Usporedba mogućnosti jezika SQL i biblioteke Pandas za rukovanje podacima

Grubić, Andrija

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Andrija Grubić

Usporedba mogućnosti jezika SQL i biblioteke Pandas za rukovanje podacima

Diplomski rad

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA I INFORMATIKA; SMJER: NASTAVNIČKI

Andrija Grubić

Diplomski rad

Usporedba mogućnosti jezika SQL i biblioteke Pandas za rukovanje podacima

Voditelj diplomskog rada: izv. prof. dr. sc. Maro Cvitan

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2023.

Sažetak

U radu se uspoređuje upravljanje relacijskim bazama podataka pomoću biblioteke Pandas za programski jezik Python s upravljanjem pomoću programskog jezika SQL.Pri tome je radi jednostavnosti korištena implementacija SQL-a pomoću biblioteke SQLite također za programski jezik Python.

Opisane su i uspoređene osnovne operacije u Pandasu i SQL-u, opisane su specifičnosti SQLitea, te su prikazani usporedni primjeri i izvršeni u Jupyterovoj bilježnici. Analizirano je trajanje izvršavanja konkretnih upita u SQLite-u s odgovarajućim upitima implementiranim u Pandasu ovisno o broju redova baze podataka te ovisno o postojanju indeksa u bazi. To je rađeno za veličine baza koje stanu u RAM memoriju, tj. do veličine od milijun redova. Mjerenja potvrđuju očekivano, tj. da je izvršavanje SQLitea u pravilu brže od izrašavanja odgovarajućeg koda koji koristi Pandas, posebno kad su podaci indeksirani. Na primjerima je pokazano da u nekim slučajevima SQLite može biti više redova veličina brži.

S druge strane prednost Pandasa, kao fleksibilnog alata koji omogućuje visoku produktivnost, ilustrirana na primjer računanja linearne regresije. Iz tog primjera je vidljivo da je računanje regresije u samom SQL-u relativno komplicirano budući da pretpostavljamo da je Pandas integriran s Pythonom i mnoštvom biblioteka koje između ostalog omogućuju jednostavno računanje linearne regresije.

Ključne riječi : SQL, skupovi podataka, relacijske baze podataka, baze podataka, Pandas, Jupyterova bilježnica, SQLite, DataFrame, linearna regresija.

Comparison Of Data Handling Features Of The SQL Language With The Pandas Library

Abstract

The paper compares the management of relational databases using the Pandas library for the Python programming language with management by using the SQL programming language. For the sake of simplicity, the implementation of SQL using the SQLite library for the Python programming language was also used.

The basic operations in Pandas and SQL are described and compared, along with the specific features of SQLite. Comparative examples are presented and executed in a Jupyter Notebook. The execution time of specific queries in SQLite with corresponding queries implemented in Pandas is analyzed, depending on the number of rows in the database and the presence of indexes. This was done for database sizes that fit into RAM, i.e., up to one million rows.The measurements confirm the expected result that the execution of SQLite is generally faster than the execution of the equivalent code using Pandas, especially when the data is indexed. In examples, it has been shown that in some cases SQLite can be several orders of magnitude faster

On the other hand, the advantage of Pandas as a flexible tool that enables high productivity is illustrated through an example of calculating linear regression. From this example, it is evident that performing regression calculations in SQL itself is relatively complicated, assuming that Pandas is integrated with Python and a multitude of libraries that, among other things, facilitate easy calculation of linear regression.

Keywords: SQL, data sets, relational databases, databases, Pandas, Jupyter notebook, SQLite, DataFrame, linear regression.

Sadržaj

1. Uvod

SQL predstavlja standardni jezik za rukovanje podacima spremljenim u relacijskim bazama podataka. Python, uz biblioteku Pandas, također omogućuje određeni skup operacija nad podacima smještenim u tablice. U ovom radu opisat će se mogućnosti tih pristupa te usporediti njihovu prikladnost i performanse za operacije dohvaćanja, filtriranja, sortiranja, grupiranja i spajanja.

Prije same usporedbe mogućnosti, proći će se kroz bitne stvari vezane za SQL, Pandas pa i same podatke s kojima se radi. Sljedeće poglavlje bit će posvećeno relacijskim bazama podataka jer će se u njima pohranjivati podatci nad kojima se vrše upiti pa je bitno znati njihovu strukturu i glavna obilježja. U trećem poglavlju proći će se kroz glavna obilježja SQL-a i Pandasa. Osim toga proučit će se još jedna Pythonova biblioteka; sqlite3. Nju koristimo za pisanje SQL koda, ali pošto se u nekim aspektima razlikuje od čistog SQL-a bitno ju je posebno opisati. Nakon toga dolazi poglavlje u kojem se izrađuje baza nad kojom će se vršiti upiti. Za izradu baze i usporedbu mogućnosti jezika SQL i biblioteke Pandas za rukovanje podatcima spomenute baze koristit će se programski jezik Python te biblioteke sqlite3 i Pandas. Sve se radi u Jupyterovoj bilježnici, a rezultati naredbi prikazivat će se ili Pandasovim DataFrameom ili običnim ispisom, ovisno o tome kako se dolazi do podataka.

U petom poglavlju uspoređuje se kod raznih upita čiji su rezultati prikazani različitim pristupima ispisa. U prvom pristupu piše se SQL kod i dohvaćeni podatci se spremaju u listu skupova. Ispis tog pristupa radi se jednostavnom for petljom. Drugi pristup koristi Pandasov DataFrame u koji je spremljena tablica, što znači da je po jedna tablica iz baze spremljena u jedan DataFrame. Korištenjem Pythonovih i Pandasovih mogućnosti nad tim DataFrameom dolazi se do željenih podataka koji se također ispisuju u obliku DataFramea. Cilj rada je usporediti ta dva pristupa, no radi dodatne analize radit će se još jedan pristup. On je kombinacija SQL-a i Pandasa, odnosno koristi Pandasovu metodu `read_sql_query()` koja omogućava da se u Pandasov DataFrame učitaju podatci dobiveni pisanjem SQL upita. Rezultat tog načina također će se ispisivati u obliku DataFramea. Šesto poglavlje govori o načinu na koji se mjeri brzina izvršavanja različitih operacija, prikazuje rezultate mjerenja te sadržava usporedbu dobivenih rezultata.

2. Baze podataka

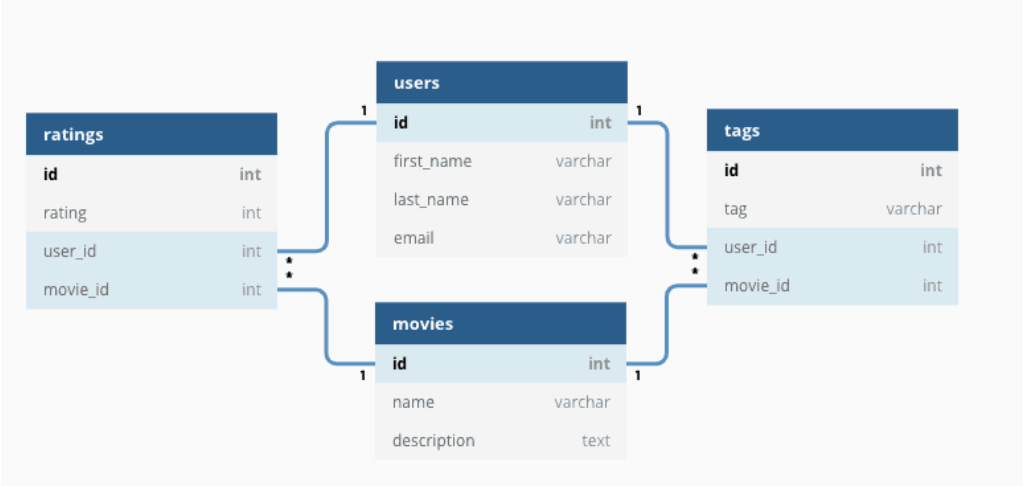
Baza podataka je organizirana kolekcija strukturiranih informacija ili podataka, uobičajeno pohranjenih elektronički u kompjuterski sustav. Za upravljanje i ispitivanje baza podataka koristi se sustav upravljanja bazom podataka (SUBP) odnosno *database management systems* (DBMS). Podatci, SUBP i aplikacije povezane s njima zajedno čine sustav baze podataka. [2]

U svijetu tehnologija za rad s bazama podataka postoje dva glavna tipa baza podataka, a to su: SQL i NoSQL, odnosno postoje relacijske i nerelacijske baze podataka. Razlika je u tome kako su građene, tipu podataka koje pohranjuju te kako ih pohranjuju. Relacijske baze su strukturirane kao telefonski imenik koji sadrži brojeve i adrese korisnika. Nerelacijske baze su više kao dokument, odnosno kao mape datoteka koje sadrže sve podatke o osobi, od njegove adrese i broja telefona, do toga što ta osoba voli kupovati, jesti itd. [2]

Pošto će se kasnije uspoređivati Pandas i SQL bitno je reći nešto više o relacijskim bazama podataka.

2.1. Relacijske baze podataka

Relacijska baza podataka je kolekcija informacija koja organizira podatke s definiranim vezama kako bi im se lakše pristupalo. Bazirana je na relacijskom modelu podataka, predloženom 1970. godine od strane E.F.Codda. Za upravljanje i ispitivanje relacijskih baza podataka koristi se sustav upravljanja relacijskom bazom podataka (SURBP) odnosno *relational database management system* (RDBMS) od kojih je većina napravljena tako da koristi SQL za vršenje upita i ažuriranje podataka. [4]



Slika 2.1 Relacijska baza podataka

Relacijske baze podataka pohranjuju informacije o srodnim objektima unutar relacija. Relacije se sastoje od redova i stupaca. Standardni API (*application program interface*) relacijskih baza podataka je SQL (*structured query language*) no o njemu će se reći više nešto kasnije. Svaka tablica, odnosno relacija, u relacijskoj bazi podataka sadrži jednu ili više kategorija podataka u stupcima. Svaki red, odnosno zapis ili n-torka, dodjeljuje vrijednost svakom svojstvu (stupcu). Svaka relacija ima jedinstven primarni ključ (*primary key*) koji određuje podatke u tablici. Veza između relacija uspostavlja se pomoću stranih ključeva (*foreign key*), odnosno poljem relacije koje je povezano s primarnim ključem neke druge relacije.[3] Postoje definirana pravila za očuvanje integriteta podataka kojih se treba pridržavati kako bi baza bila točna i kako bi se moglo pristupati podatcima u njoj.

Postoji puno prednosti korištenja relacijske baze podataka. Podatci su kategorizirani, odnosno administratori baza podataka lako mogu kategorizirati i pohranjivati podatke u relacije te ih nakon toga filtrirati i nad njima vršiti upite kako bi dobili korisne informacije. Relacijske baze podataka lako se proširuju i ne ovise o fizičkoj građi baze. Nakon što je originalna baza stvorena, novi podatci mogu se dodavati bez potreba za modificiranjem postojećih. Pomoću SQL-a se čak i kompleksni upiti mogu jednostavno pisati i primjenjivati. SQL je jako dobar i po pitanju sigurnosti jer administrator baze podataka može dodjeljivati ili oduzimati prava pristupu podatcima na način koji mu najviše odgovara. Osim toga podatci su pohranjeni samo jednom čime se eliminira dupliciranje podataka, ali i više korisnika može pristupati istoj bazi podataka.

S druge strane, relacijske baze podataka zahtijevaju određenu razinu planiranja strukture jer stupci moraju biti definirani i podatci moraju točno spadati u propisane kategorije. To stvara probleme vezane uz druge manjkavosti, poput održavanja i manjka fleksibilnosti. Ima problema i s održavanjem jer svaki put kada se u bazu podataka dodaju podatci, ona se mora optimizirati što zahtjeva dodatni posao za administratore baze podataka. Relacijske baze podataka su dosta nefleksibilne, odnosno nisu idealne za baratanje velikim količinamanestrukturiranih podataka. Podatci koji nisu lako opisivi i aktivno se mijenjaju nisu
optimalni za ovaj način pohranjivanja podataka jer ako se podatci mijenjaju onda se i shema baze mora mijenjati s njima. Osim toga, teško je rukovati relacijskim bazama podatka preko više servera jer kako se podatci mijenjaju količinski i svojstveno, tako se mijenja i struktura baze pa korištenje više servera loše utječe na efikasnost.

Neke od najpopularnijih relacijskih baza podataka uključuju Microsoft SQL Server, Oracle Database, MySQL te IBM DB2. U današnje doba sve su popularnije Cloud relacijske baze podataka. Neke od njih uključuju Google Cloud SQL, SQL Azure, Oracle Cloud te IBM DB2 on Cloud.

3. SQL I PANDAS

3.1 SQL

Strukturirani upitni jezik (*Structured Query Language*), odnosno SQL, poznat i kao „sequel“ zbog povijesnih razloga, je strukturni upitni jezik, programski jezik visoke razine. Najpopularniji je računalni jezik za izradu, traženje, ažuriranje i brisanje podataka iz relacijskih baza podataka. Standardiziran je preko standarda ANSI (*American National Standards Institute*) i ISO (*International Organization for Standardization*). Stvoren je 1970-tih i od onda ga regularno koriste administratori baza podatka, analizatori podataka i ostali razvojni inženjeri koji puno rade s podatcima. [5]

SQL upiti i ostale operacije napravljeni su kao komande napisane u obliku izjava. Sam SQL služi za modificiranje tablica baze podataka i njene strukture te dohvaćanje podataka. Baziran je na relacijskoj algebri i teoriji skupova. Deklarativan je jezik, ali uključuje i proceduralne elemente. SQL izjave počinju sa SQL naredbom, a završavaju s točkom sa zarezom i „neosjetljive“ su na velika i mala slova, što znači da se mogu pisati malim slovima, velikim slovima ili kombinacijom tog dvoje. Bez obzira na to, uobičajeno se ključne riječi (*keywords*) pišu velikim slovima, a ostalo malim slovima.

Pošto je SQL programski jezik dizajniran za pristup, modificiranje i izvlačenje informacija iz relacijskih baza podataka, on posjeduje naredbe i sintaksu za izvršavanje tih procesa. SQL naredbe s obzirom na svoju „prirodu“ podijeljene su na jezik definiranja podataka (*Data Definition Language*), jezik manipuliranja podatcima (*Data Manipulation Language*), jezik upita (*Data Query Language)*, jezik za kontrolu nad podatcima (*Data Control Language*) i jezik upravljanja transakcijama (*Transaction Control Language*). [6]

Data Definition Language:

- Poznat i kao DDL.
- Naredbe služe za definiranje podataka u tablicama.
- Naredba CREATE stvara novu tablicu, pogled (*view*) tablice ili neki drugi objekt baze podataka.
- Naredba ALTER modificira postojeće objekte baze podataka, npr. tablicu.

- Naredba DROP služi za brisanje tablice, pogleda (*view*) tablice ili nekog drugog objekta baze podataka. Uz nju su usko povezane naredbe ADD, REMOVE, RENAME itd.

Data Manipulation Language:

- Poznat i kao DML.
- Naredbe služe za manipuliranje podataka u postojećim tablicama dodavanjem, mijenjanjem ili brisanjem podataka.
- Koristi ih se u tablicama koje su već obrađene DDL naredbama.
- Naredba INSERT stvara novi zapis.
- Naredba UPDATE modificira zapise.
- Naredba DELETE briše zapise.

Data Query Language:

- Poznat i kao DQL.
- Često ga se izostavlja i grupira pod *Data Manipulation Language*.
- Sastoji se od samo jedne naredbe, SELECT.
- Naredba SELECT koristi se za dohvaćanje određenih zapisa iz jedne ili više tablica.

Data Control Language:

- Poznat i kao DCL.
- Služi za davanje i oduzimanje prava korisnicima.
- Administratori baza podataka imaju sva prava u bazi podataka i svu kontrolu nad pravima.
- Naredba GRANT daje prava korisnicima.
- Naredba REVOKE oduzima prava korisnicima.

Transaction Control Language:

- Poznat i kao TCL.
- Bitan jer se ne spremaju sve promjene nad bazom automatski.

- Naredba COMMIT sprema promjene napravljene s npr. INSERT, DELETE, UPDATE naredbama tako da svi korisnici mogu vidjeti i imati pristup promijenjenoj verziji.
- Naredba ROLLBACK vraća bazu u stanje kakvo je bilo neposredno nakon zadnjeg korištenja naredbe COMMIT, odnosno poništava sve promijene na bazi koje su se dogodile nakon zadnje COMMIT naredbe.

Tipovi podataka u SQL-u podijeljeni su u tri kategorije. Pred definirani tipovi podataka (predefined data types) koji su intrinzično podržani implementacijom, konstruirani tipovi podataka (constructed data types) koji su određeni tipom konstruktora kojim se definiraju te korisničko-definirani tipovi podataka (user-defined data types) koji su usporedivi klasama iz objektno orijentiranog programiranja, a rade se iz pred definiranih tipova.

Pred definirani tipovi podataka su najučestaliji pa je bitno navesti njih i njihove podtipove. *Character* tipovi sadrže CHAR, VARCHAR, CLOB, *National character* tipovi sadrže NCHAR, NCHAR, NCHAR VARYING, NCLOB. Binarni tipovi sadrže BINARY, VARBINARY, BLOB. Numerički tipovi sadrže NUMERIC, DECIMAL, INTEGER, FLOAT, REAL, DECFLOAT. *Datatime* tipovi sadrže DATE, TIME, TIMESTAMP. Osim toga postoje još i Interval, *Boolean*, XML i JSON tipovi.

SQL upit sastoji se od nekoliko važnih ključnih riječi. Između tih riječi dodaju se specifikacije kojima se biraju podatci i njihov prikaz. Kostur klasičnog SQL upita, odnosno pravilno poredane najbitnije ključne riječi, nalaze se ispod. Specifikacije su zamijenjene s „…“.

