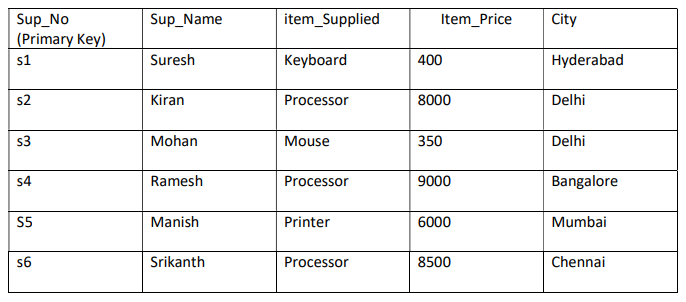
1. **CREATE A SUPPLIER TABLE AS SHOWN BELOW :**



**1. Write a query to retrieve all records from the Supplier table**

SELECT \*FROM Supplier;

**2. Write a query to retrieve the number of suppliers:**

SELECT COUNT(\*) AS TotalSuppliers FROM Supplier;

**3. Write a query to retrieve the highest item price:**

SELECT MAX(Item\_Price) AS HighestPrice FROM Supplier;

**4. Write a query to retrieve suppliers with item prices greater than 5000:**

SELECT \*FROM Supplier WHERE Item\_Price > 5000;

**5. Write a query to update the city for a specific supplier:**

UPDATE Supplier SET City = 'NewCity' WHERE Sup\_No = 's1'; SELECT \*FROM Supplier;

**6. Write a query to delete a supplier record based on supplier number:**

DELETE FROM Supplier WHERE Sup\_No = 's2'; SELECT \*FROM Supplier;

**7. Write a query to display Supplier numbers and Supplier name whose name starts with “R”:**

SELECT Sup\_No, Sup\_Name FROM Supplier WHERE Sup\_Name LIKE 'R%';

**8. Write a query to display the name of suppliers who supply Processors and whose city is Delhi:**

SELECT Sup\_Name FROM Supplier WHERE Item\_Supplied = 'Processor' AND City =’Delhi';

**9. Write a query to increase the price of Keyboard by 200:**

UPDATE Supplier SET Item\_Price = Item\_Price + 200 WHERE Item\_Supplied = 'Keyboard';

**10. Write a query to display supplier numbers, Supplier names and item price for suppliers in delhi in the ascending order of item price:**

SELECT Sup\_No, Sup\_Name, Item\_Price FROM Supplier WHERE City = 'Delhi' ORDER BY Item\_Price ASC;

**11. Write a query to add a new column called CONTACTNO:**

ALTER TABLE Supplier ADD CONTACTNO VARCHAR(20); SELECT \* FROM Supplier;

**12. Write a query to delete the record whose item price is the lowest of all the items supplied:**

DELETE FROM Supplier WHERE Item\_Price = (SELECT MIN(Item\_Price) FROM Supplier); SELECT \* FROM Supplier;

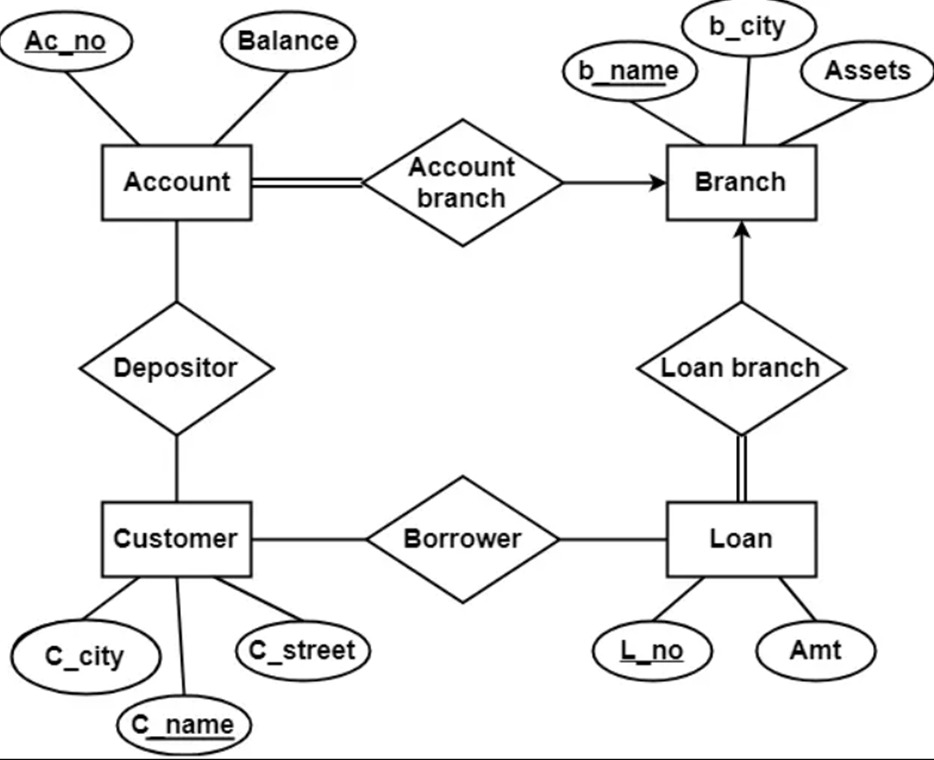
**13. Write a query to display the records in the descending order of item price for each item supplied:**

SELECT \*FROM Supplier ORDER BY Item\_Supplied, Item\_Price DESC;

**14. Write a query to display the records of suppliers who supply items other than Processor or Keyboard:**

SELECT \*FROM Supplier WHERE Item\_Supplied NOT IN ('Processor', 'Keyboard');

**2. CREATE THE TABLE ACCORDING TO THE FOLLOWING ER DIAGRAM**



Account (Ac\_no, Balance, b\_name)

Branch (b\_name , b\_city , Assets)

Loan (L\_no , Amt , b\_name)

Borrower (C\_name , L\_no)

Customer (C\_name , C\_street , C\_city)

Depositor (C\_name , Ac\_no)

**TABLE CREATION**

* CREATE TABLE Branch( b\_name VARCHAR(50) PRIMARY KEY, b\_city VARCHAR(50), Assets DECIMAL(15,2));
* CREATE TABLE Customer ( C\_name VARCHAR(50) PRIMARY KEY, C\_street VARCHAR(100), C\_city VARCHAR(50));
* CREATE TABLE Account ( Ac\_no INT PRIMARY KEY, Balance DECIMAL(10,2), b\_name VARCHAR(50), FOREIGN KEY (b\_name) REFERENCES Branch(b\_name));
* CREATE TABLE Loan ( L\_no INT PRIMARY KEY, Amt DECIMAL(10,2), b\_name VARCHAR(50), FOREIGN KEY (b\_name) REFERENCES Branch(b\_name));
* CREATE TABLE Borrower( C\_name VARCHAR(50), L\_no INT, FOREIGN KEY (C\_name) REFERENCES Customer(C\_name), FOREIGN KEY (L\_no) REFERENCES Loan(L\_no));
* CREATE TABLE Depositor ( C\_name VARCHAR(50), Ac\_no INT, FOREIGN KEY (C\_name) REFERENCES Customer(C\_name), FOREIGN KEY (Ac\_no) REFERENCES Account(Ac\_no));

