NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF APPLIED SCIENCE

Department of Informatics and Analytics

COURSE: ENTERPRISE DATA MANAGEMENT

LECTURER: MRS. N. MARABADA

DUE DATE: 10 MAY 2024

TOPIC: GROUP PROJECT - ROBUST DISASTER MANAGEMENT STRATEGY

GROUP 4 MEMBERS

BRANDON B NGWENYA N02018717C

BRIAN T MUTSETSA N02016615R

GUGULETHU T MAFU N02018963B

PAUL M NJINI N02019341C

CYNTHIA PONDIWA N02018183F

BYRON R MUTIMUSAKWA N02018714T

12. Disaster Recovery Plan for Enterprise Database

Introduction

In this documentation, we will outline a robust disaster recovery plan for an enterprise database, using an Employee Management System(CUSTOMTKINTER MODERN TKINTER PROJECT, 2023) as our use case. This plan includes regular backups, replication, and failover mechanisms to minimize data loss and ensure business continuity. The Employee Management System manages employee records in an SQLite database, and our solution involves automated and manual processes to protect and recover this critical data.

System Overview

The Employee Management System (EMS) is a desktop application developed using Python and the Tkinter library. It allows for CRUD (Create, Read, Update, Delete) operations on employee records stored in an SQLite database (**Employees.db**). The application also includes a disaster recovery feature implemented through regular backups and database replication.

The EMS consists of the following components:

- **Login Module** (**login.py**): Handles user authentication.
- **Main Application** (**main.py**): Manages the user interface and interaction.
- **Database Operations** (**database.py**): Manages database CRUD operations.
- **Backup and Replication** (**database_backup.py**): Handles backup, replication, and restoration.

Key Files

database.py

Handles the database operations for the Employee Management System.

CODE:

```
import sqlite3
import os
import shutil
from datetime import datetime

def create_table():
    conn = sqlite3.connect('Employees.db')
    cursor = conn.cursor()

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS Employees(
            id TEXT PRIMARY KEY,
            name TEXT,
            role TEXT,
            gender TEXT,
            status TEXT)''')
```

```
    conn.commit()
    conn.close()

def fetch_employees():
    conn = sqlite3.connect('Employees.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM Employees')
    employees = cursor.fetchall()
    conn.close()
    return employees

def insert_employee(id, name, role, gender, status):
    conn = sqlite3.connect('Employees.db')
    cursor = conn.cursor()

    cursor.execute('INSERT INTO Employees (id, name, role, gender, status)
VALUES (?, ?, ?, ?, ?)',
                   (id, name, role, gender, status))
    conn.commit()
    conn.close()


def delete_employee(id):
    conn = sqlite3.connect('Employees.db')
    cursor = conn.cursor()
    cursor.execute('DELETE FROM Employees WHERE id = ?', (id,))
    conn.commit()
    conn.close()

def update_employee(new_name, new_role, new_gender, new_status, id):
    conn = sqlite3.connect('Employees.db')
    cursor = conn.cursor()
    cursor.execute('UPDATE Employees SET name = ?, role = ?, gender = ?,
status = ? WHERE id = ?',
                   (new_name, new_role, new_gender, new_status, id))
    conn.commit()
    conn.close()

def id_exists(id):
    conn = sqlite3.connect('Employees.db')
    cursor = conn.cursor()
    cursor.execute('SELECT COUNT(*) FROM Employees WHERE id = ?', (id,))
    result = cursor.fetchone()
    conn.close()
    return result[0] > 0

create_table()
```

database_backup.py
Manages the backup and replication processes.

