



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

NoSQL

Module Code : ACCA7002 Database Systems

*Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales*

www.uwtsd.ac.uk

Objectives

➤ Introduction to NoSQL

➤ **MongoDB**

➤ **Activities in MongoDB**

www.uwtsd.ac.uk

*Man Geni Addysg
Uwch yng Nghymru*

*Where Higher Education
Began in Wales*

Introduction to NoSQL Concepts

What is NoSQL?

- ✓ any non-relational database
- ✓ “non-SQL” or “not only SQL.”
- ✓ store data in a more natural and flexible way.
- ✓ adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data.
- ✓ designed to handle and store large volumes of unstructured and semi-structured data.

no fixed format, no predefined structure, and cannot be organised neatly in rows and columns.



- ## YouTube-8M Segments Dataset



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David



Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales

Example- semi structured:

The screenshot displays the Microsoft Outlook web application interface. The top navigation bar includes the Outlook logo, a search bar, and various utility icons. Below this, a ribbon menu shows 'Home', 'View', 'Help', 'Message', 'Insert', 'Format text', and 'Options'. The 'Message' tab is active, showing a rich text editor with a font dropdown set to 'Aptos' and size '11'. The left sidebar contains a 'Favorites' section with 'Inbox' (9 items), 'Expenses', and 'Invoices' (2 items). Below this, the account 'katyreid@outlook.com' is selected, showing its 'Inbox' (9 items), 'Drafts', 'Sent Items', 'Deleted Items', 'Junk Email', and 'Archive'. At the bottom of the sidebar, there is a link to 'Add account' and a prompt to 'Upgrade to Microsoft 365 with premium Outlook features'. The main content area is divided into two panes. The left pane shows a list of emails under the 'Focused' tab. The emails include: a pinned email from Elvia Atkins about 'Surprise Birthday Planning' (8:32 AM); a 'Today' section with an email from Contoso Airlines about a confirmed flight reservation (2:41 PM); an email from Lydia Bauer with 'Team Pictures' (11:10 AM); an email from Daisy Philips about a 'Yoga Workshop' (9:16 AM); a 'Yesterday' section with an email from Margie's Travel about a 'Confirmation Letter - MPOWMQ' (2:54 PM) which includes an 'RSVP' button for 'Fri 11/22/2023 2:35 PM'; and a partially visible email from Amanda Brady. The right pane shows a detailed view of an email from 'katyreid@outlook.com' to 'Dianne Russell'. The subject is 'Re: Apartment unit tour'. The email body contains the following text: 'Hi Dianne, I recently came across your apartment listing and I am interested in exploring the space. Could you please arrange a tour at your earliest convenience? I am available on Wednesday and Fridays after lunch, but I am flexible and willing to adjust my schedule. During the tour, I would like to view the apartment itself and any available communal areas or amenities. Additionally, it would be helpful to discuss lease terms, rental pricing, and the application process. Please let me know the next steps to schedule the tour. You can reach me at this email address. Best regards, Katy Reid'. A status bar at the bottom right indicates 'Draft saved at 11:00 AM'.

Relational Databases (RDB)

Relational DB-classified based on implementation models

- Traditional RDB(MYSQL,postgresql oracledb...),
- distributed DB(amazon aurora,google spanner,cockroach DB...)
- Cloud based RDB(azure SQL DB,google cloud SQL...)

Types of NoSQL Databases

- Document-oriented database
- Key-Value (KV) Stores
- Column Family Data stores or Wide column data stores
- Graph Databases

Document-oriented database

- Designed to store, manage, and query semi-structured data, typically in the form of documents
- flexible storage and retrieval of data in formats like JSON (JavaScript Object Notation), BSON (Binary JSON), or XML
- Each document in a document database is a collection of **key-value pairs**, where keys are the field names and values can be any type of data, such as strings, arrays, or nested documents.

Product 1: Name: "Laptop" Price: 1000 Brand: "Brand A" Features: ["8GB RAM", "500GB SSD", "15-inch screen"]	Product 2: Name: "Phone" Price: 500 Brand: "Brand B" Features: ["4GB RAM", "128GB storage", "6-inch screen"] Warranty: "2 years"
---	---

Key characteristics of document databases include:

Flexible Schema

Scalability

designed to scale horizontally, meaning they can easily handle large volumes of data by distributing the data across multiple servers.

High Performance

optimized for fast reads and writes, making them suitable for applications that require high-performance data access.

Example

Imagine storing blog posts:

Document 1 (JSON):

```
{  
  "title": "My first blog",  
  "author": "Alex",  
  "tags": ["tech", "coding"],  
  "likes": 20  
}
```

Document 2 (JSON):

```
{  
  "title": "Travel diary",  
  "author": "Alex",  
  "comments": ["Nice!", "Love  
it!"]  
}
```

Use Cases:

used in content management systems, e-commerce platforms, and applications that manage user profiles, blogs, and other types of data with a flexible, evolving structure.

Article:	Video Post:
<pre>{ "title": "How to Learn Python", "author": "John Doe", "content": "Python is a versatile programming language...", "tags": ["Programming", "Python"] }</pre>	<pre>{ "title": "Learn Python in 10 Minutes", "video_url": "https://example.com/video123", "description": "A quick tutorial on Python.", "tags": ["Tutorial", "Python"] }</pre>

Similar Example

User Profile Document:	Blog Post Document:
<pre>{ "username": "johndoe123", "email": "johndoe@example.com", "preferences": { "theme": "dark", "notifications": "enabled" }, "friends": ["janedoe456", "mikesmith789"] }</pre>	<pre>{ "title": "My First Blog Post", "author": "johndoe123", "content": "This is my first blog post. I'm excited to share!", "comments": [{ "user": "janedoe456", "comment": "Great post!" }, { "user": "mikesmith789", "comment": "I agree, very interesting!" }], "tags": ["Introduction", "Personal"] }</pre>

Popular Document DBs

Some popular types of NoSQL databases that follow the document-oriented model are:

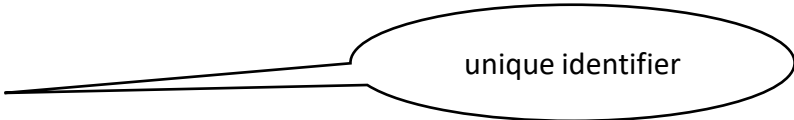
- MongoDB
- CouchDB
- Amazon DocumentDB

Key-Value (KV) databases

- stores data as **pairs of keys and values**..
- **Key**: A unique identifier, like a name or ID.
- **Value**: The data or information associated with that key, which can be anything (a string, number, image, etc.).