**INSERT DATA**

* INSERT INTO Branch (b\_name, b\_city, Assets) VALUES ('Branch3', 'City3', 800000.00), ('Branch4', 'City4', 900000.00);
* INSERT INTO Customer (C\_name, C\_street, C\_city) VALUES ('Customer7', 'Street7', 'City5'), ('Customer8', 'Street8', 'City6'), ('Customer9', 'Street9', 'City5');
* INSERT INTO Account (Ac\_no, Balance, b\_name) VALUES (7, 7000.00, 'Branch3'), (8, 8000.00, 'Branch4'), (9, 9000.00, 'Branch4');
* INSERT INTO Loan (L\_no, Amt, b\_name) VALUES (107, 50000.00, 'Branch3'), (108, 60000.00, 'Branch4'), (109, 70000.00, 'Branch3');
* INSERT INTO Borrower (C\_name, L\_no) VALUES ('Customer7', 107), ('Customer8', 108), ('Customer9', 109);
* INSERT INTO Depositor (C\_name, Ac\_no) VALUES ('Customer7', 7), ('Customer8', 8), ('Customer9',9);

select \* from Branch;

select \* from Customer;

select \* from Account;

select \* from Loan;

select \* from Borrower;

select \* from Depositor;

SELECT b\_name FROM Branch;

SELECT C\_name FROM Customer;

SELECT Ac\_no, Balance FROM Account;

**QUERIES:**

1. **Retrieve the average balance of all accounts:**

SELECT AVG(Balance) AS AverageBalance FROM Account;

1. **Retrieve the maximum loan amount:**

SELECT MAX(Amt) AS MaximumLoanAmount FROM Loan;

1. **Retrieve the maximum loan amount:**

SELECT MAX(Amt) AS MaximumLoanAmount FROM Loan;

1. **Retrieve the names of customers residing in City5:**

SELECT C\_name FROM Customer WHERE C\_city = 'City5';

1. **Retrieve the account numbers and balances of accounts with a balance greater than 5000**

SELECT Ac\_no, Balance FROM Account WHERE Balance > 5000;

1. **Add a new column called "Email" to the "Customer" table:**

ALTER TABLE Customer ADD Email VARCHAR(100);

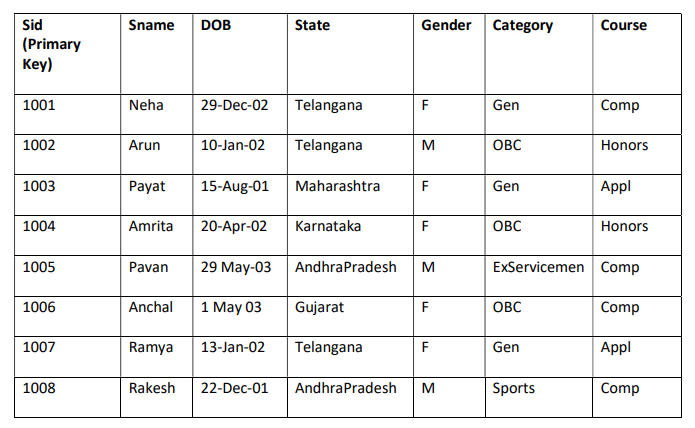
1. **Rename the "b\_city" column in the "Branch" table to "BranchCity":**

ALTER TABLE Branch RENAME COLUMN b\_city TO BranchCity;

1. **Modify the data type and length of the "C\_street" column in the "Customer" table:**

ALTER TABLE Customer ALTER COLUMN C\_street VARCHAR2(200);

**3. CREATE THE FOLLOWING TABLE. (student)**



1. **Write sql query to display the students who are not from Telangana or Andhra Pradesh:**

SELECT \* FROM students WHERE State NOT IN ('Telangana', 'AndhraPradesh');

1. **Write sql query to display all the female students enrolled under Comp course and who belong to OBC**

SELECT \*FROM students

WHERE Course = 'Comp' AND Gender = 'F' AND Category = 'OBC';

1. **Write sql query to display the student ids, names, and their present age**

select sid, sname, extract(year from sysdate) - extract(year from dob) as age

from student;

1. **Write sql query to display the students in the ascending order of their names for each Course SELECT Sid, Sname, Course**

FROM Students

ORDER BY Course, Sname acs;

1. **Write sql query to add two new columns Contact no and Email to the existing fields:**

ALTER TABLE Students

ADD (ContactNo VARCHAR(20),

Email VARCHAR(100));

1. **Write sql query to enter the value into the columns contact no and email:**

UPDATE Students SET ContactNo = '1234567890', Email = 'neha@gmail.com'WHERE Sid = 1001;

UPDATE Students SET ContactNo = '9049315885', Email = 'arun@gmail.com' WHERE Sid = 1002;

UPDATE Students SET ContactNo = '8855994426', Email = 'payat@gmail.com' WHERE Sid = 1003;

UPDATE Students SET ContactNo = '7958647712', Email = Amritha@gmail.com' WHERE Sid = 1004;

UPDATE Students SET ContactNo = '6958224564', Email = 'pavan@gmail.com' WHERE Sid = 1005;

UPDATE Students SET ContactNo = '6932001452', Email = 'neha@gmail.com' WHERE Sid = 1006;

UPDATE Students SET ContactNo = '7896589942', Email = 'ramya@gmail.com' WHERE Sid = 1007;

UPDATE Students SET ContactNo = '6325894422', Email = 'rakesh@gmail.com' WHERE Sid = 1008;

1. **Write sql query to display all the Student names where the length of the name is 5 characters:**

SELECT Sname FROM Students WHERE Length(Sname) = 5;

1. **Write sql query to delete all the students records who have enrolled for Comp course and who are born after 2002:**

DELETE FROM Students

WHERE Course = 'Comp' AND extract(year from dob) > 2002;

1. **Write sql query to display all the Student names prefixed with Mr. Ms. Based on Gender column:**

SELECT

CASE UPPER(Gender)

WHEN 'M' THEN 'Mr. ' || SName

WHEN 'F' THEN 'Ms. ' || SName

ELSE SName

END

AS PrefixedName

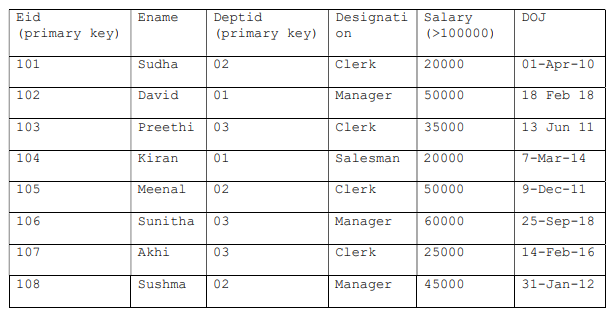
**FROM Student;**

**4**. **CREATE THE TWO TABLES AS SHOWN BELOW WITH THE GIVEN CONSTRAINTS:**

**Table name: Employee**

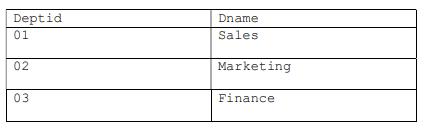
**Constraints Ed is Primary key and Dept id is foreign key**

**Salary should not be less than 10000**



**Tablename: Department**

Constraints Deptid Primary key and Dname is NOT NULL

****

1. **Write sql query to display all the employees who earn more than average salary of all the employees in the company:**

SELECT \* FROM Employees

WHERE Salary > (SELECT AVG (Salary) FROM Employees);

