DBMS PRACTICAL CODES

Assignment 2:

create table Department(dep\_id int not null auto\_increment , dep\_name varchar(20) ,primary key(dep\_id));

create table Employees( emp\_id int not null auto\_increment, emp\_name varchar(30) , salary int ,dept\_id int, primary key(emp\_id),foreign key (dept\_id) references Department(dep\_id));

insert into Department (dep\_name) values ('Administration' ), ('customer\_service' ),( 'finance') ,('human Resource') ,( 'Sales');

select \* from Department;

insert into Employees(emp\_name ,salary , dept\_id) values ( 'abhinash' ,5000,4),('ankur' ,6500,1),('sachin', 8000,5) , ('gourav',7200,3),('manish',5600,null);

select \* from Employees;

select t1.emp\_id , t1.emp\_name ,t2.dep\_name from Employees as t1 left join Department as t2 on t1.dept\_id = t2.dep\_id ;

create view EmployeeDepartment as select t1.emp\_id , t1.emp\_name ,t2.dep\_name from Employees as t1 left join Department as t2 on t1.dept\_id = t2.dep\_id ;

select \* from EmployeeDepartment ;

create or replace view EmployeeDepartment as select t1.emp\_id , t1.emp\_name ,t2.dep\_name ,t1.salary from Employees as t1 left join Department as t2 on t1.dept\_id = t2.dep\_id ;

select \* from EmployeeDepartment ;

insert into Employees(emp\_name ,salary , dept\_id) values ( 'abhinash2' ,6000,3),('ankur2' ,6600,2),('sachin2', 7000,null);

update Employees set salary = 6000 where emp\_id =1 ;

select \* from EmployeeDepartment where dep\_name is null;

delete from employees where salary= 8000;

set sql\_safe\_updates = 0;

delete from employees where salary= 8000;

SET SQL\_SAFE\_UPDATES = 0;

drop view EmployeeDepartment;

**2.B**

CREATE TABLE ProjectLocations (

proj\_id INT PRIMARY KEY,

addr VARCHAR(50) NOT NULL

);

CREATE TABLE Staff (

eid INT PRIMARY KEY,

ename VARCHAR(50) NOT NULL,

addr VARCHAR(50),

salary DECIMAL(10, 2),

commission DECIMAL(10, 2)

);

INSERT INTO Staff (eid, ename, addr, salary, commission) VALUES

(1, 'Amit', 'Pune', 35000, 5000),

(2, 'Sneha', 'Pune', 25000, NULL),

(3, 'Savita', 'Nasik', 28000, 2000),

(4, 'Pooja', 'Mumbai', 19000, NULL),

(5, 'Sagar', 'Mumbai', 25000, 3000),

(6, 'Rohit', 'Jaipur', 40000, NULL),

(7, 'Poonam', 'Patna', 45000, 2000),

(8, 'Arjun', 'Delhi', 20000, 900),

(9, 'Rahul', 'Nagpur', 60000, 5000),

(10, 'Dulquer', 'Kochi', 30000, 1000);

INSERT INTO ProjectLocations (proj\_id, addr) VALUES

(10, 'Mumbai'),

(20, 'Pune'),

(30, 'Jalgaon'),

(40, 'Nagpur'),

(50, 'Delhi'),

(60, 'Kochi'),

(70, 'Pune'),

(80, 'Nasik');

SELECT DISTINCT addr

FROM Staff

WHERE addr IS NOT NULL;

SELECT MAX(salary) AS MaxSalary, MIN(salary) AS MinSalary, AVG(salary) AS AvgSalary, SUM(salary) AS TotalSalary FROM Staff;

SELECT \* FROM Staff ORDER BY salary ASC;

SELECT ename FROM Staff WHERE addr IN ('Nasik', 'Pune');

SELECT ename FROM Staff WHERE commission IS NULL;

UPDATE Staff SET addr = 'Nashik' WHERE ename = 'Amit';

SELECT \* FROM Staff WHERE ename = 'Amit';

SELECT \* FROM Staff WHERE ename LIKE 'A%';

SELECT addr, COUNT(\*) AS StaffCount FROM Staff GROUP BY addr;

SELECT addr FROM Staff

UNION

SELECT addr FROM ProjectLocations;

Assignment3:

create table pack\_grades(

grade\_id int primary key,

grade\_name varchar(50),

min\_price int,

max\_price int

);

create table packages(

pack\_id int PRIMARY KEY,

speed int,

start\_date DATE,

monthly\_payment int,

sector\_id int

);

CREATE TABLE customers (

cust\_id INT PRIMARY KEY,

fname VARCHAR(50),

lname VARCHAR(50),

dob DATE,

joindate DATE,

city VARCHAR(50),

state VARCHAR(50),

street VARCHAR(50),

main\_phone\_num VARCHAR(50),

sec\_phone\_num VARCHAR(50),

fax INT,

monthly\_discount INT,

pack\_id INT,

FOREIGN KEY (pack\_id) REFERENCES packages(pack\_id)

);

CREATE TABLE Sectors (

sector\_id INT PRIMARY KEY,

sector\_name VARCHAR(50)

);

show tables;

INSERT INTO pack\_grades (grade\_id, grade\_name, min\_price, max\_price) VALUES

(1, 'Basic', 100, 500),

(2, 'Standard', 501, 1000),

(3, 'Premium', 1001, 1500);

INSERT INTO Sectors (sector\_id, sector\_name) VALUES

(1, 'Business'),

(2, 'Private');

