ವಿಶ್ವೇಶ್ವರಯ್ಯ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ, ಬೆಳಗಾವಿ
VISVESVARAYA TECHNOLOGICAL UNIVERSITY - BELAGAVI

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. Road, V.V. Pura, Bengaluru -560 004

Department of CSE (DATA SCIENCE)

BCS403

Database Management System

Laboratory Manual

IV- Semester

Prepared By

Prof. Shankar Gowda B. N.
Professor & Head,
CSE (Data Science), BIT

Bangalore Institute of Technology

K. R. Road, V. V. Pura, Bengaluru- 560004

Department of CSE (Data Science)

VISION

The vision of CSE (Data Science) Department is to establish a state of art academic department thattrains the next generation of students as data scientists who will solve complex challenges and innovate through world-class research.

MISSION

M1	To educate students in a field that has ushered in a once-in-a-generation revolution.
M2	To provide an environment for leading edge research that has a strong and rapid impact on the economy.
M3	To establish a “Center of Excellence" to facilitate Data creators, providers, managers, curators, and users.
M4	To inculcate the strong qualities of leadership and Entrepreneurship.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO-1	To provide Graduates with a strong understanding of basic and advanced methods in statistical inference, machine learning, data visualization, data mining, and big data, all of which are essential skills for a high-performing Data Scientist.
PEO-2	The Graduate will have the ability to apply the fundamental and advanced skills to assist with data-based decision making.
PEO-3	Graduates will have the ability to strengthen the level of expertise through higher studies and research.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO-1	The graduates of the program will have the ability to apply modern artificial intelligence and deep learning methods to complex prediction and recognition tasks.
PSO-2	The graduates of the program will Play an analytical role in a company where one can design, implement, and evaluate advanced statistical models and approaches for application to the company’s most complex problem.

PROGRAM OUTCOMES (POs)

The Engineering Graduates will be able to:

1. **Engineering knowledge**:

   Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**:

   Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**:

   Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**:

   Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**:

   Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**:

   Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**:

   Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**:

   Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**:

   Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**:

    Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**:

    Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**:

    Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

1. CREATE A TABLE CALLED EMPLOYEE & EXECUTE THE FOLLOWING.

EMPLOYEE (EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)

CREATE TABLE EMPLOYEE(

EMPNO NUMBER(5),

ENAME VARCHAR(20),

JOB VARCHAR(15),

MANAGER_NO NUMBER (5),

SAL NUMBER(6),

COMMISSION NUMBER (6));

1. Create a user and grant all permissions to the user.

GRANT ALL ON EMPLOYEE TO SHANKAR;

2. Insert the any three records in the employee table contains attributes

EMPNO, ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback.

EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
101	RAM	MANAGER		200000	
102	SHYAM	SALESMAN	101	150000	10000
103	RADHA	DESIGNER	101	160000	20000
104	JANE	TESTER	102	100000	

INSERT INTO EMPLOYEE VALUES (‘E1’, ‘RAM’, , ‘MANAGER’, 200000,);

INSERT INTO EMPLOYEE VALUES (‘E2’, ‘SHYAM’, ‘SALESMAN’, ‘E1’, 160000,10000);

INSERT INTO EMPLOYEE VALUES (‘E3’, ‘RADHA, ‘DESIGNER, ‘E1’, 150000,20000);

INSERT INTO EMPLOYEE VALUES (‘E4’, ‘JANE, ‘TESTER, ‘E2’, 100000,);

ROLLBACK;

Check the result.

3. Add primary key constraint and not null constraint to the employee table.

```
ALTER TABLE EMPLOYEE
ADD
CONSTRAINT PK_EMPLOYEE PRIMARY KEY(EMPNO),
CONSTRAINT FK_EMPLOYEE FOREIGN KEY(MANAGER_NO) REFERENCES
EMPLOYEE (EMPNO));
```

4. Insert null values to the employee table and verify the result

```
INSERT INTO EMPLOYEE VALUES ( , 'SHYAM', 'SALESMAN', 101, 160000,10000);
INSERT INTO EMPLOYEE VALUES (E2, 'SHYAM', 'SALESMAN', 101, 160000,10000);
INSERT INTO EMPLOYEE VALUES (E2, 'SHYAM', 'SALESMAN', 102, 160000,10000);
```

2. CREATE A TABLE CALLED EMPLOYEE THAT CONTAIN ATTRIBUTES EMPNO, ENAME, JOB, MGR, SAL

```
CREATE TABLE EMP (
EMPNO NUMBER(5),
ENAME VARCHAR(20),
JOB VARCHAR(15),
MANAGER_NO CHAR(5),
SAL NUMBER(5),
CONSTRAINT PK_EMP PRIMARY KEY(EMPNO),
CONSTRAINT FK_EMP FOREIGN KEY(MANAGER_NO) REFERENCES EMP
(EMPNO));
```

Execute the following.

