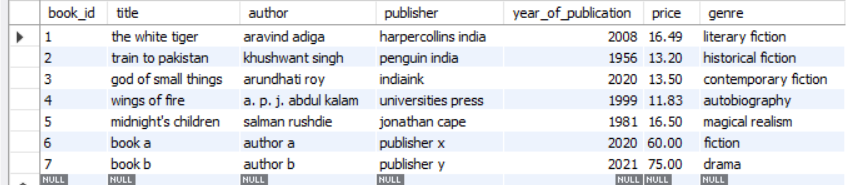
**insert into books (title, author, publisher, year\_of\_publication, price, genre) values**

**('book a', 'author a', 'publisher x', 2020, 60.00, 'fiction'),**

**('book b', 'author b', 'publisher y', 2021, 75.00, 'drama');**

**commit;**

**select \* from books;**

****

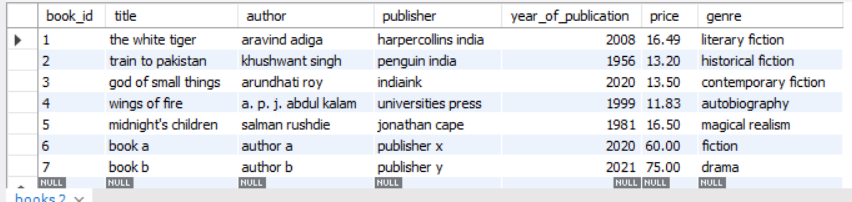
**start transaction;**

**insert into books (title, author, publisher, year\_of\_publication, price, genre)**

**values ('book c', 'author c', 'publisher z', 2022, 85.00, 'mystery');**

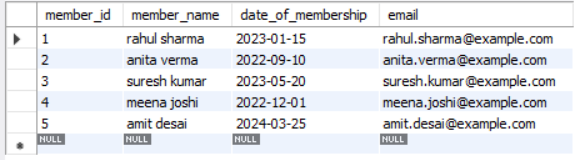
**rollback;**

**select \* from books;**

****

**Lab: Set a SAVEPOINT before making updates to the members table, perform some updates, and then roll back to the SAVEPOINT.**

**Ans.**

****

**savepoint before\_update;**

**update members**

**set email = 'temp1@example.com'**

**where member\_id = 1;**

**update members**

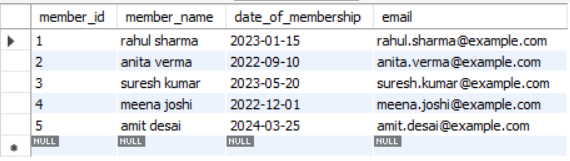
**set email = 'temp2@example.com'**

**where member\_id = 2;**

**rollback to savepoint before\_update;**

**commit;**

**select \* from members;**

****

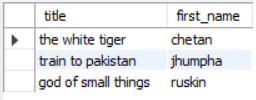
1. **Perform an INNER JOIN between books and authors tables to display the title of books and their respective authors' names.**

**Ans.**

**select books.title, authors.first\_name**

**from books join authors**

**on books.book\_id = authors.author\_id;**

****

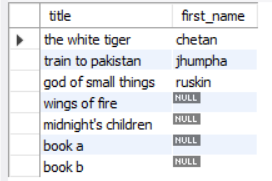
**Lab: Use a FULL OUTER JOIN to retrieve all records from the books and authors tables, including those with no matching entries in the other table.**

**Ans.**

**select books.title, authors.first\_name**

**from books left join authors**

**on books.book\_id = authors.author\_id;**

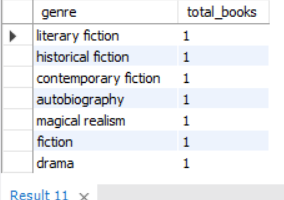
****

1. **Group books by genre and display the total number of books in each genre.**

**Ans.**

**select genre, count(\*) as total\_books from books**

**group by genre;**

****

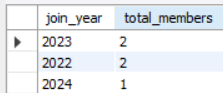
**Lab : Group members by the year they joined and find the number of members who joined each year.**

**Ans.**

**select year(date\_of\_membership) as join\_year, count(\*) as total\_members**

**from members**

**group by year(date\_of\_membership);**

****

1. **Write a stored procedure to retrieve all books by a particular author.**

**Ans.**

**delimiter //**

**create procedure get\_books\_by\_author(in author\_name varchar(100))**

**begin**

**select \* from books**

**where author = author\_name;**

**end //**

**delimiter ;**

**call get\_books\_by\_author('arundhati roy');**

****

**Lab: Write a stored procedure that takes book\_id as an argument and returns the price of the book.**

**Ans.**

**delimiter //**

**create procedure get\_book\_price(in b\_id int)**

**begin**

**select price from books**

**where book\_id = b\_id;**

**end //**

**delimiter ;**

**call get\_book\_price(2);**

****

1. **Create a view to show only the title, author, and price of books from the books table.**

