ICEAS IMPACT College of Engineering & Applied Sciences

LAB MANUAL

AY: 2023 – 24 (EVEN SEMESTER)

SEMESTER: 4

DEPT(s): CSE-ALLIED BRANCHES

SUBJECT NAME: DATABASE MANAGEMENT SYSTEM

SUBJECT CODE: BCS403

ICEAS IMPACT College of Engineering & Applied Sciences

PRACTICAL COMPONENT OF IPCC*(May cover all / major modules)*

Sl.NO	Experiments
1	Create a table called Employee & execute the following. **Employee(EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)** 1. Create a user and grant all permissions to theuser. 2. Insert the any three records in the employee table contains attributes EMPNO,ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result. 3. Add primary key constraint and not null constraint to the employee table. 4. Insert null values to the employee table and verify the result.
2	Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following. 1. Add a column commission with domain to the Employeetable. 2. Insert any five records into the table. 3. Update the column details of job 4. Rename the column of Employ table using alter command. 5. Delete the employee whose Empno is 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. 2. Count number of employee names from employeetable 3. Find the Maximum age from employee table. 4. Find the Minimum age from employeetable. 5. Find salaries of employee in Ascending Order. 6. Find grouped salaries of employees.
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)**
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)**
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.
7	Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.

Course outcomes (Course Skill Set):
At the end of the course, the student will be able to:

- Describe the basic elements of a relational database management system
- Design entity relationship for the given scenario.
- Apply various Structured Query Language (SQL) statements for database manipulation.
- Analyse various normalization forms for the given application.
- Develop database applications for the given real world problem.
- Understand the concepts related to NoSQL databases.

Experiment 1:

Create a table called Employee & execute the following.

create database ai_ml_b3;

use ai_ml_b3;

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

create table Employee(empno int,ename varchar(25),job varchar(20),manager_no int,sal decimal(10,2),commission decimal(8,2));

desc Employee;

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

create user 'aiml_b3'@'localhost' identified by 'aiml';

grant all privileges on Employee to 'aiml_b3'@'localhost';

2. Insert the any three records in the employee table contains attributes EMPNO,ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result.

SET autocommit=0;

INSERT INTO Employee (EMPNO, ENAME, JOB,

MANAGER_NO, SAL,COMMISSION) VALUES

(101, 'John Doe', 'Manager', NULL, 50000, 2000),

(102, 'Jane Smith', 'Developer', 101, 45000, 1500),

(103, 'Bob Johnson', 'Analyst', 101, 40000, 1000);

select * from Employee;

rollback;

select * from Employee;

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

alter table Employee add primary key(empno);

alter table Employee modify ename varchar(50) NOT NULL;

desc Employee;

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

INSERT INTO EMPLOYEE
(EMPNO,ENAME,JOB,MANAGER_NO,SAL,COMMISSION) VALUES
(105,NULL,’MANAGER’,NULL,50000,2000);

Experiment 2:

Create a table called Emp that contain attributes

EMPNO,ENAME,JOB, MGR,SAL & execute the following.

```
create table Emp(empno int primary key,ename varchar(25),
job varchar(20),mgr  int,sal decimal(10,2));
```

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

```
alter table Emp add commission decimal(8,2);
```

2. Insert any five records into the table.

```
INSERT INTO Emp (empno, ename, job,
mgr, sal,commission) VALUES
 (101, 'John Doe', 'Manager', NULL, 50000, 2000),
(102, 'Jane Smith', 'Developer', 101, 45000, 1500),
 (103, 'Bob Johnson', 'Analyst', 101, 40000, 1000),
(104, 'John Smith', 'Cashier', 101, 40000, 1800),
(105, 'Bob Doe', 'Developer', 101, 60000, 2000);
```

3. Update the column details of job

```
alter table Emp modify job varchar(100);
```

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

```
alter  table Emp rename to Emp1; (using alter) or
rename table Emp to Emp1; (without using alter)
```

5. Delete the employee whose empno is 105.

```
delete from Emp where empno=105;
```

Experiment 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 Employee(E_id int primary key,E_name varchar(50),
Age int,Salary Decimal(10,2));

Desc Employee

```
Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| e_id   | int           | NO   | PRI | NULL    |       |
| e_name | varchar(50)   | YES  |     | NULL    |       |
| age    | int           | YES  |     | NULL    |       |
| sal    | decimal(10,2) | YES  |     | NULL    |
```

Insert into Employee values(1,'John Doe',30,50000.0),
(2,'Jane Smith',25,45000.0),
(3,'Michael Johnson',35,60000.0),
(4,Emily Brown',28,48000.0),
(5,'David Lee',32,55000.0));

2. Count number of employee names from employee table
select count(e_name) as No_of_Employees from Employee;

3. Find the Maximum age from employee table.
select max(age) as MaxAge from Employee;

4. Find the Minimum age from employee table.
select min(age) as MinAge from Employee;

5. Find salaries of employee in Ascending Order.
select sal as Employee_Salary from Employee order by sal asc;

6. Find grouped salaries of employees.
select sal,count(*) as Employee_Count from Employee group by sal;

Experiment 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.

