

პროგრამირება

Python

ლექცია 4-5: PYTHON-ში რელაციურ მონაცემთა ბაზებთან მუშაობა;
რელაციური მონაცემთა ბაზები; ბაზის სტრუქტურის შექმნა,
მონაცემთა დამატება, შეცვლა და წაშლა;
მონაცემთა ანალიზის და ვიზუალიზაციის საწყისები

ლიკა სვანაძე

lika.svanadze@btu.edu.ge

ლექცია 4

Python & Database

- * Python პროგრამირების ენის მეშვეობით შესაძლებელია პროგრამა დაუკავშირდეს სხვადასხვა რელაციური მონაცემთა ბაზას, როგორიცაა SQL, MySQL, Oracle, SQLite, სხვ. ძირითადი პრინციპი ყველა ტიპის მონაცემთა ბაზასთან სამუშაოდ თითქმის მსგავსია, რაც გულისხმობს მაგ. მონაცემთა ბაზასთან დაკავშირებას, ახალი ცხრილის შექმნას (CREATE TABLE), ცხრილიდან მონაცემების წაკითხვას (SELECT), ცხრილში მონაცემების ჩასმას (INSERT), განახლებას (UPDATE) ან წაშლას (DELETE).
- * Python-ში არსებობს შესაბამისი მოდულები (ინტერფეისები), რომელთა გამოყენებით კონკრეტული ტიპის მონაცემთა ბაზასთან შეგვიძლია მუშაობა.

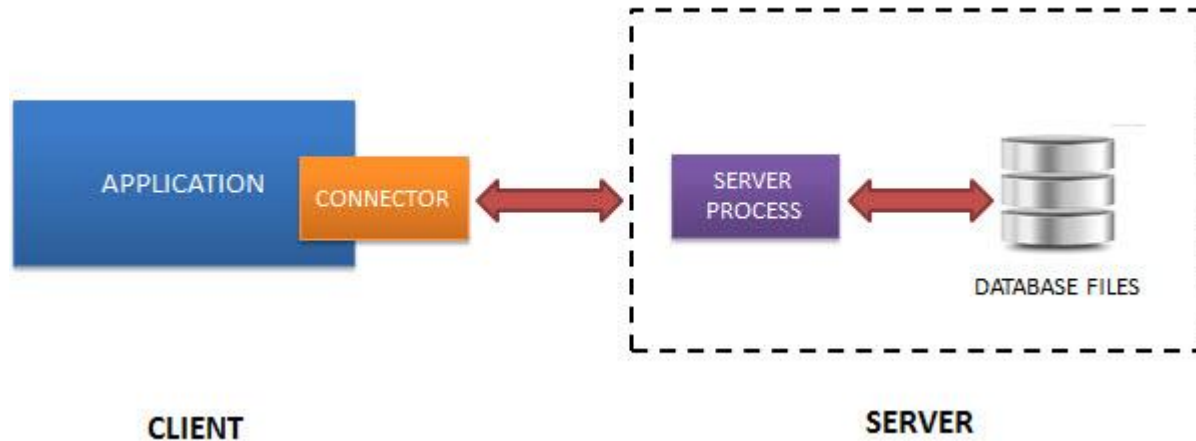
მონაცემთა ბაზა	Python-ში შესაბამისი მოდული (DB API)
MS SQL Server	pymssql, mssql
SQLite	sqlite3, pysqlite
MySQL	MySQLdb (mysql-python), mysql.connector
Oracle	cx_Oracle
PostgreSQL	psycopg

Python & DB - ეტაპები

- * **Create Connection** - ბაზასთან კავშირის დამყარება
- * **Create Statement** - შესასრულებელი ბრძანების შექმნა
- * **Execute Statement** - ბრძანების გაშვება
- * **Close connection** - ბაზასთან კავშირის გაწყვეტა

Client/server Architecture

RDBMS client/server architecture



SQLite: Server-less

- * SQLite წარმოადგენს ბიბლიოთეკას, რომელიც უზრუნველყოფს რელაციურ მონაცემთა ბაზის მართვის სისტემას. სხვა მონაცემთა ბაზებისგან განსხვავებით, იგი არის უფრო მსუბუქი ვერსია: სიმარტივის, ადმინისტრირების და რესურსების გამოყენების მხრივ. არ აქვს სერვერის მხარდაჭერა (server/client architecture) და არ საჭიროებს კონფიგურაციას. უმეტესად გამოიყენება მცირე და საშუალო ზომის ბაზებისთვის. ეფექტურად გამოიყენება მობილურ აპლიკაციებში და სხვა ელექტრონულ გაჯეტებში.

SQLite server-less architecture



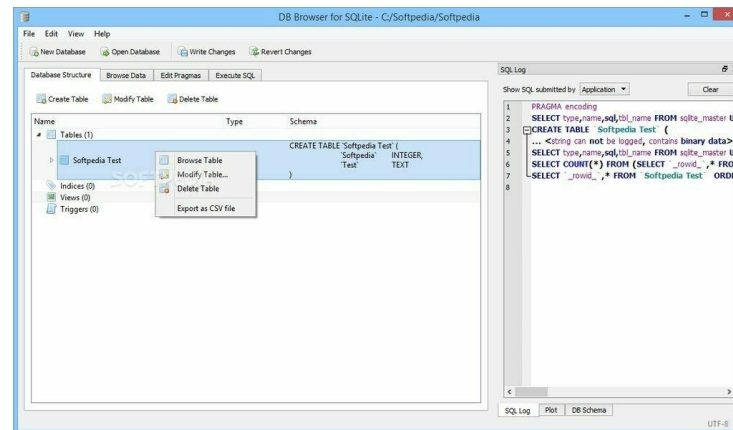
<https://www.sqlite.org/features.html>

SQLite Download & Viewer

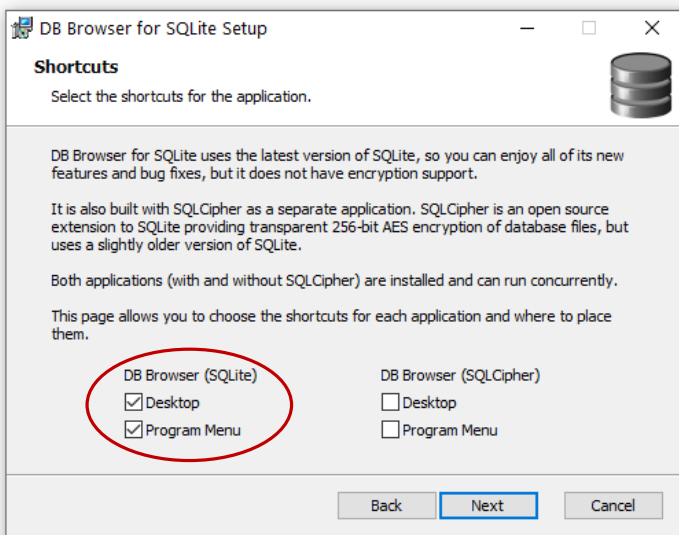
- * SQLite მონაცემთა ბაზასთან სამუშაოდ Python-ში გამოვიყენებთ **sqlite3** ბიბლიოთეკას, რომელიც სტანდარტულად მოყვება ინტერპრეტატორს.
- * სასურველი გვექონდეს ბაზისთვის viewer: **DB Browser for SQLite** (შემდეგ სლაიდზე)
- * დოკუმენტაცია sqlite3 მოდულის: <https://docs.python.org/3/library/sqlite3.html>
- * SQLite-ის შესახებ ლიტერატურა (წიგნები): <https://www.sqlite.org/books.html>
- * თუ გსურთ ვინდოუსის ტერმინალიდან SQLite ბაზასთან მუშაობა
 - უნდა გადმოწეროთ sqlite: <https://www.sqlite.org/download.html> აირჩიეთ Precompiled Binaries for Windows (A bundle of command-line tools.....)
 - ტერმინალში სამუშაო ბრძანებები შეგიძლიათ იხილოთ: <https://www.sqlite.org/cli.html>