SELECT… FROM… WHERE…

GROUP BY… HAVING…

ORDER BY…

LIMIT… OFFSET…

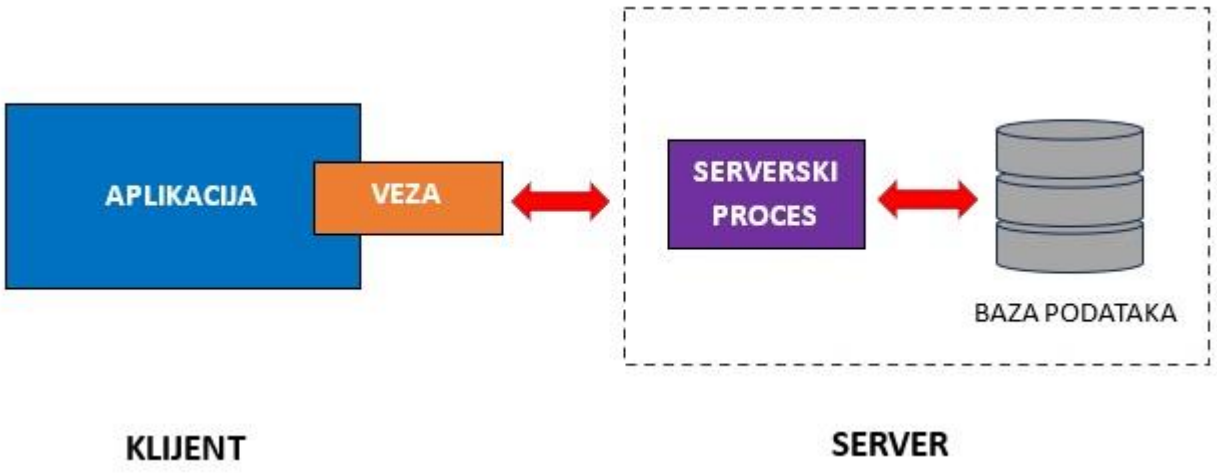
3.2 SQLite

SQLite je biblioteka programskog jezika C koja pruža sustav upravljanja relacijskom bazom podataka (SURBP). Lite u nazivu znači „*lightweight*“ odnosno da je lagan u pogledu postavljanja, administracije baze podataka, brz, pouzdan i da ne zahtjeva puno resursa. Ugrađen je u sve mobilne telefone te u većinu osobnih računala. Također dolazi u sklopu puno drugih aplikacija koje se koriste svakodnevno. Izvorni kod SQLitea je javan i dostupan svima za korištenje u bilo koju svrhu. [7]

Svojstvo SQLitea zbog kojeg se najviše razlikuje od ostalih je to da je baza podataka spremljena u jedan dokument.[8] Zbog toga je jako pristupačna. Kopiranje baze podataka jako je jednostavno te se njezino dijeljenje može postići čak i slanjem privitka u e-mailu.

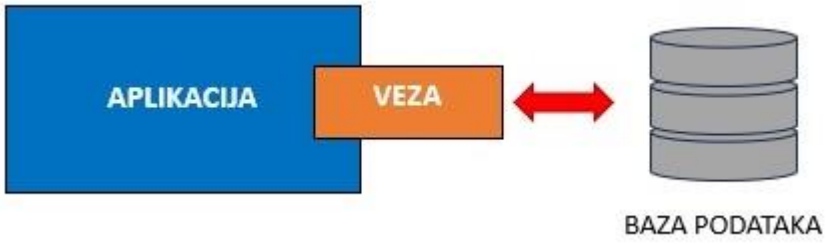
SQLite, za razliku od ostalih softwarea baza podataka, ne provjerava tipove podataka i dozvoljava korisnicima da spremaju podatke bilo kojeg tipa u bilo koji stupac. [9] Ostali softveri bi u tim slučajevima odbili unesene podatke jer se ne slažu sa shemom tablice ili baze. Glavna obilježja SQLitea su to da je bez servera (*serverless*), samostalan (*self-contained*), ne-konfiguracijski (*zero-configuration*) i transakcijski (*transactional*).

Uobičajeno, sustavi upravljanja relacijskim bazama podataka (SURBP) poput MySQL-a ili PostgreSQL-a zahtijevaju poseban serverski proces kako bi radili. Aplikacije koje žele pristup bazi podataka koriste TCP/IP protokol kako bi slale i primale zahtjeve – to se zove klijent/server arhitektura (*client/server architecture*). Na slici ispod prikazan je dijagram koji ilustrira SURBP klijent/server arhitekturu.



Slika 3.1 SURBP klijent/server arhitektura

SQLite ne radi na taj način, jer kako je već napomenuto, ne zahtjeva server kako bi radio. Baza podataka SQLitea integrirana je s aplikacijom koja joj pristupa. To znači da aplikacije koje „komuniciraju“ s SQLite bazom podataka čitaju i pišu podatke direktno iz datoteka baza podataka spremljenih na disku. Na slici ispod prikazan je dijagram koji ilustrira SQLiteovu *server-less* arhitekturu.



Slika 3.2 Prikaz server-less arhitekture

SQLite je samostalan (*self-contained*) jer zahtjeva minimalnu podršku operacijskog sustava ili nekih vanjskih biblioteka. To znači da se SQLite može koristiti u bilo kojoj okolini što ga čini jako pogodnim za uređaje poput iPhonea, Android mobitela, igrače konzole itd. Uz to još je i napravljen koristeći ANSI-C, a izvorni kod je dostupan kao „sqlite3.c“ i „sqlite3.h“ pa se pri izradi bilo koje aplikacije koja koristi SQLite samo treba umetnuti te dokumente u projekt i kompajlirati ih s vlastitim kodom.

SQLite je ne-konfiguracijski (*zero-configuration*) jer ga se ne treba „instalirati“ prije upotrebe i ne koristi nikakve konfiguracijske datoteke.

Kada se kaže da je SQLite transakcijski podrazumijeva se da su sve transakcije u SQLiteu u potpunosti u skladu ACID-a, odnosno svi upiti i promjene su „*Atomic*“, „*Consistent*“, „*Isloated*“, „*Durable*“. To znači da su sve promjene unutar jedne transakcije kompletne ili se uopće ne događaju, čak i u ekstremnim slučajevima poput rušenja aplikacije, rušenja operativnog sustava ili nestanka napajanja.

3.3 Pandas

Pandas je Pythonova biblioteka „otvorenog koda“ (*open source*) koja se najčešće koristi za analizu podataka i strojno učenje. Napravljen je na temelju još jedne Pythonove biblioteke; Numpy, koja pruža mogućnosti za rad s više-dimenzionalnim listama. Kao jedan od najpopularnijih biblioteka za baratanje podatcima, Pandas radi dobro u kombinaciji s većinom ostalih „*data science*“ modula unutar Pythonovog ekosustava i uobičajeno je uključen u svaku Pythonovu distribuciju. Pandas teži tome da bude temelj visoke razine praktičnog analiziranja podataka iz stvarnog svijeta u Pythonu. [10]

Pandas posjeduje brz i efikasan DataFrame objekt za manipulaciju podataka s uključenim indeksiranjem, ali i alate za čitanje i pisanje podataka za različite strukture podataka kao što su CSV (*Comma Separated Values*) i tekstualne datoteke, Microsoft Excel, SQL baze podataka itd. Integrirano je i baratanje s nepostojećim podatcima, dok se postojeći skupovi podataka mogu fleksibilno oblikovati i pivotirati te agregirati ili transformirati pomoću snažne funkcije za grupiranje. Pandas posjeduje veliku brzinu spajanja (*merge*) i združivanja (*join*) skupova podataka, ali i intuitivan način rada s visoko-dimenzionalnim podatcima u nisko-dimenzionalnim strukturama podataka. Bitni dijelovi biblioteke pisani su u C-u ili Cythonu zbog čega su performanse visoko optimizirane. Zbog svega toga Pandas se koristi u akademske i komercijalne svrhe, poput financija, neuroznanosti, ekonomije, statistike, reklamiranja, web analize, itd.

Od svega spomenutog, ono što najviše obilježava Pandas je DataFrame objekt pa je bitno izdvojiti neke od njegovih funkcija nad podatcima: čišćenje, normalizacija, spajanje i združivanje, vizualizacija, statistička analiza, provjera te učitavanje i spremanje. [11]

4. Izrada baze

Već je rečeno da će se sav kod pisati u Jupyterovoj bilježnici i da će se koristiti Python, Pandas i sqlite3 pa je sada sve spremno za izradu baze. Za početak je potrebno uvesti biblioteke ili module koji će se koristiti naredbom import.

```
import pandas as pd
import sqlite3
```

Stvaranje lokalne baze „baza.db“ i uspostavljanje veze s tom bazom vrši se na sljedeći način:

```
conn = sqlite3.connect("C:/Users/marij/AndrijaDipl/baza.db")
```

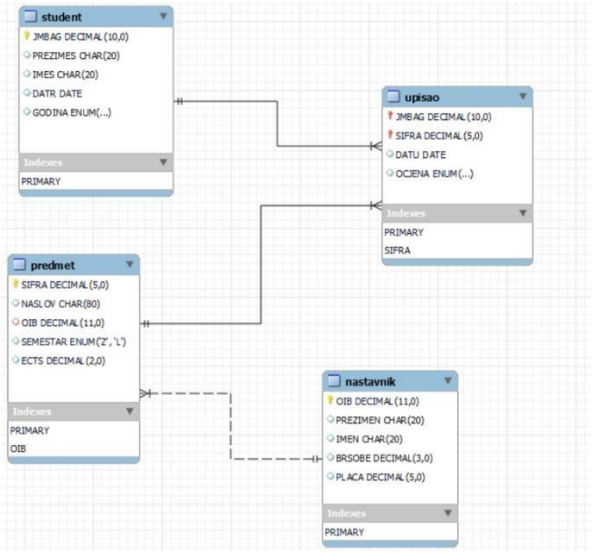
Kako bi se moglo izvršavati SQL naredbe i dohvaćati rezultate SQL upita potrebno je definirati kursor baze podataka.

```
c = conn.cursor()
```

Spremanje promjena vrši se naredbom `commit()`, a poništavanje svog provedenog koda do zadnje naredbe `commit()` postiže se naredbom `rollback()`.

```
conn.commit()
conn.rollback()
```

Prije stvaranja tablica i njihovog popunjavanja poželjno je imati osmišljenu strukturu baze, tipove podataka stupaca u tablicama te njihove primarne i strane ključeve. Bazu podataka popunit će se tablicama STUDENT, PREDMET, NASTAVNIK, UPISAO, a njihove veze, stupci i tipovi podataka prikazani su na sljedećem *Entity Relationship Diagramu*(ERD).



Slika 4.1 Prikaz ERD-a baze koja se stvara. Napravljen pomoću MySQL Workbencha.

Stvaranje tablice postiže se naredbom CREATE TABLE. Sintaksa za stvaranje tablice je :

```
CREATE TABLE table_name

(

column1 datatype constraints,

column2 datatype constraints,

.

.

columnN datatype constraints,

);
```

Toj sintaksi često se dodaje i „IF NOT EXISTS“ dio radi izbjegavanja operacijskih grešaka u slučaju da tablica s tim imenom već postoji. Koristeći sintaksu za stvaranje tablice te praćenjem logike i uvjetima koje postavlja ERD potrebno je napraviti tablice STUDENT, PREDMET, NASTAVNIK i UPISAO.

Stvaranje tablice STUDENT sa stupcima JMBAG, PREZIMES, IMES, DATR, GODINA u kojoj je primarni ključ JMBAG :

```
c.execute('''
CREATE TABLE IF NOT EXISTS STUDENT
(
    JMBAG INTEGER UNSIGNED NOT NULL PRIMARY KEY,
    PREZIMES CHAR(20),
    IMES CHAR(20),
    DATR CHAR(10),
    GODINA TEXT CHECK( GODINA IN (1,2,3,4,5) ) DEFAULT '1'
)
''')
```

Stvaranje tablice NASTAVNIK sa stupcima OIB, PREZIMEN, IMEN, BRSOBE, PLACA u kojoj je primarni ključ OIB :

```
c.execute('''
CREATE TABLE IF NOT EXISTS NASTAVNIK
(
    OIB INTEGER UNSIGNED NOT NULL PRIMARY KEY,
    PREZIMEN CHAR(20),
    IMEN CHAR(20),
    BRSOBE INTEGER UNSIGNED,
    PLACA INTEGER UNSIGNED
)
''')
```

Stvaranje tablice PREDMET sa stupcima SIFRA, NASLOV, OIB, SEMESTAR, ECTS u kojoj je primarni ključ SIFRA, a strani ključ OIB:

```
c.execute('''
CREATE TABLE IF NOT EXISTS PREDMET
(
    SIFRA INTEGER UNSIGNED NOT NULL PRIMARY KEY,
    NASLOV VARCHAR(80),
    OIB INTEGER UNSIGNED,
    SEMESTAR TEXT CHECK( SEMESTAR IN ('L','Z') ) DEFAULT 'Z',
    ECTS INTEGER UNSIGNED,
    FOREIGN KEY (OIB) REFERENCES NASTAVNIK(OIB)
);
''')
```

Stvaranje tablice UPISAO sa stupcima JMBAG, SIFRA, DATU, OCJENA u kojoj su ujedno i primarni i strani ključevi JMBAG i SIFRA:

```
c.execute('''
CREATE TABLE IF NOT EXISTS UPISAO
(
    JMBAG INTEGER UNSIGNED NOT NULL,
    SIFRA INTEGER UNSIGNED NOT NULL,
    DATU CHAR(10),
    OCJENA TEXT CHECK( OCJENA IN ('2','3','4','5') ) DEFAULT NULL,
    PRIMARY KEY(JMBAG, SIFRA),
    FOREIGN KEY (JMBAG) REFERENCES STUDENT(JMBAG),
    FOREIGN KEY (SIFRA) REFERENCES PREDMET(SIFRA)
);
''')
```

Punjenje tablica postiže se naredbom INSERT INTO. Sintaksa za punjenje tablice je :

```
INSERT INTO table_name (col1, col2, .., colN)

VALUES

(val11, val12, .., val1N),

(val21, val22, .., val2N),

(val31, val32, .., val3N),

.

.

(valN1, valN2, .., valNN);
```

Punjenje tablice STUDENT:

```
c.execute('''
INSERT INTO STUDENT(JMBAG, PREZIMES, IMES, DATR, GODINA)
VALUES
(36398757, 'Markovic', 'Marko',"1991-09-05", 1),
(36448430,"Grubisic","Katica","1989-09-11",3),
(165043021,"Kolar", "Ivan","1990-02-27",3),
(246022858,"Vukovic","Janko","1992-09-01",1),
(246022859,"Tokic","Slavko","1989-02-04",4),
(246022875,"Vukovic","Mirko","1988-07-21",4),
(246022888,"Puskarec","Janja", "1986-07-29",5),
(1191203289,"Petrovic","Petar","1990-11-05",2),
(1191205897,"Jankovic","Marija","1992-01-01",1),
(1192130031,"Horvat","Dragica","1990-05-12",2);
''')
```

Punjenje tablice NASTAVNIK:

```
c.execute('''
INSERT INTO NASTAVNIK(OIB, PREZIMEN, IMEN, BRSOBE, PLACA)
VALUES
(13257600947,"Cantor","Georg",102,12000),
(25810043761,"Goedel","Kurt",305,6000),
(33571209458,"Codd","Edgar",127,8000),
(34571209458,"Gold","Sarah",127,13000),
(44102179316,"Klein","Felix",252,12000),
(50076128203,"Pascal","Blaise",101,11000),
(67554120551,"Levinov","Boris",315,9000),
(67741205512,"Turing","Alan",315,10000);
''')
```

Punjenje tablice PREDMET:

```
c.execute('''
INSERT INTO PREDMET(SIFRA, NASLOV, OIB, SEMESTAR, ECTS)
VALUES
(56000,"Programiranje",44102179316,"L",5),
(56001,"Baze podataka",33571209458,"L",5),
(56002,"Programiranje u C-u",67741205512,"Z",5),
(72001,"Linearna algebra",44102179316,"Z",6),
(72005,"Matematicka analiza",25810043761,"L",6),
(72009,"Analiticka geometrija",44102179316,"L",5);
''')
```

Punjenje tablice UPISAO:

```
c.execute('''
INSERT INTO UPISAO(JMBAG, SIFRA, DATU, OCJENA)
VALUES
(36398757,72001,"2010-09-05",NULL),
(36448430,56001,"2009-09-04",5),
(165043021,56000,"2009-06-10",NULL),
(165043021,56001,"2008-10-03",3),
(165043021,56002,"2009-06-10",NULL),
(165043021,72001,"2008-10-03",4),
(246022858,56001,"2010-09-03",NULL),
(246022858,56002,"2010-09-25",NULL),
(246022858,72001,"2010-09-03",NULL),
(246022858,72009,"2010-09-25",NULL),
(246022888,72009,"2010-09-25",5),
(1191203289,56001,"2009-10-03",3),
(1191203289,56002,"2009-10-03",2),
(1191203289,72001,"2007-10-01",3),
(1191205897,72009,"2010-09-05",NULL),
(1192130031,56002,"2009-09-15",4),
(1192130031,72001,"2009-09-15",5),
(1192130031,72009,"2009-09-15",2);
''')
```

Baza je sada spremna za uporabu, ali da bi se mogli uspoređivati SQL upiti koji se vrše nad podatcima u bazi i mogućnosti biblioteke Pandas, podatke iz tablica baze potrebno je „spremiti“ i u Pandas.DataFrame. Svaka tablica baze dobit će odgovarajući DataFrame s odgovarajućim nazivom i podatcima. Postupak punjenja DataFrameova kao i njihov ispis nalazi se na sljedećoj stranici.

