PRACTICALS

✅ PART – 1: PRACTICAL

1. Multi-indexing

```
import pandas as pd

data = {
    "Country": ["India", "India", "USA", "USA"],
    "Year": [2020, 2021, 2020, 2021],
    "GDP": [2.9, 3.1, 21.4, 22.0]
}

df = pd.DataFrame(data)
df = df.set_index(["Country", "Year"])
print(df)
```

Task:

- Print GDP of India for 2021
- Print all data for USA

---

2. Merge two DataFrames

```
students = pd.DataFrame({
    "Roll": [1, 2, 3],
    "Name": ["Amit", "Reena", "John"]
})

marks = pd.DataFrame({
    "Roll": [1, 2, 3],
    "Marks": [88, 92, 75]
})

result = pd.merge(students, marks, on="Roll")
print(result)
```

Task:

- Add a new column "Grade" based on Marks
- Display only students with Marks > 80

---

3. Pivot Table

```
sales = pd.DataFrame({
    "Product": ["Pen", "Pen", "Book", "Book"],
    "Year": [2021, 2022, 2021, 2022],
    "Sales": [50, 60, 100, 120]
})
```

Task:

- Create pivot: Product vs Year
- Display only 2022 sales

---

4. Melt Example

```
df = pd.DataFrame({
    "Name": ["Amit"],
```

```
    "Math": [90],
    "English": [85]
})
```

Task:

- Convert into long format using melt()

---

5. GroupBy & Aggregation

```
df = pd.DataFrame({
    "Dept": ["IT", "IT", "HR", "HR"],
    "Salary": [50000, 55000, 40000, 42000]
})
```

Task:

- Calculate average salary of each department
- Find min & max salary of each department

---

6. Rolling Window

```
import pandas as pd

s = pd.Series([10, 20, 30, 40, 50])
```

Task:

- Calculate rolling average with window = 3
- Create cumulative sum

✅ PART – 2: PARCTICALS

Assignment: Intermediate Data Analysis Using Pandas
Course: Data Analysis Using Python
Unit: 1 – Intermediate Pandas Operations
Total Marks: 30

Q1. Multi-Indexing (5 marks)

Using the given dataset:

Country	State	Year	Population
India	Gujarat	2020	63
India	Gujarat	2021	64
India	Maharashtra	2020	112
India	Maharashtra	2021	113
USA	California	2020	40
USA	California	2021	41

Tasks:

a) Create a DataFrame
b) Set Country, State, Year as Multi-index
c) Retrieve population of Gujarat in 2021
d) Retrieve all data of USA

Q2. Merging (5 marks)

Using datasets:

Students.csv

Roll	Name	Dept
1	Amit	IT
2	Sneha	HR
3	Raj	IT

Marks.csv

Roll	Subject	Marks
1	Python	85
2	Python	78
3	Python	92

Tasks:

a) Merge both datasets on Roll
b) Display students who scored more than 80
c) Group by Department and find average score

Q3. Pivoting & Reshaping (5 marks)

Using:

Product	Quarter	Sales
Pen	Q1	120
Pen	Q2	135
Pencil	Q1	90
Pencil	Q2	110

Tasks:

a) Create a pivot table (Product vs Quarter)
b) Convert pivot back into long format using melt()

Q4. GroupBy & Aggregation (7 marks)

Using:

Emp	Dept	Salary
E1	IT	50000
E2	IT	52000
E3	HR	42000
E4	HR	43000

Tasks:

a) Group by Dept and calculate:

- mean
- max
- min

b) Count number of employees in each Department

Q5. Rolling Functions (8 marks)

Given sales data:

Day	Sales
1	100

2	200
3	300
4	400
5	500

Tasks:
a) Find rolling average with window=3
b) Create cumulative sum
c) Plot rolling mean graph (optional, extra credit)

1.2 Efficient Computation with NumPy
1.2.1 Broadcasting, slicing, filtering, conditional logic
1.2.2 Matrix algebra and statistical computations
These examples are designed for students who understand Python basics and want hands-on NumPy practice.

✅ **1.2.1 Broadcasting – Practical Examples**

📌 **Example 1: Add a 1D Array to Every Row of a 2D Array**

```
import numpy as np

A = np.array([
    [10, 20, 30],
    [40, 50, 60],
    [70, 80, 90]
])

b = np.array([1, 2, 3])   # 1D vector

result = A + b
print(result)
```

✓ **Output:**
Every row gets **[1,2,3]** added automatically (broadcasting).

