**CHAPTER-1**

**DAY TO DAY ACTIVITIES**



**Internship Program on Python for BE-3rd Sem students**

**From 9th Sep to 28th Sep 2024 (During 3rd semester vacations).**

**Student Name: Lakshitha P USN No: 3BR23CD050 Branch: CSE-DS**

| **INDEX PAGE** | | | |
| --- | --- | --- | --- |
| **Day** | **Date** | **Content Covered** | **Signature of the** |
| **faculty in-charge** |
| **1** | **9.09.24** | **Introduction to Python, Setup & Installation, First Python Program, Variables, Data Types,**  **and Basic I/O** |  |
| **2** | **10.09.24** | **Control Structures: If-else, Loops, Functions and Modules** |  |
| **3** | **11.09.24** | **Lists, Tuples, and Dictionaries, File Handling** |  |
| **4** | **12.09.24** | **Exception Handling, Practice exercises on Python basics** |  |
| **5** | **13.09.24** | **Introduction to OOP, Classes, and Objects** |  |
| **6** | **14.09.24** | **Inheritance, Polymorphism, and Encapsulation** |  |
| **7** | **15.09.24** | **Abstract Classes and Interfaces** |  |
| **8** | **17.09.24** | **Practice exercises on OOP concepts** |  |
| **9** | **18.09.24** | **Introduction to DSA, Arrays, and Linked Lists** |  |
| **10** | **19.09.24** | **Stacks and Queues** |  |
| **11** | **20.09.24** | **Trees and Graphs** |  |
| **12** | **21.09.24** | **Searching and Sorting Algorithms** |  |
| **13** | **23.09.24** | **Project Building & Presentations** |  |
| **14** | **24.09.24** | **Project Building & Presentations** |  |
| **15** | **25.09.24** | **Project Building & Presentations** |  |
| **16** | **26.09.24** | **Project Building & Presentations** |  |
| **17** | **27.09.24** | **Project Building & Presentations** |  |
| **18** | **28.09.24** | **Project Building & Presentations** |  |

**CHAPTER-2**

**COMPANY PROFILE**

**Company Name: EZ Trainings and Technologies Pvt. Ltd.**

**Introduction:**

EZ Trainings and Technologies Pvt. Ltd. is a dynamic and innovative organization dedicated to providing comprehensive training solutions and expert development services. Established with a vision to bridge the gap between academic learning and industry requirements, we specialize in college trainings for students, focusing on preparing them for successful placements. Additionally, we excel in undertaking development projects, leveraging cutting-edge technologies to bring ideas to life.

**Mission:**

Our mission is to empower the next generation of professionals by imparting relevant skills and knowledge through specialized training programs. We strive to be a catalyst in the career growth of students and contribute to the technological advancement of businesses through our development projects.

**Services:**

**College Trainings:**

* Tailored training programs designed to enhance the employability of students.
* Industry-aligned curriculum covering technical and soft skills.
* Placement assistance and career guidance.

**Development Projects:**

* End-to-end development services, from ideation to execution.
* Expertise in diverse technologies and frameworks.
* Custom solutions to meet specific business needs.

**Locations:** Hyderabad | Delhi NCR

At EZ Trainings and Technologies Pvt. Ltd., we believe in transforming potential into excellence

**CHAPTER-3**

# ABSTRACT

* A database backup tool is an essential utility that ensures the integrity, security, and availability of data by creating periodic snapshots or copies of a database's contents.
* The first step in automating SQL database backups is to connect to the database using Python. We will use the pyodbc package to connect to the database and execute SQL commands. In this code, we create a connection object using the pyodbc. connect() method and pass in the connection parameters.
* As data is a critical asset for any organization, safeguarding it from corruption, accidental loss, or disaster scenarios is paramount. This tool facilitates the automation of database backup processes, allowing administrators to schedule backups, manage versions, and store them either locally or in remote/cloud storage systems.
* The tool typically supports various types of backups such as full, incremental, and differential, providing flexibility based on the size and criticality of the database.
* Advanced features may include encryption to protect sensitive data, compression to save storage space, and notifications to inform users about the success or failure of backup operations.
* Database backup tools are designed to be user-friendly, supporting various database management systems (DBMS) like MySQL, PostgreSQL, SQL Server, and Oracle, among others. With increasing volumes of data, the tool must efficiently manage backup files while ensuring quick recovery options in case of failure or data corruption.
* In summary, a robust database backup tool enhances data resilience, ensures compliance with industry standards, and provides peace of mind for organizations in handling their valuable data assets.
* It is important to backup the databases providing an opportunity to recover the data and be available for processing again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake. Backups are also essential as a safeguard before upgrading a SQL installation, and they could be used to transfer a SQL installation to another system. In this paper the database backup strategies and recovery models for various specific cases are discussed and explained. Some solutions for recovering and restoring of large database have suggested. Backup plan based on weekly full, daily differential and hourly transaction log backup has proposed.