CODE:

```
import sqlite3
import os
import shutil
from datetime import datetime
import threading

# Function to create the backups table
def create_backups_table():
    try:
        conn = sqlite3.connect('Backups.db')
        c = conn.cursor()
        c.execute('''CREATE TABLE IF NOT EXISTS backups (
                        id INTEGER PRIMARY KEY,
                        date TEXT,
                        time TEXT,
                        type TEXT,
                        location TEXT
                        )''')
        conn.commit()
        conn.close()
        print("Backups table created successfully.")
    except Exception as e:
        print(f"Error creating backups table: {str(e)}")

# Function to insert a new backup record into the backups table
def insert_backup_record(date, time, backup_type, location):
    try:
        conn = sqlite3.connect('Backups.db')
        c = conn.cursor()
        c.execute('''INSERT INTO backups (date, time, type, location)
                     VALUES (?, ?, ?, ?)''', (date, time, backup_type,
location))
        conn.commit()
        conn.close()
        print("Backup record inserted successfully.")
    except Exception as e:
        print(f"Error inserting backup record: {str(e)}")

# Function to perform database backup
def backup_database(target_folder=None):
    try:
        source_db = "Employees.db"
        timestamp = datetime.now()
```

```
        formatted_date = timestamp.strftime("%Y-%m-%d")
        formatted_time = timestamp.strftime("%H-%M-%S")
        formatted_time_db = timestamp.strftime("%H:%M:%S")

        if target_folder:
            backup_file = os.path.join(target_folder,
f"Backup_{formatted_date}_{formatted_time}.bak")
        else:
            target_folder = "Backups"
            if not os.path.exists(target_folder):
                os.makedirs(target_folder)
            backup_file = os.path.join(target_folder,
f"Backup_{formatted_date}_{formatted_time}.bak")

        # Insert backup record into the backups table
        insert_backup_record(formatted_date, formatted_time_db, "Manual
Backup", backup_file)

        source_conn = sqlite3.connect(source_db)
        backup_conn = sqlite3.connect(backup_file)

        with source_conn:
            source_conn.backup(backup_conn)

        print("Backup completed successfully.")

        return backup_file
    except Exception as e:
        print(f"Error during backup: {str(e)}")
        return None

# Function to perform auto database backup
def auto_backup_database():
    try:
        source_db = "Employees.db"
        timestamp = datetime.now()
        formatted_date = timestamp.strftime("%Y-%m-%d")
        formatted_time = timestamp.strftime("%H-%M-%S")
        formatted_time_db = timestamp.strftime("%H:%M:%S")

        target_folder = "Backups"
        if not os.path.exists(target_folder):
            os.makedirs(target_folder)
        backup_file = os.path.join(target_folder,
f"Auto_Backup_{formatted_date}_{formatted_time}.bak")

        # Insert backup record into the backups table
```

```
        insert_backup_record(formatted_date, formatted_time_db, "Automatic
Backup", backup_file)

        source_conn = sqlite3.connect(source_db)
        backup_conn = sqlite3.connect(backup_file)

        with source_conn:
            source_conn.backup(backup_conn)

        print("Auto Backup completed successfully.")

        return backup_file
    except Exception as e:
        print(f"Error during backup: {str(e)}")
        return None

# Function to replicate database
def replicate_database():
    try:
        source_db = "Employees.db"
        target_folder = "Replicates"
        if not os.path.exists(target_folder):
            os.makedirs(target_folder)
        timestamp = datetime.now()
        formatted_date = timestamp.strftime("%Y-%m-%d")
        formatted_time = timestamp.strftime("%H-%M-%S")
        formatted_time_db = timestamp.strftime("%H:%M:%S")
        target_db = os.path.join(target_folder,
f"Replicated_{formatted_date}_{formatted_time}.db")

        shutil.copy(source_db, target_db)
        insert_backup_record(formatted_date, formatted_time_db, "Replication",
target_db)
        print("Replication completed successfully.")

        return target_db
    except Exception as e:
        print(f"Error during replication: {str(e)}")
        return None

# Function to perform auto database replication
def auto_replicate_database():
    try:
        source_db = "Employees.db"
        target_folder = "Replicates"
        if not os.path.exists(target_folder):
            os.makedirs(target_folder)
        timestamp = datetime.now()
```

```
        formatted_date = timestamp.strftime("%Y-%m-%d")
        formatted_time = timestamp.strftime("%H-%M-%S")
        formatted_time_db = timestamp.strftime("%H:%M:%S")
        target_db = os.path.join(target_folder,
f"Auto_Replicated_{formatted_date}_{formatted_time}.db")

        shutil.copy(source_db, target_db)
        insert_backup_record(formatted_date, formatted_time_db, "Automatic
Replication", target_db)
        print("Auto Replication completed successfully.")

        return target_db
    except Exception as e:
        print(f"Error during replication: {str(e)}")
        return None

# Automated backup function
def automated_backup():
    auto_backup_database()


# Automated replication function
def automated_replication():
    auto_replicate_database()

# Function to restore the database from a backup file
def restore_database(backup_file):
    try:
        # Ensure the backup file exists
        if not os.path.exists(backup_file):
            print("Backup file does not exist.")
            return

        # Get the current database file
        current_db = "Employees.db"

        # Close any existing connections to the database
        conn = sqlite3.connect(current_db)
        conn.close()

        # Remove the current database file
        os.remove(current_db)

        # Copy the backup file to replace the current database
        shutil.copy(backup_file, current_db)

        print("Database restored successfully.")
    except Exception as e:
```

```
        print(f"Error during database restoration: {str(e)}")

# Create backups table if it doesn't exist
create_backups_table()
```