Installation: DB Browser for SQLite

- * **DB Browser for SQLite** გამოიყენება SQLite ბაზის ფაილის გასახსნელად და მის დასამუშავებლად.
- * გადმოსაწერი ლინკი: <https://sqlitebrowser.org/>
- * ინსტალაციის ბოლო ეტაპზე შეგიძლიათ აირჩიოთ, რომ DB Browser (SQLite) პროგრამის შორტკატი გამოჩნდეს Desktop-ზე (SQL Cipher არ დაგვჭირდება).



DB Browser for SQLite



განმარტებები:

- * SQL – Structured Query Language
- * ჩანაწერი (record) - წარმოადგენს ცხრილის (Table-ის) ერთ სტრიქონს
- * SQL Statement – SQL ბრძანება, რომელიც ცხრილიდან კითხულობს ინფორმაციას და აბრუნებს შედეგს (მაგ. SELECT) ან ახდენს ცხრილში მონაცემების ცვლილებას (UPDATE, INSERT, DELETE). ასევე შესაძლებელია ცხრილის დამატება/წაშლა.
- * Delete record - მთლიანი ჩანაწერის წაშლა
- * Update record - ჩანაწერის რომელიმე ველის/ველების მოდიფიცირება

მონაცემთა ბაზის შექმნა & დაკავშირება

- * მონაცემთა ბაზის შექმნა შესაძლებელია როგორც ტერმინალიდან, ასევე უშუალოდ Python-ის გამოყენებით:

Python

```
import sqlite3
conn = sqlite3.connect("my_database.sqlite")
```

Terminal

```
>>> sqlite3 my_database.sqlite
```

- * SQLite მონაცემთა ბაზის ფაილის გაფართოება შეიძლება იყოს .db, .db3, .sqlite, .sqlite3
- * connect() მეთოდის გამოყენებით ხდება მონაცემთა ბაზასთან დაკავშირება. თუ პარამეტრად გადაცემული ბაზა არ არსებობს, შექმნის. ხოლო თუ არსებობს, მას დაუკავშირდება. აბრუნებს დაკავშირებულ ობიექტს.
- * connect()-ის გამოყენების შემდეგ უნდა მოხდეს Cursor კლასის ობიექტის შემოღება, რომლის მეშვეობით ხდება ყველა ბრძანების გაშვება მთელ პროგრამაში.
- * ბაზასთან კავშირის გასაწყვეტად გამოიყენება close() მეთოდი, რომელიც იწერება ბაზასთან მუშაობის დასრულების შემდეგ.

Python

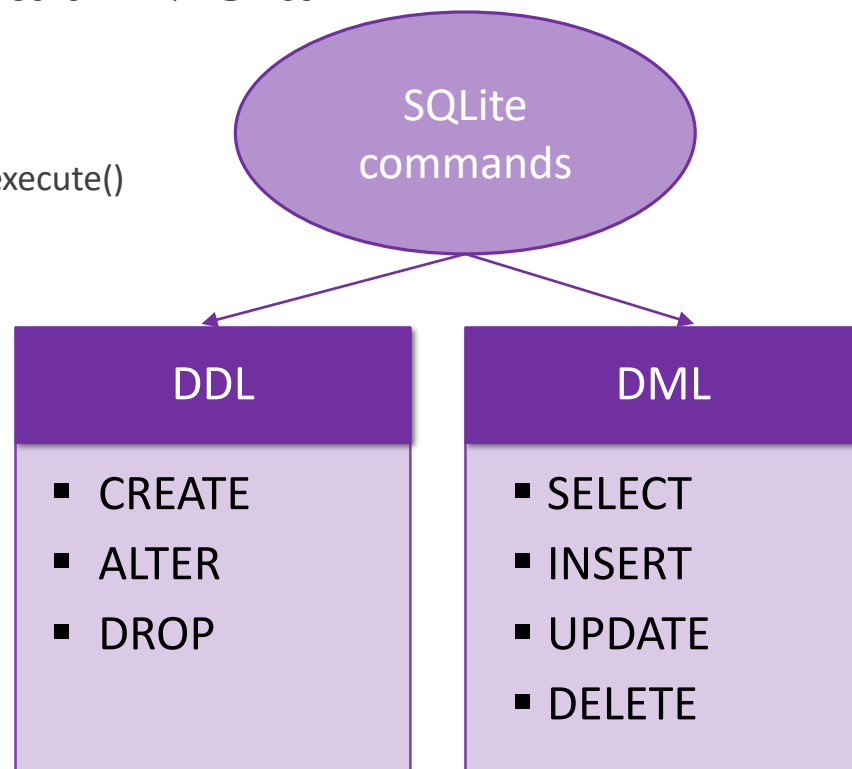
```
import sqlite3
conn = sqlite3.connect("my_database.sqlite")
cursor = conn.cursor()
...
...
conn.close()
```

SQLite Commands

* მონაცემთა ბაზებთან სამუშაო ბრძანებები შეიძლება დაიყოს შემდეგ კომპონენტებად:

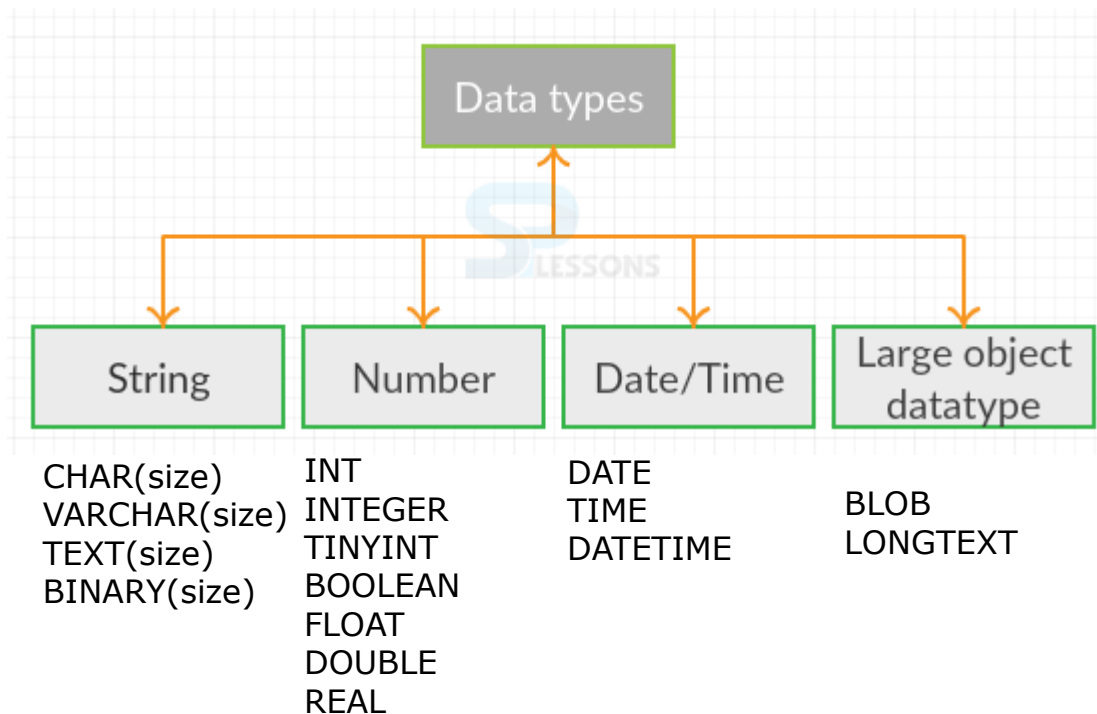
- DDL (Data Definition Language) - მონაცემთა ბაზის სტრუქტურის განსაზღვრა
- DML (Data Manipulation Language) - ბაზის მონაცემების დამუშავება

* ბრძანებების გამოყენება ხდება cursor ობიექტზე `execute()` მეთოდის გამოყენებით.



მონაცემთა ტიპები

- * ცხრილში (Table) შესაძლებელია სხვადასხვა ტიპის მონაცემების შემოღება, რომლებიც იყოფა შემდეგ კატეგორიებად:
- * დოკუმენტაცია: <https://www.sqlite.org/datatype3.html>




CREATE TABLE - ცხრილის შექმნა

- * ცხრილის შესაქმნელად უნდა დაიწეროს SQL ბრძანება CREATE TABLE, რომელიც უნდა მოთავსდეს execute მეთოდში:

SQL

```
CREATE TABLE books
(id INTEGER PRIMARY KEY AUTOINCREMENT,
title VARCHAR(50),
author VARCHAR(100),
price FLOAT);
```

Books	
id 	<pk>
title	
author	
price	

Python

```
cursor.execute('''CREATE TABLE books
(id INTEGER PRIMARY KEY AUTOINCREMENT,
title VARCHAR(50),
author VARCHAR(100),
price FLOAT);''')
```

INSERT INTO <table> - ჩანაწერის (მონაცემის) ჩასმა (1)

- * INSERT ბრძანების მეშვეობით ცხრილში ხდება მონაცემის ჩასმა.
- * შესაბამისი SQL ბრძანება შემდეგნაირად გამოიყურება:

SQL

```
INSERT INTO books (title, author, price) VALUES ('Hamlet', 'William Shakespeare', 10.5);
```

Python

```
cursor.execute("INSERT INTO books (title, author, price) VALUES ('Hamlet', 'William Shakespeare', 10.5)")
conn.commit()
```

- * Python-ში: ცვლილებების ასახვისთვის საჭიროა **commit()** მეთოდის გაშვება.

INSERT INTO <table> - ჩანაწერის (მონაცემის) ჩასმა (1)

- * Python-ში მონაცემთა ბაზასთან მუშაობის დროს (მაგ. ცხრილში მონაცემების ჩასმისას) საჭირო ხდება Python-ის ცვლადის მნიშვნელობების გამოყენება. ესეთ დროს, execute მეთოდში, მეორე პარამეტრად გადაეცემა tuple ტიპის ობიექტი. ხოლო SQL ბრძანებაში, შესაბამის ადგილას ჩაისმება სიმბოლო '?' – (question mark). კითხვის ნიშანს უწოდებენ sqlite-ის placeholder-ს.

Python

```
b_title = 'Hamlet'
b_author = 'William Shakespeare'
b_pr = 10.5
cursor.execute('INSERT INTO books (title, author, price) VALUES (?, ?, ?)', (b_title, b_author, b_pr))
conn.commit()
```

იგივე მოქმედება შეიძლება შესრულდეს შემდეგნაირად:

Python

```
book1 = ('Hamlet', 'William Shakespeare', 10.5)
cursor.execute('INSERT INTO books (title, author, price) VALUES (?, ?, ?)', book1)
conn.commit()
```

INSERT: რამდენიმე ჩანაწერის ჩასმა – **executemany**

- * ბევრი მონაცემის ერთიანად ჩასმა პარამეტრებით:
- * ბევრი მონაცემის ერთიანად ჩასასმელად, გამოიყენება `executemany()` მეთოდი.
- * მისი სინტაქსი: `executemany(sql, [parameters])`
- * შესაძლებელია რამდენიმე სტრიქონის დამატება ერთი SQL ბრძანებით, რომელსაც აქვს შემდეგი სინტაქსი:

SQL

```
INSERT INTO table_name (column_list)
VALUES
    (value_list_1),
    (value_list_2),
    ...
    (value_list_n);
```

- * Python-ში შემდეგნაირად ჩაიწერება:

Python

```
book_list = [
    ('The Book Thief', 'Markus Zusak', 17 ),
    ('Animal Farm', 'George Orwell', 13 ),
    ('The Hunger Games', 'Suzanne Collins', 17 ),
    ('Harry Potter and the Prisoner of Azkaban', 'Rowling', 25),
    ('Harry Potter and the Goblet of Fire', 'Rowling', 24 ),
    ('Macbeth', 'William Shakespeare', 29 )
]

cursor.executemany('INSERT INTO books (title, author, price) VALUES (?, ?, ?)', book_list)
conn.commit()
```


SELECT: მონაცემის ამორჩევა/წაკითხვა

- * SELECT ბრძანების მეშვეობით ხდება მონაცემების წაკითხვა ცხრილიდან. ის უნდა მოთავსდეს execute() მეთოდში. შედეგად აბრუნებს ობიექტს, სადაც ცხრილის თითოეული სტრიქონი წარმოდგენილია tuple ტიპის ელემენტად.

SQL

```
SELECT * FROM books;
```

Python

```
sel_result = cursor.execute("SELECT * FROM books")  
for row in sel_result:  
    print(row)
```

Python: შედეგი

```
(1, 'Hamlet', 'William Shakespeare', 10.5)  
(2, 'The Book Thief', 'Markus Zusak', 17.0)  
(3, 'Animal Farm', 'George Orwell', 13.0)  
(4, 'The Hunger Games', 'Suzanne Collins', 17.0)  
(5, 'Harry Potter and the Prisoner of Azkaban', 'Rowling', 25.0)  
(6, 'Harry Potter and the Goblet of Fire', 'Rowling', 24.0)  
(7, 'Macbeth', 'William Shakespeare', 29.0)
```

თითოეული
სტრიქონის
ელემენტზე წვდომა
ხდება ინდექსებით,
მაგ. row[0], row[1], ა.შ.

