Next Generation Technologies Practical
B. Sc. (Information Technology) Semester – V

By munotes.in

B. Sc. (Information Technology)		Semester – V	
Course Name: Next Generation Technologies Practical		Course Code: USIT5P7 (Elective II)	
Periods per week (1 Period is 50 minutes)		3	
Credits		2	
		Hours	Marks
Evaluation System	Practical Examination	2½	50
	Internal	--	--

Practical No	Details
1	**MongoDB Basics**
a	Write a MongoDB query to create and drop database.
b	Write a MongoDB query to create, display and drop collection
c	Write a MongoDB query to insert, query, update and delete a document.
2	**Simple Queries with MongoDB**
3	**Implementing Aggregation**
a	Write a MongoDB query to use sum, avg, min and max expression.
b	Write a MongoDB query to use push and addToSet expression.
c	Write a MongoDB query to use first and last expression.
4	**Replication, Backup and Restore**
a	Write a MongoDB query to create Replica of existing database.
b	Write a MongoDB query to create a backup of existing database.
c	Write a MongoDB query to restore database from the backup.
5	**Java and MongoDB**
a	Connecting Java with MongoDB and inserting, retrieving, updating and deleting.
6	**PHP and MongoDB**
a	Connecting PHP with MongoDB and inserting, retrieving, updating and deleting.
7	**Python and MongoDB**

a	Connecting Python with MongoDB and inserting, retrieving, updating and deleting.
8	**Programs on Basic jQuery**
a	jQuery Basic, jQuery Events
b	jQuery Selectors, jQuery Hide and Show effects
c	jQuery fading effects, jQuery Sliding effects
9	**jQuery Advanced**
a	jQuery Animation effects, jQuery Chaining
b	jQuery Callback, jQuery Get and Set Contents
c	jQuery Insert Content, jQuery Remove Elements and Attribute
10	**JSON**
a	Creating JSON
b	Parsing JSON
c	Persisting JSON
11	**Create a JSON file and import it to MongoDB**
a	Export MongoDB to JSON.
b	Write a MongoDB query to delete JSON object from MongoDB

1 MongoDB Basics

a Write a MongoDB query to create and drop database.

Solution

To create and drop a MongoDB database, you can use the following commands in the MongoDB shell:

1. Create a Database:
```
use mydatabase
```
Note: Replace "mydatabase" with the name you want to give to your database.

2. Drop a Database:
```
use mydatabase
db.dropDatabase()
```
Note: Replace "mydatabase" with the name of the database you want to drop.

b Write a MongoDB query to create, display and drop collection

Solution

To create, display, and drop a collection in MongoDB, you can use the following commands:

1. Create a Collection:

To create a collection, you need to insert at least one document into it. MongoDB automatically creates the collection when you insert the first document. Here's an example of creating a collection named "mycollection" and inserting a document into it:

```
use mydatabase
db.mycollection.insertOne({ name: "John", age: 30 })
```

Note: Replace "mydatabase" with the name of the database where you want to create the collection. Also, the document inserted here is just an example; you can insert any valid document structure you want.

2. Display a Collection:
To display the documents in a collection, you can use the `find()` method. Here's an example:

```
use mydatabase
db.mycollection.find()
```

This command will display all the documents in the "mycollection" collection.

3. Drop a Collection:
To drop a collection, you can use the `drop()` method. Here's an example:

```
use mydatabase
db.mycollection.drop()
```

This command will drop the "mycollection" collection.

Note: Remember to replace "mydatabase" with the name of your actual database and "mycollection" with the name of the collection you want to create, display, or drop.

c Write a MongoDB query to insert, query, update and delete a document.

Solution

To insert, query, update, and delete a document in MongoDB, you can use the following commands:

1. Insert a Document:
To insert a document into a collection, you can use the `insertOne()` or `insertMany()` method. Here's an example of inserting a document into a collection named "mycollection":

```shell
use mydatabase
db.mycollection.insertOne({ name: "John", age: 30 })
```
This command will insert a document with the fields "name" and "age" into the "mycollection" collection.

2. Query a Document:
To query a document in MongoDB, you can use the `find()` method along with query filters. Here's an example of querying documents with the name "John" from the "mycollection" collection:

```shell
```

```
use mydatabase
db.mycollection.find({ name: "John" })
```

This command will return all the documents in the "mycollection" collection where the name is "John".

3. Update a Document:
To update a document in MongoDB, you can use the `updateOne()` or `updateMany()` method. Here's an example of updating the age of a document with the name "John" in the "mycollection" collection:

```shell
use mydatabase
db.mycollection.updateOne({ name: "John" }, { $set: { age: 35 } })
```

This command will update the age of the document with the name "John" to 35.

4. Delete a Document:
To delete a document in MongoDB, you can use the `deleteOne()` or `deleteMany()` method. Here's an example of deleting a document with the name "John" from the "mycollection" collection:

```shell
use mydatabase
db.mycollection.deleteOne({ name: "John" })
```

This command will delete the first document with the name "John" from the "mycollection" collection.

Note: Remember to replace "mydatabase" with the name of your actual database and "mycollection" with the name of the collection you are working with.

2 Simple Queries with MongoDB

Solution

Sure! Here are two simple queries you can perform in MongoDB:

1. Find Documents:
To retrieve documents from a collection based on certain criteria, you can use the `find()` method. Here's an example of finding all documents in a collection named "mycollection":

```shell
use mydatabase
db.mycollection.find()
```

This command will return all documents present in the "mycollection" collection.