Example:

```
{  
  "_id": "1a2b3c",  
  "title": "My first blog",  
  "author": "Alex",  
  "tags": ["tech", "coding"],  
  "likes": 20  
}
```



Key-Value (KV) databases

- Simplest type of NoSQL database
- The data is stored in the form of key and value pairs.
- **Key:** A unique identifier, like a name or ID.
- **Value:** The data or information associated with that key, which can be anything (a string, number, image, etc.).

Key-Value Databases	Document-Oriented Databases
<pre>{ "_id": "1a2b3c", "key1": "value1", "userID123": "sessionData", "productID456": "productDetails" }</pre>	<pre>{ "userID": "123", "name": "John Doe", "address": { "street": "123 Main St", "city": "Metropolis" }, "orders": [{"orderID": "456", "amount": 29.99}, {"orderID": "789", "amount": 49.99}] }</pre>

Key Features of Key-Value Databases

- **Simple Structure:**
- **Fast Access:** Since each key is unique, accessing the data is very quick. You can retrieve the value simply by looking up its associated key.

Use Cases for Key-Value Databases

Caching:

Storing frequently accessed data to reduce the load on primary databases and improve performance.(like website content)

Session Management

Storing user session data such as authentication tokens, user preferences, and login information.

Shopping Cart Data

Storing temporary user shopping cart information in e-commerce applications.

Geospatial Data

Storing and retrieving location-based data, such as the current location of a vehicle or the nearest store to a user.

Popular Key value Stores would include

- Dynamo DB
- Redis
- BerkleyDB.

Wide column databases

- ✓ store data in tables, rows, and **dynamic columns**
- ✓ designed to handle large amounts of data across many distributed data

StudentID	Name	Age	Address
1	John	22	New York
2	Alice	25	Los Angeles
3	Bob	30	Chicago

Column Family: `students`

Row Key	Column 1	Column 2	Column 3	Column 4
1	{ "Name": "John", "Age": 22 }	{ "Address": "New York" }		
2	{ "Name": "Alice", "Age": 25 }	{ "Address": "Los Angeles" }		
3	{ "Name": "Bob", "Age": 30 }	{ "Address": "Chicago" }	{ "Phone": "1234" }	{ "Email": "bob@example.com" }

Use Cases for Wide Column Databases

- Financial Services and Fraud Detection

Storing transactional data and analysing patterns for fraud detection.

- IoT Data Storage

Storing and managing data from IoT devices, such as sensor readings, device statuses, and environmental factors.

- Healthcare Data Management

Managing patient records, medical histories, and sensor data from medical devices.

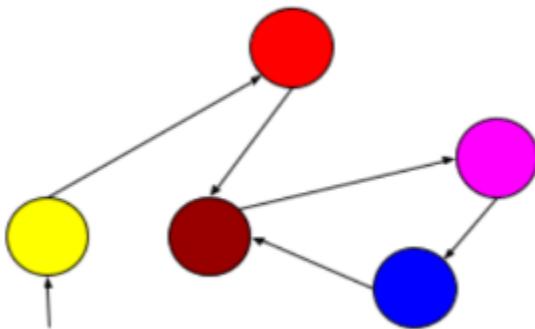
Wide Column Databases

Some of the popular Wide column data stores include

- Google's Bigtable (Google (internally for services like Search and Maps), Spotify, and Twitter (to store tweets).)
- Apache Cassandra (Netflix, eBay, Instagram, Uber, and Reddit.)
- Hbase (Facebook, Yahoo, and eBay)

Graph Databases

- uses graph structures to store and query data
- provides index-free adjacency - every node contains a direct pointer to its adjacent element
- useful when dealing with data that is highly interconnected, like social networks, recommendation systems, and fraud detection.



Basic Components of a Graph Database

Nodes (Vertices):

- represents an entity or object in the system
- Each node can store properties (additional data) about the entity.

Edges (Relationships):

- connections between two nodes
- Each edge can also have properties that describe the relationship further.

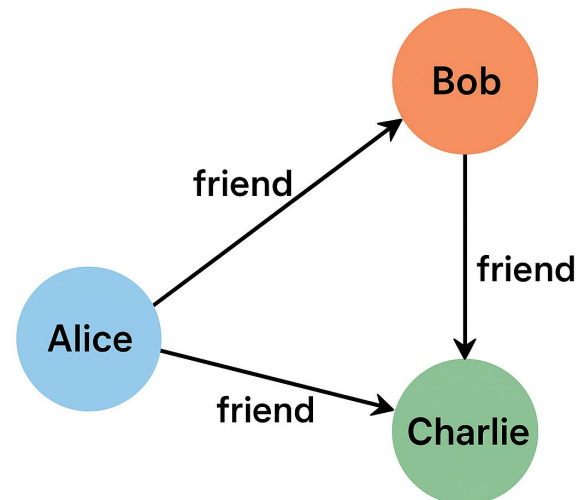
Graph Databases

- Basic Components of a Graph Database
- **Nodes (Vertices):**
 - Node: Person
 - Properties: {name: "Alice", age: 30, location: "New York"}
- **Edges (Relationships):**
 - Alice -[FRIEND]-> Bob
 - Properties: {since: 2015}

Graph query:

“Find all friends of Alice’s friends.”