Trigger: A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated. In simple words, a trigger is a collection of SQL statements with particular names that are stored in system memory. It belongs to a specific class of stored procedures that are automatically invoked in response to database server events. Every trigger has a table attached to it.
Because a trigger cannot be called directly, unlike a stored procedure, it is referred to as a special procedure. A trigger is automatically called whenever a data modification event against a table takes place, which is the main distinction between a trigger and a procedure. On the other hand, a stored procedure must be called directly.
The following are the key differences between triggers and stored procedures:

1. Triggers cannot be manually invoked or executed.
2. There is no chance that triggers will receive parameters.
3. A transaction cannot be committed or rolled back inside a trigger.

Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

1. Create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. On [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each affected row.
6. [trigger_body]: This provides the operation to be performed as the trigger is fired

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

create table customers(id int primary key,name varchar(25),age int,address varchar(50),salary decimal(10,2));

```
insert into customers(id,name,age,address,salary) values
(1,'John Doe',30,'123, Main St, Anytown, USA',50000.0),
(2,'Jane Smith',25,'456, Elm St, Othertown, USA',60000.0),
(3,'Bob John Son',40,'789, Oak St, Anothertown, USA',75000.0);


DELIMITER    //
```

Create Trigger for INSERT Operation

```
CREATE TRIGGER after_insert_salary_difference
AFTER INSERT ON CUSTOMERS
FOR EACH ROW
BEGIN
        SET @my_sal_diff = CONCAT('salary inserted is ', NEW.SALARY);
END;//
```

Create Trigger for UPDATE Operation

```
CREATE TRIGGER after_update_salary_difference
AFTER UPDATE ON CUSTOMERS
FOR EACH ROW
BEGIN
   DECLARE old_salary DECIMAL(10, 2);
   DECLARE new_salary DECIMAL(10, 2);
   SET old_salary = OLD.SALARY;
   SET new_salary = NEW.SALARY;
        SET @my_sal_diff = CONCAT('salary difference after update is ', new_salary -
old_salary);
END;//
```

Create Trigger for DELETE Operation

```
CREATE TRIGGER after_delete_salary_difference
AFTER DELETE ON CUSTOMERS
FOR EACH ROW
BEGIN
        SET @my_sal_diff = CONCAT('salary deleted is ', OLD.SALARY);
END;//

Delimiter ;
```

Testing the Trigger:

Insert Operation:

```
INSERT INTO CUSTOMERS (id, NAME, AGE, ADDRESS, SALARY)VALUES
(4,'Shankara', 35, '123 Main St', 50000.00);

SELECT @my_sal_diff AS SAL_DIFF;
```

Update Operation:

```
UPDATE CUSTOMERS SET SALARY = 55000.00 WHERE ID = 1;

SELECT @my_sal_diff AS SAL_DIFF;
```

Delete Operation:

```
DELETE FROM CUSTOMERS WHERE ID = 1;

SELECT @my_sal_diff AS SAL_DIFF;
```

Experiment 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)

1. Creating the Employee Table and insert few records

```
CREATE DATABASE COMPANY05;

USE COMPANY05;

CREATE TABLE Employee (
   E_id INT,
   E_name VARCHAR(255),
   Age INT,
   Salary DECIMAL(10, 2)
);

INSERT INTO Employee (E_id, E_name, Age, Salary)
VALUES
   (1, 'Samarth', 30, 50000.00),
   (2, 'Ramesh Kumar', 25, 45000.00),
   (3, 'Seema Banu', 35, 62000.00),
   (4, 'Dennis Anil', 28, 52000.00),
   (5, 'Rehman Khan', 32, 58000.00);
```

