It is recommended to use parameterized SQL statements to get rid of repeated SQL code statement!

* Create table

Text

Description automatically generated

* Insert a record (value)

Text

Description automatically generated

Text

Description automatically generated

* Read a record

A screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generated

* Update a record

Text

Description automatically generatedText

Description automatically generated

* Delete a record

Text

Description automatically generatedText

Description automatically generated

* A Class Structure with all CRUD methods included. Graphical user interface, text, application

  Description automatically generated
* Full DAO pattern in Python

# person\_dao.py

# Import packages

import sqlite3

# Constants

DATABASE\_URI = 'example.db'

class PersonDAO():

def create(self, data):

# Print info for debugging

print("\nCreating a person ...\n") #\n means print("\n") a blank line

print(f"data: {data}")

result = {}

# Parameterized Query i.e. question marks as placeholders for actual values

conn = None # First initialise the connection to None

try:

conn = sqlite3.connect(DATABASE\_URI)

cur = conn.cursor()

query = "INSERT INTO person VALUES (?, ?, ?);" # all columns + PK

param\_tuple = (

None, # person\_id is set to None for database to autoincrement

data['firstname'],

data['lastname']

)

cur.execute(query, param\_tuple)

result['message'] = 'Person added successfully!'

# OPTIONAL: Get the id of record inserted - cursor should be still open

# Might be useful later in more advanced cases

# e.g. when inserting records in 2 tables at the same time for 1:m transactions

inserted\_person\_id = cur.lastrowid

print(f"inserted\_person\_id: {inserted\_person\_id}")

result['person\_id'] = inserted\_person\_id

cur.close()

conn.commit()

except sqlite3.Error as error:

# This part of the code is executed if an error occured when executing any statement in the try block

result['message'] = 'Create person failed!'

print(f"Database {DATABASE\_URI} - Create person failed!")

print(error)

finally:

# The finally block is always executed - even if an exception happened

# This is the ideal place to close the connection

# It's always a good idea to check if the object exists before calling a method/function from the object

# Invoking a method on object which does not exist will cause your code to crash

if conn:

conn.close()

#print("Database closed")

#print(f"result: {result}")

return result # return the result as a dictionary

def find\_by\_id(self, person\_id):

# Print info for debugging

print("\nFinding a person ...\n")

print(f"person\_id: {person\_id}")

# Create a blank dictionary to return the result

result = {}

# Using Parameterised Query

conn = None

try:

conn = sqlite3.connect(DATABASE\_URI)

conn.row\_factory = sqlite3.Row # to be able to use row.keys()

cur = conn.cursor()

query = "SELECT \* FROM person WHERE person\_id=?;"

param\_tuple = (person\_id, ) # Works as this is a tuple of length 1

cur.execute(query, param\_tuple)

row = cur.fetchone() # get the next row - only one row in this case

if row:

# cursor.description contains the name of the columns

# Use dictionary compejension to build the dictionary

# Use list comprehension - get column names from cursor.description

# The column name is at index 0 i.e. the first position

col\_names = [description[0] for description in cur.description]

#print(f"Column names: {col\_names}")

# Using dictionary comprehension and enumerate()

# to match the column names with their index positions

d = {key: row[i] for i, key in enumerate(col\_names)}

result['person'] = d

else:

result['message'] = "Person not found!"

cur.close()

conn.commit()

except sqlite3.Error as error:

result['message'] = 'Find by id failed!'

print(f"Database {DATABASE\_URI} - Find by id failed!")

print(error)

finally:

if conn:

conn.close()

#print("Database closed")

# Note that the return is not part of the if/else block

# Ensure it's indented to the left

#print(f"result: {result}")

return result # return the result as a dictionary

def find\_by\_lastname(self, lastname):

# Print info for debugging

print("\nFinding person(s) by lastname ...\n")

print(f"lastname: {lastname}")

# Create a blank dictionary to return the result

result = {}

# Using Parameterised Query

conn = None

try:

conn = sqlite3.connect(DATABASE\_URI)

cur = conn.cursor()

#query = "SELECT \* FROM person WHERE lastname LIKE ?;" # Partial match

query = "SELECT \* FROM person WHERE lastname = ?;" # Exact match

param\_tuple = (lastname, ) # If a single value, must have a comma at the end!