1. Add a column commission with domain to the Employee table.

```
ALTER TABLE EMPLOYEE
ADD COLUMN  COMMISSION NUMBER (5);
```

2. Insert any five records into the table.

```
INSERT INTO EMPLOYEE VALUES (‘101’, ‘RAM’, , ‘MANAGER’, 200000, );
INSERT INTO EMPLOYEE VALUES ( , ‘SHYAM’, ‘SALESMAN’, 105, 160000,10000);
INSERT INTO EMPLOYEE VALUES (102, ‘SHYAM’, ‘SALESMAN’, 105, 160000,10000);
INSERT INTO EMPLOYEE VALUES (103, ‘SHYAM’, ‘SALESMAN’, 105, 160000,10000);
INSERT INTO EMPLOYEE VALUES (‘104’, ‘JANE, ‘TESTER, ‘E2’, 100000, );
INSERT INTO EMPLOYEE VALUES (‘1055’, ‘MOHAN’, ‘DESIGNER, ‘E1’,
150000,20000);
```

3. Update the column details of job

```
UPDATE EMPLOYEE
SET
JOB=’DIRECTOR’
WHERE EMPNO=’E1’;
```

4. Rename the column of Employ table using alter command.

ALTER TABLE EMP RENAME EMP AS EMP1;

5. Delete the employee whose Empno is 105.

DELETE FROM EMP WHERE EMPNO=105;

3. QUERIES USING AGGREGATE FUNCTIONS (COUNT, AVG, MIN, MAX, SUM), GROUP BY, ORDERBY. EMPLOYEE (E_ID, E_NAME, AGE, SALARY)

1. CREATE EMPLOYEE TABLE CONTAINING ALL RECORDS E_ID, E_NAME, AGE, SALARY.

   CREATE TABLE EMPL (

   EMPNO NUMBER(5),

   ENAME VARCHAR(20),

   AGE NUMBER(3),

   SAL NUMBER(5),

   CONSTRAINT PK_EMPL PRIMARY KEY(EMPNO),

   CONSTRAINT FK_EMPL FOREIGN KEY(MANAGER_NO) REFERENCES EMPL (EMPNO));

INSERT INTO EMPLOYEE VALUES (&EMPNO, '&ENAME', &AGE,&SAL);

EMPNO	ENAME	AGE	SAL
101	RAM	35	200000
102	SHYAM	35	150000
103	RADHA	40	160000
104	JANE	40	100000
105	SHAN	49	250000

2. **Count number of employee names from employee table**

   SELECT COUNT(*) FROM EMPL;

3. **Find the Maximum age from employee table.**

   SELECT MAX(AGE) FROM EMPL;

4. **Find the Minimum age from employee table.**

   SELECT MIN(AGE) FROM EMPL;

5. **Find salaries of employee in Ascending Order.**

   SELECT * FROM EMPL ORDERBY SAL;

6. **Find grouped salaries of employees.**

   SELECT * FROM EMPL

   GROUP BY AGE;

4. CREATE A ROW LEVEL TRIGGER FOR THE CUSTOMERS TABLE THAT WOULD FIRE FOR INSERT OR UPDATE OR DELETE OPERATIONS PERFORMED ON THE CUSTOMERS TABLE. THIS TRIGGER WILL DISPLAY THE SALARY DIFFERENCE BETWEEN THE OLD & NEW SALARY.

CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)

```
CREATE TABLE CUSTOMER (
CID NUMBER (5),
CNAME VARCHAR (20),
AGE NUMBER (3),
ADDRESS VARCHAR (20),
SAL NUMBER (5),
CONSTRAINT PK_CUST PRIMARY KEY(CID));
```

DESCRIPTION of the TRIGGER:

- The BEFORE INSERT OR UPDATE OR DELETE clause specifies that the trigger should fire before any insert, update, or delete operation on the CUSTOMERS table.
- FOR EACH ROW indicates that the trigger is a row-level trigger, which means it will execute once for each affected row.
- Inside the trigger, we declare two variables v_old_salary and v_new_salary to hold the old and new values of the SALARY column.
- Depending on the operation (INSERTING, UPDATING, or DELETING), different actions are taken:
- For INSERTING: Simply display the new salary (:NEW.SALARY).
- For UPDATING: Display both the old and new salaries (:OLD.SALARY and :NEW.SALARY), and calculate and display the salary difference if both values are not null.
- For DELETING: Display the old salary (:OLD.SALARY).
- DBMS_OUTPUT.PUT_LINE is used to print messages to the console, showing the relevant salary information.
- Remember to enable DBMS_OUTPUT for your session to see the output of DBMS_OUTPUT.PUT_LINE. You can enable it with the following command before running your DML statements:

```
-- Create a trigger named salary_difference_trigger
CREATE OR REPLACE TRIGGER salary_difference_trigger
BEFORE INSERT OR UPDATE OR DELETE ON CUSTOMERS
FOR EACH ROW
DECLARE
  v_old_salary CUSTOMERS.SALARY%TYPE;
  v_new_salary CUSTOMERS.SALARY%TYPE;
BEGIN
  IF INSERTING THEN
    -- For INSERT operation, display new salary
    DBMS_OUTPUT.PUT_LINE('New Salary: ' || :NEW.SALARY);

  ELSIF UPDATING THEN
    -- For UPDATE operation, display old and new salaries
    v_old_salary := :OLD.SALARY;
    v_new_salary := :NEW.SALARY;
    DBMS_OUTPUT.PUT_LINE('Old Salary: ' || v_old_salary);
    DBMS_OUTPUT.PUT_LINE('New Salary: ' || v_new_salary);

    -- Display salary difference
    IF v_old_salary IS NOT NULL AND v_new_salary IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Salary Difference: ' || (v_new_salary - v_old_salary));
    END IF;

  ELSIF DELETING THEN
    -- For DELETE operation, display old salary
    DBMS_OUTPUT.PUT_LINE('Old Salary: ' || :OLD.SALARY);
  END IF;
END;
```

5. CREATE CURSOR FOR EMPLOYEE TABLE & EXTRACT THE VALUES FROM THE TABLE. DECLARE THE VARIABLES, OPEN THE CURSOR & EXTRCT THE VALUES FROM THE CURSOR. CLOSE THE CURSOR.

EMPLOYEE (E_ID, E_NAME, AGE, SALARY)

DESCRIPTION:

- We declare variables (v_e_id, v_e_name, v_age, v_salary) to hold the data for each employee fetched from the EMPLOYEE table.
- We define a cursor emp_cursor that selects E_ID, E_NAME, AGE, and SALARY from the EMPLOYEE table.
- The OPEN emp_cursor; statement opens the cursor to start fetching rows from the EMPLOYEE table.
- The LOOP fetches rows one by one from the cursor into the declared variables (v_e_id, v_e_name, v_age, v_salary).
- Inside the loop, we print out each employee's information using DBMS_OUTPUT.PUT_LINE.
- The loop continues until all rows are fetched (emp_cursor%NOTFOUND).
- After processing all rows, the cursor is closed using CLOSE emp_cursor;.
- Exception handling (WHEN OTHERS) is included to catch any errors that might occur during cursor operations. If an error occurs, it displays an error message and ensures the cursor is closed properly.

```
DECLARE
  -- Declare variables to hold employee data
  v_e_id EMPLOYEE.E_ID%TYPE;
  v_e_name EMPLOYEE.E_NAME%TYPE;
  v_age EMPLOYEE.AGE%TYPE;
  v_salary EMPLOYEE.SALARY%TYPE;

  -- Declare cursor for the employee table
  CURSOR emp_cursor IS
    SELECT E_ID, E_NAME, AGE, SALARY
    FROM EMPLOYEE;
BEGIN
  -- Open the cursor
  OPEN emp_cursor;

  -- Loop through the cursor and fetch values into variables
  LOOP
    FETCH emp_cursor INTO v_e_id, v_e_name, v_age, v_salary;

    -- Exit the loop if no more rows to fetch
    EXIT WHEN emp_cursor%NOTFOUND;

    -- Process the fetched data (for example, display or use the values)
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_e_id);
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_e_name);
    DBMS_OUTPUT.PUT_LINE('Age: ' || v_age);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || v_salary);
    DBMS_OUTPUT.PUT_LINE('--------------------------------');
  END LOOP;
```

```
  -- Close the cursor
  CLOSE emp_cursor;
EXCEPTION
  WHEN OTHERS THEN
    -- Handle exceptions (if any)
    DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
    IF emp_cursor%ISOPEN THEN
      CLOSE emp_cursor;
    END IF;
END;
```

6. WRITE A PL/SQL BLOCK OF CODE USING PARAMETERIZED CURSOR, THAT WILL MERGE THE DATA AVAILABLE IN THE NEWLY CREATED TABLE N_ROLLCALL WITH THE DATA AVAILABLE IN THE TABLE O_ROLLCALL. IF THE DATA IN THE FIRST TABLE ALREADY EXIST IN THE SECOND TABLE THEN THAT DATA SHOULD BE SKIPPED.