1. **Write sql query to display the fields Eid, Ename and Dname:**

SELECT Employees.Eid, Employees.Ename, Departments.Dname

FROM Employees

JOIN Departments ON Employees.Deptid = Departments.Deptid;

1. **Write sql query to sort the employee table in the descending order of salaries:**

SELECT \* FROM Employees

ORDER BY Salary DESC;

1. **Write sql query to list all the job designations in the employee table without repetitions:**

SELECT DISTINCT Designation

FROM Employees;

1. **Write sql query to display all the employee details Department wise and in the ascending order of their salaries:**

SELECT \* FROM Employees

ORDER BY Deptid ASC, Salary ASC;

1. **Write sql query to display all the clerks in Deptld D2:**

SELECT \* FROM Employees

WHERE Designation = 'Clerk' AND Deptid = '02';

1. **Write sql query to display all the employees who joined in the year 2011:**

select \*from employee

where extract(year from doj)=2011;

1. **Write sql query to display all the employees who joined in the month of February:**

SELECT \* FROM Employees

WHERE MONTH(DOJ) = 2;

1. **Write sql query to display all the employees whose salary is between 30000 and 45000:**

SELECT \* FROM Employees

WHERE Salary BETWEEN 30000 AND 45000;

1. **Write sql query to display all the employee details along with their work experience in the** **company till current date:**

SELECT

Eid,

Ename,

Deptid,

Designation,

Salary,

DOJ,

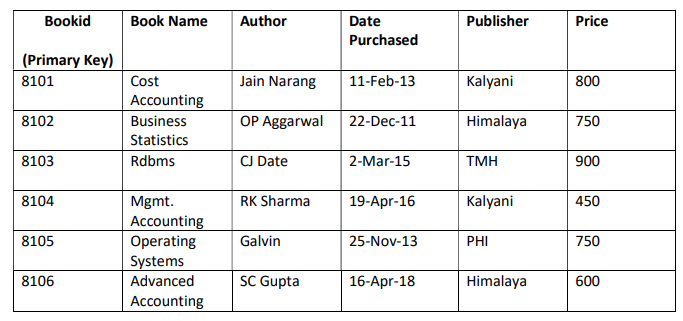
(EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOJ)) AS Experience

FROM Employee;

**5. CREATE A TABLE FOR LIBRARY INFORMATION**

**Table name: Library (book)**

Constraints: Bookld is **primary key** and Book Name is **NOT NULL**



**CREATE TABLE**

CREATE TABLE BookDetails (

BookId INT IDENTITY(1,1) NOT NULL PRIMARY KEY,

BookName VARCHAR(255) NOT NULL,

Author VARCHAR(255) NOT NULL,

DatePurchased DATE NOT NULL,

Publisher VARCHAR(255) NOT NULL,

Price INT NOT NULL

);

**INSERT QUERYS :**

INSERT INTO BookDetails (BookName, Author, DatePurchased, Publisher, Price)VALUES ('Cost Accounting', 'Jain Narang', '11-Feb-13', 'Kalyani', 800)

INSERT INTO BookDetails VALUES('Business Statistics', 'OP Aggarwal', '22-Dec-11', 'Himalaya', 750)

INSERT INTO BookDetails VALUES('Rdbms', 'CJ Date', '2-Mar-15', 'TMH', 900)

INSERT INTO BookDetails VALUES('Mgmt. Accounting', 'RK Sharma', '19-Apr-16', 'Kalyani', 450)

INSERT INTO BookDetails VALUES('Operating Systems', 'Galvin', '25-Nov-13', 'PHI', 750)

INSERT INTO BookDetails VALUES('Advanced Accounting', 'SC Gupta', '16-Apr-18', 'Himalaya', 600)

1. **Write sql query to display the list of authors from Himalaya publications:**

SELECT Author

FROM BookDetails WHERE Publisher = 'Himalaya';

1. **Write sql query to display the total cost of books purchased Publisher wise:**

SELECT Publisher, SUM(Price) AS TotalCost

FROM BookDetails

GROUP BY Publisher;

1. **Write sql query to count the total number of books under kalyani publications:**

SELECT COUNT(\*) AS TotalBooks

FROM BookDetails WHERE Publisher = 'Kalyani';

1. **Write a sql query to display the books in ascending order of date purchased:**

SELECT BookName, Author, DatePurchased, Publisher, Price

FROM BookDetails

ORDER BY DatePurchased ASC;

1. **Write sql query to create an index on the field’s book name and author:**

CREATE INDEX idx\_book\_name\_author ON BookDetails (BookName, Author);

1. **Write sql query to display the books whose price is between 500 and 700**

SELECT BookName, Author, Price FROM BookDetails

WHERE Price BETWEEN 500 AND 700;

1. **Write sql query to increase the price of all the books by 200 for publishers other than Himalaya or Kalyani:**

UPDATE BookDetails

SET Price = Price + 200

WHERE Publisher NOT IN ('Himalaya', 'Kalyani');

select \* from BookDetails;

1. **Write sql query to display the book details where author name contains the name Sharma:**

SELECT BookName, Author, Publisher, Price

FROM BookDetails WHERE Author LIKE '%Sharma%';

**Creating a view of bookid and bookname whose publisher is himalaya**

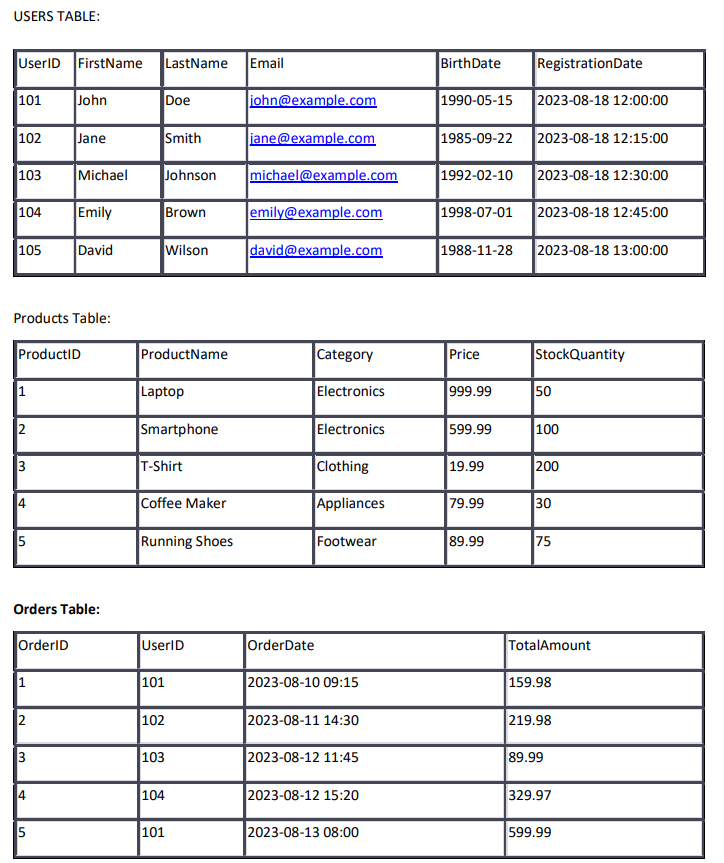
CREATE VIEW BookDetailsByHimalaya

AS SELECT BookID, BookName

FROM BookDetails WHERE Publisher = 'Himalaya';

