DBMS LAB

Course Code: B24DS02PC
BSc Data Science and Analytics
Practical Core Course
Self Learning Material

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Essential Excel Skills for Data Analysis

Course Code: B24DS02PC

Semester - II

Practical Core Course

Undergraduate Programme

BSc Data Science and Data Analytics

Self Learning Material

SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

DBMS LAB

Course Code: B24DS02PC

Semester- II

Practical Core Course

BSc Data Science and Data Analytics

Scan this QR Code for reading the SLM on a digital device.

Edition
May 2025

Copyright
© Sreenarayanaguru Open University

ISBN 978-81-989642-4-3

9 788198 964243

MESSAGE FROM VICE CHANCELLOR

Dear learner,

I extend my heartfelt greetings and profound enthusiasm as I warmly welcome you to Sreenarayanaguru Open University. Established in September 2020 as a state-led endeavour to promote higher education through open and distance learning modes, our institution was shaped by the guiding principle that access and quality are the cornerstones of equity. We have firmly resolved to uphold the highest standards of education, setting the benchmark and charting the course.

The courses offered by the Sreenarayanaguru Open University aim to strike a quality balance, ensuring students are equipped for both personal growth and professional excellence. The University embraces the widely acclaimed "blended format," a practical framework that harmoniously integrates Self-Learning Materials, Classroom Counseling, and Virtual modes, fostering a dynamic and enriching experience for both learners and instructors.

The University is committed to providing an engaging and dynamic educational environment that encourages active learning. The Study and Learning Material (SLM) is specifically designed to offer you a comprehensive and integrated learning experience, fostering a strong interest in exploring advancements in information technology (IT). The curriculum has been carefully structured to ensure a logical progression of topics, allowing you to develop a clear understanding of the evolution of the discipline. It is thoughtfully curated to equip you with the knowledge and skills to navigate current trends in IT, while fostering critical thinking and analytical capabilities.The Self-Learning Material has been meticulously crafted, incorporating relevant examples to facilitate better comprehension.

Rest assured, the university's student support services will be at your disposal throughout your academic journey, readily available to address any concerns or grievances you may encounter. We encourage you to reach out to us freely regarding any matter about your academic programme. It is our sincere wish that you achieve the utmost success.

[illegible]

Regards,
Dr. Jagathy Raj V. P. 01-05-2025

Contents

```
#include "KMotionDef.h"

int main()
{
    ch0->Amp = 250;
    ch0->output_mode=MICROSTEP_MODE;
    ch0->Vel=70.0f;
    [illegible]
    [illegible]
    [illegible]
    EnableAxisDest(0,0);

    ch1->Amp = 250;
    ch1->output_mode=MICROSTEP_MODE;
    ch1->Vel=70.0f;
    ch1->Accel=500.0f;
    ch1->Jerk =2000f;
    ch1->Lead=0.0f;
    EnableAxisDest(1,0);

    DefineCoordSystem(0,1,-1,-1);

    return 0;
}
```

CYCLE I

RDBMS I

EXPERIMENT - 1
MySQL

Objectives

- To successfully install and configure MySQL on your system.
- To be able to connect to MySQL using the command-line client.
- To create a new database and user with appropriate privileges.
- To familiarize students with the fundamental configuration of a MySQL server, including installation, basic security setup, and configuration file management.

Theory

PostgreSQL and MySQL are widely used relational database management systems. MySQL is a popular open-source relational database management system used by many organizations and individuals. This lab manual will guide you through the process of installing and configuring MySQL, a popular open-source relational database management system (RDBMS).

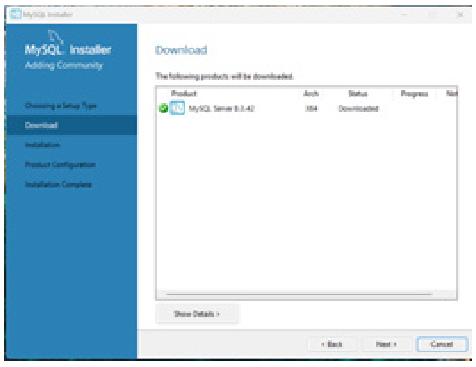
1.1.1 Installing MySQL

Step 1: Download the MySQL Installer

1. Go to the official MySQL website: https://dev.mysql.com/downloads/
2. Select your operating system and click on the "No thanks, just start my download" button.
3. Save the installer file to your computer.

Step 2: Run the MySQL Installer

1. Double-click the downloaded installer file. The following screen will appear:



Download

The following products will be downloaded.

Product	Arch	Status	Progress	Not
MySQL Server 8.0.42	X64	Downloaded		

Fig. 1.1.1 Installer window

2. Follow the on-screen instructions, accepting the default settings unless you have specific requirements.

3. When prompted, set the root password for MySQL. This is the password you'll use to access the MySQL server.

4. Complete the installation process.

Step 3: Choose Setup Type

Select a setup type:

1. *Developer Default* – includes MySQL Server, Workbench, Shell, and more.

2. *Server only* – just the MySQL Server.

3. *Custom* – lets you pick exactly what to install.

Step 4:Check Requirements

The installer will check for required software. Click **Execute** to install any missing components.

Step 5: Download & Install

Click **Execute** again to download and install the selected MySQL products.

Step 6: Configure MySQL Server

- Choose **Standalone MySQL Server.**
- Select **Development Computer** as the configuration type.
- Set the **port** (default is 3306).
- Choose the Authentication **Method**
- Set a **root password** and optionally add user accounts.

Step7:Apply Configuration

Click **Execute** to apply the configuration settings.

Step 8: Finish Setup

Once configuration is complete, click **Finish.**

1.1. 2 Basic MySQL Configuration Steps(Windows/ Linux)

1. **Launch MySQL Installer or Configuration Tool**
   - ♦ If you're using MySQL Installer, it will guide you through configuration after installation.
   - ♦ On Linux, you can configure manually via terminal or by editing config files.
2. **Server Configuration Type**
   - ♦ Choose based on your system:
   - ♦ Development Computer – for local testing.
   - ♦ Server Machine – for production environments.
3. **Port Settings**
   - ♦ Default port is 3306. You can change it if needed, but make sure it's open in your firewall.
4. **Authentication Method**
   - ♦ Choose Strong Password Encryption for better security.
   - ♦ Set a root password and optionally create additional user accounts.
5. **Windows Service (Windows only)**
   - ♦ Configure MySQL to run as a Windows Service so it starts automatically.
   - ♦ You can name the service (e.g., MySQL80).
6. **Apply Configuration**
   - ♦ Click Execute to apply all settings.
   - ♦ MySQL will initialize the data directory and start the server.

1.1.3 Optimizing MySQL Performance

1. **Enable Query Caching:**

```
SET GLOBAL query_cache_size = 64M;
```

2. **Index Optimization:**

Use EXPLAIN to analyze queries: EXPLAIN

```
SELECT * FROM users WHERE email = 'test@example.com';
```

Add indexes where necessary:

```
CREATE INDEX idx_email ON users(email);
```

3. **Optimize Tables:**

```
OPTIMIZE TABLE users;
```

4. **Monitor Performance:**

```
SHOW STATUS LIKE 'Threads_connected';
SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
```

Experiment 2
Basic Operations

Objectives

- The purpose of this exercise is to familiarize you with the basic tasks required to manage and work with relational databases.
- Creating databases and tables, adding, updating, and deleting data, as well as carrying out simple searches and transactions, will all be covered.
- Foundation to the database interactions as CRUD operations (Create, Read, Update, and Delete).
- Through practical activities, you will learn how to create effective databases and confidently carry out basic data operations.

Theory

1.2.1 Fundamental SQL operations

Understanding the fundamental SQL operations, **creating databases and tables, inserting data, updating data**, and **deleting data** is essential for effective database management. Below are the syntaxes, examples, and expected outputs for these operations.

1.2.1.1 Creating Databases and Tables

Creating databases and tables is fundamental in relational database management systems (RDBMS), as they provide the structured framework for storing and organizing data.

1. **Creating a Database:** To establish a new database, the `CREATE DATABASE` statement is utilized. This command initializes a container to hold related tables and other database objects.

   Syntax:

   ```
   CREATE DATABASE database_name;
   ```

   Example:

   ```
   CREATE DATABASE CompanyDB;
   ```

   Output:

   A new database named `CompanyDB` is created.

2. **Creating a Table:** Once the database is created and selected, tables can be defined using the `CREATE TABLE` statement. Each table consists of columns, with each column assigned a specific data type and optional constraints to enforce data integrity.

   Syntax:

   ```
   CREATE TABLE table_name (
   column1 datatype constraints,
   column2 datatype constraints, –.);
   ```

   Example:

   ```
   CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   Department VARCHAR(50) );
   ```

   Output:

   A table named `Employees` is created with columns for `EmployeeID`, `FirstName`, `LastName`, and `Department`.

1.2.1.2 Inserting Data

Inserting data into a table is a fundamental operation in SQL, enabling the addition of new records to your database. The `INSERT INTO` statement facilitates this process by specifying the target table and the values to be inserted.

Syntax:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Example:

```
INSERT INTO Employees (EmployeeID, FirstName, LastName,
Department) VALUES (1, 'John', 'Doe', 'Marketing');
```

Output:

A new record is inserted into the `INSERT` table.

1.2.1.3 Updating Data

Updating data in a relational database is a critical operation that allows you to modify existing records to ensure your data remains current and accurate. In SQL, this is accomplished using the UPDATE statement, which enables targeted changes to specific rows within a table.

Syntax:

```
UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;
```

Example:

```
UPDATE Employees

SET Department = 'Sales'

WHERE EmployeeID = 1;
```

Output:

The `Department` of the employee with `EmployeeID` 1 is updated to 'Sales'.

1.2.1.4 Deleting Data

Deleting data from a relational database is a fundamental operation that allows for the removal of specific records or entire datasets from a table. This is accomplished using the `DELETE` statement in SQL.

Syntax:

```
DELETE FROM table_name

WHERE condition;
```

Example:

```
DELETE FROM Employees

WHERE EmployeeID = 1;
```

Output:

The record with `EmployeeID` 1 is deleted from the `Employees` table.

1.2.1.5 Drop table

The `DROP TABLE` statement in SQL is used to permanently remove an existing table from a database, including all of its data, structure, indexes, triggers, constraints, and permissions. This action is irreversible, so it should be executed with caution.

Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Employees;
```

Output:

The table named `Employees` removed from your database.

Exercise Questions

Exercise 1: Create a new database named `SchoolDB` and within it, create two tables: `Students` and `Courses`.

Instructions:

1. **Create the Database:** Write an SQL statement to create a database named `SchoolDB`.

   Query:

   ```
   CREATE DATABASE SchoolDB;
   ```

2. **Create the Students Table:** Within SchoolDB, create a table named Students with the following columns.

   - `StudentID`: An integer serving as the primary key.
   - `FirstName`: A variable character field with a maximum length of 50 characters.
   - `LastName`: A variable character field with a maximum length of 50 characters.
   - `EnrollmentDate`: A date field.

Query:

```
CREATE TABLE Students (
  StudentID INT PRIMARY KEY,
  FirstName VARCHAR(50),
  LastName VARCHAR(50),
  EnrollmentDate DATE);
```

3. **Create the Courses Table:** Within SchoolDB, create a table named Courses with the following columns.

   - `CourseID`: An integer serving as the primary key.
   - `CourseName`: A variable character field with a maximum length of 100 characters.
   - `Credits`: An integer representing the number of credits for the course.

Query:

```
CREATE TABLE Courses (
CourseID INT PRIMARY KEY,
CourseName VARCHAR(100),
Credits INT);
```

Expected Output:

- The SchoolDB database is created successfully.
- The Students and Courses tables are created within SchoolDB with the specified columns and data types.

Exercise 2: Inserting sample data into the Students and Courses tables. Instructions:

1. **Insert Data into `Students`:**

   - `StudentID`: 1, `FirstName`: 'John', `LastName`: 'Doe', `EnrollmentDate`: '2023-09-01'
   - `StudentID`: 2, `FirstName`: 'Jane', `LastName`: 'Smith', `EnrollmentDate`: '2023-09-01'

- `StudentID`: 3, `FirstName`: 'Emily', `LastName`: 'Johnson', `EnrollmentDate`: '2023-09-01'

Query:

```
INSERT INTO Students (StudentID, FirstName, LastName, EnrollmentDate)

VALUES

  (1, 'John', 'Doe', '2023-09-01'),

  (2, 'Jane', 'Smith', '2023-09-01'),

  (3, 'Emily', 'Johnson', '2023-09-01');
```

4. **Insert Data into `Courses`:**

   - `CourseID`: 101, `CourseName`: 'Mathematics', `Credits`: 3
   - `CourseID`: 102, `CourseName`: 'English Literature', `Credits`: 4
   - `CourseID`: 103, `CourseName`: 'Biology', `Credits`: 3

Query:

```
INSERT INTO Courses (CourseID, CourseName, Credits)

VALUES

  (101, 'Mathematics', 3),

  (102, 'English Literature', 4),

  (103, 'Biology', 3);
```

Expected Output:

- Three records are inserted into the `Students` table.
- Three records are inserted into the `Courses` table.

1.2.2 Performing basic queries

Performing basic SQL queries is fundamental to interacting with relational databases. These queries allow you to retrieve, filter, and manipulate data effectively. Below are some essential SQL queries with their syntax, examples, and expected outputs.

1. **Retrieving All Columns from a Table**

   Syntax:

```
SELECT * FROM table_name;
```

Example:

```
SELECT * FROM Employees;
```

Output:

This query retrieves all columns and rows from the `Employees` table.

2. **Retrieving Specific Columns from a Table**

Syntax:

```
SELECT column1, column2 FROM table_name;
```

Example:

```
SELECT FirstName, LastName FROM Employees;
```

Output:

This query retrieves only the `FirstName` and `LastName` columns from all rows in the `Employees` table.

3. **Filtering Data with a WHERE Clause**

Syntax:

```
SELECT column1, column2 FROM table_name WHERE condition;
```

Example:

```
SELECT * FROM Employees WHERE Department = 'Sales';
```

Output:

This query retrieves all columns for employees who work in the 'Sales' department.

4. **Sorting Data with ORDER BY**

Syntax:

```
SELECT column1, column2 FROM table_name ORDER BY
column1 [ASC|DESC];
```

Example:

```
SELECT * FROM Employees ORDER BY LastName ASC;
```

Output:

This query retrieves all employee records sorted by `LastName` in ascending order.

5. **Using Aggregate Functions**

Syntax:

```
SELECT AGGREGATE_FUNCTION(column) FROM table_name;
```

Example:

```
SELECT COUNT(*) FROM Employees;
```

Output:

This query returns the total number of employees in the `Employees` table.

6. **Grouping Data with GROUP BY**

Syntax:

```
SELECT column1, AGGREGATE_FUNCTION(column2) FROM
table_name GROUP BY column1;
```

Example:

```
SELECT Department, COUNT(*) FROM Employees GROUP BY
Department;
```

Output:

This query retrieves the number of employees in each department.

7. **Filtering Grouped Data with HAVING**

Syntax:

```
SELECT column1, AGGREGATE_FUNCTION(column2) FROM
table_name GROUP BY column1 HAVING condition;
```

Example:

```
SELECT Department, COUNT(*) FROM Employees GROUP BY
Department HAVING COUNT(*) > 10;
```

Output:

This query retrieves departments with more than 10 employees.

8. **Joining Tables with INNER JOIN**

Syntax:

```
SELECT table1.column1, table2.column2 FROM table1 INNER
JOIN table2 ON table1.common_column = table2.common_column;
```

Example:

```
SELECT Employees.FirstName, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

Output:

This query retrieves the first names of employees along with their respective department names by joining the `Employees` and `Departments` tables on the `DepartmentID` column.

Exercise Questions

Exercise 1: Write an SQL query to retrieve the `CustomerID`, `CustomerName`, and `ContactName` of all customers located in the city of 'Berlin'.

Instructions:

1. **Assume the following table structure:**

- Customers:
    - `CustomerID`: Integer, primary key
    - `CustomerName`: VARCHAR(100)
    - `ContactName`: VARCHAR(100)
    - `City`: VARCHAR(100)
    - `Country`: VARCHAR(100)

Query:

```
SELECT CustomerID, CustomerName, ContactName
FROM Customers
WHERE City = 'Berlin';
```

Exercise 2: Write an SQL query to list the `OrderID`, `CustomerID`, and the total amount (`TotalAmount`) for each order.

Instructions:

1. **Assume the following table structure:**

- **Orders:**
    - `OrderID`: Integer, primary key

- `CustomerID`: Integer, foreign key
- `OrderDate`: DATE

◆ OrderDetails:

- `OrderDetailID`: Integer, primary key
- `OrderID`: Integer, foreign key
- `ProductID`: Integer
- `Quantity`: Integer
- `Price`: DECIMAL(10, 2)

Query:

```
SELECT O.OrderID, O.CustomerID, SUM(OD.Quantity * OD.Price) AS TotalAmount

FROM Orders O

JOIN OrderDetails OD ON O.OrderID = OD.OrderID

GROUP BY O.OrderID, O.CustomerID;
```

1.2.3 Transactions

In SQL Server, transactions are fundamental to ensuring data integrity and consistency. They allow multiple operations to be executed as a single unit, adhering to the ACID properties (Atomicity, Consistency, Isolation, Durability). Below are the syntaxes, examples, and expected outcomes for transaction operations.

1. **Beginning a Transaction**

   Syntax:

   ```
   BEGIN TRANSACTION [transaction_name];
   ```

   Example:

   ```
   BEGIN TRANSACTION TransferFunds;
   ```

   Output:

   Initiates a new transaction named 'TransferFunds'.

2. **Committing a Transaction**

   Syntax:

   ```
   COMMIT TRANSACTION [transaction_name];
   ```

Example:

```
COMMIT TRANSACTION TransferFunds;
```

Output:

All operations within the 'TransferFunds' transaction are permanently applied to the database.

3. **Rolling Back a Transaction**

Syntax:

```
ROLLBACK TRANSACTION [transaction_name];
```

Example:

```
ROLLBACK TRANSACTION TransferFunds;
```

Output:

All operations within the 'TransferFunds' transaction are undone, reverting the database to its previous state.

4. **Using Savepoints within a Transaction**

Syntax:

```
SAVE TRANSACTION savepoint_name;
```

Example:

```
BEGIN TRANSACTION;

-- Operations

SAVE TRANSACTION SavePoint1;

-- More operations

ROLLBACK TRANSACTION SavePoint1;

COMMIT TRANSACTION;
```

Output:

The transaction begins, sets a savepoint named 'SavePoint1', and if necessary, rollback to 'SavePoint1' without affecting prior operations.

Exercise Questions

Exercise 1: Write an SQL transaction to transfer funds between two bank accounts, ensuring the operation is atomic and maintains data integrity.

Instructions:

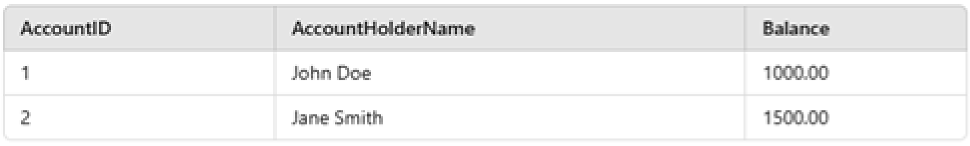
1. **Assume the following table structure:**
    - Accounts:
        - `AccountID`: Integer, primary key
        - `AccountHolderName`: VARCHAR(100)
        - `Balance`: DECIMAL(10, 2)
2. **Write the SQL transaction:**

```
-- Start the transaction

BEGIN TRANSACTION;

-- Variables for the transfer

DECLARE @FromAccountID INT = 1;

DECLARE @ToAccountID INT = 2;

DECLARE @TransferAmount DECIMAL(10, 2) = 500.00;

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - @TransferAmount

WHERE AccountID = @FromAccountID;

-- Add the amount to the destination account

UPDATE Accounts

SET Balance = Balance + @TransferAmount

WHERE AccountID = @ToAccountID;

-- Check for any errors

IF @@ERROR <> 0

BEGIN
```

```
-- Rollback transaction in case of error
ROLLBACK TRANSACTION;
END
ELSE
BEGIN
 -- Commit the transaction if no errors
COMMIT TRANSACTION;
END;
```

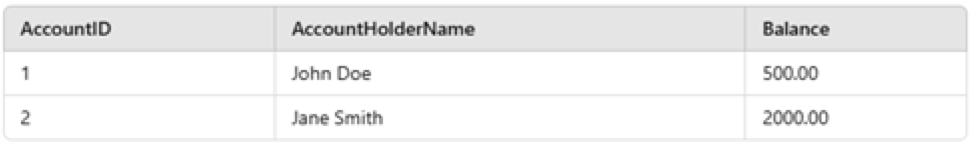
Expected Output:

- Before Transaction:



AccountID	AccountHolderName	Balance
1	John Doe	1000.00
2	Jane Smith	1500.00

- After Transaction:



AccountID	AccountHolderName	Balance
1	John Doe	500.00
2	Jane Smith	2000.00

Experiment Questions

1. Create a simple student database, perform insertion, update, and deletion operations.
2. Perform a simple banking transaction where money is transferred between two accounts.

EXPERIMENT - 3
Using tools for database management.

Objectives

- To install and configure MySQL Workbench.
- To create and manage databases using MySQL Workbench.
- To design and implement database schemas
- To perform Create, Read, Update and Delete operations
- To optimize database performance

Theory

1.3.1 MySQL Workbench

MySQL Workbench Is a powerful tool for database management, development, and administration designed for use with MySQL databases. Through this lab manual, students will gain hands-on experience in creating and managing databases using MySQL Workbench.

Getting Started with MySQL Workbench

1. **Installation:** Download and install MySQL Workbench from the official MySQL website.
2. **Connect to MySQL Server:** Launch MySQL Workbench, create a new connection, and connect to your MySQL server instance.
3. **Explore the Dashboard:** Familiarize yourself with the dashboard that gives you an overview of your databases, connections, and server status.
4. **Start using features:** Use the Schema Navigator to explore your databases, create new databases, or execute queries in the SQL editor.

1.3.2 Executing Basic Tasks in MySQL Workbench

Step 1: Connecting to a MySQL Server

1. Open MySQL Workbench.
2. Click on the **Server** tab.
3. Click on the + button to create a new connection.

4. Enter the following information:
   - **Hostname**: The hostname or IP address of your MySQL server.
   - **Username**: The username to use to connect to the MySQL server.
   - **Password**: The password for the username.
5. Click on the **Test Connection** button to test the connection.
6. If the connection is successful, click on the **OK** button.

Step 2: Creating a Database

1. Right-click on the **Databases** tab and select **Create Schema.**
2. Enter the name of the database you want to create.
3. Click on the **Apply** button.

Step 3: Creating a Table

1. Right-click on the database you created and select **Create Table**.
2. Enter the name of the table you want to create.
3. Add the columns for the table.
4. Click on the **Apply** button.

Step 4: Populating a Table

1. Right-click on the table you created and select **Insert Rows.**
2. Enter the data for the table.
3. Click on the **Apply** button.

Step 5: Writing SQL Queries

1. Click on the **SQL Development** tab.
2. Enter the SQL query you want to execute.
3. Click on the **Execute** button.

Step 6: Visualizing Data

1. Click on the **EER Diagram** tab.
2. Select the tables you want to visualize.
3. Click on the **Create Diagram** button.

1.3.3 Quick MySQL Workbench Operations

Step 1: Connecting to a MySQL Server

1. Open MySQL Workbench.
2. Server tab.
3. "+" (New Connection).
4. Hostname:`localhost`, Username: root, Password: `your_password`.
5. Test Connection.
6. OK.

Step 2: Creating a Database

1. Right-click "Schemas".
2. Create Schema.
3. Name: `test_db`.
4. Apply.

Step 3: Creating a Table

1. Right-click `test_db`.
2. Create a Table.
3. Name: `users`, Columns: id INT PRIMARY KEY AUTO_INCREMENT, `name` VARCHAR(255).
4. Apply.

Step 4: Populating a Table

1. Right-click `users`.
2. Select Rows (Insert Rows).
3. Enter data: `name`: "John", "Jane".
4. Apply.

Step 5: Writing SQL Queries

1. SQL Development tab.

2. `SELECT * FROM test_db.users;.`
3. Execute.

Step 6: Visualizing Data

1. EER Diagram tab.
2. Select `users`.
3. Create a Diagram.

1.3.4 Sample Experiment questions

1. Perform the following tasks using MySQL Workbench
   - ♦ Connect to a local MySQL Server using MySQL Workbench.
   - ♦ Create a new database named school_db.
   - ♦ Create a table named students with the following columns:

     student_id (INT, Primary Key, Auto Increment, Not Null)
     first_name (VARCHAR(255), Not Nul

     last_name (VARCHAR(255), Not Null

     age (INT)

   - ♦ Insert at least three records into the students table.
   - ♦ Write and execute a SQL query to display all students whose age is greater than 17.
   - ♦ Generate and view the **EER Diagram** for the school_db schema using the **Reverse Engineer** tool.

Step 1: Connecting to a MySQL Server

1. **Open MySQL Workbench**. (You'll see the Workbench home screen.)
2. **Click on the "Server" tab**. (In the left-hand sidebar.)
3. **Click on the "+" button (Add a new connection)**. (Usually near "MySQL Connections".)
4. **Enter the following information:**
   - **Connection Name:** "Local MySQL Server" (or something descriptive)
   - **Hostname:** "127.0.0.1" (or "localhost" if your server is on the same machine)

- **Port:** "3306" (the default MySQL port)
- **Username:** "root" (or your MySQL username)
- **Password:** (Enter your MySQL root password)

5. **Click on the "Test Connection" button**. (A dialog box will appear.)
6. **If the connection is successful, click "OK"**. (Then click "OK" on the connection dialog to save the connection.)

Step 2: Creating a Database (Schema)

1. **Right-click in the "Navigator" panel (left side) under the "Schemas" tab.** (If you don't see schemas, make sure you are connected to the server)
2. **Select "Create Schema..."** (A new tab or dialog will appear.)
3. **Enter the name of the database you want to create:** "school_db"
4. **Click "Apply"**. (A SQL script will appear; review it, then click "Apply" again to execute.)
5. **Click "Finish."**

Step 3: Creating a Table

1. **Right-click on the "school_db" schema in the "Navigator" panel.**
2. **Select "Create Table..."** (A table editor will appear.)
3. **Enter the name of the table:** "students"
4. **Add the following columns:**
   - `student_id`: INT, PRIMARY KEY, AUTO_INCREMENT, NOT NULL
   - `first_name`: VARCHAR(255), NOT NULL
   - `last_name`: VARCHAR(255), NOT NULL
   - `age`: INT
5. **Click "Apply"**. (Review the SQL script, then click "Apply" again.)
6. **Click "Finish."**

Step 4: Populating a Table

1. **Right-click on the "students" table in the "Navigator" panel.**

2. **Select "Select Rows - Limit 1000"** (or similar, to open the table data editor).

3. **Click on the last row of the result grid, and enter the following data:**

    - `student_id`: (Leave blank; it will auto-increment)
    - `first_name`: "Alice"
    - `last_name`: "Smith"
    - `age`: 18

4. **Repeat step 3 to add more rows (e.g., "Bob Jones", "Charlie Brown")**

5. **Click "Apply Changes" (the icon looks like a document with a check mark) to save the data.**

Step 5: Writing SQL Queries

1. **Click the "SQL Development" tab (the SQL icon, usually a wrench/hammer**). (Or click the SQL icon in the toolbar, to open a new query tab)

2. **Enter the following SQL query:**

    SQL

    ```
    SELECT * FROM school_db.students WHERE age > 17;
    ```

3. **Click the "Execute" button.** (The query results will appear below the editor.)