INSERT INTO packages (pack\_id, speed, start\_date, monthly\_payment, sector\_id) VALUES

(1, 100, '2023-01-01', 300, 1),

(2, 200, '2023-01-01', 700, 2),

(3, 500, '2023-01-01', 1200, 1);

INSERT INTO customers (cust\_id, fname, lname, dob, joindate, city, state, street, main\_phone\_num, sec\_phone\_num, fax, monthly\_discount, pack\_id) VALUES

(101, 'John', 'Doe', '1985-06-15', '2006-03-15', 'New York', 'NY', '5th Ave', '123-456-7890', '098-765-4321', 12345, 20, 1),

(102, 'Jane', 'Smith', '1990-11-20', '2010-06-25', 'Los Angeles', 'CA', 'Sunset Blvd', '234-567-8901', '987-654-3210', 23456, 25, 2),

(103, 'Alice', 'Taylor', '1988-01-12', '2015-04-10', 'Chicago', 'IL', 'Michigan Ave', '345-678-9012', '876-543-2109', 34567, 15, 3);

select \* from pack\_grades;

select \* from packages;

select \* from customers;

select \* from sectors;

-- query 1:

select c.fname,c.lname,c.pack\_id,p.speed

from customers as c

inner join packages as p

on c.pack\_id=p.pack\_id;

-- query2:

select c.fname,c.lname,c.pack\_id,p.speed

from customers as c

inner join packages as p

on c.pack\_id=p.pack\_id

where c.pack\_id=2 or c.pack\_id=3 order by c.lname ASC;

-- query 3:

select p.pack\_id,p.speed,p.monthly\_payment,s.sector\_name

from packages p

inner join sectors s

on p.sector\_id=s.sector\_id;

show tables;

describe sectors;

-- query 4:

select c.fname,c.lname,c.pack\_id,p.speed,p.monthly\_payment,s.sector\_name

from packages p

inner join customers c on c.pack\_id=p.pack\_id

inner join sectors s on p.sector\_id=s.sector\_id;

-- query5:

select c.fname,c.lname,c.pack\_id,p.speed,p.monthly\_payment,s.sector\_name

from packages p

inner join customers c on c.pack\_id=p.pack\_id

inner join sectors s on p.sector\_id=s.sector\_id

where s.sector\_name='Business';

-- query6:

SELECT c.lname, c.fname, c.joindate, p.pack\_id, p.speed, s.sector\_name

FROM customers c

INNER JOIN packages p ON c.pack\_id = p.pack\_id

INNER JOIN sectors s ON p.sector\_id = s.sector\_id

WHERE s.sector\_name = 'Private' AND YEAR(c.joindate) = 2006;

select \* from customers;

select \* from sectors;

select \* from packages;

SELECT p.pack\_id, p.speed, p.monthly\_payment, pg.grade\_id

FROM packages AS p

LEFT JOIN pack\_grades AS pg ON p.pack\_id = pg.grade\_id

UNION

SELECT p.pack\_id, p.speed, p.monthly\_payment, pg.grade\_id

FROM packages AS p

RIGHT JOIN pack\_grades AS pg ON p.pack\_id = pg.grade\_id;

select c.fname,c.lname,p.speed,p.monthly\_payment

from customers as c

inner join packages as p

on c.pack\_id=p.pack\_id;

SELECT p.pack\_id, p.speed, p.monthly\_payment, s.sector\_name

FROM packages p

INNER JOIN Sectors s ON p.sector\_id = s.sector\_id;

SELECT p.pack\_id, p.speed, p.monthly\_payment, c.fname, c.lname

FROM packages p

LEFT JOIN customers c ON p.pack\_id = c.pack\_id;

SELECT c1.lname, c1.fname, c1.monthly\_discount

FROM customers c1

join customers c2 on c1.monthly\_discount < c2.monthly\_discount

where c1.cust\_id=103;

SELECT p1.pack\_id, p1.speed

FROM packages p1

JOIN packages p2 ON p1.speed = p2.speed

WHERE p2.pack\_id = 1;

Assignment4:(a)

-- Create Database

create database library;

use library;

-- Create Table Borrower

create table Borrower(Rollno int(4), Name varchar(20), DateofIssue date, NameofBook varchar(30), Status varchar(10));

-- Insert Data into Borrower Table

insert into Borrower values(14, 'Ram', '2022-09-19', 'Operating System', 'I');

insert into Borrower values(27, 'Soham', '2022-07-24', 'Object Oriented Programming', 'I');

insert into Borrower values(34, 'Mohan', '2022-06-12', 'Microprocessor', 'I');

insert into Borrower values(48, 'Om', '2022-04-19', 'Mechanics', 'I');

-- Select All from Borrower

select \* from Borrower;

-- Output:

-- | Rollno | Name | DateofIssue | NameofBook | Status |

-- |--------|-------|--------------|----------------------------|--------|

-- | 14 | Ram | 2022-09-19 | Operating System | I |

-- | 27 | Soham | 2022-07-24 | Object Oriented Programming | I |

-- | 34 | Mohan | 2022-06-12 | Microprocessor | I |

-- | 48 | Om | 2022-04-19 | Mechanics | I |

-- Create Table Fine

create table Fine(Rollno int(4), Date date, Amount int(10));

-- Create Procedure to Calculate Fine

delimiter //

create procedure calc\_Fine(in r int(10), in b varchar(30))

begin

declare doi date;

declare diff int(3);

select DateofIssue into doi from Borrower where Rollno = r and NameofBook = b;

select datediff(curdate(), doi) into diff;

if diff >= 15 and diff <= 30 then

insert into Fine values(r, curdate(), diff \* 5);

end if;

if diff > 30 then

insert into Fine values(r, curdate(), diff \* 50);

e nd if;

end //

delimiter ;