You can also specify filter conditions to retrieve specific documents. For example, to find documents where the "age" field is greater than 25, you can use the following query:

```shell
use mydatabase
db.mycollection.find({ age: { $gt: 25 } })
```

This command will return all documents in the "mycollection" collection where the "age" field is greater than 25.

2. Count Documents:

To count the number of documents in a collection that match certain criteria, you can use the `count()` method. Here's an example of counting the documents in the "mycollection" collection:

```shell
use mydatabase
db.mycollection.count()
```

This command will return the total number of documents in the "mycollection" collection.

You can also count documents based on specific conditions. For example, to count documents where the "age" field is greater than 25, you can use the following query:

```shell
use mydatabase
db.mycollection.count({ age: { $gt: 25 } })
```

This command will return the count of documents in the "mycollection" collection where the "age" field is greater than 25.

Note: Remember to replace "mydatabase" with the name of your actual database and "mycollection" with the name of the collection you are working with.

3 Implementing Aggregation

a Write a MongoDB query to use sum, avg, min and max expression.

Solution

To use sum, avg, min, and max expressions in MongoDB, you can utilize the Aggregation Framework. The Aggregation Framework provides powerful tools to perform complex data analysis and calculations. Here's an example query that demonstrates the usage of sum, avg, min, and max expressions:

```shell
use mydatabase
db.mycollection.aggregate([
  {
    $group: {
      _id: null,
      totalSum: { $sum: "$quantity" },
      average: { $avg: "$price" },
      minimum: { $min: "$age" },
      maximum: { $max: "$age" }
    }
  }
])
```

Let's break down the query:

1. `use mydatabase` - Switches to the specified database. Replace "mydatabase" with the name of your actual database.

2. `db.mycollection.aggregate([...])` - Executes an aggregation pipeline on the "mycollection" collection. Replace "mycollection" with the name of your actual collection.

3. `$group` stage - Groups documents together for aggregation. Here, we use `_id: null` to group all documents into a single group.

4. `totalSum` - Calculates the sum of the "quantity" field in all documents using the `$sum` expression.

5. `average` - Calculates the average of the "price" field in all documents using the `$avg` expression.

6. `minimum` - Finds the minimum value of the "age" field across all documents using the `$min` expression.

7. `maximum` - Finds the maximum value of the "age" field across all documents using the `$max` expression.

By running this query, you will receive the total sum, average, minimum, and maximum values based on the specified fields in your collection.

b Write a MongoDB query to use push and addToSet expression.

Solution

To use the `push` and `addToSet` expressions in MongoDB, you can leverage them within update operations. Here's an example query that demonstrates their usage:

```shell
use mydatabase
```

```
db.mycollection.updateOne(
  { _id: ObjectId("document_id") },
  {
    $push: { items: "new_item" },
    $addToSet: { uniqueItems: "new_item" }
  }
)
```

Let's break down the query:

1. `use mydatabase` - Switches to the specified database. Replace "mydatabase" with the name of your actual database.

2. `db.mycollection.updateOne(...)` - Updates a single document in the "mycollection" collection. Replace "mycollection" with the name of your actual collection.

3. `{ _id: ObjectId("document_id") }` - Specifies the filter condition to find the document you want to update. In this example, we use the document's `_id` field and its corresponding ObjectId.

4. `$push` - Adds an element to an array field. Here, we use it to push the value "new_item" into the "items" array field.

5. `$addToSet` - Adds an element to an array field only if it doesn't already exist. In this example, we add "new_item" to the "uniqueItems" array field.

By executing this query, you will update a specific document in the collection. The `$push` expression adds the specified value to the "items" array, even if it already exists. Conversely, the `$addToSet` expression

ensures that "new_item" is added to the "uniqueItems" array only if it's not already present.

Remember to replace "mydatabase" with the name of your actual database, "mycollection" with the name of your collection, and "document_id" with the ObjectId of the document you want to update.

c Write a MongoDB query to use first and last expression.

Solution

In MongoDB, there is no specific operator or expression called "first" or "last" in the same context as sum, avg, min, and max. However, you can achieve similar functionality using the aggregation framework along with other operators. Here's an example query to demonstrate finding the first and last values in a collection:

```shell
use mydatabase
db.mycollection.aggregate([
  {
    $sort: { _id: 1 }
  },
  {
    $group: {
      _id: null,
      firstValue: { $first: "$fieldname" },
      lastValue: { $last: "$fieldname" }
    }
  }
])
```

Let's break down the query:

1. `use mydatabase` - Switches to the specified database. Replace "mydatabase" with the name of your actual database.

2. `db.mycollection.aggregate([...])` - Executes an aggregation pipeline on the "mycollection" collection. Replace "mycollection" with the name of your actual collection.

3. `$sort` stage - Sorts the documents in ascending order based on the `_id` field. You can replace `_id` with any suitable field in your collection.

4. `$group` stage - Groups documents together for aggregation. Here, we use `_id: null` to group all documents into a single group.

5. `firstValue` - Retrieves the value of the "fieldname" field from the first document within the group using the `$first` operator. Replace "fieldname" with the desired field in your collection.

6. `lastValue` - Retrieves the value of the "fieldname" field from the last document within the group using the `$last` operator. Again, replace "fieldname" with the desired field in your collection.

