trigger

create database triggers;

use triggers;

create table customers

(cust\_id int, age int, name varchar(30));

select \* from customers;

delimiter //

create trigger age\_verify

before insert on customers

for each row

if new.age < 0 then set new.age = 0;

end if; //

insert into customers values (101, 20, "Tanavi"),

(102, -21, "Yatharth"),

(103, 70, "Arun");

CREATE TABLE Student (

id INT PRIMARY KEY,

name VARCHAR(20),

grade VARCHAR(3)

);

delimiter //

CREATE TRIGGER t1 BEFORE INSERT ON Student

FOR EACH ROW

BEGIN

SET NEW.grade = UPPER(NEW.grade);

END;

//

insert into student values(101, "Tanu", "a"),(102, "Sid", "B");

SELECT \* FROM triggers.Student;

delimiter //

CREATE TRIGGER t2 BEFORE INSERT ON Student

FOR EACH ROW

BEGIN

IF (NEW.grade NOT IN ('A', 'B', 'C', 'D', 'E')) THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Invalid grade';

END IF;

END; //

insert into student values(103, "Manu", "F"),(104, "Anu", "d"),(105, "Om", "H"),(106, "Ram", "A");

-- Perform implementation of Arithmetic operations and built in functions in SQL

-- queries on school database.

CREATE TABLE School (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(20),

age INT,

grade FLOAT

);

INSERT INTO School VALUES (82, 'Prarthana', 14, 85.5);

INSERT INTO School VALUES (89, 'Darshana', 10, 92.8);

INSERT INTO School VALUES (79, 'Bhavana', 8, 78.2);

INSERT INTO School VALUES (87, 'Shambhoo', 6, 91.7);

-- Arithmetic Operations

SELECT student\_id, age + 1 AS incremented\_age FROM School;

SELECT student\_id, grade - 5 AS adjusted\_grade FROM School;

SELECT student\_id, age \* 2 AS doubled\_age FROM School;

SELECT student\_id, grade / 10 AS grade\_percentage FROM School;

-- Built-in Functions

SELECT COUNT(\*) AS total\_students FROM School;

SELECT AVG(grade) AS average\_grade FROM School;

SELECT MIN(age) AS min\_age FROM School;

SELECT MAX(grade) AS max\_grade FROM School;

SELECT student\_id, ROUND(grade) AS rounded\_grade FROM School;

-- Perform implementation of Group function in SQL queries on employee database.

use triggers;

CREATE TABLE Employee (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(20),

department VARCHAR(20),

salary int

);

INSERT INTO Employee VALUES (1, 'Tanavi', 'IT', 5000);

INSERT INTO Employee VALUES (2, 'Siddhant', 'HR', 5300);

INSERT INTO Employee VALUES (3, 'Shruti', 'Sales', 4500);

INSERT INTO Employee VALUES (4, 'Apeksha', 'IT', 5500);

INSERT INTO Employee VALUES (5, 'Yash', 'Communication', 5600);

INSERT INTO Employee VALUES (6, 'Sakshi', 'Finance', 7000);

SELECT salary, COUNT(\*) AS total\_employees

FROM Employee

GROUP BY salary;

select department, count(department)

from Employee

group by department;

SELECT department, SUM(salary) AS total\_salary

FROM Employee

GROUP BY department;

SELECT department, AVG(salary) AS average\_salary

FROM Employee

GROUP BY department;

SELECT department, MIN(salary) AS min\_salary, MAX(salary) AS max\_salary

FROM Employee

GROUP BY department;

SELECT department, SUM(salary) AS total\_salary

FROM Employee

GROUP BY department

ORDER BY total\_salary DESC

LIMIT 3;

SELECT SUM(salary) AS total\_salary

FROM Employee;

-- Perform implementation of Processing Date and Time functions in SQL queries on library database.