Step 6: Visualizing Data (EER Diagram)

1. **Click the "Database" tab in the top menu.**
2. **Select "Reverse Engineer..."**
3. **Select your connection and click next.**
4. **Select the "school_db" schema and click next.**
5. **Click next, then execute.**
6. **Click next, then finish.**
7. **The EER Diagram of your table will now be displayed.**

1.3.5 Lab practice experiments

Exercise Question No. 1 Scenario: Library Database

Database Name: `library_db`

Table Name: `books`

Columns:

- `book_id` (INT, PRIMARY KEY, AUTO_INCREMENT)
- `title` (VARCHAR(255))
- `author` (VARCHAR(255))
- `publication_year` (INT)

Questions:

1. **Connecting to a MySQL Server**
2. **Creating a Database** - create a new database named 'library_db' using MySQL Workbench.
3. **Creating a Table**
   - "Using the 'library_db' database, create a table named 'books' with the columns 'book_id', 'title', 'author', and 'publication_year'.
   - Specify 'book_id' as the primary key and set it to auto-increment.
4. **Populating a Table**
   - Insert the following book entries into the 'books' table using MySQL Workbench's 'Insert Rows' functionality. ('The Hobbit', 'J.R.R. Tolkien', 1937) and ('Pride and Prejudice', 'Jane Austen', 1813).
   - After entering the data, save the changes to the table.
5. **Writing SQL Queries**
   - Write and execute a SQL query to retrieve all books published after 1900 from the 'books' table.
   - Write and execute a SQL query to find all books written by 'Jane Austen'.
6. **Visualizing Data**
   - How to visualize the 'books' table and its columns?
   - After selecting the 'books' table, Display or generate the diagram..

Exercise Question No. 2 Scenario: Product Inventory

Database Name: `inventory`

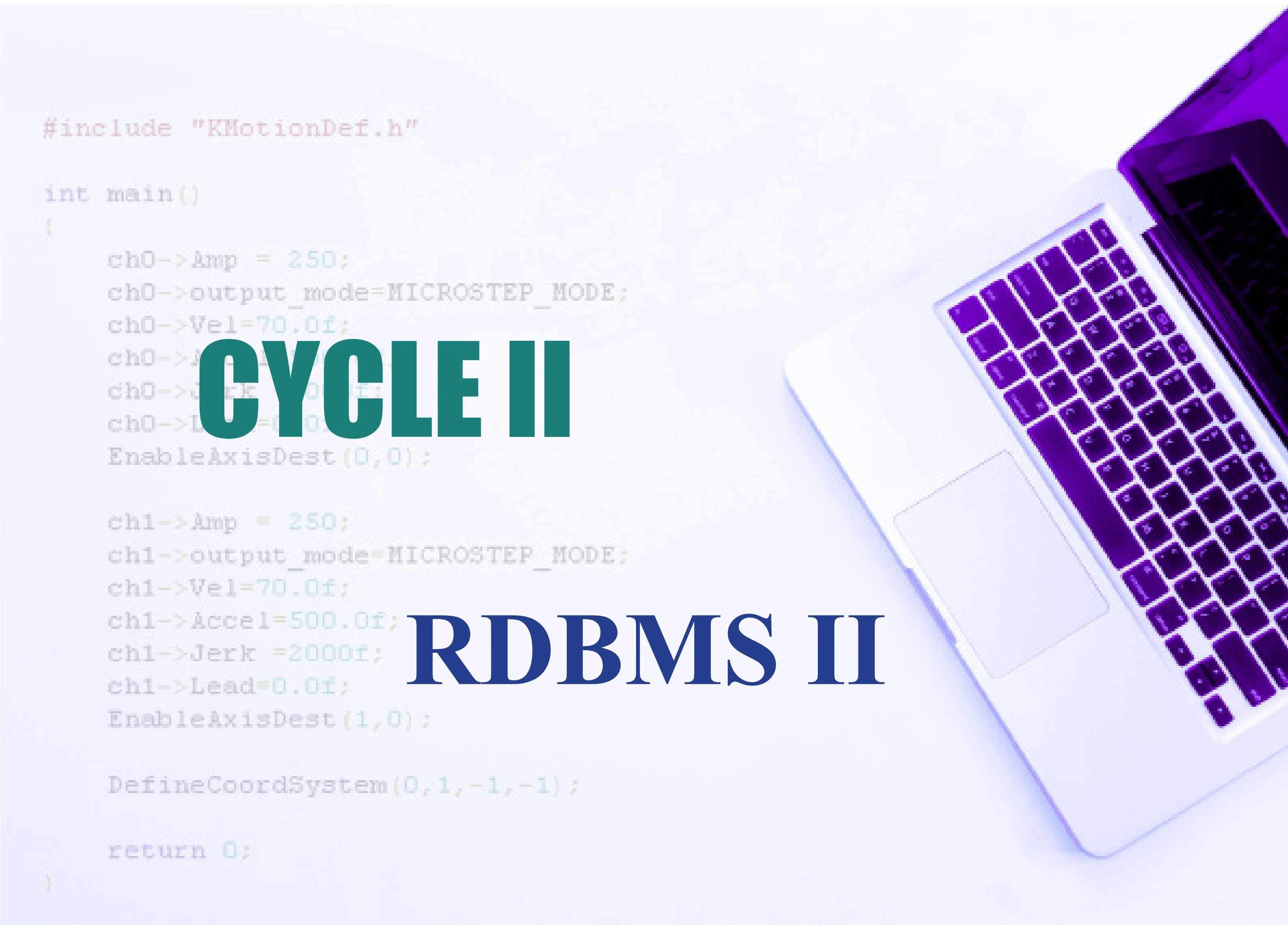
Table Name: `items`

Columns:

- `item_id` (INT, PRIMARY KEY, AUTO_INCREMENT)
- `item_name` (VARCHAR(255))
- `quantity` (INT)
- `price` (DECIMAL(10, 2))

Questions:

1. **Connecting to a MySQL Server**
2. **Creating a Database named inventory,**
3. **Creating a Table** (columns 'item_id', 'item_name', 'quantity', and 'price'). 'item_id' as the primary key and set it to automatically increment.
4. **Populating a Table by inserting data.**
5. **Writing SQL Queries**
    - Write and execute a SQL query to retrieve all items from the 'items' table where the quantity is greater than 20..
    - Write and execute a simple 'SELECT' query to view all data from the 'items' table.
6. **Visualizing Data**



CYCLE II

RDBMS II

Experiment 1 : Database Backups

Topic 1: Performing database backups

Objective

To understand and perform various methods of backing up a MySQL database using tools such as `mysqldump`, phpMyAdmin, MySQL Workbench, and file system snapshots.

1. Introduction

Backing up a MySQL database is a critical task to protect data against accidental loss, system failures, or corruption. This lab will demonstrate multiple methods of creating backups in MySQL. Each method suits different requirements—command-line, GUI-based, or system-level backup.

2. Tools/Software Required

- MySQL Server
- Command Line Interface / Terminal
- phpMyAdmin (Web-based tool)
- MySQL Workbench (Desktop GUI)
- LVM / File System Snapshot Tools (for advanced users)

3. Backup Methods in MySQL

Method 1: Backup Using `mysqldump` (Command-Line Tool)

Description:

`mysqldump` is a command-line utility that exports the database into a `.sql` file containing all the SQL statements to recreate schema and data.

Steps:

1. **Open Command Line/Terminal.**

Run the following command:

```
mysqldump -u root -p my_database > my_database_backup.sql
```

2.
   - Replace `root` with your username.

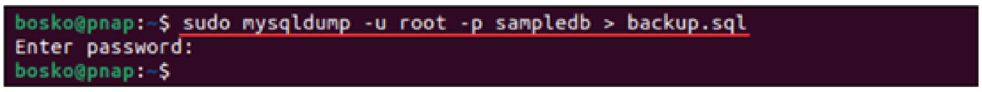
- Replace `my_database` with the name of your database.
- `my_database_backup.sql` is the output backup file.

3. Enter your MySQL password when prompted.
4. A `.sql` file will be generated in the working directory.

Example:

```
mysqldump -u student_user -p college_db > college_db_
backup.sql
```

For example:



```
bosko@pnap:~$ sudo mysqldump -u root -p sampledb > backup.sql
Enter password:
bosko@pnap:~$
```

When prompted, enter your password and press **Enter** to confirm. Wait for the process to complete.

Method 2: Backup Using phpMyAdmin (Graphical Web Interface)

Description:

phpMyAdmin is a browser-based tool that allows you to export database backups with just a few clicks.

Steps:

1. Log in to **phpMyAdmin** through your browser.
2. Select the database (e.g., college_db) from the left panel.
3. Click on the **Export** tab.
4. Choose:
   - Quick Export for simple backup.
   - Custom Export for selecting specific tables or formats.
5. Click **Go** to download the backup .sql file.

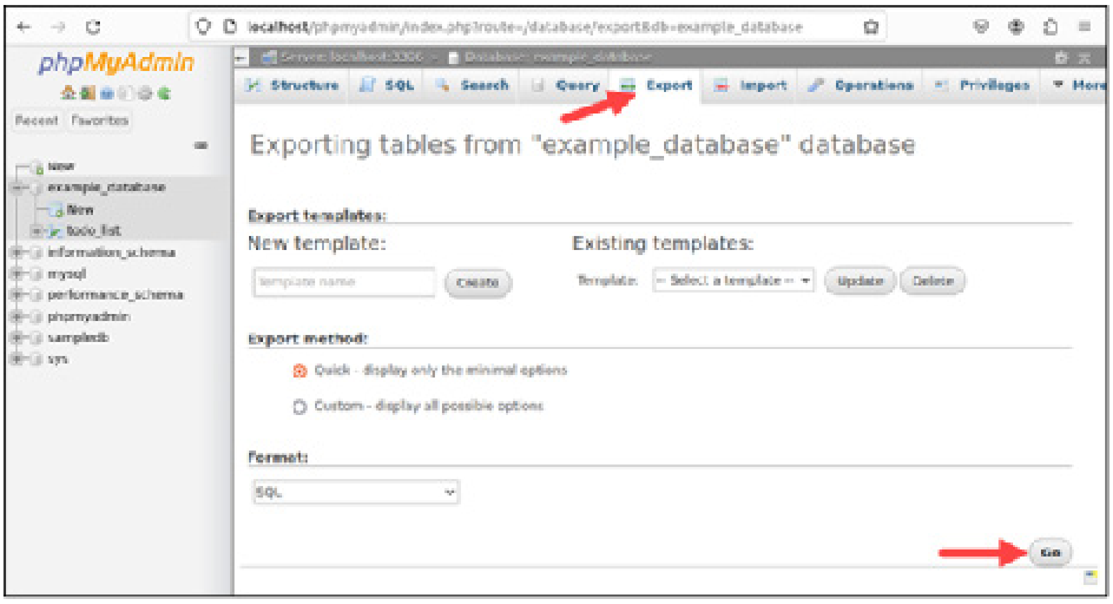
Example:

Export `student_records` database using Quick Export option

See fig. 2.1.1

Result:

`student_records.sql` downloaded.



Exporting tables from "example_database" database

Export templates:

New template: template name

Existing templates: Template: -- Select a template --

Export method:

[✓ Quick - display only the minimal options]

Custom - display all possible options

Format:

SQL

Fig. 2.1.1 Export student records database using quick export option

Method 3: Backup Using MySQL Workbench (Desktop GUI Tool)

Description:

MySQL Workbench offers a GUI-based option for exporting data and schema.

Steps:

1. Open **MySQL Workbench.**
2. Connect to your database server.
3. From the top menu, click: Server → Data Export.
4. Select the database (e.g., college_db) from the list.
5. Choose the tables to export and the desired export format.
6. Click **Start Export** to begin backup.

See fig. 2.1.2 and fig. 2.1.3

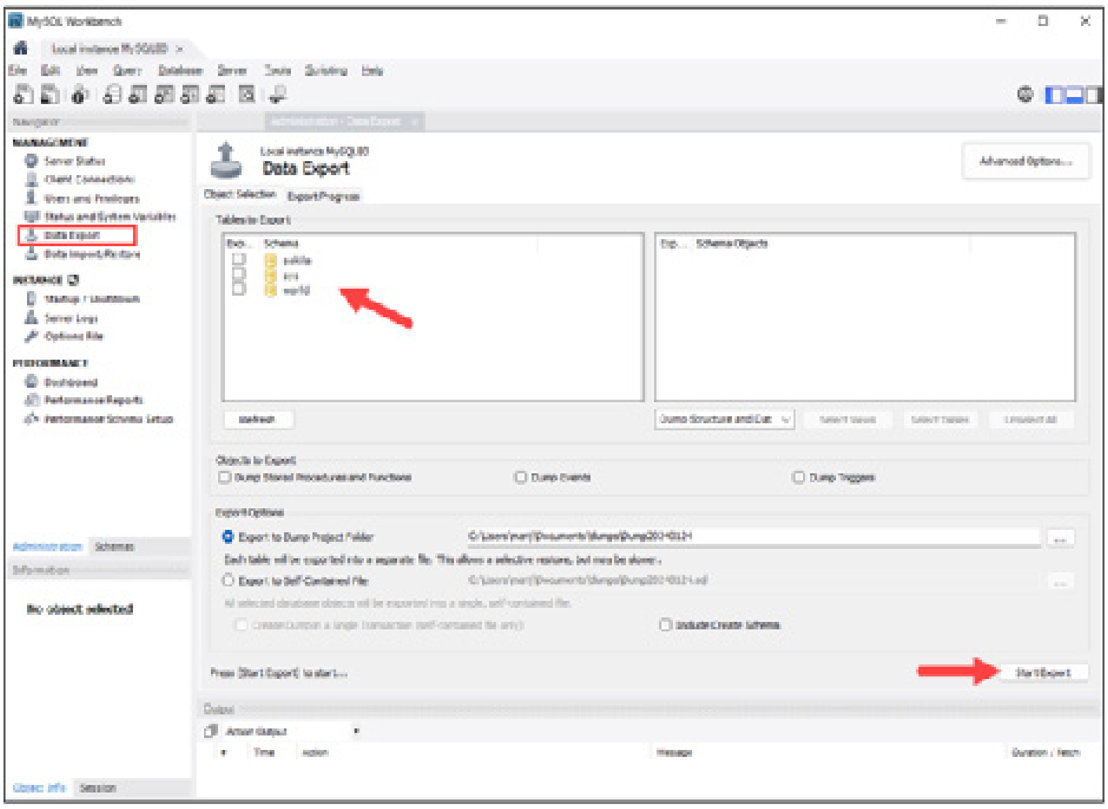


Fig. 2.1.2 MySQL workbench screen 1

The Backup Wizard displays the backup progress and a confirmation when it is completed.

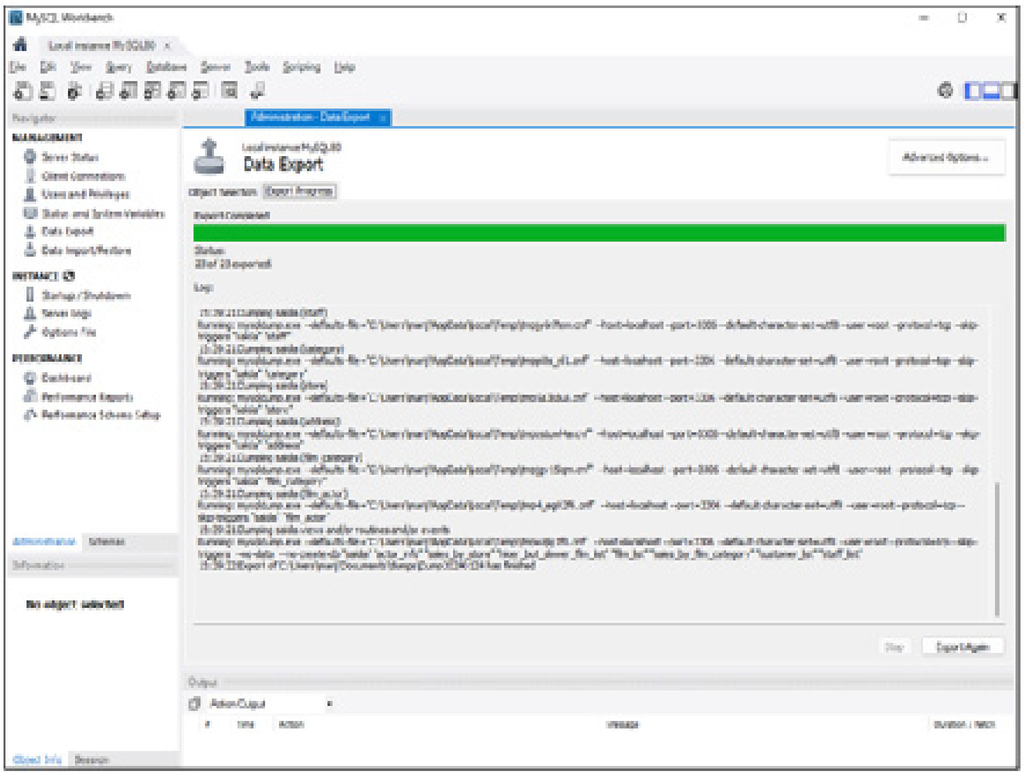


Fig. 2.1.3 MySQL workbench screen 2

Example:

Exporting selected tables like `students`, `courses` from `college_db` will create .sql dump files in the specified folder.

Method 4: Backup Using File System Snapshot (Advanced Method)

Description:

This method captures a physical snapshot of the MySQL data directory using file system tools like **LVM**. It's ideal for backing up large databases.

Steps:

(Optional) **Stop MySQL** service for consistency:

```
sudo systemctl stop mysql
```

1. **Create a snapshot using LVM:**

```
lvcreate --size 1G --name mysql_snap vg0/mysql_lv
```

2. **Restart MySQL service:**

```
sudo systemctl start mysql
```

3. **Copy snapshot files to backup storage.**

Note: Requires system-level access and pre-configured logical volumes.

Table 2.1.1 Comparison Table of Backup Methods

Method	Interface	Use Case	Skill Level	Backup Format
`mysqldump`	CLI	Small/medium logical backup	Intermediate	`.sql` file
phpMyAdmin	GUI	Web-based easy backup	Beginner	`.sql` file
MySQL Workbench	GUI	Desktop-based selective backup	Beginner	`.sql` file
File System Snapshot	System	Large physical backup (live system)	Advanced	Physical files

Regular database backups are essential for data integrity and recovery. The method used depends on the system environment, size of the database, and user expertise. It is recommended to schedule backups periodically and test recovery procedures to ensure data safety.

Topic 2: Restoring Databases from Backups

How to Restore MySQL Database

There are several ways to restore a MySQL database, and the method you choose depends on factors such as the backup format, the database size, and the specific requirements of your setup.

The sections below show different ways of restoring a MySQL database.

Method 1: Restore All Databases in MySQL

The `mysqldump` utility allows you to restore a single database or all databases on the server in bulk. To restore all databases in MySQL, use the `mysql` command-line tool with an SQL dump file that contains the structure and data for all databases.

Follow the steps below:

1. Note the location of your SQL dump file containing the structure and data for all databases.

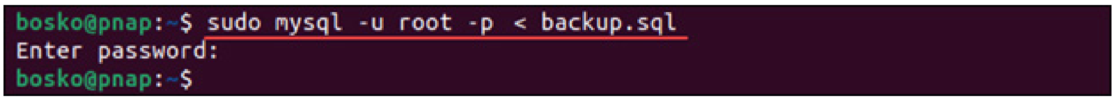
2. Use the `mysql` command to restore the dump file. The syntax is:

```
mysql -u username -p < backup.sql
Copy
```

♦ Replace `username` with your MySQL username.

♦ Replace `backup.sql` with the path to your SQL dump file.

For example:



```
bosko@pnap:~$ sudo mysql -u root -p  < backup.sql
Enter password:
bosko@pnap:~$
```

The command reads the SQL commands from the dump file and executes them to recreate all the databases.

After running the above command, MySQL prompts you for the password. Provide it and press Enter to confirm. It may take a long time to restore the databases if the SQL dump file is large. Also, ensure you have sufficient privileges to create databases and tables.

Method 2: Restore MySQL Database Using phpMyAdmin

phpMyAdmin is a popular web-based administration tool for MySQL. It is a convenient option for smaller databases. However, other methods like command-line tools or MySQL clients are more suitable for larger databases or when you need more control over the restoration process.

Follow the steps below to restore a MySQL database using phpMyAdmin.

1. Clear Old Database Information

It's important to clear old data before restoring a backup. If there is old data, it isn't overwritten when you restore the backup, causing duplicate tables, errors, and conflicts.

1. Open phpMyAdmin, and from the navigation pane on the left, choose the **database** you want to restore.
2. Select the Check all box near the bottom and click the Drop button to clear the old data.

See fig. 2.1.4

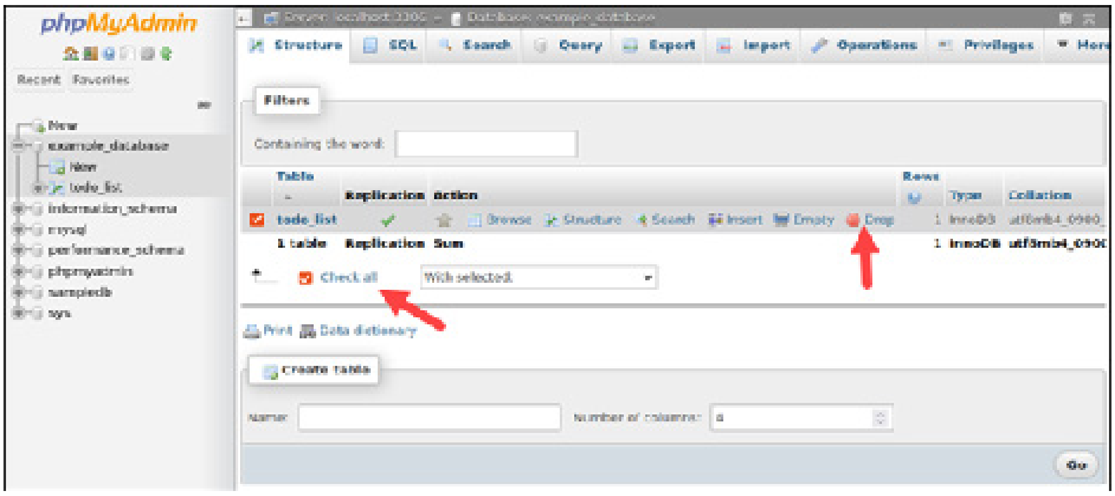


Fig. 2.1.4 Clearing old database information

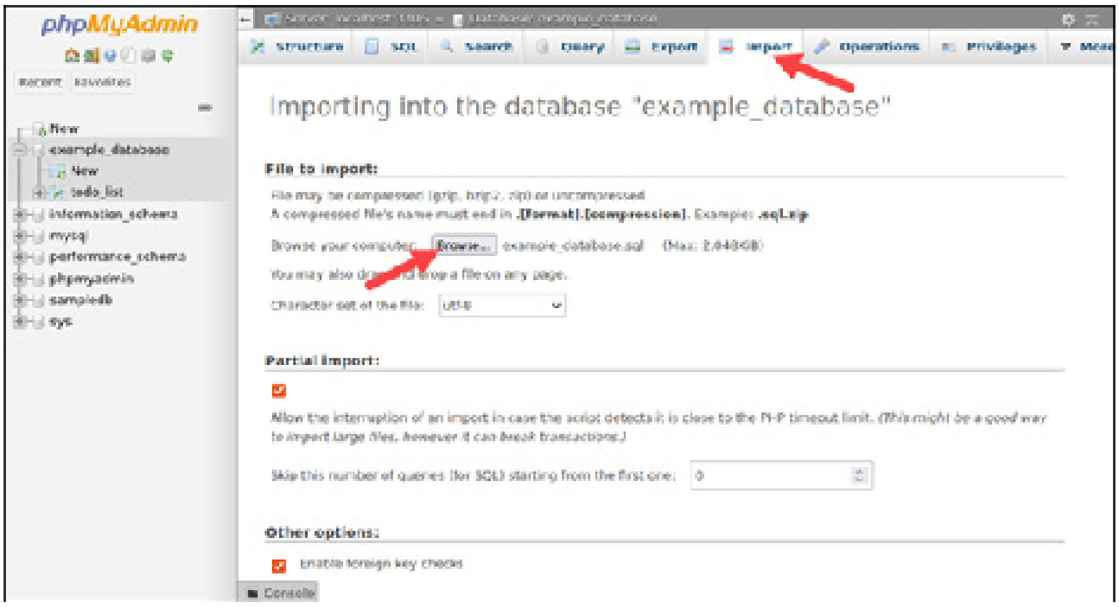
3. The tool prompts if you want to proceed. Click Yes.

The process removes the existing data, clearing the way for restoring the database.

2. Restore Database Information

In phpMyAdmin, use the Import tool to restore a database.

1. In the top menu, click Import.
2. The first section is labeled File to import. A couple of lines down, there's a button labeled Browse.... Click the button and find the backup file.
3. Use the dialog box to find the export file you want to restore. Leave all the options set to default. If you created your backup with different options, you can select them here.



Importing into the database "example_database"

File to import:

File may be compressed (gzip, bzip2, zip) or uncompressed.
A compressed file's name must end in **.[format].[compression]**. Example: **.sql.zip**

Browse your computer: Browse... example_database.sql (Max: 2,048KiB)

You may also drag and drop a file on any page.

Character set of the file: utf-8

Partial import:

☑ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit. *(This might be a good way to import large files, however it can break transactions.)*

Skip this number of queries (for SQL) starting from the first one: 0

Other options:

☑ Enable foreign key checks

Fig. 2.1.5 Importing in to database

4. Click Go to start the process.

See fig. 2.1.5

The program outputs the details and states whether the restore has been completed successfully.

Note: Depending on the server configuration, there are limitations on the file size you can upload through phpMyAdmin. If your SQL dump file is too large, consider using other methods like the command line or breaking the file into smaller parts. Additionally, if the restoration process takes a long time, it could be interrupted because the PHP web server has timeout limits.

Method 3: Restore a Specific Table in MySQL Database

To restore a specific table in MySQL, you can use the `mysql` command-line tool or a MySQL graphical user interface like **MySQL Workbench**. For this section, we will use the command line.

Assuming you have an SQL dump file that contains the specific table you want to restore, use the following syntax:

```
mysql -u username -p dbname < backup.sql
```

- ♦ Replace `username` with your MySQL username.
- ♦ `dbname` is your database name.
- ♦ Replace `backup.sql` with the path to your SQL dump file.

To restore a specific table from the dump file, use the `--tables` option along with the `mysql` command:

```
mysql -u username -p dbname --tables table_name < backup.
sql
```

Replace `table_name` with the name of the specific table you want to restore.

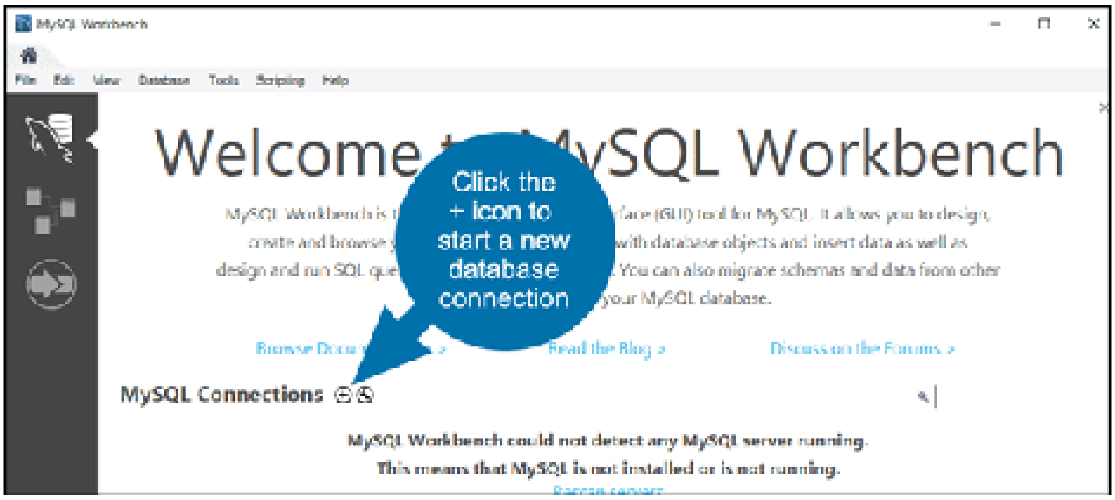
In this tutorial, we will go through the steps to restore a database from a backup. We will also cover the necessary configuration to connect to your database with MySQL Workbench. This will cover everything for you in one spot, in case you've never done .

Configuring MySQL Workbench to Connect to Your Database

Many commercial hosts block outside database connections, so you may have to add your home or office IP address to a remote access list. Check with your host to see what their requirements are. If your website uses cPanel, you can set up a remote connection in Databases > Remote MySQL.