STUDENT:

```
c.execute('''SELECT * FROM STUDENT;''')
STUDENT = pd.DataFrame(c.fetchall(), columns = ["JMBAG", "PREZIMES",
"IMES", "DATR", "GODINA"])
```

NASTAVNIK:

```
c.execute('''SELECT * FROM NASTAVNIK''')
NASTAVNIK = pd.DataFrame(c.fetchall(), columns = ["OIB", "PREZIMEN",
"IMEN", "BRSOBE", "PLACA"])
```

PREDMET:

```
c.execute('''SELECT * FROM PREDMET;''')
PREDMET = pd.DataFrame(c.fetchall(), columns = ["SIFRA", "NASLOV", "OIB",
"SEMESTAR", "ECTS"])
```

UPISAO:

```
c.execute('''SELECT * FROM UPISAO;''')
UPISAO = pd.DataFrame(c.fetchall(), columns = ["JMBAG", "SIFRA", "DATU",
"OCJENA"])
```

DataFrameovi se mogu puniti i direktno, a ne samo pomoću SQL-a. Primjer stvaranja kopije tablice PREDMET pomoću liste s podatcima prikazan je ispod:

```
dataPredmet = [(56000,"Programiranje",44102179316,"L",5),
(56001,"Baze podataka",33571209458,"L",5),
(56002,"Programiranje u C-u",67741205512,"Z",5),
(72001,"Linearna algebra",44102179316,"Z",6),
(72005,"Matematicka analiza",25810043761,"L",6),
(72009,"Analiticka geometrija",44102179316,"L",5)]

PREDMETCOPY = pd.DataFrame(dataPredmet, columns = ['SIFRA', 'NASLOV',
'OIB', 'SEMESTAR', 'ECTS'])
```

5. Usporedba upita

U sljedećim potpoglavljima uspoređivat će se funkcionalnosti SQL-a i Pandasa. Svaka naredba bit će objašnjena. Bit će pokazana sintaksa koja se koristi, a zatim će se na bazi podataka napravljenoj u četvrtom poglavlju vršiti razni upiti. Tekstom u kurzivu bit će objašnjeno što se općenito želi postići korištenjem naredbe, a red ispod objašnjenja, koji je točno cilj/zadatak na konkretnoj bazi. Nakon toga bit će ukratko objašnjeno kako se zadani ciljevi postižu s SQL-om, a kako Pandasom. Konačno kod i ispis bit će prikazani na slikama ispod objašnjenja.

5.1 Select

Select je najkorištenija naredba u SQL-u. Koristi se za dohvaćanje podataka iz baze. Dohvaćati se mogu cijele tablice ili samo podatci koji zadovoljavaju postavljene uvjete. Podatci koji su dohvaćeni spremljeni su u rezultantnu tablicu, zvanu još i rezultantnim skupom (*result-set*). Naredbom SELECT specificiraju se stupci koje se želi prikazati kao rezultat upita i, opcionalno, alternativne nazive spomenutih stupaca. Dio koda između
SELECT i FROM označava imena stupaca tablice koje se želi dohvatiti, a dio koda nakon FROM označava tablicu/tablice iz koje se uzimaju podatci.[12] Sintaksa za korištenje SELECT naredbe :

```
SELECT column1, column2, .., columnN FROM table_name;
```
[22]

Dohvaćanje svih podataka iz tablice.

U sljedećim primjerima dohvaćat će se svi podatci iz tablice STUDENT. SQL: zvjezdicom (*) se odabiru svi podatci, a iza naredbe FROM se nalazi željena tablica. Podatci koji se žele dohvatiti spremaju se u varijablu *data* pomoću metode kursora `fetchall()`. Ona dohvaća sve redove rezultantnog skupa SQL upita i vraća ih u obliku liste n-torki. Ispis se tada obavlja jednostavnom for petljom. Pandas: pošto su svi podatci iz tablice STUDENT već spremljeni u Pandasov DataFrame istog imena potrebno je samo ispisati sadržaj spomenute varijable.

```
c.execute('''SELECT * FROM STUDENT;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(36398757, 'Markovic', 'Marko', '1991-09-05', '1')
(36448430, 'Grubisic', 'Katica', '1989-09-11', '3')
(165043021, 'Kolar', 'Ivan', '1990-02-27', '3')
(246022858, 'Vukovic', 'Janko', '1992-09-01', '1')
(246022859, 'Tokic', 'Slavko', '1989-02-04', '4')
(246022875, 'Vukovic', 'Mirko', '1988-07-21', '4')
(246022888, 'Puskarec', 'Janja', '1986-07-29', '5')
(1191203289, 'Petrovic', 'Petar', '1990-11-05', '2')
(1191205897, 'Jankovic', 'Marija', '1992-01-01', '1')
(1192130031, 'Horvat', 'Dragica', '1990-05-12', '2')
```

Primjer 5.1 Dohvaćanje svih podataka tablice pomoću SQL-a

```
STUDENT
```

	JMBAG	PREZIMES	IMES	DATR	GODINA
0	36398757	Markovic	Marko	1991-09-05	1
1	36448430	Grubisic	Katica	1989-09-11	3
2	165043021	Kolar	Ivan	1990-02-27	3
3	246022858	Vukovic	Janko	1992-09-01	1
4	246022859	Tokic	Slavko	1989-02-04	4
5	246022875	Vukovic	Mirko	1988-07-21	4
6	246022888	Puskarec	Janja	1986-07-29	5
7	1191203289	Petrovic	Petar	1990-11-05	2
8	1191205897	Jankovic	Marija	1992-01-01	1
9	1192130031	Horvat	Dragica	1990-05-12	2

Primjer 5.2 Dohvaćanje svih podataka tablice pomoću Pandasa

```
df = pd.read_sql_query('''SELECT *FROM STUDENT;''', conn)
df
```

	JMBAG	PREZIMES	IMES	DATR	GODINA
0	36398757	Markovic	Marko	1991-09-05	1
1	36448430	Grubisic	Katica	1989-09-11	3
2	165043021	Kolar	Ivan	1990-02-27	3
3	246022858	Vukovic	Janko	1992-09-01	1
4	246022859	Tokic	Slavko	1989-02-04	4
5	246022875	Vukovic	Mirko	1988-07-21	4
6	246022888	Puskarec	Janja	1986-07-29	5
7	1191203289	Petrovic	Petar	1990-11-05	2
8	1191205897	Jankovic	Marija	1992-01-01	1
9	1192130031	Horvat	Dragica	1990-05-12	2

Primjer 5.3 Dohvaćanje svih podataka tablice pomoću kombinacije SQL-a i Pandasa

Dohvaćanje više kolona iz jedne tablice.

U sljedećim primjerima dohvaćat će se imena, prezimena i JMBAG-ovi iz tablice STUDENT. SQL: između SELECT i FROM naredbi nalaze se traženi stupci, a nakon FROM je opet željena tablica. Pandas: u željeni DataFrame se šalje lista naziva stupaca koje želimo dohvatiti i pri ispisu se dobivaju svi podatci tih stupaca.

```
c.execute('''SELECT IMES, PREZIMES, JMBAG FROM STUDENT;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
('Marko', 'Markovic', 36398757)
('Katica', 'Grubisic', 36448430)
('Ivan', 'Kolar', 165043021)
('Janko', 'Vukovic', 246022858)
('Slavko', 'Tokic', 246022859)
('Mirko', 'Vukovic', 246022875)
('Janja', 'Puskarec', 246022888)
('Petar', 'Petrovic', 1191203289)
('Marija', 'Jankovic', 1191205897)
('Dragica', 'Horvat', 1192130031)
```

Primjer 5.4 Dohvaćanje više kolona iz jedne tablice pomoću SQL-a

```
STUDENT[["IMES", "PREZIMES", "JMBAG"]]
```

	IMES	PREZIMES	JMBAG
0	Marko	Markovic	36398757
1	Katica	Grubisic	36448430
2	Ivan	Kolar	165043021
3	Janko	Vukovic	246022858
4	Slavko	Tokic	246022859
5	Mirko	Vukovic	246022875
6	Janja	Puskarec	246022888
7	Petar	Petrovic	1191203289
8	Marija	Jankovic	1191205897
9	Dragica	Horvat	1192130031

Primjer 5.5 Dohvaćanje više kolona iz jedne tablice pomoću Pandasa

```
df = pd.read_sql_query('''SELECT IMES, PREZIMES, JMBAG FROM STUDENT;''',
conn)
df
```

	IMES	PREZIMES	JMBAG
0	Marko	Markovic	36398757
1	Katica	Grubisic	36448430
2	Ivan	Kolar	165043021
3	Janko	Vukovic	246022858
4	Slavko	Tokic	246022859
5	Mirko	Vukovic	246022875
6	Janja	Puskarec	246022888
7	Petar	Petrovic	1191203289
8	Marija	Jankovic	1191205897
9	Dragica	Horvat	1192130031

Primjer 5.6 Dohvaćanje više kolona iz jedne tablice pomoću kombinacije SQL-a i Pandasa

Dodavanje nove kolone i korištenje aliasa.

U sljedećim primjerima dohvaćat će se svi podatci iz tablice STUDENT, ali uz to još i JMBAG svakog studenta pomnožen s dva. Taj stupac bit će imenovan DUPLI_JMBAG. SQL: u dijelu uz SELECT zvjezdicom se odabiru svi podatci, a iza zareza, jednostavnom aritmetičkom operacijom, dobiva se i željeni dodatni stupac. Ovim načinom dohvaćanja vide se samo podatci koji se nalaze u stupcima, a ne i nazivi stupaca. Pandas: Korištenjem metode `assign()` DataFrameu se dodaje novi stupac, a u njemu se nalaze podatci određeni unesenim parametrima. Iz kombiniranog primjera vidljivo je da ovaj SQLupit stvarno stvara kolonu imena DUPLI_JMBAG.

```
c.execute('''SELECT *, JMBAG * 2 AS DUPLI_JMBAG FROM STUDENT;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(36398757, 'Markovic', 'Marko', '1991-09-05', '1', 72797514)
(36448430, 'Grubisic', 'Katica', '1989-09-11', '3', 72896860)
(165043021, 'Kolar', 'Ivan', '1990-02-27', '3', 330086042)
(246022858, 'Vukovic', 'Janko', '1992-09-01', '1', 492045716)
(246022859, 'Tokic', 'Slavko', '1989-02-04', '4', 492045718)
(246022875, 'Vukovic', 'Mirko', '1988-07-21', '4', 492045750)
(246022888, 'Puskarec', 'Janja', '1986-07-29', '5', 492045776)
(1191203289, 'Petrovic', 'Petar', '1990-11-05', '2', 2382406578)
(1191205897, 'Jankovic', 'Marija', '1992-01-01', '1', 2382411794)
(1192130031, 'Horvat', 'Dragica', '1990-05-12', '2', 2384260062)
```

Primjer 5.7 Dodavanje nove kolone i korištenje aliasa pomoću SQL-a

```
STUDENT.assign(DUPLI_JMBAG = STUDENT["JMBAG"] * 2)
```

	JMBAG	PREZIMES	IMES	DATR	GODINA	DUPLI_JMBAG
0	36398757	Markovic	Marko	1991-09-05	1	72797514
1	36448430	Grubisic	Katica	1989-09-11	3	72896860
2	165043021	Kolar	Ivan	1990-02-27	3	330086042
3	246022858	Vukovic	Janko	1992-09-01	1	492045716
4	246022859	Tokic	Slavko	1989-02-04	4	492045718
5	246022875	Vukovic	Mirko	1988-07-21	4	492045750
6	246022888	Puskarec	Janja	1986-07-29	5	492045776
7	1191203289	Petrovic	Petar	1990-11-05	2	2382406578
8	1191205897	Jankovic	Marija	1992-01-01	1	2382411794
9	1192130031	Horvat	Dragica	1990-05-12	2	2384260062

Primjer 5.8 Dodavanje nove kolone i korištenje aliasa pomoću Pandasa

```
df = pd.read_sql_query('''SELECT *, JMBAG * 2 AS DUPLI_JMBAG FROM
STUDENT;''', conn)
df
```

	JMBAG	PREZIMES	IMES	DATR	GODINA	DUPLI_JMBAG
0	36398757	Markovic	Marko	1991-09-05	1	72797514
1	36448430	Grubisic	Katica	1989-09-11	3	72896860
2	165043021	Kolar	Ivan	1990-02-27	3	330086042
3	246022858	Vukovic	Janko	1992-09-01	1	492045716
4	246022859	Tokic	Slavko	1989-02-04	4	492045718
5	246022875	Vukovic	Mirko	1988-07-21	4	492045750
6	246022888	Puskarec	Janja	1986-07-29	5	492045776
7	1191203289	Petrovic	Petar	1990-11-05	2	2382406578
8	1191205897	Jankovic	Marija	1992-01-01	1	2382411794
9	1192130031	Horvat	Dragica	1990-05-12	2	2384260062

Primjer 5.9 Dodavanje nove kolone i korištenje aliasa pomoću kombinacije SQL-a i Pandasa

5.2 Where

Koristi se za dohvaćanje podataka po određenom kriteriju i za filtriranje podataka po odgovarajućem uzorku. [13] Sintaksa :

`SELECT column1,column2 FROM table_name WHERE column_name operator value;` [22]

Dohvaćanje svih stupaca pod određenim uvjetom.

U sljedećim primjerima dohvaćat će se svi studenti pod uvjetom da su trenutačno upisani u drugu godinu studija. SQL: kao i prije, naredbom SELECT odabiru se svi podatci iz tablice STUDENT, ali postavljanjem uvjeta korištenjem naredbe WHERE rezultantni skupse smanjuje. Pandas: iako postoji više načina za filtriranje podataka DataFrameova,
najintuitivniji i najpraktičniji način je tzv. „*boolean indexing*“. U DataFrame se tada šalje serija(*Pandas.Series*) True ili False objekata, a ispisuju se samo True redovi. Odnosno:

```
druga_godina = STUDENT["GODINA"] == '2';
druga_godina
```

```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7     True
8    False
9     True
Name: GODINA, dtype: bool
```

Iz ovog se vidi da bi rezultantni skup trebao sadržavati samo dva reda. Slanjem tih podataka u DataFrame STUDENT dobivaju se traženi podatci.

```
c.execute('''SELECT * FROM STUDENT WHERE GODINA = 2;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(1191203289, 'Petrovic', 'Petar', '1990-11-05', '2')
(1192130031, 'Horvat', 'Dragica', '1990-05-12', '2')
```

Primjer 5.10 Dohvaćanje svih stupaca pod određenim uvjetom pomoću SQL-a

```
STUDENT[STUDENT["GODINA"] == '2']
```

	JMBAG	PREZIMES	IMES	DATR	GODINA
7	1191203289	Petrovic	Petar	1990-11-05	2
9	1192130031	Horvat	Dragica	1990-05-12	2

Slika 5.11 Dohvaćanje svih stupaca pod određenim uvjetom pomoću Pandasa

```
df = pd.read_sql_query('''SELECT * FROM STUDENT WHERE GODINA = 2;''',
conn)
df
```

	JMBAG	PREZIMES	IMES	DATR	GODINA
0	1191203289	Petrovic	Petar	1990-11-05	2
1	1192130031	Horvat	Dragica	1990-05-12	2

Primjer 5.12 Dohvaćanje svih stupaca pod određenim uvjetom pomoću kombinacije SQL-a i Pandasa

5.3 AND, OR

U SQL-u, AND i OR operatori koriste se za filtriranje podataka i dohvaćanje preciznih rezultata baziranih na uvjetima. Osim toga služe i za kombiniranje više uvjeta. AND operator prikazuje samo one redove kod kojih su oba uvjeta istinita, a OR prikazuje redove kod kojih je barem jedan uvjet istinit.[14] Sintaksa:

```
SELECT * FROM table_name WHERE condition1 AND condition2 and ...conditionN; [22]
```

```
SELECT * FROM table_name WHERE condition1 OR condition2 OR... conditionN; [22]
```

Dohvaćanje svih stupaca tablice uz dva obavezna uvjeta.

U sljedećim primjerima dohvaćat će se svi nastavnici pod uvjetom da su u sobi s većim brojem od 200 te da im je plaća veća od 9000. SQL: select naredbom odabiru se svi stupci iz tablice NASTAVNIK, a u WHERE djelu postavljena su dva uvjeta odvojena sa AND koji će filtrirati podatke. Pandas: pošto je i ovo filtriranje podataka, i ono se izvršava metodom „*boolean indexing*“ samo što su uvjeti odvojeni operatorom &.

```
c.execute('''SELECT * FROM NASTAVNIK WHERE BRSOBE > 200 AND PLACA > 9000;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(44102179316, 'Klein', 'Felix', 252, 12000)
(67741205512, 'Turing', 'Alan', 315, 10000)
```

Primjer 5.13 Dohvaćanje svih stupaca tablice uz dva obavezna uvjeta pomoću SQL-a

```
NASTAVNIK[(NASTAVNIK["BRSOBE"] > 200) & (NASTAVNIK["PLACA"] > 9000)]
```

	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
4	44102179316	Klein	Felix	252	12000
7	67741205512	Turing	Alan	315	10000

Primjer 5.14 Dohvaćanje svih stupaca tablice uz dva obavezna uvjeta pomoću Pandasa

```
df = pd.read_sql_query('''SELECT * FROM NASTAVNIK WHERE BRSOBE > 200 AND
PLACA > 9000;''', conn)
df
```

	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
0	44102179316	Klein	Felix	252	12000
1	67741205512	Turing	Alan	315	10000

Primjer 5.15 Dohvaćanje svih stupaca tablice uz dva obavezna uvjeta pomoću kombinacije SQL-a i Pandasa

Dohvaćanje svih stupaca tablice uz dva uvjeta od kojih barem jedan mora biti ispunjen.