SELECT: fetchone, fetchall, fetchmany

- * fetchone(), fetchall() და fetchmany() ფუნქციები წარმოადგენს კურსორის (cursor)-ის მეთოდებს.
- * fetchone() მეთოდის გამოყენებისას შესაძლებელია გაშვებული SQL ბრძანების შედეგის თითო-თითოდ სტრიქონების დაბრუნება:

Python

```
cursor.execute("SELECT * FROM books")
record = cursor.fetchone()
print("Title = ", record[1])

record = cursor.fetchone()
print("Title = ", record[1])
```

შედეგი

Title = Hamlet

Title = The Book Thief

- * fetchall() მეთოდით ხდება SQL ბრძანების შედეგის ერთიანად დაბრუნება და წარმოდგენილია სიის სახით, რომლის თითოეული ელემენტი არის ცხრილის სტრიქონი (ჩანაწერი) tuple ტიპის.
- * fetchmany(<number>) მეთოდში პარამეტრად გადაეცემა რიცხვი და აბრუნებს შესაბამისი რაოდენობის ჩანაწერს.

Python

```
cursor.execute("SELECT * FROM books")
records = cursor.fetchmany(4)
for record in records:
    print(record)
```

შედეგი

```
(1, 'Hamlet', 'William Shakespeare', 10.5)
(2, 'The Book Thief', 'Markus Zusak', 17.0)
(3, 'Animal Farm', 'George Orwell', 13.0)
(4, 'The Hunger Games', 'Suzanne Collins', 17.0)
```



ლექცია 5

SELECT ცვლადის გამოყენებით - ':' placeholder

- * SELECT ბრძანების დაწერისას, შესაძლოა დაგვჭირდეს python-ის ცვლადის ჩართვა SQL ბრძანებაში. SQL ბრძანებაში '?'-ის გარდა, ასევე შეგვიძლია გამოვიყენოთ ':' სიმბოლო (placeholder)

```
pr_value = 15
cursor.execute("SELECT * FROM books WHERE price>:pr" , {'pr':pr_value})
records = cursor.fetchall()
for record in records:
    print(record)
```

- * თუ რამდენიმე ცვლადის ჩართვა გვსურს:

```
pr_value = 15
author_value = 'Rowling'
cursor.execute("SELECT * FROM books WHERE author=:author_name AND price>:pr",
               {'author_name':author_value, 'pr':pr_value})
```

- * ასევე შეგვიძლიათ გამოვიყენოთ უკვე განხილული placeholder – '?' შემდეგნაირად:

```
pr_value = 15
cursor.execute("SELECT * FROM books WHERE price>?", (pr_value, ))
```

- * გაითვალისწინეთ, რომ placeholder გამოიყენება ცხრილში არსებული მნიშვნელობების (values) ნაცვლად, და არა სვეტის ან ცხრილის დასახელებისთვის. ცხრილის ან სვეტის დასახელებისთვის გამოვიყენოთ format() ფუნქცია ან %s.

SELECT: შედეგი ლექსიკონის სახით

- * იმ შემთხვევაში თუ გვსურს რომ ცხრილიდან ამოღებული ინფორმაციას მივწვდეთ არა მხოლოდ ინდექსებით, არამედ ველის დასახელებითაც (მაგ, id, title, author, price), ამ შემთხვევაში ვიყენებთ Row კლასს. იგი უნდა მიენიჭოს connection-ის row_factory ატრიბუტს; შედეგად row_factory იძახებს sqlite3.Row-ს, რომელიც გარდაქმნის სტანდარტულ tuple-ს უფრო გამოყენებად ობიექტად (ანუ ცხრილის სტრიქონის ცალკეულ ელემენტებზე წვდომა შესაძლებელია არა მარტო ინდექსით, არამედ ველის დასახელებითაც).
- * Row კლასს აქვს keys() მეთოდი, რომელიც აბრუნებს ცხრილის ველების (სვეტების) დასახელებას სიის სახით:

```
conn.row_factory = sqlite3.Row
cursor = conn.cursor()
result = cursor.execute("SELECT * from books").fetchone()
print(result)
print(tuple(result))
print(result.keys())
print(result['title'])
print(result[1])
```

შედეგი

```
<sqlite3.Row object at 0x10d5dff10>
(1, 'Hamlet', 'William Shakespeare', 10.5)
['id', 'title', 'author', 'price']
Hamlet
Hamlet
```

SELECT: მონაცემების ამორჩევა

- * SELECT ბრძანების შესრულება შესაძლებელია მოხდეს სხვადასხვა პირობის გათვალისწინებით. შეამოწმეთ შემდეგი SQL ბრძანების შედეგები:

```
cursor.execute("SELECT DISTINCT author FROM books")
cursor.execute("SELECT count(DISTINCT author) AS count_row FROM books")
cursor.execute("SELECT * FROM books WHERE author='Rowling'")
cursor.execute("SELECT * FROM books WHERE author<>'Rowling'")
cursor.execute("SELECT * FROM books WHERE price>15")
cursor.execute("SELECT * FROM books WHERE author<>'Rowling' AND price>15")
cursor.execute("SELECT * FROM books WHERE NOT author='Rowling' AND price>15")
cursor.execute("SELECT * FROM books ORDER BY author")
cursor.execute("SELECT * FROM books ORDER BY author DESC")
cursor.execute("SELECT * FROM books ORDER BY author, price")
```

UPDATE: მონაცემების მოდიფიცირება

* UPDATE ბრძანების მეშვეობით შესაძლებელია ცხრილის მონაცემების შეცვლა.

SQL Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

SQL Example

```
UPDATE books
SET price=12
WHERE id=1;
```

Python Example

```
cursor = conn.cursor()
query = 'UPDATE books SET price=12 WHERE id=1'
cursor.execute(query)
print(cursor.rowcount)
conn.commit()
```

* კურსორს აქვს rowcount ატრიბუტი, რომელიც აბრუნებს ჩანაწერების რაოდენობას, რომლის განახლებაც მოხდა.

DELETE: მონაცემების ცხრილიდან წაშლა

- * DELETE ბრძანების მეშვეობით შესაძლებელია ცხრილის მონაცემების წაშლა გარკვეული პირობის მიხედვით.

SQL Syntax

```
DELETE FROM table_name WHERE condition;
```

SQL Example

```
DELETE FROM books WHERE author='William Shakespeare';
```

Python Example

```
cursor = conn.cursor()
query = "DELETE FROM books WHERE author='William Shakespeare'"
cursor.execute(query)
print(cursor.rowcount)
conn.commit()
```

- * თუ გვინტერესებს ცხრილში რამდენი ჩანაწერის წაშლა მოხდა, შეგვიძლია გამოვიყენოთ კურსორის rowcount ატრიბუტი, რომელიც აბრუნებს იმ ჩანაწერების რაოდენობას, რომლებიც წაიშალა ცხრილიდან.