See fig. 2.1.6

Open MySQL Workbench and click the + icon to start a new database connection.


MySQL Workbench
File Edit View Database Tools Scripting Help
Welcome to MySQL Workbench
Click the + icon to start a new database connection
Browse Documentation >
Read the Blog >
Discuss on the Forums >
MySQL Connections

MySQL Workbench could not detect any MySQL server running.
This means that MySQL is not installed or is not running.

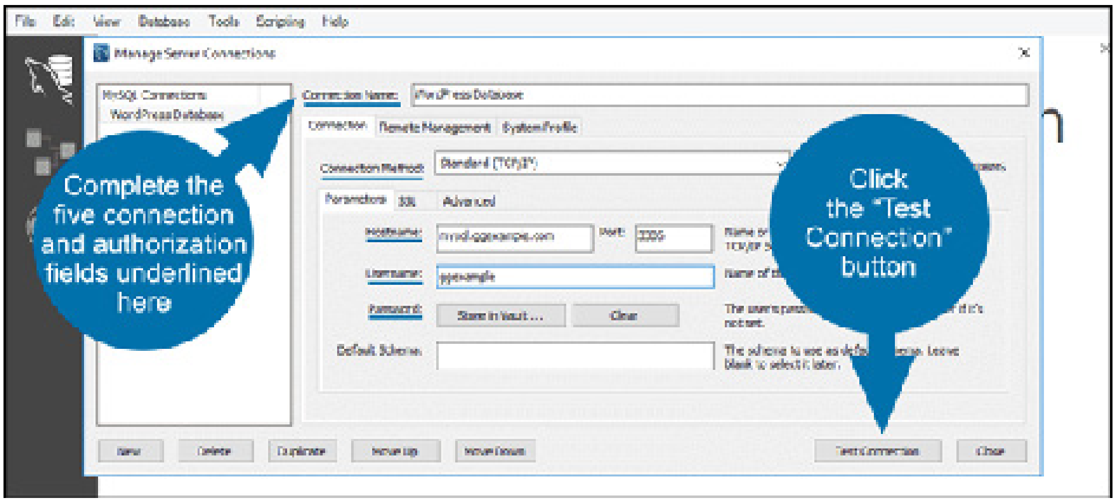
Fig. 2.1.6 Configuring MySQL workbench 1

Complete the five connection and authorization fields underlined below.

- Give the connection a name.
- Choose "Standard (TCP/IP) as the "Connection Method" (SSH connection configuration is available if your host requires it).
- Enter the MySQL server hostname or IP address.
- Enter the MySQL database username.
- Click the "Store in Vault…" button to enter the database password (if you don't want to store the password, skip this field).

Click the "Test Connection" button.

See fig. 2.1.7, 2.1.8, 2.1.9



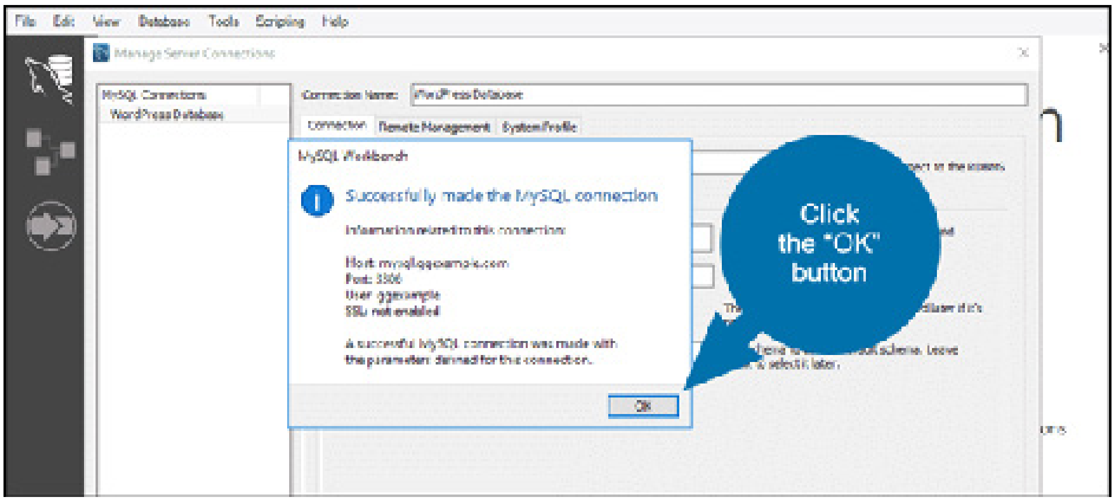
File Edit View Database Tools Scripting Help
Manage Server Connections
MySQL Connections
Complete the five connection and authorization fields underlined here
Click the "Test Connection" button
New
Delete
Duplicate
Move Up
Move Down
Test Connection
Close

Connection Name: WordPress Database
Connection | Remote Management | System Profile
Connection Method: Standard (TCP/IP)
Parameters | SSL | Advanced
Hostname: mysql.ggexample.com   Port: 3306
Username: ggexample
Password: Store in Vault ... Clear
Default Schema:
The schema to use as default schema. Leave blank to select it later.

Fig. 2.1.7 MySQL connection 1

If you get a "Cannot Connect to Database Server" error, check your entries in the connection fields.

If everything is correct, you'll see the successful connection box. Click the "OK" button and move on from there.



File Edit View Database Tools Scripting Help
Manage Server Connections
MySQL Connections
WordPress Database
Click the "OK" button
OK

MySQL Workbench

Successfully made the MySQL connection

Information related to this connection:

Host: mysql.ggexample.com
Port: 3306
User: ggexample
SSL: not enabled

A successful MySQL connection was made with the parameters defined for this connection.

Fig. 2.1.8 MySQL connection 2

Now click the "OK" button in "Manage Server Connections" to close the connection test window.

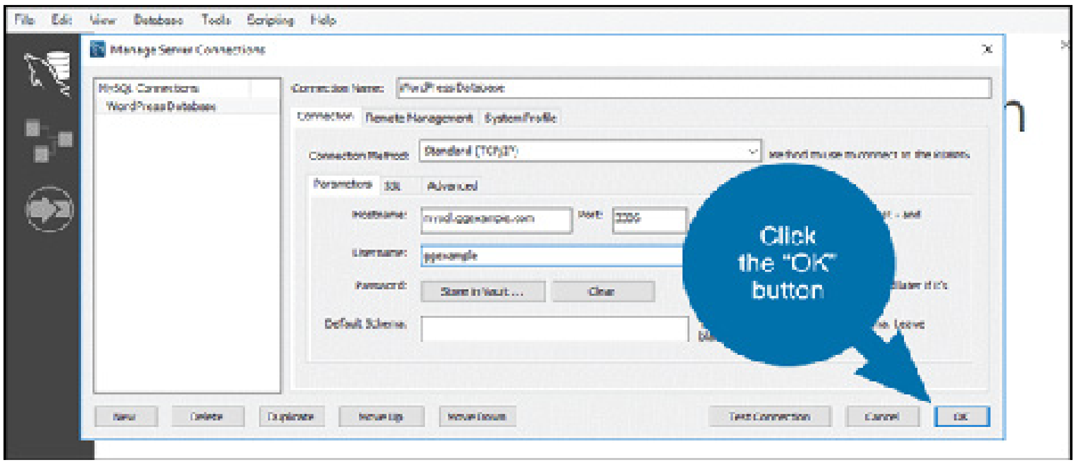


Fig. 2.1.9 MySQL connection 3

Configuring MySQL Workbench to Restore (Import) Your Database

Click the box for the database connection that you just set up.

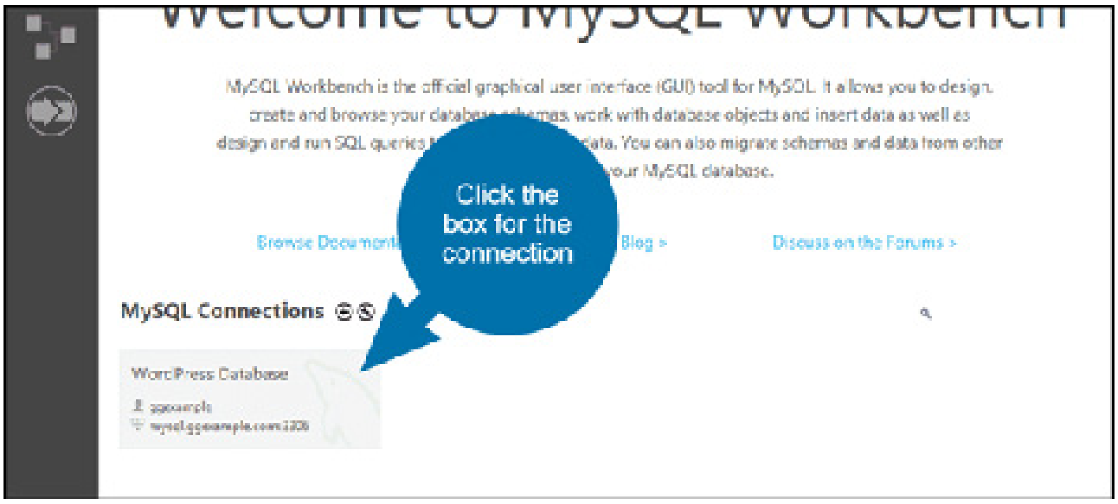


Fig. 2.1.10 MySQL work bench to restore

Click the “Data Import/Restore” link.

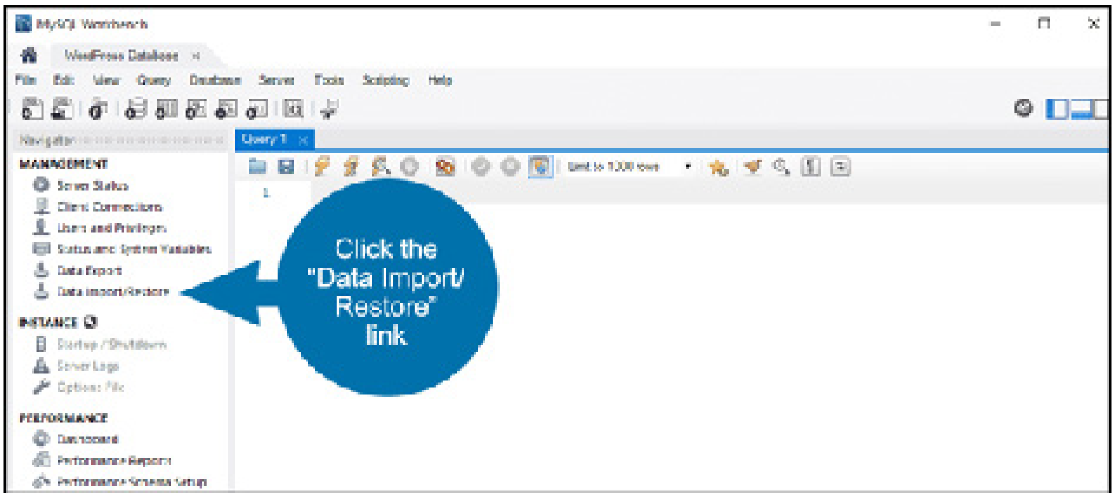


Fig. 2.1.11 MySQL data import/restore

For this tutorial, we're assuming you are restoring a "Self-Contained File" backup. See "Making a MySQL Database Backup With MySQL Workbench," for an explanation of the difference between a self-contained file and a dump project folder.

Select "Import from Self-Contained File," and locate the backup file that will be used for restoration.

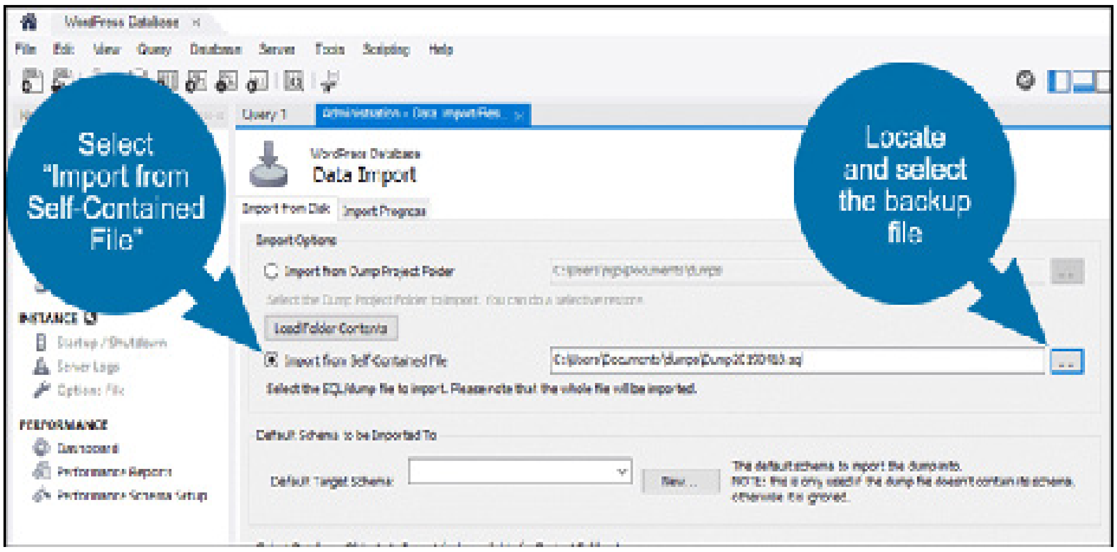

Select "Import from Self-Contained File"
Locate and select the backup file
WordPress Database
Data Import
Import from Disk
Import Progress
Import Options
Import from Dump Project Folder
Load Folder Contents
Import from Self-Contained File
Select the SQL/dump file to import. Please note that the whole file will be imported.
Default Schema to be Imported To
Default Target Schema:
New...
The default schema to import the dump into.
NOTE: this is only used if the dump file doesn't contain its schema, otherwise it is ignored.
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup

Fig. 2.1.12 Default target schema selection

Select the "Default Target Schema" from the drop-down. The drop-down should be pre-populated with the schema name from the backup file.

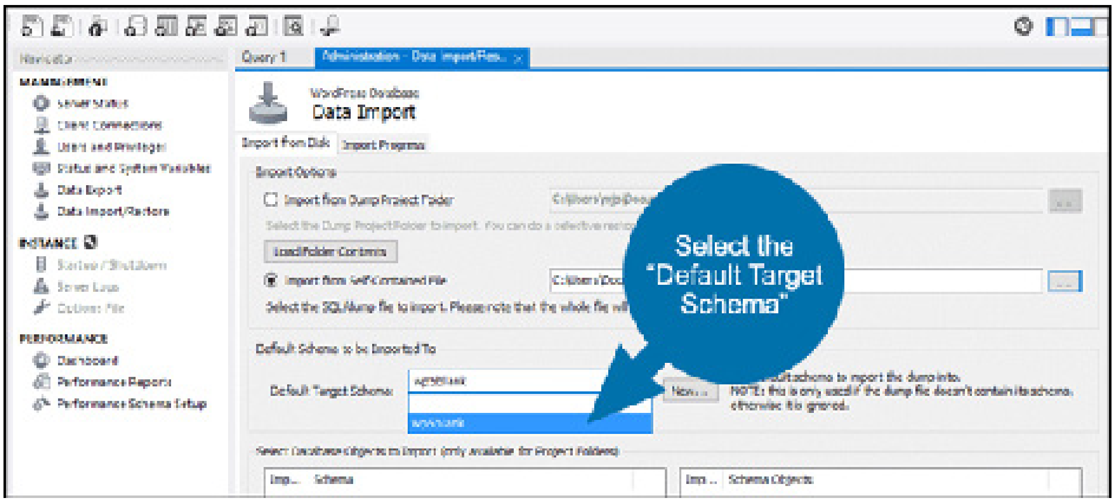


Fig. 2.1.13 Default target schema selection

Since we are restoring the entire database from a self-contained file, "Select Database Objects to Import" is left blank because there is no need to select specific tables.

Make sure "Dump Structure and Data" is selected from the dropdown.

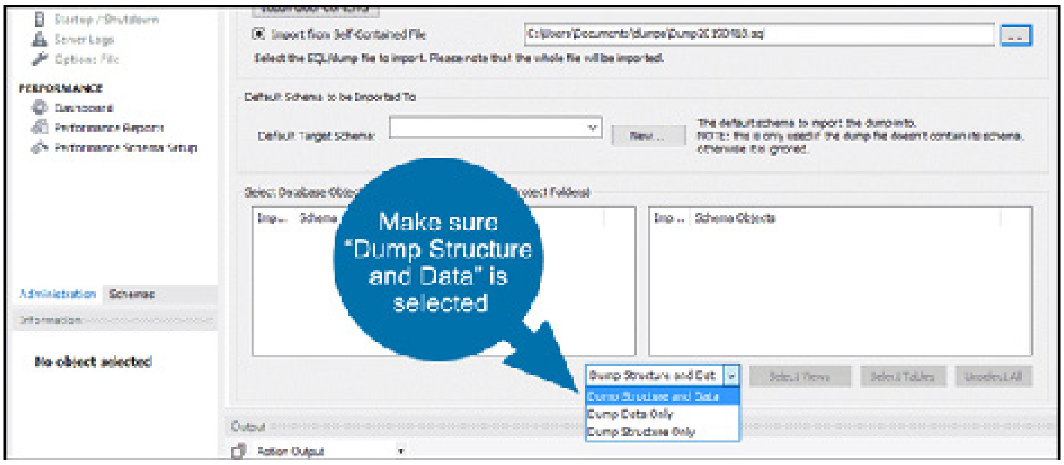


Fig. 2.1.14 Dump structure and data selection

Click the "Import Progress" tab.

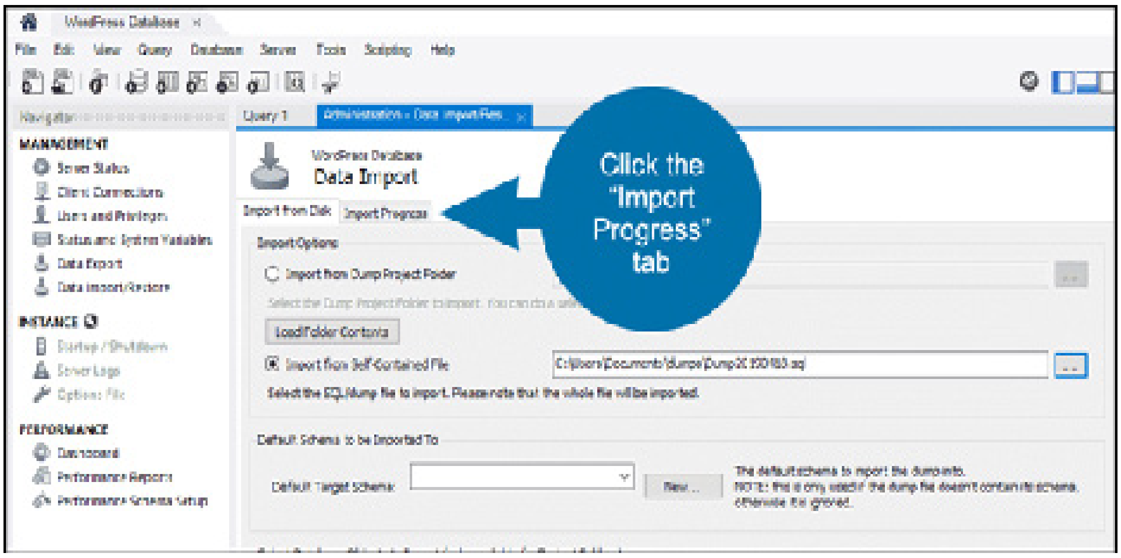


Fig. 2.1.15 Import progress tab selection

Click the "Start Import" button.

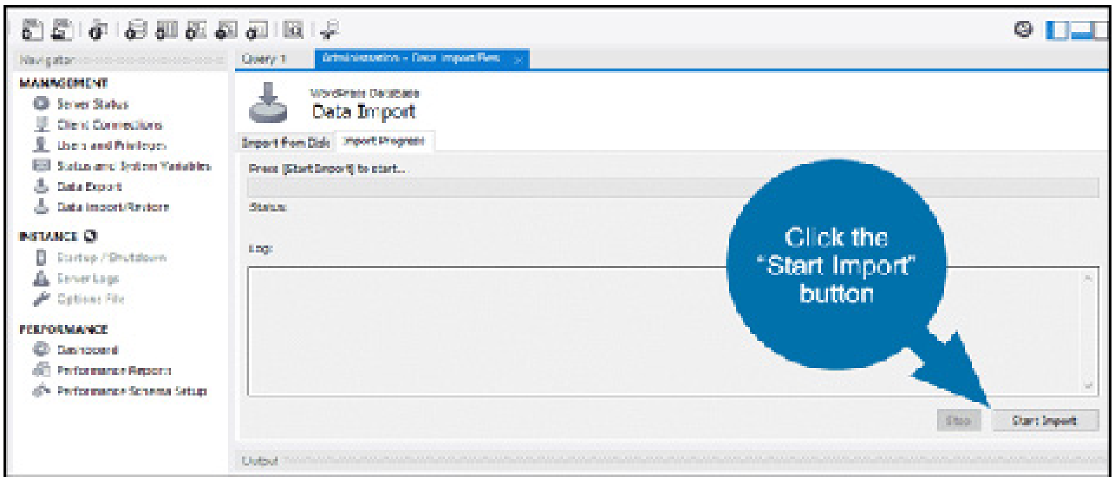
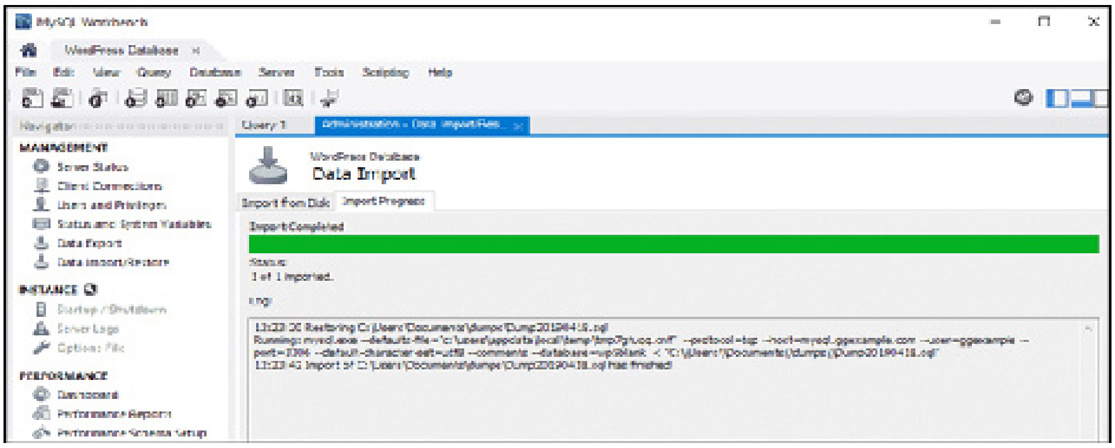


Fig. 2.1.16 Start import selection

When the restoration is complete, you’ll see an “Import Completed” dialog.



Import Completed

Status:
1 of 1 imported.

Log:

```
13:23:30 Restoring C:\Users\Documents\dumps\Dump20190418.sql
Running: mysql.exe --defaults-file="c:\users\appdata\local\temp\tmp7g\ucq.cnf" --protocol=tcp --host=mysql.ggexample.com --user=ggexample --port=3306 --default-character-set=utf8 --comments --database=wp[illegible] < "C:\\Users\\Documents\\dumps\\Dump20190418.sql"
13:23:43 Import of C:\Users\Documents\dumps\Dump20190418.sql has finished
```

Fig. 2.1.17 Select import completed

That’s it! You have successfully restored a MySQL database from a backup with MySQL Workbench.

EXPERIMENT 2

PERFORMANCE MONITORING OF DATABASE USING BUILT-IN TOOLS

Objective

The objective of this lab is to:

- Learn how to monitor and analyze the performance of MySQL Database Management System (DBMS) using built-in tools and commands.
- Understand how to gather real-time performance statistics using various MySQL tools.
- Analyze query performance using EXPLAIN, SHOW STATUS, and the Performance Schema.
- Identify potential performance bottlenecks (e.g., slow queries, high memory usage) and optimize MySQL configurations.
- Use tools like SHOW PROCESSLIST, InnoDB Status, and the Slow Query Log to troubleshoot and monitor MySQL queries.

Theory

MySQL is a widely-used relational database management system that offers several built-in tools for performance monitoring. These tools help DBAs and developers diagnose issues related to query performance, memory usage, CPU consumption, disk I/O, and other system resources.

Key Tools for MySQL Performance Monitoring:

1. **SHOW STATUS**: Provides server-wide statistics, including the number of queries, active threads, and buffer pool usage.
2. **SHOW VARIABLES**: Displays the current configuration settings, including memory and cache settings.
3. **EXPLAIN**: Analyzes SQL queries by showing the execution plan and identifying performance bottlenecks, such as table scans or inefficient joins.
4. **SHOW PROCESSLIST**: Displays all active connections and running queries, helping identify long-running or stuck queries.
5. **Performance Schema:** Collects detailed performance data on SQL queries, wait events, and system resources, providing insights into resource usage and bottlenecks.

6. **Slow Query Log**: Logs queries that take longer than a specified time to execute, allowing DBAs to analyze and optimize slow queries.

These tools are essential for performance monitoring and tuning in MySQL, helping optimize the database for better efficiency, faster query execution, and reduced resource usage.

In this lab, you will use various MySQL commands to monitor the performance of a MySQL database. Follow the steps below:

2.2.1 Using SHOW STATUS

Run the SHOW STATUS command to gather server statistics.
SHOW STATUS;

- Focus on metrics like:
  - Threads_connected: Number of active connections to the database.
  - Innodb_buffer_pool_reads: Number of reads from the disk.
  - Slow_queries: Number of slow queries executed.
- Analyze the status variables to identify any potential performance issues.

2.2.2 Using EXPLAIN to Analyze Query Performance

Choose a slow or inefficient query from the database and run the EXPLAIN command:
EXPLAIN SELECT * FROM orders WHERE customer_id = 12345;

- Review the EXPLAIN output, which shows the execution plan of the query.
  - **type**: The type of join used (e.g., ALL for full table scan, which is inefficient).
  - **rows**: The number of rows MySQL expects to scan.
  - **key**: The index used (if any) for the query.
- Use the output to optimize the query, such as adding appropriate indexes.

2.2.3 Using SHOW PROCESSLIST to Monitor Active Queries

Run the following to see all active processes:
SHOW PROCESSLIST;

- Identify long-running or blocking queries. If a query has been running for an extended period, investigate why it is taking so long.

2.2.4 Enabling and Analyzing the Slow Query Log

Enable the slow query log by editing your MySQL configuration file (my.cnf or my.ini) to include:

```
slow_query_log = 1
slow_query_log_file = /path/to/slow-query.log
long_query_time = 1  # Log queries that take longer than 1 second
```

- After enabling it, run some queries and analyze the log file.
    - Use mysqldumpslow or pt-query-digest from the Percona Toolkit to analyze the log and identify queries that need optimization.

2.2.5 Using the Performance Schema

Query the Performance Schema for a deeper look at query performance: SELECT * FROM performance_schema.events_statements_summary_by_digest;

- Review the query statistics, such as execution times and resource usage.
- Use this data to identify resource-heavy queries or areas for optimization.

2.2.6 Using SHOW ENGINE INNODB STATUS

Check the InnoDB engine's performance status with:

```
SHOW ENGINE INNODB STATUS;
```

- Analyze the output for signs of locking issues, deadlocks, and buffer pool usage.

Sample Question

Q1. You are monitoring a MySQL database and notice that the queries are running slowly. You suspect that the performance might be related to locking or resource contention. You need to identify which queries are currently being executed and check for any long-running queries that could be causing the issue.