U sljedećim primjerima dohvaćat će se svi nastavnici pod uvjetom da su u sobi s većim brojem od 200 ili da imaju plaću veću od 9000. SQL: select naredbom odabiru se svi stupci iz tablice NASTAVNIK, a u WHERE djelu postavljena su dva uvjeta odvojena s OR koji će filtrirati podatke. Pandas: pošto je i ovo filtriranje podataka, i ono se izvršava metodom „*boolean indexing*“ samo što su uvjeti odvojeni s operatorom |.

```
c.execute('''SELECT * FROM NASTAVNIK WHERE BRSOBE > 200 OR PLACA >
9000;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(13257600947, 'Cantor', 'Georg', 102, 12000)
(25810043761, 'Goedel', 'Kurt', 305, 6000)
(34571209458, 'Gold', 'Sarah', 127, 13000)
(44102179316, 'Klein', 'Felix', 252, 12000)
(50076128203, 'Pascal', 'Blaise', 101, 11000)
(67554120551, 'Levinov', 'Boris', 315, 9000)
(67741205512, 'Turing', 'Alan', 315, 10000)
```

Primjer 5.16 Dohvaćanje svih stupaca tablice uz dva uvjeta od kojih barem jedan mora biti ispunjenpomoću SQL-a

```
NASTAVNIK[(NASTAVNIK["BRSOBE"] > 200) | (NASTAVNIK["PLACA"] > 9000)]
```

	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
0	13257600947	Cantor	Georg	102	12000
1	25810043761	Goedel	Kurt	305	6000
3	34571209458	Gold	Sarah	127	13000
4	44102179316	Klein	Felix	252	12000
5	50076128203	Pascal	Blaise	101	11000
6	67554120551	Levinov	Boris	315	9000
7	67741205512	Turing	Alan	315	10000

Primjer 5.17 Dohvaćanje svih stupaca tablice uz dva uvjeta od kojih barem jedan mora biti ispunjenpomoću Pandasa

```
df = pd.read_sql_query('''SELECT * FROM NASTAVNIK WHERE BRSOBE > 200 OR
PLACA > 9000;''', conn)
df
```

	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
0	13257600947	Cantor	Georg	102	12000
1	25810043761	Goedel	Kurt	305	6000
2	34571209458	Gold	Sarah	127	13000
3	44102179316	Klein	Felix	252	12000
4	50076128203	Pascal	Blaise	101	11000
5	67554120551	Levinov	Boris	315	9000
6	67741205512	Turing	Alan	315	10000

Primjer 5.18 Dohvaćanje svih stupaca tablice uz dva uvjeta od kojih barem jedan mora biti ispunjenpomoću kombinacije SQL-a i Pandasa

5.4 NULL

Bitno je napomenuti da se NULL vrijednosti razlikuju od objekata kojima je vrijednost nula. NULL, odnosno NaN i None kada se radi o Pythonu označavaju da varijabla nema vrijednost, dok nula (0) znači da je to vrijednost varijable.

Dohvaćanje svih stupaca tablice uz uvjet da postoje vrijednosti koje nisu definirane.

U sljedećim primjerima iz tablice UPISAO dohvaćat će se svi redovi u kojima OCJENA nije unesena. SQL: WHERE dio pomoću IS NULL filtrira podatke. Pandas: metodom `isna()` detektira nedefinirane vrijednosti.

```
c.execute('''SELECT * FROM UPISAO WHERE OCJENA IS NULL;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(36398757, 72001, '2010-09-05', None)
(165043021, 56000, '2009-06-10', None)
(165043021, 56002, '2009-06-10', None)
(246022858, 56001, '2010-09-03', None)
(246022858, 56002, '2010-09-25', None)
(246022858, 72001, '2010-09-03', None)
(246022858, 72009, '2010-09-25', None)
(1191205897, 72009, '2010-09-05', None)
```

Primjer 5.19 Dohvaćanje svih stupaca tablice uz uvjet da postoje vrijednosti koje nisu definiranepomoću SQL-a.

```
UPISAO[UPISAO["OCJENA"].isna()]
```

	JMBAG	SIFRA	DATU	OCJENA
0	36398757	72001	2010-09-05	None
2	165043021	56000	2009-06-10	None
4	165043021	56002	2009-06-10	None
6	246022858	56001	2010-09-03	None
7	246022858	56002	2010-09-25	None
8	246022858	72001	2010-09-03	None
9	246022858	72009	2010-09-25	None
14	1191205897	72009	2010-09-05	None

Primjer 5.20 Dohvaćanje svih stupaca tablice uz uvjet da postoje vrijednosti koje nisu definiranepomoću Pandasa.

```
df = pd.read_sql_query('''SELECT * FROM UPISAO WHERE OCJENA IS NULL''',
conn)
df
```

	JMBAG	SIFRA	DATU	OCJENA
0	36398757	72001	2010-09-05	None
1	165043021	56000	2009-06-10	None
2	165043021	56002	2009-06-10	None
3	246022858	56001	2010-09-03	None
4	246022858	56002	2010-09-25	None
5	246022858	72001	2010-09-03	None
6	246022858	72009	2010-09-25	None
7	1191205897	72009	2010-09-05	None

Primjer 5.21 Dohvaćanje svih stupaca tablice uz uvjet da postoje vrijednosti koje nisu definiranepomoću kombinacije SQL-a i Pandasa

Dohvaćanje svih stupaca tablice uz uvjet da ne postoje vrijednosti koje nisu definirane.

U sljedećim primjerima iz tablice UPISAO dohvaćat će se svi redovi u kojima je OCJENA unesena. SQL: WHERE dio pomoću IS NOT NULL filtrira podatke. Pandas: Metodom notna() detektira definirane vrijednosti.

```
c.execute('''SELECT * FROM UPISAO WHERE OCJENA IS NOT NULL;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(36448430, 56001, '2009-09-04', '5')
(165043021, 56001, '2008-10-03', '3')
(165043021, 72001, '2008-10-03', '4')
(246022888, 72009, '2010-09-25', '5')
(1191203289, 56001, '2009-10-03', '3')
(1191203289, 56002, '2009-10-03', '2')
(1191203289, 72001, '2007-10-01', '3')
(1192130031, 56002, '2009-09-15', '4')
(1192130031, 72001, '2009-09-15', '5')
(1192130031, 72009, '2009-09-15', '2')
```

Primjer 5.22 Dohvaćanje svih stupaca tablice uz uvjet da ne postoje vrijednosti koje nisu definiranepomoću SQL-a

```
UPISAO[UPISAO["OCJENA"].notna()]
```

	JMBAG	SIFRA	DATU	OCJENA
1	36448430	56001	2009-09-04	5
3	165043021	56001	2008-10-03	3
5	165043021	72001	2008-10-03	4
10	246022888	72009	2010-09-25	5
11	1191203289	56001	2009-10-03	3
12	1191203289	56002	2009-10-03	2
13	1191203289	72001	2007-10-01	3
15	1192130031	56002	2009-09-15	4
16	1192130031	72001	2009-09-15	5
17	1192130031	72009	2009-09-15	2

Primjer 5.23 Dohvaćanje svih stupaca tablice uz uvjet da ne postoje vrijednosti koje nisu definiranepomoću Pandasa

```
df = pd.read_sql_query('''SELECT * FROM UPISAO WHERE OCJENA IS NOT
NULL''', conn)
df
```

	JMBAG	SIFRA	DATU	OCJENA
0	36448430	56001	2009-09-04	5
1	165043021	56001	2008-10-03	3
2	165043021	72001	2008-10-03	4
3	246022888	72009	2010-09-25	5
4	1191203289	56001	2009-10-03	3
5	1191203289	56002	2009-10-03	2
6	1191203289	72001	2007-10-01	3
7	1192130031	56002	2009-09-15	4
8	1192130031	72001	2009-09-15	5
9	1192130031	72009	2009-09-15	2

Primjer 5.24 Dohvaćanje svih stupaca tablice uz uvjet da ne postoje vrijednosti koje nisu definiranepomoću kombinacije SQL-a i Pandasa

5.5 Group by

Ova naredba služi za raspoređivanje identičnih podataka u grupe. Koristi se zajedno sa SELECT naredbom, a u upitu dolazi nakon WHERE naredbe, no prije HAVING ili ORDER BY, ako ih se uopće koristi. [15] Sintaksa:

```
SELECT column1, function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1; [22]
```

Grupiranje po jednom stupcu.

U sljedećim primjerima za svaku sobu dohvaćat će se koliko je nastavnika u njoj i kolika je prosječna plaća po sobi. SQL: grupiranje se radi po stupcu BRSOBE pa je standard da taj stupac ide u ispis. Osim njega dohvaćaju se svi stupci na kojima se vrše funkcije da bi se dobili željeni podatci, odnosno AVG(PLACA) i COUNT(*) da bi se dobila prosječna plaća i broj nastavnika za svaku sobu. Pandas: u Pandasu se isti učinak postiže korištenjem metode `groupby()`. Ona razdvaja skup podataka na grupe, zatim vrši agregatne funkcije i konačno povezuje grupe u cjelinu. Metoda `agg()` omogućuje slanje rječnika u grupirani DataFrame i njime određuje koja se funkcija izvršava na koji stupac. Uvodi se biblioteka numpy radi korištenja funkcija `mean()` i `size()`.

```
c.execute('''SELECT BRSOBE, AVG(PLACA) AS SREDNJA_PLACA, COUNT(*)
AS BROJ_NASTAVNIKA FROM NASTAVNIK GROUP BY BRSOBE;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(101, 11000.0, 1)
(102, 12000.0, 1)
(127, 10500.0, 2)
(252, 12000.0, 1)
(305, 6000.0, 1)
(315, 9500.0, 2)
```

Primjer 5.25 Grupiranje po jednom stupcu pomoću SQL-a

```
import numpy as np
NASTAVNIK.groupby("BRSOBE").agg({"PLACA":np.mean, "OIB":np.size})
```

BRSOBE	PLACA	OIB
101	11000	1
102	12000	1
127	10500	2
252	12000	1
305	6000	1
315	9500	2

Primjer 5.26 Grupiranje po jednom stupcu pomoću Pandasa

```
df = pd.read_sql_query('''SELECT BRSOBE, AVG(PLACA) AS SREDNJA_PLACA,
COUNT(*) AS BROJ_NASTAVNIKA FROM NASTAVNIK GROUP BY BRSOBE;''', conn)
df
```

	BRSOBE	SREDNJA_PLACA	BROJ_NASTAVNIKA
0	101	11000.0	1
1	102	12000.0	1
2	127	10500.0	2
3	252	12000.0	1
4	305	6000.0	1
5	315	9500.0	2

Primjer 5.27 Grupiranje po jednom stupcu pomoću kombinacije SQL-a i Pandasa

Grupiranje po više stupaca.

U sljedećim primjerima za svaki semestar dohvaćat će se koliko predmeta nosi koliko ECTS bodova. SQL: sve isto kao prethodni primjer, samo se dodaje još jedna kolona u GROUP BY dio. Pandas: grupiranje po više kolona postiže se slanjem liste stupaca u `groupby()` metodu.

```
c.execute('''SELECT SEMESTAR, ECTS, COUNT(*) AS
BROJ_PREDMETA FROM PREDMET GROUP BY ECTS, SEMESTAR;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
('L', 5, 3)
('Z', 5, 1)
('L', 6, 1)
('Z', 6, 1)
```

Primjer 5.28 Grupiranje po više stupaca pomoću SQL-a

```
PREDMET.groupby(["SEMESTAR", "ECTS"]).agg({"SIFRA":[np.size]})
```

		size
SEMESTAR	ECTS	
L	5	3
	6	1
Z	5	1
	6	1

Primjer 5.29 Grupiranje po više stupaca pomoću Pandasa

```
df = pd.read_sql_query('''SELECT SEMESTAR, ECTS, COUNT(*) AS
BROJ_PREDMETA FROM PREDMET GROUP BY ECTS, SEMESTAR;''', conn)
df
```

	SEMESTAR	ECTS	BROJ_PREDMETA
0	L	5	3
1	Z	5	1
2	L	6	1
3	Z	6	1

Primjer 5.30 Grupiranje po više stupaca pomoću kombinacije SQL-a i Pandasa

5.6 Inner join

Odabire sve redove iz obje tablice koje se spajaju sve dok je postavljeni uvjet ispunjen. Ti redovi tada čine rezultantni skup operacije. [16] Sintaksa:

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column; [23]
```

Jednostavni INNER JOIN s dvije tablice.

U sljedećim primjerima dohvaćat će se svi podatci o svakom predmetu i nastavniku koji drži taj predmet. SQL: rezultat je postignut spajanjem tablica PREDMET i NASTAVNIK pod uvjetom da imaju jednaki OIB. Pandas: slanjem tablica koje se želi spojiti i uvjeta po kojem se spajaju metoda `merge()` izvodi *inner join*. Bitno je napomenuti da za razliku od SQL-a, Pandas spaja željene redove i kada su oba NULL jer to smatra ispunjenim uvjetom jednakosti. Kako bi se bolje vidjelo kako djeluje *inner join*, a kasnije i *left join* u tablicu PREDMET dodaju se još 2 retka, odnosno predmeta, koja neće držati niti jedan nastavnik.

```
c.execute('''SELECT * FROM PREDMET P INNER
JOIN NASTAVNIK N ON P.OIB = N.OIB;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(56000, 'Programiranje', 44102179316, 'L', 5, 44102179316, 'Klein', 'Felix', 252, 12000)
(56001, 'Baze podataka', 33571209458, 'L', 5, 33571209458, 'Codd', 'Edgar', 127, 8000)
(56002, 'Programiranje u C-u', 67741205512, 'Z', 5, 67741205512, 'Turing', 'Alan', 315, 10000)
(72001, 'Linearna algebra', 44102179316, 'Z', 6, 44102179316, 'Klein', 'Felix', 252, 12000)
(72005, 'Matematicka analiza', 25810043761, 'L', 6, 25810043761, 'Goedel', 'Kurt', 305, 6000)
(72009, 'Analiticka geometrija', 44102179316, 'L', 5, 44102179316, 'Klein', 'Felix', 252, 12000)
```

Primjer 5.31 Jednostavni INNER JOIN s dvije tablice pomoću SQL-a

```
pd.merge(PREDMET, NASTAVNIK, on = "OIB")
```

	SIFRA	NASLOV	OIB	SEMESTAR	ECTS	PREZIMEN	IMEN	BRSOBE	PLACA
0	56000	Programiranje	44102179316	L	5	Klein	Felix	252	12000
1	72001	Linearna algebra	44102179316	Z	6	Klein	Felix	252	12000
2	72009	Analiticka geometrija	44102179316	L	5	Klein	Felix	252	12000
3	56001	Baze podataka	33571209458	L	5	Codd	Edgar	127	8000
4	56002	Programiranje u C-u	67741205512	Z	5	Turing	Alan	315	10000
5	72005	Matematicka analiza	25810043761	L	6	Goedel	Kurt	305	6000

Primjer 5.32 Jednostavni INNER JOIN s dvije tablice pomoću Pandasa

```
df = pd.read_sql_query('''SELECT * FROM PREDMET P INNER
JOIN NASTAVNIK N ON P.OIB = N.OIB;''', conn)
df
```

	SIFRA	NASLOV	OIB	SEMESTAR	ECTS	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
0	56000	Programiranje	44102179316	L	5	44102179316	Klein	Felix	252	12000
1	56001	Baze podataka	33571209458	L	5	33571209458	Codd	Edgar	127	8000
2	56002	Programiranje u C-u	67741205512	Z	5	67741205512	Turing	Alan	315	10000
3	72001	Linearna algebra	44102179316	Z	6	44102179316	Klein	Felix	252	12000
4	72005	Matematicka analiza	25810043761	L	6	25810043761	Goedel	Kurt	305	6000
5	72009	Analiticka geometrija	44102179316	L	5	44102179316	Klein	Felix	252	12000

Primjer 5.33 Jednostavni INNER JOIN s dvije tablice pomoću kombinacije SQL-a i Pandasa

5.7 Left outer join

Poznata i pod nazivom LEFT JOIN, ova naredba vraća cijelu „lijevu“ tablicu plus sve redove „desne“ tablice koji ispunjavaju uvjet *joina*. [16] Sintaksa:

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column; [23]
```

Jednostavni LEFT JOIN s dvije tablice.

U sljedećim primjerima dohvaćat će se svi podatci o svakom predmetu i nastavniku koji drži taj predmet, ali i svi predmeti neovisno o tome postoji li nastavnik koji ih predaje. SQL: ista logika kao *inner join* samo se u upitu koristi LEFT OUTER JOIN. Pandas: koristi se ista funkcija kao i za *inner join*, ali uz uvjet *„how = „left“* čime se daje uputa metodi `merge()` da treba izvršiti LEFT OUTER JOIN. Iz ispisa se vidi da postoje dva predmeta (Programiranje 2 i Baze podataka 2) koje ne drži niti jedan nastavnik. Kada njih ne bi bilo, rezultantni skup *left joina* bio bi jednak kao i kod *inner joina*.

```
c.execute('''SELECT * FROM PREDMET P LEFT OUTER JOIN
NASTAVNIK N ON P.OIB = N.OIB;''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(56000, 'Programiranje', 44102179316, 'L', 5, 44102179316, 'Klein', 'Felix', 252, 12000)
(56001, 'Baze podataka', 33571209458, 'L', 5, 33571209458, 'Codd', 'Edgar', 127, 8000)
(56002, 'Programiranje u C-u', 67741205512, 'Z', 5, 67741205512, 'Turing', 'Alan', 315, 10000)
(72001, 'Linearna algebra', 44102179316, 'Z', 6, 44102179316, 'Klein', 'Felix', 252, 12000)
(72005, 'Matematicka analiza', 25810043761, 'L', 6, 25810043761, 'Goedel', 'Kurt', 305, 6000)
(72009, 'Analiticka geometrija', 44102179316, 'L', 5, 44102179316, 'Klein', 'Felix', 252, 12000)
(5603244, 'Programiranje 2', None, 'L', 5, None, None, None, None, None)
(5600112, 'Baze podataka 2', None, 'L', 5, None, None, None, None, None)
```

Primjer 5.34 Jednostavni LEFT JOIN s dvije tablice pomoću SQL-a

```
pd.merge(PREDMET, NASTAVNIK, on = "OIB", how = "left")
```

	SIFRA	NASLOV	OIB	SEMESTAR	ECTS	PREZIMEN	IMEN	BRSOBE	PLACA
0	56000	Programiranje	44102179316	L	5	Klein	Felix	252	12000
1	56001	Baze podataka	33571209458	L	5	Codd	Edgar	127	8000
2	56002	Programiranje u C-u	67741205512	Z	5	Turing	Alan	315	10000
3	72001	Linearna algebra	44102179316	Z	6	Klein	Felix	252	12000
4	72005	Matematicka analiza	25810043761	L	6	Goedel	Kurt	305	6000
5	72009	Analiticka geometrija	44102179316	L	5	Klein	Felix	252	12000

Primjer 5.35 Jednostavni LEFT JOIN s dvije tablice pomoću Pandasa

```
df = pd.read_sql_query('''SELECT * FROM PREDMET P LEFT OUTER JOIN
NASTAVNIK N ON P.OIB = N.OIB;''', conn)
df
```

	SIFRA	NASLOV	OIB	SEMESTAR	ECTS	PREZIMEN	IMEN	BRSOBE	PLACA
0	56000	Programiranje	44102179316	L	5	Klein	Felix	252	12000
1	56001	Baze podataka	33571209458	L	5	Codd	Edgar	127	8000
2	56002	Programiranje u C-u	67741205512	Z	5	Turing	Alan	315	10000
3	72001	Linearna algebra	44102179316	Z	6	Klein	Felix	252	12000
4	72005	Matematicka analiza	25810043761	L	6	Goedel	Kurt	305	6000
5	72009	Analiticka geometrija	44102179316	L	5	Klein	Felix	252	12000

Primjer 5.36 Jednostavni LEFT JOIN s dvije tablice pomoću kombinacije SQL-a i Pandasa

5.8 Union

Naredba UNION koristi se za spajanje dvije odvojene SELECT naredbe i time stvara rezultantni skup koji predstavlja uniju oba *select* dijela upita. Stupci u oba *select* dijela moraju biti napisani istim redoslijedom, imati isti tip podataka te ih mora biti jednak broj. Rezultantni skup naredbe UNION sadržavat će samo jedinstvene zapise pa u slučaju potrebeza svim zapisima (i onima koji se ponavljaju više puta) koristi se naredba UNION ALL.
[17] Sintaksa naredbe UNION:

```
SELECT column_name(s) FROM table1 UNION SELECT
column_name(s) FROM table2; [23]
```

Odnosno za UNION ALL:

```
SELECT column_name(s) FROM table1 UNION ALL SELECT
column_name(s) FROM table2; [23]
```

Unija dva upita s duplikatima.