CREATE TABLE Library (

book\_id INT PRIMARY KEY,

book\_title VARCHAR(100),

author VARCHAR(100),

published DATE,

borrowed DATETIME

);

INSERT INTO Library VALUES (1, 'The God of Small Things', 'Arundhati Roy', '1997-06-04', '2022-01-15 09:30:00');

INSERT INTO Library VALUES (2, 'A Suitable Boy', 'Vikram Seth', '1993-01-24', '2022-02-10 14:45:00');

INSERT INTO Library VALUES (3, 'The White Tiger', 'Aravind Adiga', '2008-03-12', '2022-03-25 11:15:00');

INSERT INTO Library VALUES (4, 'The Immortals of Meluha', 'Amish Tripathi', '2010-01-31', '2022-04-20 16:20:00');

SELECT curdate();

SELECT NOW() AS current\_datetime;

SELECT book\_title, YEAR(published) AS publish\_year FROM Library;

SELECT book\_title, MONTHNAME(published) AS publish\_month FROM Library;

SELECT book\_title, DAY(published) AS publish\_day FROM Library;

SELECT book\_title, HOUR(borrowed) AS borrowed\_hour FROM Library;

SELECT book\_title, MINUTE(borrowed) AS borrowed\_minute FROM Library;

SELECT book\_title, SECOND(borrowed) AS borrowed\_second FROM Library;

SELECT book\_title, DATE\_ADD(published, INTERVAL 7 DAY) AS due\_date FROM Library;

SELECT book\_title, DATE\_SUB(borrowed, INTERVAL 2 HOUR) AS return\_time FROM Library;

# trigger---------------------------------------------

create database temp;

use temp;

create table company(

name varchar(45),

id int primary key

);

create table history(

name varchar(45),

id int ,

action varchar(45)

);

delimiter //

create trigger trigger\_B

before delete on company

for each row

begin

insert into history values (old.name,old.id,"delete before");

end;//

delimiter //

create trigger trigger\_A

after delete on company

for each row

begin

insert into history values (old.name,old.id,"delete after");

end;//

insert into company values ('ram', 24);

select \* from company;

select \* from history;

delete from company where id = 23;

delimiter //

create trigger update\_B

before update on company

for each row

begin

insert into history values (old.name,old.id,"update before");

end;//

delimiter //

create trigger insert\_B

before insert on company

for each row

begin

insert into history values (new.name,new.id,"insert before");

end;//

update company SET id = 22 where id = 21;

use temp;

create table emp(

name varchar(45),

id int primary key,

salary int(10)

);

INSERT INTO emp (name, id, salary) VALUES ('John Doe', 1, 50000);

INSERT INTO emp (name, id, salary) VALUES ('Jane Smith', 2, 60000);

INSERT INTO emp (name, id, salary) VALUES ('Mark Johnson', 3, 55000);

INSERT INTO emp (name, id, salary) VALUES ('Emily Davis', 4, 70000);

INSERT INTO emp (name, id, salary) VALUES ('Michael Brown', 5, 45000);

INSERT INTO emp (name, id, salary) VALUES ('Sarah Wilson', 6, 65000);

INSERT INTO emp (name, id, salary) VALUES ('David Thompson', 7, 80000);

select \* from emp;

delimiter //

create procedure tn()

begin

declare x varchar(45);

declare y int default 0;

declare z int(10) default 0;

declare finished int default 0;

declare cur cursor for select name, id, salary from emp order by salary desc limit 5;

declare continue handler for not found set finished =1;

open cur;

lp:loop

fetch cur into x,y,z;

if finished = 1 then leave lp;

end if;

select x,y,z;

end loop lp;

close cur;

end;//

call tn();

-- Perform implementation of Complex queries and set operators in SQL queries on online store database.

use temp;

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

customer\_name VARCHAR(20),

country VARCHAR(10)

);

CREATE TABLE Orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

INSERT INTO Customers VALUES (1, 'Vaishnavi', 'USA');

INSERT INTO Customers VALUES (2, 'Vaibhav', 'India');

