JSON handling:-

```python
import json

file_path = 'mobile.json'
with open(file_path, 'r') as file:
    data = json.load(file)
if "users" in data:
    users_data = data["users"]
    for user_id, user_info in users_data.items():
        print(f"User ID: {user_id}")
        print(f"Name: {user_info['name']}")
        print(f"Email: {user_info['email']}")
        print(f"Phone: {user_info['phone']}")
        address = user_info.get('address', {})
        print("Address:")
        print(f"  Street: {address.get('street', 'N/A')}")
        print(f"  City: {address.get('city', 'N/A')}")
        print(f"  State: {address.get('state', 'N/A')}")
        print(f"  ZIP: {address.get('zip', 'N/A')}")
        is_active = user_info.get('is_active', False)
        print(f"Active Status: {'Active' if is_active else 'Inactive'}")
        print("------------------------------")
else:
    print("No user data found in the JSON file.")
```

```json
{
    "users": {
      "user123": {
        "name": "John Doe",
        "email": "john.doe@example.com",
        "phone": "123-456-7890",
        "address": {
          "street": "123 Main St",
          "city": "Anytown",
          "state": "CA",
          "zip": "12345"
        },
        "is_active": true
      },
      "user456": {
        "name": "Jane Smith",
        "email": "jane.smith@example.com",
        "phone": "987-654-3210",
        "address": {
          "street": "456 Oak St",
          "city": "Sometown",
          "state": "NY",
```

```
          "zip": "56789"
        },
        "is_active": false
      },
      "user789": {
        "name": "Alice Johnson",
        "email": "alice.johnson@example.com",
        "phone": "111-222-3333",
        "address": {
          "street": "789 Pine St",
          "city": "Anothercity",
          "state": "TX",
          "zip": "67890"
        },
        "is_active": true
      },
      "user202": {
        "name": "Eva Brown",
        "email": "eva.brown@example.com",
        "phone": "777-888-9999",
        "address": {
          "street": "202 Maple St",
          "city": "Lastcity",
          "state": "WA",
          "zip": "89012"
        },
        "is_active": false
      }
    }
  }
```

XML:-

```
<library>
    <book>
        <title>The Secret</title>
        <author>Rhonda Byrne</author>
        <genre>Self-help</genre>
    </book>
    <book>
        <title>The Alchemist</title>
        <author>Paulo Coelho</author>
        <genre>Self-help</genre>
    </book>
    <book>
        <title>Psychology of Money</title>
        <author>Morgan Housel</author>
```

```xml
        <genre>Finance</genre>
    </book>
    <book>
        <title>Kite Runner</title>
        <author>Khaled Hosseini</author>
        <genre>Fiction</genre>
    </book>
    <book>
        <title>Steve Jobs: A Biography</title>
        <author>Walter Isaacson</author>
        <genre>Biography</genre>
    </book>
    <book>
        <title>The Innovators</title>
        <author>Walter Isaacson</author>
        <genre>Non-Fiction</genre>
    </book>
</library>
```

```python
from lxml import etree

# Parse the XML file
tree = etree.parse('data.xml')

# Execute an XPath query to get all book titles
titles = tree.xpath('//book/title/text()')
print("Titles of books:\n")
for title in titles:
    print(title)

# Execute an XPath query to get all books in the 'Biography' genre
biography_books = tree.xpath("//book[genre='Biography']")
print("\nData Science Books:")
for book in biography_books:
    print(f"Title: {book.find('title').text}, Author:
{book.find('author').text}, Genre: {book.find('genre').text}")

# Execute an XPath query to get all books written by 'Khaled Hosseini'
author = tree.xpath("//book[author='Khaled Hosseini']")
print("\nBooks by Khaled Hosseini:\n")
for book in author:
    print(f"Title: {book.find('title').text} \nAuthor:
{book.find('author').text}\nGenre: {book.find('genre').text}\n")

# Execute an XPath query to get books in the 'Biography' genre written by
author 'Paulo Coelho'
biography_books_paulocoelho = tree.xpath("//book[genre='Self-help]" and
"//book[author ='Paulo Coelho']")
```

```
print("\nBiography Books by Paulo Coelho:\n")
for book in biography_books_paulocoelho:
    print(f"Title: {book.find('title').text}\n")
```