# CHAPTER-4

# INTRODUCTION OF THE PROJECT

* In today's digital age, databases are critical for managing and storing data. Whether used for business processes, customer information, or application management, ensuring that data is safe from corruption or loss is essential.
* Database backups provide a mechanism to create copies of the database at a specific point in time, allowing recovery in case of system failure, accidental deletion, or other unforeseen events.
* Python has gained popularity due to its versatility, ease of use, and extensive libraries that support database management and file handling. This project focuses on leveraging Python to build a tool that automates database backups, ensures data security, and provides flexible storage options.
* In today’s digital landscape, data is a valuable asset for businesses. Losing critical data can lead to severe consequences, making regular database backups essential. However, performing backups manually can be time-consuming and error-prone. In this blog post, we will explore a Python script that automates database backups and enables additional post-backup actions.
* I have developed this python script in which you can use it automatically and also manually to take the db backup of MySQL, MongoDB and PostgreSQL and we can use certain actions like copy it to a backup directory or store it in another server or push it to S3 to store the backup and also it sends mail after the backup is taken successfully.
* Data is one of the most valuable assets for modern businesses, making database backup and recovery processes critical components of data management. Database backups are essential for safeguarding information from loss, corruption, or unforeseen incidents such as hardware failures or cyberattacks. These backups allow businesses to restore their databases to a specific point in time, ensuring business continuity and minimizing downtime.
* This project implements a Python-based Database Backup Manager using object-oriented principles and threading for efficient scheduling. The tool allows users to create, update, read, delete, and monitor backups of database systems. Additionally, it leverages multithreading to schedule backups asynchronously, ensuring non-blocking operations and improving performance.
* The backup system consists of the following key components:
* 1. Backup Record: A class that defines a backup instance, storing details such as the backup ID, database details, scheduled time, and current status.
* 2. Backup Manager: This class acts as the core manager for handling the creation, reading, updating, deleting, and scheduling of database backups.
* 3. Threading for Backup Scheduling: The tool utilizes Python's threading to schedule backups in the background, ensuring that the main application flow is uninterrupted.
* 4. Unit Testing: A comprehensive suite of unit tests is implemented using the unittest framework to ensure the accuracy and robustness of the backup manager's functionalities.
* The Backup Manager is designed to be scalable, thread-safe, and flexible, capable of handling multiple backups simultaneously while ensuring data integrity. The use of a threading lock guarantees that concurrent modifications to the backup list do not lead to race conditions, preserving the consistency of backup data.
* This document outlines the implementation of the Backup Manager, including its core functionalities, thread-safe operations, and unit testing to validate the system's behavior.

# CHAPTER-5

# MODULE DESCRIPTION

This Python module provides a simple, thread-safe Database Backup Manager system. It consists of two main classes—BackupRecord and BackupManager—that manage the creation, updating, deletion, and scheduling of database backups. Here's an overview of each component:

1. BackupRecord Class

The BackupRecord class is a data structure that represents a single backup record. It holds key information about the backup, such as:

backup\_id: A unique identifier for each backup.

db\_details: A dictionary containing the database details (e.g., name, user).

scheduled\_time: The time at which the backup is scheduled to occur.

status: The current status of the backup, initialized to "scheduled".

Methods:

\_init\_(self, backup\_id, db\_details, scheduled\_time, status="scheduled"): Constructor to initialize a backup record with the given information.

update\_status(self, new\_status): A method to update the status of the backup (e.g., from "scheduled" to "completed").