SELECT \* FROM BookDetailsByHimalaya;

**6. QUERIES ON EMP AND DEPT TABLES**



**CREATE TABLE**

CREATE TABLE Users (

UserID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100) UNIQUE,

BirthDate DATE,

RegistrationDate DATETIME DEFAULT CURRENT\_TIMESTAMP

);

INSERT INTO Users (UserID, FirstName, LastName, Email, BirthDate) VALUES

(101, 'John', 'Doe', 'john@example.com', '1990-05-15'),

(102, 'Jane', 'Smith', 'jane@example.com', '1985-09-22'),

(103, 'Michael', 'Johnson', 'michael@example.com', '1992-02-10'),

(104, 'Emily', 'Brown', 'emily@example.com', '1998-07-01'), (

105, 'David', 'Wilson', 'david@example.com', '1988-11-28');

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2),

StockQuantity INT

);

INSERT INTO Products (ProductID, ProductName, Category, Price, StockQuantity) VALUES

(1, 'Laptop', 'Electronics', 999.99, 50),

(2, 'Smartphone', 'Electronics', 599.99, 100),

(3, 'T-Shirt', 'Clothing', 19.99, 200),

(4, 'Coffee Maker', 'Appliances', 79.99, 30),

(5, 'Running Shoes', 'Footwear', 89.99, 75);

CREATE TABLE Orders2 (

OrderID INT PRIMARY KEY,

UserID INT,

OrderDate DATETIME, -- Use DATETIME instead of TIMESTAMP

TotalAmount DECIMAL(10, 2),

FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

INSERT INTO Orders2 (OrderID, UserID, OrderDate, TotalAmount) VALUES

(1, 101, '2023-08-10 09:15', 159.98),

(2, 102, '2023-08-11 14:30', 219.98),

(3, 103, '2023-08-12 11:45', 89.99),

(4, 104, '2023-08-12 15:20', 329.97),

(5, 101, '2023-08-13 08:00', 599.99);

1. **Retrieve the first and last names of all users from the "Users" table:**

SELECT FirstName, LastName

FROM Users;

1. **Retrieve the total number of orders in the "Orders" table:**

SELECT COUNT(\*) AS TotalOrders

FROM Orders;

1. **Retrieve the names of all products in the "Electronics" category:**

SELECT ProductName

FROM Products

WHERE Category = 'Electronics';

1. **Retrieve the total price of all orders from the "Orders" table:**

SELECT SUM(TotalAmount) AS TotalPrice

FROM Orders;

1. **Retrieve the top 5 most expensive products from the "Products" table:**

SELECT \*

FROM (

SELECT productname, price

FROM product

ORDER BY price DESC

)

WHERE ROWNUM <= 5;

1. **Retrieve the orders placed by the user with UserID 101, including their order dates and total** **amounts:**

SELECT OrderID, OrderDate, TotalAmount

FROM Orders

WHERE UserID = 101;

1. **Retrieve the average stock quantity of products in each category:**

SELECT Category, AVG(StockQuantity) AS AvgStockQuantity

FROM Products GROUP BY Category;

1. **Retrieve the number of orders placed by each user, ordered by the highest number first:**

SELECT UserID, COUNT(\*) AS OrderCount FROM Orders GROUP BY UserID ORDER BY OrderCount DESC;

1. **Retrieve the users who have never placed an order, along with their registration dates:**

SELECT Users.UserID, Users.FirstName, Users.LastName, Users.RegistrationDate

FROM Users

LEFT JOIN Orders ON Users.UserID = Orders.UserID

WHERE Orders.OrderID IS NULL;

1. **Calculate the difference in days between the order date and the registration date for each order:**

SELECT Orders.OrderID, Users.FirstName, Users.LastName, DATEDIFF(DAY, Users.RegistrationDate, Orders.OrderDate) AS DaysDifference

FROM Orders

JOIN Users ON Orders.UserID = Users.UserID;

**PL/SQL**

**7. WRITE A PL/SQL PROGRAM TO FIND ADDITION OF TWO NUMBERS.**

SQL> declare

a number(6);

b number(6);

c number(6);

begin

a:=&a;

b:=&b;

c:=a+b;

dbms\_output.put\_line('sum is'||c);

end;

**OUTPUT:**

Enter value for a: 43

old 6: a:=&a;

new 6: a:=43;

Enter value for b: 5

old 7: b:=&b;

new 7: b:=5;

sum is48

**8.PL/SQL to find the factorial of a number**

SQL> declare

n number (2);

i number(2):=1;

f number(3):=1;

begin

n:=5;

loop

exit when i>n;

f:=f\*i;

i:=i+1;

end loop;

dbms\_output.put\_line('factorial is' ||f);

end;

**OUTPUT:**

factorial is120

**9. PL/SQL for demo on for loop**

SQL> declare

i number(2);

begin

for i in 1..10

loop

dbms\_output.put\_line('i='||i);

end loop;

end;

**OUTPUT:**

i=1

i=2

i=3

i=4

i=5

i=6

i=7

i=8

i=9

i=10

**10.PL/SQL for case structure**

DECLARE

a NUMBER(10);

b NUMBER(10);

res NUMBER(30);

op VARCHAR2(1);

BEGIN

a := &a;

b := &b;

op := '&op'; -- use quotes to handle character input

CASE op

WHEN '+' THEN res := a + b;

WHEN '-' THEN res := a - b;

WHEN '\*' THEN res := a \* b;

ELSE res := 0;

END CASE;

DBMS\_OUTPUT.PUT\_LINE(a || ' ' || op || ' ' || b || ' = ' || res);

END;

/

**OUTPUT:**

SQL> /

3+4=7

**11. PL/SQL for simple loop**

SQL> declare

i number(2);

begin

i:=1;

loop

dbms\_output.put\_line('i='||i);

exit when i=5;

i:=i+1;

end loop;

end;

**OUTPUT:**

i=1

i=2

i=3

i=4

i=5

**12. PL/SQL to increase the value by 10**

SQL> declare

n number(2):=5;

begin

n:=n+10;

dbms\_output.put\_line('value of n is:'||n);

end;

**OUTPUT:**

value of n is:15

**13. write a PL/SQL program to implement all arithmetic operations**

SQL> declare

a number(6);

b number(6);

c number(6);

d number(6);

e number(6);

f number(6);

begin

a:=8;

b:=4;

c:=a+b;

d:=a-b;

e:=a\*b;

f:=a/b;

dbms\_output.put\_line('sum is'||c);

dbms\_output.put\_line('sub is'||d);

dbms\_output.put\_line('mul is'||e);

dbms\_output.put\_line('div is'||f);

end;

**OUTPUT:**

sum is12

sub is4

mul is32

div is2

**14. write A PL/SQL program to find square square, cube, double of a number**

SQL> declare

a number(6);

s number(6);

c number(6);

begin

a:=55;

s:=a\*a;

c:=a\*a\*a;

d=a\*2;

dbms\_output.put\_line('sq is'||s);

dbms\_output.put\_line('cube is'||c);

end;

**OUTPUT:**

sq is3025

cube is166375

**15. write a PL/SQL program to swap two numbers**

SQL> declare

a number(6);

b number(6);

temp number(6);

begin

a:=4;

b:=5;

temp:=a;

a:=b;

b:=temp;

dbms\_output.put\_line('after swapping '||’a=’||a||’b=’||b);

end;