By running this query, you will obtain the first and last values of the specified field within the collection. Note that the order is determined by the sorting applied in the `$sort` stage.

4 Replication, Backup and Restore

a Write a MongoDB query to create Replica of existing database.

Solution

To create a replica set in MongoDB, you need to perform several steps both on the primary server and on the replica servers. Here's an overview of the process:

1. Launch MongoDB Instances:
Start MongoDB instances on separate servers or virtual machines that will act as the primary and replica servers. Make sure each server has the same MongoDB installation and configuration.

2. Configure Primary Server:
On the primary server, modify the MongoDB configuration file (usually `mongod.conf`) to include the replica set configuration. Add the following lines:

```plaintext
replication:
  replSetName: "myreplicaset"
```

Replace "myreplicaset" with the name you want to give your replica set.

3. Start Primary Server:
Start MongoDB on the primary server with the modified configuration. You can use the following command:

```shell
mongod --config /path/to/mongod.conf
```

Replace "/path/to/mongod.conf" with the actual path to your configuration file.

4. Connect to Primary Server:
Connect to the primary server using the MongoDB shell:

```shell
mongo
```

5. Initialize Replica Set:
In the MongoDB shell, initiate the replica set using the `rs.initiate()` command:

```shell
rs.initiate()
```

This command will set up the replica set with the primary server.

6. Add Replica Servers:
Connect to each replica server individually using the MongoDB shell. For each replica server, issue the `rs.add()` command to add it to the replica set. Here's an example:

```shell
rs.add("replica_server_address:port")
```

Replace "replica_server_address" with the IP address or hostname of the replica server and "port" with the MongoDB port number (usually 27017).

Repeat this step for each replica server you want to add.

7. Verify Replica Set:
To verify the status of the replica set, use the `rs.status()` command in the MongoDB shell:

```shell
rs.status()
```

This command will display information about the replica set, including the primary server and replica servers.

b Write a MongoDB query to create a backup of existing database.

Solution

To create a backup of an existing MongoDB database, you can use the `mongodump` tool provided by MongoDB. The `mongodump` tool allows you to create a binary dump of your database, including all the data and indexes. Here's how you can use it:

1. Open a command-line interface or terminal.

2. Navigate to the directory where the `mongodump` tool is located. This tool is typically located in the MongoDB installation directory's "bin" folder.

3. Execute the following command to create a backup of the database:

```shell
```

mongodump --db mydatabase --out /path/to/backup/folder
```

Replace "mydatabase" with the name of the database you want to back up and "/path/to/backup/folder" with the path to the directory where you want to store the backup. Make sure the specified directory is writable by the user executing the command.

4. MongoDB will create a backup of the specified database in the provided backup folder. It will generate a separate BSON file for each collection in the database.

**c Write a MongoDB query to restore database from the backup.**

**Solution**

To restore a MongoDB database from a backup created with `mongodump`, you can use the `mongorestore` tool. The `mongorestore` tool allows you to restore a previously created backup and recreate the database and its collections. Here's how you can use it:

1. Open a command-line interface or terminal.

2. Navigate to the directory where the `mongorestore` tool is located. This tool is typically located in the MongoDB installation directory's "bin" folder.

3. Execute the following command to restore the database:

```shell
mongorestore --db mydatabase /path/to/backup/folder
```
```

Replace "mydatabase" with the name you want to give to the restored database and "/path/to/backup/folder" with the path to the backup folder where the BSON files are stored.

4. MongoDB will restore the database from the backup files and recreate the collections and their data.

By running this command, you will restore the specified database from the backup created with `mongodump`. MongoDB will recreate the database and its collections, populating them with the data from the backup files.

Note: The `mongorestore` tool assumes that the backup files are in the default BSON format created by `mongodump`. If you have a compressed or archived backup, you may need to extract it first before using `mongorestore`.

Remember to replace "mydatabase" with the desired name for the restored database and "/path/to/backup/folder" with the actual path to the backup folder.

5 Java and MongoDB

a Connecting Java with MongoDB and inserting, retrieving, updating and deleting.

Solution

To connect Java with MongoDB and perform basic CRUD operations (insert, retrieve, update, delete), you can use the official MongoDB Java Driver. Here's an example that demonstrates each operation:

1. Add MongoDB Java Driver Dependency:
First, make sure you have the MongoDB Java Driver added to your Java project. You can add it to your build tool configuration (e.g., Maven or Gradle) or include the JAR file directly in your project.

2. Connecting to MongoDB:
To connect to MongoDB from your Java code, you need to create an instance of the `MongoClient` class, specifying the connection details. Here's an example:

```java
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;

public class Main {
    public static void main(String[] args) {
        // Establish a connection to MongoDB
        MongoClient mongoClient =
MongoClients.create("mongodb://localhost:27017");

        // Perform operations...
```

```
        // Close the MongoDB client
        mongoClient.close();
    }
}
```

Replace `"mongodb://localhost:27017"` with the appropriate MongoDB connection string, including the hostname and port number.

3. Insert a Document:
To insert a document into a collection, you can use the `insertOne()` method of the `MongoCollection` class. Here's an example:

```java
import com.mongodb.client.MongoCollection;
import org.bson.Document;

// ...

// Get a reference to the collection
MongoCollection<Document> collection =
mongoClient.getDatabase("mydatabase").getCollection("mycollection");

// Create a document
Document document = new Document("name", "John").append("age", 30);

// Insert the document
collection.insertOne(document);
```

Replace `"mydatabase"` with the name of your database and `"mycollection"` with the name of the collection where you want to insert the document.