DESCRIPTION:

- A cursor c_new_rollcall is defined to fetch records from N_ROLLCALL.
- The block iterates through each record (n_rec) retrieved by the cursor.
- For each record, a check is performed (SELECT COUNT(*) INTO v_exists) to see if a record with the same o_id already exists in O_ROLLCALL.
- If v_exists = 0, it means the record does not exist in O_ROLLCALL, so the record is inserted into O_ROLLCALL.
- If v_exists > 0, the record already exists in O_ROLLCALL, and a message is displayed indicating that the record is skipped.
- The COMMIT statement is used to commit the changes after all records have been processed.
- Exception handling is included (WHEN OTHERS) to catch and display any errors that might occur during the process, followed by a rollback (ROLLBACK) to undo any changes made in case of an error.
- Replace o_id, column1, column2, etc., with the actual column names from your N_ROLLCALL and O_ROLLCALL tables. Adjust the cursor and column mappings according to your table schema.

PL/SQL BLOCK OF CODE

```
DECLARE
  CURSOR c_new_rollcall IS
    SELECT * FROM N_ROLLCALL;

  v_exists NUMBER;
BEGIN
  FOR n_rec IN c_new_rollcall LOOP

 -- Check if the record exists in O_ROLLCALL
    SELECT COUNT(*) INTO v_exists
    FROM O_ROLLCALL
         WHERE o_id = n_rec.o_id;                    -- Assuming 'o_id' is the unique identifier
    -- If record does not exist, insert into O_ROLLCALL
    IF v_exists = 0 THEN
      INSERT INTO O_ROLLCALL (o_id, column1, column2, ...) -- Include relevant columns
          VALUES (n_rec.o_id, n_rec.column1, n_rec.column2, ...);     -- Map columns accordingly
    ELSE
-- Record already exists, so skip
      DBMS_OUTPUT.PUT_LINE('Record with o_id ' || n_rec.o_id || ' already exists. Skipping.');
    END IF;
  END LOOP;
    COMMIT;                                                  -- Commit changes
  DBMS_OUTPUT.PUT_LINE('Merge process completed.');
EXCEPTION
  WHEN OTHERS THEN
                                                             -- Handle exceptions
    DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
    ROLLBACK;                                          - - Rollback changes if error occurs
END;
```

7. INSTALL AN OPEN SOURCE NOSQL DATA BASE MANGODB & PERFORM BASIC CRUD (CREATE, READ, UPDATE & DELETE) OPERATIONS. EXECUTE MANGODB BASIC QUERIES USING CRUD OPERATIONS.

To install an open-source NoSQL database, you can choose from several popular options like MongoDB, Apache Cassandra, Couchbase, Redis, or Apache HBase.

Installing MongoDB

Step 1: Choose the Installation Method

MongoDB provides various installation methods depending on your operating system and preferences:

- **Package Manager:** Some operating systems offer MongoDB packages that can be installed using a package manager (e.g., apt for Ubuntu, yum for CentOS).
- **Official MongoDB Repository:** Add the MongoDB repository to your system and install MongoDB using the package manager.
- **MongoDB Download Center:** Download the MongoDB Community Server binaries directly from the MongoDB website.

Step 2: Installation Steps (Linux Example)

Here's a general guide for installing MongoDB on a Linux system (e.g., Ubuntu):

1. **Add MongoDB Repository (for Ubuntu):**

   # Import the MongoDB GPG key

   wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -

   # Add MongoDB repository

   echo "deb [arch=amd64,arm64] https://repo.mongodb.org/apt/ubuntu $(lsb_release -cs)/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list

2. **Install MongoDB:**

   # Update package list

   sudo apt update

   # Install MongoDB

   sudo apt install -y mongodb-org

3. **Start MongoDB Service:**

   # Start MongoDB service

   sudo systemctl start mongod

   # Enable MongoDB to start on boot

   sudo systemctl enable mongod

Step 3: Verify Installation

After installation, you can verify MongoDB status and connect to the MongoDB shell:

Check MongoDB service status:

sudo systemctl status mongod

Access MongoDB shell:

mongo

Step 4: Explore MongoDB

Now that MongoDB is installed, you can start working with it:

- Create databases, collections, and documents.
- Perform CRUD (Create, Read, Update, Delete) operations.
- Use MongoDB Compass (GUI tool) or command-line tools (mongo, mongodump, mongorestore, etc.) to manage MongoDB databases.