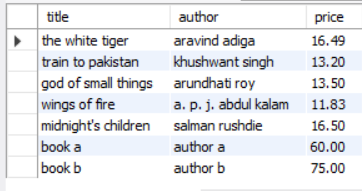
**Ans.**

**create view view\_books\_summary as**

**select title, author, price**

**from books;**

**select \* from view\_books\_summary;**

****

**Lab: Create a view to display members who joined before 2020.**

**Ans.**

**create view view\_members\_before\_2020 as**

**select \***

**from members**

**where date\_of\_membership < '2020-01-01';**

**select \* from view\_members\_before\_2020;**

****

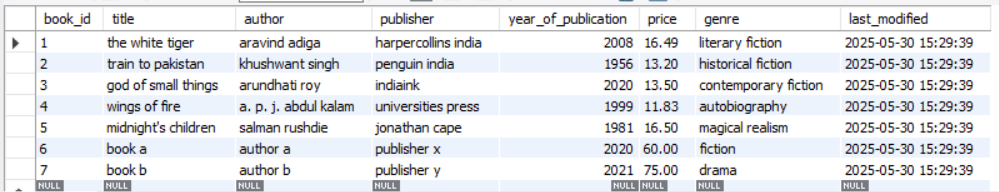
1. **Create a trigger to automatically update the last\_modified timestamp of the books table whenever a record is updated.**

**Ans.**

**alter table books**

**add column last\_modified timestamp default current\_timestamp on update current\_timestamp;**

**select \* from books;**

****

**Lab:** **Create a trigger that inserts a log entry into a log\_changes table whenever a DELETE operation is performed on the books table.**

**Ans.**

**delimiter //**

**create trigger trg\_log\_delete\_books**

**after delete on books**

**for each row**

**begin**

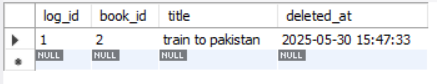
**insert into log\_change (book\_id, title) values (old.book\_id, old.title);**

**end //**

**delimiter ;**

**delete from books where book\_id = 2;**

**select \* from log\_change;**

****

1. **Perform a transaction that includes inserting a new member, setting a SAVEPOINT, and rolling back to the savepoint after making updates.**

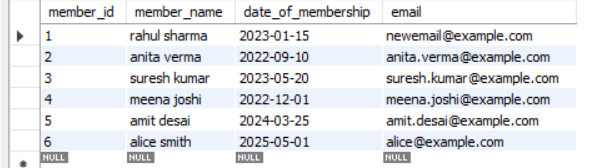
**Ans.**

**start transaction;**

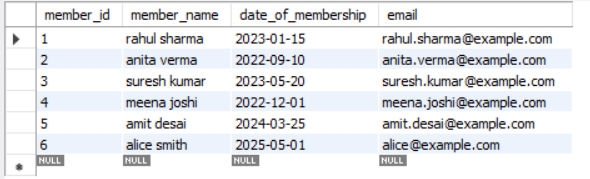
**insert into members (member\_name, date\_of\_membership, email)**

**values ('alice smith', '2025-05-01', 'alice@example.com');**

**savepoint sp\_after\_insert;**

****

**rollback to savepoint sp\_after\_insert;**

****

**Commit;**

**Lab: Use COMMIT after successfully inserting multiple books into the books table, then use ROLLBACK to undo a set of changes made after a savepoint.**

**Ans.**

**start transaction;**

**insert into books (title, author, publisher, year\_of\_publication, price, genre) values**

**('book one', 'author a', 'publisher x', 2010, 45.00, 'fiction'),**

**('book two', 'author b', 'publisher y', 2015, 55.00, 'mystery'),**

**('book three', 'author c', 'publisher z', 2020, 65.00, 'biography');**

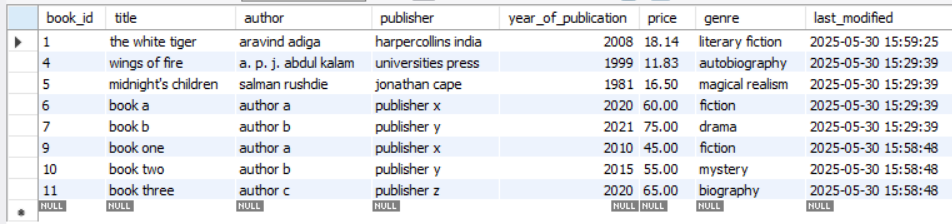
**commit;**

**start transaction;**

**savepoint sp\_before\_update;**

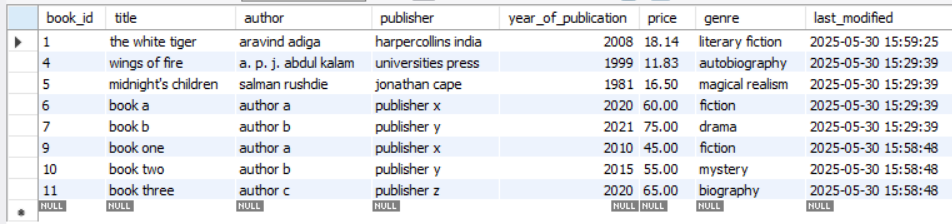
**update books set price = price \* 1.1 where book\_id = 1;**

**select \* from books;**

****

**rollback to savepoint sp\_before\_update;**

**select \* from books;**

****

**commit;**