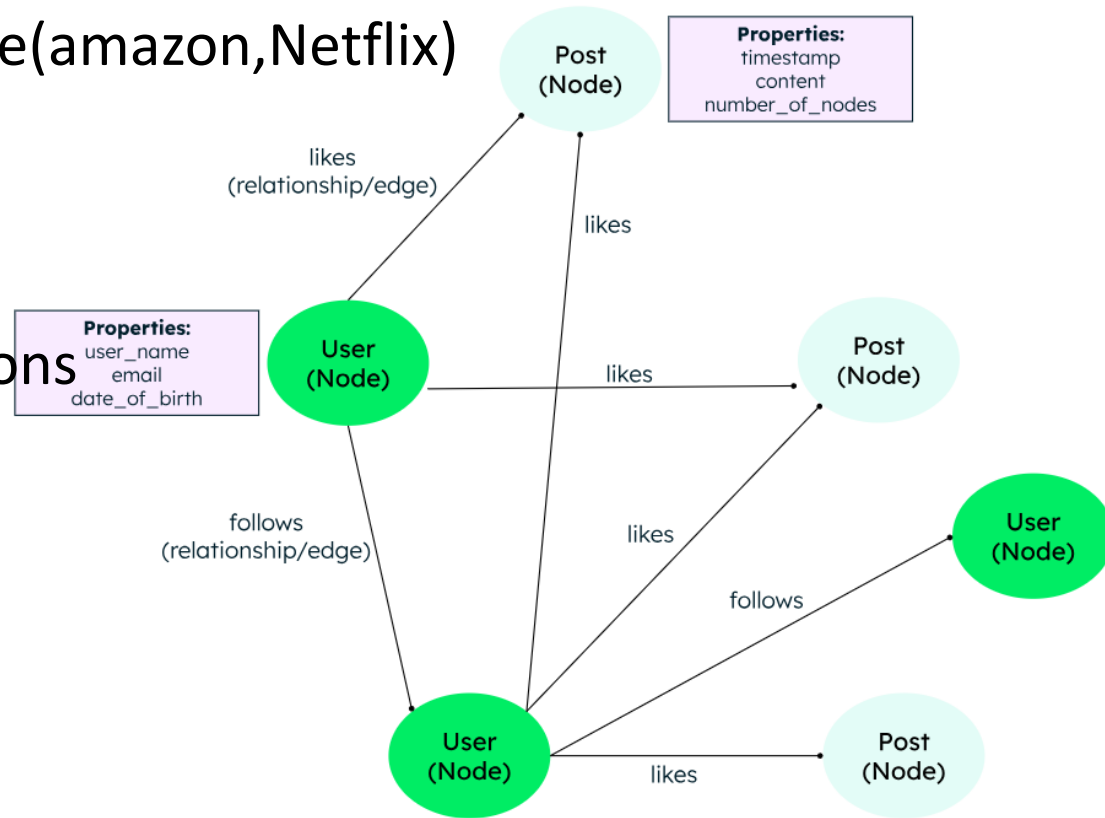
✓ Very fast in graph DB.



Graph Databases: Use Case

- Typical use cases for graph databases would include

- Social networking site(facebook, linkedin)
- Recommendation engine(amazon,Netflix)
- Logistics
- Knowledge Graphs
- Fraud detection
- Network and IT Operations



Some of the popular graph databases :

- Neo4j
 - uses the **Cypher** query language to interact with the graph.
 - used for social networks and recommendation systems.
- Amazon Neptune
 - used in fraud detection and knowledge graph scenarios.
- ArangoDB

Knowledge Check

You are building a **social network** where users can follow and connect with each other. The system needs to efficiently query relationships between users, such as "Who follows whom?" and "What are the mutual connections between two users?"

Which NoSQL database type would you choose?

Graph Database

- Reason: Social networks involve complex relationships between users that are best handled by a graph database. Nodes represent users, and edges represent connections.

Knowledge Check

You are designing an e-commerce catalog for an online store. The catalog contains a variety of products with different attributes (e.g., clothing, electronics, and books). Each product has different fields, such as name, price, description, and category, but not all products have the same attributes.

Which NoSQL database type would you choose?

Document Store

Reason: A document store allows flexible schema design, which is perfect for handling diverse product types with varying attributes. You can store product details in JSON format, making it easier to manage without a fixed schema.

Knowledge Check

You need a **real-time analytics** system to monitor temperature data from thousands of sensors spread across multiple locations. The system collects large amounts of time-series data, and you need to efficiently query temperature readings for specific time periods.

Which NoSQL database type would you choose?

Wide column databases

Reason: wide column stores are optimized for handling time-series data, where you frequently query large volumes of data by specific columns (e.g., time or location). These databases are well-suited for read-heavy analytics tasks.

MongoDB

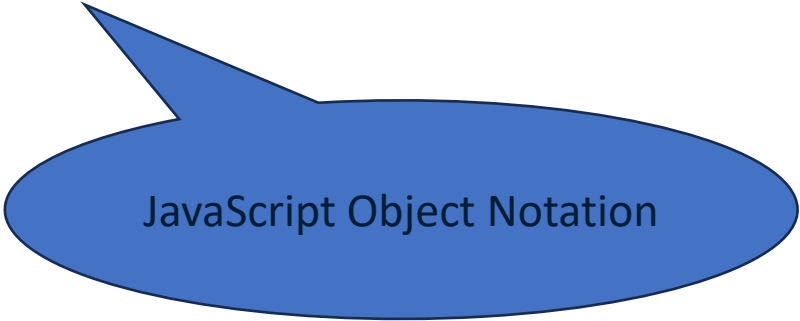
MongoDB:

- A document-oriented NoSQL database.
- Stores data in **JSON-like documents**.
- Schema-less (flexible structure for data storage).
- Ideal for hierarchical or unstructured data.

What is NoSQL and MongoDB?

MongoDB:

- Stores data in **JSON-like documents**.