**OUTPUT:**

after swapping a=5 b=4

**16. PL/SQL program to find multiplication table**

SQL> declare

n number(3):=5;

i number(3):=1;

p number(3);

begin

loop

p:=n\*i;

dbms\_output.put\_line(n||'\*'||i||'='||p);

exit when i>10;

i:=i+1;

end loop;

end;

**OUTPUT:**

5\*1=5

5\*2=10

5\*3=15

5\*4=20

5\*5=25

5\*6=30

5\*7=35

5\*8=40

5\*9=45

5\*10=50

**17. PL/SQL to determine whether a year is leap year or not**

DECLARE

year NUMBER(4);

BEGIN

year := &year; -- take input from user

IF (MOD(year, 400) = 0) THEN

DBMS\_OUTPUT.PUT\_LINE(year || ' is a leap year.');

ELSIF (MOD(year, 100) = 0) THEN

DBMS\_OUTPUT.PUT\_LINE(year || ' is not a leap year.');

ELSIF (MOD(year, 4) = 0) THEN

DBMS\_OUTPUT.PUT\_LINE(year || ' is a leap year.');

ELSE

DBMS\_OUTPUT.PUT\_LINE(year || ' is not a leap year.');

END IF;

END;

/

**18. PL/SQL to delete an item whose number=4**

SQL> select \* from item;

ITEMNUM ITEMNAME

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

1 soap

2 watch

3 water

4 water

5 earring

SQL>

Declare

Rows\_deleted number(10);

Begin

Delete from item

Where item\_num=4;

Rows\_deleted:=sql%rowcount

If rows\_deleted>0 then

Dbms\_output.put\_line(‘sucessfully row has been deleted’);

Else

Dbms\_output.put\_line(‘no rows selected’);

End if;

End;

**OUTPUT:**

PL/SQL procedure successfully completed.

SQL> select \* from item;

ITEMNUM ITEMNAME

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

1 soap

2 watch

3 watch

5 earring

**19. PL/SQL program for inverting a number**

DECLARE

original\_num NUMBER := 12345;

inverted\_num NUMBER := 0;

remainder NUMBER(10);

BEGIN

WHILE

original\_num > 0 LOOP

remainder := MOD(original\_num, 10);

inverted\_num := inverted\_num \* 10 + remainder;

original\_num := (original\_num - remainder) / 10;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Inverted number: ' || inverted\_num);

END;

**OUTPUT:**

the given number is43

the inverted number is34

**ar**

**20. PL/SQL program to calculate the area of a circle for value of radius ranging from 3 to 7**

DECLARE

radius NUMBER;

area NUMBER;

BEGIN

FOR radius IN 3..7 LOOP

area := 3.14159 \* radius \* radius; -- Area = π \* r^2

DBMS\_OUTPUT.PUT\_LINE('Radius: ' || radius || ' -> Area: ' || area);

END LOOP;

END;

/**OUTPUT:**

area is27

area is48

area is75

area is108

area is147

**21. write a program to print empno, ename ,job and salary of an employee given empno**

DECLARE

v\_empno NUMBER := &eid;

BEGIN

FOR rec IN (SELECT eid, ename, , salary,commission

FROM employeetable

WHERE eid = v\_empno)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee Number: ' || rec.eid);

DBMS\_OUTPUT.PUT\_LINE('Employee Name : ' || rec.ename);

DBMS\_OUTPUT.PUT\_LINE('commision ' || rec.commission);

DBMS\_OUTPUT.PUT\_LINE('salary : Rs. ' || rec.salary);

END LOOP;

END;

/

**OUTPUT:**

SQL> /

7839 king president 5500

**22. PL/SQL block that will increase the salary by 100 if salary is greater than 1000 for a given empno.**

DECLARE

v\_empno NUMBER := 101; -- Enter the employee number here

BEGIN

UPDATE employeetable

SET salary = salary + 100

WHERE eid = v\_empno

AND salary > 1000;

-- Optional: display confirmation

DBMS\_OUTPUT.PUT\_LINE('Salary updated for employee number: ' || v\_empno);

END;

/

**OUTPUT:**

increased salary is3075

**23. PL/SQL block using the cursor to display details of all employees from emp table whose sum of sal and comm. Is>2000**

declare

mempno emp.empno%type;

mename emp.ename%type;

msal emp.sal%type;

cursor cl is

select empno, ename, sal+nvl(comm,0)from emp where sal+nvl(comm,0)>2000;

begin

open cl;

dbms\_output.put\_line('employeeeno'||'empname'||'netsalary');

loop

fetch cl into mempno,mename,msal;

dbms\_output.put\_line(mempno||''||mename||''||msal);

exit when cl%notfound;

end loop;

close cl;

end;

SQL> /

employeeenoempnamenetsalary

7499allen2060

7566jones2800

7654martin2775

7698blake2800

7782clark2800

7788scott3300

7839king5500

7902ford3300

7902ford3300

**24. PL/SQL Program to update salary of an employee whose empno and increment mentioned**.

SQL> create or replace procedure empupdate(eno emp.empno%type,incr number)as

begin

update emp set sal=sal+1000 where empno=7788;

commit;

dbms\_output.put\_line('updated successfully');

end;

**OUTPUT:**

Procedure created.

updated successfully

PL/SQL procedure successfully completed.

SQL> select ename,sal from emp where empno=7788;

ENAME SAL

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

SCOTT 4000

**25. PL/SQL to block which accepts the empno from the user and display the details of the employee. When the user enters an empno that is not in the emp table then the PL/SQL block must display an appropriate message to the user**

SQL> declare

mempno emp.empno%type;

begin

select empno into mempno from emp where empno=7698;

dbms\_output.put\_line(mempno||'found');

exception

when no\_data\_found then

dbms\_output.put\_line('employee number is not found');

end;

**OUTPUT:**

7698found

**26. PL/SQL that creates a trigger that inserts or update the values of ename, job as lower case string**

SQL> create or replace trigger lowername

before insert or update on emp for each row

begin

:new.ename:=lower(:new.ename);

:new.job:=lower(:new.job);

end;

**OUTPUT:**

Trigger created.

SQL> update emp set job='MANAGER' where ename='SMITH';

1 row updated.

SQL> select \* from emp;

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

---------- ---------- --------- ---------- --------- ---------- -------------------- --------------------7369 smith manager 7902 17-DEC-80 800 - 20

**27. PL/SQL to implement standalone procedure in INOUT mode**

declare

a number;

procedure squarenum(x IN OUT number) IS

begin

x:=x\*x;

end;

begin

a:=23;

squarenum(a);

dbms\_output.put\_line('square of 23'||a);

end;

**OUTPUT:**

square of 23529

Statement processed

**28. PL/SQL to create a function to findout total number of Employees in the emp table**

create or replace function totalemp

return number is total number(3):=0;

begin

select count(\*) into total from emp;

return total;

end;

**OUTPUT:**

Function created.