Additional Notes

- **Configuration:** MongoDB configuration files (mongod.conf) are typically located in /etc/mongod.conf on Linux systems. Modify these files to adjust MongoDB settings (e.g., port number, data directory).
- **Security:** Ensure that MongoDB is properly secured (e.g., enabling authentication, setting up firewalls, configuring TLS/SSL).
- **Documentation:** Refer to the official MongoDB documentation for detailed instructions, configuration options, and best practices.

NOSQL DATA BASE BASIC CRUD (CREATE, READ, UPDATE & DELETE) OPERATIONS

- NoSQL databases provide flexibility in data modeling and querying compared to traditional relational databases.
- **Document Model:** NoSQL databases are schema-less or schema-flexible, allowing documents within the same collection or bucket to have varying structures.
- **Scalability:** NoSQL databases excel at horizontal scalability, making them suitable for large-scale distributed systems.
- **Consistency Models:** NoSQL databases may offer different consistency models (e.g., eventual consistency, strong consistency) based on use case requirements.
- **Transactions:** Some NoSQL databases support multi-document transactions, while others focus on eventual consistency and atomic operations at a single-document level.
- When working with NoSQL databases, it's essential to understand the specific features and capabilities of the database you're using, as each NoSQL database may have unique query syntax, data manipulation techniques, and operational considerations.

1. **Create (Insert) Documents**

In NoSQL databases like MongoDB or Couchbase, documents are inserted into collections (in MongoDB) or buckets (in Couchbase). Each document is a JSON-like object.

Example (MongoDB):

```
// Insert a document into a collection
db.users.insertOne({
  name: "John Doe",
  age: 30,
  email: "john@example.com"
});

// Insert multiple documents into a collection
db.users.insertMany([
  { name: "Alice Smith", age: 25, email: "alice@example.com" },
  { name: "Bob Johnson", age: 35, email: "bob@example.com" }
]);
```

2. Read (Query) Documents

```
// Find all documents in a collection
db.users.find();

// Find documents matching specific criteria
db.users.find({ age: { $gte: 30 } }); // Find users where age is greater than or equal to 30

// Find a single document by a specific field
db.users.findOne({ email: "john@example.com" });
```

3. **Update Documents**

   Updating documents allows modifying existing data within the database.

```
// Update a document
db.users.updateOne(
  { name: "Alice Smith" },
  { $set: { age: 26 } }
);

// Update multiple documents
db.users.updateMany(
  { age: { $gte: 30 } },
  { $set: { status: "senior" } }
);
```

4. Delete Documents

Deleting documents removes data from the database.

```
// Delete a document
db.users.deleteOne({ name: "Bob Johnson" });
// Delete multiple documents
db.users.deleteMany({ age: { $lt: 25 } }); // Delete users younger than 25
```

3. EXECUTE MANGODB BASIC QUERIES USING CRUD OPERATIONS.

- To execute basic CRUD (Create, Read, Update, Delete) operations in MongoDB, you'll use the MongoDB shell (**mongo**) or a MongoDB client like MongoDB Compass.

Below are examples of how to perform these operations using MongoDB's query syntax.

1. Connect to MongoDB

First, start the MongoDB shell (mongo) and connect to your MongoDB instance:

mongo

2. Basic CRUD Operations Examples:

Create (Insert) Documents

To insert documents into a collection:

javascript

```
// Switch to or create a database (e.g., mydatabase)
use mydatabase;

// Insert a single document into a collection (e.g., users)
db.users.insertOne({ name: "John Doe", age: 30, email: "john@example.com" });

// Insert multiple documents into a collection
db.users.insertMany([
  { name: "Alice Smith", age: 25, email: "alice@example.com" },
  { name: "Bob Johnson", age: 35, email: "bob@example.com" }
]);
```

Read (Query) Documents

To query documents from a collection:

javascript

```
// Find all documents in the users collection
db.users.find();
```

```
// Find documents matching specific criteria
db.users.find({ age: { $gte: 30 } }); // Find users where age is greater than or equal to 30

// Find a single document by a specific field
db.users.findOne({ email: "john@example.com" });
```

Update Documents

To update existing documents in a collection:

javascript

```
// Update a single document
db.users.updateOne(
    { name: "Alice Smith" },
    { $set: { age: 26 } }
);
// Update multiple documents
db.users.updateMany(
    { age: { $gte: 30 } },
    { $set: { status: "senior" } }
);
```

Delete Documents

To delete documents from a collection:

javascript

```
// Delete a single document
db.users.deleteOne({ name: "Bob Johnson" });

// Delete multiple documents
db.users.deleteMany({ age: { $lt: 25 } }); // Delete users younger than 25
```