Tietokantojen perusteet, kevät 2020

29.02.2020

firstname lastname
student number
e-mail

Harjoitustyön kuvaus

Sovellukseen on toteutettu kaikki tehtävänannon vaatimat toiminnot (enemmän tai vähemmän toimivasti). Käyttäjä voi luoda uuden tietokannan, lisätä tietokantaan uusia paikkoja, asiakkaita, paketteja ja tapahtumia, sekä hakea tietokannasta paketteja tai tapahtumia. Käyttäjä voi myös halutessaan testata tietokannan tehokkuutta. Sovellus on toteutettu Python-kielellä yksinkertaisena komentoriviohjelmana.

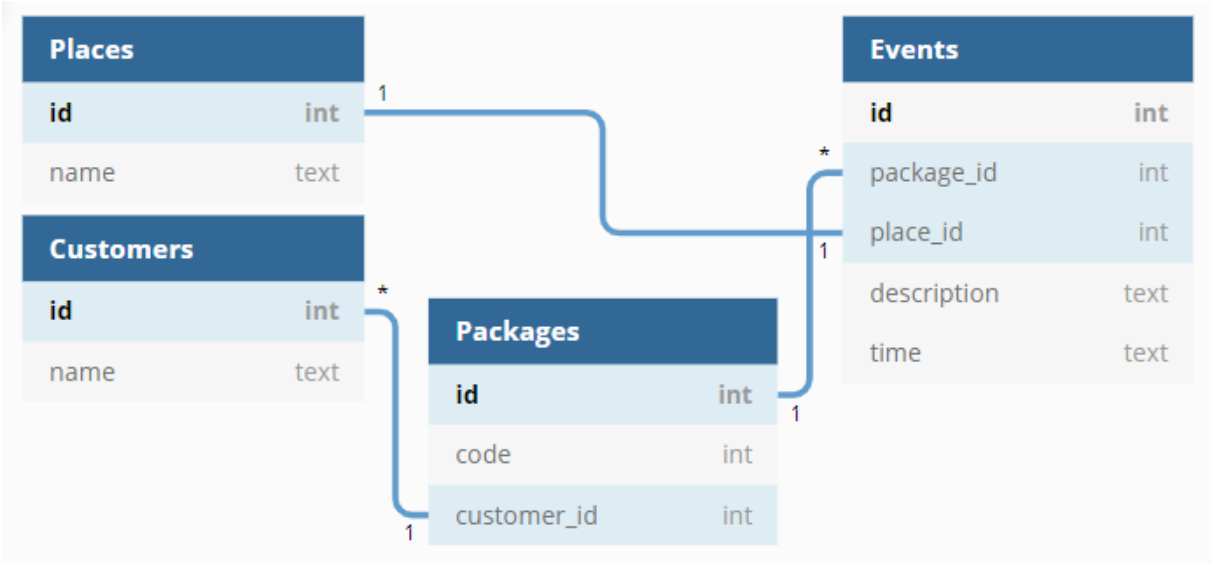
Käyttäjä voi luoda uuden tietokannan, jos sellaista ei vielä ole olemassa. Jos tietokanta löytyy jo ennestään, käyttäjä saa virheilmoituksen yrittäessään luoda tietokantaa. Uusi paikka lisätään antamalla paikan nimi, ja samoin uusi asiakas lisätään antamalla asiakkaan nimi. Uutta pakettia lisätessä käyttäjältä pyydetään lisättävän paketin seurantakoodi sekä pakettiin liittyvän – tietokannasta ennestään löytyvän – asiakkaan nimi. Jos asiakkaan nimi ei ole tietokannassa, käyttäjä saa virheilmoituksen, eikä pakettia voida lisätä tietokantaan. Uusi tapahtuma lisätään antamalla seurattavan paketin seurantakoodi, tapahtuman paikan nimi sekä sanallinen kuvaus tapahtumasta. Sekä paikan että paketin tulee olla valmiiksi tietokannassa tapahtuman lisäyshetkellä. Jos jompikumpi puuttuu, käyttäjälle annetaan virheilmoitus, ja tapahtuma jätetään lisäämättä. Paketin tapahtumia voi hakea seurantakoodin perusteella, ja asiakkaan kaikki paketit sekä niihin liittyvien tapahtumien määrä haetaan asiakkaan nimen perusteella.

Järjestelmän tehokkuustesti suoritetaan erillisellä tietokannalla, jossa on sama rakenne kuin varsinaisessa käyttäjän käyttämässä tietokannassa. Tehokkuustestin aluksi uusi tietokanta luodaan, ja testin päätyttyä tietokantatiedosto poistetaan. Varsinaiseen tietokantaan ei kosketa tehokkuustestin aikana, eikä sitä poisteta missään vaiheessa, myöskään silloin kun käyttäjä poistuu sovelluksesta. Tehokkuustestissä suoritetaan tehtävänannon vaatimat toiminnot ja tulostetaan jokaisen yksittäisen vaiheen kesto millisekunteina käyttäjälle.