Steps to solve:

1. **Check the Current Active Queries:**

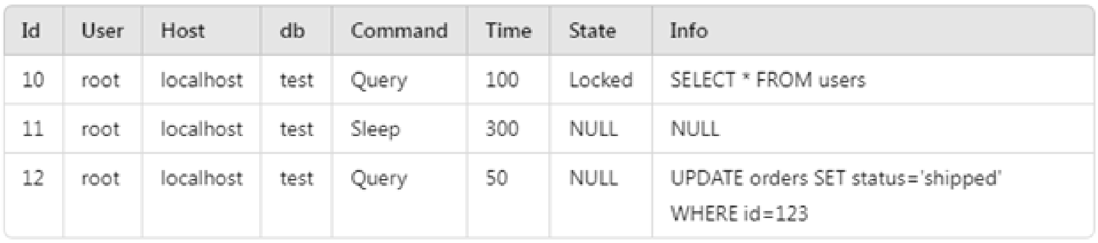
Use the SHOW PROCESSLIST command to see a list of active queries and identify if any queries are in a "Locked" or "Sleep" state for too long.

```
SHOW PROCESSLIST;
```

Output:

The output should display the following columns: **Id, User, Host, db, Command, Time, State, and Info.**

Example output



Id	User	Host	db	Command	Time	State	Info
10	root	localhost	test	Query	100	Locked	SELECT * FROM users
11	root	localhost	test	Sleep	300	NULL	NULL
12	root	localhost	test	Query	50	NULL	UPDATE orders SET status='shipped' WHERE id=123

Fig 2.2.1 Sample output using SHOW PROCESSLIST

Query ID 10 has been running for 100 seconds and is in a Locked state.

Query ID 11 has been idle for 300 seconds.

Q2. Run a slow or inefficient query on a sample table and analyze it using the EXPLAIN command:

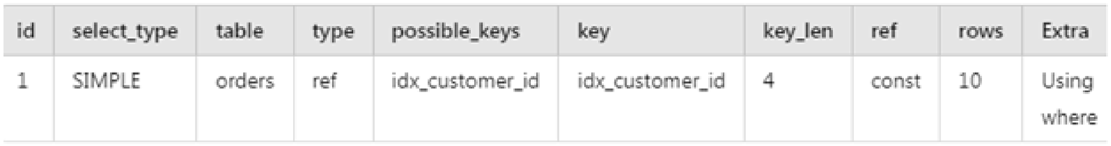
EXPLAIN SELECT * FROM orders WHERE customer_id = 12345;

Answer:

The EXPLAIN command provides the query execution plan, which includes details on how MySQL plans to retrieve data for the given query. The output of EXPLAIN will include the following columns:

- **id:** The step identifier of the query.
- **select_type:** The type of query, such as SIMPLE or PRIMARY.
- **table:** The table involved in this query step.
- **type:** The join type, which indicates whether a full table scan (ALL) or index scan (index) is used.
- **possible_keys:** Shows which indexes might be used for the query. Here, it lists idx_customer_id, which suggests an index on customer_id.
- **key:** The index used, if any.
- **key_len:** The length of the index used (in this case, 4 bytes).
- **ref:** The constant or value used to access the index. Here, const indicates that the customer_id value is used for lookups.
- **rows:** The estimated number of rows the query will scan. In this case, it estimates 10 rows will be returned based on the query plan.
- **Extra:** Additional information, such as "Using where" or "Using index," indicating how the query is being optimized.

If the type column shows ALL, it means MySQL is performing a full table scan, which is inefficient. In this case, adding an index on customer_id would likely improve performance by enabling index-based lookups instead of scanning the entire table.



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	orders	ref	idx_customer_id	idx_customer_id	4	const	10	Using where

Fig 2.2.2 Sample output using EXPLAIN

- **Efficiency:** The query uses an index (idx_customer_id), which should make it more efficient than a full table scan. However, if the table contains a large number of rows and the index isn't selective (i.e., many rows share the same customer_id), this can still result in scanning a significant number of rows.
- **Optimization Opportunities:**
    - If you find that the query is slow despite using an index, you might want to check the distribution of customer_id values. If there are many rows for the same customer, the query might still have to scan many rows.
    - Additionally, reviewing the database schema and ensuring that the customer_id index is optimized for the type of queries you run (e.g., a covering index) might further improve performance.

Lab Practice Questions

1. **Using SHOW STATUS:**

    Execute the following query:

    SHOW STATUS LIKE 'Threads_connected';

    - What does the Threads_connected variable indicate? What impact does a high number of active connections have on the database performance, and how can you manage connections effectively?

2. **Using EXPLAIN to Optimize Queries:**

    Run the following query:

    EXPLAIN SELECT * FROM products WHERE category_id = 10;

    - Review the EXPLAIN output. What information does it provide about the query execution, and how would you improve the query's performance based on this analysis?

Experiment 3

Identifying, Troubleshooting and Monitoring

Objectives

- Understand the nature and impact of performance bottlenecks in RDBMS environments.
- Learn methodologies to identify, analyze, and resolve performance issues within relational databases.
- Gain hands-on experience with tools and techniques used in database performance optimization.

Theory

2.3.1 Understanding Performance Bottlenecks in RDBMS

Performance bottlenecks in Relational Database Management Systems (RDBMS) can significantly degrade application efficiency and user satisfaction. Common causes include inefficient SQL queries, inadequate indexing, hardware limitations, and suboptimal database configurations. For instance, poorly structured queries can lead to excessive resource consumption, while insufficient indexing may result in full table scans, both contributing to slower response times. Additionally, limited CPU, memory, or storage resources can constrain database performance, and improper configuration settings may hinder scalability and efficiency.

To address these challenges, it's essential to implement effective monitoring and optimization strategies. Regularly analyzing query execution plans helps identify and rectify inefficient queries. Proper indexing enhances data retrieval performance, reducing the need for full table scans. Upgrading hardware resources or adjusting resource allocation can alleviate hardware-related bottlenecks. Furthermore, tuning database configurations to align with workload requirements ensures optimal performance. By proactively managing these aspects, organizations can enhance the responsiveness and reliability of their database systems, leading to improved user experiences and operational efficiency.

2.3.1.1 Common Causes of Bottlenecks

Performance bottlenecks in Relational Database Management Systems (RDBMS) can significantly impact application efficiency and user satisfaction. Common causes include:

1. **Inefficient SQL Queries:** Poorly written queries can lead to excessive resource consumption and slow response times.

2. **Lack of Proper Indexing:** Absence or misuse of indexes can result in full table scans, increasing query execution time.

3. **Locking and Blocking:** Concurrent transactions may cause locks, leading to delays in data access.

4. **Hardware Limitations:** Insufficient CPU, memory, or disk I/O capacity can constrain database performance.

5. **Suboptimal Configuration:** Improper database settings can hinder performance and scalability.

2.3.1.2 Importance of Identifying Bottlenecks

Identifying bottlenecks in a Relational Database Management System (RDBMS) is crucial for maintaining optimal performance and ensuring efficient data retrieval and processing. Bottlenecks, such as inefficient queries, inadequate indexing, or hardware limitations, can lead to increased response times and reduced system throughput. By proactively identifying and addressing these issues, organizations can enhance user satisfaction and maintain a competitive edge in today's data-driven landscape.

Moreover, recognizing performance bottlenecks allows for informed decision-making regarding resource allocation and system optimization. Understanding the root causes of performance issues enables database administrators to implement targeted solutions, such as query optimization, proper indexing, and hardware upgrades, thereby improving overall system efficiency. This proactive approach not only enhances the reliability and scalability of the RDBMS but also contributes to cost savings by preventing the need for more extensive corrective measures in the future.

2.3.1.3 Methods for Identifying Bottlenecks

Identifying performance bottlenecks in a Relational Database Management System (RDBMS) is crucial for maintaining optimal performance and ensuring efficient data processing. The following methods are commonly employed to detect and address these bottlenecks:

1. **Monitor System Performance:** Utilize performance monitoring tools to track key metrics such as CPU usage, memory utilization, disk I/O, and network activity. These metrics help in pinpointing resource constraints that may be affecting database performance.

2. **Analyze Query Execution Plans:** Examine the execution plans of SQL queries to understand how the database engine processes them. This analysis can reveal inefficiencies like unnecessary full table scans or suboptimal join operations, guiding necessary optimizations.

3. **Utilize SQL Profiling Tools:** Employ SQL profiling tools to gather detailed information on query execution times, resource consumption, and performance patterns. This data assists in identifying slow-running queries

and understanding their impact on overall system performance.

4. **Evaluate Index Usage:** Assess the effectiveness of existing indexes and identify missing or redundant ones. Proper indexing can significantly enhance data retrieval speeds and reduce query execution times.

5. **Review Database Configuration Settings:** Ensure that database parameters are optimized for the specific workload and environment. Suboptimal configurations can lead to inefficient resource utilization and degraded performance.

By systematically applying these methods, organizations can effectively identify and resolve performance bottlenecks in their RDBMS, leading to improved efficiency and a better user experience.

2.3.2 Strategies for Troubleshooting Bottlenecks

Troubleshooting performance bottlenecks in Relational Database Management Systems (RDBMS) is essential for maintaining efficient data processing and ensuring optimal application performance. The following steps can assist in identifying and resolving these bottlenecks:

1. **Monitor Database Performance:** Regular monitoring is crucial to detect anomalies and potential issues early.

    - **Utilize Performance Monitoring Tools:** Employ database-specific tools to track metrics such as query response times, transaction throughput, and resource utilization. These tools can help identify trends and pinpoint areas requiring attention.

    - **Analyze System Logs:** Review logs for errors, warnings, and slow query reports to gain insights into underlying problems affecting performance.

2. **Identify Resource Bottlenecks:** Determine which system resources are limiting performance. Adequate hardware resources are vital for optimal database performance. To address hardware-related bottlenecks:

    - **Assess Resource Utilization:** Monitor CPU, memory, disk I/O, and network bandwidth to identify resource shortages.

    - **Upgrade Hardware Components:** Enhance server components such as processors, RAM, or storage systems to accommodate increased workloads.

    - **Implement Load Balancing:** Distribute database load across multiple servers to prevent any single server from becoming a bottleneck.

    - **CPU:** High CPU usage may indicate inefficient queries or inadequate hardware. Identifying and optimizing resource-intensive queries can alleviate CPU pressure.

    - **Memory:** Insufficient memory can lead to increased disk I/O operations, slowing down performance. Ensuring adequate memory allocation for the database can mitigate this issue.

- ♦ **Disk I/O:** Slow disk operations can bottleneck data retrieval and storage processes. Upgrading storage systems or optimizing data access patterns can help address disk I/O limitations.

3. **Optimize Queries:** Efficient queries reduce load and improve response times.

   - ♦ **Analyze Execution Plans:** Examine how queries are executed to identify inefficiencies such as unnecessary full table scans or improper join operations.
   - ♦ **Refactor Queries:** Simplify complex queries, ensure proper indexing, and avoid retrieving unnecessary data to enhance performance, ensuring that filters and joins are appropriately used to minimize processed data.
   - ♦ ***Avoid SELECT :** Retrieve only the necessary columns instead of using `SELECT *`, reducing the amount of data transferred and processed.

4. **Manage Indexes:** Proper indexing accelerates data retrieval but requires careful management.

   - ♦ **Create Necessary Indexes:** Identify columns frequently used in search conditions and joins, and index them appropriately.
   - ♦ **Remove Redundant Indexes:** Eliminate unused or duplicate indexes to reduce maintenance overhead and potential performance degradation.
   - ♦ **Monitor and Maintain Indexes:** Regularly assess the effectiveness of existing indexes, removing redundant ones and adding new ones as query patterns evolve.
   - ♦ **Choose Suitable Index Types:** Understand the different types of indexes, such as B-tree and bitmap indexes, and apply them based on the specific use case.

5. **Adjust Database Configuration:** Adjusting database settings to align with best practices and workload requirements can significantly improve performance. Key areas include:

   - ♦ **Memory Allocation:** Configure buffer pools and cache sizes to match the demands of your applications.
   - ♦ **Connection Management:** Optimize connection pooling and set appropriate limits to manage concurrent user access effectively.
   - ♦ **Parameter Optimization:** Fine-tune database parameters such as query timeout settings, parallel processing limits, and logging levels to suit specific application needs.

6. **Upgrade Hardware Resources:** When software optimizations are insufficient, hardware upgrades may be necessary.

- **Enhance Server Components:** Upgrading CPUs, adding memory, or improving storage systems can provide the necessary resources to handle increased workloads.
- **Implement Load Balancing:** Distributing the database load across multiple servers can prevent any single server from becoming a performance bottleneck.

7. **Regular Maintenance:** Proactive maintenance ensures sustained database performance.
    - **Update Statistics:** Keep database statistics current to enable the optimizer to make informed decisions.
    - **Rebuild Indexes:** Regularly rebuild or reorganize fragmented indexes to maintain efficient data access paths.

By systematically implementing these strategies, organizations can effectively reduce resource consumption, improve execution speed, and enhance the overall performance of their RDBMS.

Exercise Questions

Exercise 1: Analyzing Query Execution with Profiling Tools

Objective: Learn to use query profiling tools to identify performance bottlenecks in SQL queries.

Scenario: You have a table named Orders with columns: `OrderID, CustomerID, OrderDate`, and `TotalAmount`. You execute the following query to retrieve orders placed on a specific date:

```
SELECT *

FROM Orders

WHERE OrderDate = '2023-10-01';
```

Tasks:

1. **Enable Query Profiling:** Turn on profiling to gather detailed performance metrics. In SQL Server, use:

```
SET STATISTICS TIME ON;

SET STATISTICS IO ON;
```

2. **Execute the Query:** Run the provided SQL query to collect performance data.

3. Analyze Profiling Results: Review the output to assess CPU time and I/O operations. High values may indicate inefficiencies.

4. Optimize the Query: If the profiling indicates a full table scan, consider creating an index on the `OrderDate` column:

```
CREATE INDEX idx_order_date ON Orders(OrderDate);
```

5. Re-evaluate Performance: Re-run the query with profiling enabled to observe improvements.

Expected Outcome:

- ♦ Before indexing, the query may exhibit high CPU and I/O usage due to full table scans.
- ♦ After indexing, reduced resource consumption and faster execution times should be evident.

Exercise 2: Monitoring and Analyzing Wait Statistics

Objective: Understand how to monitor wait statistics to identify resource contention issues affecting database performance.

Scenario: Your database has been experiencing slow performance, and you suspect that resource waits might be contributing to the issue.

Tasks:

1. **Retrieve Wait Statistics:** Execute the following query to identify the top wait types:

```
SELECT TOP 5 wait_type, SUM(wait_time_ms) AS total_wait_time_ms

FROM sys.dm_os_wait_stats

GROUP BY wait_type

ORDER BY total_wait_time_ms DESC;
```

2. **Analyze Results:** Identify which wait types have the highest total wait times. Common wait types include `PAGEIOLATCH_SH` (indicating slow disk I/O) and `LCK_M_X` (indicating blocking due to locks).

3. **Investigate Causes:**

    - For I/O-related waits, examine disk performance and consider optimizing queries or indexes to reduce I/O operations.
    - For locking waits, analyze query concurrency and consider strategies like query tuning or isolation level adjustments.

4. **Implement Solutions:** Based on the analysis, apply appropriate optimizations such as indexing, query rewriting, or hardware upgrades.

5. **Monitor Impact:** After implementing changes, re-run the wait statistics query to assess improvements.

Expected Outcome:

- Identification of the primary causes of waits affecting database performance.
- Implementation of targeted optimizations leading to reduced wait times and improved system responsiveness.

2.3.3 Get started with External Monitoring Tools

Third-party (External) monitoring tools are essential for maintaining the performance, availability, and security of Relational Database Management Systems (RDBMS). They offer advanced features beyond native monitoring capabilities, enabling database administrators (DBAs) to proactively manage and troubleshoot database environments.

2.3.3.1 Advantages of Third-Party Database Monitoring Tools

- **Enhanced Performance Insights:** These tools provide real-time monitoring of database operations, allowing DBAs to detect and address performance issues promptly.
- **Comprehensive Alerting Systems:** They offer sophisticated alerting mechanisms that notify administrators of potential problems, facilitating swift corrective actions.
- **Scalability:** As organizations grow, these tools can scale to monitor increasing data loads, ensuring consistent performance monitoring without degradation.

2.3.3.2 Challenges Addressed by Third-Party Tools

- **Complexity of Multi-Platform Environments:** With the adoption of diverse database platforms, third-party tools provide a unified monitoring interface, simplifying management across different systems.
- **Handling Large Data Volumes:** They efficiently manage and analyze large datasets, ensuring timely reporting and analysis without data loss.

2.3.3.3 Considerations for Selecting a Monitoring Tool

- **Integration Capabilities:** Ensure the tool integrates seamlessly with existing infrastructure and supports various database platforms.
- **Customization and Flexibility:** The tool should allow customization to cater to specific organizational needs and workflows.
- **Cost vs. Benefit Analysis:** Evaluate the tool's features against its cost to determine its return on investment.

2.3.3.4 Popular Tools for Database Monitoring

Effective database monitoring is crucial for maintaining optimal performance, ensuring availability, and preventing potential issues within database systems. Several tools are available to assist database administrators in these tasks. Here are some notable database monitoring tools:

1. ManageEngine Applications Manager

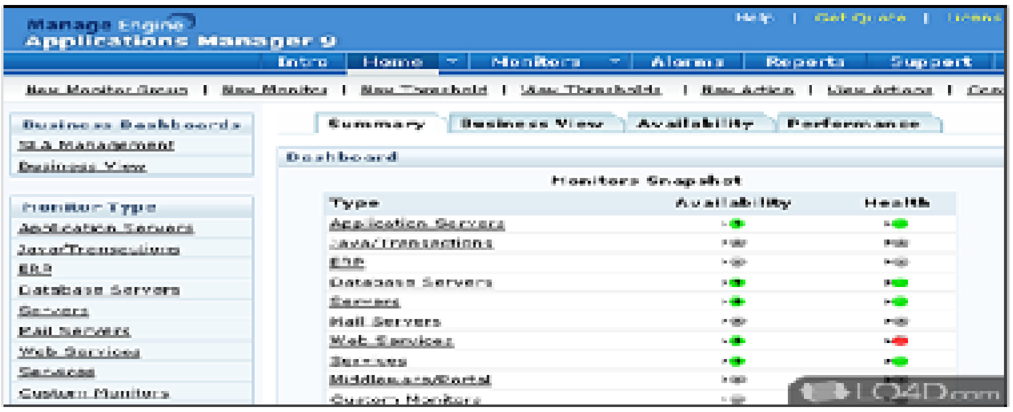
ManageEngine Applications Manager is a performance and IT infrastructure monitoring tool designed to address modern infrastructure challenges and ensure smooth operations for businesses and their users. It combines deep application performance monitoring with infrastructure, hybrid cloud, and digital experience monitoring, all accessible from a single platform. With its code-level insights, automation capabilities, and performance issue detection, IT and DevOps teams can streamline their processes and improve business outcomes. The tool helps businesses optimize application performance, reduce mean time to resolution (MTTR), enhance DevOps workflows, support cloud migration, and respond quickly to incidents, ensuring a seamless user experience. The dashboard of ManageEngine Applications Manager is shown in Fig.2.3.1.

Benefits

- Comprehensive monitoring of applications, servers, and databases with real-time diagnostics to minimize downtime.
- Intuitive user interface that simplifies navigation and enhances operational efficiency.
- Automation of routine monitoring tasks, allowing IT teams to focus on strategic initiatives.
- Cost-effective solution, suitable for both small and large businesses.

Things to consider

- The user interface may occasionally face responsiveness issues, which can affect the overall user experience.
- The tool may generate a high volume of alerts, including false positives, which could lead to alert fatigue for IT staff.


ManageEngine Applications Manager 9

Help | Get Quote | Licens

Intro | Home | Monitors | Alarms | Reports | Support

New Monitor Group | New Monitor | New Threshold | View Thresholds | New Action | View Actions | Con

Business Dashboards
- SLA Management
- Business View

Monitor Type
- Application Servers
- Java/Transactions
- ERP
- Database Servers
- Servers
- Mail Servers
- Web Services
- Services
- Custom Monitors

Summary | Business View | Availability | Performance

Dashboard

Monitors Snapshot

Type	Availability	Health
Application Servers	🟩	🟩
Java/Transactions	○	○
ERP	○	○
Database Servers	🟩	🟩
Servers	🟩	🟩
Mail Servers	○	○
Web Services	🟩	🟥
Services	🟩	🟩
Middleware/Portal	○	○
Custom Monitors	○	○

Fig. 2.3.1 Dashboard of ManageEngine Applications Manager

2. Dynatrace

Dynatrace (NYSE: DT) aims to ensure that software operates flawlessly. Our integrated platform merges extensive observability and ongoing runtime application security with Davis hypermodal AI, delivering insights and intelligent automation from vast amounts of data. This empowers innovators to upgrade and streamline cloud operations, release software more quickly and securely, and guarantee seamless digital experiences. This is why leading global organizations rely on the Dynatrace platform to speed up their digital transformation efforts. The dashboard of Dynatrace is shown in Fig.2.3.2.

Benefits

- Its user-friendly interface and extensive range of features.
- Offer a comprehensive, holistic view of our complete application environment.

Things to consider

- Dynatrace's pricing may be considered relatively high, particularly for smaller teams or businesses that are just beginning.

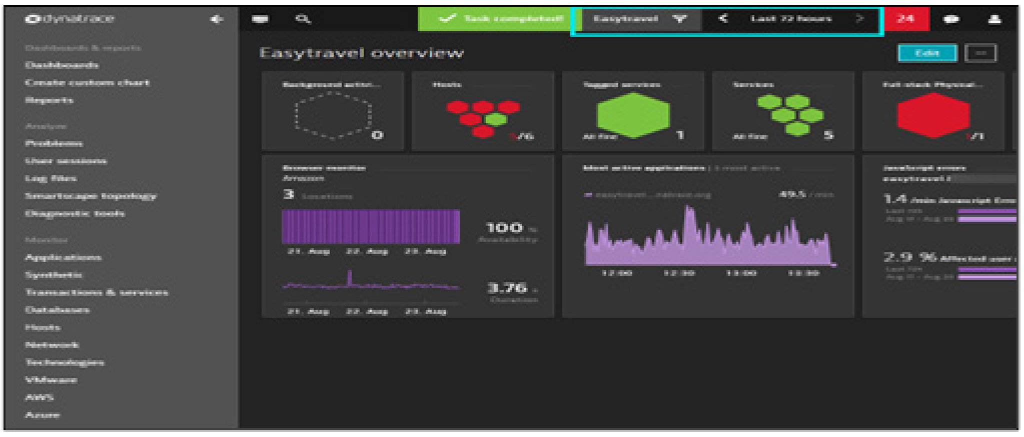


Fig. 2.3.2 Dashboard of Dynatrace

3. Datadog

Datadog is a monitoring, security, and analytics platform designed for developers, IT operations teams, security engineers, and business users in the cloud era. This SaaS platform seamlessly integrates and automates infrastructure monitoring, application performance monitoring, and log management, delivering unified, real-time visibility into its customers' entire technology stack. Organizations of all sizes and across various industries utilize Datadog to facilitate digital transformation and cloud migration, foster collaboration among development, operations, security, and business teams, expedite application time to market, decrease problem resolution time, secure applications and infrastructure, analyze user behavior, and monitor essential business metrics. The dashboard of Datadog is shown in Fig.2.3.3.

Benefits

- ♦ Finding and fixing bugs before users encounter them is a significant advantage.
- ♦ Datadog offers numerous useful features and integrations for applications, including metrics and monitoring.
- ♦ It provides customizable dashboards.
- ♦ Supports cost optimization.

Things to consider

- ♦ The setup documentation can be somewhat challenging to comprehend.

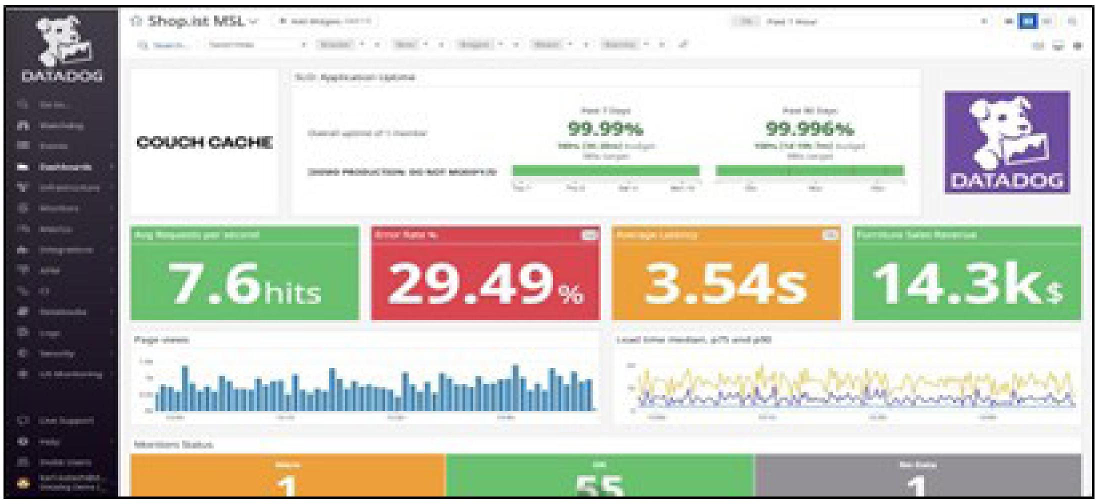


Fig. 2.3.3 Dashboard of Datadog

4. SolarWinds Database Performance Analyzer

This cross-platform solution monitors database performance for both cloud and on-premises databases. With machine learning-driven anomaly detection and in-depth wait-time analysis, you can diagnose performance issues in minutes rather than days. Access to both real-time and historical data provides precise answers to address critical problems, while expert guidance through query and table-tuning advisors enables proactive optimization of your enterprise. The dashboard of SolarWinds Database Performance Analyzer is shown in Fig.2.3.4.

Benefits

- A simple setup process using a command line and PowerShell module allows for easy target addition and management, with conditions that can be directly exported.
- Includes tools for quickly identifying the root cause of issues by maintaining a history of events.
- Features an excellent alert system to notify users of any problems.
- Outstanding customer support, with helpful personnel readily available for assistance.

Things to consider

- There is potential for enhancing performance monitoring. At times, accessing top SQL can be challenging due to the complexities of the GUI.

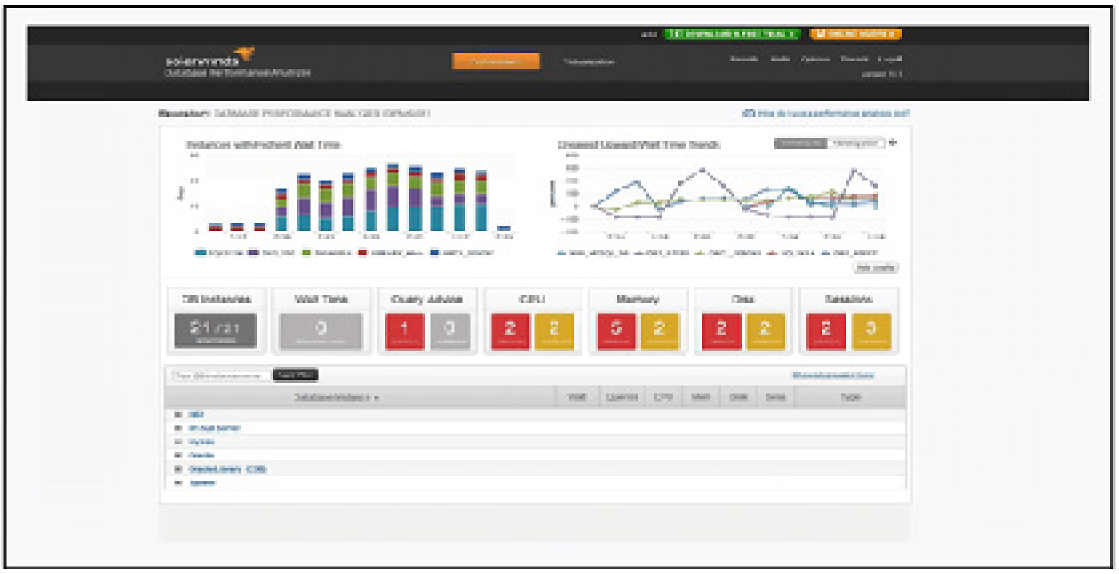


Fig. 2.3.4 Dashboard of SolarWinds Database Performance Analyzer

5. New Relic Database Monitoring

New Relic pioneered cloud APM for application engineers and is now a leader in observability. It serves as a trusted source of data-driven decision-making for engineers throughout the software stack and lifecycle. With around 25 million engineers worldwide spanning over 25 roles, New Relic empowers them to gain real-time insights and track performance trends, helping businesses become more resilient and deliver exceptional customer experiences. The dashboard of New Relic Database Monitoring is shown in Fig.2.3.5.

