Demo Program - High-Level Language with Embedded SQL Statements

May 14, 2023

1 Introduction

The "Demo Program: High-Level Language with Embedded SQL Statements" is a Python notebook file that demonstrates how to implement SQL in Python via SQLite. The code includes embedded SQL statements that allow the program to interact with a database and perform various operations such as querying data, inserting new records, updating existing data, and deleting records.

2 [Optional] Remove existing database

Just in case you have to run this again

```
[ ]: import os
     os.remove("data.db")
```

3 Import SQLite3 module

```
[ ]: import sqlite3
```

4 Connect to the SQLite database

```
[ ]: connection = sqlite3.connect('data.db')
```

5 Create a cursor, a pointer to the database

```
[ ]: cursor = connection.cursor()
```

6 Build a simple table

```
[ ]: create_table = "CREATE TABLE IF NOT EXISTS people (id integer primary key,␣
      ↪first_name text, last_name text)"
     cursor.execute(create_table)  # and we have to execute the query
```

```
[ ]: <sqlite3.Cursor at 0x29360cc4540>
```

7 Create a table for the items

```
[ ]: create_table_items = "CREATE TABLE IF NOT EXISTS items (id integer primary key,␣
       ↪name text, price real)"
     cursor.execute(create_table_items)
```

```
[ ]: <sqlite3.Cursor at 0x29360cc4540>
```

8 Let's insert some data into the table

```
[ ]: insert_hitagi = "INSERT INTO people VALUES (1, 'Hitagi', 'Senjougahara')"
     cursor.execute(insert_hitagi)
```

```
[ ]: <sqlite3.Cursor at 0x29360cc4540>
```

9 Worst way possible to insert data into the table

```
[ ]: insert_nozomu = "INSERT INTO people VALUES ({}, '{}', '{}')".format(
         999, 'Nozomu', 'Itoshiki')
     cursor.execute(insert_nozomu)  # bobby tables, anyone?
```

```
[ ]: <sqlite3.Cursor at 0x29360cc4540>
```

10 Safer way to insert data into the table

```
[ ]: insert = "INSERT INTO people VALUES (?, ?, ?)"
     cursor.execute(insert, (2, 'Madoka', 'Kaname'))
```

```
[ ]: <sqlite3.Cursor at 0x29360cc4540>
```

11 Why use that instead of string formatting?

Ever heard of SQL injection? It's where the user can enter SQL commands into a form and have them executed on the server. This is a huge security risk. If we sanitize the data first and not just shove it into the SQL command, we can avoid this.

https://xkcd.com/327/

https://bobby-tables.com/python

So PLEASE, don't use string formatting to insert data into SQL commands. Use the parameter substitution instead.

12 After we insert data, we have to commit it

If you don't commit, the data won't be saved to the "database" (SQLite stores the whole DB in a single file)

```
[ ]: connection.commit()
```

13 Now, let's retrieve the data from the table

You'll get a list of tuples. Each tuple is a row in the table

To actually use it, you must write a function to convert it into an object or a dictionary

Or even better: ORM

```
[ ]: select = "SELECT * FROM people"
     for row in cursor.execute(select):
         print(row)
```

```
(1, 'Hitagi', 'Senjougahara')
(2, 'Madoka', 'Kaname')
(999, 'Nozomu', 'Itoshiki')
```

14 Insert multiple data into the table

We can insert multiple data into the table with `executemany()` and a list of tuples.

```
[ ]: people = [
         (3, 'Tsubasa', 'Hanekawa'),
         (4, 'Mayoi', 'Hachikuji'),
         (5, 'Suruga', 'Kanbaru'),
         (6, 'Nadeko', 'Sengoku'),
         (7, 'Karen', 'Araragi'),
         (8, 'Tsukihi', 'Araragi'),
         (9, 'Shinobu', 'Oshino'),
         (10, 'Meme', 'Oshino'),
         (11, 'Deishu', 'Kaiki'),
         (12, 'Izuko', 'Gaen'),
         (13, 'Yozuru', 'Kagenui'),
         (14, 'Yotsugi', 'Ononoki'),
         (15, 'Ougi', 'Oshino'),
         (16, 'Tooe', 'Gaen'),
     ]

     # we use executemany() to insert multiple data. it can read tuples, lists, and␣
      ↪dictionaries
     cursor.executemany("INSERT INTO people VALUES (?, ?, ?)", people)
     connection.commit()
```