Bplus Trees:-

```
import tempfile
import time
from bplustree import BPlusTree

def print_menu():
    print("\n******** B+ Tree ********")
    print("1. Insert")
    print("2. Search")
    print("3. Exit")
    return input("Enter your choice: ")

temp_file = tempfile.NamedTemporaryFile().name

class SimpleSerializer:
    def serialize(self, obj, size):
        return obj.ljust(size, b'\0')

bplus_tree = BPlusTree(order=4, filename=temp_file,
serializer=SimpleSerializer())

while True:
    choice = print_menu()

    if choice == '1':
        value = int(input("Enter Element to Insert : "))
        # Convert the integer to bytes
        value_bytes = str(value).encode('utf-8')
        bplus_tree[value_bytes] = value_bytes
        print("Element inserted into the B+ tree.")
    elif choice == '2':
        value = int(input("Enter Element to Search : "))
        # Convert the integer to bytes for search
        value_bytes = str(value).encode('utf-8')

        # Measure the time taken to search
        start_time = time.time()
        if value_bytes in bplus_tree:
            print("Value found in the tree")
        else:
            print("Value not found in the tree")
        end_time = time.time()
```

```
        print("Time taken to search: {} seconds".format(end_time -
start_time))
    elif choice == '3':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please enter a valid option.")

# Close the tree after use
bplus_tree.close()
```

BTree:-

```
from bintrees import FastRBTree
import time

def print_menu():
    print("\n******** B Tree ********")
    print("1. Insert")
    print("2. Search")
    # print("3. Show Tree")
    print("3. Exit")
    return input("Enter your choice: ")

# Create an empty B tree
bplus_tree = FastRBTree()

while True:
    choice = print_menu()

    if choice == '1':
        value = int(input("Enter Element to Insert : "))
        # Insert into the B tree
        bplus_tree.insert(value, value)
        print("Element inserted into the Btree.")
    elif choice == '2':
        value = int(input("Enter Element to Search : "))
        start_time = time.time()  # Record start time
        # Search in the B+ tree
        if value in bplus_tree:
            print("Value found in the tree")
        else:
            print("Value not found in the tree")
        end_time = time.time()  # Record end time
        search_time = (end_time - start_time) *1000 # Convert seconds to
milliseconds
        print(f"Time taken for search: {search_time} ms")
```

```
    elif choice == '3':
        # print("B Tree:")
        # Display the B tree
        # print([value for _, value in bplus_tree.items()])

    # elif choice == '4':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please enter a valid option.")
```

2pc:-

```
n = int(input("Enter number of participants: "))
reply = []

class Coordinator():

    def phase1(self):
        for i in range(n):
            print(f"Coordinator to participant {i+1}: PREPARE")

    def voting(self):
        print("-----VOTING PHASE-----")
        for i in range(n):
            response = input(f"Enter 1 if participant {i+1} is READY and 0 if
NOT READY: ")
            reply.append(response)

    def phase2(self,participants):
        print("-----INDIVIDUAL VOTES-----")
        for i in range(n):
            if reply[i] == "1":
                participants[i].commit()
            elif reply[i] == "0":
                participants[i].abort()

    def result(self):
        print("-----DECISION PHASE-----")
        if "0" in reply:
            print("Transaction ABORTED as all participating sites NOT READY!")
        else:
            print("Transaction COMMITTED as all participating sites READY!")
        print("-----THE END-----")

class Participant:
    def __init__(self, number):
        self.number = number
```

```
    def commit(self):
        print(f"Participant {self.number}: COMMIT. Prepared")

    def abort(self):
        print(f"Participant {self.number}: ABORT. Not prepared")

participants = [Participant(i + 1) for i in range(n)]

coordinator = Coordinator()
coordinator.phase1()
coordinator.voting()
coordinator.phase2(participants)
coordinator.result()
```