Benefits

- It provides users with a comprehensive view of application performance, enabling proactive identification and resolution of issues before they escalate.

- The customizable dashboards and alerting features make it easy to monitor key metrics and stay in control of your system's health.

Things to consider

- For beginners, the platform can be challenging to learn, as navigating the wide range of tools and metrics available may feel overwhelming.

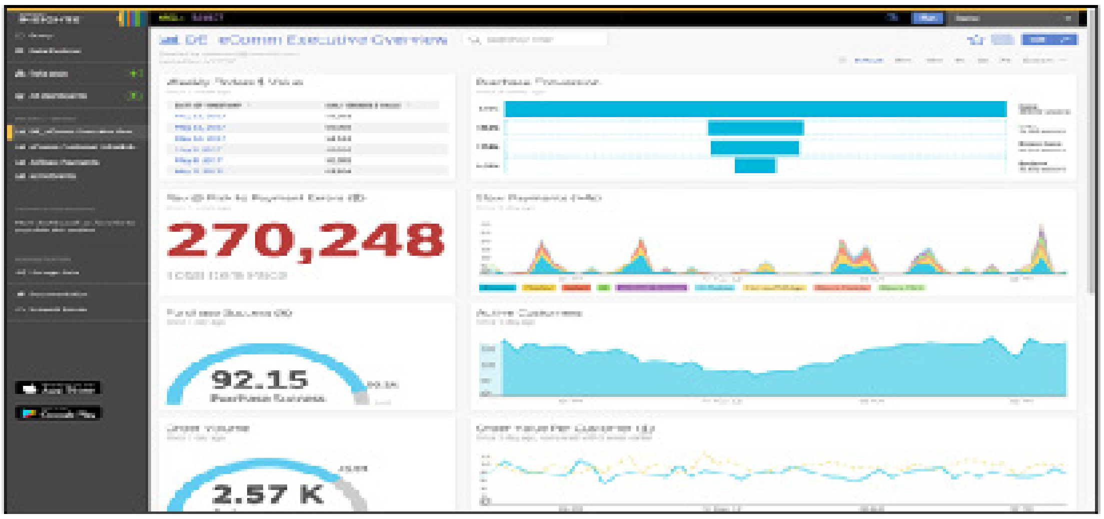


Fig. 2.3.5 Dashboard of New Relic Database Monitoring

6. AppDynamics Database Monitoring

Backed by Cisco, AppDynamics is dedicated to helping companies view their technology from a business perspective, enabling them to unite and focus on what truly matters. We are revolutionizing the observability landscape and streamlining the complexities of digital transformation for the world's largest enterprises. The AppDynamics Business Observability Platform accelerates organizational transformation by offering deep business insights across the technology stack, aligning teams with common goals, and empowering technologists to take decisive, confident action. The dashboard of AppDynamics Database Monitoring is shown in Fig.2.3.6.

Benefits

- AppDynamics is a highly versatile tool that integrates seamlessly with a wide range of third-party services, including SaaS, cloud platforms, and other data monitoring tools.
- It excels in identifying and monitoring problematic transactions, helping to boost business productivity.
- Additionally, configuring servers, analytics, metrics, and alerts is straightforward, making it an efficient solution for monitoring and performance management.

Things to consider

- The costs can be quite high for larger organizations.

- Competitors offer similar services at more competitive pricing.

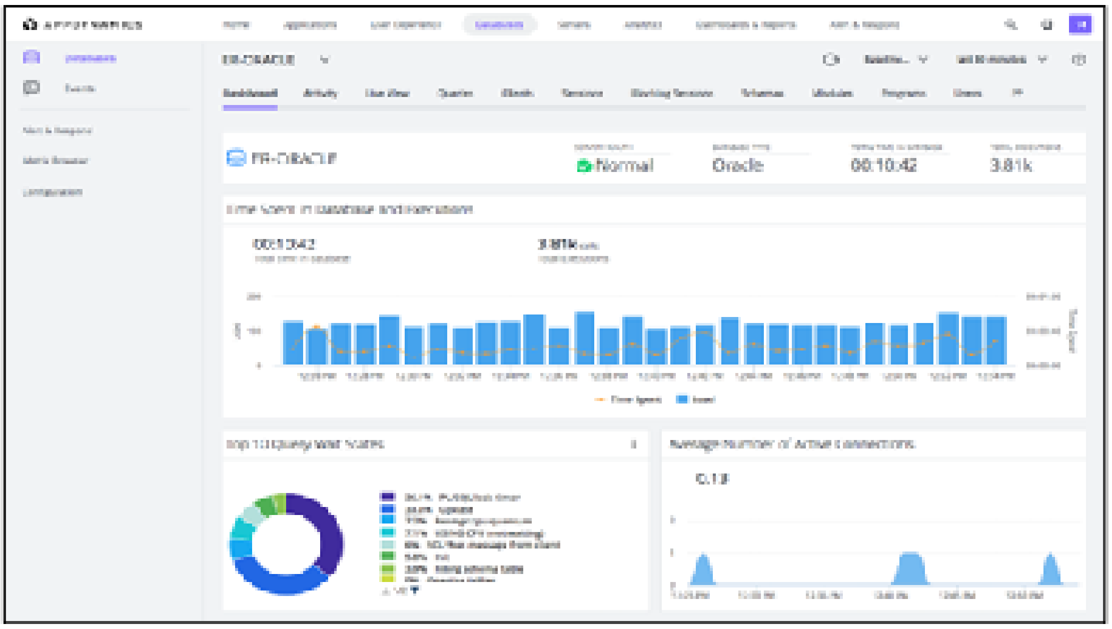


Fig. 2.3.6 Dashboard of AppDynamics Database Monitoring

7. SQL Diagnostic Manager

Decreased availability and performance of SQL Server can significantly affect the vital applications it supports. SQL Diagnostic Manager for SQL Server offers comprehensive monitoring, covering the entire SQL Server environment and delivering the most detailed diagnostics available. It monitors physical, virtual, and cloud environments, tracks queries and plans to resolve blocks and locks, provides predictive alerts to minimize false notifications, and includes expert advice with actionable scripts. The dashboard of SQL Diagnostic Manager is shown in Fig.2.3.7.

Benefits

- Easy to install and configure
- Users like query optimization
- Detailed descriptions of the query

Things to consider

- The initial setup is time-consuming.

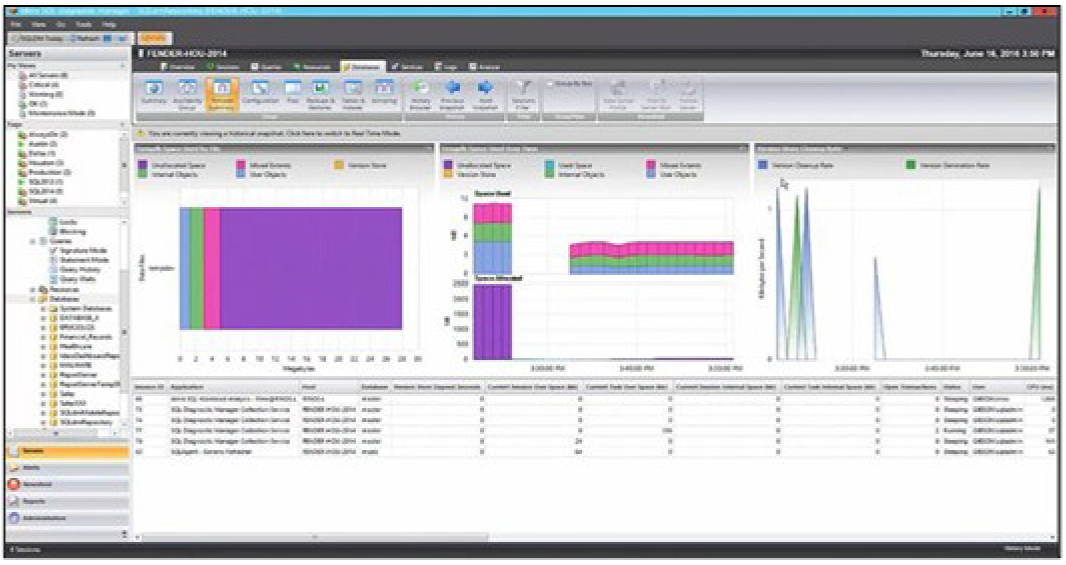


Fig. 2.3.7 Dashboard of SQL Diagnostic Manager

8. Foglight for Databases

Quest's Foglight for Databases provides a comprehensive solution for database monitoring, seamlessly integrating performance tracking across various platforms, whether cloud-based or on-premises. It supports relational, non-relational, open-source, and cloud-exclusive databases. With advanced diagnostic tools, organizations can actively manage and optimize both real-time and historical performance, troubleshoot issues, and improve the overall health of their database environment. The dashboard of Foglight for Databases is shown in Fig.2.3.8.

Benefits

- ♦ Navigating the environment overview provides a quick and convenient summary of the activity across hosts and clusters, making it easier to monitor the system.
- ♦ The Alarm module is highly effective, offering detailed reports on the status of physical disks.

Things to consider

- ♦ The Foglight application server is resource-intensive and consumes significant memory. Therefore, if you plan to host it on-premises, you must account for the additional hardware costs required to support its performance.

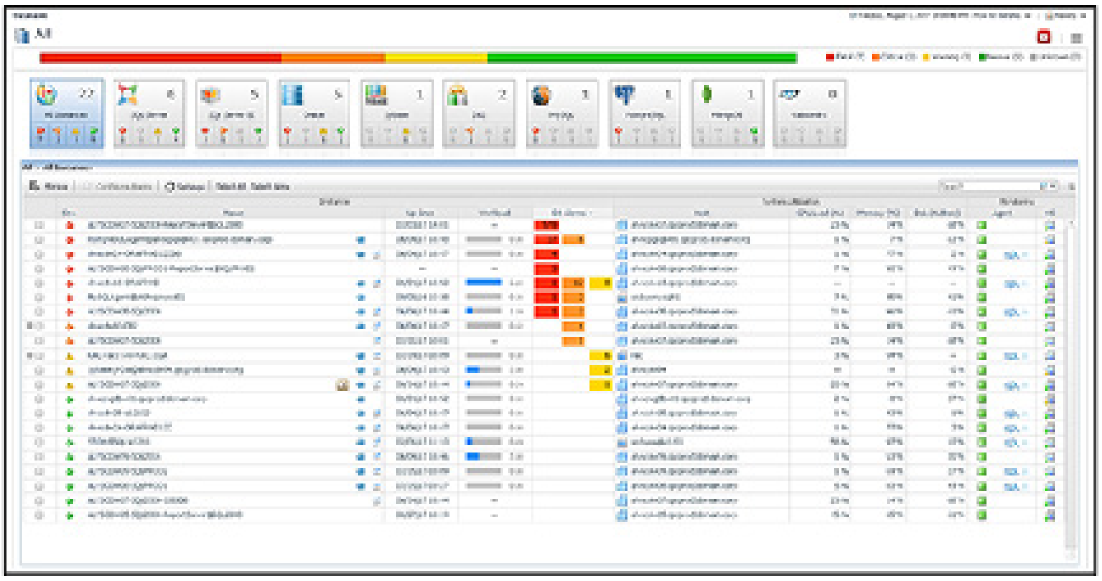


Fig. 2.3.8 Dashboard of Foglight for Databases

9. Redgate SQL Monitor

Redgate designs innovative, user-friendly software to help data professionals maximize the value of any database through comprehensive Database DevOps solutions. Trusted by over 200,000 users worldwide, including 92% of Fortune 100 companies, Redgate delivers simple yet powerful tools. Redgate Monitor allows you to oversee your entire database ecosystem from a unified dashboard. It enables proactive risk management with instant issue diagnosis and customizable alerts, regardless of where your databases are hosted. Say goodbye to downtime, customer complaints, and 3 am wake-up calls. The dashboard of Redgate SQL Monitor is shown in Fig.2.3.9.

Benefits

- The platform provides a user-friendly interface with real-time dashboards and intuitive navigation, making management easier.
- A highly customizable alerting system delivers timely notifications about performance issues or potential bottlenecks, helping save time and prevent escalation.
- Clear, actionable performance reports with visual insights simplify troubleshooting and are supported by comprehensive reference documentation.
- The installation process is straightforward and well-supported.
- Adding or removing servers from the monitoring dashboard is quick and hassle-free.

Things to consider

- The update installation process could be refined to match the product's overall high-quality experience.

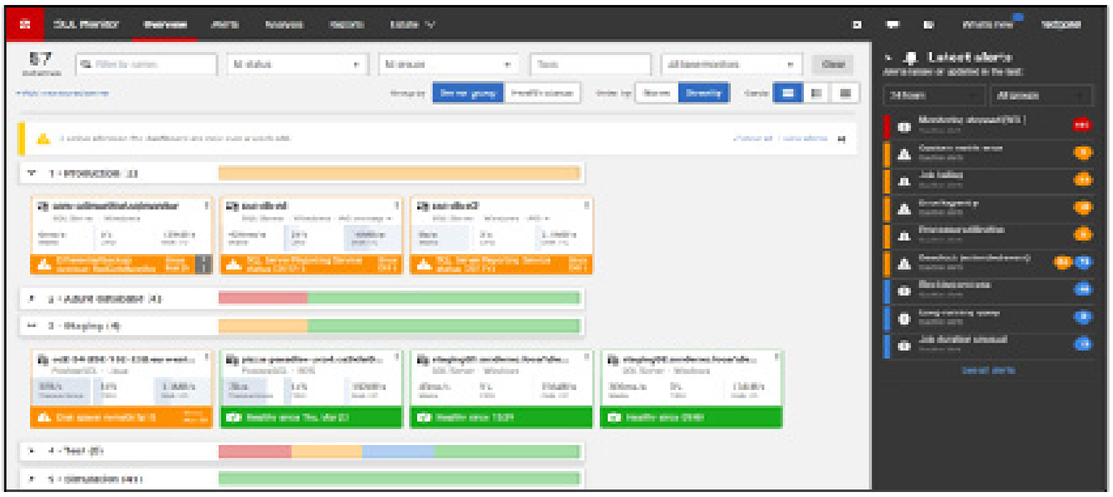


Fig. 2.3.9 Dashboard of Redgate SQL Monitor

10. Prometheus

An open-source monitoring system featuring a dimensional data model, a versatile query language, an efficient time series database, and a contemporary alerting method. The dashboard of Prometheus is shown in Fig.2.3.10.

Benefits

- ♦ It can gather various metrics from multiple applications simultaneously and provides customizable alerts based on the collected data to suit user preferences.
- ♦ Additionally, it offers strong support for Kubernetes infrastructure and is completely free to use.

Things to consider

- ♦ Troubleshooting high memory usage can be challenging.
- ♦ Update processes tend to be slow, leading to delays in improvements.

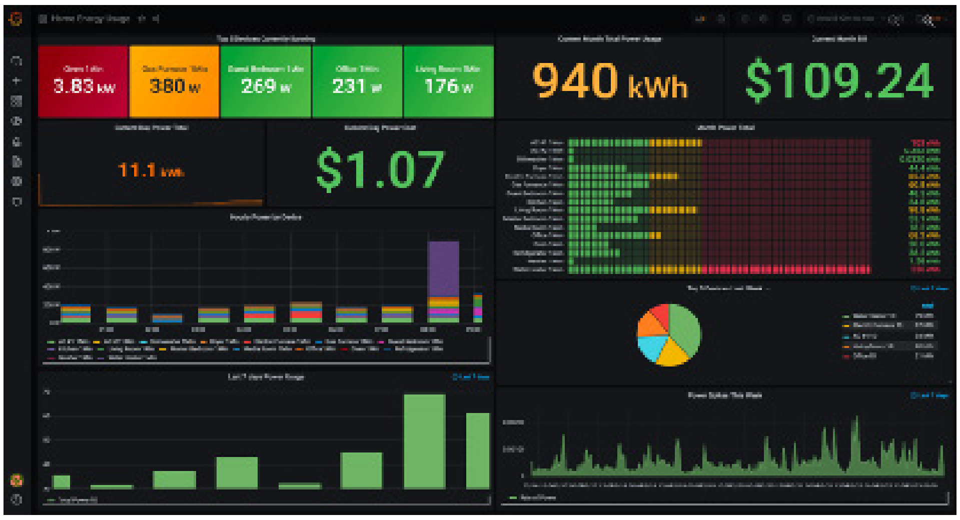


Fig. 2.3.10 Dashboard of Prometheus

Exercise Questions

Exercise 1: Configuring Alerts in Datadog for PostgreSQL

Objective: Set up Datadog to monitor a PostgreSQL database and configure alerts for specific performance thresholds.

Steps:

1. **Sign Up and Install Datadog Agent:**
   - Create a Datadog account.
   - Install the Datadog Agent on your PostgreSQL server by following the instructions provided in the Datadog documentation.
2. **Enable PostgreSQL Integration:**
   - In the Datadog web interface, navigate to the Integrations page and enable the PostgreSQL integration.
   - Configure the integration by providing connection details and credentials for your PostgreSQL instance.
3. **Create a Dashboard:**
   - Set up a new dashboard in Datadog to visualize key PostgreSQL metrics such as active connections, cache hit rate, and replication lag.
4. **Configure Alerts:**
   - Define alert conditions for critical metrics. For example, set an alert to trigger if the number of active connections exceeds a certain threshold, indicating potential connection saturation.
   - Specify notification channels (e.g., email, Slack) to receive alerts.

Expected Outcome: Datadog will actively monitor your PostgreSQL database, and you will receive timely alerts when predefined performance thresholds are breached, allowing for proactive database management.

Experiment Questions

1. **Profiling SQL Queries to Detect Performance Bottlenecks**

Use SQL Server's built-in profiling tools to identify slow-performing queries and understand their resource consumption

2. **Using SolarWinds Database Performance Analyzer to Monitor Database Performance**

Utilize SolarWinds Database Performance Analyzer, a graphical tool, to monitor and analyze SQL Server events, helping to identify performance bottlenecks.

CYCLE III

NoSQL

Experiment-1

Installation of configuration of distributed and wide – column store DB

Objectives

- Understand the architecture and working of Apache Cassandra.
- Learn how to install and configure Cassandra in both Windows and Linux environments.
- Perform basic operations in Cassandra, such as creating a keyspace, tables, inserting, updating, and retrieving data.
- Gain hands-on experience in executing CQL (Cassandra Query Language) commands.

Theory

Apache Cassandra is an open-source, distributed NoSQL database designed to handle large amounts of data across multiple commodity servers with high availability and fault tolerance. It is optimized for scalability and is widely used in big data applications.

3.1.1 Installation of Apache Cassandra

Apache Cassandra is a highly scalable, distributed NoSQL database designed for handling large amounts of data across multiple servers. Known for its fault tolerance, high availability, and decentralized architecture, Cassandra is widely used in big data applications. Installing Apache Cassandra involves setting up Java, configuring environment variables, and ensuring proper system requirements before starting the database service.

Step 1: Install Java (JDK)

Apache Cassandra requires Java to run. Before installing Cassandra, ensure that Java is installed on your system.

For Ubuntu/Linux:

1. **Open a terminal:**
   - You can open the terminal using Ctrl + Alt + T.
2. **Update the package list:**
   - Run the following command to update the system package list:

```
sudo apt update
```

3. **Install Java:**

   - Install OpenJDK 11 by running:

```
Install OpenJDK 11 by running:
```

4. **Verify installation:**

   - Check if Java is installed correctly by running:

```
java  -version
```

If installed correctly, you should see output similar to openjdk version "11.x.x".

For Windows:

1. **Download JDK:**

   - Visit Oracle's official website and download the latest version of JDK.

2. **Install JDK:**

   - Run the downloaded installer and follow the on-screen instructions.

3. **Set Environment Variables:**

   - Open **Control Panel > System > Advanced system settings.**
   - Click on **Environment Variables.**
   - Under **System Variables**, click **New**, and set **JAVA_HOME** to the JDK installation path (e.g., C:\Program Files\Java\jdk-11).

4. **Verify installation:**

   - Open Command Prompt and run:

You should see a message indicating that Java is installed.

Step 2: Download and Install Apache Cassandra

For Ubuntu/Linux:

1. **Add the Apache Cassandra repository:**

   - Run the following command to add the official Cassandra repository to your system:

```
echo "deb https://debian.cassandra.apache.org 41x main" | sudo tee -a /etc/apt/
sources.list.d/cassandra.list
```

2. **Add the repository key:**

   ♦ This ensures that the repository is trusted:

```
curl https://downloads.apache.org/cassandra/KEYS | sudo apt-key add -
```

3. **Update the package list:**

   ♦ Refresh the package list to include Cassandra:

```
sudo apt update
```

4. **Install Cassandra:**

   ♦ Install Apache Cassandra using:

```
sudo apt install cassandra -y
```

5. **Start Cassandra:**

   ♦ Start the Cassandra service by running:

```
sudo systemctl start cassandra
```

6. **Check if Cassandra is running:**

   ♦ Verify its status:

```
sudo systemctl status cassandra
```

   ♦ If running, you should see "active (running)" in the output.

For Windows:

1. **Download Cassandra:**

   ♦ Visit Apache Cassandra's website and download the latest stable release.

2. **Extract the Cassandra files:**

   ♦ Unzip the downloaded file to a preferred location, such as

```
    C:\Cassandra
```

3. **Set Environment Variables:**

   ♦ Open Control Panel > System > Advanced system settings.

   ♦ Click Environment Variables and set CASSANDRA_HOME to the extracted folder path.

4. **Start Cassandra:**

    - Open Command Prompt as Administrator and navigate to the Cassandra bin folder:

```
cd C:\Cassandra\bin
```

    - Run Cassandra in the foreground:

```
cassandra -f
```

5. **Verify installation:**

    - Check the Cassandra cluster status:

```
nodetool status
```

If Cassandra is running correctly, you will see information about the cluster.

Step 3: Configuring Cassandra

Locate the Cassandra****Step 3: Configuring Cassandra

1. **** configuration file:****

    - The main configuration file is named cassandra.yaml.
    - Linux: /etc/cassandra/cassandra.yaml
    - Windows: C:\Cassandra\conf\cassandra.yaml

2. **Open the file in a text editor:**

```
sudo nano /etc/cassandra/cassandra.yaml
```

3. **Modify key settings:**

    - Cluster Name: Change the default cluster name to your desired name.
    - Seeds: Define the IP addresses of the seed nodes for a multi-node setup.
    - Listen Address: Set it to 0.0.0.0 if running on multiple nodes.

4. **Save and exit the file:**

    - In the nano editor, press CTRL+X, then Y, then Enter.

5. **Restart Cassandra to apply changes:**

```
sudo systemctl restart cassandra
```

Step 4: Connecting to Cassandra

1. **Open Cassandra Query Language Shell (CQLSH):**

```
cqlsh
```

2. **Verify the connection:**

   ♦ Run the following command in CQLSH:

```
SELECT release_version FROM system.local;
```

It should display the installed Cassandra version.

3.1.2 Basic operations of cassandra

Cassandra allows users to store and manage large amounts of data in a distributed manner using **Cassandra Query Language** (CQL). Below are basic operations in cassandra.

3.1.2.1 Creating a keyspace

A keyspace in Cassandra is similar to a database in relational databases. It defines replication settings and serves as a container for tables.

Syntax:

```
CREATE KEYSPACE my_keyspace WITH replication = {'class': 'SimpleStrategy',
'replication_factor': 1};
```

♦ SimpleStrategy is used for single-node clusters.

♦ replication_factor defines how many copies of data exist.

Switch to the keyspace:

```
USE my_keyspace;
```

3.1.2.2 Creating a table

In Cassandra, a table is used to store data in a structured way. The CREATE TABLE command defines the table's name, columns, and data types. For example, the users table has three columns: id (a unique identifier), name (text), and age (an integer).

Syntax:

```
CREATE TABLE users (
   id UUID PRIMARY KEY,
   name text,
   age int
);
```

- UUID is used for unique user identification.
- PRIMARY KEY ensures uniqueness.

3.1.2.3 Inserting Data into a Table

In Cassandra, we use the INSERT statement to add records into a table in a similar way to SQL databases.

Syntax:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

- table_name: Name of the table where the record is inserted.
- column1, column2, ...: Names of the columns in the table.
- value1, value2, ...: Corresponding values to be inserted.

Example: Inserting Data into a Table

Consider a "students" table with the following schema:

```
CREATE TABLE students (

    id UUID PRIMARY KEY,

    name TEXT,

    age INT,

    department TEXT

);
```

Now, we can insert records into this table using the INSERT statement:

```
INSERT INTO students (id, name, age, department)

VALUES (uuid(), 'Alice', 20, 'Computer Science');
```

3.1.2.4 Retrieving Data

Fetching data from a table in Cassandra is done using the SELECT statement. It allows users to retrieve data based on specific conditions or fetch all records from a table.

Syntax:

```
SELECT column1, column2, ... FROM table_name WHERE condition;
```

- column1, column2, ... → Specifies the columns to retrieve.
- table_name → The table from which data is fetched.
- WHERE condition → Specifies filtering conditions.

Example

Let's assume we have a "students" table with the following structure:

```
CREATE TABLE students (
    id UUID PRIMARY KEY,
    name TEXT,
    age INT,
    department TEXT
);
```

1. Fetching All Records

To retrieve all rows and columns from the students table:

```
SELECT * FROM students;
```

2. Fetching Specific Columns

If we only want to retrieve the name and age columns:

```
SELECT name, age FROM students;
```

3. Filtering Data Using WHERE Clause

Cassandra uses a distributed architecture, which means data is stored across multiple nodes. To efficiently retrieve data, Cassandra requires queries to use the partition key in the WHERE clause. Unlike traditional SQL databases, Cassandra does not support arbitrary filtering unless a secondary index is created.

For example, to retrieve data for a specific student based on id:

```
SELECT * FROM students WHERE id = 123e4567-e89b-12d3-a456-426614174000;
```

In this query, id = 123e4567-e89b-12d3-a456-426614174000 represents a Universally Unique Identifier (UUID).

4.Using ALLOW FILTERING for Non-Indexed Columns

If you attempt to filter by a non-primary key column (e.g., department), Cassandra does not allow it by default:

```
SELECT * FROM students WHERE department = 'Computer Science';
```

This query will fail unless department is an indexed column. To force execution, use ALLOW FILTERING.

```
SELECT * FROM students WHERE department = 'Computer Science' ALLOW
FILTERING;
```

5. Retrieving Data with LIMIT

```
SELECT * FROM students LIMIT 5;
```

3.1.2.5 Updating Data

In Cassandra, the UPDATE statement is used to modify existing records in a table. Unlike traditional SQL databases, Cassandra does not perform row-level locking or transactions by default. Instead, it writes a new version of the data to ensure high availability and scalability.

```
UPDATE users SET age = 26 WHERE id = <UUID>;
```

This query updates the age column to 26 for the user with the given UUID. The WHERE clause identifies which row to update (must include the partition key).

3.1.2.6 Deleting Data

In Cassandra, we can remove data using the DELETE statement or clear an entire table using TRUNCATE.

1. Deleting a Specific Record

We can remove a particular row from a table by specifying the partition key in the WHERE clause.

Syntax:

```
DELETE FROM users WHERE id = <UUID>;
```

This query Deletes the user record where id = <UUID>. The WHERE condition must include the partition key (id)

Example

```
CREATE TABLE users (
    id UUID PRIMARY KEY,  -- Partition Key
    name TEXT,
    age INT,
    email TEXT
);
```

```
DELETE FROM users WHERE id = 123e4567-e89b-12d3-a456-426614174000;
```

2. Deleting Specific Column Data

Instead of deleting the entire row, you can delete only specific columns from a record.

Example: Remove Only the email Column

```
DELETE email FROM users WHERE id = 123e4567-e89b-12d3-a456-426614174000;
```

The email column is removed, but other details (like name and age) remain.

3. Deleting All Data from a Table

To remove all rows from a table, use the TRUNCATE command.

```
TRUNCATE users;
```

This query will deletes all records from the users table.

3.1.2.7 Dropping a Table or Keyspace

Cassandra allows you to permanently delete a table or an entire keyspace using the DROP command.