2. Create a Stored Procedure with Cursor

To create a cursor for the **Employee** table, extract values using the cursor, and then close the cursor in MySQL, you’ll need to use stored procedures that support cursor operations.

```
DELIMITER //

CREATE PROCEDURE fetch_employee_data()
BEGIN
   -- Declare variables to store cursor values
   DECLARE emp_id INT;
   DECLARE emp_name VARCHAR(255);
   DECLARE emp_age INT;
   DECLARE emp_salary DECIMAL(10, 2);

   -- Declare a cursor for the Employee table
   DECLARE emp_cursor CURSOR FOR
      SELECT E_id, E_name, Age, Salary
```

```
    FROM Employee;

  -- Declare a continue handler for the cursor
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET @finished = 1;

  -- Open the cursor
  OPEN emp_cursor;

  -- Initialize a variable to control cursor loop
  SET @finished = 0;

  -- Loop through the cursor results
  cursor_loop: LOOP
    -- Fetch the next row from the cursor into variables
    FETCH emp_cursor INTO emp_id, emp_name, emp_age, emp_salary;

    -- Check if no more rows to fetch
    IF @finished = 1 THEN
      LEAVE cursor_loop;
    END IF;

    -- Output or process each row (for demonstration, print the values)
    SELECT CONCAT('Employee ID: ', emp_id, ', Name: ', emp_name, ', Age: ',
emp_age, ', Salary: ', emp_salary) AS Employee_Info;
  END LOOP;

  -- Close the cursor
  CLOSE emp_cursor;
END//

DELIMITER ;
```

In this stored procedure (**fetch_employee_data**):

- We declare variables (**emp_id**, **emp_name**, **emp_age**, **emp_salary**) to store values retrieved from the cursor.
- A cursor (**emp_cursor**) is declared to select **E_id**, **E_name**, **Age**, and **Salary** from the **Employee** table.
- We declare a continue handler (**CONTINUE HANDLER**) for **NOT FOUND** condition to handle the end of cursor data.
- The cursor is opened (**OPEN emp_cursor**), and a loop (**cursor_loop**) is used to fetch each row from the cursor.

- We fetch values into the variables and process them within the loop (for demonstration, we print the values using a **SELECT** statement).
- The loop continues until all rows are fetched (**@finished = 1**).
- Finally, the cursor is closed (**CLOSE emp_cursor**).

3. Execute the Stored Procedure
Once the stored procedure **fetch_employee_data** is created, you can execute it to fetch and process data from the **Employee** table:

CALL fetch_employee_data();
Output:
```
+-----------------------------------------------------------+
| Employee_Info                                    |
+-----------------------------------------------------------+
| Employee ID: 1, Name: Samarth, Age: 30, Salary: 50000.00 |
+-----------------------------------------------------------+
1 row in set (0.07 sec)

+-----------------------------------------------------------------+
| Employee_Info                                              |
+-----------------------------------------------------------------+
| Employee ID: 2, Name: Ramesh Kumar, Age: 25, Salary: 45000.00 |
+-----------------------------------------------------------------+
1 row in set (0.07 sec)

+---------------------------------------------------------------+
| Employee_Info                                           |
+---------------------------------------------------------------+
| Employee ID: 3, Name: Seema Banu, Age: 35, Salary: 62000.00 |
+---------------------------------------------------------------+
1 row in set (0.07 sec)

+----------------------------------------------------------------+
| Employee_Info                                            |
+----------------------------------------------------------------+
| Employee ID: 4, Name: Dennis Anil, Age: 28, Salary: 52000.00 |
+----------------------------------------------------------------+
1 row in set (0.07 sec)

+----------------------------------------------------------------+
| Employee_Info                                            |
+----------------------------------------------------------------+
| Employee ID: 5, Name: Rehman Khan, Age: 32, Salary: 58000.00 |
+----------------------------------------------------------------+
1 row in set (0.07 sec)

Query OK, 0 rows affected (0.07 sec)
```

- The stored procedure **fetch_employee_data** declares variables (**emp_id**, **emp_name**, **emp_age**, **emp_salary**) to store values retrieved from the cursor.
- A cursor (**emp_cursor**) is declared for the **Employee** table to select **E_id**, **E_name**, **Age**, and **Salary**.
- The cursor is opened (**OPEN emp_cursor**), and the **FETCH** statement retrieves the first row from the cursor into the declared variables.
- A **WHILE** loop processes each row fetched by the cursor (**SQLSTATE() = '00000'** checks for successful fetching).
- Within the loop, you can perform operations or output the values of each row.
- The **CLOSE** statement closes the cursor after processing all rows.