JavaScript Object Notation



Activity PostgreSQL vs MongoDB

- **Create Table using PostgreSQL:**

```
CREATE TABLE students (  
    id SERIAL PRIMARY KEY,  
    name VARCHAR(50),  
    age INT,  
);  
  
CREATE TABLE student_subjects (  
    id SERIAL PRIMARY KEY,  
    student_id INT,  
    subject VARCHAR(50),  
    FOREIGN KEY (student_id) REFERENCES  
    students(id));
```

Activity PostgreSQL vs MongoDB

Insert Data using PostgreSQL:

-- Insert a student

```
INSERT INTO students (name, age) VALUES ('John', 22);
```

-- Insert the subjects as separate rows in the student_subjects table

```
INSERT INTO student_subjects (student_id, subject) VALUES  
(1, 'Math'),      (1, 'Science');
```

-- Select all students with their subjects

```
SELECT s.name, s.age, ss.subject
```

```
FROM students s
```

```
JOIN student_subjects ss ON s.id =  
ss.student_id;
```

Activity PostgreSQL vs MongoDB

- Query Data **using PostgreSQL:**

```
SELECT * FROM students WHERE age >= 20;
```

```
db.createCollection("students")  
db.createCollection("student_subjects")
```

Activity PostgreSQL vs MongoDB

- **Insert a Document using MongoDB:**

```
db.students.insertOne({  
  name: "John",  
  age: 22,  
  subjects: ["Math", "Science"]  
});
```

Insert many documents into a collection

```
db.students.insertMany([
  {
    name: "John",
    age: 22,
    subjects: ["Math",
"Science"]
  },
  {
    name: "Anna",
    age: 20,
    subjects:
["English", "History"]
  },
```

```
{
    name: "Mark",
    age: 23,
    subjects:
["Physics"]
  }
]);
```

Activity PostgreSQL vs MongoDB

Query all students with subjects

```
db.students.find({});
```

Find Data using MongoDB:

```
db.students.find({  
  age: { $gte: 20 }  
});
```

Activity PostgreSQL vs MongoDB

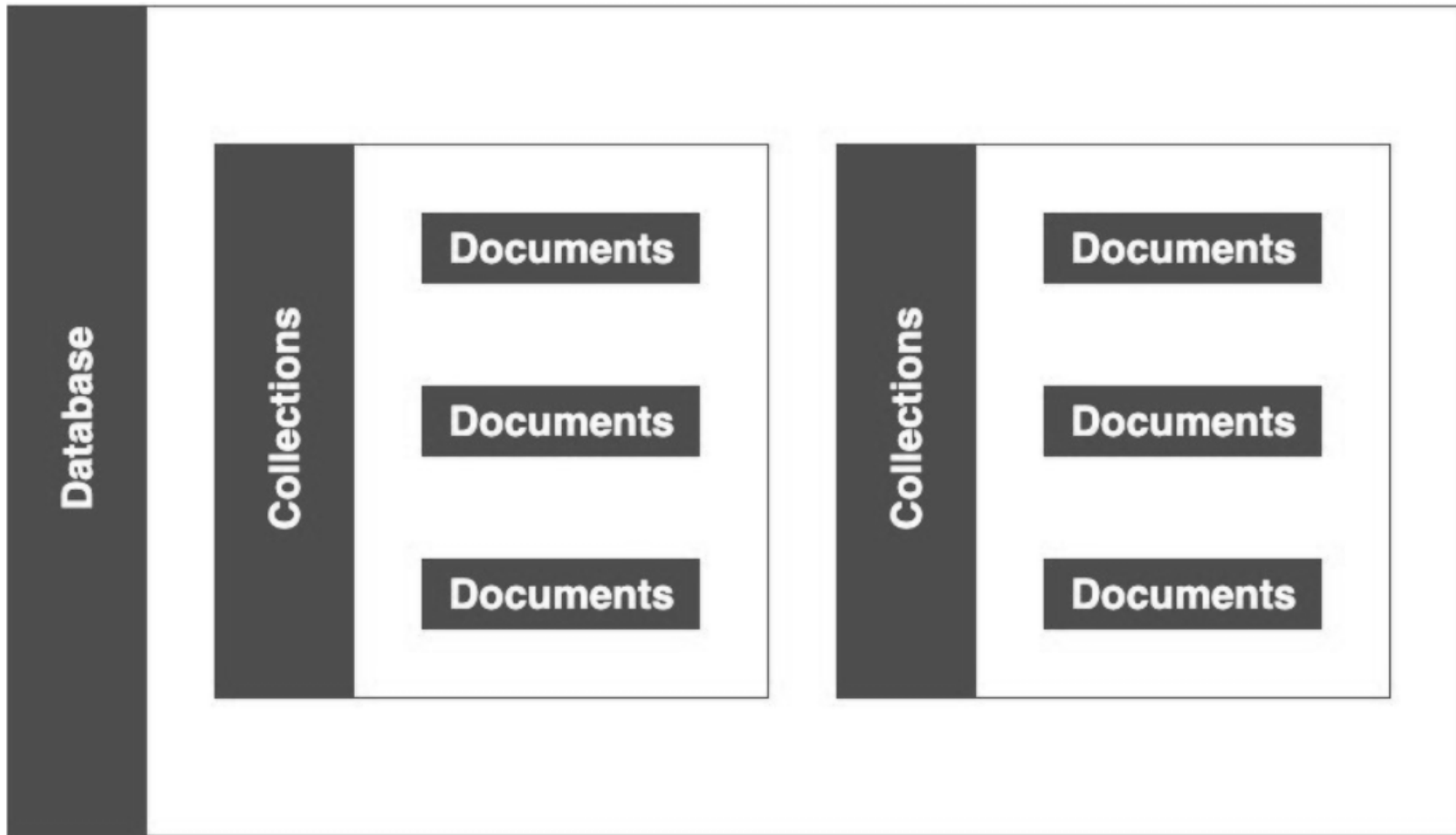
- **Update a Document using MongoDB:**

```
db.students.updateOne({ name: "John" }, { $set: { age: 23 } });
```

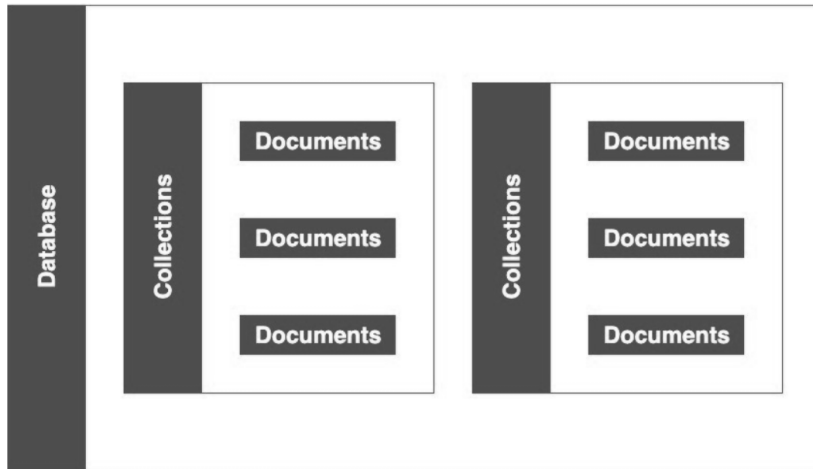
Delete

```
db.students.deleteOne({ name: "John" });
```

MongoDB – Database, Collection, and Document



MongoDB – Database, Collection, and Document



- **Database:** The entire library (e.g., a building that contains many bookshelves).
- **Collection:** A bookshelf in the library (grouping of related books).
- **Document:** A single book on the shelf (the actual data with details like title, author, and pages).