15 Now, let's see the result

```
[ ]: select = "SELECT * FROM people"
     for row in cursor.execute(select):
         print(row)

     # ok good
```

```
(1, 'Hitagi', 'Senjougahara')
(2, 'Madoka', 'Kaname')
(3, 'Tsubasa', 'Hanekawa')
(4, 'Mayoi', 'Hachikuji')
(5, 'Suruga', 'Kanbaru')
(6, 'Nadeko', 'Sengoku')
(7, 'Karen', 'Araragi')
(8, 'Tsukihi', 'Araragi')
(9, 'Shinobu', 'Oshino')
(10, 'Meme', 'Oshino')
(11, 'Deishu', 'Kaiki')
(12, 'Izuko', 'Gaen')
(13, 'Yozuru', 'Kagenui')
(14, 'Yotsugi', 'Ononoki')
(15, 'Ougi', 'Oshino')
(16, 'Tooe', 'Gaen')
(999, 'Nozomu', 'Itoshiki')
```

15.1 Or just use for loops and string formatting. works too

(not recommeded, string formatting is prone to SQL injection).

```
[ ]: morepeople = [
         (17, 'John', 'Price'),
         (18, 'Soap', 'MacTavish'),
         (19, 'Simon', 'Riley'),
         (20, 'Kyle', 'Garrick'),
         (21, 'Alejandro', 'Vargas'),
         (22, 'Kate', 'Laswell'),
     ]

     for person in morepeople:
         cursor.execute(insert, person)

     connection.commit()
```

15.2 Let's see how bad it is to use string formatting

[ ]:
```python
anglefish = [
    (23, 'Miho', 'Nishizumi'),
    (24, 'Saori', 'Takebe'),
    (25, 'Hana', 'Isuzu'),
    (26, 'Yukari', 'Akiyama'),
    (27, 'Mako', 'Reizei'),
]

for person in anglefish:
    # now imagine instead of 'yukari', it's 'yukari'; DROP TABLE people; -- '
    execution = "INSERT INTO people VALUES ({}, '{}', '{}')".format(
        person[0], person[1], person[2])
    cursor.execute(execution)

connection.commit()
```

16 Let's specifically select a person

[ ]:
```python
select_araragi = "SELECT * FROM people WHERE last_name = 'Araragi'"
for row in cursor.execute(select_araragi):
    print(row)

print()

select_lessthanfive = "SELECT * FROM people WHERE id < 5"
for row in cursor.execute(select_lessthanfive):
    print(row)

print()

# you can use fetch methods to get the data, like this
select_taskforce141 = "SELECT * FROM people WHERE id < 23 AND id > 16"
cursor.execute(select_taskforce141)
# fetchall() returns the whole result set as a list of tuples
print(cursor.fetchall())
# if you fetch, the cursor will move to the next row.

print()

# for this, we have to execute the query again
cursor.execute(select_taskforce141)
# fetchmany() returns the first n rows of the result set, in this case, Captain␣
 ↪Price and Soap
print(cursor.fetchmany(2))
# fetchone() returns the first row of the result set, in this case, Ghost
```

```
print(cursor.fetchone())
# fetching again returns the next row, in this case, Gaz
print(cursor.fetchone())
```

```
(7, 'Karen', 'Araragi')
(8, 'Tsukihi', 'Araragi')

(1, 'Hitagi', 'Senjougahara')
(2, 'Madoka', 'Kaname')
(3, 'Tsubasa', 'Hanekawa')
(4, 'Mayoi', 'Hachikuji')

[(17, 'John', 'Price'), (18, 'Soap', 'MacTavish'), (19, 'Simon', 'Riley'), (20,
'Kyle', 'Garrick'), (21, 'Alejandro', 'Vargas'), (22, 'Kate', 'Laswell')]

[(17, 'John', 'Price'), (18, 'Soap', 'MacTavish')]
(19, 'Simon', 'Riley')
(20, 'Kyle', 'Garrick')
```

17 Let's insert some items into the items table

[ ]:
```
items = [
    (1, 'Ramen', 100),
    (2, 'Bread', 50),
    (3, 'Coffee', 150),
    (4, 'Tea', 100),
    (5, 'Soda', 100),
    (6, 'Water', 50),
    (7, 'M4A1', 1000),
    (8, 'AK-47', 1000),
    (9, 'Saiga 12 with Dragon Breath', 7000),
    (10, 'B&T APC556 tuned by wzstats.gg', 2500),
    (11, 'Gunship Killstreak', 20000),
    (12, 'Juggernaut Killstreak', 15000),
    (13, 'Tactical Nuke Killstreak', 25000),
    (14, 'Love', 2.21),
]

cursor.executemany("INSERT INTO items VALUES (?, ?, ?)", items)
connection.commit()
```