📌 **Example 2: Normalize Columns Using Broadcasting**

```
A = np.array([[5, 10, 15],
              [2, 4,  6],
              [8, 16, 24]])

col_mean = A.mean(axis=0)

normalized = A - col_mean     # broadcasting subtracts mean from each column
print(normalized)
```

📌 **Example 3: Scalar Broadcasting**

```
arr = np.array([10, 20, 30, 40])
result = arr * 1.18   # 18% GST added
print(result)
```

✅ 1.2.1 Slicing – Practical Examples

📌 Example 4: Slice Middle Elements of a Matrix

```
A = np.arange(1, 17).reshape(4,4)
print(A[1:3, 1:3])   # take center 2x2 block
```

📌 Example 5: Reverse a Matrix Row-wise & Column-wise

```
A = np.arange(1, 10).reshape(3,3)

print(A[::-1])       # reverse rows
print(A[:, ::-1])    # reverse columns
```

✅ 1.2.1 Filtering & Conditional Logic

📌 Example 6: Filter Values Greater Than 50

```
A = np.array([10, 55, 72, 30, 99])
filtered = A[A > 50]
print(filtered)
```

📌 Example 7: Replace Negative Values with Zero

```
A = np.array([5, -2, 8, -10, 3])
A[A < 0] = 0
print(A)
```

📌 Example 8: Filter Rows with Condition (2D Array)

Example: Select rows where column 1 > 5

```
A = np.array([
    [1, 3],
    [4, 7],
    [6, 9],
    [2, 10]
])

result = A[A[:, 1] > 5]
print(result)
```

🚩 Combination Example (Broadcasting + Filtering)

📌 Example 9: Grading System

Add 5 bonus marks, then select students scoring ≥ 40.

```
marks = np.array([32, 67, 25, 90, 55])
curved = marks + 5              # broadcasting
passed = curved[curved >= 40]   # filtering
print(passed)
```

✅ 1.2.2 Matrix Algebra – Intermediate Examples

Import NumPy:
import numpy as np

📌 Example 10: Matrix Multiplication

```
A = np.array([[1, 2],
              [3, 4]])

B = np.array([[5, 6],
              [7, 8]])

C = A @ B
print(C)
```

✓ @ or np.dot() performs matrix multiplication.

📌 Example 11: Find the Determinant

```
A = np.array([[3, 8],
              [4, 6]])

det = np.linalg.det(A)
print(det)
```

📌 Example 12: Solve a System of Linear Equations

Solve:
3x + y = 9
x + 2y = 8

```
A = np.array([[3, 1],
              [1, 2]])

b = np.array([9, 8])

solution = np.linalg.solve(A, b)
print(solution)
```

📌 Example 13: Matrix Inverse

```
A = np.array([[2, 5],
              [1, 3]])

invA = np.linalg.inv(A)
print(invA)
```

📌 Example 14: Eigenvalues & Eigenvectors

(Used in PCA, ML algorithms)

```
A = np.array([[4, 2],
              [1, 3]])

vals, vecs = np.linalg.eig(A)
```

```
print("Eigenvalues:", vals)
print("Eigenvectors:\n", vecs)
```

✅ 1.2.2 Statistical Computations

📌 Example 15: Mean, Median, Variance, Std Dev

```
data = np.array([12, 18, 25, 30, 45])

print("Mean:", data.mean())
print("Median:", np.median(data))
print("Variance:", data.var())
print("Std Dev:", data.std())
```

📌 Example 16: Column-wise Stats for 2D Array

```
A = np.array([
    [10, 20, 30],
    [40, 50, 60],
    [70, 80, 90]
])

print(A.mean(axis=0))   # column-wise mean
print(A.sum(axis=1))    # row-wise sum
```

📌 Example 17: Z-score Normalization (used in ML)

```
A = np.array([10, 20, 30, 40, 50])

z = (A - A.mean()) / A.std()
print(z)
```

📌 Example 18: Covariance Matrix

```
X = np.array([1,2,3,4,5])
Y = np.array([2,4,5,4,5])

cov_matrix = np.cov(X, Y)
print(cov_matrix)
```

📌 Example 19: Correlation Coefficient

```
corr = np.corrcoef(X, Y)
print(corr)
```

1.3 Working with Complex Data

1.3.1 Date/time series operations

1.3.2 Reading and writing data from Excel, JSON, XML

Example 1: Monthly Sales Report (Time Series)

```
import pandas as pd
```

```
df = pd.DataFrame({
    "date": pd.date_range(start="2024-01-01", periods=10, freq="D"),
    "sales": [100,120,90,110,150,170,130,125,160,180]
})

df['date'] = pd.to_datetime(df['date'])
df = df.set_index('date')

monthly_sales = df.resample('M').sum()
print(monthly_sales)
```

🔷 Example 2: Extracting Yearly & Weekly Trends

```
df['year'] = df.index.year
df['week'] = df.index.isocalendar().week
print(df.groupby('year')['sales'].sum())
print(df.groupby('week')['sales'].mean())
```

🔷 Example 3: Reading Multiple Excel Sheets

```
xls = pd.ExcelFile("company.xlsx")

employees = pd.read_excel(xls, "Employees")
departments = pd.read_excel(xls, "Departments")
```

🔷 Example 4: Converting JSON API Data to DataFrame

```
json_data = [
    {"id":1, "name":"Amit", "address": {"city":"Surat","state":"GJ"}},
    {"id":2, "name":"Neha", "address": {"city":"Pune","state":"MH"}}
]

df = pd.json_normalize(json_data)
print(df)
```

🔷 Example 5: Load XML Student Records

```
data = pd.read_xml("students.xml")
print(data.head())
```

Unit-2: Database Integration and Data Preprocessing

Each file has **one clear responsibility**, which is good practice for students and projects.