main.py
The main application code that includes the UI for managing employees and invoking backup/replication actions.

CODE:

```
from tkinter import filedialog
import customtkinter
from tkinter import *
from tkinter import ttk
import tkinter as tk
from tkinter import messagebox
import database
import time
import database_backup
import shutil
import os
import sys

def scheduled_backup():
    database_backup.automated_backup()
    messagebox.showinfo("Auto Backup", "The database has been backed up.")
    database_backup.automated_replication()
    messagebox.showinfo("Auto Replicate", "The database has been replicated.")
    app.after(60000, scheduled_backup)

def backup_to_folder():
    folder_selected = filedialog.askdirectory()
    if folder_selected:
        backup_file = database_backup.backup_database(folder_selected)
        messagebox.showinfo("Success", f"The database has been backed up to
{backup_file}")

def replicate_database():
    database_backup.replicate_database()
    messagebox.showinfo("Success", "The database has been replicated.")

def restore_backup():
    pass

def back_up_database():
```

```
    database_backup.backup_database()
    messagebox.showinfo("Success", "The database has been backed up.")


def download_strategy():
    def resource_path(relative_path):
        """ Get the absolute path to the resource, which works for both
development and PyInstaller bundle. """
        try:
            # PyInstaller creates a temp folder and stores path in _MEIPASS
            base_path = sys._MEIPASS
        except Exception:
            base_path = os.path.abspath(".")

        return os.path.join(base_path, relative_path)

    # Path to the strategy PDF file
    strategy_file =
resource_path('Strategy/Disaster_Recovery_Plan_for_Enterprise_Database.pdf')

    print(f"Looking for strategy file at: {strategy_file}")

    # Check if the strategy file exists
    if not os.path.isfile(strategy_file):
        print("The strategy file does not exist.")
        messagebox.showerror("Error", "The strategy file does not exist.")
        return

    # Ask user where to save the file
    save_location = filedialog.asksaveasfilename(defaultextension=".pdf",
filetypes=[("PDF files", "*.pdf")])

    if save_location:
        try:
            with open(strategy_file, 'rb') as src_file:
                with open(save_location, 'wb') as dest_file:
                    dest_file.write(src_file.read())
            print(f"Strategy PDF has been saved to {save_location}")
            messagebox.showinfo("Success", f"Strategy PDF has been saved to
{save_location}")
        except FileNotFoundError:
            print("The strategy file was not found.")
            messagebox.showerror("Error", "The strategy file was not found.")
        except PermissionError:
            print("Permission denied. Please check your file permissions.")
            messagebox.showerror("Error", "Permission denied. Please check
your file permissions.")
        except Exception as e:
            print(f"Failed to save the strategy PDF: {e}")
```

```
            messagebox.showerror("Error", f"Failed to save the strategy PDF:
{e}")


def run_main_app():
    global app
    app = customtkinter.CTk()
    app.title('Employee Management System')
    app.geometry('950x520')
    app.config(bg='#161c25')
    app.resizable(False, False)

    font1 = ('Roboto', 20, 'bold')
    font2 = ('Roboto', 12, 'bold')

    def add_to_treeview():
        employees = database.fetch_employees()
        tree.delete(*tree.get_children())
        for employee in employees:
            tree.insert('', END, values=employee)

    def clear(*clicked):
        if clicked:
            tree.selection_remove(tree.focus())
        id_entry.delete(0, END)
        name_entry.delete(0, END)
        role_entry.delete(0, END)
        variable1.set('Male')
        variable2.set('Active')

    def display_data(event):
        selected_item = tree.focus()
        if selected_item:
            item_values = tree.item(selected_item)