1. Dropping a Table

To delete a table (and all its data), use the DROP TABLE command.

Syntax:

```
DROP TABLE table_name;
```

Example: Deleting a users Table

```
DROP TABLE users;
```

2. Dropping a Keyspace

A keyspace is like a database in Cassandra. Dropping a keyspace removes all tables inside it.

Syntax:

```
DROP KEYSPACE keyspace_name;
```

Example: Deleting a Keyspace

```
DROP KEYSPACE my_keyspace;
```

This command removes the entire keyspace, along with all its tables and data.

Sample Experiment Question

You are given a task to manage an employee database using Cassandra. Perform the following operations step by step:

1. Create a keyspace named company with a replication factor of 2.
2. Use the keyspace for subsequent operations.
3. Create a table employees with the following fields:
   - emp_id (UUID, Primary Key)
   - name (TEXT)
   - age (INT)
   - position (TEXT)
   - salary (DECIMAL)
4. Insert three employee records into the table.
5. Retrieve all records from the employees table.
6. Update the salary of an employee based on their emp_id.
7. Delete an employee record based on their emp_id.
8. Drop the employees table after all operations.

Step 1: Create a Keyspace

```
CREATE KEYSPACE company

WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 2};
```

Output

```
Keyspace company created successfully.
```

Step 2: Use the Keyspace

```
USE company;
```

Output

```
Now using keyspace company.
```

Step 3: Create the employees Table

```
CREATE TABLE employees (

    emp_id UUID PRIMARY KEY,

    name TEXT,

    age INT,

    position TEXT,

    salary DECIMAL

);
```

Output

```
Table employees created successfully.
```

Step 4: Insert Three Employee Records

```
INSERT INTO employees (emp_id, name, age, position, salary)

VALUES (uuid(), 'John ', 30, 'Software Engineer', 75000.00);


INSERT INTO employees (emp_id, name, age, position, salary)

VALUES (uuid(), 'Jane ', 28, 'Data Scientist', 80000.00);


INSERT INTO employees (emp_id, name, age, position, salary)

VALUES (uuid(), 'Michael ', 35, 'Project Manager', 95000.00);
```

Output

```
Records inserted successfully.
```

Step 5: Retrieve All Employee Records

```
SELECT * FROM employees;
```

Output

Table 3.1.1 Output of Retrieve command

emp_id	name	age	position	salary
550e8400-e29b-41d4-a716-446655440000	John	30	Software Engineer	75000.00
660e1234-f19c-44d4-b656-336622778999	Jane	28	Data Scientist	80000.00
770e4321-b45a-41d4-b716-778899112233	Michael	35	Project Manager	95000.00

Step 6: Update an Employee's Salary

Assume the employee ID is 550e8400-e29b-41d4-a716-446655440000

```
UPDATE employees

SET salary = 85000.00

WHERE emp_id = 550e8400-e29b-41d4-a716-446655440000;
```

Output

```
Salary updated successfully.
```

John's salary is now 85,000.00.

Step 7: Delete an Employee Record

```
DELETE FROM employees

WHERE emp_id = 550e8400-e29b-41d4-a716-446655440000;
```

Output

```
Record deleted successfully.
```

Step 8: Drop the Table

```
DROP TABLE employees;
```

Output

```
Table employees dropped successfully.
```

Lab Practice Questions

Question 1

Create a keyspace named company_db with SimpleStrategy and a replication factor of 3. Inside company_db, create a table employees with the following columns:

- emp_id (UUID, Primary Key)
- name (TEXT)
- age (INT)
- designation (TEXT)
- salary (DECIMAL)
- dept_id(UUID)

1. Insert three employee records into the employees table.
2. Retrieve all employee records from the table.
3. Fetch details of an employee using emp_id.
4. Update the salary of an employee based on their emp_id.
5. Delete an employee record using their emp_id.
6. Remove all records from the employees table without deleting its structure.
7. Drop the employees table.
8. Drop the entire company_db keyspace.

Question 2

Create a keyspace named organization with SimpleStrategy and a replication factor of 2. Create a table departments with the following fields:

- dept_id (UUID, Primary Key)

- dept_name (TEXT)
- location (TEXT)

1. Insert at least three department records.
2. Retrieve all department records from the table.
3. Fetch a department's details using its dept_id.
4. Update the location of a department using dept_id.
5. Delete a specific department record using dept_id.
6. Remove all records from the departments table while keeping the structure.
7. Drop the departments table.
8. Drop the organization keyspace.

EXPERIMENT - 2
MongoDB

Objectives

- Familiarize the MongoDB platform
- Installing MongoDB
- Apply basic commands in MongoDB

Theory

MongoDB is a database that helps store and manage data in a simple and flexible way. Instead of using tables like a traditional database (such as MySQL), MongoDB stores data in documents. Example: Instead of a table for students, MongoDB stores student records as documents inside a "Students" collection.

Example of a MongoDB document :

```
{
  "name": "Aarav",
  "age": 13,
  "class": "8th",
  "subjects": ["Math", "Science", "English"]
}
```

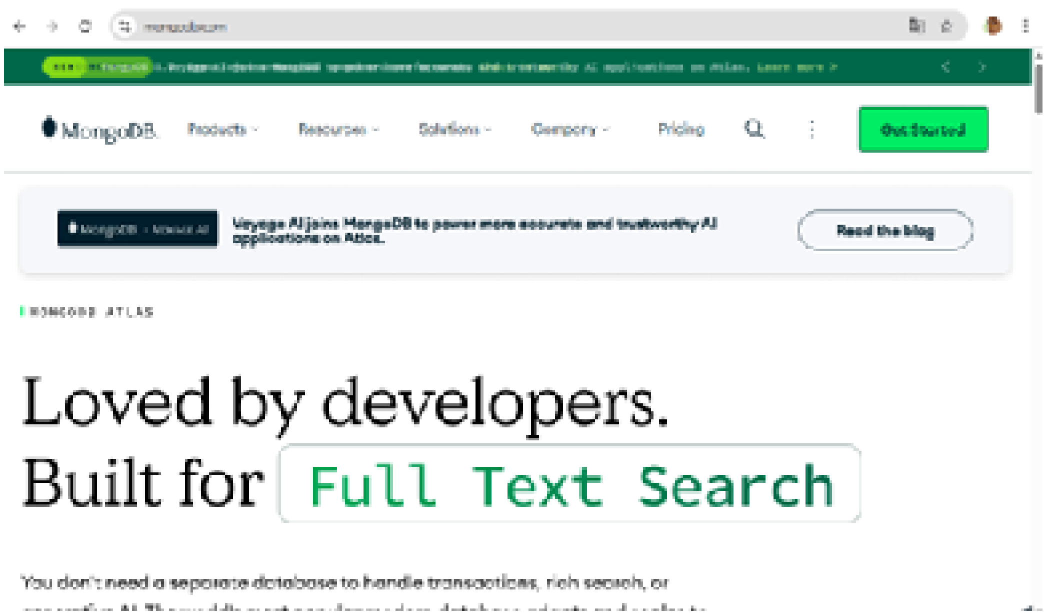
Each document is like a record but more flexible than a table row.

Unlike SQL databases, MongoDB does not require a fixed structure. Different documents in the same collection can have different fields. MongoDB can handle large amounts of data quickly. It is great for storing social media posts, chat messages, and real-time data. MongoDB can be used with Python, JavaScript, Java, C++, and many more! Popular for web applications, mobile apps, and IoT projects.

3.2.1 Installing MongoDB

1. Search mongoDB in google. Click on the official website of MongoDB

Below given is the home page of the official website of MongoDB


Products Resources Solutions Company Pricing Get Started

Voyage AI joins MongoDB to power more accurate and trustworthy AI applications on Atlas.

Read the blog

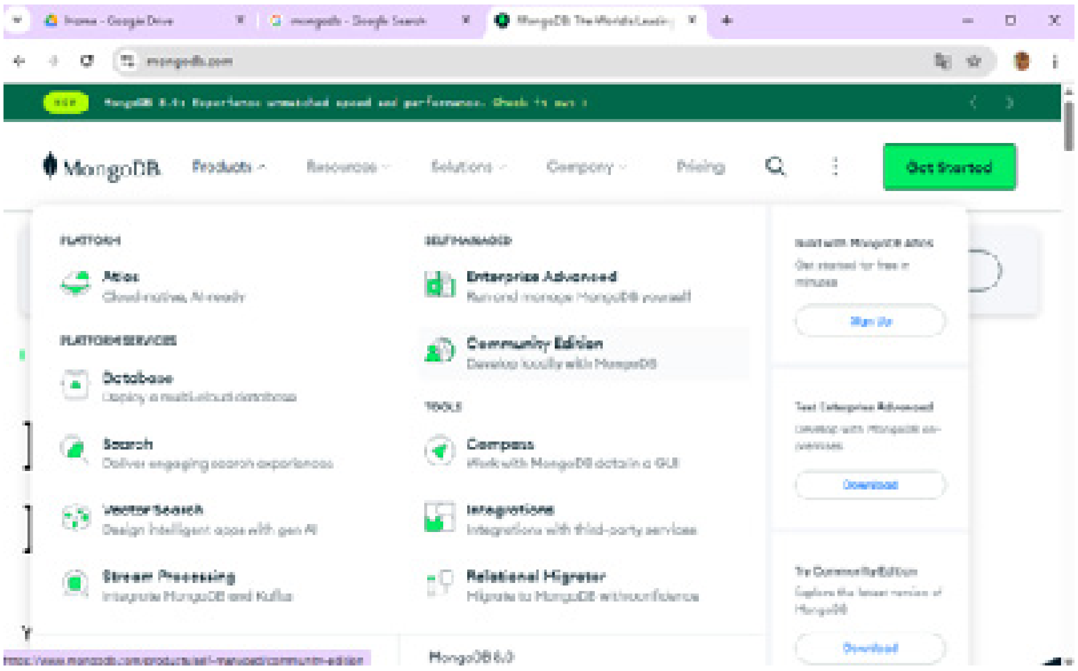
MONGODB ATLAS

Loved by developers.
Built for Full Text Search

You don't need a separate database to handle transactions, rich search, or

Fig. 3.2.1 Home page of the official website of MongoDB

2. Click on "products" → Community Edition. See Fig.3.2.2


Products Resources Solutions Company Pricing Get Started

PLATFORM

Atlas
Cloud-native, AI-ready

PLATFORM SERVICES

Database
Deploy a multi-cloud database

Search
Deliver engaging search experiences

Vector Search
Design intelligent apps with gen AI

Stream Processing
Integrate MongoDB and Kafka

SELF MANAGED

Enterprise Advanced
Run and manage MongoDB yourself

Community Edition
Develop locally with MongoDB

TOOLS

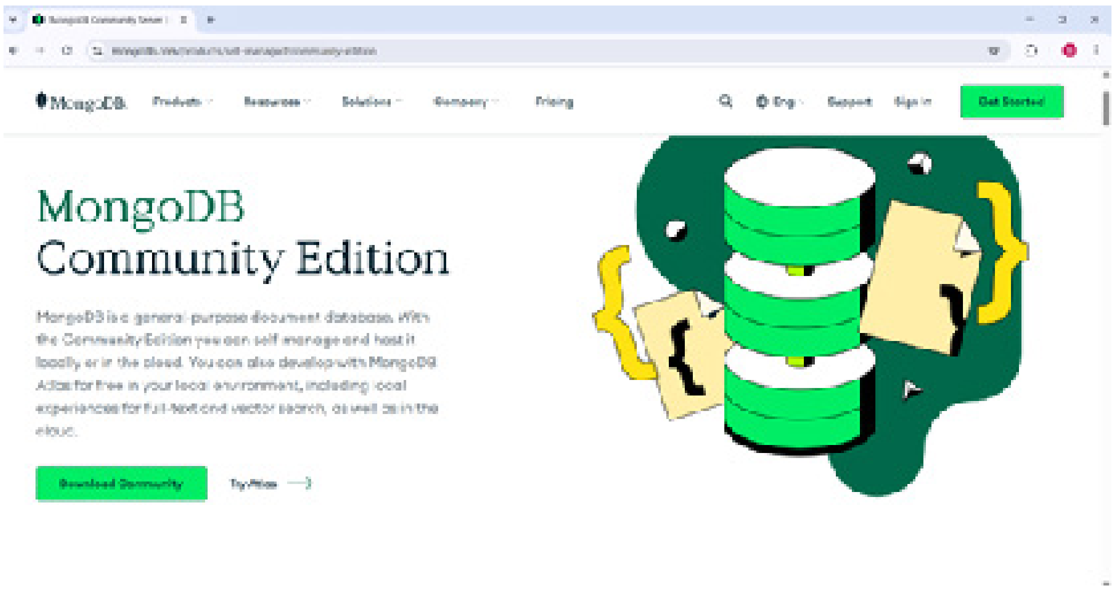
Compass
Work with MongoDB data in a GUI

Integrations
Integrations with third-party services

Relational Migrator
Migrate to MongoDB with confidence

Fig. 3.2.2 Community edition

3. Click on “Download Community”.

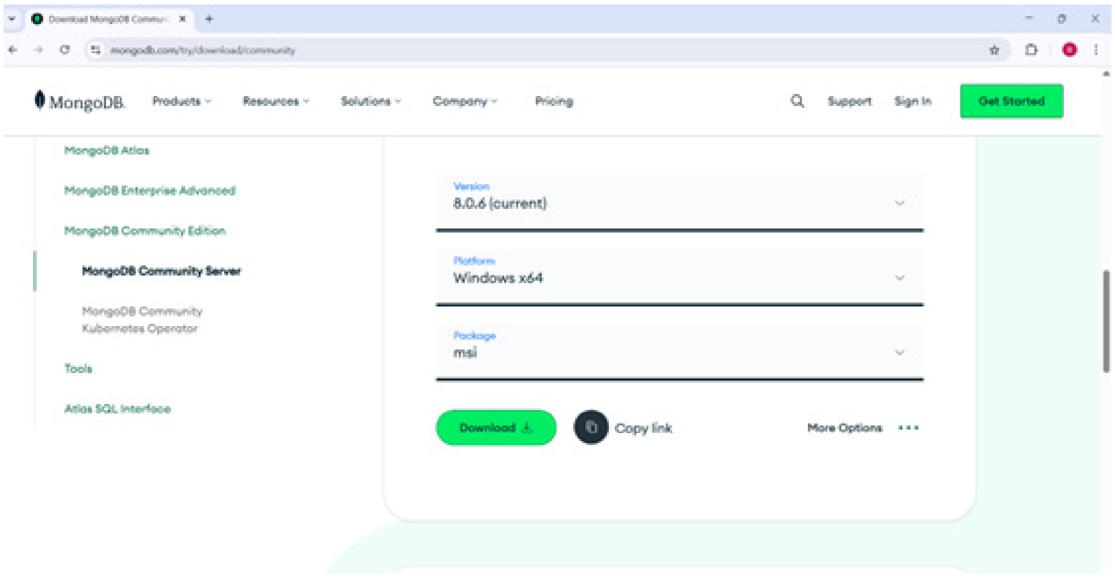


MongoDB Community Edition

MongoDB is a general purpose document database. With the Community Edition you can self manage and host it locally or in the cloud. You can also develop with MongoDB Atlas for free in your local environment, including local experiences for full-text and vector search, as well as in the cloud.

Fig. 3.2.3 Download Community

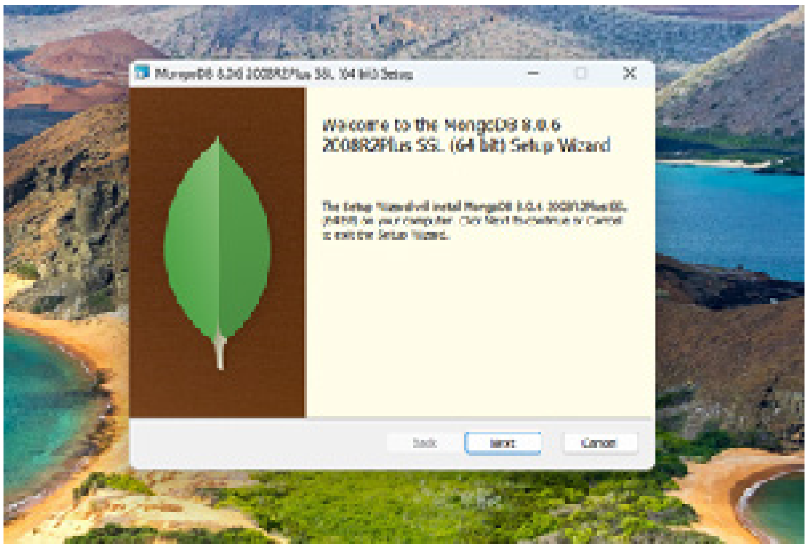
4. Select package msi and click download.



Version: 8.0.6 (current)
Platform: Windows x64
Package: msi

Fig. 3.2.4 Version, Platform and Package

5. Right click and Open the downloaded file.



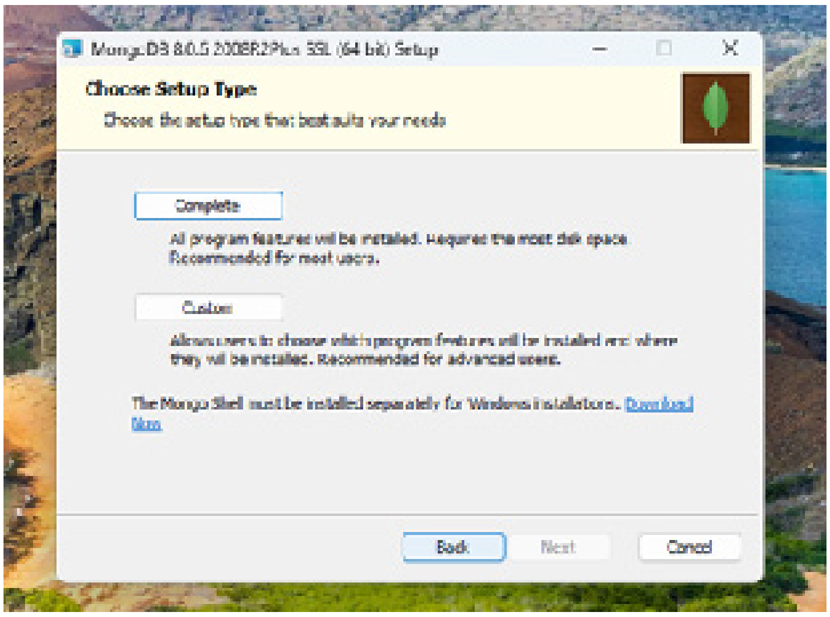
MongoDB 8.0.6 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 8.0.6 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 8.0.6 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Fig. 3.2.5 MongoDB setup wizard

6. Click next



MongoDB 8.0.5 2008R2Plus SSL (64 bit) Setup

Choose Setup Type

Choose the setup type that best suits your needs

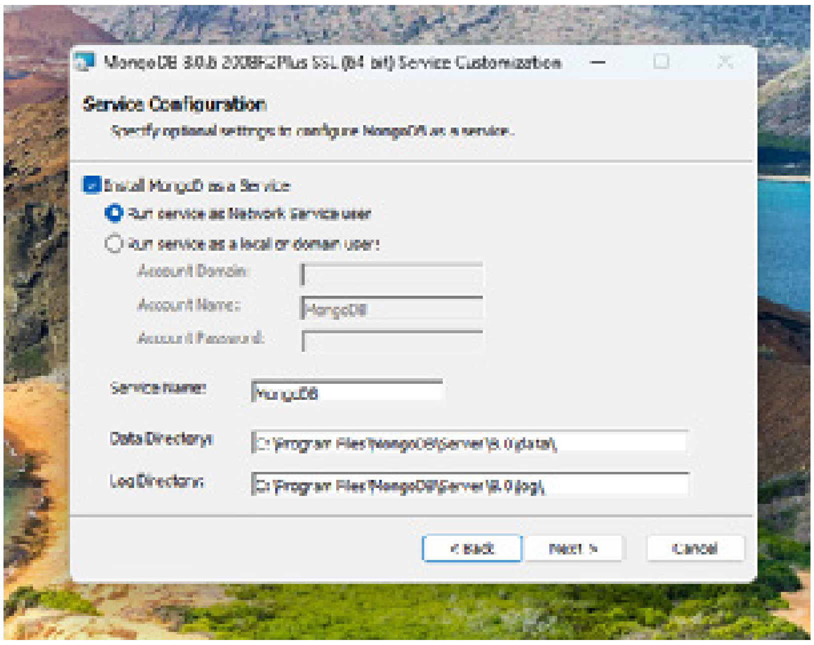
All program features will be installed. Requires the most disk space. Recommended for most users.

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 Here

Fig. 3.2.6 Choose setup type

7. Select Complete and click next



MongoDB 8.0.6 2008R2Plus SSL (64 bit) Service Customization

Service Configuration

Specify optional settings to configure MongoDB as a service.

[✓] Install MongoDB as a Service

[●] Run service as Network Service user

[ ] Run service as a local or domain user:

Account Domain:

Account Name: MongoDB

Account Password:

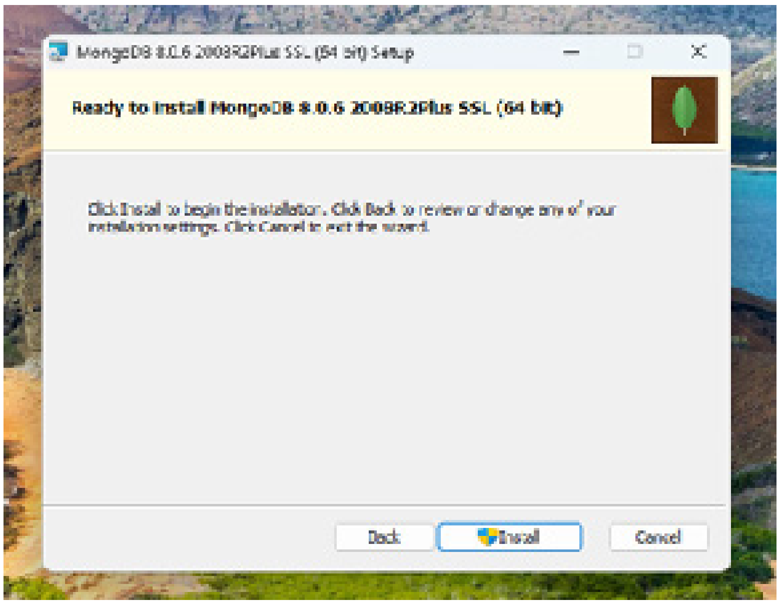
Service Name: MongoDB

Data Directory: C:\Program Files\MongoDB\Server\8.0\data\

Log Directory: C:\Program Files\MongoDB\Server\8.0\log\

Fig. 3.2.7 Service Configuration

8. Finally Install



MongoDB 8.0.6 2008R2Plus SSL (64 bit) Setup

Ready to install MongoDB 8.0.6 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.

Fig. 3.2.8 Ready to install

3.2.2 Set Environment variables

1. Open the Control Panel, navigate to System, then Advanced system settings.

2. Click "Environment Variables" and choose "System Variables".

3. Double click “Path”

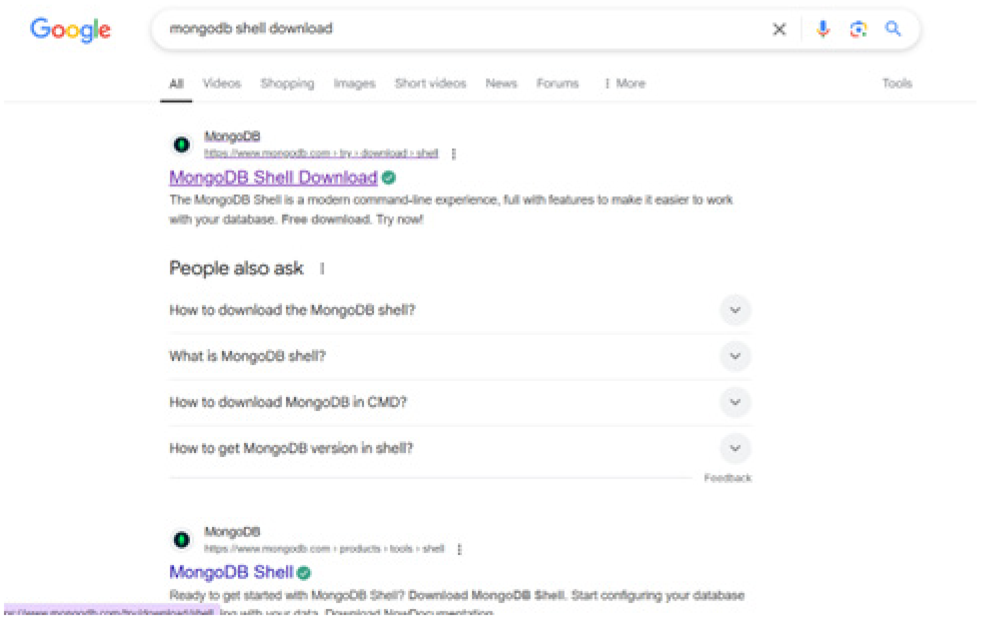
4. Click "New" to add a new variable or "Edit" to modify an existing one.

5. Enter the variable name (ie, the URL of the path) and click OK.

6. Restart your terminal or command prompt for the changes to take effect.

3.2.3 Install mongodb Shell

1. Search mongodb shell download. Click the first link.



mongodb shell download

MongoDB
https://www.mongodb.com › try › download › shell

MongoDB Shell Download

The MongoDB Shell is a modern command-line experience, full with features to make it easier to work with your database. Free download. Try now!

People also ask

How to download the MongoDB shell?

What is MongoDB shell?

How to download MongoDB in CMD?

How to get MongoDB version in shell?