პროგრამის სტრუქტურა

- * პითონში მუშაობისას უმჯობესია პროგრამა დაყოფილი იყოს მცირე ფრაგმენტებად ფუნქციების სახით, რაც აადვილებს კოდის კითხვადობას და ასევე უზრუნველყოფს მის ოპტიმალურ გამოყენებას. მაგ. ფუნქციებად შეიძლება იყოს გაწერილი ბაზასთან კავშირი, მონაცემების ბაზიდან დაბრუნება.
- * `main()` ფუნქცია არის პროგრამის ძირითადი ფუნქცია, რომლიდანაც ხდება სხვა ფუნქციების გამოძახება. `main()` ფუნქციის შესრულება ხდება არსებული ფაილის (პროგრამის) გაშვებისას.
- * `__name__ == '__main__'` პირობა ამოწმებს არის თუ არა პროგრამა გაშვებული არსებული ფაილიდან. ამ შემთხვევაში, `__name__` ცვლადის მნიშვნელობა არის `'__main__'`. თუ არსებული ფაილი, სხვა ფაილშია იმპორტირებული და სხვა ფაილიდან ხდება გამოძახება, ასეთ შემთხვევაში `__name__` ცვლადის მნიშვნელობა არის ფაილის სახელწოდება.
- * იხილეთ პრაქტიკული ამოცანა ლექცია 11-ის `classwork.py` ფაილში (ატვირთულია Google Classroom-ში)

მაგალითი

```
import sqlite3

def db_connection(file_name):
    return sqlite3.connect(file_name)

def select_all(conn):
    print("---- SELECTED ALL DATA ----")
    cursor = conn.cursor()
    query = "SELECT * from books"
    cursor.execute(query)
    records = cursor.fetchall()
    for record in records:
        print(record)

def main():
    conn = db_connection("book_db.sqlite")
    select_all(conn)
    conn.close()

if __name__ == '__main__':
    main()
```

შედეგი

```
---- SELECTED ALL DATA ----
(1, 'Hamlet', 'William Shakespeare', 10.5)
(2, 'The Book Thief', 'Markus Zusak', 17.0)
(3, 'Animal Farm', 'George Orwell', 13.0)
(4, 'The Hunger Games', 'Suzanne Collins', 17.0)
(5, 'Harry Potter and the Prisoner of Azkaban', 'Rowling', 25.0)
(6, 'Harry Potter and the Goblet of Fire', 'Rowling', 24.0)
(7, 'Macbeth', 'William Shakespeare', 29.0)
```

დამატებითი საკითხი: Tuple

- ერთ-ერთი მონაცემთა ტიპია Tuple, რომელიც ძალიან გავს სიას (List-ს). სიის მსგავსად, tuple წარმოადგებს მონაცემების მიმდევრობას.
- Tuple-ის შექმნა: ელემენტები გამოყოფილი უნდა იყოს მძიმით. აღწერის დროს უმჯობესია ელემენტები ჩასვით მრგვალ ფრჩხილებში, თუმცა თუ არ აქვს მითითებული პროგრამა მაინც აღიქვამს როგორც Tuple ტიპის მონაცემს.

```
p = (4, 5, 7, 8)
```

```
p = 4, 5, 7, 8
```

- ერთ ელემენტიანი Tuple-ის შემთხვევაში, პირველი ელემენტის შემდეგ უნდა იყოს სიმბოლო მძიმე, წინააღმდეგ შემთხვევაში პროგრამა არ აღიქვამს როგორც tuple-ს.

Python Console

```
>>> t=(4)
>>> type(t)
<class 'int'>
>>> t=(4,)
>>> type(t)
<class 'tuple'>
```

- Tuple-ის ელემენტებზე წვდომა შესაძლებელია ინდექსის გამოყენებით (მსგავსად list-ისა). მაგ. `p[0]` არის პირველი ელემენტის მნიშვნელობა.
- სიისაგან განსხვავებით, Tuple-ის შექმნის შემდეგ, მისი ელემენტების წაშლა, შეცვლა, დამატება არ არის შესაძლებელი.
- tuple-ებთან (სიების მსგავსად) გამოიყენება სხვადასხვა ფუნქციები: მაგ. `len()`, `max()`, `min()`, `sum()`, `sorted()`, `tuple()` და სხვ.
- tuple დამატებით შესაძლოა გამოვიყენოთ ფუნქციისთვის არუგემენტების გადასაცემად (იხილეთ მაგალითი მომდევნო სლაიდზე).

დამატებითი საკითხი: Tuple - *args

- დაწერეთ ფუნქცია, რომელსაც პარამეტრად გადაეცემა რიცხვები. ფუნქციამ უნდა იპოვოს ამ რიცხვების საშუალო არითმეტიკული. გამოიძახეთ ფუნქცია და იპოვეთ შემდეგი რიცხვების საშუალო არითმეტიკული (გაითვალისწინეთ რომ გამოძახების დროს ფუნქციას არგუმენტად შეიძლება გადაეცეს სხვადასხვა რაოდენობის პარამეტრები):
 - ა) 4, 6, 8
 - ბ) 5, 16, 3, 12
 - გ) 4, 7, 1, 10, 26, 35
- როდესაც არ ვიცით ფუნქციას რამდენი პარამეტრი აქვს, ესეთ დროს ვიყენებთ Tuple-ს. ფუნქციის აღწერის დროს ცვლადის სახელს ვუწერთ *-ს შემდეგნაირად. არ არის აუცილებელი ფუნქციის არგუმენტს ერქვას args, მთავარია რომ მითითებული იყოს სიმბოლო * :

შესრულება

```
def jami(*args):  
    return (sum(args)/len(args))  
  
s1=jami(4,6,8)  
print(s1)  
s2=jami(5, 16, 3, 12)  
print(s2)  
s3=jami(4, 7, 1, 10, 26, 30)  
print(s3)
```

შედეგი:

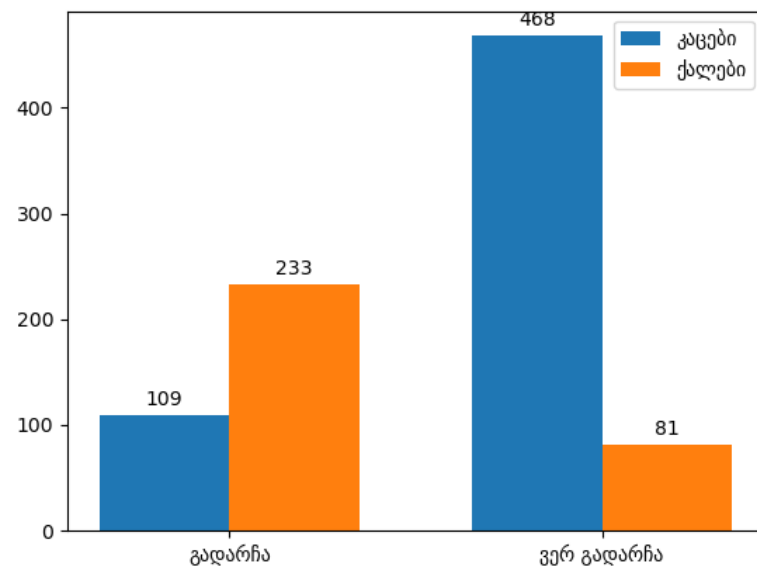
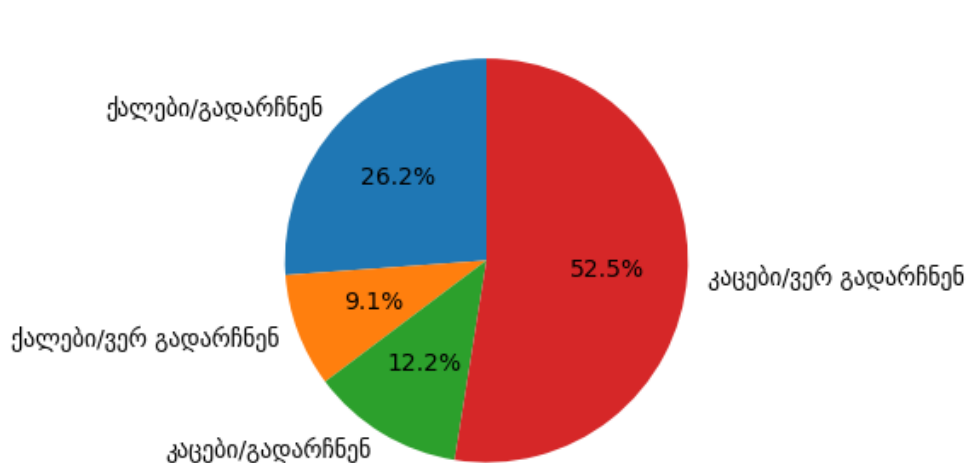
```
6.0  
9.0  
13.0
```

მონაცემთა ანალიზი და ვიზუალიზაცია

- * მას შემდეგ რაც ბაზიდან ინფორმაციას წავიკითხავთ, საჭიროა წამოღებული მონაცემების ანალიზი (მაგ. სტატისტიკური ანალიზი) და მათი ეფექტური ვიზუალიზაცია.
- * მონაცემთა ანალიზი და ვიზუალიზაცია პროგრამირების ერთ-ერთი მნიშვნელოვანი კომპონენტია და პითონი ფართოდ გამოიყენება სხვადასხვა ტიპის მონაცემთა ანალიზისთვის.
- * პითონში არსებობს სხვადასხვა მოდულები რომლებიც გამოიყენება მონაცემთა ანალიზისთვის, როგორიცაა numpy, pandas,, ხოლო ვიზუალიზაციისთვის იყენებენ matplotlib, seaborn.
- * მონაცემები გრაფიკულად შესაძლებელია წარმოდგენილი იყოს ჰისტოგრამების, წრიული დიაგრამების (pie charts), plots (მათემატიკური ფუნქციები/გრაფიკები)-ის საშუალებით.
- * Matplotlib-ის ოფიციალური საიტი: <https://matplotlib.org/>
- * მოდულების დაყენება შესაძლებელია pip-ის მეშვეობით შემდეგნაირად:
 - * pip install matplotlib
 - * pip install numpy

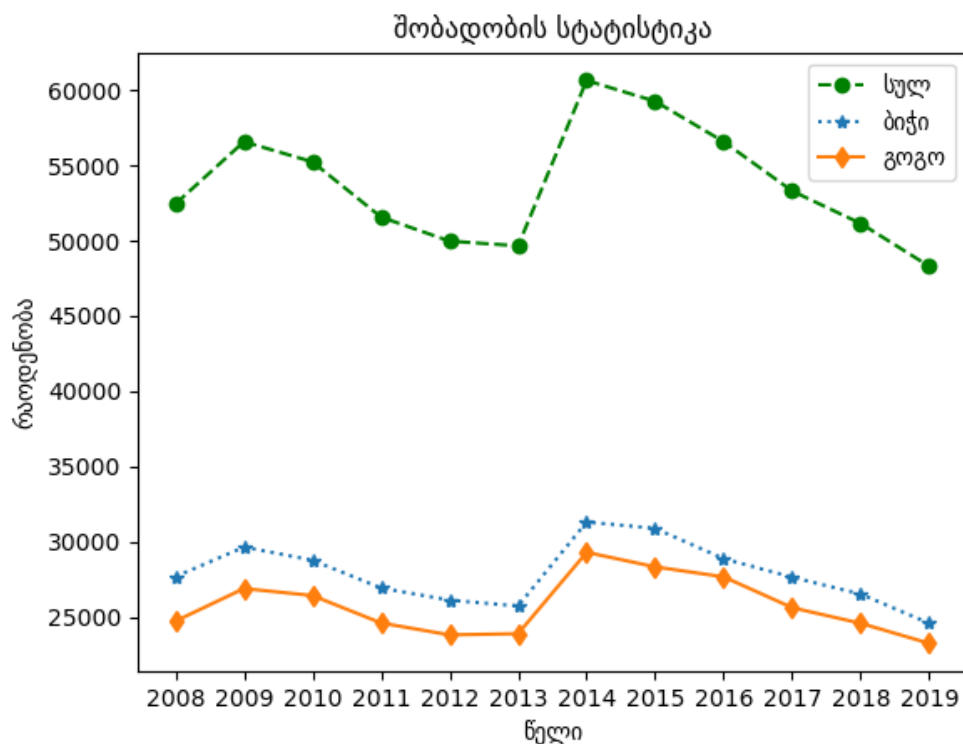
matplotlib - სავარჯიშო 1

- * matplotlib ბიბლიოთეკის გამოყენებით შესაძლებელია შევექმნათ შემდეგი სახის გრაფიკული გამოსახულებები. იხილეთ პრაქტიკული სავარჯიშოების კოდი ლექცია 11-ის classwork.py ფაილში (ატვირთულია Google Classroom-ში). სურათზე მოცემულია titanic.sqlite ბაზის მონაცემების მიხედვით შესაბამისი დიაგრამები:



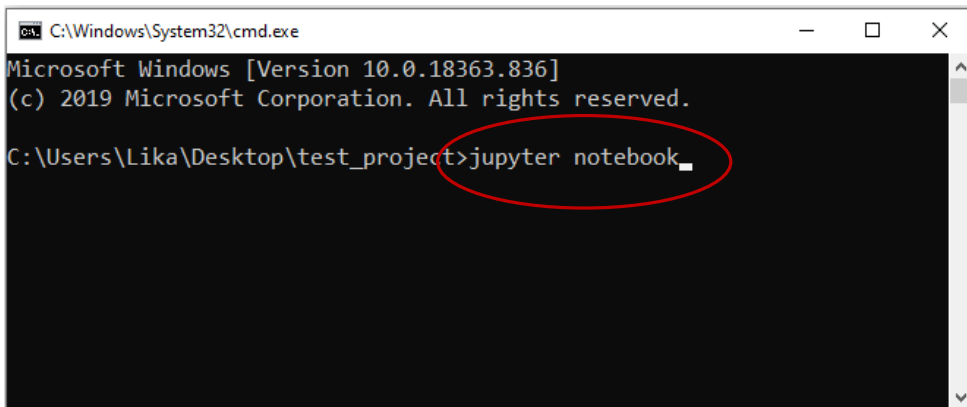
matplotlib - სავარჯიშო 2

- * სურათზე მოცემულია შობადობის სტატისტიკა საქართველოში წლების მიხედვით. გამოსახულია 3 წირი რომელიც ასახავს ჯამურ მონაცემებს და სქესის მიხედვით შობადობას. მონაცემები აღებულია საქართველოს სტატისტიკის ეროვნული სამსახურის ოფიციალური ვებ-გვერდიდან: <https://www.geostat.ge/>
- * იხილეთ ამ სავარჯიშოს კოდი ლექცია 11-ის classwork.py ფაილში (ატვირთულია Google Classroom-ში):