Experiment 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.

```
CREATE DATABASE  Exp6_b2;

Use  Exp6_b2;

CREATE TABLE N_RollCall (
    student_id INT PRIMARY KEY,
    Student_name VARCHAR (255),
    birth_date DATE);

CREATE TABLE O_RollCall (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(255),
    birth_date DATE);
```

2. Add Sample Records to both tables

```
INSERT INTO O_RollCall (student_id, student_name, birth_date)
     VALUES
         (1, 'Shivanna', '1995-08-15'),
         (3, 'Cheluva', '1990-12-10');



INSERT INTO N_RollCall (student_id, student_name, birth_date)
     VALUES
         (1, 'Shivanna', '1995-08-15'),  -- Common record with O_RollCall
         (2, 'Bhadramma', '1998-03-22'),
         (3, 'Cheluva', '1990-12-10'), -- Common record with O_RollCall
         (4, 'Devendra', '2000-05-18'),
         (5, 'Eshwar', '1997-09-03');
```

3. Define the Stored Procedure

Next, let’s define the ‘merge_rollcall_data’ stored procedure to merge records from ‘N_RollCall’ into ‘O_RollCall’, skipping existing records.

```
DELIMITER //

CREATE PROCEDURE merge_rollcall_data()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE n_id INT;
    DECLARE n_name VARCHAR(255);
    DECLARE n_birth_date DATE;
    DECLARE n_cursor CURSOR FOR
     SELECT student_id, student_name,
       birth_date FROM N_RollCall;
    DECLARE CONTINUE HANDLER FOR NOT FOUND
        SET done = TRUE;
    OPEN n_cursor;
    cursor_loop: LOOP
    FETCH n_cursor INTO n_id, n_name,
         n_birth_date;
     IF done THEN
            LEAVE cursor_loop;
     END IF;
      IF NOT EXISTS (
            SELECT 1
            FROM O_RollCall
            WHERE student_id = n_id
        ) THEN
       INSERT INTO O_RollCall (student_id,
           student_name, birth_date)
        VALUES (n_id, n_name, n_birth_date);
       END IF;
    END LOOP;
    CLOSE n_cursor;
END//

DELIMITER ;
```

- The stored procedure **merge_rollcall_data** uses a cursor (**n_cursor**) to iterate through the records of the **N_RollCall** table.
- Inside the cursor loop (**cursor_loop**), each record (**n_id**, **n_name**, **n_date**) from **N_RollCall** is fetched and checked against the **O_RollCall** table.
- If the record does not already exist in **O_RollCall** (checked using **NOT EXISTS**), it is inserted into **O_RollCall**.
- The cursor loop continues until all records from **N_RollCall** have been processed.
- The cursor is then closed (**CLOSE n_cursor**).

4. Execute the Stored Procedure
Finally, execute the ‘merge_rollcall_data’ stored procedure to merge records from ‘N_RollCall’ to ‘O_RollCall’ while skipping existing records:

```
CALL merge_rollcall_data();
```

5. Verify Records in ‘O_RollCall’

After executing the procedure, verify the records in the ‘O_RollCall’ table to confirm that new records from ‘N_RollCall’ have been inserted, while existing common records have been skipped.

```
SELECT * from O_RollCall;
+------------+--------------+------------+
| student_id | student_name | birth_date |
+------------+--------------+------------+
|          1 | Shivanna     | 1995-08-15 |<-- Common record, not duplicated
|          2 | Bhadramma    | 1998-03-22 |<-- New record from N_RollCall
|          3 | Cheluva      | 1990-12-10 |<-- Common record, not duplicated
|          4 | Devendra     | 2000-05-18 |<-- New record from N_RollCall
|          5 | Eshwar       | 1997-09-03 |<-- New record from N_RollCall
+------------+--------------+------------+
```

Experiment 7:

Install an Open Source NoSQL Data base MongoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.

1. Installing Open Source NoSQL Database MongoDB.

MongoDB is an open-source document-oriented database. It is categorized under the **NoSQL(Not only SQL)** database because the storage and retrieval of data in MongoDB are not in the form of tables. This is the general introduction to MongoDB now we will learn **how to install MongoDB in Windows.**

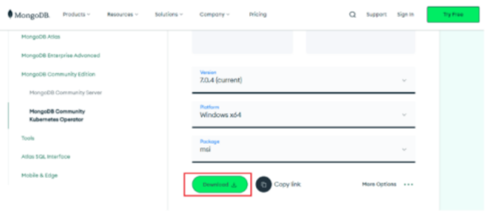
Requirements to Install MongoDB on Windows

- MongoDB 4.4 and later only support 64-bit versions of Windows.
- MongoDB 7.0 Community Edition supports the following 64-bit versions of Windows on x86_64 architecture:
  - Windows Server 2022
  - Windows Server 2019
  - Windows 11

Steps to Install MongoDB on Windows using MSI

To install MongoDB on Windows, first, download the MongoDB server and then install the MongoDB shell. The Steps below explain the installation process in detail and provide the required resources for the smooth **download and install MongoDB**.