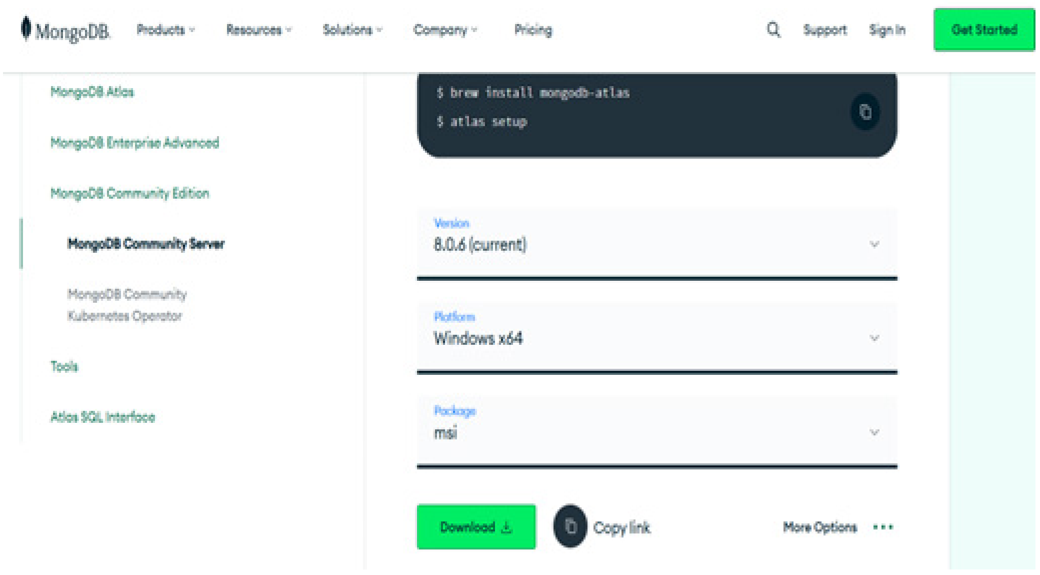
MongoDB
https://www.mongodb.com › products › tools › shell

MongoDB Shell

Ready to get started with MongoDB Shell? Download MongoDB Shell. Start configuring your database

Fig. 3.2.9 Link for MongoDB Shell download

2. Select package “msi” and download.



```
$ brew install mongodb-atlas
$ atlas setup
```

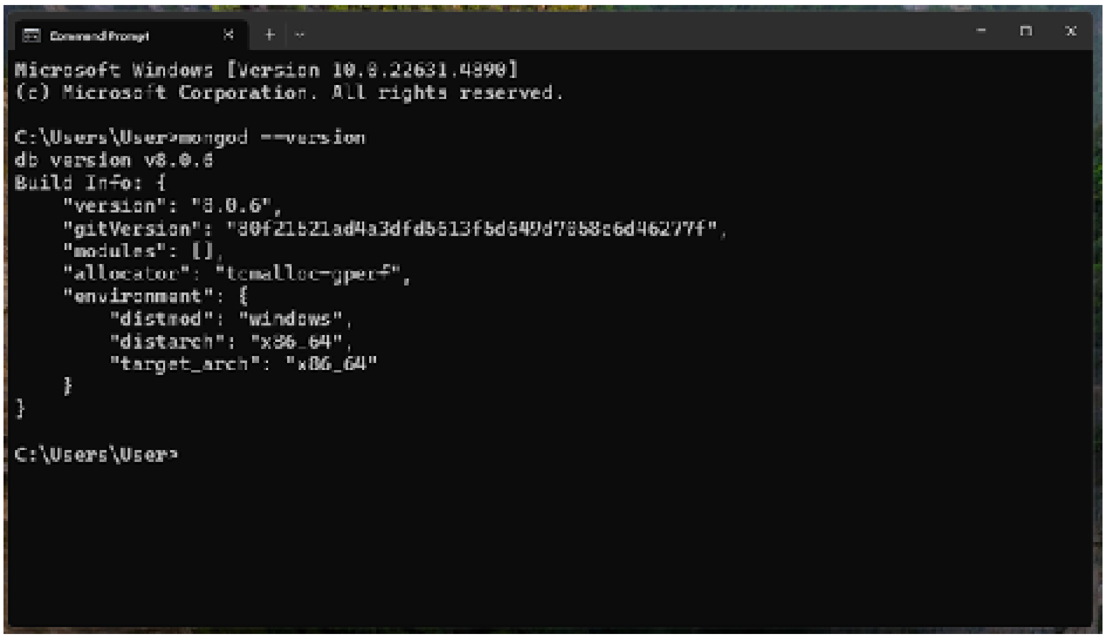
Version
8.0.6 (current)

Platform
Windows x64

Package
msi

Fig. 3.2.10 Download option

3. Go to command prompt and type mongod -- version. You will get the details of the MongoDB version installed in your computer as shown in the figure Fig.3.2.11.



```
Microsoft Windows [Version 10.0.22631.4890]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>mongod --version
db version v8.0.6
Build Info: {
    "version": "8.0.6",
    "gitVersion": "80f21521ad4a3dfd5613f5d649d7058c6d46277f",
    "modules": [],
    "allocator": "tcmalloc-gperf",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}

C:\Users\User>
```

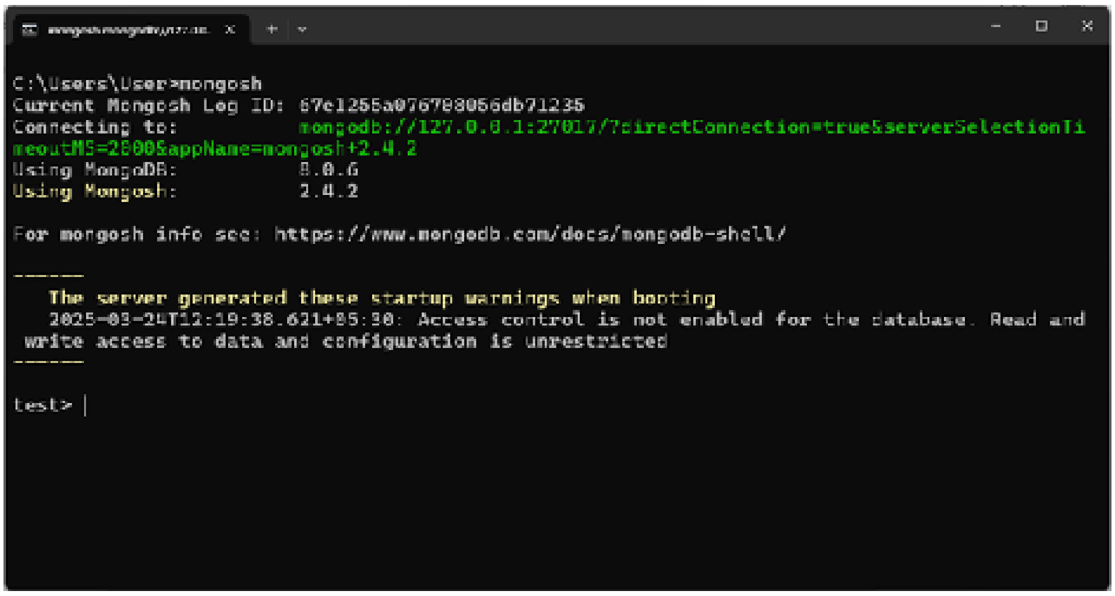
Fig. 3.2.11 Details of MongoDB version installed

Note : To install mongodb in Ubuntu, refer to the address https://www.youtube.com/watch?v=SNgaUYu5o1oHow to Install MongoDB 8 on Ubuntu 24.04 LTS Linux

3.2.4 How to Test MongoDB After Installation?

To Start Mongodb,

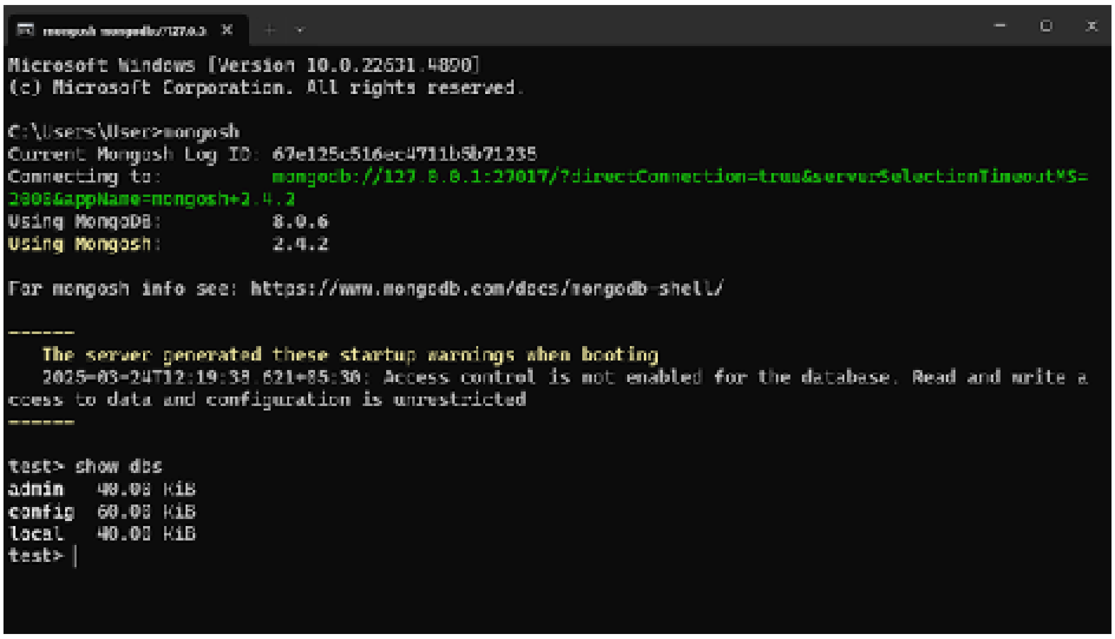
Go to Command prompt→ Type mongosh → press enter . See the figure



```
C:\Users\User>mongosh
Current Mongosh Log ID: 67e1255a0767080 56db71235
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTi
meoutMS=2000&appName=mongosh+2.4.2
Using MongoDB:          8.0.6
Using Mongosh:          2.4.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-03-24T12:19:38.621+05:30: Access control is not enabled for the database. Read and
 write access to data and configuration is unrestricted
------

test> |
```

Fig. 3.2.12 Starting MongoDB

To see the default databases, type show dbs , the default databases can be seen as shown in the figure.



```
Microsoft Windows [Version 10.0.22631.4890]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>mongosh
Current Mongosh Log ID: 67e125c516ec4711b5b71235
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=
2000&appName=mongosh+2.4.2
Using MongoDB:          8.0.6
Using Mongosh:          2.4.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-03-24T12:19:38.621+05:30: Access control is not enabled for the database. Read and write a
ccess to data and configuration is unrestricted
------

test> show dbs
admin   40.00 KiB
config  60.00 KiB
local   40.00 KiB
test> |
```

Fig. 3.2.13 Showing default databases

3.2.5 Basic Commands in MongoDB

These are fundamental commands used for database management, collection operations, document manipulation, querying, updating, and deleting data.

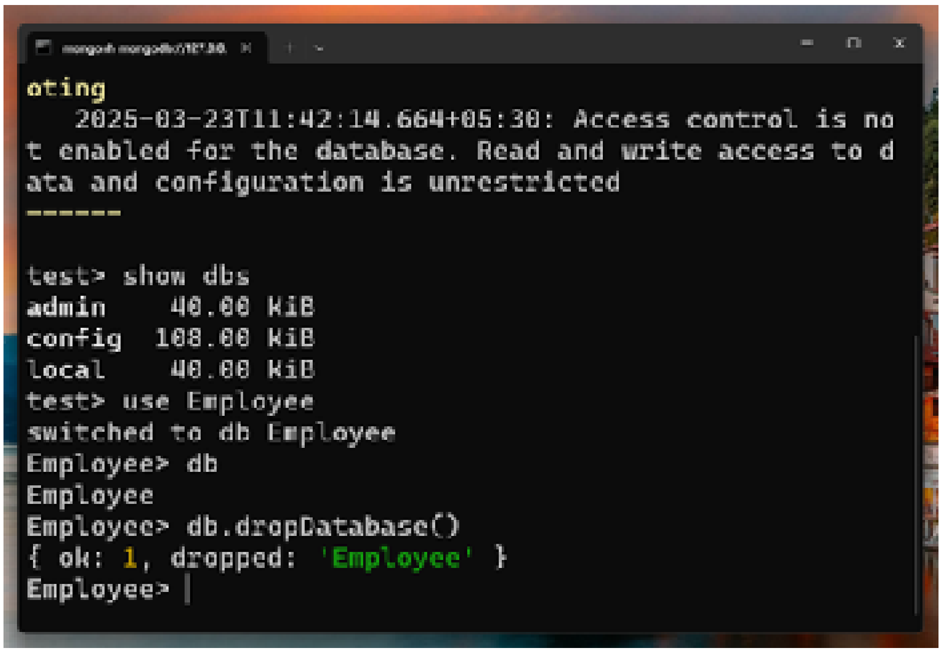
A. DATABASE COMMANDS

These commands help **create, select, and manage databases** in MongoDB.

Table 3.2.1 Database Commands

Command	Description	Syntax
show dbs	Displays all databases	show dbs
use database_name	Switch to or create a database	use Employee
db	Show the current active databases	db
db.dropDatabase( )	Deletes the current database	db.dropDatabase

The following figure will show you the outputs of all commands discussed above



```
oting
   2025-03-23T11:42:14.664+05:30: Access control is no
t enabled for the database. Read and write access to d
ata and configuration is unrestricted
------

test> show dbs
admin     40.00 KiB
config   108.00 KiB
local     40.00 KiB
test> use Employee
switched to db Employee
Employee> db
Employee
Employee> db.dropDatabase()
{ ok: 1, dropped: 'Employee' }
Employee> 
```

Fig. 3.2.14 Illustration of Database commands

B. COLLECTION COMMANDS

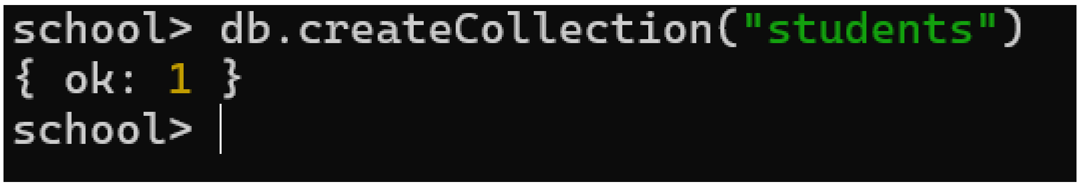
Collections in MongoDB are like tables in relational databases. These commands allow you to create and manage collections.

Table. 3.2.2 Collection Commands

Command	Description	Syntax
`show collections`	Lists all collections in the database	`show collections`
`db.createCollection("collection_name")`	Creates a new collection	`db.createCollection("students")`
`db.collection.drop()`	Deletes a collection	`db.students.drop()`

Examples

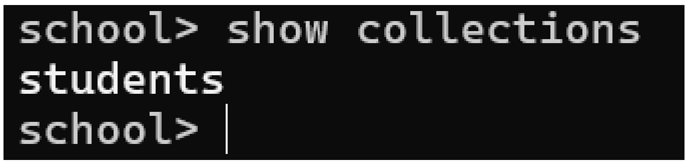
1. db.createCollection( )



```
school> db.createCollection("students")
{ ok: 1 }
school> 
```

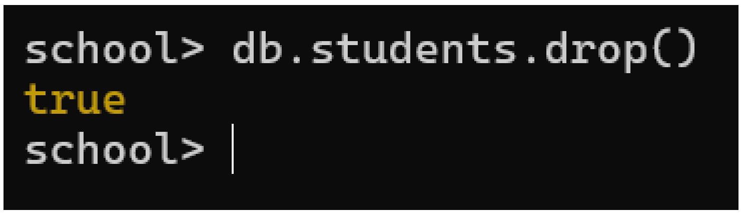
This command creates a collection named "students".

2. show collections



```
school> show collections
students
school> |
```

3. db.collection.drop( )



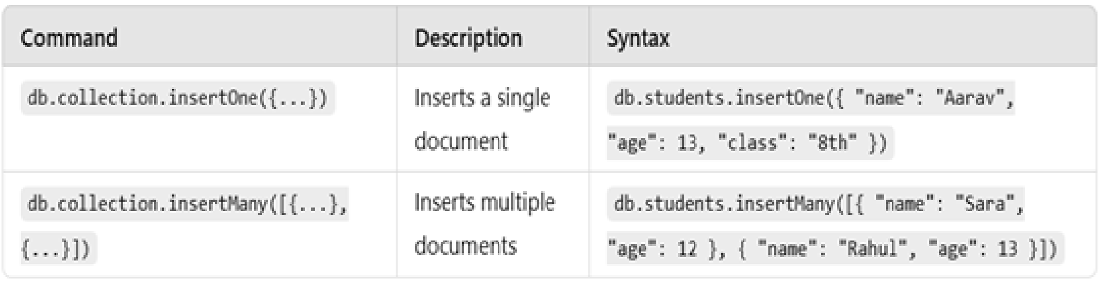
```
school> db.students.drop()
true
school> |
```

True indicates that the collection was successfully dropped.

C. DOCUMENT COMMANDS

Documents in MongoDB are like **rows** in relational databases. These commands help **insert data** into collections.

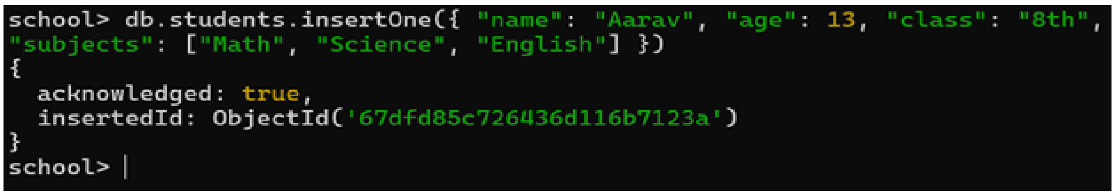
Table 3.2.3 Document Commands



Command	Description	Syntax
`db.collection.insertOne({...})`	Inserts a single document	`db.students.insertOne({ "name": "Aarav", "age": 13, "class": "8th" })`
`db.collection.insertMany([{...}, {...}])`	Inserts multiple documents	`db.students.insertMany([{ "name": "Sara", "age": 12 }, { "name": "Rahul", "age": 13 }])`

Examples

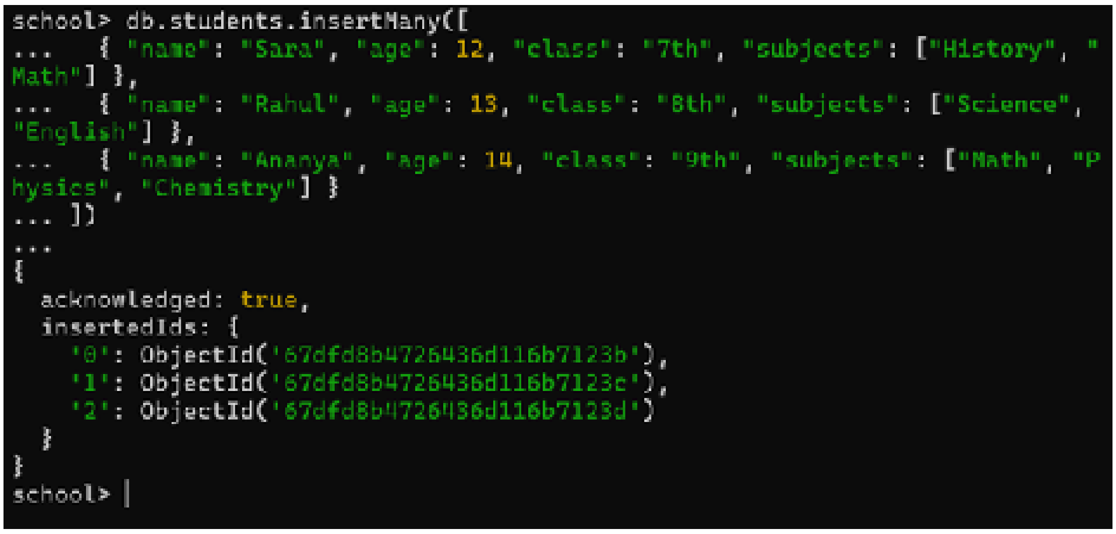
1. insertOne( )



```
school> db.students.insertOne({ "name": "Aarav", "age": 13, "class": "8th",
"subjects": ["Math", "Science", "English"] })
{
  acknowledged: true,
  insertedId: ObjectId('67dfd85c726436d116b7123a')
}
school> |
```

This inserts **one student record** into the `students` collection. `insertedId` is automatically generated.

2. insertMany( )



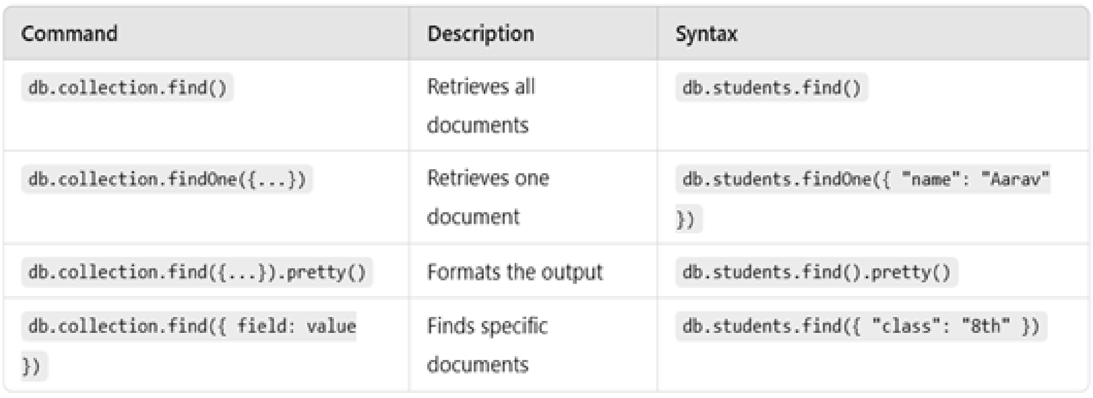
```
school> db.students.insertMany([
...   { "name": "Sara", "age": 12, "class": "7th", "subjects": ["History", "
Math"] },
...   { "name": "Rahul", "age": 13, "class": "8th", "subjects": ["Science",
"English"] },
...   { "name": "Ananya", "age": 14, "class": "9th", "subjects": ["Math", "P
hysics", "Chemistry"] }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67dfd8b4726436d116b7123b'),
    '1': ObjectId('67dfd8b4726436d116b7123c'),
    '2': ObjectId('67dfd8b4726436d116b7123d')
  }
}
school> |
```

This inserts three student records into the `students` collection. "`insertedIds`" contains the unique IDs assigned to each document.

D. QUERY COMMANDS

Query commands help **search and retrieve documents** from a collection.

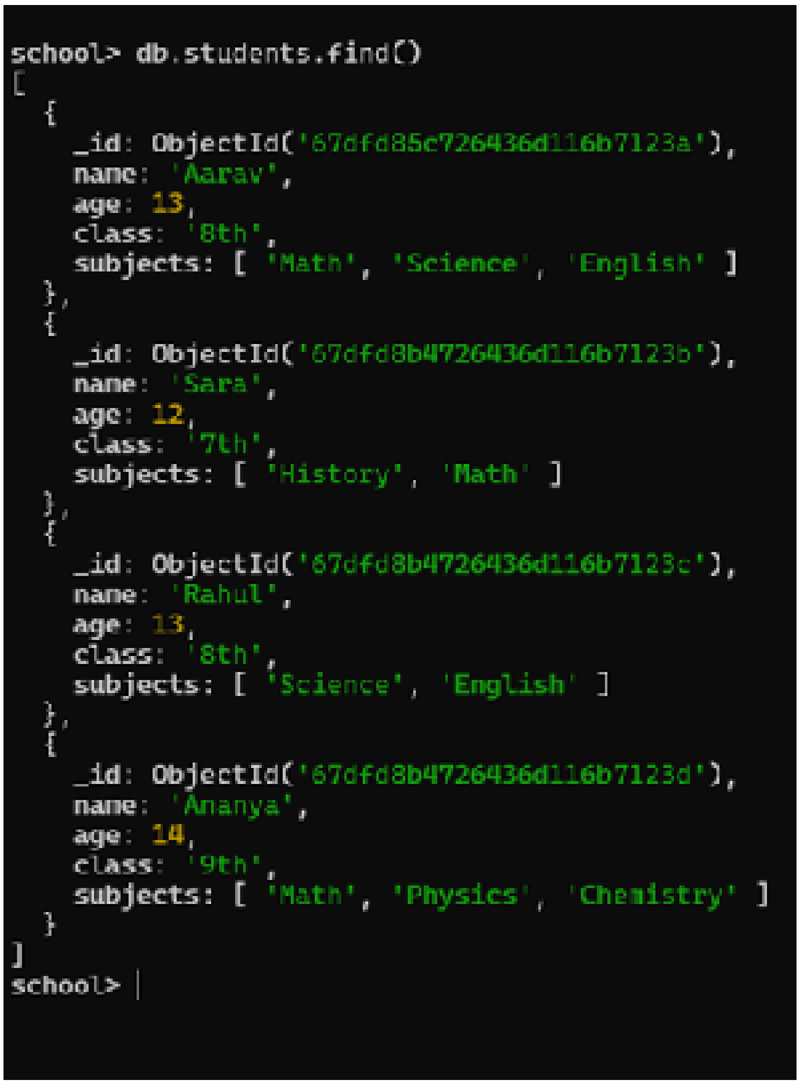
Table 3.2.4. Query Commands



Command	Description	Syntax
`db.collection.find()`	Retrieves all documents	`db.students.find()`
`db.collection.findOne({...})`	Retrieves one document	`db.students.findOne({ "name": "Aarav" })`
`db.collection.find({...}).pretty()`	Formats the output	`db.students.find().pretty()`
`db.collection.find({ field: value })`	Finds specific documents	`db.students.find({ "class": "8th" })`

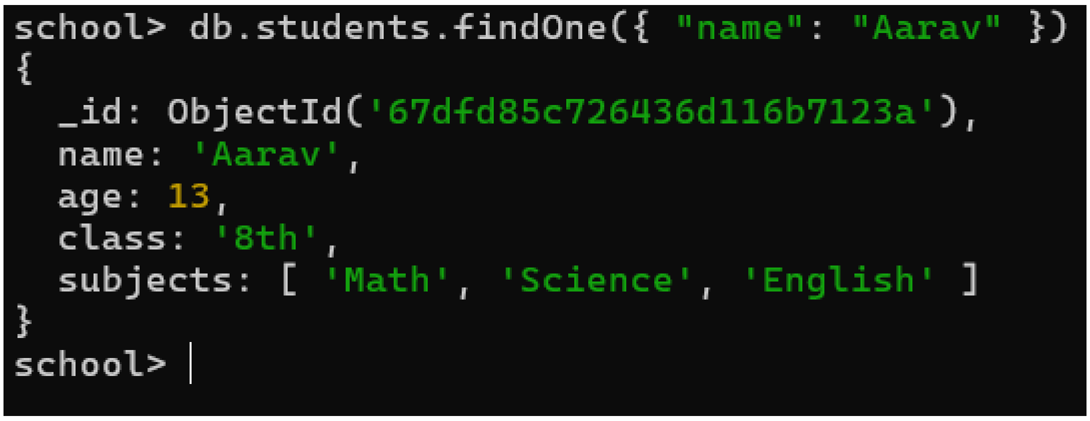
Examples

1. find( )



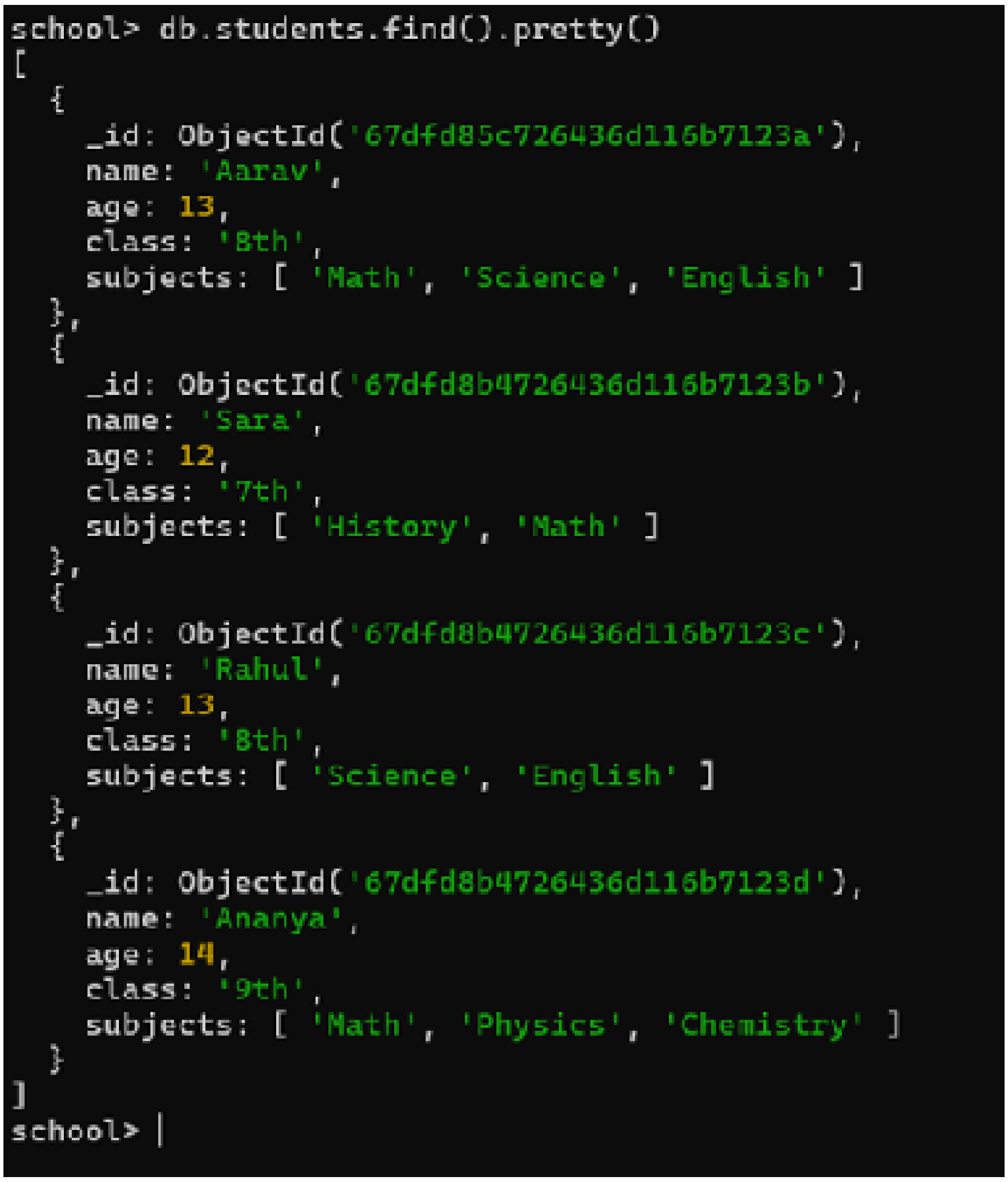
```
school> db.students.find()
[
  {
    _id: ObjectId('67dfd85c726436d116b7123a'),
    name: 'Aarav',
    age: 13,
    class: '8th',
    subjects: [ 'Math', 'Science', 'English' ]
  },
  {
    _id: ObjectId('67dfd8b4726436d116b7123b'),
    name: 'Sara',
    age: 12,
    class: '7th',
    subjects: [ 'History', 'Math' ]
  },
  {
    _id: ObjectId('67dfd8b4726436d116b7123c'),
    name: 'Rahul',
    age: 13,
    class: '8th',
    subjects: [ 'Science', 'English' ]
  },
  {
    _id: ObjectId('67dfd8b4726436d116b7123d'),
    name: 'Ananya',
    age: 14,
    class: '9th',
    subjects: [ 'Math', 'Physics', 'Chemistry' ]
  }
]
school> 
```