cur.execute(query, param\_tuple)

rows = cur.fetchall()

if rows:

print(f"rows: {rows}")

# Convert the list of row objects to a list of dictionaries

# This query could return more than one persons - so create a list

list\_persons = [] # Create an empty list to append person dicts

for x in rows: # rows is a list of SQlite objects - process one by one

# cursor.description contains the name of the columns

# Use dictionary compejension to build the dictionary

# Use list comprehension - get olumn names from cursor.description

# The column name is at index 0 i.e. the first position

col\_names = [description[0] for description in cur.description]

#print(f"Column names: {col\_names}")

# Using dictionary comprehension and enumerate()

# to match the column names with their index positions

d = {key: x[i] for i, key in enumerate(col\_names)} # works

list\_persons.append(d) # Append the person dict to the person list

# Store the person list in the result dict under key "persons"

result['persons'] = list\_persons

else:

result['message'] = "No persons found!"

cur.close()

conn.commit()

except sqlite3.Error as error:

result['message'] = 'Find by lastname failed!'

print(f"Database {DATABASE\_URI} - Find by lastname failed!")

print(error)

finally:

if conn:

conn.close()

#print("Database closed")

#print(f"result: {result}")

return result # return the result as a dictionary

def find\_all(self):

# Print info for debugging

print("\nFinding all persons ...\n")

# Create a blank dictionary to return the result

result = {}

# Using Parameterised Query

conn = None

try:

conn = sqlite3.connect(DATABASE\_URI)

cur = conn.cursor()

query = "SELECT \* FROM person;"

#param\_tuple = ()

#cur.execute(query, param\_tuple)

cur.execute(query)

rows = cur.fetchall()

if rows:

print(f"rows: {rows}")

# Convert the list of row objects to a list of dictionaries

# This query could return more than one persons - so create a list

list\_persons = [] # Create an empty list to append person dicts

for x in rows: # rows is a list of SQLite objects - process one by one

# cursor.description contains the name of the columns

# Use dictionary comprehension to build the dictionary

# Use list comprehension - get column names from cursor.description

# The column name is at index 0 i.e. the first position

col\_names = [description[0] for description in cur.description]

#print(f"Column names: {col\_names}")

# Using dictionary comprehension and enumerate()

# to match the column names with their index positions

d = {key: x[i] for i, key in enumerate(col\_names)} # works

list\_persons.append(d) # Append the person dict to the person list

pass

# After the for loop

# Store person list in result dict under key "persons" - PLURAL

result['persons'] = list\_persons

else:

result['message'] = "No persons found!"

cur.close()

conn.commit()

except sqlite3.Error as error:

result['message'] = 'Find all failed!'

print(f"Database {DATABASE\_URI} - Find all failed!")

print(error)

finally:

if conn:

conn.close()

#print("Database closed")

#print(f"result: {result}")

return result # return the result as a dictionary

def find\_ids(self):

"""

This is a special method similar to find\_all but returns person\_ids only,

not the full details

"""

# Print info for debugging

print("\nFinding all person ids ...\n")

# Create a blank dictionary to return the result

result = {}

# Using Parameterised Query

conn = None

try:

conn = sqlite3.connect(DATABASE\_URI)

cur = conn.cursor()

query = "SELECT person\_id FROM person;"

cur.execute(query)

rows = cur.fetchall()

if rows:

result['person\_ids'] = [x[0] for x in rows] # List comprehension to grab first element of the tuple

else:

result['message'] = "No persons found!"

cur.close()

conn.commit()

except sqlite3.Error as error:

result['message'] = 'Find ids failed!'

print(f"Database {DATABASE\_URI} - Find ids failed!")

print(error)

finally:

if conn:

conn.close()

#print("Database closed")

#print(f"result: {result}")

return result # return the result as a dictionary

def update(self, person\_id, data):

# Print info for debugging

print("\nUpdating person ...\n")

print(f"person\_id: {person\_id}")

print(f"data: {data}")

# Create a blank dictionary to return the result

result = {}

# Using Parameterised Query

conn = None

try:

conn = sqlite3.connect(DATABASE\_URI)

cur = conn.cursor()

# Update all the attributes in person table except person\_id

query = """UPDATE person

SET

firstname=?,

lastname=?