            if 'values' in item_values:
                clear()
                values = item_values['values']
                id_entry.insert(0, values[0])
                name_entry.insert(0, values[1])
                role_entry.insert(0, values[2])
                variable1.set(values[3])
                variable2.set(values[4])
        else:
            pass

    def restore_database():
        backup_file = filedialog.askopenfilename(filetypes=[("Database Files",
"*.db *.bak")])
        if backup_file:
```

```
            database_backup.restore_database(backup_file)
            messagebox.showinfo("Success", "Database has been successfully
restored.")
            add_to_treeview()

    def delete():
        selected_item = tree.focus()
        if not selected_item:
            messagebox.showerror('Error', 'Choose an employee to delete.')
        else:
            id = id_entry.get()
            name = name_entry.get()
            database.delete_employee(id)
            add_to_treeview()
            clear()
            messagebox.showinfo('Success', name + ' has been deleted from the
database.')

    def update():
        selected_item = tree.focus()
        if not selected_item:
            messagebox.showerror('Error', 'Choose an employee to update.')
        else:
            id = id_entry.get()
            name = name_entry.get()
            role = role_entry.get()
            gender = variable1.get()
            status = variable2.get()
            database.update_employee(name, role, gender, status, id)
            add_to_treeview()
            clear()
            messagebox.showinfo('Success', name + ' of ID: ' + id + ' has been
updated.')

    def insert():
        id = id_entry.get()
        name = name_entry.get()
        role = role_entry.get()
        gender = variable1.get()
        status = variable2.get()
        if not (id and name and role and gender and status):
            messagebox.showerror('Error', 'Enter all fields.')
        elif database.id_exists(id):
            messagebox.showerror('Error', 'ID already exists.')
        else:
            database.insert_employee(id, name, role, gender, status)
            add_to_treeview()
```

```
            messagebox.showinfo('Success', name + ' has been added to the
database.')

    id_label = customtkinter.CTkLabel(app, font=font1, text='ID:',
text_color='#fff', bg_color='#161C25')
    id_label.place(x=20, y=20)

    id_entry = customtkinter.CTkEntry(app, font=font1, text_color='#000',
fg_color='#fff', border_color='#0C9295', border_width=2, width=180)
    id_entry.place(x=100, y=20)

    name_label = customtkinter.CTkLabel(app, font=font1, text='Name:',
text_color='#fff', bg_color='#161C25')
    name_label.place(x=20, y=80)

    name_entry = customtkinter.CTkEntry(app, font=font1, text_color='#000',
fg_color='#fff', border_color='#0C9295', border_width=2, width=180)
    name_entry.place(x=100, y=80)

    role_label = customtkinter.CTkLabel(app, font=font1, text='Role:',
text_color='#fff', bg_color='#161C25')
    role_label.place(x=20, y=140)

    role_entry = customtkinter.CTkEntry(app, font=font1, text_color='#000',
fg_color='#fff', border_color='#0C9295', border_width=2, width=180)
    role_entry.place(x=100, y=140)

    gender_label = customtkinter.CTkLabel(app, font=font1, text='Gender:',
text_color='#fff', bg_color='#161C25')
    gender_label.place(x=20, y=200)

    options = ['Male', 'Female']
    variable1 = StringVar()

    gender_options = customtkinter.CTkComboBox(app, font=font1,
text_color='#000', fg_color='#fff', dropdown_hover_color='#0C9295',
button_color='#0C9295', button_hover_color='#0C9295', border_color='#0C9295',
width=180, variable=variable1, values=options, state='readonly')
    gender_options.set('Male')
    gender_options.place(x=100, y=200)

    status_label = customtkinter.CTkLabel(app, font=font1, text='Status:',
text_color='#fff', bg_color='#161C25')
    status_label.place(x=20, y=260)

    options_2 = ['Active', 'Inactive']
    variable2 = StringVar()
```

```