-- Call Procedure to Calculate Fine for Ram (Operating System)

call calc\_Fine(14, 'Operating System');

-- Output for Fine:

-- | Rollno | Date | Amount |

-- |--------|------------|--------|

-- | 14 | 2024-10-25 | 38400 |

-- Call Procedure to Calculate Fine for Soham (OOP)

call calc\_Fine(27, 'Object Oriented Programming');

-- Call Procedure to Calculate Fine for Mohan (Microprocessor)

call calc\_Fine(34, 'Microprocessor');

-- Call Procedure to Calculate Fine for Om (Mechanics)

call calc\_Fine(48, 'Mechanics');

-- Select All from Fine

select \* from Fine;

-- Output:

-- | Rollno | Date | Amount |

-- |--------|------------|--------|

-- | 14 | 2024-10-25 | 38400 |

-- | 27 | 2024-10-25 | 41250 |

-- | 34 | 2024-10-25 | 43350 |

-- | 48 | 2024-10-25 | 46050 |

-- Create Procedure to Submit Book

delimiter //

create procedure submit(in r int(2))

begin

update Borrower set Status = 'R' where Rollno = r;

delete from Fine where Rollno = r;

end //

delimiter ;

-- Call Procedure to Submit Books

call submit(14);

call submit(27);

call submit(48);

call submit(34);

-- Select All from Fine (After submission)

select \* from Fine;

-- Output:

-- | Rollno | Date | Amount |

-- |--------|------|--------|

-- | | | | -- Empty, as all fines are cleared

-- Select All from Borrower (After submission)

select \* from Borrower;

-- Output:

-- | Rollno | Name | DateofIssue | NameofBook | Status |

-- |--------|-------|--------------|----------------------------|--------|

-- | 14 | Ram | 2022-09-19 | Operating System | R |

-- | 27 | Soham | 2022-07-24 | Object Oriented Programming | R |

-- | 34 | Mohan | 2022-06-12 | Microprocessor | R |

-- | 48 | Om | 2022-04-19 | Mechanics | R |

Assignment 4(b):

CREATE TABLE IF NOT EXISTS areas (

radius INT,

area DECIMAL(10, 2)

);

DELIMITER //

CREATE PROCEDURE Calculate\_Circle\_Areas()

BEGIN

DECLARE v\_radius INT DEFAULT 5;

DECLARE v\_area DECIMAL(10, 2);

loop\_label: LOOP

SET v\_area = 3.14159 \* v\_radius \* v\_radius;

INSERT INTO areas (radius, area) VALUES (v\_radius, v\_area);

SET v\_radius = v\_radius + 1;

IF v\_radius > 9 THEN

LEAVE loop\_label;

END IF;

END LOOP loop\_label;

COMMIT;

END //

DELIMITER ;

CALL Calculate\_Circle\_Areas();

select \* from areas;

Assignment 5:

-- Step 1: Create the Database and Use It

CREATE DATABASE Score;

USE Score;

-- Step 2: Create Tables

CREATE TABLE stud\_marks (

name VARCHAR(20),

total\_marks INT(5)

);

CREATE TABLE Result (

roll\_no INT(3) PRIMARY KEY,

name VARCHAR(20),

class VARCHAR(20)

);

-- Step 3: Insert Data into stud\_marks

INSERT INTO stud\_marks VALUES ('Suresh', 995);

INSERT INTO stud\_marks VALUES ('Harish', 865);

INSERT INTO stud\_marks VALUES ('Samart', 920);

INSERT INTO stud\_marks VALUES ('Mohan', 1000);

INSERT INTO stud\_marks VALUES ('Soham', 745);

-- Step 4: View Data in stud\_marks

SELECT \* FROM stud\_marks;

-- Step 5: Insert Data into Result

INSERT INTO Result (roll\_no, name) VALUES (1, 'Suresh');

INSERT INTO Result (roll\_no, name) VALUES (2, 'Harish');

INSERT INTO Result (roll\_no, name) VALUES (3, 'Samart');

INSERT INTO Result (roll\_no, name) VALUES (4, 'Mohan');

INSERT INTO Result (roll\_no, name) VALUES (5, 'Soham');

-- Step 6: View Data in Result

SELECT \* FROM Result;

-- Step 7: Create Procedure for Grade Assignment

DELIMITER //

CREATE PROCEDURE proc\_Grade(IN r INT(2), OUT grade CHAR(25))

BEGIN

DECLARE m INT(4);

SELECT total\_marks INTO m FROM stud\_marks WHERE name = (SELECT name FROM Result WHERE roll\_no = r);

IF m >= 990 AND m <= 1500 THEN

SELECT 'Distinction' INTO grade;

UPDATE Result SET class = 'Distinction' WHERE roll\_no = r;

ELSEIF m >= 900 AND m <= 989 THEN

SELECT 'FirstClass' INTO grade;

UPDATE Result SET class = 'FirstClass' WHERE roll\_no = r;

ELSEIF m >= 825 AND m <= 899 THEN

SELECT 'SecondClass' INTO grade;

UPDATE Result SET class = 'SecondClass' WHERE roll\_no = r;

ELSE

SELECT '--' INTO grade;

UPDATE Result SET class = '--' WHERE roll\_no = r;

END IF;

END //

DELIMITER ;