18 Let's retrieve the data from the items table

[ ]:
```
select = "SELECT * FROM items"
for row in cursor.execute(select):
    print(row)
```

```
# ok good now we have a database with some data in it
```

```
(1, 'Ramen', 100.0)
(2, 'Bread', 50.0)
(3, 'Coffee', 150.0)
(4, 'Tea', 100.0)
(5, 'Soda', 100.0)
(6, 'Water', 50.0)
(7, 'M4A1', 1000.0)
(8, 'AK-47', 1000.0)
(9, 'Saiga 12 with Dragon Breath', 7000.0)
(10, 'B&T APC556 tuned by wzstats.gg', 2500.0)
(11, 'Gunship Killstreak', 20000.0)
(12, 'Juggernaut Killstreak', 15000.0)
(13, 'Tactical Nuke Killstreak', 25000.0)
(14, 'Love', 2.21)
```

19 Table to store many to many relationship

```
[ ]: create_table = "CREATE TABLE IF NOT EXISTS purchases (id integer primary key␣
      ↪autoincrement not null, person_id integer, item_id integer, FOREIGN␣
      ↪KEY(person_id) REFERENCES people(id), FOREIGN KEY(item_id) REFERENCES␣
      ↪items(id))"
     cursor.execute(create_table)
     connection.commit()
```

20 Let's insert some data into the table

Note: You may have to actually write some code for the employees to easily log purchases.

For example:

```
while loop:

  if input == 'exit':
      break
  else:
    if user input and item input corresponds to an id:
      insert it into the table
    else:
      print 'invalid input' or something idk maybe raise an exception
```

```
[ ]: purchases = [
         # hanekawa bought the APC556
         (None, 3, 10),  # hanekawa's id is 3, and the APC556's id is 10
         # madoka bought the bread
```

```
    (None, 2, 2),  # madoka's id is 2, and the bread's id is 2
    # miho bought the Gunship Killstreak
    (None, 23, 11),  # miho's id is 23, and the Gunship Killstreak's id is 11
    # captain price is hungry
    (None, 17, 1),  # captain price's id is 17, and the ramen's id is 1
    # maybe thirsty too
    (None, 17, 5),  # captain price's id is 17, and the soda's id is 5
    # ghost also craves the ramen
    (None, 19, 1),  # ghost's id is 19, and the ramen's id is 1
    # and finally, kaiki got the tactical nuke
    (None, 11, 13),  # kaiki's id is 11, and the tactical nuke's id is 13
    # nah, nadeko also got 25 killstreak
    (None, 6, 13),  # nadeko's id is 6, and the tactical nuke's id is 13
]

cursor.executemany("INSERT INTO purchases VALUES (?, ?, ?)", purchases)
connection.commit()  # don't forget to commit
```

21 Let's query some data

Let's say we want to get the name of the person who bought the ramen

Let's go line by line:

`SELECT people.first_name, people.last_name`

we want to get the first name and last name of the person

`FROM people`

from the people table

`JOIN purchases`

join the purchases table

`ON people.id = purchases.person_id`

where the id of the people table is equal to the person_id of the purchases table

`JOIN items`

join the items table

`ON items.id = purchases.item_id`

where the id of the items table is equal to the item_id of the purchases table

`WHERE items.name = 'Ramen'`

where the name of the items table is equal to 'Ramen'

Note: You can use WHERE items.name LIKE '%Ramen%' to get all items that contains the word 'Ramen'.

Which should be Captain Price and Ghost. Let's print it out

[ ]:
```python
query = (
    """
    SELECT people.first_name, people.last_name
    FROM people
    JOIN purchases
    ON people.id = purchases.person_id
    JOIN items
    ON items.id = purchases.item_id
    WHERE items.name = 'Ramen'
    """
)  # we use JOIN to join tables together

for row in cursor.execute(query):
    print(row)
```

```
('John', 'Price')
('Simon', 'Riley')
```

22 Let's say we want the data on Hanekawa's purchase

Let's go line by line:

`SELECT people.first_name, people.last_name, items.name, items.price`

we want to get the first name, last name, item name, and item price

`FROM people`

from the people table

`JOIN purchases`

join the purchases table

`ON people.id = purchases.person_id`

where the id of the people table is equal to the person_id of the purchases table