**29. PL/SQL to handle ZERO\_DIVIDE Exception**

declare

x number:=5;

y number:=0;

begin

dbms\_output.put\_line('division is'||x/y);

exception

when ZERO\_DIVIDE then

dbms\_output.put\_line('check the denominator');

end;

**OUTPUT:**

check the denominator

**MONGO DB**

**30. PRACTICE CRUD (CREATE, READ, UPDATE, AND DELETE) OPERATIONS ON THE MONGODB**

**Create Operation**



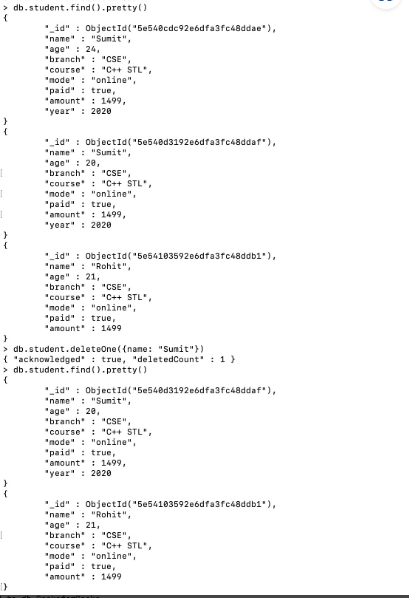
**Read Operation**



**Update Operation**



**Delete Operation**



db.students.insertOne([

{ id: 1, name: 'Ryan', gender: 'M' }

]);

db.students.find();

OUTPUT

mycompiler\_mongodb> ... ... {

acknowledged: true,

insertedId: ObjectId('68b54c88b4c22dfa996b128c')

}

mycompiler\_mongodb> [

{

'0': { id: 1, name: 'Ryan', gender: 'M' },

\_id: ObjectId('68b54c88b4c22dfa996b128c')

}

]

mycompiler\_mongodb>

[Execution complete with exit code 0]

db.students.insertMany([

{ id: 1, name: 'Ryan', gender: 'M' },

{ id: 2, name: 'sita', gender: 'F' }

]);

db.students.find();

OUTPUT

mycompiler\_mongodb> ... ... ... {

acknowledged: true,

insertedIds: {

'0': ObjectId('68b54d47784ce11ab56b128c'),

'1': ObjectId('68b54d47784ce11ab56b128d')

}

}

mycompiler\_mongodb> [

{

\_id: ObjectId('68b54d47784ce11ab56b128c'),

id: 1,

name: 'Ryan',

gender: 'M'

},

{

\_id: ObjectId('68b54d47784ce11ab56b128d'),

id: 2,

name: 'sita',

gender: 'F'

db.students.insertMany([

{ id: 1, name: 'Ryan', gender: 'M' },

{ id: 2, name: 'sita', gender: 'F' }

]);

db.students.findOne({gender:'F'});

Output

mycompiler\_mongodb> ... ... ... {

acknowledged: true,

insertedIds: {

'0': ObjectId('68b54dd82354dabdc86b128c'),

'1': ObjectId('68b54dd82354dabdc86b128d')

}

}

mycompiler\_mongodb> {

\_id: ObjectId('68b54dd82354dabdc86b128d'),

id: 2,

name: 'sita',

gender: 'F'

}

mycompiler\_mongodb>

[Execution complete with exit code 0]

db.students.insertMany([

{ id: 1, name: 'Ryan', gender: 'M' ,group:'Bsc',age:21},

{ id: 2, name: 'Rian', gender: 'M' ,group:'Bsc',age:23},

{ id: 1, name: 'sita', gender: 'F' ,group:'Bcom',age:22},

]);

db.students.find();

OUTPUT:

mycompiler\_mongodb> ... ... ... ... {

acknowledged: true,

insertedIds: {

'0': ObjectId('68b5534a37f0f634786b128c'),

'1': ObjectId('68b5534a37f0f634786b128d'),

'2': ObjectId('68b5534a37f0f634786b128e')

}

}

mycompiler\_mongodb> [

{

\_id: ObjectId('68b5534a37f0f634786b128c'),

id: 1,

name: 'Ryan',

gender: 'M',

group: 'Bsc',

age: 21

},

{

\_id: ObjectId('68b5534a37f0f634786b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bsc',

age: 23

},

{

\_id: ObjectId('68b5534a37f0f634786b128e'),

id: 1,

name: 'sita',

gender: 'F',

group: 'Bcom',

age: 22

}

]

mycompiler\_mongodb>

[Execution complete with exit code 0]

db.students.insertMany([

{ id: 1, name: 'Ryan', gender: 'M' ,group:'Bsc',age:21},

{ id: 2, name: 'Rian', gender: 'M' ,group:'Bsc',age:23},

{ id: 3, name: 'sita', gender: 'F' ,group:'Bcom',age:22},

]);

db.students.find({age:23}).pretty();

OUTPUT

mycompiler\_mongodb> ... ... ... ... {

acknowledged: true,

insertedIds: {

'0': ObjectId('68b554855446f110eb6b128c'),

'1': ObjectId('68b554855446f110eb6b128d'),

'2': ObjectId('68b554855446f110eb6b128e')

}

}

mycompiler\_mongodb> [

{

\_id: ObjectId('68b554855446f110eb6b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bsc',

age: 23

}

]

mycompiler\_mongodb>

[Execution complete with exit code 0]

db.students.insertMany([

{ id: 1, name: 'Ryan', gender: 'M' ,group:'Bsc',age:21},

{ id: 2, name: 'Rian', gender: 'M' ,group:'Bsc',age:23},

{ id: 1, name: 'sita', gender: 'F' ,group:'Bcom',age:22},

]);

db.students.find();

db.students.updateOne({name:"Rian"},{$set:{age: 25}});

db.students.find();

OUTPUT:

mycompiler\_mongodb> ... ... ... ... {

acknowledged: true,

insertedIds: {

'0': ObjectId('68b55acbf311486d1a6b128c'),

'1': ObjectId('68b55acbf311486d1a6b128d'),

'2': ObjectId('68b55acbf311486d1a6b128e')

}

}

mycompiler\_mongodb> [

{

\_id: ObjectId('68b55acbf311486d1a6b128c'),

id: 1,

name: 'Ryan',

gender: 'M',

group: 'Bsc',

age: 21

},

{

\_id: ObjectId('68b55acbf311486d1a6b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bsc',

age: 23

},

{

\_id: ObjectId('68b55acbf311486d1a6b128e'),

id: 1,

name: 'sita',

gender: 'F',

group: 'Bcom',

age: 22

}

]

mycompiler\_mongodb> {

acknowledged: true,

insertedId: null,

matchedCount: 1,

modifiedCount: 1,

upsertedCount: 0

}

mycompiler\_mongodb> [

{

\_id: ObjectId('68b55acbf311486d1a6b128c'),

id: 1,

name: 'Ryan',

gender: 'M',

group: 'Bsc',

age: 21

},

{

\_id: ObjectId('68b55acbf311486d1a6b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bsc',

age: 25

},

{

\_id: ObjectId('68b55acbf311486d1a6b128e'),

id: 1,

name: 'sita',

gender: 'F',

group: 'Bcom',

age: 22

}

]

mycompiler\_mongodb>

[Execution complete with exit code 0]

db.students.insertMany([

{ id: 1, name: 'Ryan', gender: 'M' ,group:'Bsc',age:21},

{ id: 2, name: 'Rian', gender: 'M' ,group:'Bsc',age:23},

{ id: 1, name: 'sita', gender: 'F' ,group:'Bcom',age:22},

]);

db.students.find();

db.students.updateMany({},{$set:{age: 30}});

db.students.find();

OUTPUT:

mycompiler\_mongodb> ... ... ... ... {

acknowledged: true,

insertedIds: {

'0': ObjectId('68b55b259464640b426b128c'),

'1': ObjectId('68b55b259464640b426b128d'),

'2': ObjectId('68b55b259464640b426b128e')