-- Step 8: Create Function to Get Grade

DELIMITER //

CREATE FUNCTION func\_Grade(r INT(2)) RETURNS VARCHAR(25) DETERMINISTIC

BEGIN

DECLARE grade VARCHAR(25);

CALL proc\_Grade(r, grade);

RETURN grade;

END //

DELIMITER ;

-- Step 9: Test the Function with Various Roll Numbers

SELECT func\_Grade(1);

SELECT func\_Grade(2);

SELECT func\_Grade(3);

SELECT func\_Grade(4);

SELECT func\_Grade(5);

-- Step 10: View Updated Data in Result Table

SELECT \* FROM Result;

Assignment 6:

create table oldEmp(

rollNum int Primary Key,

name varchar(100),

salary int);

insert into oldEmp values

(1,'Aditi',50000),

(2,'Aditya',49000),

(3,'Anurag',55000),

(4,'Asim',60000),

(5,'Aryan',57000),

(6,'Aneesh',54000),

(7,'Abhishek',52000),

(8,'Amit',51000),

(9,'Anay',58000);

select \* from oldEmp;

+---------+----------+--------+

| rollNum | name | salary |

+---------+----------+--------+

| 1 | Aditi | 50000 |

| 2 | Aditya | 49000 |

| 3 | Anurag | 55000 |

| 4 | Asim | 60000 |

| 5 | Aryan | 57000 |

| 6 | Aneesh | 54000 |

| 7 | Abhishek | 52000 |

| 8 | Amit | 51000 |

| 9 | Anay | 58000 |

+---------+----------+--------+

create table newEmp(

rollNum int Primary Key,

name varchar(100),

salary int);

insert into newEmp values

(2,'Aditya',49000),

(5,'Aryan',57000),

(6,'Aneesh',54000),

(9,'Anay',58000);

Select \* from newEmp;

+---------+--------+--------+

| rollNum | name | salary |

+---------+--------+--------+

| 2 | Aditya | 49000 |

| 5 | Aryan | 57000 |

| 6 | Aneesh | 54000 |

| 9 | Anay | 58000 |

+---------+--------+--------+

delimiter //

create procedure compareTables()

begin

declare r int;

declare n varchar(100);

de

clare s int;

declare c int;

declare exit\_loop int(1);

declare cur cursor for select rollNum,name,salary from oldEmp;

declare continue handler for not found set exit\_loop=1;

open cur;

simple\_loop:LOOP

fetch cur into r,n,s;

select count(rollNum) into c from newEmp where rollNum=r;

if c=0 then

insert into newEmp values(r,n,s);

end if;

if exit\_loop=1 then

close cur;

leave simple\_loop;

end if;

end LOOP simple\_loop;

end //

delimiter ;

call compareTables();

select \* from newEmp;

+---------+----------+--------+

| rollNum | name | salary |

+---------+----------+--------+

| 1 | Aditi | 50000 |

| 2 | Aditya | 49000 |

| 3 | Anurag | 55000 |

| 4 | Asim | 60000 |

| 5 | Aryan | 57000 |

| 6 | Aneesh | 54000 |

| 7 | Abhishek | 52000 |

| 8 | Amit | 51000 |

| 9 | Anay | 58000 |

+---------+----------+--------+

Assignment 7:

create table library(

rollnum int,

name varchar(100),

issuedate date,

bookname varchar(100));

create table library\_audit(

rollnum int,

name varchar(100),

issuedate date,

bookname varchar(100),

operationTime timestamp);

insert into library values

(1,'Aditi','2021-08-15','JAVA'),

(2,'Aditya','2021-08-01','DBMS'),

(3,'Anurag','2021-08-31','TOC'),

(4,'Asim','2021-09-10','CNS'),

(5,'Aryan','2021-09-04','DS'),

(6,'Aneesh','2021-08-20','HCI'),

(7,'Abhishek','2021-08-13','PYTHON');

mysql> select\* from library;

+---------+----------+------------+----------+

| rollnum | name | issuedate | bookname |

+---------+----------+------------+----------+

| 1 | Aditi | 2021-08-15 | JAVA |

| 2 | Aditya | 2021-08-01 | DBMS |

| 3 | Anurag | 2021-08-31 | TOC |

| 4 | Asim | 2021-09-10 | CNS |

| 5 | Aryan | 2021-09-04 | DS |

| 6 | Aneesh | 2021-08-20 | HCI |

| 7 | Abhishek | 2021-08-13 | PYTHON |

+---------+----------+------------+----------+

--Update Trigger

delimiter //

create trigger after\_update

after update

on library for each row

begin

insert into library\_audit values(old.rollnum,old.name,old.issuedate,old.bookname,now());

end //

delimiter //

create trigger before\_update

before update

on library for each row

begin

insert into library\_audit values(old.rollnum,old.name,old.issuedate,old.bookname,now());

end //

update library set bookname = 'C++' where rollnum = 1;

mysql> select \* from library;

+---------+----------+------------+----------+

| rollnum | name | issuedate | bookname |

+---------+----------+------------+----------+

| 1 | Aditi | 2021-09-30 | C++ |

| 2 | Aditya | 2021-08-01 | DBMS |

| 3 | Anurag | 2021-08-31 | TOC |

| 4 | Asim | 2021-09-10 | CNS |

| 5 | Aryan | 2021-09-04 | DS |

| 6 | Aneesh | 2021-08-20 | HCI |

| 7 | Abhishek | 2021-08-13 | PYTHON |

+---------+----------+------------+----------+

mysql> select \* from library\_audit;