📌 1. insert.py

Create table and insert records

```
import sqlite3

# Connect to database
conn = sqlite3.connect("college.db")
cursor = conn.cursor()

# Create table
cursor.execute("""
CREATE TABLE IF NOT EXISTS students(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    marks INTEGER
)
""")

# Insert single record
cursor.execute("INSERT INTO students(name, marks) VALUES (?, ?)", ("Rahul", 85))

# Insert multiple records
students = [
    ("Amit", 90),
    ("Neha", 82),
    ("Karan", 95)
]
cursor.executemany("INSERT INTO students(name, marks) VALUES (?, ?)",
students)

conn.commit()
print("Table created and data inserted successfully.")

conn.close()
```

📌 2. update.py

Update an existing record

```
import sqlite3

conn = sqlite3.connect("college.db")
cursor = conn.cursor()

# Update marks of a student
cursor.execute("UPDATE students SET marks = ? WHERE name = ?", (88, "Amit"))
```

```
conn.commit()
print("Record updated successfully.")

conn.close()
```

📌 3. delete.py

Delete a specific record

```
import sqlite3

conn = sqlite3.connect("college.db")
cursor = conn.cursor()

# Delete student by name
cursor.execute("DELETE FROM students WHERE name = ?", ("Karan",))

conn.commit()
print("Record deleted successfully.")

conn.close()
```

📌 4. drop.py

Drop table if it exists

```
import sqlite3

conn = sqlite3.connect("college.db")
cursor = conn.cursor()

# Drop table
cursor.execute("DROP TABLE IF EXISTS students")

conn.commit()
print("Table dropped successfully (if it existed).")

conn.close()
```

📌 5. tables.py

Display all tables in the database

```
import sqlite3

conn = sqlite3.connect("college.db")
cursor = conn.cursor()

# Fetch all tables
cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
tables = cursor.fetchall()

print("Tables in the database:")
```

```
for table in tables:
    print(table[0])

conn.close()
```

📌 6. `display.py`

****Display all records from the table****

```
import sqlite3

# Connect to database
conn = sqlite3.connect("college.db")
cursor = conn.cursor()

# Fetch all student records
cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()

# Display records
print("Students Records:")
print("ID\tName\tMarks")
print("-" * 30)

for row in rows:
    print(f"{row[0]}\t{row[1]}\t{row[2]}")

# Close connection
conn.close()
```

OUTPUT

```
Students Records:
ID  Name    Marks
------------------------------
1   Rahul   85
2   Amit    88
3   Neha    82
```

Student Tip (Exam Point of View)

* `fetchall()` → fetches all rows
* `fetchone()` → fetches one row
* `fetchmany(n)` → fetches `n` rows

Below is a FULL CRUD (Create, Read, Update, Delete) example using Pandas with SQLite. This is exam-ready, practical, and beginner-friendly. You can run it directly.
Full CRUD using Pandas + SQLite (Complete Code)

```
import pandas as pd
```

```
import sqlite3

# Step 1: Connect to SQLite Database
conn = sqlite3.connect("college.db")

# ---------------- CREATE ----------------
# Create DataFrame (Insert Records)
students_df = pd.DataFrame({
    "name": ["Amit", "Neha", "Rahul"],
    "marks": [85, 90, 78]
})

# Insert DataFrame into SQLite Table
students_df.to_sql(
    name="students_pandas",
    con=conn,
    if_exists="replace",   # replace table if exists
    index=False
)

print("CREATE: Data inserted successfully\n")

# ---------------- READ ----------------
# Read data from SQLite into DataFrame
read_df = pd.read_sql_query(
    "SELECT * FROM students_pandas",
    conn
)

print("READ: Data from database")
print(read_df, "\n")

# ---------------- UPDATE ----------------
# Update marks of Amit to 95
read_df.loc[read_df["name"] == "Amit", "marks"] = 95

# Write updated DataFrame back to SQLite
read_df.to_sql(
    name="students_pandas",
    con=conn,
    if_exists="replace",
    index=False
)

print("UPDATE: Marks updated\n")

# Verify update
updated_df = pd.read_sql_query(
    "SELECT * FROM students_pandas",
    conn
)
print(updated_df, "\n")

# ---------------- DELETE ----------------
# Delete record where name = 'Rahul'
delete_df = updated_df[updated_df["name"] != "Rahul"]

# Write DataFrame after deletion
```

```
delete_df.to_sql(
    name="students_pandas",
    con=conn,
    if_exists="replace",
    index=False
)

print("DELETE: Record deleted\n")

# Final data check
final_df = pd.read_sql_query(
    "SELECT * FROM students_pandas",
    conn
)
print("FINAL DATA:")
print(final_df)

# Close Connection
conn.close()
```

CRUD Explanation (For Exams)

Operation	Pandas Action	SQLite Action
CREATE	DataFrame + to_sql()	Insert records
READ	read_sql_query()	Fetch data
UPDATE	Modify DataFrame	Replace table
DELETE	Filter rows	Replace table

Menu-Driven CRUD system using SQLite (Python).