`JOIN items`

join the items table

`ON items.id = purchases.item_id`

where the id of the items table is equal to the item_id of the purchases table

`WHERE people.first_name = 'Hanekawa'`

where the first name of the people table is equal to 'Hanekawa'

Which should be the APC556. Let's print it out

```
[ ]: query = (
         """
         SELECT people.first_name, people.last_name, items.name, items.price
         FROM people
         JOIN purchases
         ON people.id = purchases.person_id
         JOIN items
         ON items.id = purchases.item_id
         WHERE people.first_name = 'Tsubasa'
         """
     )

     cursor.execute(query)
     print(cursor.fetchone()) # we only want one row, so we use fetchone()
```

```
('Tsubasa', 'Hanekawa', 'B&T APC556 tuned by wzstats.gg', 2500.0)
```

23 Let's say we want to get the total amount of money spent by each person

Let's go line by line:

`SELECT people.first_name, people.last_name, SUM(items.price)`

we want to get the first name, last name, and the sum of the price of all items bought by the person

`FROM people`

from the people table

`JOIN purchases`

join the purchases table

`ON people.id = purchases.person_id`

where the id of the people table is equal to the person_id of the purchases table

`JOIN items`

join the items table

`ON items.id = purchases.item_id`

where the id of the items table is equal to the item_id of the purchases table

`GROUP BY people.id`

group the data by the id of the people table

Note that Yukari doesn't have any purchases, so her total amount spent is 0, and she's not included in the result

```
[ ]: query = (
         """
         SELECT people.first_name, people.last_name, SUM(items.price)
         FROM people
         JOIN purchases
         ON people.id = purchases.person_id
         JOIN items
         ON items.id = purchases.item_id
         GROUP BY people.id
         """
     )

     for row in cursor.execute(query):
         print(row)
```

```
('Madoka', 'Kaname', 50.0)
('Tsubasa', 'Hanekawa', 2500.0)
('Nadeko', 'Sengoku', 25000.0)
('Deishu', 'Kaiki', 25000.0)
('John', 'Price', 200.0)
('Simon', 'Riley', 100.0)
('Miho', 'Nishizumi', 20000.0)
```

24 User input

Let's add new people to the database

Note: You may have to actually write some code for the employees to easily add new people to the database

```
[ ]: while True:
         print("Enter your data. Enter 'done' to stop")
         id = input("Enter your id: ")

         if id == 'done':
             connection.commit()
             break
         if id.isnumeric() == False:
             print("Invalid input. Please enter a number")
             continue
         first_name = input("Enter your first name: ")
         last_name = input("Enter your last name: ")
         cursor.execute("INSERT INTO people VALUES (?, ?, ?)",
                        (id, first_name, last_name))

     # Try:
     # 1. Adding a person with an id that already exists, which raises an exception␣
      ↪(can be handled)
```

```
# 2. Adding a person with a non-numeric id, which is already handled by␣
 ↪continuing the loop
# 3. Bobby Tables. You know what I mean
```

```
Enter your data. Enter 'done' to stop
```

25 Let's recheck the data

[ ]:
```
for row in cursor.execute("SELECT * FROM people"):
    print(row)
```

```
(1, 'Hitagi', 'Senjougahara')
(2, 'Madoka', 'Kaname')
(3, 'Tsubasa', 'Hanekawa')
(4, 'Mayoi', 'Hachikuji')
(5, 'Suruga', 'Kanbaru')
(6, 'Nadeko', 'Sengoku')
(7, 'Karen', 'Araragi')
(8, 'Tsukihi', 'Araragi')
(9, 'Shinobu', 'Oshino')
(10, 'Meme', 'Oshino')
(11, 'Deishu', 'Kaiki')
(12, 'Izuko', 'Gaen')
(13, 'Yozuru', 'Kagenui')
(14, 'Yotsugi', 'Ononoki')
(15, 'Ougi', 'Oshino')
(16, 'Tooe', 'Gaen')
(17, 'John', 'Price')
(18, 'Soap', 'MacTavish')
(19, 'Simon', 'Riley')
(20, 'Kyle', 'Garrick')
(21, 'Alejandro', 'Vargas')
(22, 'Kate', 'Laswell')
(23, 'Miho', 'Nishizumi')
(24, 'Saori', 'Takebe')
(25, 'Hana', 'Isuzu')
(26, 'Yukari', 'Akiyama')
(27, 'Mako', 'Reizei')
(999, 'Nozomu', 'Itoshiki')
```

26 Close the connection

After we're done, we can close the connection

[ ]:
```
connection.close()
```