+---------+-------+------------+----------+---------------------+

| rollnum | name | issuedate | bookname | operationTime |

+---------+-------+------------+----------+---------------------+

| 1 | Aditi | 2021-08-15 | JAVA | 2021-09-30 13:14:36 |

+---------+-------+------------+----------+---------------------+

--Delete Trigger

delimiter //

create trigger after\_delete

after delete

on library for each row

begin

insert into library\_audit values(old.rollnum,old.name,old.issuedate,old.bookname,now());

end //

delimiter //

create trigger before\_delete

before delete

on library for each row

begin

insert into library\_audit values(old.rollnum,old.name,old.issuedate,old.bookname,now());

end //

delete from library where rollnum=7;

mysql> select \* from library;

+---------+--------+------------+----------+

| rollnum | name | issuedate | bookname |

+---------+--------+------------+----------+

| 1 | Aditi | 2021-09-30 | C++ |

| 2 | Aditya | 2021-08-01 | DBMS |

| 3 | Anurag | 2021-08-31 | TOC |

| 4 | Asim | 2021-09-10 | CNS |

| 5 | Aryan | 2021-09-04 | DS |

| 6 | Aneesh | 2021-08-20 | HCI |

+---------+--------+------------+----------+

mysql> select \* from library\_audit;

+---------+----------+------------+----------+---------------------+

| rollnum | name | issuedate | bookname | operationTime |

+---------+----------+------------+----------+---------------------+

| 1 | Aditi | 2021-08-15 | JAVA | 2021-09-30 13:14:36 |

| 7 | Abhishek | 2021-08-13 | PYTHON | 2021-09-30 13:18:28 |

+---------+----------+-----

Assignment 8:

Step 1:

CREATE DATABASE example\_db;

USE example\_db;

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL UNIQUE

);

select \* from users;

Step2: pip install mysql-connector-python

step3:

import mysql.connector

from mysql.connector import Error

# Establish the connection

def create\_connection():

connection = None

try:

connection = mysql.connector.connect(

host='localhost', # Your MySQL server host

user='root', # Replace with your MySQL username

password='sachin\_sql3059', # Replace with your MySQL password

database='example\_db' # Your database name

)

print("Connected to MySQL database.")

except Error as e:

print(f"Error: {e}")

return connection

# Execute a generic query

def execute\_query(connection, query):

cursor = connection.cursor()

try:

cursor.execute(query)

connection.commit()

print("Query executed successfully.")

except Error as e:

print(f"Error: {e}")

# CREATE - Add a new user

def add\_user(connection, name, email):

query = "INSERT INTO users (name, email) VALUES (%s, %s);"

cursor = connection.cursor()

try:

cursor.execute(query, (name, email))

connection.commit()

print(f"User '{name}' added successfully.")

except Error as e:

print(f"Error: {e}")

# READ - Fetch all users

def read\_users(connection):

cursor = connection.cursor()

cursor.execute("SELECT \* FROM users;")

return cursor.fetchall()

# UPDATE - Update user details

def update\_user\_email(connection, user\_id, new\_email):

query = "UPDATE users SET email = %s WHERE id = %s;"

cursor = connection.cursor()

try:

cursor.execute(query, (new\_email, user\_id))

connection.commit()

print(f"User with ID {user\_id} updated successfully.")

except Error as e:

print(f"Error: {e}")

# DELETE - Remove a user

def delete\_user(connection, user\_id):

query = "DELETE FROM users WHERE id = %s;"

cursor = connection.cursor()

try:

cursor.execute(query, (user\_id,))

connection.commit()

print(f"User with ID {user\_id} deleted successfully.")

except Error as e:

print(f"Error: {e}")

if \_name\_ == "\_main\_":

conn = create\_connection()

if conn:

# Create - Add a new user

add\_user(conn, "abhibasg", "abhi@example.com")

# Read - Retrieve all users

print("\nUsers in the database:")

users = read\_users(conn)

for user in users:

print(user)

# Update - Update user email

print("\nUpdating email for user with ID 1:")

update\_user\_email(conn, 1, "newemail@example.com")

# Read - Retrieve all users to verify the update

print("\nUsers after update:")

users = read\_users(conn)

for user in users:

print(user)

# Delete - Remove a user

print("\nDeleting user with ID 1:")

delete\_user(conn, 1)

# Read - Retrieve all users to verify the deletion

print("\nUsers after deletion:")

users = read\_users(conn)

for user in users:

print(user)

# Close the connection

conn.close()

Step4: python file\_name.py

GROUP B:

Assignment 1:

# MongoDB CRUD Operations Solution for "student" Collection

# =========================================================

db.createCollection(“student”);

# 1. Create a collection named "student" with the following documents:

db.student.insertMany([

{ "roll\_no": 1, "name": "Amit","addr":"Loni"},

{ "roll\_no": 2, "name": "Shashir", "addr": { "At":"Loni", "Tal": "Rahata", "Dist": "Ahemadnagar"} },

{ "roll\_no": 11, "name": "Sachin", "percent\_marks": 60.23, "addr": "Pune" },

{ "roll\_no": 3, "name": "Rahul", "addr": "Kolhar" }

]);

# Expected Output: Inserted 4 documents

# 2. Insert new documents into the "student" collection

db.student.insertMany([

{ "roll\_no": 4, "name": "Vikram", "addr": "Nashik" },

{ "roll\_no": 5, "name": "Anjali", "addr": "Mumbai" },

{ "roll\_no": 6, "name": "Priya", "addr": "Aurangabad" }

]);