WHERE

person\_id = ?;"""

param\_tuple = (

data['firstname'],

data['lastname'],

person\_id)

cur.execute(query, param\_tuple)

result['message'] = 'Person Updated!'

cur.close()

conn.commit()

except sqlite3.Error as error:

result['message'] = 'Person NOT updated!'

print(f"Database {DATABASE\_URI} - Update person failed")

print(error)

finally:

if conn:

conn.close()

#print("Database closed")

#print(f"result: {result}")

return result

def delete(self, person\_id):

# Print info for debugging

print("\nDeleting person ...\n")

print(f"person\_id: {person\_id}")

# Create a blank dictionary to return the result

result = {}

# Using Parameterised Query

conn = None

try:

conn = sqlite3.connect(DATABASE\_URI)

cur = conn.cursor()

query = "DELETE FROM person WHERE person\_id = ?;"

param\_tuple = (person\_id, )

cur.execute(query, param\_tuple)

result['message'] = 'Person deleted!'

cur.close()

conn.commit()

except sqlite3.Error as error:

result['message'] = 'Person NOT deleted!'

print(f"Database {DATABASE\_URI} - Delete person failed")

print(error)

finally:

if conn:

conn.close()

#print("Database closed")

#print(f"result: {result}")

return result # return the result as a dictionary

# Dao implementation

# 1. Instantiate the person DAO

person\_dao = PersonDAO()

# 2. Set up the data to insert

data = {

'firstname':"Hamish",

'lastname': "Pollard"

}

# 3. Call the create() method on the DAO

# Pass the data as param

# And store the returned value somewhere

result = person\_dao.create(data)

* Testing DAO patterns

Graphical user interface, text, application

Description automatically generated

* Testing Dao

# person\_dao\_test.py

# Import the DAO

from full\_dao import PersonDAO

def test\_create():

# 1. Instantiate the person DAO

person\_dao = PersonDAO()

# 2. Set up the data to insert

data = {

'firstname':"Hamish",

'lastname': "Pollard"

}

# 3. Call the create() method on the DAO

# Pass the data as param

# And store the returned value somewhere

result = person\_dao.create(data)

# 4. Do something with the results e.g. print it

print(result)

def test\_find\_by\_id():

# 1. Instantiate the person DAO

person\_dao = PersonDAO()

# 2. Setup the ID to find

person\_id = 1

# 3. Call the find\_by\_id() method on the DAO

# Pass the data as param

# And store the returned value somewhere

result = person\_dao.find\_by\_id(person\_id)

# 4. Do something with the results e.g. print it

print(result)

def test\_find\_all():

# 1. Instantiate the person DAO

person\_dao = PersonDAO()

# 2. Call the find\_all() method

# And store the returned value somewhere

result = person\_dao.find\_all()

# 4. Do something with the results e.g. print it

print(result)

def test\_find\_by\_lastname():

# 1. Instantiate the person DAO

person\_dao = PersonDAO()

# 2. Setup the lastname to find

lastname = "Pollard"

# 3. Call the find\_by\_lastname() method on the DAO

# Pass the data as param

# And store the returned value somewhere

result = person\_dao.find\_by\_lastname(lastname)

# 4. Do something with the results e.g. print it

print(result)

def test\_find\_ids():

# 1. Instantiate the person DAO

person\_dao = PersonDAO()

# 2. Call the find\_ids() method on the DAO

result = person\_dao.find\_ids()

# 3. Print the result

print(result)

def test\_update():

# 1. Instantiate the person DAO

person\_dao = PersonDAO()

# 2. Set up the data to update i.e. ID and data

person\_id = 1

data = {}

data['firstname'] = "Joe"

data['lastname'] = "Pollard"

# 3. Call the update() method on the DAO

# Pass the ID and data as params

# And store the returned value somewhere

result = person\_dao.update(person\_id, data)

# 4. Do something with the results e.g. print it

print(result)

def test\_delete():

# 1. Instantiate the person DAO

person\_dao = PersonDAO()

# 2. Set up the ID to update

person\_id = 1

# 3. Call the update() method on the DAO

# Pass the ID and data as params

# And store the returned value somewhere

result = person\_dao.delete(person\_id)

# 4. Do something with the results e.g. print it

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

test\_create()

test\_find\_by\_id()

test\_find\_all()

test\_find\_by\_lastname()

test\_find\_ids()

test\_update()

test\_delete()