2. BackupManager Class

The BackupManager class is responsible for managing multiple BackupRecord instances and ensuring thread-safe operations through the use of a threading Lock. It provides several methods for creating, reading, updating, and deleting backups, as well as scheduling backups to run asynchronously using Python's threading capabilities.

Methods:

\_init\_(self): Initializes an empty list of backups and a thread lock to ensure safe access to shared data.

create\_backup(self, db\_details): Creates a new backup, assigns it a unique ID, schedules the backup to run one hour from the current time, and appends it to the backup list. Returns the unique backup ID.

read\_backup(self, backup\_id): Reads and retrieves a specific backup record by its ID. If not found, returns None.

update\_backup(self, backup\_id, new\_db\_details): Updates the database details of a specific backup record if it exists, and returns True if successful, otherwise False.

delete\_backup(self, backup\_id): Deletes a specific backup by its ID from the list, ensuring thread safety. Returns True if deletion was successful, otherwise False.

schedule\_database\_backups(self, db\_details): Schedules a backup to run in a separate thread, allowing the program to continue running without being blocked. Returns the backup thread.

\*\*monitor\_backup\_status(self, backup\_id)

# CHAPTER-6

# ALGORITHM

The algorithm used in the Database Backup Manager is designed to handle backup creation, updates, deletion, scheduling, and status monitoring in a thread-safe manner. The process includes the following steps and strategies:

1. Backup Creation Process

The backup creation process involves:

Input: Database details (e.g., name, user).

Output: A unique backup ID and a scheduled backup record.

Steps:

1. Generate a Unique Backup ID:

The backup ID is derived from the length of the backups list plus one. This ensures that each backup gets a unique identifier.

2. Schedule Backup Time:

The backup is scheduled to occur one hour after the current time, calculated using datetime.datetime.now() + datetime.timedelta(hours=1).

3. Create a Backup Record:

A BackupRecord object is created with the backup ID, database details, and the scheduled time.

4. Append the Backup Record:

The new backup record is appended to the backups list.

5. Thread Safety:

A Lock is used to ensure that multiple threads cannot modify the backups list at the same time. This prevents race conditions and data corruption.

2. Reading a Backup

Steps:

1. Input: Backup ID.

2. Search for the Backup:

Iterate through the list of backups to find the backup with the matching ID.

3. Return the Backup:

If found, return the corresponding BackupRecord. Otherwise, return None.

4. Thread Safety:

The method acquires a lock to ensure that reading from the shared backups list is safe in a multi-threaded environment.

3. Updating a Backup

Steps:

1. Input: Backup ID and new database details.

2. Search for the Backup:

Call the read\_backup() method to locate the backup.

3. Update the Database Details:

If the backup is found, update the db\_details with the new values.

4. Return Status:

Return True if the update was successful; otherwise, return False.

5. Thread Safety:

A lock ensures safe access to the backups list during the update operation.

4. Deleting a Backup

Steps:

1. Input: Backup ID.

2. Search for the Backup:

Iterate through the list of backups to find the one with the matching ID.

3. Remove the Backup:

If found, delete the corresponding backup from the list.

4. Return Status:

Return True if the deletion was successful; otherwise, return False.

5. Thread Safety:

A lock is used to prevent multiple threads from deleting the same backup simultaneously or from causing inconsistencies in the backup list.

5. Scheduling Database Backups

Steps:

1. Input: Database details.

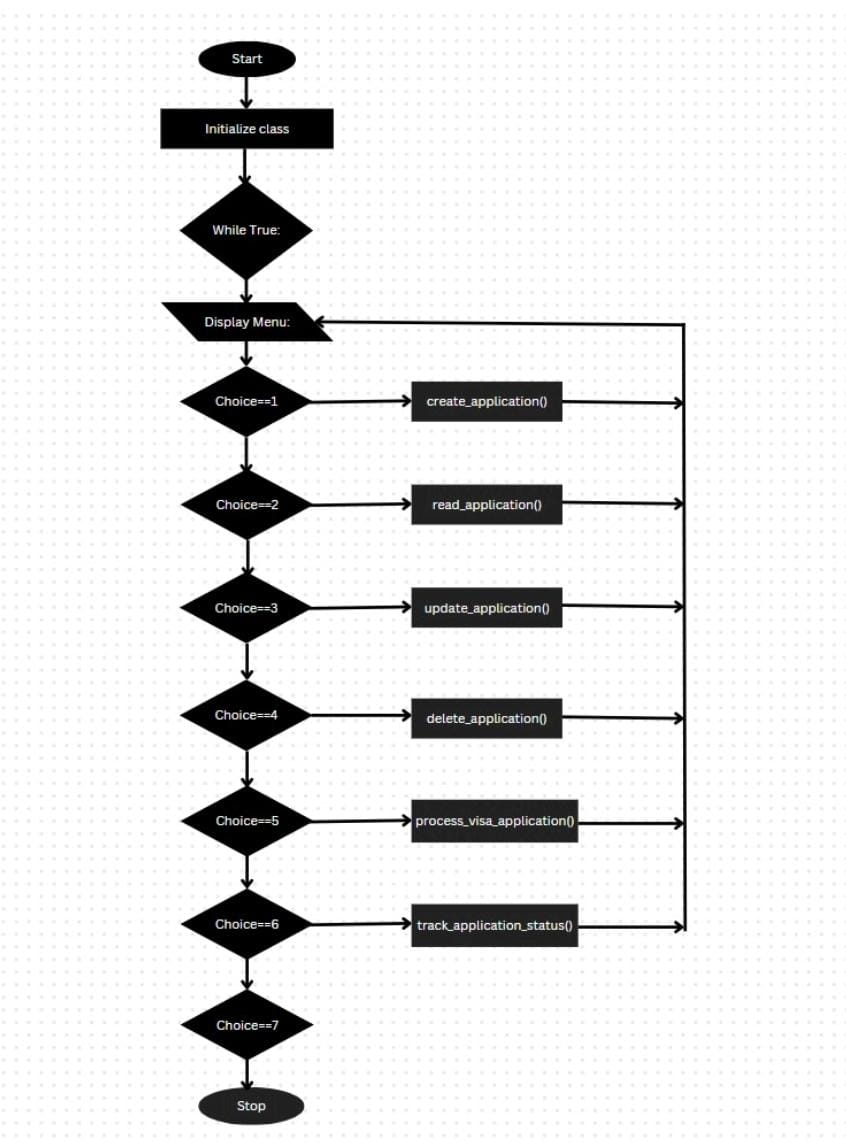
2. Create a Backup Thread:

A new thread is spawned to execute the create\_backup() method asynchronously. This allows the program to continue execution without waiting for the backup creation process to complete.

3. Return the Thread:

# CHAPTER-7

# FLOWCHART



# CHAPTER-8

# SOURCE CODE

import datetime

from threading import Thread

import unittest

import time

class BackupRecord:

    def \_init\_(self, backup\_id, db\_details, scheduled\_time, status="scheduled"):

        self.backup\_id = backup\_id

        self.db\_details = db\_details

        self.scheduled\_time = scheduled\_time

        self.status = status

    def update\_status(self, new\_status):

        self.status = new\_status

class BackupManager:

    def \_init\_(self):

        self.backups = []

    def create\_backup(self, db\_details):

        backup\_id = len(self.backups) + 1

        scheduled\_time = datetime.datetime.now() + datetime.timedelta(hours=1)

        backup = BackupRecord(backup\_id, db\_details, scheduled\_time)

        self.backups.append(backup)

        return backup.backup\_id

    def read\_backup(self, backup\_id):

        for backup in self.backups:

            if backup.backup\_id == backup\_id:

                return backup

        return None

    def update\_backup(self, backup\_id, new\_db\_details):

        backup = self.read\_backup(backup\_id)

        if backup:

            backup.db\_details = new\_db\_details

            return True

        return False

    def delete\_backup(self, backup\_id):

        for i, backup in enumerate(self.backups):

            if backup.backup\_id == backup\_id:

                del self.backups[i]

                return True

        return False

    def \_simulate\_backup\_process(self, backup\_id):

