

1. Complehensive NoSal Exercise

barite insertion queries (MangoDB Syntax) for sample data, demonstrating schema-less plexibility (e.g., some user have "bio", other don't). document-based system (MangoDB): collections for users and posts, with embedded nested documents for anments. Diesign a NOSOIL schema for a Social media appusing a

by a key like user-id, and create a materialized view

for Popular Posts (if Supported)

generated data such as posts, comments, likes and Profile information: A NOSOIL document-oriented database like MongoDB is well become:

It provides a flexible schema . It supports nested/embedded documents . It is rightly scalable with horizontal shanking.

In this sutema design, we will use two main collections.

O users Collection

> Each document corresponds to a single user -> The users collection holds data about registered was のはないの あるかっとう

-> fields one flexible

+ Structure:



" id": objected

name"; "Rohul"

"email": " xyz Qgmail. 6m"

bio": " Dreamer

" profile -pic": https://cdn.app.om/woolspg "joined_data": 130Datel "2025-01-01"

followers": [" 1002", " 1003"]

following : [" voon "]

. _ id : Mongo DB's unique identified

· user-id: Application-level unique 10 for the user

· bio and Profile-pic one optional frelds showing MongaDBS scheme less nature.

· follower and following arrays help in implementing social connection

1 Posts Weckon

> Each Post references the user who created it through user id -> The posts collection stores all the posts made by use's

> Comments one stored as embedded/ nested documents in side the Fost -> Posts may contain text, media on both

Structure :-

" "id" objected Post_id": "PIOI"

"user_id": " 0001"

Content": "My first post on this appl"
"media": het ps. If I an app. am / phil [pg";
"created at": Iso Date (" 2025-01-10")
"likes": Uso
"amments: [
"comment id": "coo2";
"text": Nice post Rahud?"
"created at": 150 Date (" 2025-01-11")
"s
"user_id": "0002";
"user_id": "0003";
"text: "Nice";
"so Date ("2025-01-11")
"s
"created at at": 150 Date ("2025-01-11")

· Gard past has past-id, content, media, coneated-at and likes.

· Garments is an array of nested documents.

· foothcomment includes its own comment—id, the user—id & the commenter text, and timestamp.

· Embedding comments inside pasts - improves performance belowse the application can fetch a post with all its comments in a style

Offeribility

Describility original Medica Features -

GNOM Ple Insertion Synatax for one document Insertion Syntam for Many documents db-allection_nome.insat ([insert users with and ab. collection_name.inspatonofs b) Insertion Queries (MONGODB) abousers insertane (& Menapli Insertien Systam usa-id":" uoo!" "nome": "Rahul" "email": "xyz@gmail. Come" "bio": "enployen" " field 1" . " Value " fieldi": " value!") "Prefile - Pic": " https:// Dahwipg" field 2 " " value 2", " field 1" " values" field 3": "value 3" Joined date": new Date ("2025-01-01") without optional fields like bio, Profile - Di

db.users.insertone({""user_id": "uooz",
"name": "mahesh",

"email": "Abc@gmail.come";
"Joined_date": new Date ("2025-01-05")

3);

· User vool has bid and Profile-Pic · User vool does not have these fields (MongoDB allows missing fields).

OSharding and Materialized View in Mongo DB.

1. Implementing Sharding on posts allection:

Sharding is used to distribute data across multiple sorvers to improve performance and scalability in large databases.

Steps to shoold posts collection by user-id:

1. enable shording on the database sh. enable Shording (" social - app")

=) this prepares the database for shortding

2 - create on index on the short key
db. posts. createInder (\$usor_id:13)

> The shord key (uson_id) must be indexed.

3. shord the Collection

sh. shandCollection (* Social-app. posts*, & user_id:13)

> Mongo DB splits the Posts collection across shands bosed on

(user-id. -> All Posts of a Particular user will usually goto the game shard, balancing the data. Supports horizontal scaling · Oruevies filtering by user-id one faster · Large Collections do not overload a single Server 2. Creating a Materialized view for popular posts 7. A Materialized view stores the Precomputed results do a query for faster access > In MongoDB, Materialized Views can be Simulated using aggregation with sout or & merge En: create a view of Posts with more than loolikes. db.posts.aggregate([& & match: & likes: & \$9t:100333, ? \$ project : { post_id: 1, user_id: 1, content: 1, likes: 13? 9 dout: "popular_posts"3 The Resultis stored in new collection called Popular - Posts -) whenever this query is run, it updates the materialized view. Benefits: ->orvick access to frequently grueried data

- Reduces Computation during runtime.

(ASSIGNMENT -II

2. Soil Functions: Date, Time, Numeric, and String a Consider a table orders with columns (Order D), Order Date Customer Name, Total Amount).

b. Write an Soil query to select orders placed in the last 30 days using DATE functions (e.g., CURDATE(),

DATE-ADDOD.