This **single program** combines **Create, Insert, Display, Update, Delete, Drop, and Show Tables** — perfect for **lab practicals, viva, and mini-projects**.

📁 **Menu-Driven CRUD System (SQLite + Python)** =📄 **crud_menu.py**

```
import sqlite3

# ---------------- DATABASE CONNECTION ----------------
def connect_db():
    return sqlite3.connect("college.db")

# ---------------- CREATE TABLE ----------------
def create_table():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS students(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT,
        marks INTEGER
    )
    """)
    conn.commit()
    conn.close()
    print("Table created successfully.")
```

```
# ---------------- INSERT RECORD ----------------
def insert_record():
    conn = connect_db()
    cursor = conn.cursor()

    name = input("Enter student name: ")
    marks = int(input("Enter marks: "))

    cursor.execute("INSERT INTO students(name, marks) VALUES (?, ?)", (name,
marks))
    conn.commit()
    conn.close()
    print(" Record inserted successfully.")

# ---------------- DISPLAY RECORDS ----------------
def display_records():
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM students")
    rows = cursor.fetchall()

    print("\nID\tName\tMarks")
    print("-" * 30)
    for row in rows:
        print(f"{row[0]}\t{row[1]}\t{row[2]}")

    conn.close()

# ---------------- UPDATE RECORD ----------------
def update_record():
    conn = connect_db()
    cursor = conn.cursor()

    sid = int(input("Enter student ID to update: "))
    new_marks = int(input("Enter new marks: "))

    cursor.execute("UPDATE students SET marks = ? WHERE id = ?", (new_marks,
sid))
    conn.commit()
    conn.close()
    print(" Record updated successfully.")

# ---------------- DELETE RECORD ----------------
def delete_record():
    conn = connect_db()
    cursor = conn.cursor()
```

```
    sid = int(input("Enter student ID to delete: "))
    cursor.execute("DELETE FROM students WHERE id = ?", (sid,))
    conn.commit()
    conn.close()
    print(" Record deleted successfully.")

# ---------------- DROP TABLE ----------------
def drop_table():
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("DROP TABLE IF EXISTS students")
    conn.commit()
    conn.close()
    print(" Table dropped successfully.")

# ---------------- SHOW TABLES ----------------
def show_tables():
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
    tables = cursor.fetchall()

    print("\nTables in database:")
    for table in tables:
        print(table[0])

    conn.close()

# ---------------- MAIN MENU ----------------
def main_menu():
    while True:
        print("""
========= STUDENT DATABASE MENU =========
1. Create Table
2. Insert Record
3. Display Records
4. Update Record
5. Delete Record
6. Drop Table
7. Show Tables
8. Exit
=========================================
""")
        choice = input("Enter your choice (1-8): ")

        if choice == "1":
            create_table()
```

```
        elif choice == "2":
            insert_record()
        elif choice == "3":
            display_records()
        elif choice == "4":
            update_record()
        elif choice == "5":
            delete_record()
        elif choice == "6":
            drop_table()
        elif choice == "7":
            show_tables()
        elif choice == "8":
            print(" Exiting program. Thank you!")
            break
        else:
            print("Invalid choice. Try again.")

# ---------------- PROGRAM START ----------------
main_menu()
```

▶️ **Sample Menu Output**

```
========= STUDENT DATABASE MENU =========
1. Create Table
2. Insert Record
3. Display Records
4. Update Record
5. Delete Record
6. Drop Table
7. Show Tables
8. Exit
=========================================
```

2.1.2 Connecting to MongoDB using pymongo

Data Analytics / Python practical

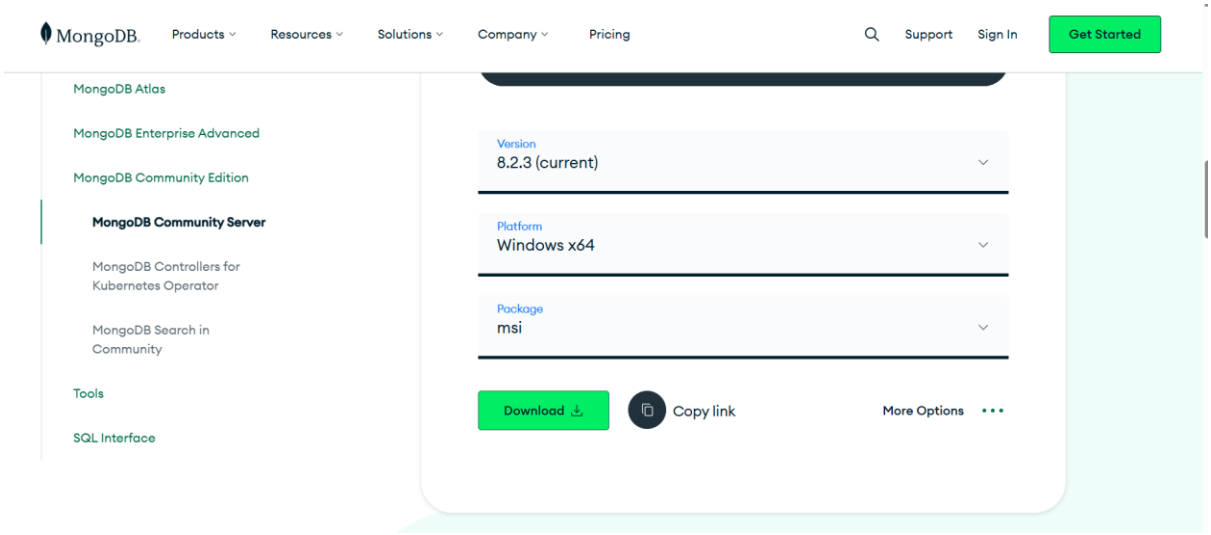
Below is a **complete step-by-step guide** to **install MongoDB** and **connect it with Python for CRUD operations**. Language is **simple**, **practical**, and **viva-ready**.