# Expected Output: Inserted 3 documents

# 3. Retrieve all student details

db.student.find();

# 4. Display only the "name" field from the student collection

db.student.find({}, { "name": 1, "\_id": 0 });

# Expected Output:

# { "name": "Amit" }

# { "name": "Shashir" }

# { "name": "Sachin" }

# { "name": "Rahul" }

# { "name": "Vikram" }

# { "name": "Anjali" }

# { "name": "Priya" }

# 5. Display first 5 student details (use limit())

db.student.find().limit(5);

# Expected Output: First 5 documents in the collection

# 6. Fetch the remaining documents after the first 5 (use skip())

db.student.find().skip(5);

# Expected Output: Remaining documents after the first 5

# 7. Find details of students whose roll\_no is between 5 and 10

db.student.find({ "roll\_no": { "$gte": 5, "$lte": 10 } });

# Expected Output: Documents with roll\_no 5 and 6

# 8. Display student details where address is either "Loni" or "Pune"

db.student.find({ "$or": [{ "addr": "Loni" }, { "addr": "Pune" }] });

# Expected Output: Documents with addr "Loni" or "Pune"

# 9. Sort the student documents in ascending order by roll\_no

db.student.find().sort({ "roll\_no": 1 });

# Expected Output: All documents sorted by roll\_no in ascending order

# 10. Remove documents with roll\_no less than 8 and greater than 3

db.student.deleteMany({ "roll\_no": { "$lt": 8, "$gt": 3 } });

# Expected Output: Removes documents with roll\_no 4, 5, 6

# 11. Remove all documents with address as "Loni"

db.student.deleteMany({ "addr": "Loni" });

# Expected Output: Removes documents with addr as "Loni"

# 12. Add these details to a document where \_id=10 (Use save() equivalent)

db.student.updateOne(

{ "roll\_no": 10 },

{ "$set": { "name": "Ram", "addr": "Pune", "roll\_no": 60 } },

{ "upsert": true }

);

# Expected Output: Inserts or updates document with roll\_no 60

# 13. Update the address of roll\_no 1 to a nested document structure

db.student.updateOne(

{ "roll\_no": 1 },

{ "$set": { "addr": { "At": "Sangamner", "Tal": "Sangamner", "Dist": "Ahemadnagar" } } }

);

# Expected Output: Updates addr field for roll\_no 1

# 14. Create an index on the roll\_no field

db.student.createIndex({ "roll\_no": 1 });

# Expected Output: Creates an index on roll\_no field

# 15. Remove the "student" collection

db.student.drop();

# Expected Output: Drops the "student" collection

Assignment2:

db.createCollection(‘employee’);

// Insert sample data for employees

db.employee.insertMany([

{ "custID": "A123", "Amount": 500, "status": "A" },

{ "custID": "A123", "Amount": 250, "status": "A" },

{ "custID": "8212", "Amount": 200, "status": "A" },

{ "custID": "A123", "Amount": 300, "status": "D" },

{ "custID": "A123", "Amount": 1100, "status": "A" }

]);

// Output:

// Documents successfully inserted into the employee collection

// 3. Find total amount for each individual customer ID with status "A"

db.employee.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$custID", totalAmount: { $sum: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "totalAmount": 1850 }, { "\_id": "8212", "totalAmount": 200 }]

// 4. Find minimum amount for individual customer ID with status "A"

db.employee.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$custID", minAmount: { $min: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "minAmount": 250 }, { "\_id": "8212", "minAmount": 200 }]

// 5. Find maximum amount for individual customer ID with status "A"

db.employee.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$custID", maxAmount: { $max: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "maxAmount": 1100 }, { "\_id": "8212", "maxAmount": 200 }]

// 6. Find average amount for individual customer ID with status "A"

db.employee.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$custID", avgAmount: { $avg: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "avgAmount": 616.66666 }, { "\_id": "8212", "avgAmount": 200 }]

// 7. Amount of the first record for each customer ID with status "A"

db.employee.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$custID", firstAmount: { $first: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "firstAmount": 500 }, { "\_id": "8212", "firstAmount": 200 }]

// 8. Amount of the last record for each customer ID with status "A"

db.employee\_.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$custID", lastAmount: { $last: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "lastAmount": 1100 }, { "\_id": "8212", "lastAmount": 200 }]

// 9. Create array of amounts for each customer ID with status "A"

db.employee.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "amounts": [500, 250, 1100] }, { "\_id": "8212", "amounts": [200] }]

// 10. Total amount after sorting for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, Amount: 1 } },

{ $group: { \_id: "$custID", totalAmount: { $sum: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "totalAmount": 2150 }, { "\_id": "8212", "totalAmount": 200 }]

// 11. Average amount after sorting for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, Amount: 1 } },

{ $group: { \_id: "$custID", avgAmount: { $avg: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "avgAmount": 537.5 }, { "\_id": "8212", "avgAmount": 200 }]

// 12. Minimum amount after sorting for each customer ID

db.employee\_.aggregate([

{ $sort: { custID: 1, Amount: 1 } },

{ $group: { \_id: "$custID", minAmount: { $min: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "minAmount": 250 }, { "\_id": "8212", "minAmount": 200 }]

// 13. Maximum amount after sorting for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, Amount: 1 } },

{ $group: { \_id: "$custID", maxAmount: { $max: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "maxAmount": 1100 }, { "\_id": "8212", "maxAmount": 200 }]

// 14. Amount of first record after sorting for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, Amount: 1 } },

{ $group: { \_id: "$custID", firstAmount: { $first: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "firstAmount": 250 }, { "\_id": "8212", "firstAmount": 200 }]