}

}

mycompiler\_mongodb> [

{

\_id: ObjectId('68b55b259464640b426b128c'),

id: 1,

name: 'Ryan',

gender: 'M',

group: 'Bsc',

age: 21

},

{

\_id: ObjectId('68b55b259464640b426b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bsc',

age: 23

},

{

\_id: ObjectId('68b55b259464640b426b128e'),

id: 1,

name: 'sita',

gender: 'F',

group: 'Bcom',

age: 22

}

]

mycompiler\_mongodb> {

acknowledged: true,

insertedId: null,

matchedCount: 3,

modifiedCount: 3,

upsertedCount: 0

}

mycompiler\_mongodb> [

{

\_id: ObjectId('68b55b259464640b426b128c'),

id: 1,

name: 'Ryan',

gender: 'M',

group: 'Bsc',

age: 30

},

{

\_id: ObjectId('68b55b259464640b426b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bsc',

age: 30

},

{

\_id: ObjectId('68b55b259464640b426b128e'),

id: 1,

name: 'sita',

gender: 'F',

group: 'Bcom',

age: 30

}

]

mycompiler\_mongodb>

[Execution complete with exit code 0]

DELETE

db.students.insertMany([

{ id: 1, name: 'Ryan', gender: 'M' ,group:'Bsc',age:21},

{ id: 2, name: 'Rian', gender: 'M' ,group:'Bsc',age:23},

{ id: 1, name: 'sita', gender: 'F' ,group:'Bcom',age:22},

]);

db.students.find();

db.students.deleteOne({name:"Rian"});

db.students.find();

OUTPUT:

mycompiler\_mongodb> ... ... ... ... {

acknowledged: true,

insertedIds: {

'0': ObjectId('68b55bae51d06cddf56b128c'),

'1': ObjectId('68b55bae51d06cddf56b128d'),

'2': ObjectId('68b55bae51d06cddf56b128e')

}

}

mycompiler\_mongodb> [

{

\_id: ObjectId('68b55bae51d06cddf56b128c'),

id: 1,

name: 'Ryan',

gender: 'M',

group: 'Bsc',

age: 21

},

{

\_id: ObjectId('68b55bae51d06cddf56b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bsc',

age: 23

},

{

\_id: ObjectId('68b55bae51d06cddf56b128e'),

id: 1,

name: 'sita',

gender: 'F',

group: 'Bcom',

age: 22

}

]

mycompiler\_mongodb> { acknowledged: true, deletedCount: 1 }

mycompiler\_mongodb> [

{

\_id: ObjectId('68b55bae51d06cddf56b128c'),

id: 1,

name: 'Ryan',

gender: 'M',

group: 'Bsc',

age: 21

},

{

\_id: ObjectId('68b55bae51d06cddf56b128e'),

id: 1,

name: 'sita',

gender: 'F',

group: 'Bcom',

age: 22

}

]

mycompiler\_mongodb>

[Execution complete with exit code 0]

db.students.find({

$where: function() {

return this.id === 1;

}

});

AND

db.students.find({

gender: "M",

group: "Bsc"

});

OR

db.students.find({

$or: [

{ gender: "F" },

{ group: "Bcom" }

]

});

Output

mycompiler\_mongodb> ... ... ... ... ... [

{

\_id: ObjectId('68b56256088cf424076b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bcom',

age: 23

},

{

\_id: ObjectId('68b56256088cf424076b128e'),

id: 3,

name: 'sita',

gender: 'F',

group: 'Ba',

age: 22

}

]

mycompiler\_mongodb>

mycompiler\_mongodb>

[Execution complete with exit code 0]

Sort

db.students.find().sort({ age: -1 }); desecndind..1 ascending

output

mycompiler\_mongodb> [

{

\_id: ObjectId('68b562dc6fc39684226b128d'),

id: 2,

name: 'Rian',

gender: 'M',

group: 'Bcom',

age: 23

},

{

\_id: ObjectId('68b562dc6fc39684226b128e'),

id: 3,

name: 'sita',

gender: 'F',

group: 'Ba',

age: 22

},

{

\_id: ObjectId('68b562dc6fc39684226b128c'),

id: 1,

name: 'Ryan chowdary',

gender: 'M',

group: 'Bsc',

age: 21

}

]

mycompiler\_mongodb>

mycompiler\_mongodb>

mycompiler\_mongodb>

[Execution complete with exit code 0]

Limit

db.students.find().sort({ age: 1 }).limit(2);

output

mycompiler\_mongodb> [

{

\_id: ObjectId('68b56375117540bbe36b128c'),

id: 1,

name: 'Ryan chowdary',

gender: 'M',

group: 'Bsc',

age: 21

},

{

\_id: ObjectId('68b56375117540bbe36b128e'),

id: 3,

name: 'sita',

gender: 'F',

group: 'Ba',

age: 22

}

]

mycompiler\_mongodb>

mycompiler\_mongodb>

[Execution complete with exit code 0]

INDEXES

Create a Sample Collection

db.students.insertMany([

{ id: 1, name: 'Ryan', age: 21, group: 'Bsc' },

{ id: 2, name: 'Sita', age: 22, group: 'Bcom' },

{ id: 3, name: 'Rian', age: 23, group: 'Bsc' },

{ id: 4, name: 'Anu', age: 20, group: 'Bcom' }

]);

2. Create an Index on a Field

Example: Create an index on the name field (ascending)

db.students.createIndex({ name: 1 });

* 1 = ascending order
* -1 = descending order (used for sorting, not direction of search)

3. Check Existing Indexes

db.students.getIndexes();

You’ll see the default \_id\_ index and your custom one:

[

{ "v": 2, "key": { "\_id": 1 }, "name": "\_id\_" },

{ "v": 2, "key": { "name": 1 }, "name": "name\_1" }

]

4. Use the Index (Query on name)

db.students.find({ name: 'Rian' });

This query will now use the index on name, making it faster especially on large datasets.

5. Compound Index (on multiple fields)

Create an index on both group and age:

db.students.createIndex({ group: 1, age: -1 });

This helps for queries like:

db.students.find({ group: 'Bcom' }).sort({ age: -1 });

6. Remove an Index

db.students.dropIndex("name\_1");

(Use getIndexes() first to see the index name)

AGGREGATION

db.students.insertMany([

{ name: 'Ryan', gender: 'M', group: 'Bsc', age: 21, marks: 85 },

{ name: 'Sita', gender: 'F', group: 'Bcom', age: 22, marks: 90 },

{ name: 'Rian', gender: 'M', group: 'Bsc', age: 23, marks: 78 },

{ name: 'Anu', gender: 'F', group: 'Bcom', age: 20, marks: 88 },

{ name: 'John', gender: 'M', group: 'Bsc', age: 24, marks: 70 }

]);

1. Group by group and get average marks

js

Copy code

db.students.aggregate([

{

$group: {

\_id: "$group",

averageMarks: { $avg: "$marks" }

}

}

]);

Output:

mycompiler\_mongodb> ... ... ... ... ... ... ... [

{ \_id: 'Bsc', averageMarks: 77.66666666666667 },

{ \_id: 'Bcom', averageMarks: 89 }

]

json

{ "\_id": "Bsc", "averageMarks": 77.67 }

{ "\_id": "Bcom", "averageMarks": 89 }

2. Count number of students per gender

js

Copy code

db.students.aggregate([

{

$group: {

\_id: "$gender",

count: { $sum: 1 }

}

}

]);

3. Match (filter) and then group

Find average marks of only male students:

js

Copy code

db.students.aggregate([

{ $match: { gender: "M" } },

{

$group: {

\_id: "$group",

avgMarks: { $avg: "$marks" }

}

}

]);

4. Sort by average marks descending

js

Copy code

db.students.aggregate([

{

$group: {

\_id: "$group",

avgMarks: { $avg: "$marks" }

}

},

{ $sort: { avgMarks: -1 } }

]);

5. Project (select specific fields)

js

Copy code

db.students.aggregate([

{

$project: {

\_id: 0,

name: 1,

age: 1,

marks: 1,

isPassed: { $cond: [ { $gte: [ "$marks", 80 ] }, true, false ] }

}

}

]);

This adds a new field isPassed based on marks.