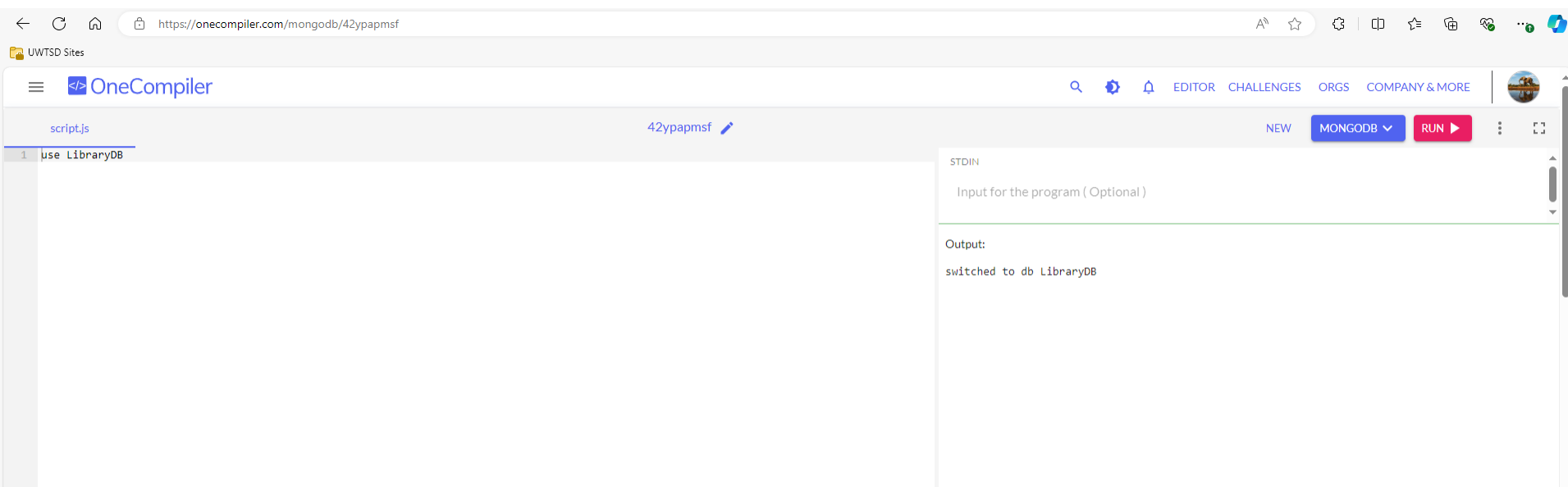
MongoDB – Database, Collection, and Document

- Create the Database
 - Open the MongoDB shell (or)
 - use a GUI tool like MongoDB Compass (or)
 - <https://onecompiler.com/mongodb>

use LibraryDB

MongoDB – Database, Collection, and Document

use LibraryDB



The screenshot shows a web browser at the URL <https://onecompiler.com/mongodb/42ypapmsf>. The OneCompiler interface includes a top navigation bar with links for EDITOR, CHALLENGES, ORGS, and COMPANY & MORE. The main editor area shows a file named `script.js` with the following code:

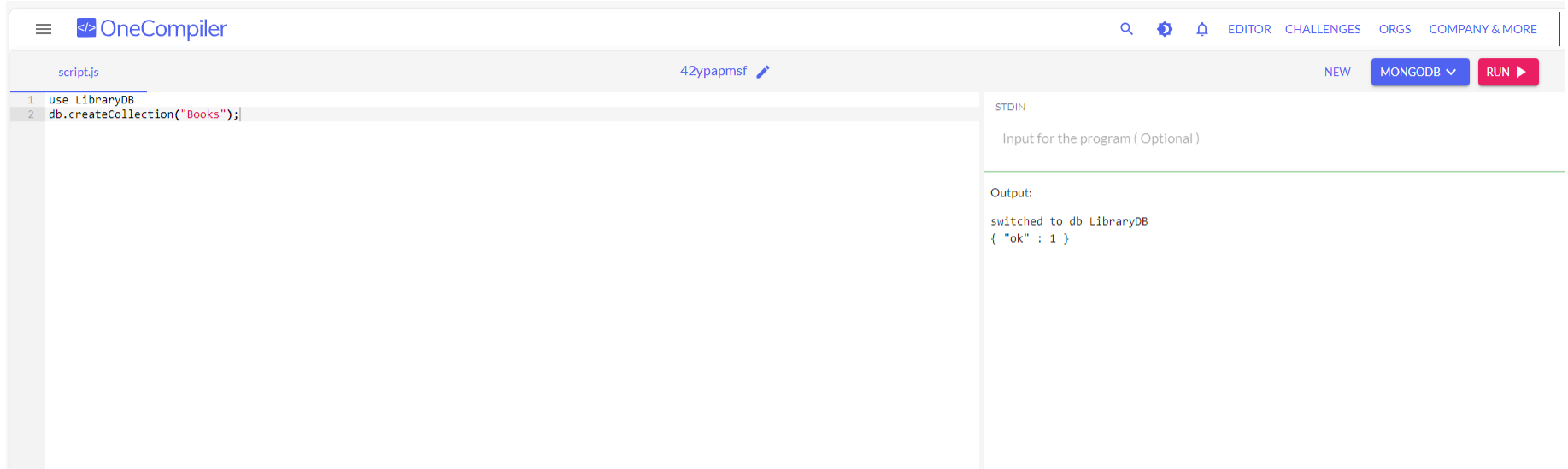
```
1 use LibraryDB
```

On the right side of the editor, there is a panel with a 'MONGODB' dropdown menu and a 'RUN' button. Below this, the 'STDIN' section is empty, and the 'Output' section displays the message: 'switched to db LibraryDB'.