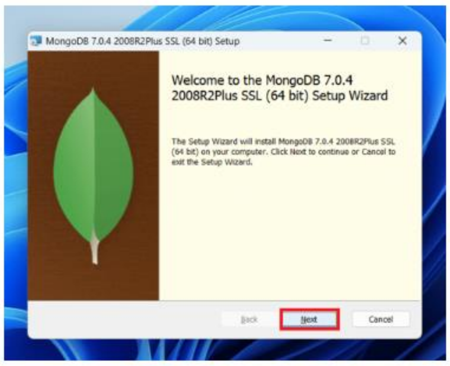
Step 1: Go to the MongoDB Download Center to download the MongoDB Community Server.


Version
7.0.4 (current)
Platform
Windows x64
Package
msi

Here, You can select any version, Windows, and package according to your requirement. For Windows, we need to choose:

- Version: 7.0.4
- OS: Windows x64
- Package: msi

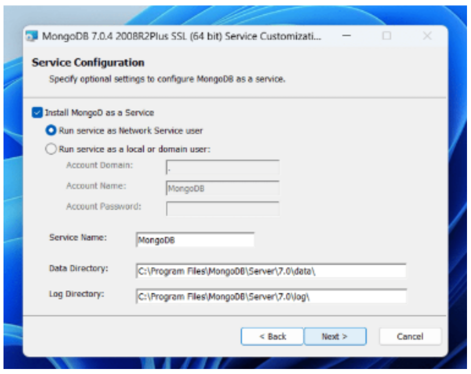
Step 2: When the download is complete open the msi file and click the *next button* in the startup screen:


MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 7.0.4 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Step 3: Now accept the End-User License Agreement and click the next button:


MongoDB 7.0.4 2008R2Plus SSL (64 bit) Service Customizati...
< Back
Next >
Cancel

Service Configuration

Specify optional settings to configure MongoDB as a service.

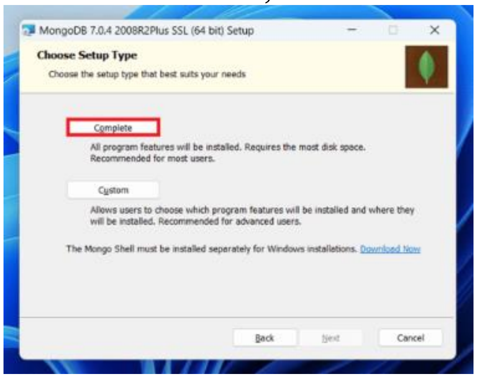
☑ Install MongoD as a Service

● Run service as Network Service user

○ Run service as a local or domain user:

Account Domain:	.
Account Name:	MongoDB
Account Password:	
Service Name:	MongoDB
Data Directory:	C:\Program Files\MongoDB\Server\7.0\data\
Log Directory:	C:\Program Files\MongoDB\Server\7.0\log\

Step 4: Now select the *complete option* to install all the program features. Here, if you can want to install only selected program features and want to select the location of the installation, then use the *Custom option:*



Choose Setup Type

Choose the setup type that best suits your needs

Complete

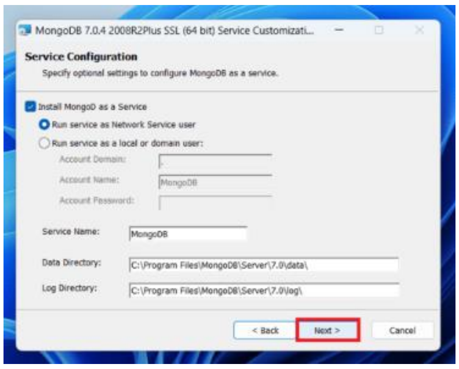
All program features will be installed. Requires the most disk space. Recommended for most users.

Custom

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

The Mongo Shell must be installed seperately for Windows installations. Download Now

Step 5: Select "Run service as Network Service user" and copy the path of the data directory. Click Next:


MongoDB 7.0.4 2008R2Plus SSL (64 bit) Service Customizati...
< Back
Next >
Cancel

Service Configuration

Specify optional settings to configure MongoDB as a service.

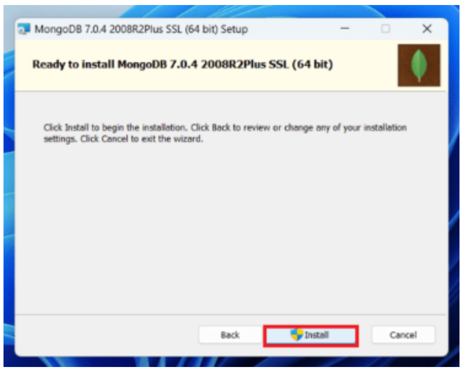
☑ Install MongoD as a Service

● Run service as Network Service user

○ Run service as a local or domain user:

Account Domain:	.
Account Name:	MongoDB
Account Password:	
Service Name:	MongoDB
Data Directory:	C:\Program Files\MongoDB\Server\7.0\data\
Log Directory:	C:\Program Files\MongoDB\Server\7.0\log\

Step 6: Click the *Install button* to start the MongoDB installation process:

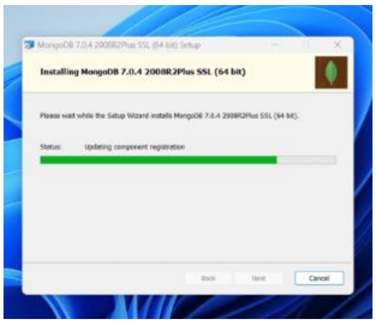


MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup

Ready to install MongoDB 7.0.4 2008R2Plus SSL (64 bit)

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Step 7: *After clicking on the install button installation of MongoDB begins:*



MongoDB 7.0.4 2008R2Plus SSL (64 bit) Setup

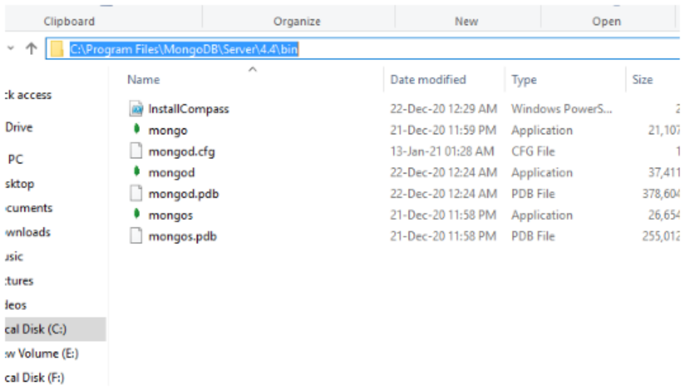
Installing MongoDB 7.0.4 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 7.0.4 2008R2Plus SSL (64 bit).

Status: Updating component registration

Step 8: *Now click the* **Finish button** *to complete the MongoDB installation process:*

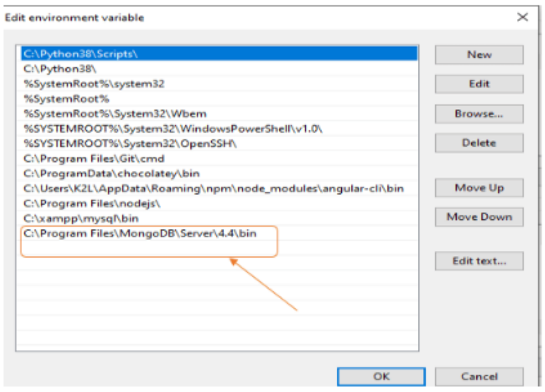
Step 9: *Now we go to the location where MongoDB installed in step 5 in your system and copy the bin path:*



C:\Program Files\MongoDB\Server\4.4\bin

Name	Date modified	Type	Size
InstallCompass	22-Dec-20 12:29 AM	Windows PowerS...	2
mongo	21-Dec-20 11:59 PM	Application	21,107
mongod.cfg	13-Jan-21 01:28 AM	CFG File	1
mongod	22-Dec-20 12:24 AM	Application	37,411
mongod.pdb	22-Dec-20 12:24 AM	PDB File	378,604
mongos	21-Dec-20 11:58 PM	Application	26,654
mongos.pdb	21-Dec-20 11:58 PM	PDB File	255,012

Step 10: *Now, to create an environment variable open system properties >> Environment Variable >> System variable >> path >> Edit Environment variable and paste the copied link to your environment system and click Ok:*


Edit environment variable

```
C:\Python38\Scripts\
C:\Python38\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Git\cmd
C:\ProgramData\chocolatey\bin
C:\Users\K2L\AppData\Roaming\npm\node_modules\angular-cli\bin
C:\Program Files\nodejs\
C:\xampp\mysql\bin
C:\Program Files\MongoDB\Server\4.4\bin
```

Step 11: *After setting the environment variable, we will run the MongoDB server, i.e. mongod. So, open the command prompt and run the following command:*

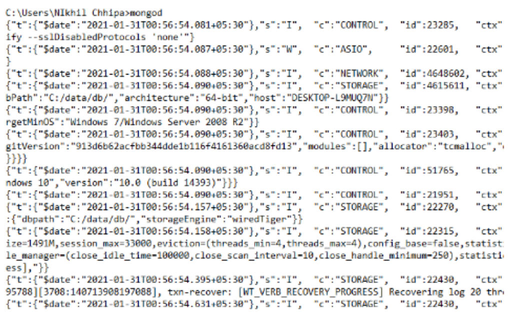
```
mongod
```

When you run this command you will get an error i.e. C:/data/db/ not found.