🔷 **STEP 1: What is MongoDB?**

MongoDB is a **NoSQL, document-based database** where data is stored in **collections** as **JSON-like documents** instead of tables.

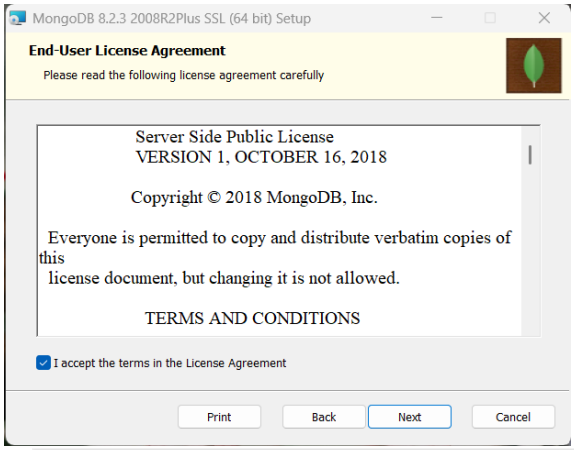
🔷 **STEP 2: Download MongoDB (Windows)**

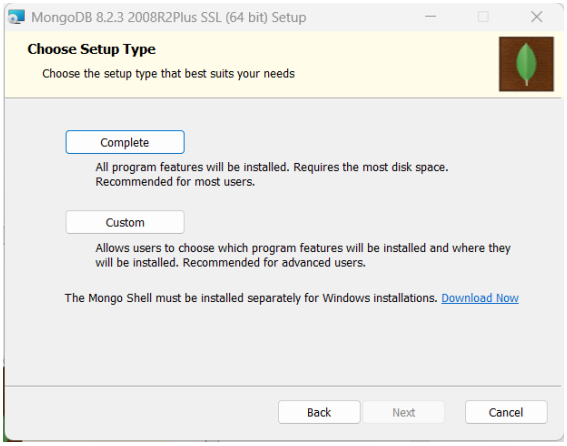
https://www.mongodb.com/try/download/community

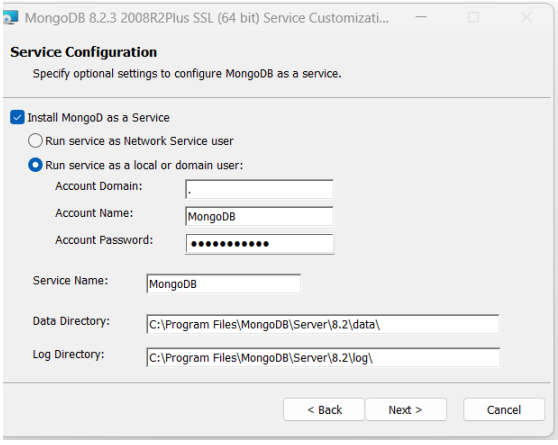

MongoDB
Products
Resources
Solutions
Company
Pricing
Support
Sign In
Get Started
MongoDB Atlas
MongoDB Enterprise Advanced
MongoDB Community Edition
MongoDB Community Server
MongoDB Controllers for Kubernetes Operator
MongoDB Search in Community
Tools
SQL Interface
Version
8.2.3 (current)
Platform
Windows x64
Package
msi
Download
Copy link
More Options


Try MongoDB Community Edition
The community version of our distributed document database provides powerful ways to query and analyze your data.


MongoDB 8.2.3 2008R2Plus SSL (64 bit) Setup
Welcome to the MongoDB 8.2.3 2008R2Plus SSL (64 bit) Setup Wizard
The Setup Wizard will install MongoDB 8.2.3 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel


MongoDB 8.2.3 2008R2Plus SSL (64 bit) Setup
End-User License Agreement
Please read the following license agreement carefully
Server Side Public License
VERSION 1, OCTOBER 16, 2018
Copyright © 2018 MongoDB, Inc.
Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.
TERMS AND CONDITIONS
I accept the terms in the License Agreement
Print
Back
Next
Cancel