MongoDB – Database, Collection, and Document

use LibraryDB

```
db.createCollection("Books");
```



The screenshot shows the OneCompiler web interface. The top navigation bar includes a search icon, a settings icon, a bell icon, and links for EDITOR, CHALLENGES, ORGS, and COMPANY & MORE. Below the navigation bar, the editor area shows a file named 'script.js' with two lines of code: '1 use LibraryDB' and '2 db.createCollection("Books");'. The right sidebar contains a 'STDIN' section with the text 'Input for the program (Optional)' and an 'Output' section showing the result: 'switched to db LibraryDB' and '{ "ok" : 1 }'. The interface also features a 'NEW' button, a 'MONGODB' dropdown menu, and a 'RUN' button.

```
1 use LibraryDB
2 db.createCollection("Books");
```

switched to db LibraryDB
{ "ok" : 1 }

MongoDB – Database, Collection, and Document

use LibraryDB

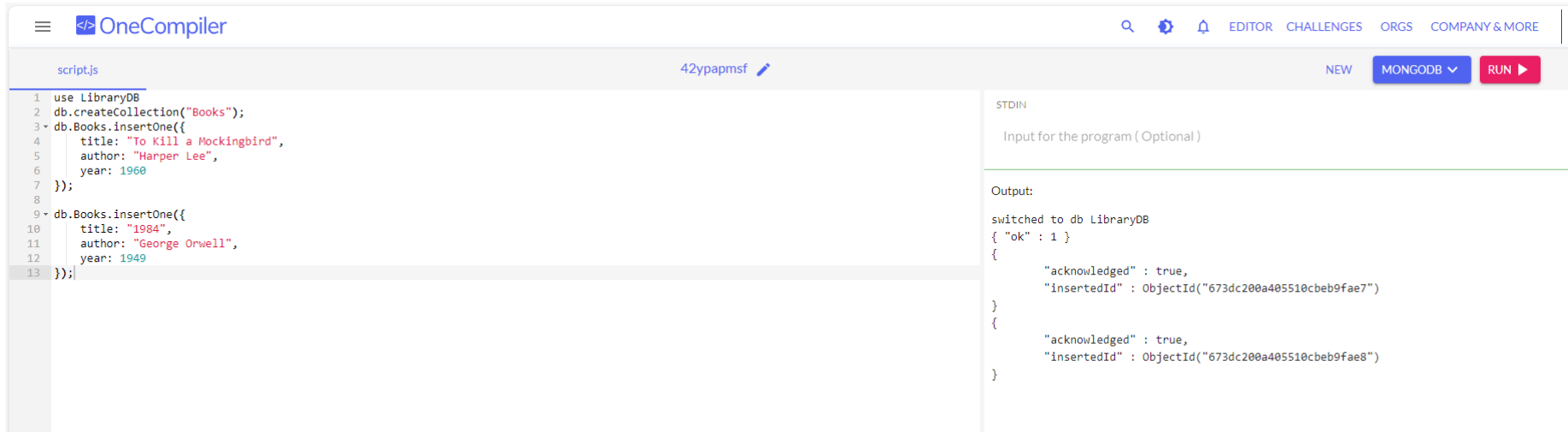
```
db.createCollection("Books");
```

```
db.Books.insertOne({  
  title: "To Kill a Mockingbird",  
  author: "Harper Lee",  
  year: 1960  
});
```

MongoDB – Database, Collection, and Document

```
db.Books.insertOne({ title: "1984",  
author: "George Orwell",  
year: 1949});
```

MongoDB – Database, Collection, and Document



The screenshot shows the OneCompiler web IDE interface. The editor on the left contains a JavaScript script for interacting with a MongoDB database. The script defines a 'LibraryDB' database and a 'Books' collection, then inserts two documents. The first document is for 'To Kill a Mockingbird' by Harper Lee (1960), and the second is for '1984' by George Orwell (1949). The right-hand panel displays the output of the script, showing the MongoDB connection status and the successful insertion of both documents, each with an 'acknowledged' status of true and a unique 'insertedId'.

```
1 use LibraryDB
2 db.createCollection("Books");
3 db.BOOKS.insertOne({
4   title: "To Kill a Mockingbird",
5   author: "Harper Lee",
6   year: 1960
7 });
8
9 db.BOOKS.insertOne({
10  title: "1984",
11  author: "George Orwell",
12  year: 1949
13 });
```

STDIN

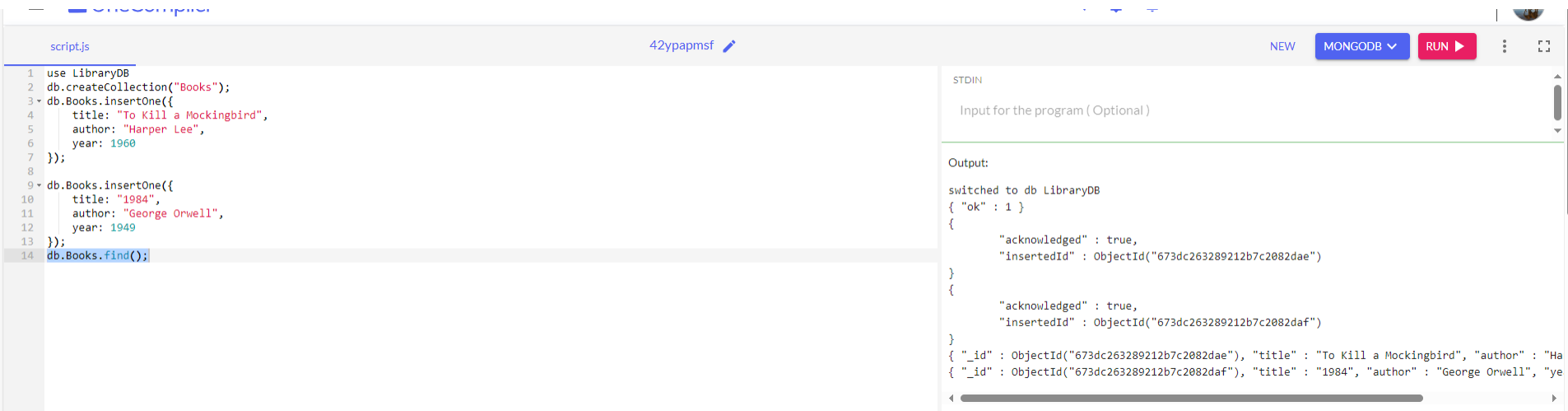
Input for the program (Optional)