Step 12: *Now, Open C drive and create a folder named "data" inside this folder create another folder named "db". After creating these folders. Again open the command prompt and run the following command:*

```
mongod
```

Now, this time the MongoDB server(i.e., mongod) will run successfully.

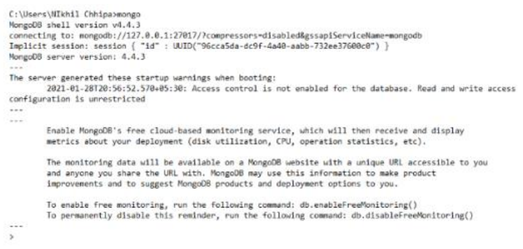


```
C:\Users\NIkhil Chhipa>mongod
{"t":{"$date":"2021-01-31T00:56:54.081+05:30"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx"
ify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2021-01-31T00:56:54.087+05:30"},"s":"W",  "c":"ASIO",     "id":22601,   "ctx"
}
{"t":{"$date":"2021-01-31T00:56:54.088+05:30"},"s":"I",  "c":"NETWORK",  "id":4648602, "ctx"
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"STORAGE",  "id":4615611, "ctx"
bPath":"C:/data/db/","architecture":"64-bit","host":"DESKTOP-L9MUQ7N"}}
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"CONTROL",  "id":23398,   "ctx"
rgetMinOS":"Windows 7/Windows Server 2008 R2"}}
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"CONTROL",  "id":23403,   "ctx"
gitVersion":"913d6b62acfbb344dde1b116f4161360acd8fd13","modules":[],"allocator":"tcmalloc","
}}}}
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"CONTROL",  "id":51765,   "ctx"
ndows 10","version":"10.0 (build 14393)"}}}
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"CONTROL",  "id":21951,   "ctx"
{"t":{"$date":"2021-01-31T00:56:54.157+05:30"},"s":"I",  "c":"STORAGE",  "id":22270,   "ctx"
:{"dbpath":"C:/data/db/","storageEngine":"wiredTiger"}}
{"t":{"$date":"2021-01-31T00:56:54.158+05:30"},"s":"I",  "c":"STORAGE",  "id":22315,   "ctx"
ize=1491M,session_max=33000,eviction=(threads_min=4,threads_max=4),config_base=false,statist
le_manager=(close_idle_time=100000,close_scan_interval=10,close_handle_minimum=250),statisti
ess],"}}
{"t":{"$date":"2021-01-31T00:56:54.395+05:30"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx"
95788][3708:140713908197088], txn-recover: [WT_VERB_RECOVERY_PROGRESS] Recovering log 20 thr
{"t":{"$date":"2021-01-31T00:56:54.631+05:30"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx"
```

Run mongo Shell

Step 13: *Now we are going to connect our server (mongod) with the mongo shell. So, keep that mongod window and open a new command prompt window and write* **mongo.** *Now, our mongo shell will successfully connect to the mongod.*

Important Point: *Please do not close the mongod window if you close this window your server will stop working and it will not able to connect with the mongo shell.*



```
C:\Users\NIkhil Chhipa>mongo
MongoDB shell version v4.4.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("96cca5da-dc9f-4a40-aabb-732ee37600c0") }
MongoDB server version: 4.4.3
---
The server generated these startup warnings when booting:
        2021-01-28T20:56:52.570+05:30: Access control is not enabled for the database. Read and write access
configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

2. Start MongoDB Shell
   Launch the MongoDB shell to perform basic CRUD operartions.
3. Switch a Database (optional)
   If you want to use a specific database switch to that database using the ‘use’ command. If the database doesn’t exist, MongoDB will create it implicitly when you insert data into it:

```
test> use bookDB
        switched to db bookDB
bookDB>
```

4. Create the ‘ProgrammingBooks’ Collection:
   To create the **ProgrammingBooks** collection, use the **createCollection()** method. This step is optional because MongoDB will automatically create the collection when you insert data into it, but you can explicitly create it if needed:
   bookDB> db.createCollection("ProgrammingBooks")
   This command will create an empty **ProgrammingBooks** collection in the current database (**bookDB**).
5. Insert Operations:
   a. Insert 5 Documents into the ProgrammingBooks Collection :

   Now, insert 5 documents representing programming books into the ProgrammingBooks collection using the insertMany() method:

```
bookDB> db.ProgrammingBooks.insertMany([
  {
   title: "Clean Code: A Handbook of Agile Software Craftsmanship",
   author: "Robert C. Martin",
   category: "Software Development",
   year: 2008
  },
  {
   title: "JavaScript: The Good Parts",
   author: "Douglas Crockford",
   category: "JavaScript",
   year: 2008
  },
  {
   title: "Design Patterns: Elements of Reusable Object-Oriented Software",
   author: "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides",
   category: "Software Design",
   year: 1994
  },
  {
   title: "Introduction to Algorithms",
   author: "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford
Stein",
   category: "Algorithms",
   year: 1990
  },
  {
    title: "Python Crash Course: A Hands-On, Project-Based Introduction to
Programming",
    author: "Eric Matthes",
    category: "Python",
    year: 2015
  }
])
```

b. Insert a Single Document into ProgrammingBooks:

Use the insertOne() method to insert a new document into the ProgrammingBooks collection:

```
bookDB> db.ProgrammingBooks.insertOne({
  title: "The Pragmatic Programmer: Your Journey to Mastery",
  author: "David Thomas, Andrew Hunt",
  category: "Software Development",
  year: 1999
})
```

6. Read(Query) Operations:
   a. Find all the Documents.
      To retrieve all the documents from the 'ProgrammingBooks' collection:

```
bookDB> db.ProgrammingBooks.find().pretty()
```

Output:

```
[
  {
    _id: ObjectId('……'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('……'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('……'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Software Design',
    year: 1994
  },
  {
    _id: ObjectId('……'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford
Stein',
    category: 'Algorithms',
    year: 1990
  },
  {
    _id: ObjectId('……'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  },
  {
    _id: ObjectId('……'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Software Development',
    year: 1999
  }
]
```

b. Find Documents Matching a Condition:
   To find books published after the year 2000:

```
bookDB> db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()
[
  {
    _id: ObjectId('……'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('……'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('……'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

7. Update Operations:
   a. Update Single Document:
      To update a specific book (eg. Change the author of a book)

```
bookDB>db.ProgrammingBooks.updateOne(
  { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
  { $set: { author: "Robert C. Martin (Uncle Bob)" } }
)
```

//verify by displaying books published in year 2008

```
bookDB> db.ProgrammingBooks.find({ year: { $eq: 2008 } }).pretty()
[
  {
    _id: ObjectId('…………..'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('…………..'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  }
]
```

   b. Update Multiple Documents:
      To update multiple books (e.g., update the category of books published before 2010):

```
bookDB> db.ProgrammingBooks.updateMany(
  { year: { $lt: 2010 } },
  { $set: { category: "Classic Programming Books" } }
)
```

//verify the update operation by displaying books published before year 2010

```
bookDB> db.ProgrammingBooks.find({ year: { $lt: 2010 } }).pretty()
[
  {
    _id: ObjectId('…………..'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('…………..'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('…………..'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Classic Programming Books',
    year: 1994
  },
  {
    _id: ObjectId('…………..'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford
Stein',
    category: 'Classic Programming Books',
    year: 1990
  },
  {
    _id: ObjectId('…………..'),
    title: 'The Pragmatic Programmer: Your Journey to Mastery',
    author: 'David Thomas, Andrew Hunt',
    category: 'Classic Programming Books',
    year: 1999
  }
]
```

8. Delete Operations:

   a. Delete a Single Document:

   To delete a specific book from the collection (e.g., delete a book by title):

```
bookDB> db.ProgrammingBooks.deleteOne({ title: "JavaScript: The Good Parts" })
{ acknowledged: true, deletedCount: 1 }
```

You can check whether the specified document is deleted by displaying the contents of the collection.

   b. Delete Multiple Documents:

To delete multiple books based on a condition (e.g., delete all books published before 1995):

```
bookDB> db.ProgrammingBooks.deleteMany({ year: { $lt: 1995 } })
{ acknowledged: true, deletedCount: 2 }
```

You can check whether the specified documents were deleted by displaying the contents of the collection.

   c. Delete all the Documents in a Collection:

   To delete all documents in a collection (e.g., **ProgrammingBooks**), use the **deleteMany()** method with an empty filter **{}**:

//delete all documents in a collection

```
bookDB> db.ProgrammingBooks.deleteMany({})
{ acknowledged: true, deletedCount: 3 }
```

//verify by displaying the collection

```
bookDB> db.ProgrammingBooks.find().pretty()
```

9. Delete the Collection using drop():
   To delete a collection named **ProgrammingBooks**, use the **drop()** method with the name of the collection:

```
bookDB> show collections
ProgrammingBooks

bookDB> db.ProgrammingBooks.drop()
true

bookDB> show collections

bookDB>
```

The command **db.ProgrammingBooks.drop( )** will permanently delete the **ProgrammingBooks** collection from the current database (**bookDB**).

After deleting the collection, you can verify that it no longer exists by listing all collections in the database using the command ‘**show collections**’.