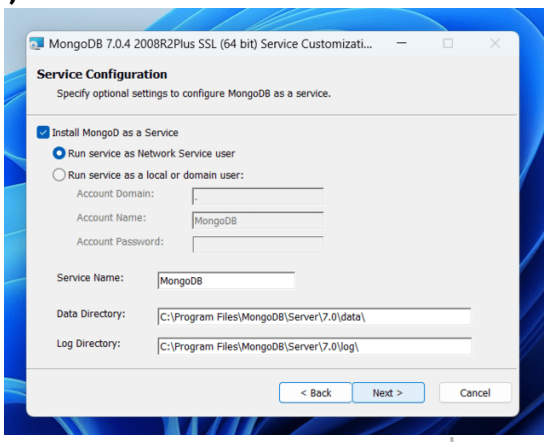

MongoDB 8.2.3 2008R2Plus SSL (64 bit) Setup
Choose Setup Type
Choose the setup type that best suits your needs
Complete
All program features will be installed. Requires the most disk space. Recommended for most users.
Custom
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.
The Mongo Shell must be installed separately for Windows installations. Download Now
Back
Next
Cancel

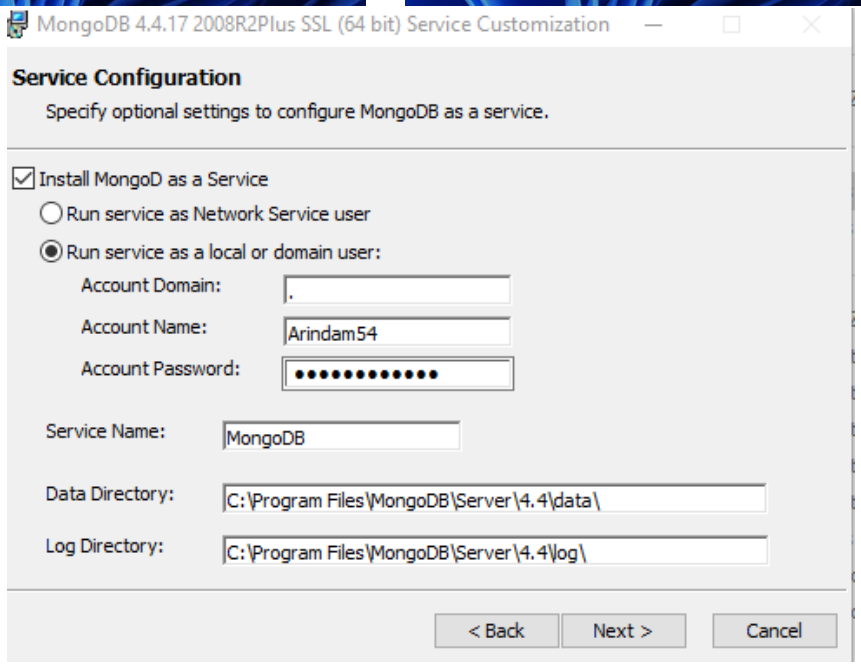

MongoDB 8.2.3 2008R2Plus SSL (64 bit) Service Customizati...
Service Configuration
Specify optional settings to configure MongoDB as a service.
Install MongoD as a Service
Run service as Network Service user
Run service as a local or domain user:
Account Domain:
Account Name:
MongoDB
Account Password:
Service Name:
MongoDB
Data Directory:
C:\Program Files\MongoDB\Server\8.2\data\
Log Directory:
C:\Program Files\MongoDB\Server\8.2\log\
< Back
Next >
Cancel

✓ **Steps**

1. Open browser → https://www.mongodb.com
2. Go to **Products** → **MongoDB Community Server**
3. Select:
   - OS: **Windows**
   - Package: **MSI**
4. Click **Download**

🔷 STEP 3: Install MongoDB (Windows)