Output:

```
switched to db LibraryDB
{ "ok" : 1 }
{
  "acknowledged" : true,
  "insertedId" : ObjectId("673dc200a405510cbeb9fae7")
}
{
  "acknowledged" : true,
  "insertedId" : ObjectId("673dc200a405510cbeb9fae8")
}
```

MongoDB – Retrieve Data

Query the collection to see all documents:
`db.Books.find();`



The screenshot shows a MongoDB CLI window with a script named `script.js` being executed. The script contains the following code:

```
1 use LibraryDB
2 db.createCollection("Books");
3 db.Books.insertOne({
4   title: "To Kill a Mockingbird",
5   author: "Harper Lee",
6   year: 1960
7 });
8
9 db.Books.insertOne({
10  title: "1984",
11  author: "George Orwell",
12  year: 1949
13 });
14 db.Books.find();
```

The output of the script is displayed in the right-hand pane:

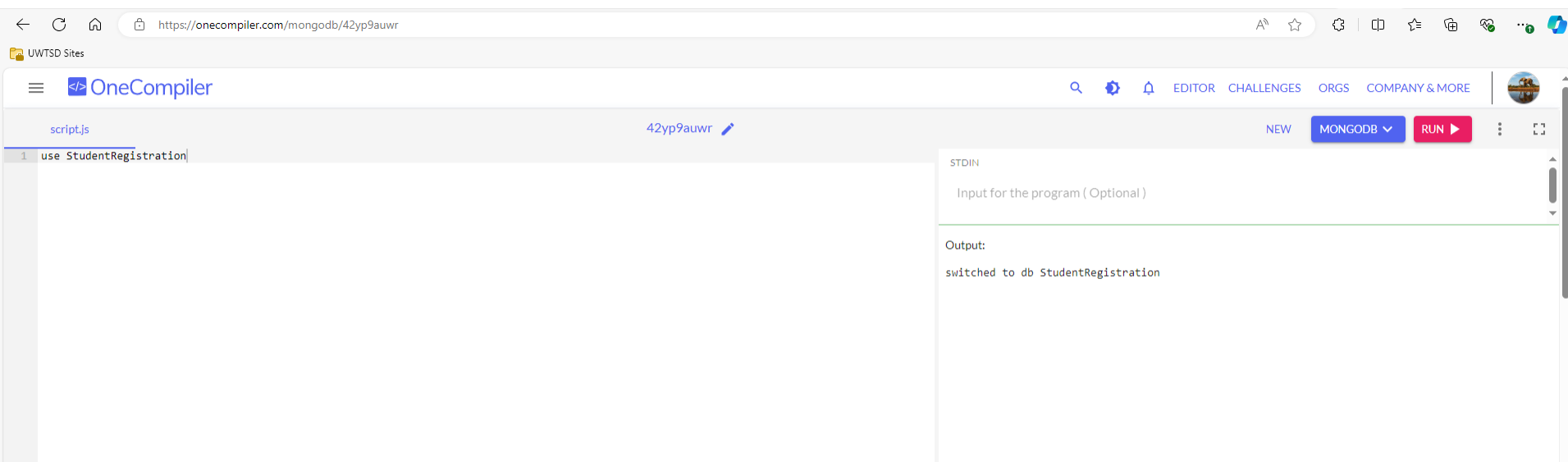
```
switched to db LibraryDB
{ "ok" : 1 }
{
  "acknowledged" : true,
  "insertedId" : ObjectId("673dc263289212b7c2082dae")
}
{
  "acknowledged" : true,
  "insertedId" : ObjectId("673dc263289212b7c2082daf")
}
{ "_id" : ObjectId("673dc263289212b7c2082dae"), "title" : "To Kill a Mockingbird", "author" : "Ha
{ "_id" : ObjectId("673dc263289212b7c2082daf"), "title" : "1984", "author" : "George Orwell", "ye
```

MongoDB Activity

- [MongoDB Online Editor https://onecompiler.com/mongodb](https://onecompiler.com/mongodb)
- **Student Course Registration System**
 - In this system:
 - Students register for courses.
 - Each course has details like name, instructor, and credits.
 - Students have details like name, age, and the courses they've registered for.

Steps to Create the MongoDB Database

1. Set Up the MongoDB Database use StudentRegistration



Steps to Create the MongoDB Database

2. Create/insert a collection(s) for Students

insertOne and insertMany

```
db.students.insertMany([
```

```
{
```

```
  student_id: 1,
```

```
  name: "Alice",
```

```
  age: 20,
```

```
  courses: [
```

```
    { course_id: "C101", name: "Mathematics" },
```

```
    { course_id: "C102", name: "Physics" } ]
```

```
],
```

```
}
```

Steps to Create the MongoDB Database

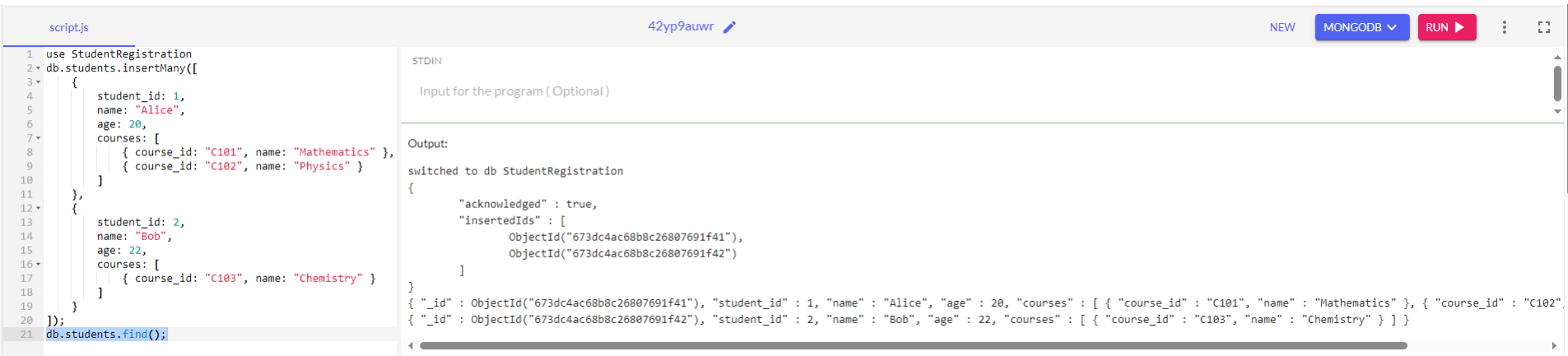
2. Create/insert a collection(s) for Students

```
{
  student_id: 2,
  name: "Bob",
  age: 22,
  courses: [
    { course_id: "C103", name: "Chemistry" }
  ]
}
]);
```