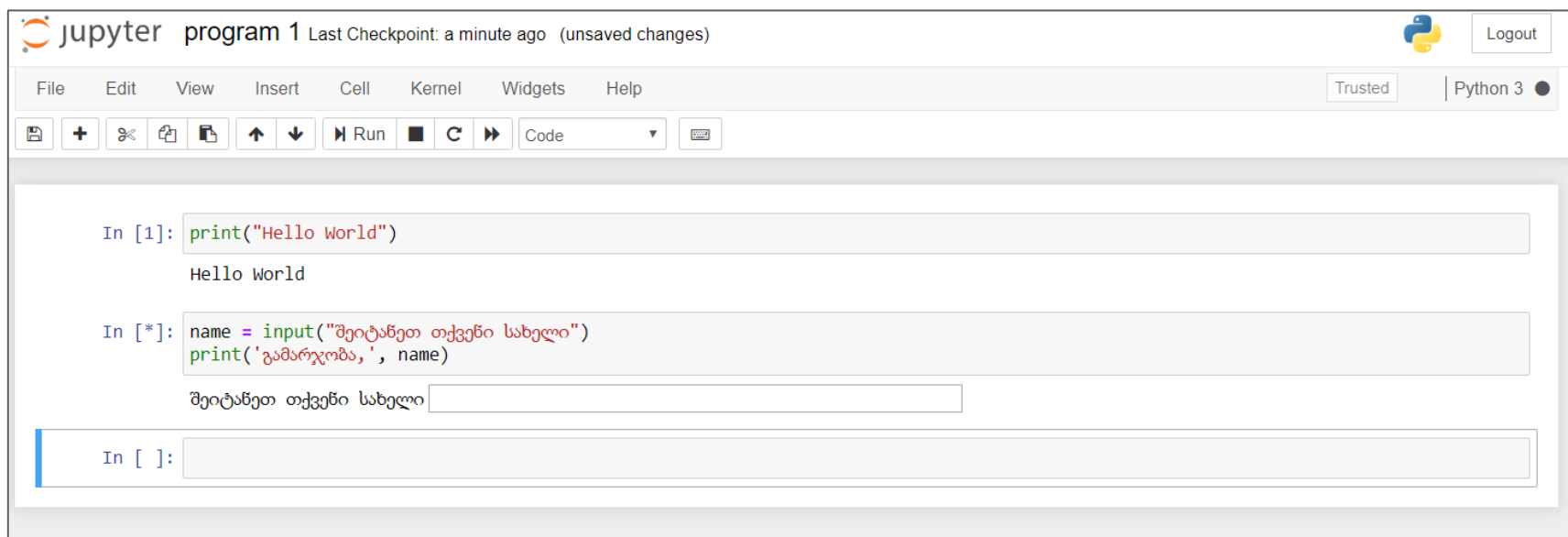
jupyter notebook

- * Jupyter Notebook წარმოადგენს ვებ აპლიკაციას სადაც live რეჟიმში შესაძლებელია კოდის წერა; უმეტესად გამოიყენება მონაცემთა ანალიზისთვის და ვიზუალიზაციის დროს.
- * ოფიციალური საიტი: <https://jupyter.org/>
- * ინსტალაცია: `pip install notebook` (დააყენეთ ვინდოუსის ტერმინალში)
- * პროგრამის გაშვება:
 - გადადით იმ საქაღალდეში სადაც გსურთ რომ გქონდეთ (ან უკვე გაქვთ) პროგრამის ფაილები.
 - გაუშვით `cmd` და გადადით ამ საქაღალდეში (`cd` ბრძანებით, ან საქაღალდეში სათაურში ჩაწერეთ `cmd` და გაეშვება ტერმინალი ამავე საქაღალდის მისამართის მითითებით);
 - ტერმინალში ჩაწერეთ `jupyter notebook`;
 - გაეშვება ვებ-აპლიკაცია `localhost`-ზე, სადაც დაინახავთ თქვენი საქაღალდის შიგთავსს.
 - დაამატეთ ახალი ფაილი (ან გახსენით უკვე შექმნილი `ipynb` გაფართოების ფაილი) და დაწერეთ სასურველი ბრძანებები



jupyter notebook

- * მიღებულ ინტერფეისში შესაძლებელია პითონის კოდის გაშვება.
- * ფაილის შენახვა ავტომატურად ხდება იმავე საქალაქო დეშო, რომელიც გაშვებული გაქვთ ტერმინალში. ფაილის ფორმატია .ipynb, თუმცა შესაძლებელია მისი შენახვა სხვა ფორმატშიც (მაგ. .py)
- * შესაძლებელია ბაზასთან დაკავშირება და მონაცემების ანალიზი/ვიზუალიზაცია.



სავარჯიშო 1:

- ❖ მოცემულია მონაცემთა ბაზა “book_db” (იხილეთ თანდართული მონაცემთა ბაზის ფაილი), რომელიც შეიცავს 3 ცხრილს: books, users, purchase. შეასრულეთ მომდევნო სლაიდზე მითითებული დავალებები.

book_db (SQLite 3)

Tables (3)

- books
- purchase
- users

book_db მონაცემთა ბაზა

	Name	Data type	Primary Key
1	id	INTEGER	
2	title	VARCHAR (50)	
3	author	VARCHAR (1...	
4	price	FLOAT	

books ცხრილის სტრუქტურა

	id	title	author	price
1	1	Hamlet	William Shakespeare	10.5
2	2	The Book Thief	Markus Zusak	17
3	3	Animal Farm	George Orwell	13
4	4	The Hunger Games	Suzanne Collins	17
5	5	Harry Potter and the Prisoner of Azkaban	Rowling	25
6	6	Harry Potter and the Goblet of Fire	Rowling	24
7	7	Macbeth	William Shakespeare	29

books ცხრილის მონაცემები

	Name	Data type	Primary Key
1	id	INTEGER	
2	username	VARCHAR (100)	
3	balance	FLOAT	

user ცხრილის სტრუქტურა

	id	username	balance
1	1	Mariam Barbakadze	150
2	2	Giorgi Sharikadze	120
3	3	Davit Giorgadze	40.6
4	4	Eka Mamaladze	35

user ცხრილის მონაცემები

Table name: purchase			
	Name	Data type	Primary Key
1	id	INTEGER	
2	user_id	INTEGER	
3	book_id	INTEGER	
4	purchase_date	DATE	

purchase ცხრილის სტრუქტურა

	id	user_id	book_id	purchase_date
1	1	2	3	2018-07-28
2	2	3	1	2018-05-21
3	3	1	5	2017-03-28
4	4	2	4	2018-01-04
5	5	1	3	2018-01-28
6	6	2	1	2017-04-18

purchase ცხრილის მონაცემები

სავარჯიშო 1:

დაწერეთ პროგრამა, რომელშიც ჩამოაყალიბებთ შემდეგ SQL ბრძანებებს:

- დაბეჭდეთ book ცხრილის ყველა წიგნი შექსპირის ავტორობით
- დაბეჭდეთ წიგნები, რომელიც დაწერილია შექსპირის ან Rowling-ის მიერ
- დაბეჭდეთ წიგნები რომლის ფასი არ აღემატება 20 ლარს.
- დაბეჭდეთ წიგნის ავტორები (განმეორებების გარეშე).
- დაბეჭდეთ იმ მომხარებელთა სახელები, რომლებსაც ბალანსზე აქვთ 100 ლარზე მეტი თანხა.
- დაბეჭდეთ წიგნების შესახებ მონაცემები, რომლებიც გაიყიდა 2018 წლამდე.
- დაბეჭდეთ იმ მომხარებლების მონაცემები, რომლებმაც 2018 წელს იყიდეს წიგნი.
- დაბეჭდეთ იმ მომხმარებლების მონაცემები, რომლებსაც აქვთ ერთი წიგნი მაინც შესყიდული.
- დაბეჭდეთ იმ წიგნების დასახელება, რომლებიც ერთხელ მაინც გაიყიდა
- დაბეჭდეთ ის წიგნები, რომლებიც არცერთხელ არ გაყიდულა (გამოიყენეთ LEFT JOIN).
- დაბეჭდეთ მომხმარებლების მონაცემები და მათ მიერ შესყიდული წიგნების დასახელებები (გამოიყენეთ INNER JOIN). დაალაგეთ მონაცემები მომხმარებლის სახელის მიხედვით (ალფაბეტი).

სავარჯიშო 2:

- * გადმოწერეთ titanic.sqlite ბაზა, რომელშიც არის ერთი ცხრილი passengers. ცხრილში მოცემულია ტიტანიკზე მყოფი 891 მგზავრის მონაცემები. გააკეთეთ შემდეგ სლაიდზე მოცემული დავალებები. ცხრილში არსებული სვეტებია (ველებია):

ველის დასახელება	აღწერა
PassengerId	მგზავრის ID
Survived	0 - ვერ გადარჩა; 1 - გადარჩა
Pclass	ბილეთის კლასის ნომერი: 1 - პირველი კლასის ბილეთი; 2 - მეორე კლასის ბილეთი; 3 - მესამე კლასის ბილეთი
Name	მგზავრის სახელი
Sex	სქესი
Age	ასაკი
SibSp	Sib (Siblings/დედმამიშვილები), Sp (Spouse/მეუღლე); დედმამიშვილების და მეუღლის რაოდენობა, რომლებიც იმყოფებოდნენ გემზე
Parch	Par (Parents/მშობლები), ch (Children/შვილები); მშობლების და შვილების რაოდენობა, რომლებიც იმყოფებოდნენ გემზე
Ticket	ბილეთის ნომერი
Fare	ბილეთის ფასი
Cabin	კაბინის ნომერი
Embarked	ჩასხდომის ადგილი: C = Cherbourg, Q = Queenstown, S = Southampton

სავარჯიშო 2:

1. უმჯობესია თითოეული დავალება (ბაზიდან მონაცემების წაკითხვა) ჩაწეროთ ფუნქციის სახით. დააკვირდით, შესაძლებელია თუ არა ერთი ფუნქცია გამოიყენოთ რამდენიმე დავალებისთვის - პარამეტრების გადაცემის გზით. ვინაიდან პროცენტის გამოთვლა ყველა დავალებაში გაქვთ, სასურველია ზოგადი შემთხვევისთვის ჩაწეროთ ფუნქცია, რომელიც დააბრუნებს პროცენტულ მონაცემს.

- * დაითვალოთ იმ მგზავრების რაოდენობა, რომლებიც იყვნენ ქალები. გამოთვალოთ პროცენტული რაოდენობებიც (მაგ. რამდენი პროცენტი იყო ქალი).
- * დაითვალოთ იმ მგზავრების რაოდენობა, რომლებიც იყვნენ კაცები. გამოთვალოთ პროცენტული რაოდენობებიც (მაგ. რამდენი პროცენტი იყო კაცი).
- * დაითვალოთ ქალების რაოდენობა, რომლებიც გადარჩა (ვერ გადარჩა). გამოთვალოთ პროცენტული რაოდენობებიც.
- * დაითვალოთ კაცების რაოდენობა, რომლებიც გადარჩა (ვერ გადარჩა). გამოთვალოთ პროცენტული რაოდენობებიც.
- * დაითვალოთ იმ მგზავრების რაოდენობა, რომლებსაც ქონდათ პირველი (მეორე, მესამე) კლასის ბილეთები. გამოთვალოთ პროცენტული რაოდენობებიც.
- * დაითვალოთ საშუალო ბილეთის ფასი თითოეული კლასის ბილეთისთვის.

2. **დიაგრამები (მე-5 ლექციის დავალება):**

- * titanic.sqlite ბაზის გამოყენებით ააგეთ გრაფიკი (მოიფიქრეთ გრაფიკის ტიპი), რომელიც აჩვენებს გაყიდული ბილეთების პროცენტულ რაოდენობას ბილეთის კლასის მიხედვით (Pclass ველი).
- * titanic.sqlite ბაზის გამოყენებით ააგეთ გრაფიკი (მოიფიქრეთ გრაფიკის ტიპი), რომელიც აჩვენებს გაყიდული ბილეთების რიცხვით რაოდენობას ბილეთის კლასის მიხედვით (Pclass ველი) და გადარჩენის/ვერ გადარჩენის მიხედვით.

სავარჯიშო 3:

- * geo_stat.sqlite ბაზაში მოცემულია ინფორმაცია საქართველოში მოსახლეობის შესახებ (იხილეთ ბაზის ფაილი სალექციო მასალებში). ცხრილში არსებული სვეტებია (ველებია)

ველის დასახელება	აღწერა
Id	ჩანაწერის ID
year	წლები 2000-დან 2020-მდე
population	მოსახლეობის რაოდენობა წლების მიხედვით
birth	შობადობის რაოდენობა წლების მიხედვით
marriage	ქორწინებათა რაოდენობა წლების მიხედვით
divorce	განქორწინებათა რაოდენობა წლების მიხედვით

- * დაწერეთ შესაბამისი SQL ბრძენაბები და ააგეთ გრაფიკი (წრფივი) თითოეული ველისათვის წლების მიხედვით (მაგ. პოპულაციის გრაფიკი, შობადობის გრაფიკი, ქორწინების/განქორწინების გრაფიკი). დააკვირდით დინამიკაში კონკრეტული მონაცემების ცვლილებას.