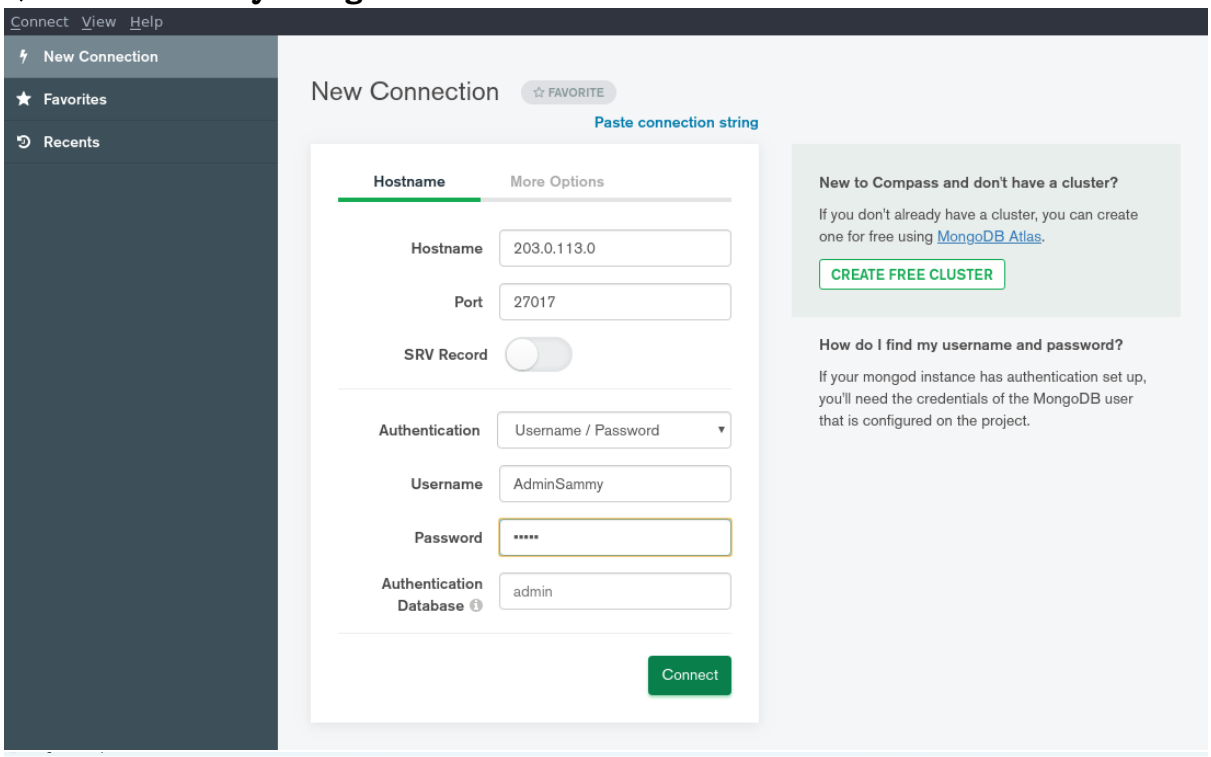


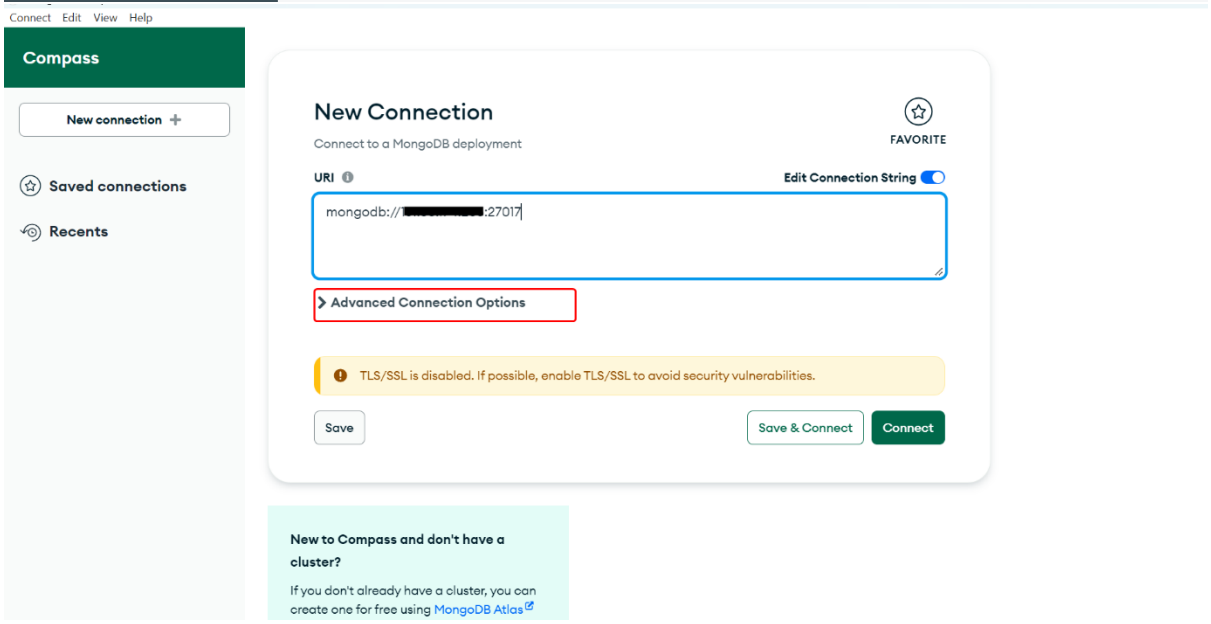
✓ **Installation Steps**

1. Double-click downloaded **.msi** file
2. Click **Next**
3. Accept License Agreement
4. Choose **Complete Installation**
5. ✓ Check **Install MongoDB as a Service**
6. Service Name: MongoDB Server
7. Click **Install**
8. Finish installation

✅ MongoDB service starts automatically

🔷 STEP 4: Verify MongoDB Installation





Method 1: Using MongoDB Compass (GUI)

1. Open **MongoDB Compass**
2. Connection String:
3. mongodb://localhost:27017
4. Click **Connect**

If connected → MongoDB installed successfully ✅

🔷 STEP 5: Install Python MongoDB Driver (PyMongo)

✓ **Command**

pip install pymongo

✓ PyMongo is required to connect **Python ↔ MongoDB**

🔷 STEP 6: Connect MongoDB with Python

📄 Python Connection Code

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["collegeDB"]
collection = db["students"]

print("Connected to MongoDB successfully")
```

✓ collegeDB → Database

✓ students → Collection (auto-created)

🔷 STEP 7: CRUD Operations Using Python + MongoDB

🟢 INSERT (Create)