U sljedećim primjerima dohvaćat će se svi podatci o studentima čija imena počinju slovom „M“ i svi podatci o studentima čija prezimena počinju na slovo „M“. Ispis će uključivati i duplikate. SQL: jedan *select* dio odabire sve studente čija imena počinju na „M“, drugi dio odabire sve studente čija prezimena počinju na „M“ a UNION ALL ih sve spaja u jedan ispis. Pandas: metoda `concat()` u jedan ispis spaja sve članove liste koji se pošalju u nju.

```
c.execute('''SELECT * FROM STUDENT WHERE PREZIMES LIKE 'M%'
UNION ALL SELECT * FROM STUDENT WHERE IMES LIKE 'M%';''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(36398757, 'Markovic', 'Marko', '1991-09-05', '1')
(36398757, 'Markovic', 'Marko', '1991-09-05', '1')
(246022875, 'Vukovic', 'Mirko', '1988-07-21', '4')
(1191205897, 'Jankovic', 'Marija', '1992-01-01', '1')
```

Primjer 5.37 Unija dva upita s duplikatima pomoću SQL-a

```
pd.concat([STUDENT[STUDENT["IMES"].str.get(0) == 'M'],
           STUDENT[STUDENT["PREZIMES"].str.get(0) == 'M']])
```

	JMBAG	PREZIMES	IMES	DATR	GODINA
0	36398757	Markovic	Marko	1991-09-05	1
5	246022875	Vukovic	Mirko	1988-07-21	4
8	1191205897	Jankovic	Marija	1992-01-01	1
0	36398757	Markovic	Marko	1991-09-05	1

Primjer 5.38 Unija dva upita s duplikatima pomoću Pandasa

```
df = pd.read_sql_query('''SELECT * FROM STUDENT WHERE PREZIMES LIKE 'M%'
UNION ALL SELECT * FROM STUDENT WHERE IMES LIKE 'M%';''', conn)
df
```

	JMBAG	PREZIMES	IMES	DATR	GODINA
0	36398757	Markovic	Marko	1991-09-05	1
1	36398757	Markovic	Marko	1991-09-05	1
2	246022875	Vukovic	Mirko	1988-07-21	4
3	1191205897	Jankovic	Marija	1992-01-01	1

Primjer 5.39 Unija dva upita s duplikatima pomoću kombinacije SQL-a i Pandasa

Unija dva upita bez duplikata.

U sljedećim primjerima dohvaćat će se svi podatci o studentima čija imena počinju na slovo „M“ i svi podatci o studentima čija prezimena počinju na slovo „M“. Ispis neće uključivati duplikate. SQL: jedan *select* dio odabire sve studente čija imena počinju na „M“, drugi dio odabire sve studente čija prezimena počinju na „M“. UNION ih spaja u jedan ispis nakon što izbaci sve duplikate. Pandas: izostavljanje duplikata, odnosno repliciranje naredbe UNION iz SQL-a postiže se dodavanjem metode `drop_duplicates()`.

```
c.execute('''SELECT * FROM STUDENT WHERE PREZIMES LIKE 'M%'
UNION SELECT * FROM STUDENT WHERE IMES LIKE 'M%';''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(36398757, 'Markovic', 'Marko', '1991-09-05', '1')
(246022875, 'Vukovic', 'Mirko', '1988-07-21', '4')
(1191205897, 'Jankovic', 'Marija', '1992-01-01', '1')
```

Primjer 5.40 Unija dva upita bez duplikata pomoću SQL-a

```
pd.concat([STUDENT[STUDENT["IMES"].str.get(0) == 'M'],
           STUDENT[STUDENT["PREZIMES"].str.get(0) ==
'M']]).drop_duplicates()
```

	JMBAG	PREZIMES	IMES	DATR	GODINA
0	36398757	Markovic	Marko	1991-09-05	1
5	246022875	Vukovic	Mirko	1988-07-21	4
8	1191205897	Jankovic	Marija	1992-01-01	1

Primjer 5.41 Unija dva upita bez duplikata pomoću Pandasa

```
df = pd.read_sql_query('''SELECT * FROM STUDENT WHERE PREZIMES LIKE 'M%'
UNION SELECT * FROM STUDENT WHERE IMES LIKE 'M%';''', conn)
df
```

	JMBAG	PREZIMES	IMES	DATR	GODINA
0	36398757	Markovic	Marko	1991-09-05	1
1	246022875	Vukovic	Mirko	1988-07-21	4
2	1191205897	Jankovic	Marija	1992-01-01	1

Primjer 5.42 Unija dva upita bez duplikata pomoću kombinacije SQL-a i Pandasa

5.9 Pivot, case

Pivotiranje je proces transformiranja jedne tablice u drugu kako bi se postigao pregledniji prikaz podataka. [18] Može se reći i da se podatci iz redova prebacuju u nazive stupaca. U SQL-u to se može postići operatorom PIVOT, ali i s više CASE naredbi. CASE prolazi kroz uvjete i vraća vrijednost prvog ispunjenog uvjeta, odnosno radi poput *if-else* logike u programiranju. [19] Sintaksa:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END; [23]
```

Pivotiranje

U sljedećim primjerima dohvaćat će se koliko studenata ima na svakoj godini studija. SQL: pošto SQLite ne podržava funkciju PIVOT(), rezultat je postignut korištenjem CASE naredbe za svaku godinu i prikazivanjem toga u stupcima. Pandas: rezultat je postignut metodom `pivot_table()` kojoj se kao parametri šalju tablica s kojom se radi, red i stupac koji se pivotiraju te agregatna funkcija koju se primjenjuje.

```
c.execute('''
SELECT
    count(CASE WHEN GODINA = 1 THEN JMBAG Else null END) as '1',
     count(CASE WHEN GODINA = 2 THEN JMBAG Else null END) as '2',
      count(CASE WHEN GODINA = 3 THEN JMBAG Else null END) as '3',
       count(CASE WHEN GODINA = 4 THEN JMBAG Else null END) as '4',
        count(CASE WHEN GODINA = 5 THEN JMBAG Else null END) as '5'
FROM STUDENT
GROUP BY GODINA;
''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(3, 0, 0, 0, 0)
(0, 2, 0, 0, 0)
(0, 0, 2, 0, 0)
(0, 0, 0, 2, 0)
(0, 0, 0, 0, 1)
```

Primjer 5.43 Pivotiranje pomoću SQL-a

```
df = pd.pivot_table(STUDENT, values = 'JMBAG', columns = 'GODINA',
                    aggfunc = np.count_nonzero)
df
```

GODINA	1	2	3	4	5
JMBAG	3	2	2	2	1

Primjer 5.44 Pivotiranje pomoću Pandasa

```
df = pd.read_sql_query('''
SELECT
    count(CASE WHEN GODINA = 1 THEN JMBAG Else null END) as '1',
     count(CASE WHEN GODINA = 2 THEN JMBAG Else null END) as '2',
      count(CASE WHEN GODINA = 3 THEN JMBAG Else null END) as '3',
       count(CASE WHEN GODINA = 4 THEN JMBAG Else null END) as '4',
        count(CASE WHEN GODINA = 5 THEN JMBAG Else null END) as '5'
FROM STUDENT
GROUP BY GODINA;
''', conn)
df
```

	1	2	3	4	5
0	3	0	0	0	0
1	0	2	0	0	0
2	0	0	2	0	0
3	0	0	0	2	0
4	0	0	0	0	1

Primjer 5.45 Pivotiranje pomoću kombinacije SQL-a i Pandasa

5.10 Update, delete

UPDATE naredba koristi se za ažuriranje podataka postojećih tablica u bazi podataka. Ažuriranje je moguće na jednom ili više stupaca. [21] Sintaksa:

```
UPDATE table_name SET column1 = value1, column2 = value2,...

WHERE condition; [22]
```

DELETE naredba koristi se za brisanje podataka postojećih tablica u bazi podataka. Ovisno o uvjetu u WHERE briše jedan ili više zapisa. Sintaksa:

`DELETE FROM table_name WHERE some_condition;` [22]

Ažuriranje podataka

U sljedećim primjerima sve plaće nastavnika iznad 11 000 smanjivat će se za 1000. SQL: naredbama UPDATE i SET mijenja se vrijednost plaće kada je ispunjen WHERE uvjet. Izvršavaju se dva upita, jedan za ažuriranje, a drugi za dohvaćanje podataka. Pandas: metodom `.loc()` odabiru se samo redovi kojima je plaća veća od 11000 i onda se te redove ažurira.

```
c.execute('''UPDATE NASTAVNIK SET PLACA = PLACA - 1000 WHERE PLACA >
11000;''')
c.execute('''SELECT * FROM NASTAVNIK''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(13257600947, 'Cantor', 'Georg', 102, 11000)
(25810043761, 'Goedel', 'Kurt', 305, 6000)
(33571209458, 'Codd', 'Edgar', 127, 8000)
(34571209458, 'Gold', 'Sarah', 127, 12000)
(44102179316, 'Klein', 'Felix', 252, 11000)
(50076128203, 'Pascal', 'Blaise', 101, 11000)
(67554120551, 'Levinov', 'Boris', 315, 9000)
(67741205512, 'Turing', 'Alan', 315, 10000)
```

Primjer 5.46 Ažuriranje podataka pomoću SQL-a

```
NASTAVNIK.loc[NASTAVNIK["PLACA"] > 11000, "PLACA"] -= 1000
NASTAVNIK
```

	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
0	13257600947	Cantor	Georg	102	11000
1	25810043761	Goedel	Kurt	305	6000
2	33571209458	Codd	Edgar	127	8000
3	34571209458	Gold	Sarah	127	12000
4	44102179316	Klein	Felix	252	11000
5	50076128203	Pascal	Blaise	101	11000
6	67554120551	Levinov	Boris	315	9000
7	67741205512	Turing	Alan	315	10000

Primjer 5.47 Ažuriranje podataka pomoću Pandasa

```
c.execute('''UPDATE NASTAVNIK SET PLACA = PLACA - 1000 WHERE PLACA >
11000;''')
```

```
df = pd.read_sql_query('''SELECT * FROM NASTAVNIK;''', conn)
df
```

	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
0	13257600947	Cantor	Georg	102	11000
1	25810043761	Goedel	Kurt	305	6000
2	33571209458	Codd	Edgar	127	8000
3	34571209458	Gold	Sarah	127	12000
4	44102179316	Klein	Felix	252	11000
5	50076128203	Pascal	Blaise	101	11000
6	67554120551	Levinov	Boris	315	9000
7	67741205512	Turing	Alan	315	10000

Primjer 5.48 Ažuriranje podataka pomoću kombinacije SQL-a i Pandasa

Brisanje podataka

U sljedećim primjerima brisat će se svi zapisi iz tablice NASTAVNIK u kojima je plaća manja od 9000. SQL: naredba DELETE briše sve zapise iz tablice NASTAVNIK u kojima je ostvaren uvjet iz WHERE djela. Pandas: u Pandasu se ne brišu redovi nego se odabiru i ispisuju oni koji zadovoljavaju uvjet. Iz tog razloga primjetna je razlika u rednim brojevima indeksa. Korištenjem SQL-a indeksi kreću od nula i povećavaju se za jedan svakim sljedećim retkom, dok su korištenjem Pandasa prikazani onakvi kakvi bi bili u originalnoj tablici. To je moguće izbjeći korištenjem metode `reset_index()`.

```
c.execute('''DELETE FROM NASTAVNIK WHERE PLACA < 9000;''')
c.execute('''SELECT * FROM NASTAVNIK''')
data = c.fetchall()
for i in data:
    print(i)
```

```
(13257600947, 'Cantor', 'Georg', 102, 12000)
(34571209458, 'Gold', 'Sarah', 127, 13000)
(44102179316, 'Klein', 'Felix', 252, 12000)
(50076128203, 'Pascal', 'Blaise', 101, 11000)
(67554120551, 'Levinov', 'Boris', 315, 9000)
(67741205512, 'Turing', 'Alan', 315, 10000)
```

Primjer 5.49 Brisanje podataka pomoću SQL-a

```
NASTAVNIK = NASTAVNIK.loc[NASTAVNIK["PLACA"] >= 9000]
NASTAVNIK
```

	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
0	13257600947	Cantor	Georg	102	12000
3	34571209458	Gold	Sarah	127	13000
4	44102179316	Klein	Felix	252	12000
5	50076128203	Pascal	Blaise	101	11000
6	67554120551	Levinov	Boris	315	9000
7	67741205512	Turing	Alan	315	10000

Primjer 5.50 Brisanje podataka pomoću Pandasa

```
c.execute('''DELETE FROM NASTAVNIK WHERE PLACA < 9000;''')
df = pd.read_sql_query('''SELECT * FROM NASTAVNIK;''', conn)
df
```

	OIB	PREZIMEN	IMEN	BRSOBE	PLACA
0	13257600947	Cantor	Georg	102	12000
1	34571209458	Gold	Sarah	127	13000
2	44102179316	Klein	Felix	252	12000
3	50076128203	Pascal	Blaise	101	11000
4	67554120551	Levinov	Boris	315	9000
5	67741205512	Turing	Alan	315	10000

Primjer 5.51 Brisanje podataka pomoću kombinacije SQL-a i Pandasa

6. Brzina izvršavanja vs. praktičnost

U ovom poglavlju uspoređivat će se brzina izvršavanja SQL upita i Pandas naredbi nad tablicama različitih dimenzija, počevši od tablica s deset redaka pa sve do tablica s čak 10^6 redaka. Isto će se provjeravati i na indeksiranim tablicama kako bi se dodatno prikazale mogućnosti i razlike ova dva pristupa. Rezultati mjerenja bit će prikazani grafovima i tablicama. Nakon toga pokazat će se razlika u praktičnosti SQL-a i Pandasa na primjeru računanja nagiba jednog od grafova.

Mjerenja vezana za ovo poglavlje rađena su pri sljedećim uvjetima: Pandas verzija 1.5.3, SQLite3 verzija 2.6.0, Windows 11, 32GB RAM-a, procesor s 12 jezgri.

6.1 Usporedba brzine izvršavanja osnovnih upita u SQLiteu i Pandasu

Mjerit će se brzine izvršavanja operacija dohvaćanja, filtriranja, sortiranja, grupiranja te spajanja tablica. Odnosno SELECT, WHERE, SORT, GROUP BY, JOIN upiti u SQLiteu te ekvivalentne naredbe u Pythonovoj biblioteci Pandas. Testirat će se i brzina izvršavanja spomenutih naredbi na indeksiranim i neindeksiranim tablicama te samo vrijeme potrebno da se tablice različitih broja redaka indeksiraju u SQLiteu i Pandasu.

Mjerenje vremena izvršavanja radi se pomoću `timeit.timeit` funkcije. U nju će se slati *`setup`*, odnosno string u kojem se nalazi kod koji se treba izvršiti prije mjerenja vremena, *`stmt`*; string u kojem se nalazi naredba čije vrijeme se želi izmjeriti, te konačno i *`number`*; broj koji govori koliko puta će se izvršiti mjerenje naredbe. Ostale parametre u funkciju se ne šalje pa ih se neće ni detaljnije komentirati.

Osim mjerenja vremena bit će prikazani grafički i tablični prikaz podataka te nagib svakog od grafova. *Kod* pomoću kojeg su rađena mjerenja nalazi se na sljedećim stranicama.

Uvođenje potrebnih biblioteka i spajanje na bazu:

```
import pandas as pd
import sqlite3
import timeit
import random
import matplotlib.pyplot as plt
import numpy as np
import math as m
from scipy.optimize import curve_fit
conn = sqlite3.connect("C:/Users/andra/Diplomski/baza.db")
c = conn.cursor()
```

Pomoćne funkcije i lista s oznakama za grafove :

```
def codeTimeOneTable(n, command, info, j, column):
    string = firstTable
    s = string.format(n, j, column)
    return timeit.timeit(setup = s, stmt = command, number = 1000)/1000

def codeTimeTwoTables(n, command, info, j, column):
    string = firstTable + secondTable
    s = string.format(n, j, column)
    return timeit.timeit(setup = s, stmt = command, number = 1000)/1000

def linearFunc(x,intercept,slope):
    y = intercept + slope * x
    return y

info = [['r^', 'SQLite Bez Indeksa', 'red'], ['go', 'Pandas Bez Indeksa',
'green'], ['ys', 'SQLite S Indeksom', 'yellow'], ['kv', 'Pandas S
Indeksom', 'black']]
infoIndex = [['bo', 'SQLite Indeksiranje', 'blue'], ['rx', 'Pandas
Indeksiranje', 'red']]
```

Stvaranje prve tablice:

```
firstTable ="""
N = {0}
index = {1}
columnToIndex = {2}
firstSifra = list(range(N))
random.shuffle(firstSifra)

c.execute('''drop table if exists NASTAVNIK2;''')
c.execute('''
CREATE TABLE IF NOT EXISTS NASTAVNIK2
(
    OIB INTEGER UNSIGNED NOT NULL PRIMARY KEY,
    SIFRA INTEGER UNSIGNED,
    PREZIMEN CHAR(20),
    IMEN CHAR(20),
    BRSOBE INTEGER UNSIGNED,
    PLACA INTEGER UNSIGNED
)
''')

for i in range(1, N):
    c.execute('''
    INSERT INTO NASTAVNIK2(OIB, SIFRA, PREZIMEN, IMEN, BRSOBE, PLACA)
    VALUES
    (%d, %d,"PREZIME%d","imen",%d, %d)
    ;
    '''%((N+N*0.1)-1-i, firstSifra[i-1], i, random.randint(0,
int(math.sqrt(N))), i%10) )

if index == 2:
    if columnToIndex == 1:
         c.execute('''CREATE INDEX IF NOT EXISTS ind ON
NASTAVNIK2(BRSOBE);''');
    else:
        c.execute('''CREATE INDEX IF NOT EXISTS ind ON
NASTAVNIK2(SIFRA);''');

c.execute('''SELECT * FROM NASTAVNIK2''');
NASTAVNIK2 = pd.DataFrame(c.fetchall(), columns = ["OIB", "SIFRA",
"PREZIMEN", "IMEN", "BRSOBE", "PLACA"])

if index == 3:
    if columnToIndex == 1:
        NASTAVNIK2.set_index('BRSOBE', inplace=True)
        NASTAVNIK2.sort_index(inplace = True)
    else:
        NASTAVNIK2.set_index('SIFRA', inplace=True)
        NASTAVNIK2.sort_index(inplace = True)
"""
```

Stvaranje druge tablice:

```
secondTable ="""
secondSifra = list(range(N))
random.shuffle(secondSifra)

c.execute('''drop table if exists PREDMET2;''')
c.execute('''
CREATE TABLE IF NOT EXISTS PREDMET2
(
    REDNI_BROJ INTEGER UNSIGNED NOT NULL PRIMARY KEY,
    SIFRA INTEGER UNSIGNED,
    NASLOV VARCHAR(80),
    BRSOBE INTEGER UNSIGNED,
    SEMESTAR TEXT CHECK( SEMESTAR IN ('L','Z') ) DEFAULT 'Z',
    ECTS INTEGER UNSIGNED,
    FOREIGN KEY (BRSOBE) REFERENCES NASTAVNIK2(BRSOBE),
    FOREIGN KEY (SIFRA) REFERENCES NASTAVNIK2(SIFRA)
);
''')
for i in range(1, N):
    c.execute('''
    INSERT INTO PREDMET2(REDNI_BROJ, SIFRA, NASLOV, BRSOBE, SEMESTAR,
ECTS)
    VALUES
    (%d, %d,"Programiranje%d",%d,"L", %d)
    ;
    '''%(i, secondSifra[i-1], i, random.randint(0, N), random.randint(0,
N)))

if index == 2:
    c.execute('''CREATE INDEX IF NOT EXISTS ind ON PREDMET2(SIFRA);''');

c.execute('''SELECT * FROM PREDMET2;''')
PREDMET2 = pd.DataFrame(c.fetchall(), columns = ["REDNI_BROJ", "SIFRA",
"NASLOV", "BRSOBE", "SEMESTAR", "ECTS"])

if index == 3:
        PREDMET2.set_index('SIFRA',inplace=True)
        PREDMET2.sort_index(inplace = True)
"""
```

Funkcija za crtanje grafova i izradu DataFramea u kojem se nalaze mjerenja:

```
def graf(dimension, command, info, fun, indexing, column):
    num = 4

    if indexing:
        df = pd.DataFrame({
            "Dimenzija":[],
            "SQL_Stvaranje_Indeksa": [],
            "Pandas_Stvaranje_Indeksa": []
            })
        dimArray = np.array([])
        indexingArray = np.array([])
        indexingPandasArray = np.array([])
        arrays = [indexingArray, indexingPandasArray]
        num = 2

    else:
        df = pd.DataFrame({
            "Dimenzija":[],
            "SQLite_Vrijeme": [],
            "Pandas_Vrijeme": [],
            "SQLite_Vrijeme_S_Indeksom": [],
            "Pandas_Vrijeme_S_Indeksom": [],
        })
        dimArray = np.array([])
        sqlArray = np.array([])
        pandasArray = np.array([])
        indexedSqlArray = np.array([])
        indexedPandasArray = np.array([])
        arrays = [sqlArray, pandasArray, indexedSqlArray,
indexedPandasArray]

    f = 10
    k = 0
    while f <= dimension + 1:
        i = int(f + 0.5)
        f = f * 10**(1/3)
        dimArray = np.append(dimArray, m.log(i, 10))
        for j in range(num):
            timed = fun(i, command[j], info[j], j, column)
            arrays[j] = np.append(arrays[j], m.log(timed, 10))

        if indexing:
            df.loc[i] = [m.log(i, 10), arrays[0][k], arrays[1][k]]
        else:
            df.loc[i] = [m.log(i, 10), arrays[0][k], arrays[1][k],
arrays[2][k], arrays[3][k]]

        df = df.rename_axis('N')
        k = k + 1


    for j in range (num):
```

```
a_fit1,cov1=curve_fit(linearFunc,dimArray[int(2*len(dimArray)/3):],arrays
[j][int(2*len(dimArray)/3):])
        inter1 = a_fit1[0]
        slope1 = a_fit1[1]
        d_inter1 = np.sqrt(cov1[0][0])
        d_slope1 = np.sqrt(cov1[1][1])
        plt.plot(dimArray,arrays[j], info[j][0], label=info[j][1],
fillstyle = 'none')
        yfit1 = inter1 + slope1 * dimArray
        plt.plot(dimArray,yfit1, color  =  info[j][2])
        print(f'Nagib za {info[j][1]} je {slope1}, uz grešku
        {d_slope1}')

    plt.legend()
    plt.xlabel('log(N)')
    plt.ylabel('log(vrijeme/s)')
    plt.show()