// 15. Amount of last record after sorting for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, Amount: 1 } },

{ $group: { \_id: "$custID", lastAmount: { $last: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "lastAmount": 1100 }, { "\_id": "8212", "lastAmount": 200 }]

// Continue with steps 16-30 using the same aggregation patterns

// 16. After sorting records, create an array of amounts for each customer ID

db.employee\_.aggregate([

{ $sort: { custID: 1, Amount: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } }

]);

// Output: [{ "\_id": "A123", "amounts": [250, 500, 1100] }, { "\_id": "8212", "amounts": [200] }]

// 17. Without considering the first record, find total amount for each customer ID

db.employee\_.aggregate([

{ $sort: { \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, totalAmount: { $sum: { $slice: ["$amounts", 1,{ $size: "$amounts" }] } } } }

]);

// Output: [{ "\_id": "A123", "totalAmount": 1350 }, { "\_id": "8212", "totalAmount": 0 }]

// 18. Without considering the first record, find average amount for each customer ID

db.employee.aggregate([

{ $sort: { \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, avgAmount: { $avg: { $slice: ["$amounts", 1, { $size: "$amounts" }] } } } }

]);

// Output: [{ "\_id": "A123", "avgAmount": 450 }, { "\_id": "8212", "avgAmount": 0 }]

// 19. Without considering the first record, find maximum amount for each customer ID

db.employee.aggregate([

{ $sort: { \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, maxAmount: { $max: { $slice: ["$amounts", 1, { $size: "$amounts" }] } } } }

]);

// Output: [{ "\_id": "A123", "maxAmount": 1100 }, { "\_id": "8212", "maxAmount": 0 }]

// 20. Without considering the first record, find minimum amount for each customer ID

db.employee.aggregate([

{ $sort: { \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, minAmount: { $min: { $slice: ["$amounts", 1, { $size: "$amounts" }] } } } }

]);

// Output: [{ "\_id": "A123", "minAmount": 250 }, { "\_id": "8212", "minAmount": 0 }]

// 21. Without considering the first record, create an array of amounts for each customer ID

db.employee.aggregate([

{ $sort: { \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, amounts: { $slice: ["$amounts", 1, { $size: "$amounts" }] } } }

]);

// Output: [{ "\_id": "A123", "amounts": [250, 1100] }, { "\_id": "8212", "amounts": [] }]

// 22. Without considering the first record, find amount of the first record for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, firstAmount: { $arrayElemAt: ["$amounts", 1] } } }

]);

// Output: [{ "\_id": "A123", "firstAmount": 250 }, { "\_id": "8212", "firstAmount": null }]

// 23. Without considering the first record, find amount of the last record for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, \_id: 1 } }, // Sort by custID and then by \_id

{

$group: {

\_id: "$custID",

amounts: { $push: "$Amount" } // Collect amounts in an array

}

},

{

$project: {

\_id: 1,

lastAmount: {

$cond: {

if: { $gt: [{ $size: "$amounts" }, 1] }, // Check if there are more than 1 amount

then: { $arrayElemAt: ["$amounts", -1] }, // Get the last amount if valid

else: 0 // Otherwise, return 0 if there are not enough entries

}

}

}

}

]);

// Output: [{ "\_id": "A123", "lastAmount": 1100 }, { "\_id": "8212", "lastAmount": 0 }]

// 24. Considering the first 2 records only, find total amount for each customer ID

db.employee\_.aggregate([

{ $sort: { custID: 1, \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, totalAmount: { $sum: { $slice: ["$amounts", 0, 2] } } } }

]);

// Output: [{ "\_id": "A123", "totalAmount": 750 }, { "\_id": "8212", "totalAmount": 200 }]

// 25. Considering the first 2 records only, find average amount for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, avgAmount: { $avg: { $slice: ["$amounts", 0, 2] } } } }

]);

// Output: [{ "\_id": "A123", "avgAmount": 375 }, { "\_id": "8212", "avgAmount": 200 }]

// 26. Considering the first 2 records only, find minimum amount for each customer ID

db.employee\_.aggregate([

{ $sort: { custID: 1, \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, minAmount: { $min: { $slice: ["$amounts", 0, 2] } } } }

]);

// Output: [{ "\_id": "A123", "minAmount": 250 }, { "\_id": "8212", "minAmount": 200 }]

// 27. Considering the first 2 records only, find maximum amount for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, maxAmount: { $max: { $slice: ["$amounts", 0, 2] } } } }

]);

// Output: [{ "\_id": "A123", "maxAmount": 500 }, { "\_id": "8212", "maxAmount": 200 }]

// 28. Considering the first 2 records only, find amount of the first record for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, \_id: 1 } },

{ $group: { \_id: "$custID", amounts: { $push: "$Amount" } } },

{ $project: { \_id: 1, firstAmount: { $arrayElemAt: ["$amounts", 0] } } }

]);

// Output: [{ "\_id": "A123", "firstAmount": 500 }, { "\_id": "8212", "firstAmount": 200 }]

// 29. Considering the first 2 records only, find amount of the last record for each customer ID

db.employee.aggregate([

{ $sort: { custID: 1, \_id: 1 } }, // Sort by custID and then by \_id

{

$group: {

\_id: "$custID",

amounts: { $push: "$Amount" } // Collect amounts in an array

}

},

{

$project: {

\_id: 1,

lastAmount: {

$cond: {

if: { $gte: [{ $size: "$amounts" }, 2] }, // Check if there are more than 1 amount

then: { $arrayElemAt: ["$amounts", 1] }, // Get the last amount if valid

else: { $arrayElemAt: ["$amounts", 0] } // Otherwise, return 0 if there are not enough entries