```
collection.insert_one({
    "name": "Rahul",
    "marks": 85
})
print("Record inserted")
```

🔵 READ (Display)

```
for student in collection.find():
    print(student)
```

🟡 UPDATE

```
collection.update_one(
    {"name": "Rahul"},
    {"$set": {"marks": 90}}
)
print("Record updated")
```

🔴 DELETE

```
collection.delete_one({"name": "Rahul"})
print("Record deleted")
```

🔷 STEP 8: Show All Databases & Collections

✓ **Show Databases**

```
print(client.list_database_names())
```

✓ **Show Collections**

```
print(db.list_collection_names())
```

📝 Exam / Viva Important Points

Topic	Key Point
MongoDB Type	NoSQL Database
Data Storage	BSON (Binary JSON)
Default Port	27017
Primary Key	_id (auto-generated)
Python Driver	PyMongo
Schema	Schema-less
Join Support	❌ No (uses embedding)

🧠 **SUMMARY**

MongoDB is a NoSQL document-based database that stores data in JSON-like format. Python connects to MongoDB using the PyMongo library to perform CRUD operations.

Menu-Driven CRUD System using MongoDB (Python)

📌 **Prerequisites**

```
pip install pymongo
```

Make sure **MongoDB Server** is running
(Default URL: mongodb://localhost:27017/)

mongo_crud_menu.py

```
from pymongo import MongoClient

# ---------------- DATABASE CONNECTION ----------------
def connect_db():
    client = MongoClient("mongodb://localhost:27017/")
    db = client["collegeDB"]
    collection = db["students"]
    return collection

# ---------------- INSERT RECORD ----------------
def insert_record():
    collection = connect_db()

    name = input("Enter student name: ")
    marks = int(input("Enter marks: "))

    student = {
        "name": name,
        "marks": marks
    }

    collection.insert_one(student)
    print(" Record inserted successfully.")

# ---------------- DISPLAY RECORDS ----------------
def display_records():
    collection = connect_db()
```

```
    records = collection.find()

    print("\nID\t\t\t\tName\tMarks")
    print("-" * 60)

    for record in records:
        print(f"{record['_id']}\t{record['name']}\t{record['marks']}")

# ---------------- UPDATE RECORD ----------------
def update_record():
    collection = connect_db()

    name = input("Enter student name to update: ")
    new_marks = int(input("Enter new marks: "))

    result = collection.update_one(
        {"name": name},
        {"$set": {"marks": new_marks}}
    )

    if result.matched_count > 0:
        print(" Record updated successfully.")
    else:
        print("Record not found.")

# ---------------- DELETE RECORD ----------------
def delete_record():
    collection = connect_db()

    name = input("Enter student name to delete: ")
    result = collection.delete_one({"name": name})

    if result.deleted_count > 0:
        print("Record deleted successfully.")
    else:
        print("Record not found.")

# ---------------- DROP COLLECTION ----------------
def drop_collection():
    collection = connect_db()
    collection.drop()
    print(" Students collection dropped successfully.")

# ---------------- SHOW COLLECTIONS ----------------
def show_collections():
    client = MongoClient("mongodb://localhost:27017/")
    db = client["collegeDB"]

    print("\nCollections in Database:")
```

```
    for col in db.list_collection_names():
        print(col)

# ---------------- MAIN MENU ----------------
def main_menu():
    while True:
        print("""
========= MONGODB STUDENT MENU =========
1. Insert Record
2. Display Records
3. Update Record
4. Delete Record
5. Drop Collection
6. Show Collections
7. Exit
========================================
""")

        choice = input("Enter your choice (1-7): ")

        if choice == "1":
            insert_record()
        elif choice == "2":
            display_records()
        elif choice == "3":
            update_record()
        elif choice == "4":
            delete_record()
        elif choice == "5":
            drop_collection()
        elif choice == "6":
            show_collections()
        elif choice == "7":
            print("Exiting program.")
            break
        else:
            print(" Invalid choice. Try again.")

# ---------------- PROGRAM START ----------------
main_menu()
```

OUTPUT

```
========= MONGODB STUDENT MENU =========
1. Insert Record
2. Display Records
3. Update Record
4. Delete Record
5. Drop Collection
6. Show Collections
```

7. Exit

==========

==========

💡 PRACTICAL LAB EXERCISES

==========

Exercise 1: SQLite

1. Create a database library.db
2. Create table books(id, title, author, price)
3. Insert 5 books
4. Display all books
5. Update price of a book
6. Delete a book whose price < 300
7. Load the table using Pandas

Exercise 2: MongoDB

1. Create database shop
2. Insert 5 product records
3. Update the stock of one product
4. Delete products with price < 100
5. Read collection into Pandas
6. Save DataFrame into CSV file

Exercise 3: Pandas + SQL

1. Create DataFrame of students
2. Write it into SQLite table
3. Read it back
4. Display highest marks

Exercise 4: Pandas + MongoDB

1. Create DataFrame of 10 employees
2. Insert into MongoDB
3. Fetch from MongoDB
4. Show employees with salary > 30,000