Query Optimization:-

use world;

-- SELECT

-- select * from city where CountryCode="IND";

-- optimize table city;

-- Nested

-- select Name from city

-- where Population > (select AVG(Population) from city) and

-- CountryCode in (select Code from country

-- where Population > (select AVG(Population) from country));

-- optimize table city,country;

-- Right Join

-- select * from city right join country

-- on city.CountryCode in

-- (select code from country where LifeExpectancy>80.0);

-- optimise table city,country;

-- Inner Join

```
-- select * from city inner join country
-- on city.CountryCode in
-- (select code from country where LifeExpectancy>75.0);

-- Indexing
-- create index Name on country(Name);
-- select * from country where Name = "India";
```

Query Monitor:-

QEP:

```
EXPLAIN SELECT * FROM City WHERE CountryCode = 'USA';
```

Query Statistics:

```
EXPLAIN ANALYZE SELECT * FROM City WHERE CountryCode = 'USA';

SELECT * FROM performance_schema.events_statements_summary_by_digest;

SHOW STATUS WHERE `variable_name` = 'Questions';
```

Partition:-

```
create table sales_range (sales_id int not null,cust_id int not null, cust_name varchar(50), amount int) partition by range(amount)
(
 partition p0 values less than (1000),
 partition p1 values less than (2000),
 partition p2 values less than (3000),
 partition p3 values less than (4000)
);
insert into sales_range values
(1,2,"A",500),
(2,4,"B",2995),
(3,6,"C",995),
(4,8,"D",3995),
(5,5,"E",1995);
select * from sales_range;

select * from information_schema.partitions where table_name = "sales_range";

select * from sales_range partition (p0);
```

```
create table sales_list(id int not null,amount int) partition by list(id)
(
partition p_1 values in (1,5,9),
partition p_2 values in (2,6,10),
partition p_3 values in (3,7,11),
partition p_4 values in (4,8,12)
);

insert into sales_list values (1,500),(2,600),(4,900),(9,872),(3,87);

select * from information_schema.partitions where table_name = "sales_list";
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	PARTITION_NAME	SUBPARTITION_NAME	PARTITION_ORDINAL_POSITION	SUBPARTITION_ORDINAL_POSITION
def	world	sales_list	p_1	NULL	1	NULL
def	world	sales_list	p_2	NULL	2	NULL
def	world	sales_list	p_3	NULL	3	NULL
def	world	sales_list	p_4	NULL	4	NULL

```
create table sales_hash (id int not null,amount int) partition by hash(id)
partitions 4;
insert into sales_hash values
(1,500),
(2,600),
(4,900),
(9,872),
(3,87);

select * from information_schema.partitions where table_name = "sales_hash";
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	PARTITION_NAME	SUBPARTITION_NAME	PARTITION_ORDINAL_POSITION	SUBPARTITION_ORDINAL_POSITION
def	world	sales_hash	p0	NULL	1	NULL
def	world	sales_hash	p1	NULL	2	NULL
def	world	sales_hash	p2	NULL	3	NULL
def	world	sales_hash	p3	NULL	4	NULL

```sql
create table sales_key1 (id int primary key,amount int ) partition by key()
partitions 3;
insert into sales_key values (1,500),(2,600),(4,900);

select * from information_schema.partitions where table_name = "sales_key1";
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	PARTITION_NAME	SUBPARTITION_NAME	PARTITION_ORDINAL_POSITION	SUBPARTITION_ORDINAL_POSITION
def	world	sales_key1	p0	NULL	1	NULL
def	world	sales_key1	p1	NULL	2	NULL