        # Simulate a backup process by updating status after some time

        backup = self.read\_backup(backup\_id)

        if backup:

            time.sleep(5)  # Simulate backup time

            backup.update\_status("completed")

    def schedule\_database\_backups(self, db\_details):

        backup\_id = self.create\_backup(db\_details)

        backup\_thread = Thread(target=self.\_simulate\_backup\_process, args=(backup\_id,))

        backup\_thread.start()

        return backup\_id

Show quoted text

        print("\nBackup Management Menu:")

        print("1. Create Backup")

        print("2. Read Backup")

        print("3. Update Backup")

        print("4. Delete Backup")

        print("5. Schedule Backup")

        print("0. Exit")

        choice = int(input("Enter your choice: "))

        if choice == 1:

            db\_details = {"name": input("Enter database name: "), "user": input("Enter database user: ")}

            backup\_id = manager.create\_backup(db\_details)

            print(f"Backup created with ID: {backup\_id}")

        elif choice == 2:

            backup\_id = int(input("Enter Backup ID to read: "))

            backup = manager.read\_backup(backup\_id)

            if backup:

                print(f"Backup ID: {backup.backup\_id}, Details: {backup.db\_details}, Status: {backup.status}, Scheduled Time: {backup.scheduled\_time}")

            else:

                print("Backup not found.")

        elif choice == 3:

            backup\_id = int(input("Enter Backup ID to update: "))

            new\_details = {"name": input("Enter new database name: "), "user": input("Enter new user: ")}

            if manager.update\_backup(backup\_id, new\_details):

                print("Backup updated successfully.")

            else:

                print("Backup not found.")

        elif choice == 4:

            backup\_id = int(input("Enter Backup ID to delete: "))

            if manager.delete\_backup(backup\_id):

                print("Backup deleted successfully.")

            else:

                print("Backup not found.")

        elif choice == 5:

            db\_details = {"name": "ScheduledDB", "user": "admin"}

            backup\_id = manager.schedule\_database\_backups(db\_details)

            print(f"Backup scheduled with ID: {backup\_id}. Status: scheduled")

# CHAPTER-9

# OUTPUT

Enter 1 to create backup:

Enter 2 to read the backup:

Enter 3 to update backup:

Enter 4 to delete backup:

Enter 5 to schedule backup:

Enter 0 to exit:

Enter your choice: 1

Backup created with ID: 1 Enter 1 to create backup:

Enter 2 to read the backup:

Enter 3 to update backup:

Enter 4 to delete backup:

Enter 5 to schedule backup:

Enter to exit:

Enter your choice: 3

Enter Backup ID to update: 1

Enter new database name: xyz

Enter new user: ABC

Backup updated successfully.

Enter 1 to create backup:

Enter 2 to read the backup: Enter 3 to update backup:

Enter 4 to delete backup:

Enter 5 to schedule backup:

Enter to exit:

Enter your choice: 2

Enter Backup ID to read: 1

Backup ID: 1, Details: ('name': 'xyz', 'user': 'ABC'}, Status: completed

Enter 1 to create backup:

Enter 2 to read the backup:

Enter 3 to update backup:

Enter 4 to delete backup:

Enter 5 to schedule backup:

Enter 0 to exit:

Enter your choice: 0

Exiting...

**CHAPTER-10**

# CONCLUSION

The BackupManager system provides a simple yet efficient tool for managing database backups using Python. It supports the basic CRUD operations—create, read, update, and delete—while ensuring thread safety through the use of locking mechanisms. With the ability to schedule backups in a separate thread, it allows for non-blocking operations, ensuring the system can handle multiple tasks concurrently.

The implementation demonstrates the following key features:

Thread Safety: The use of a lock ensures that the shared list of backups is protected when accessed by multiple threads, preventing data races and maintaining consistency.

Scheduling and Monitoring: The system can schedule backups to be created at a future time, and users can monitor the status of these backups.

Unit Testing: Thorough unit tests are provided to verify that all core functionalities—such as backup creation, reading, updating, deleting, and scheduling—are working correctly.

This tool can be expanded and customized further, such as by integrating actual databa

# CHAPTER-11

# REFERENCES

🡪Google