4. Retrieve Documents:
To retrieve documents from a collection, you can use the `find()` method of the `MongoCollection` class. Here's an example:

```java
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCursor;

// ...

// Retrieve all documents from the collection
FindIterable<Document> documents = collection.find();

// Iterate over the retrieved documents
MongoCursor<Document> iterator = documents.iterator();
while (iterator.hasNext()) {
    Document document = iterator.next();
    // Process the document...
}
```

This code retrieves all documents from the collection and allows you to process them as needed.

5. Update a Document:
To update a document in a collection, you can use the `updateOne()` method of the `MongoCollection` class. Here's an example:

```java
import com.mongodb.client.result.UpdateResult;
import org.bson.Document;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;

// ...

// Update the document with name "John"
UpdateResult updateResult = collection.updateOne(Filters.eq("name",
"John"), Updates.set("age", 35));
```

This code updates the "age" field of the document with name "John" to 35.

6. Delete a Document:
To delete a document from a collection, you can use the `deleteOne()` method of the `MongoCollection` class. Here's an example:

```java
import com.mongodb.client.result.DeleteResult;
import com.mongodb.client.model.Filters;

// ...

// Delete the document with name "John"
DeleteResult deleteResult = collection.deleteOne(Filters.eq("name",
"John"));
```

This code deletes the document with name "John" from the collection.

Remember to handle exceptions appropriately and adjust the code according to your specific requirements, including field names, database names, and collection names.

6 PHP and MongoDB

a Connecting PHP with MongoDB and inserting, retrieving, updating and
Deleting.

Solution

To connect PHP with MongoDB and perform basic CRUD operations (insert, retrieve, update, delete), you can use the MongoDB extension for PHP. Here's an example that demonstrates each operation:

1. Install MongoDB Extension:
First, ensure that the MongoDB extension is installed and enabled in your PHP environment. You can refer to the MongoDB extension documentation for installation instructions specific to your system.

2. Connecting to MongoDB:
To connect to MongoDB from your PHP code, you need to create an instance of the `MongoDB\Client` class, specifying the connection details. Here's an example:

```php
<?php

require 'vendor/autoload.php'; // Include the MongoDB PHP library

// Establish a connection to MongoDB
$client = new MongoDB\Client("mongodb://localhost:27017");

// Select the database and collection
$database = $client->mydatabase;
```

```
$collection = $database->mycollection;

// Perform operations...

?>
```

Replace `"mongodb://localhost:27017"` with the appropriate MongoDB connection string, including the hostname and port number.

3. Insert a Document:
To insert a document into a collection, you can use the `insertOne()` method of the `MongoDB\Collection` class. Here's an example:

```php
<?php

// Create a document
$document = [
    'name' => 'John',
    'age' => 30
];

// Insert the document
$insertOneResult = $collection->insertOne($document);

?>
```

4. Retrieve Documents:
To retrieve documents from a collection, you can use the `find()` method of the `MongoDB\Collection` class. Here's an example:

```php
<?php

// Retrieve all documents from the collection
$documents = $collection->find();

// Iterate over the retrieved documents
foreach ($documents as $document) {
    // Process the document...
}

?>
```

This code retrieves all documents from the collection and allows you to process them as needed.

5. Update a Document:
To update a document in a collection, you can use the `updateOne()` method of the `MongoDB\Collection` class. Here's an example:

```php
<?php

// Update the document with name "John"
$updateResult = $collection->updateOne(['name' => 'John'], ['$set' => ['age' => 35]]);

?>
```

This code updates the "age" field of the document with name "John" to 35.

6. Delete a Document:
To delete a document from a collection, you can use the `deleteOne()` method of the `MongoDB\Collection` class. Here's an example:

```php
<?php

// Delete the document with name "John"
$deleteResult = $collection->deleteOne(['name' => 'John']);

?>
```

This code deletes the document with name "John" from the collection.

Remember to handle exceptions appropriately and adjust the code according to your specific requirements, including field names, database names, and collection names. Additionally, make sure to include the necessary autoload file and install any required dependencies for the MongoDB PHP library.

7 Python and MongoDB

a Connecting Python with MongoDB and inserting, retrieving, updating and deleting

Solution

To connect Python with MongoDB and perform basic CRUD operations (insert, retrieve, update, delete), you can use the PyMongo library, which is the official MongoDB driver for Python. Here's an example that demonstrates each operation:

1. Install PyMongo:
First, ensure that the PyMongo library is installed. You can install it using pip:

```shell
pip install pymongo
```

2. Connecting to MongoDB:
To connect to MongoDB from your Python code, you need to create an instance of the `pymongo.MongoClient` class, specifying the connection details. Here's an example:

```python
from pymongo import MongoClient

# Establish a connection to MongoDB
client = MongoClient("mongodb://localhost:27017")

# Select the database and collection
```

```
database = client["mydatabase"]
collection = database["mycollection"]

# Perform operations...
```

Replace `"mongodb://localhost:27017"` with the appropriate MongoDB connection string, including the hostname and port number.