}

}

}

}

]);

// Output: [{ "\_id": "A123", "lastAmount": 250 }, { "\_id": "8212", "lastAmount": 200 }]

//30 .Just by considering first 2 records create array of amount for individual cust Id

db.employee.aggregate([

{ $sort: { custID: 1, \_id: 1 } }, // Sort by custID and then by \_id

{

$group: {

\_id: "$custID",

amounts: { $push: "$Amount" } // Collect all amounts for each customer ID

}

},

{

$project: {

\_id: 1,

amounts: { $slice: ["$amounts", 0, 2] } // Get the first 2 amounts

}

}

]);

// Output :

[

{ \_id: '8212', amounts: [ 200 ] },

{ \_id: 'A123', amounts: [ 500, 250 ] }

]

Assignment3:

// To create collection we can also use => db.createCollection("bank"); show collections;

// Step 1: Insert Documents into the bank Collection

db.bank.insertMany([

{ 'custid': 'c001', 'custname': 'amir', 'bank': 'SBI', 'balance': 3000, 'location': 'Pune' },

{ 'custid': 'c002', 'custname': 'amol', 'bank': 'AXIS', 'balance': 4000, 'location': 'Pune' },

{ 'custid': 'c002', 'custname': 'amir', 'bank': 'AXIS', 'balance': 5000, 'location': 'Mumbai' },

{ 'custid': 'c003', 'custname': 'amit', 'bank': 'AXIS', 'balance': 5000, 'location': 'Nagpur' },

{ 'custid': 'c004', 'custname': 'amay', 'bank': 'AXIS', 'balance': 6000, 'location': 'Nashik' },

{ 'custid': 'c005', 'custname': 'amar', 'bank': 'HDFC', 'balance': 7000, 'location': 'Pune' },

{ 'custid': 'c006', 'custname': 'vijay', 'bank': 'HDFC', 'balance': 8000, 'location': 'Mumbai' }

]);

// Step 2: Define Map and Reduce Functions

var mapFunction = function() {

emit(this.bank, this.balance); // Emit each bank with its balance

};

var reduceFunction = function(key, values) {

return Array.sum(values); // Sum up all balances for each bank

};

// Step 3: Perform MapReduce and Store Output in bankMR Collection

db.bank.mapReduce(

mapFunction,

reduceFunction,

{ out: "bankMR" }

);

// Step 4: Display Results from bankMR Collection

db.bankMR.find().pretty();

// Step 5: Display the balance of HDFC bank only from bankMR collection

db.bankMR.find({ "\_id": "HDFC" }).pretty();

// Step 6: Display the balance of SBI bank only from bankMR collection

db.bankMR.find({ "\_id": "SBI" }).pretty();

// Step 7: Display the balance of AXIS bank only from bankMR collection

db.bankMR.find({ "\_id": "AXIS" }).pretty();

Assignment 4:

from pymongo import MongoClient

from pymongo.errors import ConnectionFailure

try:

# Connect to MongoDB

client = MongoClient("mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.3.3")

# Attempt to retrieve server information as a connectivity test

client.admin.command('ping')

print("Connected to MongoDB successfully!")

# Optionally, list all databases as an additional check

databases = client.list\_database\_names()

print("Available databases:", databases)

except ConnectionFailure as e:

print("Could not connect to MongoDB:", e)

# Create the database and collection if they don't already exist

db = client["practical"]

collection = db["Sample"]

def add\_document(data):

"""Add a document to the collection, ensuring no duplicates."""

try:

# Check if a document with the same 'name' and 'city' already exists

existing\_document = collection.find\_one({"name": data["name"], "city": data["city"]})

if existing\_document:

print("Duplicate document found. No new document added.")

else:

result = collection.insert\_one(data)

print(f"Document added with id: {result.inserted\_id}")

except Exception as e:

print(f"An error occurred while adding the document: {e}")

def delete\_document(query):

"""Delete a document matching the query."""

try:

result = collection.delete\_one(query)

if result.deleted\_count > 0:

print("Document deleted.")

else:

print("No document found matching the query.")

except Exception as e:

print(f"An error occurred while deleting the document: {e}")

def update\_document(query, new\_values):

"""Update a document matching the query."""

try:

result = collection.update\_one(query, {"$set": new\_values})

if result.modified\_count > 0:

print("Document updated.")

else:

print("No document found matching the query.")

except Exception as e:

print(f"An error occurred while updating the document: {e}")

def show\_all\_documents():

"""Show all documents in the collection."""

try:

documents = collection.find()

for doc in documents:

print(doc)

except Exception as e:

print(f"An error occurred while showing documents: {e}")

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Adding documents

add\_document({"name": "Alice", "age": 25, "city": "New York"})

add\_document({"name": "ankur", "age": 20, "city": "York"})

add\_document({"name": "sachin", "age": 15, "city": "New"})

# Attempting to add a duplicate document

add\_document({"name": "Alice", "age": 25, "city": "New York"})

# Showing all documents

print("All documents:")

show\_all\_documents()

# Updating a document

update\_document({"name": "Alice"}, {"age": 26})

# Deleting a document

delete\_document({"name": "Alice"})

# Showing all documents after deletion

print("All documents after deletion:")

show\_all\_documents()

# Close the MongoDB connection

client.close()

# use practical;

# db.createCollection("Sample");

# db.Sample.find();