Steps to Create the MongoDB Database

3. Query Students

```
db.students.find();
```



The screenshot shows a MongoDB IDE interface. On the left, a script named 'script.js' contains a JavaScript snippet to insert two students into a 'StudentRegistration' database. The first student is Alice, age 20, with courses in Mathematics and Physics. The second student is Bob, age 22, with a course in Chemistry. The script ends with a query to find all students in the 'students' collection.

```
1 use StudentRegistration
2 db.students.insertMany([
3   {
4     student_id: 1,
5     name: "Alice",
6     age: 20,
7     courses: [
8       { course_id: "C101", name: "Mathematics" },
9       { course_id: "C102", name: "Physics" }
10    ],
11  },
12  {
13    student_id: 2,
14    name: "Bob",
15    age: 22,
16    courses: [
17      { course_id: "C103", name: "Chemistry" }
18    ]
19  }
20 ]);
21 db.students.find();
```

On the right, the 'Output' pane shows the result of the script execution. It indicates that the data was successfully inserted into the 'StudentRegistration' database, providing the acknowledged status, the inserted IDs, and the full details of the two students as stored in the database.

```
switched to db StudentRegistration
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("673dc4ac68b8c26807691f41"),
    ObjectId("673dc4ac68b8c26807691f42")
  ]
}
{ "_id" : ObjectId("673dc4ac68b8c26807691f41"), "student_id" : 1, "name" : "Alice", "age" : 20, "courses" : [ { "course_id" : "C101", "name" : "Mathematics" }, { "course_id" : "C102", "name" : "Physics" } ] }
{ "_id" : ObjectId("673dc4ac68b8c26807691f42"), "student_id" : 2, "name" : "Bob", "age" : 22, "courses" : [ { "course_id" : "C103", "name" : "Chemistry" } ] }
```

Steps to Create the MongoDB Database

4. Create a Collection for Courses

```
db.courses.insertMany([
  {
    course_id: "C101",
    name: "Mathematics",
    instructor: "Dr. Smith",
    credits: 3
  },
  {
    course_id: "C102",
    name: "Physics",
    instructor: "Dr. Johnson",
    credits: 4
  },
  {
    course_id: "C103",
    name: "Chemistry",
    instructor: "Dr. Williams",
    credits: 3
  }
]);
```

Querying the Data

```
db.students.find();
```

```
db.students.find({ name: "Alice" });
```

```
db.courses.find();
```

```
db.students.find({ "courses.course_id": "C101" });
```

Coursework Requirements: Data Storage Investigation (ACCA7002)

Assignment Title: Data Storage Investigation

Module Code: ACCA7002

Weightage: 50% of total module marks

Deadline: January 19, 2026

Submission: Via Turnitin (Late penalties apply)

Word Count: 2500 words (or equivalent)

Objectives and Learning Outcomes

- Purpose:
 - Investigate data storage solutions.
 - Compare NoSQL database types with relational databases.
 - Implement NoSQL and relational database solutions.
- Learning Outcomes:
 - Critically evaluate database methods, tools, and techniques.
 - Deploy these techniques in a multi-user environment.

Key Tasks

- Comparative Analysis (35%):
 - Compare Document, Graph, Key-Value, and Wide Column databases.
 - Highlight use cases for each.
 - Contrast with traditional relational databases.
- NoSQL vs Relational Databases (25%):
 - Discuss when to use NoSQL vs SQL.
- Implementation Demonstration (40%):
 - Set up NoSQL and relational databases.
 - Provide SQL and NoSQL code for the implementations.

CloudLearn Case Study Context

- Platform Overview:
 - pixelbay provides platform for art marketplace.
 - Collaborates with instructors and institutions.
- Requirements:
 - Store customers, artists, and transaction data.
 - Choose and justify an data storage solution.
- Focus:
 - Scalability, performance, and data integrity.

Implementation Requirements

- Relational Database:
 - Use SQL to create tables with constraints (e.g., foreign keys, checks).
 - Populate tables with at least 5 sample records.
 - Apply performance tuning techniques.
- NoSQL Database:
 - Demonstrate data storage using a suitable NoSQL type (e.g., Document).
 - Include example queries and operations.
- Both Approaches:
 - Compare performance and suitability for Pixel Bay .

Structure of the Report

- **Abstract:** Summary of the report.
- **Table of Contents:** Outline the structure.
- **Overview:** Goals of pixel bay and database needs.
- **NoSQL Scenarios:** Highlight where NoSQL is advantageous.
- **Evaluation:** Recommend a database solution.
- **Justification:** Discuss data organization, volume, scalability, etc.
- **Implementation:** Details of database setups.
- **Data Security and Backup:** Strategies for pixel bay.
- **Summary and Recommendation:** Final decision and reasoning.

Tips for Success

- Research:**

- Use credible, recent sources.
- Focus on academic journals and industry articles.

- Implementation:**

- Ensure your database schemas meet pixel bay's needs.
- Provide working SQL and NoSQL code examples.

- Clarity:**

- Structure your report professionally.
- Use diagrams to explain database schemas and relationships.

Academic Honesty

- Cite all sources properly using an appropriate referencing style.
- Avoid plagiarism; ensure originality in code and analysis.

Further Reading :

<https://www.mongodb.com/docs/manual/tutorial/getting-started/>

<https://www.w3schools.com/mongodb/>