3. Insert a Document:
To insert a document into a collection, you can use the `insert_one()` method of the `pymongo.collection.Collection` class. Here's an example:

```python
# Create a document
document = {
    "name": "John",
    "age": 30
}

# Insert the document
inserted_document = collection.insert_one(document)
```

4. Retrieve Documents:
To retrieve documents from a collection, you can use the `find()` method of the `pymongo.collection.Collection` class. Here's an example:

```python
# Retrieve all documents from the collection
documents = collection.find()
```

```
# Iterate over the retrieved documents
for document in documents:
    # Process the document...
```

This code retrieves all documents from the collection and allows you to process them as needed.

5. Update a Document:
To update a document in a collection, you can use the `update_one()` method of the `pymongo.collection.Collection` class. Here's an example:

```python
# Update the document with name "John"
update_result = collection.update_one({"name": "John"}, {"$set": {"age": 35}})
```

This code updates the "age" field of the document with name "John" to 35.

6. Delete a Document:
To delete a document from a collection, you can use the `delete_one()` method of the `pymongo.collection.Collection` class. Here's an example:

```python
# Delete the document with name "John"
delete_result = collection.delete_one({"name": "John"})
```

This code deletes the document with name "John" from the collection.

Remember to handle exceptions appropriately and adjust the code according to your specific requirements, including field names, database names, and collection names.

8 Programs on Basic jQuery

a jQuery Basic, jQuery Events

Solution

Certainly! Here are a couple of basic jQuery examples that demonstrate the usage of jQuery and jQuery events:

1. Basic jQuery Example:
This example shows how to use jQuery to manipulate the HTML elements on a web page.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("button").click(function() {
        $("p").toggle();
      });
    });
  </script>
</head>
<body>
  <button>Toggle Paragraph</button>
  <p>This is a paragraph.</p>
</body>
</html>
```

In this example, when the button is clicked, the paragraph toggles its visibility using the `toggle()` function provided by jQuery.

2. jQuery Events Example:
This example demonstrates how to handle events using jQuery.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("button").click(function() {
        $("p").text("Button clicked!");
      });
      $("p").mouseenter(function() {
        $(this).css("background-color", "yellow");
      });
      $("p").mouseleave(function() {
        $(this).css("background-color", "white");
      });
    });
  </script>
</head>
<body>
  <button>Click Me</button>
  <p>Hover over this paragraph.</p>
</body>
</html>
```

```

In this example, when the button is clicked, the text inside the paragraph is changed. Additionally, when the mouse enters the paragraph, its background color changes to yellow, and when the mouse leaves the paragraph, the background color changes back to white. These effects are achieved using jQuery event handlers such as `click()`, `mouseenter()`, and `mouseleave()`.

These examples illustrate the basic usage of jQuery and jQuery events. You can further explore and enhance them by trying out different jQuery functions and event handlers to achieve the desired behavior on your web page.

**b jQuery Selectors, jQuery Hide and Show effects**

**Solution**

Certainly! Here are a couple of examples that demonstrate jQuery selectors and the hide/show effects:

1. jQuery Selectors Example:
This example showcases different jQuery selectors to target specific HTML elements.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
```
```

```
    // jQuery code here
    $("#myButton").click(function() {
      $(".myParagraph").toggle();
    });
  });
  </script>
  <style>
    .myParagraph {
      display: none;
    }
  </style>
</head>
<body>
  <button id="myButton">Toggle Paragraphs</button>
  <p class="myParagraph">Paragraph 1</p>
  <p class="myParagraph">Paragraph 2</p>
  <p>Paragraph 3</p>
</body>
</html>
```

In this example, when the button with the ID "myButton" is clicked, the paragraphs with the class "myParagraph" toggle their visibility. The `$("#myButton")` selector targets the button with the ID "myButton", while `$(".myParagraph")` selector targets all elements with the class "myParagraph". The `toggle()` function is used to show or hide the selected elements.

2. jQuery Hide and Show Effects Example:
This example showcases the hide and show effects provided by jQuery.

```html
```

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#hideButton").click(function() {
        $(".myDiv").hide(1000);
      });
      $("#showButton").click(function() {
        $(".myDiv").show(1000);
      });
    });
  </script>
  <style>
    .myDiv {
      background-color: lightblue;
      width: 200px;
      height: 100px;
      display: block;
      margin-bottom: 10px;
    }
  </style>
</head>
<body>
  <button id="hideButton">Hide Divs</button>
  <button id="showButton">Show Divs</button>
  <div class="myDiv"></div>
  <div class="myDiv"></div>
</body>
</html>
```

```

In this example, when the "Hide Divs" button is clicked, the div elements with the class "myDiv" are hidden with a fading effect over a duration of 1000 milliseconds (1 second). Similarly, when the "Show Divs" button is clicked, the div elements are shown with a fading effect. The `hide()` and `show()` functions are used with the specified duration to achieve the effects.

**c jQuery fading effects, jQuery Sliding effects**

**Solution**

Certainly! Here are examples that demonstrate jQuery fading effects and sliding effects:

1. jQuery Fading Effects Example:
This example showcases the fading effects provided by jQuery.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#fadeInButton").click(function() {
        $(".myDiv").fadeIn(1000);
      });
      $("#fadeOutButton").click(function() {
```
```