    status_options = customtkinter.CTkComboBox(app, font=font1,
text_color='#000', fg_color='#fff', dropdown_hover_color='#0C9295',
button_color='#0C9295', button_hover_color='#0C9295', border_color='#0C9295',
width=180, variable=variable2, values=options_2, state='readonly')
    status_options.set('Active')
    status_options.place(x=100, y=260)

    add_button = customtkinter.CTkButton(app, command=insert, font=font1,
text_color='#fff', text='Add Employee', fg_color='#05A312',
hover_color='#00850B', bg_color='#161c25', cursor='hand2', corner_radius=15,
width=260)
    add_button.place(x=20, y=310)

    clear_button = customtkinter.CTkButton(app, command=lambda:clear(True),
font=font1, text_color='#fff', text='New Employee', fg_color='#161c25',
hover_color='#ff5002', bg_color='#161c25',
border_color='#f15704',  border_width=2, cursor='hand2', corner_radius=15,
width=260)
    clear_button.place(x=20, y=360)

    update_button = customtkinter.CTkButton(app, command=update,  font=font1,
text_color='#fff', text='Update Employee', fg_color='#161c25',
hover_color='#ff5002', bg_color='#161c25',
border_color='#f15704',  border_width=2, cursor='hand2', corner_radius=15,
width=260)
    update_button.place(x=300, y=360)

    delete_button = customtkinter.CTkButton(app, command=delete, font=font1,
text_color='#fff', text='Delete Employee', fg_color='#e40404',
hover_color='#ae0000', bg_color='#161c25',
border_color='#e40404',  border_width=2, cursor='hand2', corner_radius=15,
width=260)
    delete_button.place(x=580, y=360)

    backup_button = customtkinter.CTkButton(app, font=font1,
text_color='#fff', text='Backup Database', fg_color='#0C9295',
hover_color='#0C9350', bg_color='#161c25', cursor='hand2', corner_radius=15,
width=260, command=back_up_database)
    backup_button.place(x=20, y=410)

    backup_to_button = customtkinter.CTkButton(app, font=font1,
text_color='#fff', text='Backup To Folder', fg_color='#0C9295',
hover_color='#0C9350', bg_color='#161c25', cursor='hand2', corner_radius=15,
width=260, command=backup_to_folder)
    backup_to_button.place(x=300, y=410)

    replicate_button = customtkinter.CTkButton(app, font=font1,
text_color='#fff', text='Replicate Database', fg_color='#0C9295',
```

```
hover_color='#0C9350', bg_color='#161c25', cursor='hand2', corner_radius=15,
width=260, command=replicate_database)
    replicate_button.place(x=580, y=410)

    restore_button = customtkinter.CTkButton(app, font=font1,
text_color='#fff', text='Restore Database', fg_color='#0C9295',
hover_color='#0C9350', bg_color='#161c25', cursor='hand2', corner_radius=15,
width=260, command=restore_database)
    restore_button.place(x=20, y=460)

    # New button for downloading strategy
    download_strategy_button = customtkinter.CTkButton(app, font=font1,
text_color='#fff', text='Download Strategy', fg_color='#0C9295',
hover_color='#0C9350', bg_color='#161c25', cursor='hand2', corner_radius=15,
width=260, command=download_strategy)
    download_strategy_button.place(x=300, y=460)

    style = ttk.Style(app)
    style.theme_use('clam')
    style.configure('Treeview', font=font2, foreground='#fff',
background='#000', fieldbackground='#313837')
    style.map('Treeview', background=[('selected', '#1a8f2d')])

    tree = ttk.Treeview(app, height=15)
    tree['columns'] = ('ID', 'Name', 'Role', 'Gender', 'Status')
    tree.column('#0', width=0, stretch=tk.NO)
    tree.column('ID', anchor=tk.CENTER, width=120)
    tree.column('Name', anchor=tk.CENTER, width=120)
    tree.column('Role', anchor=tk.CENTER, width=120)
    tree.column('Gender', anchor=tk.CENTER, width=120)
    tree.column('Status', anchor=tk.CENTER, width=120)

    tree.heading('ID', text='ID')
    tree.heading('Name', text='Name')
    tree.heading('Role', text='Role')
    tree.heading('Gender', text='Gender')
    tree.heading('Status', text='Status')

    tree.place(x=300, y=20)
    tree.bind('<ButtonRelease>', display_data)

    add_to_treeview()
    app.after(60000, scheduled_backup)
    app.mainloop()

if __name__ == "__main__":
    run_main_app()
```