INSERT INTO Customers VALUES (3, 'Om', 'USA');

INSERT INTO Customers VALUES (4, 'Samadhan', 'India');

INSERT INTO Customers VALUES (5, 'Prarthana', 'Australia');

INSERT INTO Orders VALUES (1, 1, '2023-05-15');

INSERT INTO Orders VALUES (2, 2, '2023-05-20');

INSERT INTO Orders VALUES (3, 3, '2023-05-25');

INSERT INTO Orders VALUES (4, 2, '2023-05-30');

INSERT INTO Orders VALUES (5, 4, '2023-06-01');

-- customers who have orders in the last 30 days

SELECT c.customer\_name

FROM Customers c

INNER JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE o.order\_date >= CURDATE() - INTERVAL 30 DAY;

-- total number of orders by each customer

SELECT c.customer\_name, COUNT(o.order\_id) AS total\_orders

FROM Customers c

LEFT JOIN Orders o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id, c.customer\_name;

-- customers not placed any orders

SELECT c.customer\_name

FROM Customers c

LEFT JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE o.order\_id IS NULL;

-- customers from either India or USA

SELECT customer\_name

FROM Customers

WHERE country IN ('India', 'USA');

-- customers from India who have placed orders

SELECT c.customer\_name

FROM Customers c

INNER JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE c.country = 'India';

-- customers from India who not placed any orders

SELECT c.customer\_name

FROM Customers c

LEFT JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE c.country = 'India' AND o.order\_id IS NULL;

use temp;

create table students(

id int,

name varchar(45),

marks int,

result varchar(45) default null

);

insert into students values(4,"manu", 55, 'pass'),(5,'sau',30, null),(6,'sam',25, null);

select \* from students;

delimiter $$

create procedure up\_result(IN mark int)

begin

declare mymark int;

set mymark = mark;

if mymark > 35 then

update students set result='pass' where marks=mymark;

else

update students set result='fail' where marks=mymark;

end if;

end; $$

call up\_result(55);

set sql\_safe\_updates = 0;

-- Perform implementation of Complex queries and set operators in SQL queries on online store database.

use temp;

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

customer\_name VARCHAR(20),

country VARCHAR(10)

);

CREATE TABLE Orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

order\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

INSERT INTO Customers VALUES (1, 'Vaishnavi', 'USA');

INSERT INTO Customers VALUES (2, 'Vaibhav', 'India');

INSERT INTO Customers VALUES (3, 'Om', 'USA');

INSERT INTO Customers VALUES (4, 'Samadhan', 'India');

INSERT INTO Customers VALUES (5, 'Prarthana', 'Australia');

INSERT INTO Orders VALUES (1, 1, '2023-05-15');

INSERT INTO Orders VALUES (2, 2, '2023-05-20');

INSERT INTO Orders VALUES (3, 3, '2023-05-25');

INSERT INTO Orders VALUES (4, 2, '2023-05-30');

INSERT INTO Orders VALUES (5, 4, '2023-06-01');

-- customers who have orders in the last 30 days

SELECT c.customer\_name

FROM Customers c

INNER JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE o.order\_date >= CURDATE() - INTERVAL 30 DAY;

-- total number of orders by each customer

SELECT c.customer\_name, COUNT(o.order\_id) AS total\_orders

FROM Customers c

LEFT JOIN Orders o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id, c.customer\_name;

-- customers not placed any orders

SELECT c.customer\_name

FROM Customers c

LEFT JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE o.order\_id IS NULL;

-- customers from either India or USA

SELECT customer\_name

FROM Customers

WHERE country IN ('India', 'USA');

-- customers from India who have placed orders

SELECT c.customer\_name

FROM Customers c

INNER JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE c.country = 'India';

-- customers from India who not placed any orders

SELECT c.customer\_name

FROM Customers c

LEFT JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE c.country = 'India' AND o.order\_id IS NULL;