```
      $(".myDiv").fadeOut(1000);
    });
    $("#fadeToggleButton").click(function() {
      $(".myDiv").fadeToggle(1000);
    });
  });
 </script>
 <style>
  .myDiv {
    background-color: lightblue;
    width: 200px;
    height: 100px;
    display: none;
    margin-bottom: 10px;
  }
 </style>
</head>
<body>
 <button id="fadeInButton">Fade In</button>
 <button id="fadeOutButton">Fade Out</button>
 <button id="fadeToggleButton">Toggle Fade</button>
 <div class="myDiv"></div>
 <div class="myDiv"></div>
</body>
</html>
```

In this example, when the "Fade In" button is clicked, the div elements with the class "myDiv" are faded in gradually over a duration of 1000 milliseconds (1 second). Similarly, when the "Fade Out" button is clicked, the div elements are faded out. The "Toggle Fade" button toggles the

fading effect. The `fadeIn()`, `fadeOut()`, and `fadeToggle()` functions are used with the specified duration to achieve the effects.

2. jQuery Sliding Effects Example:
This example demonstrates the sliding effects provided by jQuery.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#slideUpButton").click(function() {
        $(".myDiv").slideUp(1000);
      });
      $("#slideDownButton").click(function() {
        $(".myDiv").slideDown(1000);
      });
      $("#slideToggleButton").click(function() {
        $(".myDiv").slideToggle(1000);
      });
    });
  </script>
  <style>
    .myDiv {
      background-color: lightblue;
      width: 200px;
      height: 100px;
      display: none;
      margin-bottom: 10px;
```

```
    }
  </style>
</head>
<body>
  <button id="slideUpButton">Slide Up</button>
  <button id="slideDownButton">Slide Down</button>
  <button id="slideToggleButton">Toggle Slide</button>
  <div class="myDiv"></div>
  <div class="myDiv"></div>
</body>
</html>
```

In this example, when the "Slide Up" button is clicked, the div elements with the class "myDiv" slide up and hide gradually over a duration of 1000 milliseconds (1 second). When the "Slide Down" button is clicked, the div elements slide down and become visible. The "Toggle Slide" button toggles the sliding effect. The `slideUp()`, `slideDown()`, and `slideToggle()` functions are used with the specified duration to achieve the effects.

9 jQuery Advanced

a jQuery Animation effects, jQuery Chaining

Solution

Certainly! Here are examples that demonstrate jQuery animation effects and jQuery chaining:

1. jQuery Animation Effects Example:
This example showcases animation effects provided by jQuery.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#animateButton").click(function() {
        $(".myDiv").animate({
          width: "300px",
          height: "200px",
          opacity: 0.5
        }, 1000);
      });
    });
  </script>
  <style>
    .myDiv {
      background-color: lightblue;
      width: 100px;
```

```
      height: 100px;
      margin-bottom: 10px;
    }
  </style>
</head>
<body>
  <button id="animateButton">Animate</button>
  <div class="myDiv"></div>
</body>
</html>
```

In this example, when the "Animate" button is clicked, the div element with the class "myDiv" gradually changes its width, height, and opacity over a duration of 1000 milliseconds (1 second). The `animate()` function is used to apply the animation effect by specifying the CSS properties to animate and their target values.

2. jQuery Chaining Example:
This example demonstrates the concept of chaining in jQuery.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#myButton").css("color", "blue").slideUp(2000).slideDown(2000);
    });
  </script>
```

```
</head>
<body>
  <button id="myButton">Click Me</button>
</body>
</html>
```

In this example, the `css()` function is used to change the color of the button with the ID "myButton" to blue. Then, the `slideUp()` function is called with a duration of 2000 milliseconds (2 seconds), followed by the `slideDown()` function with the same duration. These functions are chained together, allowing multiple operations to be performed on the same element in a concise and sequential manner.

By using animation effects and chaining, you can create visually appealing and interactive web experiences using jQuery.

b jQuery Callback, jQuery Get and Set Contents

Solution

Certainly! Here are examples that demonstrate jQuery callbacks and how to get and set contents using jQuery:

1. jQuery Callback Example:
This example showcases the usage of a callback function in jQuery.

```html
<!DOCTYPE html>
<html>
<head>
```

```
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#myButton").click(function() {
        $("#myDiv").fadeOut(1000, function() {
          alert("Fade out completed!");
        });
      });
    });
  </script>
  <style>
    #myDiv {
      background-color: lightblue;
      width: 200px;
      height: 100px;
    }
  </style>
</head>
<body>
  <button id="myButton">Fade Out</button>
  <div id="myDiv"></div>
</body>
</html>
```

In this example, when the "Fade Out" button is clicked, the div element with the ID "myDiv" gradually fades out over a duration of 1000 milliseconds (1 second). The fade-out effect is performed, and once completed, the provided callback function is executed. In this case, an alert is shown with the message "Fade out completed!"

2. jQuery Get and Set Contents Example:
This example demonstrates how to get and set the contents of HTML elements using jQuery.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#getTextButton").click(function() {
        var text = $("#myParagraph").text();
        alert("Text: " + text);
      });