2. findOne( )



```
school> db.students.findOne({ "name": "Aarav" })
{
  _id: ObjectId('67dfd85c726436d116b7123a'),
  name: 'Aarav',
  age: 13,
  class: '8th',
  subjects: [ 'Math', 'Science', 'English' ]
}
school> 
```

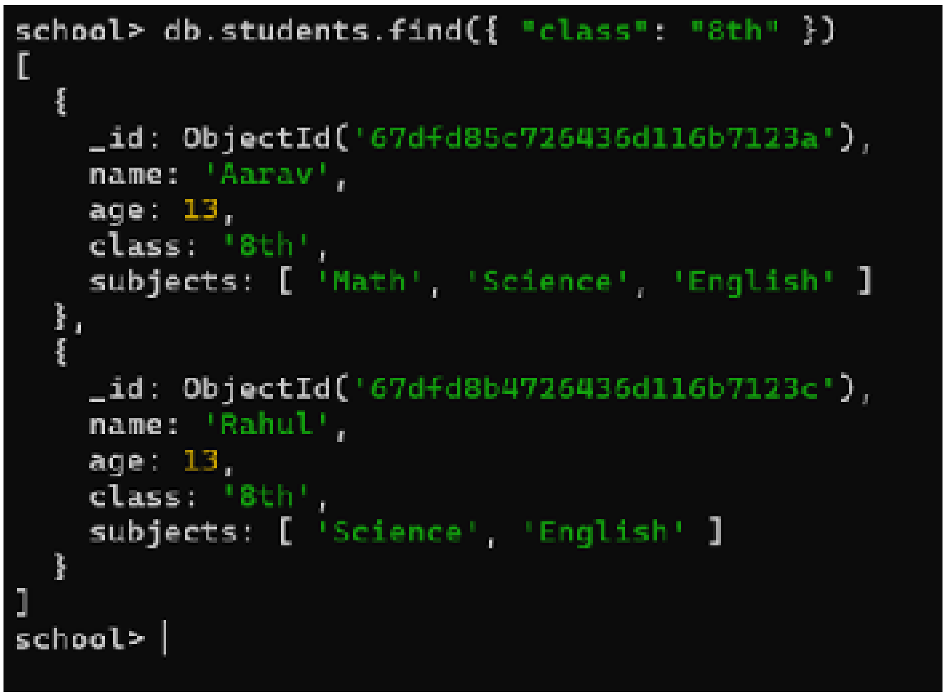
3. find( ).pretty( )



```
school> db.students.find().pretty()
[
  {
    _id: ObjectId('67dfd85c726436d116b7123a'),
    name: 'Aarav',
    age: 13,
    class: '8th',
    subjects: [ 'Math', 'Science', 'English' ]
  },
  {
    _id: ObjectId('67dfd8b4726436d116b7123b'),
    name: 'Sara',
    age: 12,
    class: '7th',
    subjects: [ 'History', 'Math' ]
  },
  {
    _id: ObjectId('67dfd8b4726436d116b7123c'),
    name: 'Rahul',
    age: 13,
    class: '8th',
    subjects: [ 'Science', 'English' ]
  },
  {
    _id: ObjectId('67dfd8b4726436d116b7123d'),
    name: 'Ananya',
    age: 14,
    class: '9th',
    subjects: [ 'Math', 'Physics', 'Chemistry' ]
  }
]
school> |
```

- db.collection.find( ) → Retrieves all documents from the collection.
- .pretty( ) → Formats the output with proper indentation.

4. find({ field: value })



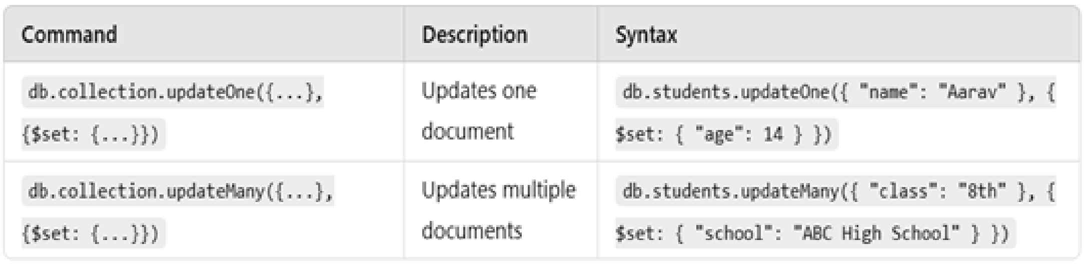
```
school> db.students.find({ "class": "8th" })
[
  {
    _id: ObjectId('67dfd85c726436d116b7123a'),
    name: 'Aarav',
    age: 13,
    class: '8th',
    subjects: [ 'Math', 'Science', 'English' ]
  },
  {
    _id: ObjectId('67dfd8b4726436d116b7123c'),
    name: 'Rahul',
    age: 13,
    class: '8th',
    subjects: [ 'Science', 'English' ]
  }
]
school> |
```

This filters results to show only students from class 8th.

E. UPDATE COMMANDS

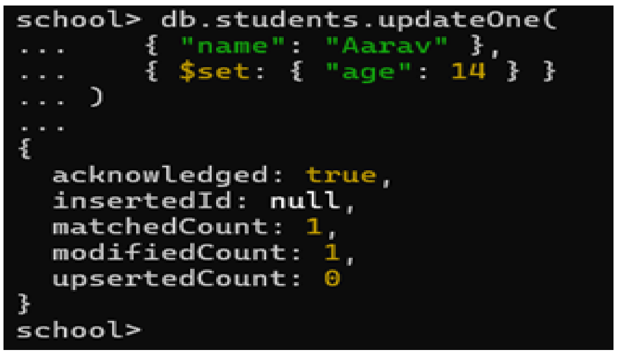
These commands modify existing documents in a collection.

Table 3.2.5 Update Commands



Command	Description	Syntax
`db.collection.updateOne({...}, {$set: {...}})`	Updates one document	`db.students.updateOne({ "name": "Aarav" }, { $set: { "age": 14 } })`
`db.collection.updateMany({...}, {$set: {...}})`	Updates multiple documents	`db.students.updateMany({ "class": "8th" }, { $set: { "school": "ABC High School" } })`

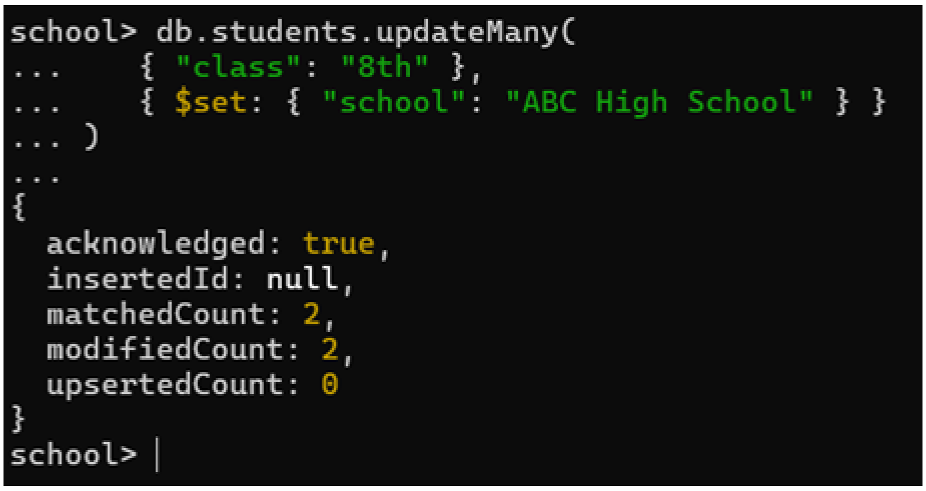
1. updateOne( )



```
school> db.students.updateOne(
...     { "name": "Aarav" },
...     { $set: { "age": 14 } }
... )
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
school>
```

- Updates Aarav's age to 14.
- Only the first matching document is updated.

2. updateMany( )



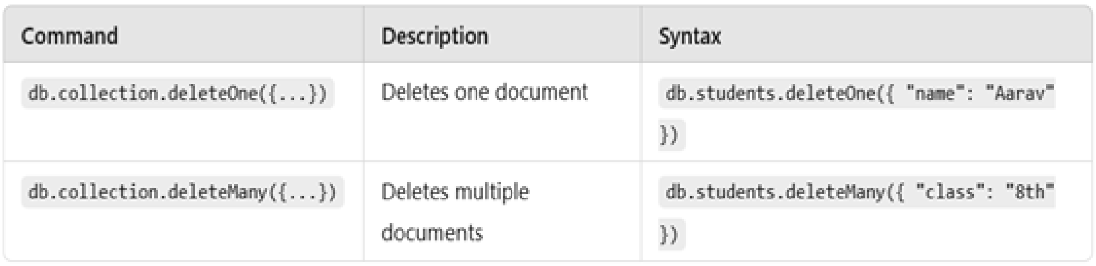
```
school> db.students.updateMany(
...     { "class": "8th" },
...     { $set: { "school": "ABC High School" } }
... )
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
school> |
```

- Updates the school field for all students in class 8th.
- "matchedCount": 2 → 2 documents matched the condition.
- "modifiedCount": 2 → 2 documents were updated.

F. DELETE COMMANDS

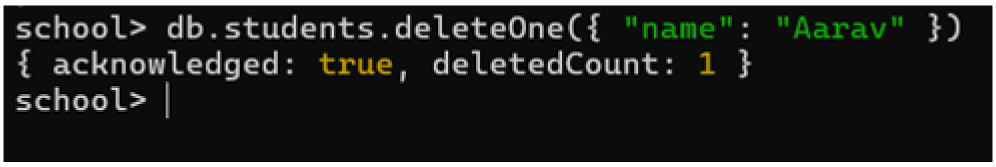
Used to remove unwanted documents from a collection.

Table 3.2.6 Delete Commands



Command	Description	Syntax
`db.collection.deleteOne({...})`	Deletes one document	`db.students.deleteOne({ "name": "Aarav" })`
`db.collection.deleteMany({...})`	Deletes multiple documents	`db.students.deleteMany({ "class": "8th" })`

1. deleteOne( )

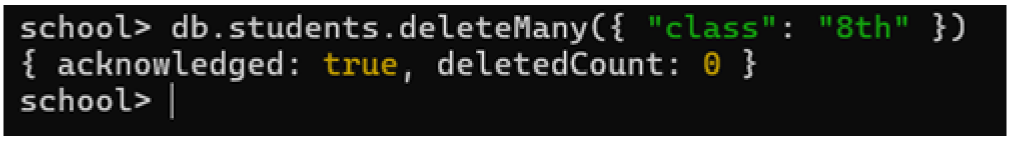


```
school> db.students.deleteOne({ "name": "Aarav" })
{ acknowledged: true, deletedCount: 1 }
school> |
```

- This command removes only the first matching document from the collection.
- Deletes only one document where "name": "Aarav".

- "deletedCount": 1 indicates that one document was deleted.

2. deleteMany( )



```
school> db.students.deleteMany({ "class": "8th" })
{ acknowledged: true, deletedCount: 0 }
school> |
```

- This command removes all documents that match a given condition.
- Deletes all students in class "8th".
- "deletedCount": 2 means two documents were deleted.

Sample Experiment questions

1. Assume we have a "**students**" **collection** with the following documents:

[

{ "_id": 1, "name": "Aarav", "age": 13, "class": "8th", "subjects": ["Math", "Science", "English"], "marks": { "Math": 85, "Science": 90 } },

{ "_id": 2, "name": "Sara", "age": 12, "class": "7th", "subjects": ["History", "Math"], "marks": { "Math": 78, "History": 88 } },

{ "_id": 3, "name": "Rahul", "age": 13, "class": "8th", "subjects": ["Science", "English"], "marks": { "Science": 82, "English": 79 } },

{ "_id": 4, "name": "Ananya", "age": 14, "class": "9th", "subjects": ["Math", "Physics", "Chemistry"], "marks": { "Math": 92, "Physics": 89 } },

{ "_id": 5, "name": "Kiran", "age": 12, "class": "7th", "subjects": ["English", "Geography"], "marks": { "English": 75, "Geography": 80 } }

]

Write the queries for the following questions:

1. Retrieve all students' details.
2. Find the student named "Aarav".

3. Find all students in class "8th".
4. Find all students who study Math.
5. Retrieve students who are older than 12.
6. Update Aarav's age to 14.
7. Add "school": "ABC High School" to all students in class "8th".
8. Remove the "subjects" field from Sara's record.
9. Increase Rahul's Science marks by 5.
10. Delete the student named "Kiran".
11. Delete all students in class "7th".
12. Remove all students from the collection but keep the collection structure.
13. Drop the entire students collection.

Answers

1. db.students.find().pretty()
2. db.students.findOne({ "name": "Aarav" })
3. db.students.find({ "class": "8th" })
4. db.students.find({ "subjects": "Math" })
5. db.students.find({ "age": { $gt: 12 } })
6. db.students.updateOne({ "name": "Aarav" }, { $set: { "age": 14 } })
7. db.students.updateMany({ "class": "8th" }, { $set: { "school": "ABC High School" } })
8. db.students.updateOne({ "name": "Sara" }, { $unset: { "subjects": "" } })
9. db.students.updateOne({ "name": "Rahul" }, { $inc: { "marks.Science": 5 } })
10. db.students.deleteOne({ "name": "Kiran" })
11. db.students.deleteMany({ "class": "7th" })
12. db.students.deleteMany({})
13. db.students.drop()

Lab Practice experiment

1. Create the following “employee” collection

```
[
  { "_id": 1, "name": "John Doe", "age": 30, "department": "HR", "salary": 50000, "skills": ["Communication", "Recruitment"] },
  { "_id": 2, "name": "Jane Smith", "age": 25, "department": "IT", "salary": 70000, "skills": ["Python", "Machine Learning"] },
  { "_id": 3, "name": "Robert Brown", "age": 35, "department": "Finance", "salary": 80000, "skills": ["Accounting", "Excel"] },
  { "_id": 4, "name": "Emily Davis", "age": 28, "department": "IT", "salary": 75000, "skills": ["Java", "Cybersecurity"] },
  { "_id": 5, "name": "Michael Wilson", "age": 40, "department": "Marketing", "salary": 60000, "skills": ["SEO", "Content Writing"] }
]
```

Write the queries for the following questions:

1. Retrieve all employees' details.
2. Find the employee named "Jane Smith".
3. Find all employees who work in the "IT" department.
4. Find all employees who have the skill "Python".
5. Retrieve employees who earn more than 60000.
6. Update "John Doe"'s salary to 55000.
7. Add a new skill "Data Analysis" to Robert Brown's skills.
8. Remove the "skills" field from Emily Davis's record.
9. Increase Michael Wilson's salary by 5000.
10. Delete the employee named "Robert Brown".
11. Delete all employees who work in the "HR" department.

12. Remove all employees except those in the "IT" department.

13. Drop the entire employee collection.

EXPERIMENT -3
Aggregation operations with MongoDB

Objective

- Define Aggregation in MongoDB
- List Aggregation Stages
- Familiarise Aggregation Syntax

Theory

MongoDB Aggregation is an advanced database feature that facilitates complex data processing and computation on collections of documents or records. It enables users to efficiently filter, group, and transform data to derive meaningful summaries.

The primary method for performing aggregation in MongoDB is through the aggregation pipeline, which functions like a multi-step data processing workflow. This pipeline processes data in stages, with each stage applying specific operations to modify or analyze the documents as they move through.

Using the aggregation pipeline, users can perform various operations such as filtering, grouping, sorting, restructuring, and executing calculations on the data.

MongoDB offers two primary methods for performing aggregation:

- Single-Purpose Aggregation
- Aggregation Pipeline

3.3.1 Single-Purpose Aggregation

Single-purpose aggregation methods handle basic queries, such as counting documents or retrieving unique field values.

1. count()

Returns the total number of documents in a collection.

Syntax:

```
db.collection.count(query)
```

Example: Count the total number of sales records in the `sales` collection.

```
db.sales.count({})
```

2. distinct()

Fetches unique values for a given field.

Syntax:

```
db.collection.distinct(field, query)
```

Example : Get all unique categories from the sales collection.

```
db.sales.distinct("category")
```

3.3.2 Aggregation Pipeline

The aggregation framework processes data through a pipeline consisting of multiple stages. Each stage transforms the documents before passing them to the next stage.

Syntax :

```
db.collection.aggregate([

  { stage1 },

  { stage2 },

  { stage3 }])
```

Example :Find the total sales per product category where sales are greater than $5000 and sort results in descending order.

```
db.sales.aggregate([

  { $match: { total: { $gt: 5000 } } },

  { $group: { _id: "$category", totalSales: { $sum: "$total" } } },

  { $sort: { totalSales: -1 } }

])
```

3.3.2.1 Main Aggregation Pipeline Operations

1. $match - Filtering Documents

Filters documents based on a condition, similar to the find() method.

Syntax :

```
{ $match: { field: value } }
```

Example :

Find all orders with status "delivered"

```
db.orders.aggregate([

{ $match: { status: "delivered" } }

])
```

2. $group - Grouping and Aggregation

Groups documents by a field and applies aggregation functions like sum, avg, count, etc.

Syntax:

```
{

 $group: {

   _id: "$fieldName",

   aggregateField: { $sum | $avg | $min | $max | $count: "$field" }

 }

}
```

Example:

Calculate total sales for each product category:

```
db.sales.aggregate([

  {

   $group: {

     _id: "$category",

     totalSales: { $sum: "$amount" }

   }

  }

])
```

3. $sort - Sorting Documents

Sorts documents in ascending (1) or descending (-1) order.

Syntax:

```
{ $sort: { field: 1 or -1 } }
```

Example:

Sort products by price in descending order:

```
db.products.aggregate([
  { $sort: { price: -1 } }
])
```

4. $project - Selecting and Transforming Fields

Includes, excludes, and creates computed fields.

Syntax:

```
{
  $project: {
    field1: 1,
    computedField: { $operation: ["$fieldA", "$fieldB"] }
  }
}
```

Example:

Show only product name and calculate total cost:

```
db.orders.aggregate([
  {
   $project: {
     productName: 1,
      totalCost: { $multiply: ["$price", "$quantity"] }
    }
  }
])
```

5. **$limit - Limiting Documents**

   Restricts the number of documents in the result.

Syntax:

```
{ $limit: number }
```

Example:

Get the top 5 most expensive products:

```
db.products.aggregate([
  { $sort: { price: -1 } },
  { $limit: 5 }
])
```

6. **$skip - Skipping Documents**

   Skips a specified number of documents.

Syntax:

```
{ $skip: number }
```

Example:

Skip the first 10 products and display the next:

```
db.products.aggregate([
  { $skip: 10 }
])
```

7. **$unwind - Expanding Arrays**

Breaks an array field into multiple documents.

Syntax:

```
{ $unwind: "$arrayField" }
```

Example:

Expand order items into separate documents:

```
db.orders.aggregate([
  { $unwind: "$items" }
])
```

8. **$lookup - Joining Collections**

Performs a left outer join with another collection.

Syntax:

```
{
  $lookup: {
    from: "foreignCollection",
    localField: "fieldA",
    foreignField: "fieldB",
    as: "newField"
  }
}
```

Example:

Join orders collection with customers:

```
db.orders.aggregate([
  {
    $lookup: {
      from: "customers",
      localField: "customerId",
      foreignField: "_id",
      as: "customerDetails"
    }
  }
])
```

9. $addFields - Adding New Fields

Adds new fields or modifies existing fields.

Syntax:

```
{ $addFields: { newField: value } }
```

Example:

Add a discount field to orders:

```
db.orders.aggregate([
  { $addFields: { discount: 10 } }
])
```

10. $count - Counting Documents

Counts the number of documents that match a query.

Syntax:

```
{ $count: "fieldName" }
```

Example:

Count total delivered orders:

```
db.orders.aggregate([
   { $match: { status: "delivered" } },
  { $count: "totalDeliveredOrders" }
])
```

11. $out - Writing Results to a Collection

Stores aggregation results into a new collection.

Syntax:

```
{ $out: "newCollectionName" }
```

Example:

Store all high-value orders into a new collection:

```
db.orders.aggregate([

  { $match: { amount: { $gt: 5000 } } },

  { $out: "highValueOrders" }

])
```

12. $merge - Merging Results into an Existing Collection

Merges results into an existing collection.

Syntax:

```
{

 $merge: {

   into: "collectionName",

   whenMatched: "merge | replace | keepExisting",

   whenNotMatched: "insert | discard"

 }

}
```

Example:

Merge sales data into salesSummary:

```
db.sales.aggregate([

  {

   $group: {

     _id: "$category",
      totalRevenue: { $sum: "$amount" }
    }
  },
  {
    $merge: {
      into: "salesSummary",
      whenMatched: "merge",
      whenNotMatched: "insert"
    }
  }
])
```

Sample Experiment Question

1. We have a sales collection that contains details about product sales, including category, product name, quantity sold, price, and customer ID

Table 3.3.1 Sales collection

id	Category	Product Name	Price	Quantity	Customer ID
1	Electronics	Laptop	2000	4	101
2	Clothing	Jacket	500	10	102
3	Furniture	Sofa	3000	2	103
4	Electronics	TV	4000	2	104
5	Clothing	T-Shirt	50	20	105

Write the queries for the following questions:

1. Which sales contain at least one item priced above 5000?
2. How can we break down the items array so each item is processed individually?
3. How can we join the sales collection with the customers collection to get customer details?
4. What is the total revenue and total sales for each category?

5. How can we sort the results by total revenue in descending order?
6. How can we rename _id to category and keep only relevant fields?
7. How can we add a discount field where total revenue > 10,000?
8. How can we retrieve only the top 5 categories based on total revenue?
9. How can we skip the first two categories from the results?
10. How many categories remain after filtering and sorting?
11. How can we store the final processed results into a new collection called salesSummary?
12. How can we merge the salesSummary data into an existing collection (finalSalesReport), updating existing records and inserting new ones?

Answers

1. // 1. $match - Filter sales where total amount > 5000

```
db.sales.aggregate([

 {

  $match: {

   "items.price": { $gt: 5000 }

  }

 },



 // 2. $unwind - Expand items array to process each item separately

 {

  $unwind: "$items"

 },



 // 3. $lookup - Join sales with customers collection to get customer details

 {
```

```
 $lookup: {
   from: "customers",
   localField: "customerId",
   foreignField: "_id",
   as: "customerDetails"
 }
},
```

// 4. $group - Group by category and calculate total revenue and total sales

```
{
 $group: {
   _id: "$category",
   totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } },
   totalSales: { $sum: "$items.quantity" }
 }
},
```

// 5. $sort - Sort by total revenue in descending order

```
db.sales.aggregate([
{ $unwind: "$items" },
{
 $group: {
   _id: "$category",
   totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } }
 }
```

```
  },
  {
    $sort: { totalRevenue: -1 }
  }
]);
```

// 6. $project - Select relevant fields and rename _id to category

```
 db.sales.aggregate([
  { $unwind: "$items" },
  {
    $group: {
      _id: "$category",
      totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } }
    }
  },
  {
    $project: {
      category: "$_id",
      totalRevenue: 1,
      _id: 0
    }
  }
]);
```

// 7. $addFields - Add discount field if total revenue > 10,000

```
 db.sales.aggregate([
  { $unwind: "$items" },
  {
   $group: {
     _id: "$category",
     totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } }
   }
  },
  {
   $project: {
     category: "$_id",
     totalRevenue: 1,
     _id: 0
   }
  }
]);
```

// 8. $limit - Limit results to the top 5 categories

```
db.sales.aggregate([
  { $unwind: "$items" },
  {
   $group: {
     _id: "$category",
     totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } }
   }
  },
 { $sort: { totalRevenue: -1 } },
 { $limit: 5 }
]);
```

// **9. $skip - Skip the first 2 categories**

```
db.sales.aggregate([
 { $unwind: "$items" },
 {
   $group: {
     _id: "$category",
     totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } }
   }
 },
 { $sort: { totalRevenue: -1 } },
 { $skip: 2 }
]);
```

// **10. $count - Count the remaining categories**

```
db.sales.aggregate([
 { $unwind: "$items" },
 {
   $group: {
     _id: "$category",
     totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } }
   }
```

```
  },
  { $sort: { totalRevenue: -1 } },
  { $count: "remainingCategories" }
]);
```

// 11. $out - Store results into a new collection

```
db.sales.aggregate([
  { $unwind: "$items" },
  {
    $group: {
      _id: "$category",
      totalRevenue: { $sum: { $multiply: ["$items.price", "$items.quantity"] } },
      totalSales: { $sum: "$items.quantity" }
    }
  },
  {
    $project: {
      category: "$_id",
      totalRevenue: 1,
      totalSales: 1,
      _id: 0
    }
  },
  { $out: "salesSummary" }
]);
```

// 12. $merge - Merge salesSummary into an existing collection

```
db.salesSummary.aggregate([
  {
   $merge: {
     into: "finalSalesReport",
     whenMatched: "merge",
     whenNotMatched: "insert"
   }
  }
]);
```

Lab Practice experiment

1. The given dataset represents sales transactions in a bookstore and stationery shop. Each record in the dataset contains details about products sold, categorized under Books and Stationery. The dataset includes essential fields such as category, product name, price, quantity sold, and customer ID

Table 3.3.2 Product table

id	Category	Product Name	Price	Quantity	Customer ID
1	Books	Science Fiction	20	3	301
2	Stationery	Notebook	5	10	302
3	Books	Mystery Novel	15	5	303
4	Stationery	Pen Set	12	7	304
5	Books	History Book	30	2	305

Write the queries for the following questions:

1. Which orders contain at least one item priced above $20?
2. What is the total revenue and total quantity sold for each category (Books & Stationery)?
3. How can we sort the categories by total revenue in descending order?

4. How can we rename id to category and keep only relevant fields?
5. How can we add a discount field where total revenue is greater than $50?
6. How can we retrieve only the top 2 categories based on total revenue?
7. How can we skip the first category from the sorted results?
8. How many categories remain after filtering and sorting?
9. How can we store the final processed results into a new collection called bookstoreSummary?
10. How can we merge the bookstoreSummary data into an existing collection (finalBookstoreReport), updating existing records and inserting new ones?

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ് വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin: 673002
Ph: 04952920228
email: rckdirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery,
Kannur, Pin: 670106
Ph: 04902990494
email: rctdirector@sgou.ac.in

Tripunithura
Govt. College
Tripunithura, Ernakulam,
Kerala, Pin: 682301
Ph: 04842927436
email: rcedirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad,
Kerala, Pin: 679303
Ph: 04662912009
email: rcpdirector@sgou.ac.in

NO TO DRUGS

തിരിച്ചിറങ്ങാൻ പ്രയാസമാണ്

ആരോഗ്യ കുടുംബക്ഷേമ വകുപ്പ്, കേരള സർക്കാർ

DBMS LAB

COURSE CODE: B24DS02PC

ISBN 978-81-989642-4-3

9 788198 964243

Sreenarayanaguru Open University

Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841