C-USE numeric functions (e.g., ROUNDL), cell) to round Total Amount to the nearest integer and calculate a 10%. discourt (arithmetic: Total Amount , 0.9).

d. Use. String functions (e.g., CONCATO), UPPERC), SUBSTRINGO) to project a formatted column "CUSTOMERSUMMARY" as "UPPER (customer Name) - Order 1D".

Insert Sample data for 4 orders and write a query combining all these.

A SOIL functions: Date, Time, Numeric , and String

a) CREATE ORDERS Table

CREATE TABLE Order ! Order D INT PRIMARY KEY, OrderDate DATE, CustomerName VARCHAR (50), Total Amount DECIMAL (10,2)):

· Order D -> unique identifier for each order

· orderDate > Date when the order was placed · customerName > Name of the customer.

DAY).

· Total Amount -> Total price & the order

b) INSERT Sample Data (4 Orders)

INSERT INTO orders (order D., Order Date, customer Name, Total Amount VALUES

(101, CURDATEC), Rohul Reddy', 1234.56),

(102, DATE_SUB (CURDATEL), INTERVAL 10 DAY),

Bharath', 789.45),

(103, DATE_SUB (CURDATE(), INTERVAL 35 DAY),

Devil', 456-78),

(104, DATE-SUB(CURDATED, INTERVAL 20 DAY).

'A83(181', 999.99);

· Order 103 is older than 30 days > will be excluded in last 30 days query.

· CURDATE() and DATE-SUBO are Date functions

9 Query Using Date & Numeric Functions

SELECT Order D. Order Date, Total Amount, ROUND CTotal Amount

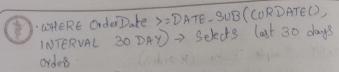
As Rounded Amount, CEIL (Total Amount * 0:9) As

Discounted Amount FROM orders WHERE

OrderDate >= DATE_SUB(CURDATE(), INTERVAL 30

· ROUND (Total Amount) -> round & to newest integer.

· CEIL (Total Amount + 0.9) > applies 10.1, discount of rounds up.



d) Query Using String Functions

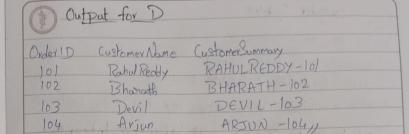
SELECT Order ID, Customer Name, CONCAT (UPPER (
Customer Name), '-', Order ID) AS Customer Summary
From Orders;

· UPPER (Customer Name) -> Converts name to Upper Case.
· CONCAT (UPPER (Customer Name), '-', Order ID) -> Creates
a formatted column customer Summary.

output for (

OrderID	OrderDate	TotalAmount	Rounded Amount	Discourte Amount
101	2025-09-29	1234.56	1235	711
102	2025-09-19	999.99	1000	900

Note: Order 103 is excluded because it is olderthan Bodays.



3. Implement Relation Ships and Referential Integrity a. Consider tables for an e-commerce system:

Customes (CustID, Name) and Order (Order Date, CustID).

b. Write Soil to create both tables, with custID as PRIMARY KEY in orders with ON DELETE CASCADE for referential integrity.

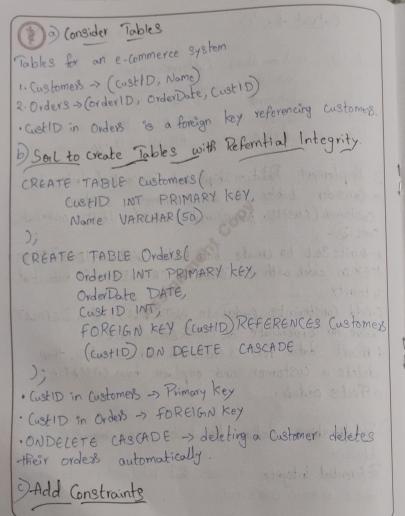
C.Add Constraints: OrderDate NOTNULL, and a DEFAULT Value for OrderDate as CURRENT_DATE.

delete a customer and explain how CASCADE affects orders.

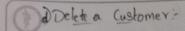
A) Relationships define how tables one connected in a database.

Referential Integrity ensures that relationships between

tables remain Consistent.



CREATE TABLE Order OrderID INT PRIMARY KEY Order Date NOT NULL DEFAULT CURRENT. DATE. CustIDINT, FOREIGN KEY (CUSTID) REFERENCES CUSTOMES (CustID) ON DELETE CASCADE · Order Date NOT NULL > ensures every order has a date . DEFAULT CURRENT_DATE > if no date is given, todays date is used. d) Insert Sample Data and Demonstrate CASCADE-Customes INSERT INTO CUSTOMERS VALUES (1, Ram); INSERT INTO Customes VALUES (2, Mah;); INSERT INTO Costomers VALUES (3, 'Sita'); Orders INSERT INTO Orders VALUES (101, 2025-09-29, 1); INSERT INTO orders VALUES (102, 2025-09-28, 2) INSERT INTO Order VALUES (103, 2025-09-27') INSERT INTO Ordes VALUES (104, 2025-09-26,3) INSERT INTO Ordes VALUES(105, 2025-09-25)



DELETE FROM Customers WHERE CustID=2;

Effect of CASCADE:

- · Customer with custID=2 is deleted
- · All orders linked to this customer (order 1D lo2 and los) are automatically deleted from the Orders table
- · Ensures referential Integrity -> no orders exist for a deleted Customen.