    return df
```

Naredbe dohvaćanja podataka čija se brzina izvršavanja mjeri prikazane su ispod:

6.1.1 `c.execute('''SELECT BRSOBE FROM NASTAVNIK2;''')`

6.1.2 `NASTAVNIK2[["BRSOBE"]]`

6.1.3 `NASTAVNIK2.index.to_frame().reset_index(drop = True)`

Kod koji se koristi za mjerenje brzine izvršavanja dohvaćanja podataka:

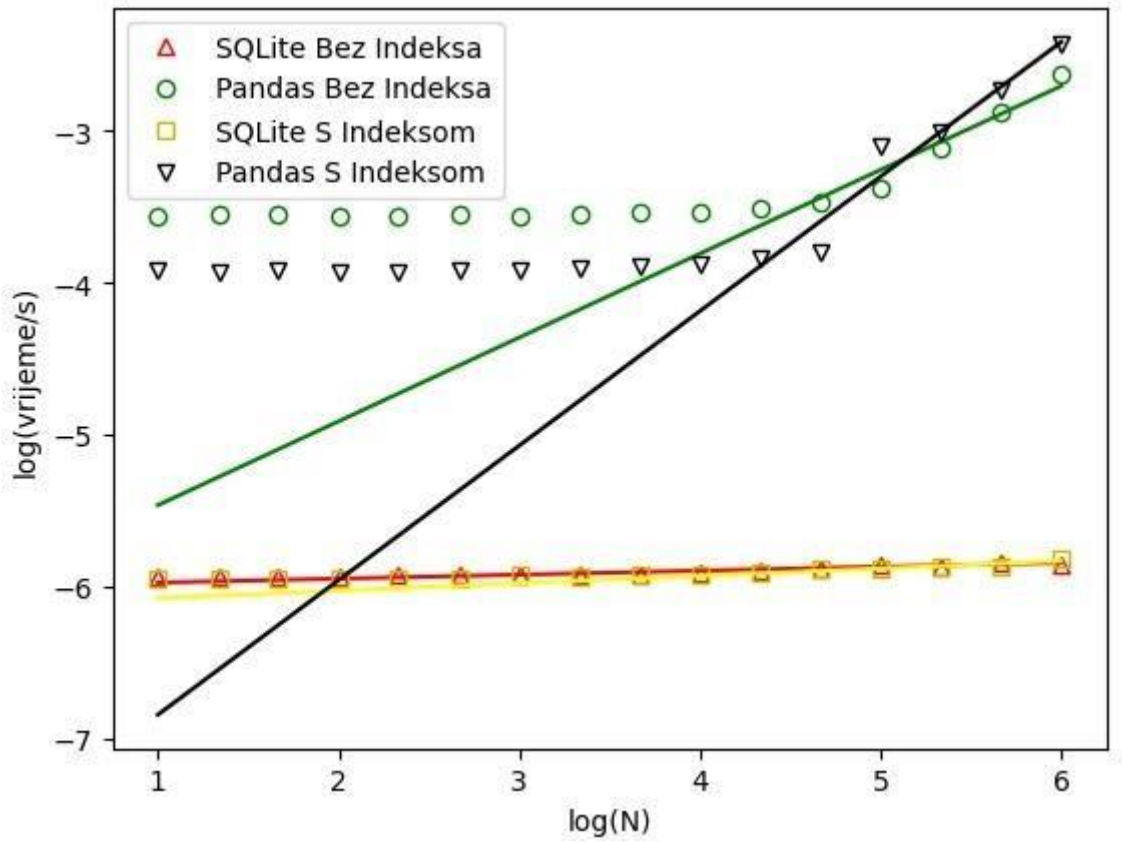
```
selectCommands = [
    """c.execute('''SELECT BRSOBE FROM NASTAVNIK2;''')""",
    """NASTAVNIK2[["BRSOBE"]]""",
    """c.execute('''SELECT BRSOBE FROM NASTAVNIK2;''')""",
    """NASTAVNIK2.index.to_frame().reset_index(drop=True)"""]
SELECT = graf(1000000, selectCommands, info, codeTimeOneTable, False, 1)
```

Naredba 6.1.1 poziva se dva puta. Prvo se mjeri njena brzina izvršavanja na neindeksiranoj tablici, a zatim na indeksiranoj tablici. Brzina izvršavanja naredbe 6.1.2 mjeri se na neindeksiranoj tablici, a 6.1.3 na indeksiranoj tablici. Mjerenje se radi na isti način za sve ostale operacije koje će se provjeravati pa će se dalje kroz rad pisati samo listing naredbi.

Tablica i graf rezultata mjerenja za dohvaćanje podataka:

N	Dimenzija	SQLite_Vrijeme	Pandas_Vrijeme	SQLite_Vrijeme_S_Indeksom	Pandas_Vrijeme_S_Indeksom
10	1.000000	-5.706991	-3.556289	-5.945962	-3.935414
22	1.342423	-5.931221	-3.562272	-5.928523	-3.933971
46	1.662758	-5.942068	-3.560713	-5.940209	-3.940642
100	2.000000	-5.933413	-3.564590	-5.941498	-3.939812
215	2.332438	-5.919085	-3.560166	-5.935168	-3.935584
464	2.666518	-5.915710	-3.563111	-5.929962	-3.933090
1000	3.000000	-5.914246	-3.557271	-5.912396	-3.923068
2154	3.333246	-5.910731	-3.553209	-5.880711	-3.900967
4642	3.666705	-5.915138	-3.549769	-5.912787	-3.913739
10000	4.000000	-5.911191	-3.535851	-5.901737	-3.896413
21544	4.333326	-5.907701	-3.516467	-5.899698	-3.872027
46416	4.666668	-5.883160	-3.470688	-5.900388	-3.822141
100000	5.000000	-5.858738	-3.401804	-5.899905	-3.742825
215443	5.333332	-5.865186	-3.126053	-5.870375	-3.008643
464159	5.666667	-5.872053	-2.905048	-5.866238	-2.704558
1000000	6.000000	-5.865313	-2.643828	-5.854058	-2.441136

Tablica 6.1 Prikaz podataka mjerenja brzine izvršavanja operacija dohvaćanja podataka u SQLiteu i Pythonovoj biblioteci Pandas nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Vrijeme u tablici je logaritmirano.



Graf 6.1 Grafički prikaz podataka mjerenja brzine izvršavanja operacija dohvaćanja podataka u SQLiteu (listing 6.1.1) i Pythonovoj biblioteci Pandas (listing 6.1.2 i listing 6.1.3) nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Nagib SQLitea bez indeksa je -0.20 ± 0.01. Nagib Pandasa bez indeksa je 0.54 ± 0.07. Nagib SQLitea s indeksom je 0.03 ± 0.01. Nagib Pandasa s indeksom je 0.96 ± 0.14.

Naredbe filtriranja podataka čija se brzina izvršavanja mjeri prikazane su ispod:

6.2.1
```
c.execute('''SELECT * FROM NASTAVNIK2 WHERE BRSOBE % 10 = 0;''')
```

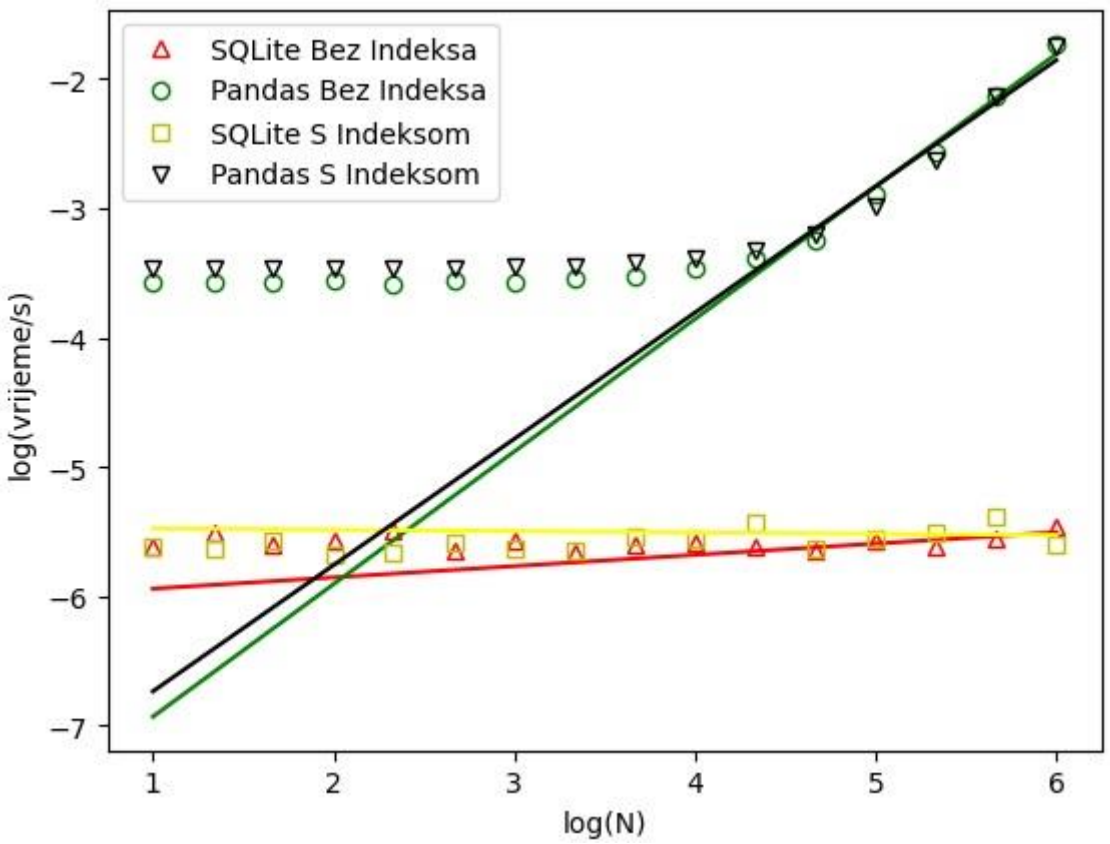
6.2.2
```
NASTAVNIK2[NASTAVNIK2["BRSOBE"] % 10 == 0].reset_index(drop = True)
```

6.2.3
```
NASTAVNIK2[NASTAVNIK2.index % 10 == 0].reset_index()
```

Tablica i graf rezultata mjerenja za filtriranje podataka:

N	Dimenzija	SQLite_Vrijeme	Pandas_Vrijeme	SQLite_Vrijeme_S_Indeksom	Pandas_Vrijeme_S_Indeksom
10	1.000000	-5.401812	-3.655545	-5.414810	-3.808856
22	1.342423	-5.629376	-3.660607	-5.626904	-3.802470
46	1.662758	-5.435050	-3.646390	-5.401943	-3.803070
100	2.000000	-5.487196	-3.649356	-5.487356	-3.800966
215	2.332438	-5.613662	-3.648222	-5.597687	-3.796161
464	2.666518	-5.623168	-3.643800	-5.625178	-3.795314
1000	3.000000	-5.614716	-3.625397	-5.622402	-3.780556
2154	3.333246	-5.619952	-3.610776	-5.616741	-3.767721
4642	3.666705	-5.618091	-3.579797	-5.607725	-3.721421
10000	4.000000	-5.617011	-3.535088	-5.620948	-3.656598
21544	4.333326	-5.610816	-3.450059	-5.611721	-3.548651
46416	4.666668	-5.598686	-3.310235	-5.601418	-3.377014
100000	5.000000	-5.586114	-3.085845	-5.594312	-3.139489
215443	5.333332	-5.586633	-2.646924	-5.589476	-2.681644
464159	5.666667	-5.585394	-2.254276	-5.585729	-2.241751
1000000	6.000000	-5.587170	-1.874155	-5.581699	-1.904664

Tablica 6.2 Prikaz podataka mjerenja brzine izvršavanja operacija filtriranja podataka u SQLiteu i Pythonovoj biblioteci Pandas nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Vrijeme u tablici je logaritmirano(u bazi 10).



Graf 6.2 Grafički prikaz podataka mjerenja brzine izvršavanja operacija filtriranja podataka u SQLiteu (listing 6.2.1) i Pythonovoj biblioteci Pandas (listing 6.2.2 i listing 6.2.3) nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Nagib SQLitea bez indeksa je 0.012 ± 0.004. Nagib Pandasa bez indeksa je 0.97 ± 0.08. Nagib SQLitea s indeksom je 0.016 ± 0.003. Nagib Pandasa s indeksom je 1.01 ± 0.07.

Naredbe sortiranje podataka čija se brzina izvršavanja mjeri prikazane su ispod:

6.3.1
```
c.execute('''SELECT * FROM NASTAVNIK2 ORDER BY BRSOBE ASC;''')
```

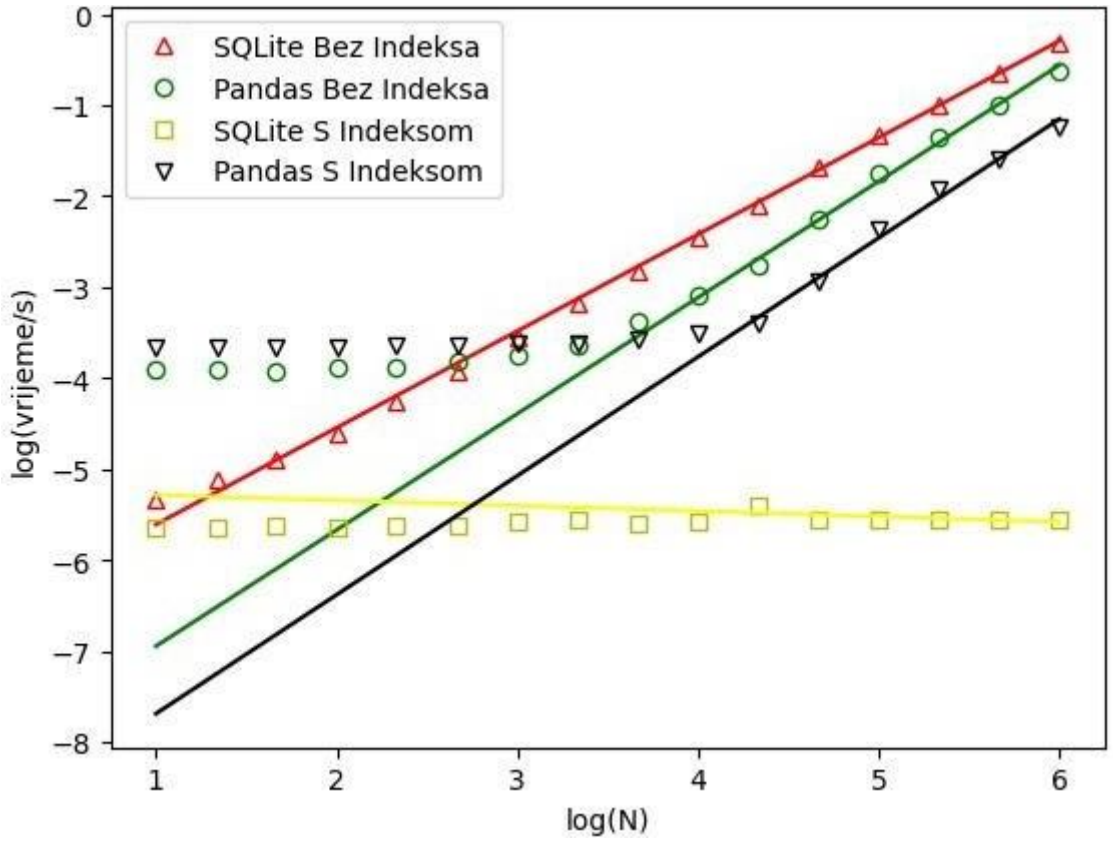
6.3.2
```
NASTAVNIK2.sort_values(by='BRSOBE').reset_index(drop = True)
```

6.3.3
```
NASTAVNIK2.reset_index()
```

Tablica i graf rezultata mjerenja za sortiranje podataka:

N	Dimenzija	SQLite_Vrijeme	Pandas_Vrijeme	SQLite_Vrijeme_S_Indeksom	Pandas_Vrijeme_S_Indeksom
10	1.000000	-5.322886	-3.935381	-5.626408	-3.672783
22	1.342423	-4.870558	-3.868414	-5.636087	-3.679554
46	1.662758	-4.900347	-3.931016	-5.630228	-3.640178
100	2.000000	-4.601701	-3.915486	-5.627254	-3.678616
215	2.332438	-4.241703	-3.897870	-5.624867	-3.670866
464	2.666518	-3.911367	-3.853436	-5.621130	-3.665979
1000	3.000000	-3.570333	-3.795028	-5.615038	-3.652416
2154	3.333246	-3.188738	-3.659156	-5.610002	-3.637184
4642	3.666705	-2.834821	-3.417693	-5.616472	-3.602267
10000	4.000000	-2.488929	-3.125597	-5.611011	-3.535291
21544	4.333326	-2.102924	-2.735666	-5.450200	-3.324318
46416	4.666668	-1.646807	-2.276043	-5.596622	-2.701746
100000	5.000000	-1.314757	-1.735835	-5.579731	-2.398529
215443	5.333332	-0.974264	-1.353408	-5.552671	-1.978874
464159	5.666667	-0.610494	-1.005958	-5.567271	-1.597369
1000000	6.000000	-0.280218	-0.659272	-5.543983	-1.280307

Tablica 6.3 Prikaz podataka mjerenja brzine izvršavanja operacija sortiranja podataka u SQLiteu i Pythonovoj biblioteci Pandas nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Vrijeme u tablici je logaritmirano.



Graf 6.3 Grafički prikaz podataka mjerenja brzine izvršavanja operacija sortiranja podataka u SQLiteu (listing 6.3.1) i Pythonovoj biblioteci Pandas (listing 6.3.2 i listing 6.3.3) nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Nagib SQLitea bez indeksa je 1.07 ± 0.03. Nagib Pandasa bez indeksa je 1.25 ± 0.06. Nagib SQLitea s indeksom je 0.03 ± 0.02. Nagib Pandasa s indeksom je 1.21 ± 0.04.

Naredbe grupiranja podataka čija se brzina izvršavanja mjeri prikazane su ispod:

6.4.1
```
c.execute('''SELECT BRSOBE, SUM(PLACA) AS PLACA FROM
NASTAVNIK2 GROUP BY BRSOBE;'''
```

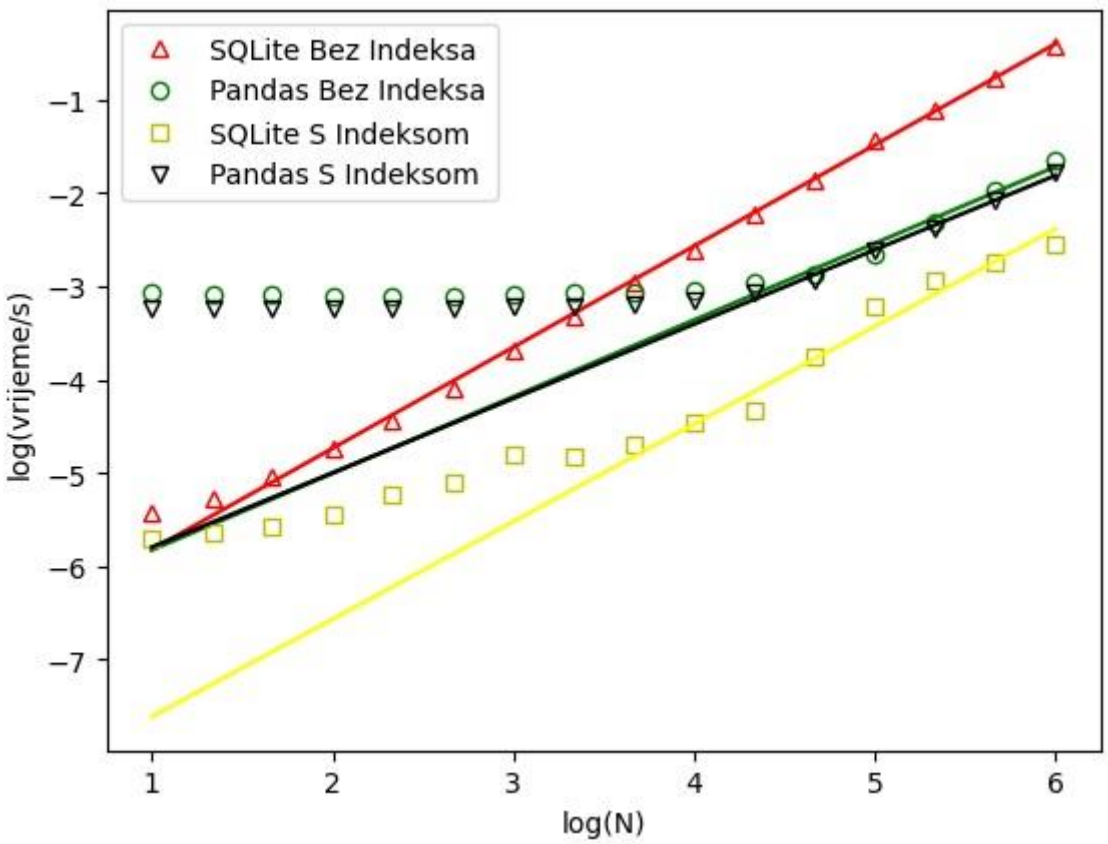
6.4.2
```
NASTAVNIK2.groupby("BRSOBE", as_index = False)["PLACA"].sum()
```

6.4.3
```
NASTAVNIK2.groupby(("BRSOBE", sort = False, as_index =
False)["PLACA"].sum()
```

Tablica i graf rezultata mjerenja za grupiranje podataka:

N	Dimenzija	SQLite_Vrijeme	Pandas_Vrijeme	SQLite_Vrijeme_S_Indeksom	Pandas_Vrijeme_S_Indeksom
10	1.000000	-5.431341	-3.100757	-5.569409	-3.236961
22	1.342423	-5.292498	-3.111781	-5.446797	-3.253400
46	1.662758	-5.052282	-3.111371	-5.602043	-3.253984
100	2.000000	-4.753600	-3.113094	-5.499599	-3.260138
215	2.332438	-4.424534	-3.102246	-5.341092	-3.250240
464	2.666518	-4.085717	-3.105925	-5.116986	-3.247598
1000	3.000000	-3.708206	-3.103837	-5.031578	-3.241871
2154	3.333246	-3.316908	-3.089102	-4.874291	-3.236483
4642	3.666705	-2.955279	-3.074669	-4.620037	-3.209163
10000	4.000000	-2.608458	-3.041167	-4.332861	-3.102056
21544	4.333326	-2.247520	-2.974213	-4.344128	-3.088833
46416	4.666668	-1.862649	-2.860229	-3.737714	-2.969330
100000	5.000000	-1.452245	-2.662190	-3.222082	-2.734637
215443	5.333332	-1.112194	-2.323372	-2.925677	-2.398797
464159	5.666667	-0.771331	-1.988722	-2.756608	-2.072304
1000000	6.000000	-0.417833	-1.656319	-2.498810	-1.766796

Tablica 6.4 Prikaz podataka mjerenja brzine izvršavanja operacija grupiranja podataka u SQLiteu i Pythonovoj biblioteci Pandas nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Vrijeme u tablici je logaritmirano.



Graf 6.4 Grafički prikaz podataka mjerenja brzine izvršavanja operacija grupiranja podataka u SQLiteu (listing 6.4.1) i Pythonovoj biblioteci Pandas (listing 6.4.2 i listing 6.4.3) nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Nagib SQLitea bez indeksa je 1.09 ± 0.02. Nagib Pandasa bez indeksa je 0.82 ± 0.07. Nagib SQLitea s indeksom je 1.1 ± 0.1. Nagib Pandasa s indeksom je 0.83 ± 0.06.

Naredbe za spajanje podataka čija se brzina izvršavanja mjeri prikazane su ispod:

6.5.1
```
c.execute('''SELECT * FROM PREDMET2 P INNER JOIN
NASTAVNIK2 N ON P.SIFRA = N.SIFRA;''')
```

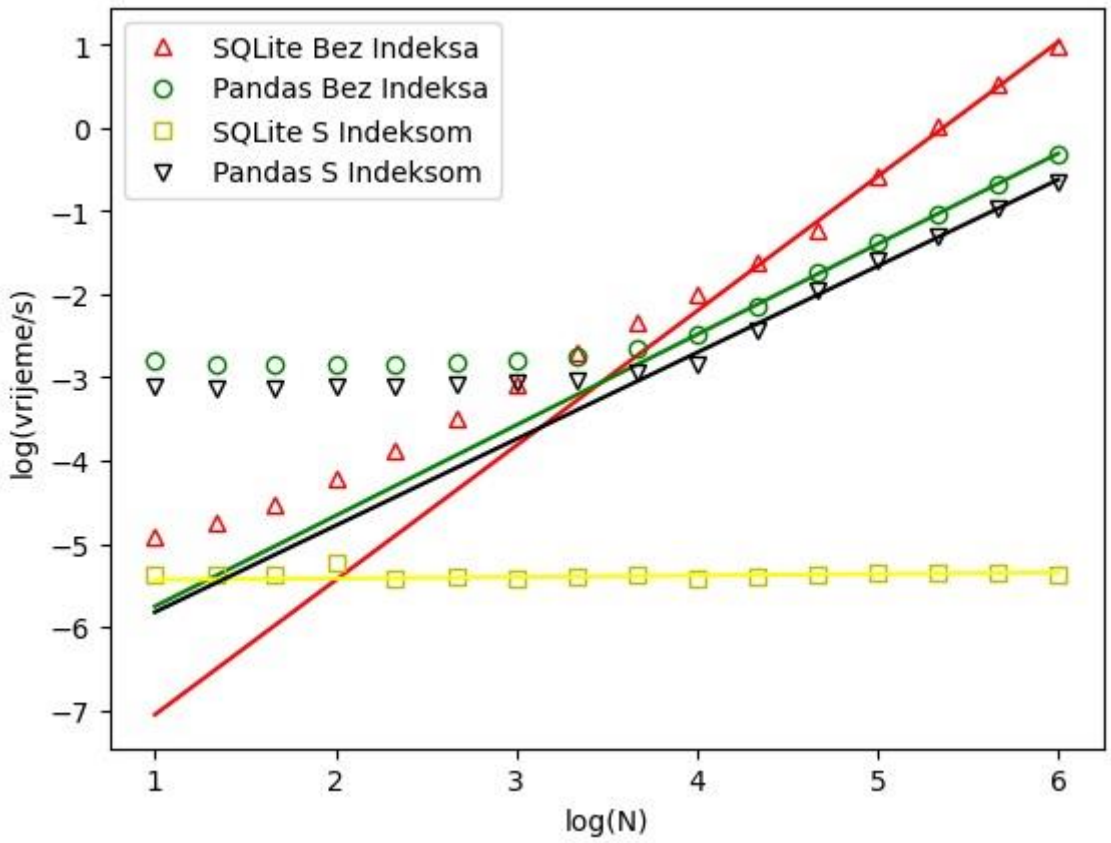
6.5.2
```
PREDMET2.merge(NASTAVNIK2, left_on='SIFRA', right_on='SIFRA')
```

6.5.3
```
PREDMET2.merge(NASTAVNIK2, left_index=True, right_index=True)
```

Tablica i graf rezultata mjerenja za spajanje podataka:

N	Dimenzija	SQLite_Vrijeme	Pandas_Vrijeme	SQLite_Vrijeme_S_Indeksom	Pandas_Vrijeme_S_Indeksom
10	1.000000	-4.886481	-2.905172	-5.448696	-3.036944
22	1.342423	-4.813018	-2.905314	-5.451635	-3.042657
46	1.662758	-4.593632	-2.900134	-5.442842	-3.031892
100	2.000000	-4.319526	-2.899042	-5.481631	-3.024975
215	2.332438	-3.988501	-2.839829	-5.468330	-2.958176
464	2.666518	-3.604549	-2.848448	-5.463454	-2.981026
1000	3.000000	-3.220422	-2.820929	-5.468227	-2.973446
2154	3.333246	-2.899546	-2.782886	-5.456118	-2.860114
4642	3.666705	-2.559249	-2.706759	-5.458421	-2.806326
10000	4.000000	-2.225967	-2.545289	-5.447940	-2.629969
21544	4.333326	-1.880645	-2.136360	-5.433350	-2.286854
46416	4.666668	-1.524418	-1.681659	-5.413480	-1.768254
100000	5.000000	-1.159709	-1.354838	-5.387089	-1.428688
215443	5.333332	-0.817967	-0.971649	-5.386010	-1.047185
464159	5.666667	-0.485563	-0.637452	-5.143676	-0.704198
1000000	6.000000	-0.130398	-0.267128	-5.384292	-0.332542

Tablica 6.5 Prikaz podataka mjerenja brzine izvršavanja operacija spajanja podataka u SQLiteu i Pythonovoj biblioteci Pandas nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Vrijeme u tablici je logaritmirano.



Graf 6.5 Grafički prikaz podataka mjerenja brzine izvršavanja operacija spajanja podataka u SQLiteu (listing 6.5.1) i Pythonovoj biblioteci Pandas (listing 6.5.2 i listing 6.5.3) nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Nagib SQLitea bez indeksa je 1.04 ± 0.01. Nagib Pandasa bez indeksa je 1.04 ± 0.03. Nagib SQLitea s indeksom je 0.029 ± 0.006. Nagib Pandasa s indeksom je 1.12 ± 0.03.

Naredbe za indeksiranje tablica čija se brzina izvršavanja mjeri prikazane su ispod:

6.6.1 `c.execute('''CREATE INDEX IF NOT EXISTS ind ON NASTAVNIK2(BRSOBE);''')`

6.6.2 `NASTAVNIK2.set_index('BRSOBE').sort_index()`

Kod koji se koristi za mjerenje brzine indeksiranja tablice razlikuje se od koda koji se koristio za mjerenje brzine izvršavanja operacija za dohvaćanje, filtriranje, sortiranje, grupiranje te spajanje i nalazi se ispod:

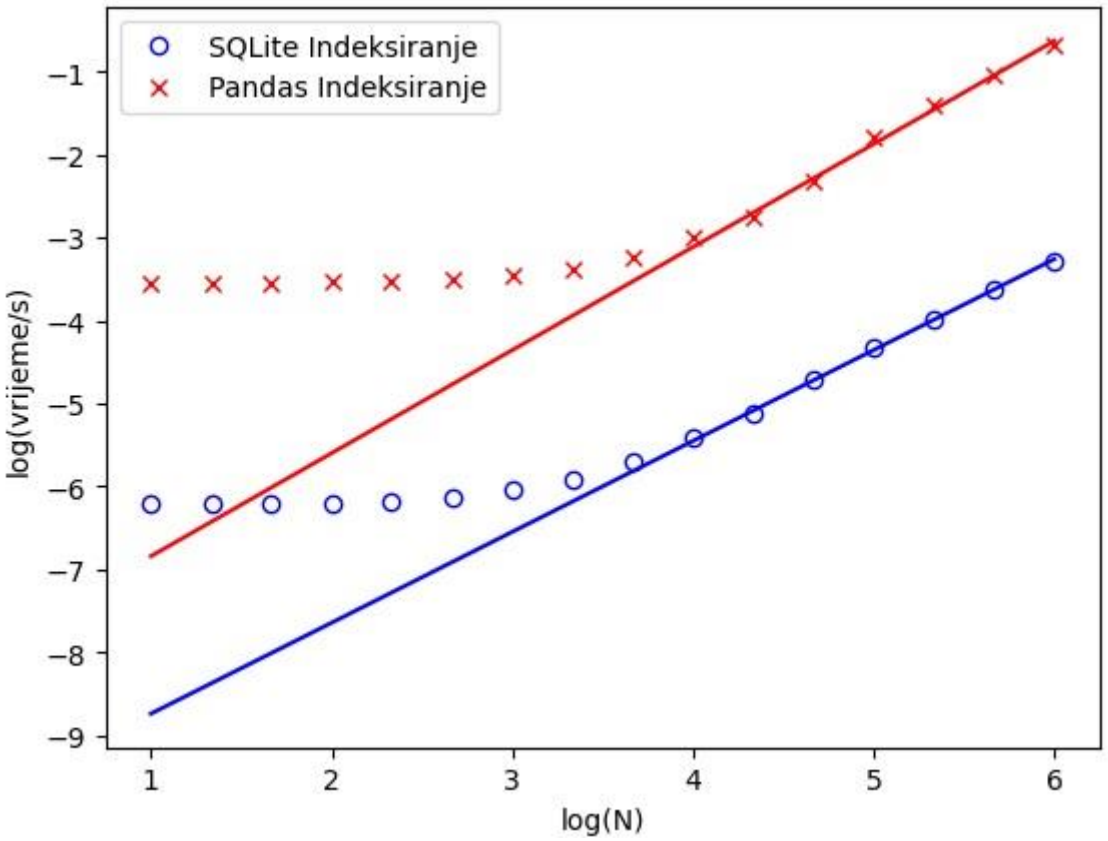
```
indexCommands = [
    """c.execute('''CREATE INDEX IF NOT EXISTS ind ON
NASTAVNIK2(BRSOBE);''')""",
    """NASTAVNIK2.set_index('BRSOBE').sort_index()"""]