6. Get top 2 scorers

js

Copy code

db.students.aggregate([

{ $sort: { marks: -1 } },

{ $limit: 2 }

]);

Common Aggregation Operators:

Operator Description

$match Filters documents

$group Groups by a field

$sort Sorts documents

$project Shapes the output (select fields, computed values)

$limit Limits result count

$avg, $sum, $max, $min, $count Aggregate functions

Mapreduce

db.students.insertMany([

{ name: 'Ryan', group: 'Bsc', marks: 85 },

{ name: 'Sita', group: 'Bcom', marks: 90 },

{ name: 'Rian', group: 'Bsc', marks: 78 },

{ name: 'Anu', group: 'Bcom', marks: 88 },

{ name: 'John', group: 'Bsc', marks: 70 }

]);

Example 1: Calculate total marks per group using MapReduce

Map function

Emits (group, marks) pairs:

js

Copy code

var mapFunction = function() {

emit(this.group, this.marks);

};

Reduce function

Sums marks for each group:

js

Copy code

var reduceFunction = function(key, values) {

return Array.sum(values);

};

Run MapReduce

js

Copy code

db.students.mapReduce(

mapFunction,

reduceFunction,

{ out: "total\_marks\_per\_group" }

);

Check output

js

Copy code

db.total\_marks\_per\_group.find();

Result:

json

Copy code

{ "\_id": "Bsc", "value": 233 }

{ "\_id": "Bcom", "value": 178 }

Example 2: Count students per group

Map function

js

Copy code

var mapFunction = function() {

emit(this.group, 1);

};

Reduce function

js

Copy code

var reduceFunction = function(key, values) {

return Array.sum(values);

};

Run MapReduce

js

Copy code

db.students.mapReduce(

mapFunction,

reduceFunction,

{ out: "count\_students\_per\_group" }

);

Download a zip code dataset at <http://media.mongodb.org/zips.json> Use mongo import to import the zip code dataset into Mongo DB. After importing the data, answer the following questions by using aggregation pipelines:

1.Find all the states that have a city called ”BOSTON”.

1. db.data1.distinct("state", { city: "BOSTON" });

2.Find all the states and cities whose names include the string ”BOST”.

1. db.data1.find(
2. {
3. $or: [
4. { "city": { $regex: "bost", $options: "i" } },
5. { "state": { $regex: "bost", $options: "i" } }
6. ]
7. },
8. {
9. \_id: 0,
10. city: 1,
11. state: 1
12. }
13. );

3. Each city has several zip codes. Find the city in each state with the most number of zip codes and rank those cities along with the states using the city populations.

3. db.data1.aggregate([

// 1. Group by state & city: count zip codes, sum population

{

$group: {

\_id: { state: "$state", city: "$city" },

zipCount: { $sum: 1 },

totalPop: { $sum: "$pop" }

}

},

// 2. Sort by state ascending, zipCount descending (to prepare for picking max zipCount city)

{

$sort: {

"\_id.state": 1,

zipCount: -1

}

},

// 3. Group by state: pick the city with max zipCount (first after sort)

{

$group: {

\_id: "$\_id.state",

city: { $first: "$\_id.city" },

zipCount: { $first: "$zipCount" },

totalPop: { $first: "$totalPop" }

}

},

// 4. Sort final output by totalPop descending (rank by population)

{

$sort: {

totalPop: -1

}

},

// 5. Project fields nicely

{

$project: {

\_id: 0,

state: "$\_id",

city: 1,

zipCount: 1,

totalPop: 1

}

}

]);

1. PL/SQL program to implement %found

DECLARE

CURSOR c\_emp IS

SELECT employee\_id, first\_name

FROM employees

WHERE ROWNUM <= 3; -- Just for demo

v\_emp\_id employees.employee\_id%TYPE;

v\_emp\_name employees.first\_name%TYPE;

BEGIN

OPEN c\_emp;

LOOP

FETCH c\_emp INTO v\_emp\_id, v\_emp\_name;

EXIT WHEN NOT c\_emp%FOUND; -- Check with %FOUND

DBMS\_OUTPUT.PUT\_LINE('Employee: ' || v\_emp\_id || ' - ' || v\_emp\_name);

END LOOP;

CLOSE c\_emp;

END;

/

Q: Write a PL/SQL block that display’s employee name and the salary of the first 3 employee’s having the highest salary

DECLARE

CURSOR c\_top\_emp IS

SELECT first\_name, salary

FROM employees

ORDER BY salary DESC

FETCH FIRST 2 ROWS ONLY; -- Get only top 2 employees

v\_name employees.first\_name%TYPE;

v\_salary employees.salary%TYPE;

BEGIN

OPEN c\_top\_emp;

LOOP

FETCH c\_top\_emp INTO v\_name, v\_salary;

EXIT WHEN c\_top\_emp%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Employee: ' || v\_name || ' | Salary: ' || v\_salary);

END LOOP;

CLOSE c\_top\_emp;

END;

/

Q: Write a PL/SQL program to find the sum of the salaries in a particular department

DECLARE

v\_dept\_id employees.department\_id%TYPE := 10; -- Change department ID as needed

v\_total\_sal NUMBER;

BEGIN

SELECT SUM(salary)

INTO v\_total\_sal

FROM employees

WHERE department\_id = v\_dept\_id;

IF v\_total\_sal IS NULL THEN

DBMS\_OUTPUT.PUT\_LINE('No employees found in department ' || v\_dept\_id);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Total Salary for Department ' || v\_dept\_id || ' = ' || v\_total\_sal);

END IF;

END;

/

Q: PL/SQL to print all employees begin with s.

DECLARE

CURSOR c\_emp IS

SELECT first\_name

FROM employees

WHERE UPPER(first\_name) LIKE 'S%'; -- Names starting with S (case-insensitive)

v\_name employees.first\_name%TYPE;

BEGIN

OPEN c\_emp;

LOOP

FETCH c\_emp INTO v\_name;

EXIT WHEN c\_emp%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || v\_name);

END LOOP;

CLOSE c\_emp;

END;

/

Q: Write a PL/SQL to print names of all managers

DECLARE

-- Variable to hold the name of the manager

v\_manager\_name employees.name%TYPE;

BEGIN

-- Loop over the query result to print the names of managers

FOR r IN (SELECT DISTINCT e.name

FROM employees e

WHERE e.manager\_id IS NOT NULL)

LOOP

-- Print the manager's name

DBMS\_OUTPUT.PUT\_LINE(r.name);

END LOOP;

END;

/