      $("#setTextButton").click(function() {
        var newText = "New text content";
        $("#myParagraph").text(newText);
      });
    });
  </script>
</head>
<body>
  <button id="getTextButton">Get Text</button>
  <button id="setTextButton">Set Text</button>
  <p id="myParagraph">Original text content</p>
</body>
</html>
```

In this example, the "Get Text" button retrieves the text content of the paragraph element with the ID "myParagraph" using the `text()` function and displays an alert with the retrieved text. The "Set Text" button sets a new text content ("New text content") to the paragraph element using the `text()` function.

By utilizing callbacks and get/set methods, you can enhance the functionality of your jQuery code and manipulate the contents of HTML elements as needed.

c jQuery Insert Content, jQuery Remove Elements and Attribute

Solution

Certainly! Here are examples that demonstrate jQuery's insert content, remove elements, and attribute manipulation capabilities:

1. jQuery Insert Content Example:
This example showcases different methods to insert content into HTML elements using jQuery.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#appendButton").click(function() {
        $(".myDiv").append("<p>New content appended</p>");
      });
```

```
    $("#prependButton").click(function() {
      $(".myDiv").prepend("<p>New content prepended</p>");
    });

    $("#beforeButton").click(function() {
      $(".myDiv").before("<div>New content before</div>");
    });

    $("#afterButton").click(function() {
      $(".myDiv").after("<div>New content after</div>");
    });
  });
  </script>
</head>
<body>
  <button id="appendButton">Append</button>
  <button id="prependButton">Prepend</button>
  <button id="beforeButton">Before</button>
  <button id="afterButton">After</button>
  <div class="myDiv">
    Existing content
  </div>
</body>
</html>
```

In this example, different buttons trigger various jQuery methods to insert content into the `div` element with the class "myDiv". The `append()`, `prepend()`, `before()`, and `after()` functions are used to add new paragraphs and divs to the element, either as appended, prepended, or positioned before or after the existing content.

2. jQuery Remove Elements and Attribute Example:
This example demonstrates how to remove elements and manipulate attributes using jQuery.

```html
<!DOCTYPE html>
<html>
<head>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script>
    $(document).ready(function() {
      // jQuery code here
      $("#removeButton").click(function() {
        $(".myDiv").remove();
      });

      $("#emptyButton").click(function() {
        $(".myDiv").empty();
      });

      $("#attrButton").click(function() {
        var href = $("a").attr("href");
        alert("Link href: " + href);

        $("a").attr("href", "https://www.example.com");
        $("a").text("Example Website");
      });
    });
  </script>
</head>
<body>
```

```
  <button id="removeButton">Remove</button>
  <button id="emptyButton">Empty</button>
  <button id="attrButton">Change Attribute</button>
  <div class="myDiv">
    <p>This div will be removed or emptied</p>
  </div>
  <a href="https://www.google.com">Google</a>
</body>
</html>
```

In this example, clicking the "Remove" button removes the `div` element with the class "myDiv" from the DOM using the `remove()` function. Clicking the "Empty" button empties the content inside the `div` using the `empty()` function.

The "Change Attribute" button demonstrates manipulating attributes. Initially, it retrieves the `href` attribute of the `a` element using the `attr()` function and displays an alert. Then, it updates the `href` attribute to "https://www.example.com" and changes the text of the `a` element to "Example Website" using the `attr()` and `text()` functions, respectively.

By utilizing jQuery's content insertion, element removal, and attribute manipulation functionalities, you can dynamically modify the content and structure of your web page.

10 JSON

a Creating JSON

Solution

To create JSON (JavaScript Object Notation) data, you can use different programming languages. Here are examples in Python, JavaScript, and PHP:

1. Creating JSON in Python:
In Python, you can use the built-in `json` module to create JSON data. Here's an example:

```python
import json

# Create a dictionary
data = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Convert the dictionary to JSON
json_data = json.dumps(data)

print(json_data)
```

This code creates a dictionary `data` and then converts it to JSON using the `json.dumps()` function. The JSON data is stored in the `json_data` variable and can be printed or used as needed.

2. Creating JSON in JavaScript:
In JavaScript, you can create JSON directly using object literals. Here's an example:

```javascript
// Create an object
var data = {
    name: "John",
    age: 30,
    city: "New York"
};

// Convert the object to JSON
var json_data = JSON.stringify(data);

console.log(json_data);
```

This code creates an object `data` and then uses the `JSON.stringify()` function to convert it to JSON. The JSON data is stored in the `json_data` variable and can be logged to the console or used as needed.

3. Creating JSON in PHP:
In PHP, you can use the `json_encode()` function to create JSON data. Here's an example:

```php
// Create an associative array
$data = array(
    "name" => "John",
    "age" => 30,
```

```
    "city" => "New York"
);

// Convert the array to JSON
$json_data = json_encode($data);

echo $json_data;
```

This code creates an associative array `$data` and then uses the `json_encode()` function to convert it to JSON. The JSON data is echoed and can be used as needed.

In all three examples, you can modify the data and structure of the objects or arrays to suit your specific needs. JSON is a lightweight and widely used data interchange format, and these examples provide a basic understanding of how to create JSON in different programming languages.

b Parsing JSON

Solution

To parse JSON (JavaScript Object Notation) data, you can use different programming languages. Here are examples in Python, JavaScript, and PHP:

1. Parsing JSON in Python:
In Python, you can use the built-in `json` module to parse JSON data. Here's an example:

```python
import json
```

```
# JSON data
json_data = '{"name": "John", "age": 30, "city": "New York"}'

# Parse JSON
data = json.loads(json_data)

# Access parsed data
name = data["name"]
age = data["age"]
city = data["city"]

print(name, age, city)
```

This code demonstrates parsing JSON using the `json.loads()` function in Python. The JSON data is stored as a string in the `json_data` variable. After parsing, the data can be accessed as a dictionary.