Backup Strategy

A comprehensive backup strategy is essential for protecting data and ensuring quick recovery in case of data loss. The EMS implements both manual and automated backup procedures to safeguard the database.

Manual Backups

Manual backups can be initiated by the user through the EMS interface. This allows users to create backups at critical points, such as before performing significant updates or maintenance.

How to Perform a Manual Backup:

1. Open the EMS.
2. Click the "Backup Database" button.
3. The system will create a backup of the **Employees.db** file in the default backup folder.

Automated Backups

Automated backups ensure that the database is regularly backed up without user intervention. The EMS is configured to perform automated backups every minute.

How Automated Backup Works:

- The **scheduled_backup** function in **main.py** triggers the **automated_backup** function in **database_backup.py**.
- The backup file is created with a timestamp to distinguish it from other backups.

Backup Storage

Backups are stored in the **Backups** directory by default. The user can also specify a different target folder when performing a manual backup.

Backup Folder Structure:

Backups/ Backup_YYYY-MM-DD_HH-MM-SS.bak

Replication Mechanisms

Replication is the process of copying data from one database to another to ensure data redundancy and improve availability. The EMS includes both manual and automated replication processes.

Manual Replication

Users can manually replicate the database to create a duplicate copy for redundancy.

How to Perform Manual Replication:

1. Open the EMS.
2. Click the "Replicate Database" button.
3. The system will create a replicated database file in the **Replicates** folder.

Automated Replication
Similar to automated backups, automated replication ensures that the database is regularly copied without user intervention.

How Automated Replication Works:
- The **scheduled_backup** function in **main.py** also triggers the **automated_replication** function in **database_backup.py**.
- The replicated file is created with a timestamp to distinguish it from other replications.

Replication Folder Structure:
Replicates/ Replicated_YYYY-MM-DD_HH-MM-SS.db

Failover Mechanisms
Failover mechanisms are designed to switch to a redundant or standby database in case the primary database fails. While the EMS does not currently implement an automated failover system, it provides the necessary tools to quickly restore the database from a backup or replicated copy.

Restoration Procedures
In case of database failure, the EMS allows users to restore the database from a backup file. This ensures that data can be quickly recovered with minimal loss.

How to Restore the Database:
1. Open the EMS.
2. Click the "Restore Database" button.
3. Select the backup file (**.bak** or **.db**) to restore.
4. The system will replace the current **Employees.db** file with the selected backup file.

Code for Restoration

CODE:

```
# Function to restore the database from a backup file
def restore_database(backup_file):
    try:
        # Ensure the backup file exists
        if not os.path.exists(backup_file):
            print("Backup file does not exist.")
            return

        # Get the current database file
        current_db = "Employees.db"

        # Close any existing connections to the database
        conn = sqlite3.connect(current_db)
        conn.close()
```

```
        # Remove the current database file
        os.remove(current_db)

        # Copy the backup file to replace the current database
        shutil.copy(backup_file, current_db)

        print("Database restored successfully.")
    except Exception as e:
        print(f"Error during database restoration: {str(e)}")
```

Conclusion

This disaster recovery plan for the Employee Management System outlines the necessary steps to protect and recover data using regular backups, replication, and restoration procedures. By implementing these strategies, the EMS ensures data integrity and availability, minimizing the risk of data loss and maintaining business continuity.

By following the outlined procedures, enterprises can confidently manage their employee data and be prepared for potential data loss scenarios. Regular backups and replication not only safeguard critical information but also provide peace of mind, knowing that the data is protected and recoverable.

Reference List

- EMPLOYEE MANAGEMENT SYSTEM PYTHON CUSTOMTKINTER MODERN TKINTER PROJECT WITH SQLITE3 DATABASE 2023, YouTube video, added by Information Tech [Online]. Available at: https://www.youtube.com/watch?v=B0BOayNs4jI [Accessed 08 May 2024].