4. Different Types of Joins Using tables Departments DeptiD, Dept Name) and Employees (EmpiD, Name, DeptiD, Salary).

a. Write SOIL for an INNER JOIN to list of employees with their dept names

b. User (EFT OUTERSOIN to Include departments with no employees with their and RIGHT OUTER JOIN for the Yeverse .

c. implement a full OUTER JOIN (or emulate if not Supported) and a CROSS JOIN to Show all Possible Combinations.

d. Insert Sample data (4 departments, 6 employees) and explain difference in results.

DJOINS:

Joins is an operation that Combines yours from two or more tables based on a related column before between them.

Tables Considered :

Departments > (Dept1D, DeptName) Employees -> (EmplD, Name, DeptlD, Salary)

a) INNER JOIN - Employees with Department Names

Tables Creation 1) De Partments

CREATE TABLE Deportments DeptID INT PRIMARY KEY,

Dept Name VAR (HAR (SO) NOT NULL

i) Employees:

CREATE TABLE Employees

EMPLD INT PRIMARY KEY, Name VARCHAR (50) NOT NULL,

DeptID INT,

Salary DECIMAL (10,20), FOREIGN KEY (DeftID) REFERENCES Deportments (1

ON DELETE SET NULL);

d) Values Insertion

INSERT INTO Departments (1, 'HR'); INSERT INTO Departments (2, "IT); INSERT INTO Departments (3, 'Sales'); INSERT INTO Departments (4, 'finance');

Employees

INSERT INTO Employees (101, 'Rahul', 1, 50000). INSERT INTO Employees (102, 'Bharath', 2,60000). INSERT INTO Employees (103), Maheshi, 2,55000). INSERT INTO Employees (104, 'ABJun', 3/45008). INSERT INTO Employees (to, 'Ravi', NULL, 40000) INSERT INTO Employees (106, 'Nehanth', 2,5800).

-> Employee 105 has no department -> Departments 1,2,3,4 all exist, but finance initially has no employee.

SELECT e. EmplD, e. Name, d. DeptName, e. Salar

FROM Employees e

INNER JOIN Departments of ON e. DeptID = d. DeptID: EmplD Name Dept Nome Salary Rapul 50000 60000 102 Bharath IT 55000 103 Mahesh 45000 Soles 104 Arjun 17 58000 106 Nehanth

Employee 105 is excluded because DeptIDISNULL

DIEFT OUTER JOIN

SELECT d. Dept 1D, d. Dept Name, e. EmplD, e. Name FROM Departments d

LEFTJOIN Employees e ON J. DeptlD = e. DeptlD;

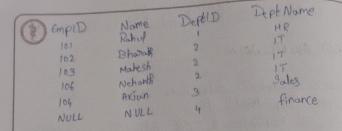
DeptID	Dept Name	EmpID	Name	
	HR	10)	Rahul	
2	17	102	Bharath	
2	1T	103	Mahesh	
2	11	106	Nehanth	
3	Salos	104	Arian	
4	FINANCE	NULL	NULL	

finance has no employees, so it is NULL

RIGHT OUTER JOIN

SELECT e. EmplD, e. Name, d. Dept ID, d. Dept Name

FROM Employees e RIGHT JOIN Deportments of ON e-DeptiD=d-DeptiD



OFULL OUTER JOIN

SELECT e. EmplD, e. Name, d. Dept1D, d. DeptName

FROM Employees e

FULL OUTER JOIN DEPORTMENTS of ON e. Depti D = d. Depti D.

EmplD	Name	DeptID	DeptName
(0)	Rahul	9	HR 1T
102	Bhoroth	2	IT
103	Mahesh	3	sortes
104	Arjun	NULL	NULL
105	Ravi 10	2	JTV. 95
106	Nehanth	4	finance
NINL	NULL		

CROSS JOIN

SELECT e. Name As Employee, d. Dept Name As Department FROM Employees e CROSS JOIN Department

Difference IN Join Type	Result Includes
INNERJOIN	only matching rows from both tables.
I EFT OUTER JOIN	All yours from Departments + matching employee
RIGHT OUTER JOIN	All YAWS from Employees + Matching departments
FULLOUTER JOIN	All rows from both Tables, with NULL's for missing
CROSSION	All possible Combinations.