2. Parsing JSON in JavaScript:
In JavaScript, you can use the `JSON.parse()` function to parse JSON data. Here's an example:

```javascript
// JSON data
var json_data = '{"name": "John", "age": 30, "city": "New York"}';

// Parse JSON
var data = JSON.parse(json_data);

// Access parsed data
var name = data.name;
```

```
var age = data.age;
var city = data.city;

console.log(name, age, city);
```

This code demonstrates parsing JSON using the `JSON.parse()` function in JavaScript. The JSON data is stored as a string in the `json_data` variable. After parsing, the data can be accessed as an object.

3. Parsing JSON in PHP:
In PHP, you can use the `json_decode()` function to parse JSON data. Here's an example:

```php
// JSON data
$json_data = '{"name": "John", "age": 30, "city": "New York"}';

// Parse JSON
$data = json_decode($json_data);

// Access parsed data
$name = $data->name;
$age = $data->age;
$city = $data->city;

echo $name, $age, $city;
```

This code demonstrates parsing JSON using the `json_decode()` function in PHP. The JSON data is stored as a string in the `$json_data` variable.

After parsing, the data can be accessed as an object or an associative array.

In all three examples, you can modify the JSON data or access nested properties based on your specific needs. JSON parsing allows you to extract and work with the structured data contained in a JSON string.

c Persisting JSON

Solution

To persist JSON (JavaScript Object Notation) data, you have various options based on your use case. Here are some common methods to persist JSON data:

1. File Storage:
You can store JSON data in files on the server or local filesystem. You can write the JSON data to a file when saving and read from the file when retrieving. Here's an example in Python:

```python
import json

# JSON data
data = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Write JSON data to a file
with open("data.json", "w") as file:
```

```
    json.dump(data, file)

# Read JSON data from the file
with open("data.json", "r") as file:
    data = json.load(file)
```

In this example, the `json.dump()` function is used to write the JSON data to a file, and the `json.load()` function is used to read the JSON data from the file.

2. Database:
You can persist JSON data in databases that support JSON storage. For example, MongoDB, PostgreSQL, and MySQL have built-in support for storing and querying JSON data. Here's an example using MongoDB and the PyMongo library in Python:

```python
from pymongo import MongoClient
import json

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017")

# Access the database and collection
db = client["mydatabase"]
collection = db["mycollection"]

# JSON data
data = {
    "name": "John",
    "age": 30,
```

```
    "city": "New York"
}

# Insert JSON data into the collection
collection.insert_one(data)

# Retrieve JSON data from the collection
result = collection.find_one()
```

In this example, the JSON data is inserted into a collection in MongoDB using the `insert_one()` method, and it is retrieved using the `find_one()` method.

3. API and Web Services:
You can send JSON data to an API or web service for persistence. This method is commonly used for communication between client-side applications and server-side APIs. The JSON data is sent as part of the request body and processed by the server to store in the desired storage mechanism.

Remember to handle security measures, such as authentication and data validation, when persisting JSON data in any of these methods.

11 Create a JSON file and import it to MongoDB

a Export MongoDB to JSON.

Solution

To export data from MongoDB to a JSON file, you can use the `mongoexport` command-line tool provided by MongoDB. Here's an example of how to export data from a MongoDB collection to a JSON file:

1. Open a terminal or command prompt.

2. Use the `mongoexport` command with the appropriate options to export the data. The basic syntax is as follows:

```shell
mongoexport --uri=<MongoDB connection URI>
--collection=<collection_name> --out=<output_file.json> --jsonArray
```

Replace `<MongoDB connection URI>` with the URI for your MongoDB server, `<collection_name>` with the name of the collection you want to export, and `<output_file.json>` with the name and path of the output JSON file.

The `--jsonArray` option ensures that the exported data is stored as a JSON array in the output file.

Here's an example command:

```shell
mongoexport --uri=mongodb://localhost:27017 --collection=mycollection
--out=output.json --jsonArray
```

This command exports the data from the "mycollection" collection in the local MongoDB server and saves it to the "output.json" file as a JSON array.

3. Execute the command, and the data will be exported to the specified JSON file.

Once the export is completed, you will have a JSON file containing the exported data from MongoDB.

You can then use this JSON file to import the data into another MongoDB database or use it for other purposes.

b Write a MongoDB query to delete JSON object from MongoDB

Solution

To delete a JSON object from MongoDB, you can use the `deleteOne()` or `deleteMany()` methods provided by the MongoDB driver. Here's an example of how to delete a JSON object from a collection in MongoDB:

```javascript
// Assuming you have connected to the MongoDB server and selected the database and collection

// Delete a single JSON object
db.collectionName.deleteOne({ field: value });

// Delete multiple JSON objects matching a criteria
```

```
db.collectionName.deleteMany({ field: value });
```

Replace `collectionName` with the actual name of your collection. In the delete operation, you need to specify the criteria to identify the JSON object(s) you want to delete.

To delete a single JSON object, use the `deleteOne()` method and provide a query object that matches the JSON object you want to delete. For example:

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

This command deletes the first document in the "myCollection" collection where the "name" field is equal to "John".

To delete multiple JSON objects that match a specific criteria, use the `deleteMany()` method and provide a query object that matches the JSON objects you want to delete. For example:

```javascript
db.myCollection.deleteMany({ age: { $gt: 30 } });
```

This command deletes all documents in the "myCollection" collection where the "age" field is greater than 30.

Make sure to adjust the field and value according to your specific JSON object and collection structure.

Note: Deleting JSON objects from a MongoDB collection is a permanent operation. Please use caution and ensure that you have a backup of your data before performing deletion operations.