INDEX = graf(1000000, indexCommands, infoIndex, codeTimeOneTable,True, 1)
```

Kao i prije, funkcija graf prima veličinu tablice, naredbe kojima mjeri brzinu izvršavanja i funkciju koja mjeri vrijeme, ali ovaj but se šalje argument *True* umjesto *False*, čime se funkciji daje do znanja da mjeri vrijeme indeksiranje tablica, a ne izvršavanja naredbi nad tablicama. Zadnji argument određuje po kojem stupcu se indeksira tablica, za 1 indeksira se stupac BRSOBE, a za 2 indeksira se stupac SIFRA.

Tablica i graf rezultata mjerenja indeksiranja tablica:

N	Dimenzija	SQL_Stvaranje_Indeksa	Pandas_Stvaranje_Indeksa
10	1.000000	-6.198871	-3.803407
22	1.342423	-6.202455	-3.805156
46	1.662758	-6.194499	-3.804136
100	2.000000	-6.192397	-3.802582
215	2.332438	-5.992978	-3.791283
464	2.666518	-6.136439	-3.786865
1000	3.000000	-6.066007	-3.770801
2154	3.333246	-5.960269	-3.747686
4642	3.666705	-5.768301	-3.689291
10000	4.000000	-5.506960	-3.635567
21544	4.333326	-5.224754	-3.511084
46416	4.666668	-4.855136	-3.085717
100000	5.000000	-4.447199	-2.567068
215443	5.333332	-4.107297	-2.130976
464159	5.666667	-3.757769	-1.781304
1000000	6.000000	-3.405992	-1.438276

Tablica 6.6 Prikaz podataka mjerenja brzine izvršavanja operacija stvaranja indeksa u SQLiteu i Pythonovoj biblioteci Pandas nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Vrijeme u tablici je logaritmirano.



Graf 6.6 Grafički prikaz podataka mjerenja brzine izvršavanja operacija stvaranja indeksa u SQLiteu (listing 6.6.1) i Pythonovoj biblioteci Pandas (listing 6.6.2) nad tablicama različitih broja redaka, odnosno tablicama različitih dimenzija. Nagib SQLitea je 1.11 ± 0.02. Nagib Pandasa je 1.25 ± 0.03.

Sažetak podataka mjerenja:

	Naredba	SQLite	Nagib_SQLite	Pandas	Nagib_Pandas
0	SELECT	-5.8653	0.0205 ± 0.0100	-2.6438	0.5431 ± 0.0712
1	SELECT_INDEKSIRAN	-5.8541	0.0309 ± 0.0060	-2.4411	0.9635 ± 0.1437
2	WHERE	-5.5931	0.0116 ± 0.0043	-1.9086	0.9658 ± 0.0783
3	WHERE_INDEKSIRAN	-5.5995	0.0161 ± 0.0106	-1.9289	1.0131 ± 0.0729
4	SORT	-0.2802	1.0654 ± 0.0271	-0.6593	1.2493 ± 0.0564
5	SORT_INDEKSIRAN	-5.5687	-0.0303 ± 0.0193	-1.2803	1.2127 ± 0.0428
6	GROUP_BY	-0.4178	1.0902 ± 0.0246	-1.6563	0.8179 ± 0.0687
7	GROUP_BY_INDEKSIRAN	-2.4988	1.0685 ± 0.1285	-1.7668	0.8260 ± 0.0577
8	JOIN	-0.1463	1.0426 ± 0.0088	-1.1690	1.0416 ± 0.0332
9	JOIN_INDEKSIRAN	-5.3846	0.0289 ± 0.0062	-0.1460	1.1204 ± 0.0256

Tablica 6.7 Prikaz vremena izvođenja svake operacije pri tablicama veličine 10^6 te pripadajući nagibi grafova. Nagibi grafova izračunati su za točke za koje $N > 10^4$

Iz tablice 6.1 i grafa 6.1 vidi se da je pri dohvaćanju podataka, odnosno korištenju SELECT naredbe iz listinga 6.1.1 u SQLiteu i njoj ekvivalentnih naredbi iz listinga 6.1.2 i 6.1.3 u Pythonovoj biblioteci Pandas najbrže izvršena naredba iz listinga 6.1.1, koja se od operacija iz listinga 6.1.2 i 6.1.3 u Pandasu izvršava brže za 2 do 3 reda veličine. Dohvaćanje svih podataka je osnovna operacija koja nije zahtjevna, tako da vrijeme prikazano na grafu interpretiramo kao „hladan pogon“ pri čemu je vidljivo da Pandas troši dosta vremena za konstruiranje rezultantnog DataFramea. Ovo treba uzeti u obzir pri čitanju ostalih grafova.

Iz tablice 6.2 i grafa 6.2 vidi se da je pri filtriranju podataka, odnosno korištenju WHERE naredbe iz listinga 6.2.1 u SQLiteu i njoj ekvivalentnih naredbi iz listinga 6.2.2 i 6.2.3 u Pythonovoj biblioteci Pandas najbrže izvršena naredba iz listinga 6.2.1, koja se od operacija iz listinga 6.2.2 i 6.2.3 u Pandasu izvršava brže za 2 do 4 reda veličine. Vrijeme izvršavanje naredbe iz listinga 6.2.2 na neindeksiranim Pandas DataFrameovima sporije je oko 1.5 puta od naredbe 6.2.3 na indeksiranim Pandas DataFrameovima.

Iz tablice 6.3 i grafa 6.3 vidi se da je pri sortiranju podataka, odnosno korištenju ORDER BY naredbe iz listinga 6.3.1 u SQLiteu i njoj ekvivalentnih naredbi iz listinga 6.3.2 i 6.3.3 u Pythonovoj biblioteci Pandas najbrže izvršena naredba iz listinga 6.3.1 na indeksiranim SQLite tablicama. Ona se izvršava 2 do 5 redova veličine brže od operacija iz listinga 6.3.2 i 6.3.3 u Pandasu. Naredba iz listinga 6.3.2 na neindeksiranim Pandas DataFrameovima izvršava se dvostruko brže od naredbe 6.3.3 na indeksiranim Pandas DataFrameovima kod tablica s brojem redaka do 10^3, a naredba iz listinga 6.3.3 brža je od naredbe iz listinga 6.3.2 čak do 10 puta kad je broj redaka tablice veći od 10^3 .

Iz tablice 6.4 i grafa 6.4 vidi se da je pri grupiranju podataka, odnosno korištenju GROUP BY naredbe iz listinga 6.4.1 u SQLiteu i njoj ekvivalentnih naredbi iz listinga 6.4.2 i 6.4.3 u Pythonovoj biblioteci Pandas najbrže izvršena naredba iz listinga 6.4.1 na indeksiranim SQLite tablicama. Ona se izvršava do 20 puta brže od naredbi iz listinga 6.4.2 i 6.4.3 u Pandasu za $N > 10^5$. Naredba iz listinga 6.4.2 na neindeksiranim Pandas DataFrameovima izvršava se oko 1.5 puta sporije od naredbe iz listinga 6.4.3 naindeksiranima Pandas DataFrameovima.

Iz tablice 6.5 i grafa 6.5 vidi se da je pri spajanju podataka, odnosno korištenju JOIN naredbe iz listinga 6.5.1 u SQLiteu i njoj ekvivalentnih naredbi iz listinga 6.5.2 i 6.5.3 u Pythonovoj biblioteci Pandas najbrže izvršena naredba iz listinga 6.5.1 na indeksiranim SQLite tablicama Ona se izvršava 2 do 5 redova veličine brže od naredbi iz listinga 6.5.2 i 6.5.3 u Pandasu. Naredba iz listinga 6.5.2 na neindeksiranim Pandas tablicama izvršava se oko 1.5 puta sporije od naredbe iz listinga 6.5.3 na indeksiranima Pandas tablicama.

Iz tablice 6.6 i grafa 6.6 vidi se da je za do 2 reda veličine brže indeksirati SQLite tablice naredbom iz listinga 6.6.1 nego indeksirati Pandas DataFrame naredbom iz listinga 6.6.2.

6.2 Usporedba prilagodbe na pravac u SQLiteu i Pandasu

Bitno je spomenuti koliko je Python, a zbog toga i Pandas, njegova biblioteka, praktičniji od SQLitea. U ovom podpoglavlju to će se napraviti na primjeru računanja nagiba grafa. U prethodnom podpoglavlju nagiba grafa izračunat je na sljedeći način:

```
def linearFunc(x,intercept,slope):
    y = intercept + slope * x
    return y

a_fit1,cov1=curve_fit(linearFunc,dimArray[int(2*len(dimArray)/3):],arrays
[j][int(2*len(dimArray)/3):])
```

Potrebna je samo funkcija *curve_fit* te poznavanje jednadžbe pravca. Ovaj pristup koristi se jer spomenuta funkcija ujedno računa i grešku pri računanju nagiba. Osim tog načina postoje funkcije poput *numpy.polyfit()* koje u jednom redu, odnosno pozivanjem jedne funkcije mogu izračunati nagib pravca. Funkcije su već napisane i optimizirane pa ih se samo treba pozvati i primijeniti na podatke.

S druge strane, kod za računanje nagiba pravca pomoću SQLitea izgleda ovako:

```
c.execute('''
SELECT
sql_slope,
sql_bar_max - dim_bar_max * sql_slope as sql_intercept
FROM
(
    SELECT
    sum((Dimenzija - dim_bar) * (SQLite_Vrijeme - sql_bar)) /
sum((Dimenzija - dim_bar) * (Dimenzija - dim_bar)) as sql_slope,
    max(dim_bar) as dim_bar_max,
    max(sql_bar) as sql_bar_max
    FROM
    (
        SELECT
        Dimenzija, avg(Dimenzija) over () as dim_bar,
        SQLite_Vrijeme, avg(SQLite_Vrijeme) over () as sql_bar
        FROM WHERES2
    )
)
''')
data = c.fetchall()
```

Osim što se kod mora pisati od početka i zauzima više linija, on nije optimiziran poput Python koda. Naravno ovo nije jedina funkcija koju se u Pythonu može samo pozvati dok se u SQLiteu mora pisati od početka. Takvih ima mnogo i zato Python i je puno praktičniji od SQLitea.

7. Zaključak

U ovom radu su s ciljem usporedbe SQL naredbi i mogućnosti Pythonove biblioteke Pandas objašnjene struktura i najbitnija svojstva relacijskih baza podataka, programskih jezika SQL-a i SQLitea te biblioteke Pandas. Sa SQL-om se prvotno upoznaje kroz izradu baze na kojoj će se vršiti usporedba upita, a onda kroz usporedbu s Pandasom, dok se s Pandasom upoznaje direktno preko usporedbe sa SQL-om. U poglavlju 6.1 mjerena su vremena izvođenja SELECT, WHERE, SORT, GROUP BY i JOIN naredbi u SQLiteu te njihov ekvivalentne operacije u biblioteci Pandas. Za svaku operaciju mjerilo se vrijeme izvršavanja nad neindekisranim, a zatim i indeksiranim tablicama veličine od deset do 10^6 redaka. U poglavlju 6.2 pokazano je koliko je Python, a samim time i Pandas praktičniji od SQLitea na primjeru računanja nagiba pravca jednog od prethodno spomenutih grafova.

Od testiranih operacija u pravilu su one izvedene pomoću indeksiranih tablica u SQLiteu najbrže, a samo indeksiranje (sortiranje) je dva reda veličine brže u SQLiteu nego u Pandasu. Za N-ove manje od otprilike 10^4 Pandas ima određeno minimalno vrijeme izvršavanja operacije ispod kojeg ne ide ni kad se N smanji. Ako se koriste toliko velike tablice da je brzina bitna, može se skratiti vrijeme izvršavanja za nekoliko redova veličine ako se umjesto Pandasa koristi SQLite. Ako se koriste operacije nad malim tablicama i takvih operacija ima jako puno da je brzina izvršavanja bitna, također se može skratiti vrijeme izvršavanja za nekoliko redova veličine ako se umjesto Pandasa koristi SQLite.

Literatura

[1] SQL and Pandas, (04.03.2018.), https://medium.com/jbennetcodes/how-to-rewrite-your-sql-queries-in-Pandas-and-more-149d341fc53e, 09.02.2023.

[2] Baza podataka, (18.11.2022.), https://hr.wikipedia.org/wiki/Baza_podataka,, 09.02.2023.

[3] Ramakrishnan R., Gehrke J.: Database Management Systems, 3rd Edition. McGraw-Hill, New York, 2002.

[4] Relational database, (09.02.2023.), https://en.wikipedia.org/wiki/Relational_database„ , 09.02.2023.

[5] SQL, (13.04.2022.), https://en.wikipedia.org/wiki/SQL, , 09.02.2023.

[6] Varga M.: Baze podataka - konceptualno, logičko i fizičko modeliranje podataka. DRIP, Zagreb, 1994

[7] SQLite, (27.12.2022.), https://www.sqlite.org/index.html, , 09.02.2023.

[8] What Is SQLite, (03.04.2022.), https://www.sqlitetutorial.net/what-is-sqlite/, 09.02.2023.

[9] SQLite, (08.01.2023.), https://www.codecademy.com/article/what-is-sqlite, 09.02.2023.

[10] About Pandas, (22.12.2022.), https://Pandas.pydata.org/about/index.html, 09.02.2023.

[11] What Can You Do With DataFrames Using Pandas?, (11.01.2023.) https://www.activestate.com/resources/quick-reads/what-is-Pandas-in-Python-everything-you-need-to-know/, 09.02.2023.

[12] SQL: Select Query, (11.12.2022.), https://www.geeksforgeeks.org/sql-select-query/, 09.02.2023.

[13] SQL: Where Clause, (19.05.2022.), https://www.geeksforgeeks.org/sql-where-clause/, 09.02.2023.

[14] SQL AND and OR operators, (04.10.2021.), https://www.geeksforgeeks.org/sql-and-and-or-operators/, 09.02.2023.

[15] SQL: Group by, (25.09.2022.), https://www.geeksforgeeks.org/sql-group-by/, 09.02.2023.

[16] SQL: Join, (15.08.2022.), https://www.geeksforgeeks.org/sql-join-set-1-inner-left-right-and-full-joins/, 09.02.2023.

[17] SQL: Union Clause, (02.09.2019.), https://www.geeksforgeeks.org/sql-union-clause/, 09.02.2023.

[18] Pivot in SQL, (24.08.2019.), https://www.geeksforgeeks.org/pivot-and-unpivot-in-sql/, 09.02.2023.

[19] The SQL CASE Expression, (15.10.2022.), https://www.w3schools.com/sql/sql_case.asp, 09.02.2023.

[20] SQL: DELETE Statement, (05.09.2022.), https://www.geeksforgeeks.org/sql-delete-statement/, 09.02.2023.

[21] SQL: UPDATE Statement, (09.01.2019.), https://www.geeksforgeeks.org/sql-update-statement/, 09.02.2023.

[22] Van der Lans R.F.: Introduction to SQL, 4th Edition. Addison-Wesley, Reading MA, 2006.

[23] Tharp A.L.: File Organization and Processing. John Wiley and Sons, New York, 1988.