Tietokantakaavio ja SQL-skeema

Sovelluksen tietokantakaavio on luotu osoitteen `http://dbdiagram.io` työkalulla.

Kuva 1: Tietokantakaavio



Koodi 1: SQL-skeema

```
1 CREATE TABLE Places (id INTEGER PRIMARY KEY, name TEXT);
2 CREATE TABLE Customers (id INTEGER PRIMARY KEY, name TEXT);
3 CREATE TABLE Packages (id INTEGER PRIMARY KEY, code TEXT, ↩
     customer_id INTEGER);
4 CREATE TABLE Events (id INTEGER PRIMARY KEY, package_id INTEGER, ↩
     place_id INTEGER, description TEXT, time TEXT);
```

Tehokkuustestin tulokset

Ennen varsinaisen tehokkuustestin alkua luodaan listat, joista nimet, indeksit ja muu tarvittava informaatio syötetään tietokantaan. Koska listojen luomiseen tuntui tuhraantuvan paljon aikaa, päädyin tallentamaan listat *pickle*-kirjaston tiedostoiksi, jotta niihin pääsisi nopeammin käsiksi. Tämä oli relevanttia lähinnä silloin, kun yritin saada tehokkuustestiä toimimaan ja ajoin ohjelmaa monta kertaa peräkkäin, mutta jätin toiminnon kuitenkin lähdekoodiin. En ole ihan varma, miten tiedon lisääminen tietokantaan olisi oikeaoppisesti kuulunut toteuttaa. En myöskään ollut ihan varma, miten ohjelman suoritusaikaa olisi parasta mitata Pythonilla, mutta päädyin lopulta käyttämään *time*-moduulin funktiota *perf_counter()*.

Taulukko 1: Tehokkuustestin tulokset

Vaiheet	Indekseittä	Indekseillä
Vaihe 1	2 ms	2 ms
Vaihe 2	2 ms	2 ms
Vaihe 3	2 ms	2 ms
Vaihe 4	2169 ms	2246 ms
Vaihe 5	138747 ms	261 ms
Vaihe 6	136108 ms	225 ms

Vaiheet 4, 5 ja 6 vievät mahdollisesti enemmän aikaa kuin olisi tarkoitus, vaikka tietokantaan on lisätty indeksit. On mahdollista, etten osannut tehdä indeksointia oikein, tai sitten muodostin kyselyiden transaktiot väärin.

Ongelmakohdat

Kahta samannimistä paikkaa tai asiakasta ei pitäisi olla mahdollista syöttää tietokantaan: käyttäjälle annetaan virheilmoitus, jos hän yrittää vaikkapa lisätä tietokantaan asiakasta, jonka nimellä löytyy jo ennestään asiakas, eikä lisäyksen anneta tapahtua. En ole ihan varma, miten tietokanta käyttäytyisi useamman samanaikaisen käyttäjän kanssa.

Lähdekoodi

Sovelluksen lähdekoodi on kopioitu alle kokonaisuudessaan. Valitettavasti koodi lienee paikoitellen varsin vaikeaselkoista. Jos aika olisi riittänyt, olisin todennäköisesti muuttanut luokkien logiikkaa ja siirtänyt osan *main*in toiminnoista muualle siistimpiin kokonaisuuksiin. Sovelluksen pitäisi kuitenkin toimia odotetulla tavalla.

Koodi 2: Lähdekoodi

```
1  import sqlite3
2  import datetime
3  import pickle
4  import random
5  import time
6  import os
7
8  ### Tarkista, että tietokanta on luotu (tiedosto löytyy).
9  def database_exists(db_filename):
```

```
10     if os.path.exists(db_filename):
11        return True
12     else:
13        print("VIRHE: Tietokantaa ei ole luotu")
14        return False
15
16  ### Tulosta suoritusaika.
17  def get_time(start_time, phase, return_time=True):
18     print("Vaihe {}: {} ms".format(phase, ↩
          round((time.perf_counter() - start_time) * 1000)))
19     if return_time:
20        return time.perf_counter()
21
22  ### Tulosta paketin tapahtumat "nätimpänä" käyttäjälle.
23  def pprint_events(events):
24     string = ""
25     for event in events:
26        string += ", ".join(event) + "\n"
27     print(string)
28
29  ### Tulosta paketin tapahtumien määrä "nätimpänä" käyttäjälle.
30  def pprint_events_counts(packages, event_counts):
31     string = ""
32     for package_code, count in zip(packages, event_counts):
33        word_form = (" tapahtuma" if count[0] == 1 else " tapahtumaa")
34        string += package_code[0] + ", " + str(count[0]) + word_form ↩
             + "\n"
35     print(string)
36
37  ### Luo pickle-tiedosto listasta (tehokkuustestiä varten).
38  def build_pickle(alist, file_name):
39     with open(file_name, "wb") as f:
40        pickle.dump(alist, f)
41
42  ### Avaa pickle-tiedosto.
43  def open_pickle(file_name):
44     with open(file_name, "rb") as f:
45        alist = pickle.load(f)
46     return alist
47
48
```

```
49 class DB(object):
50     def __init__(self):
51         self.conn = None
52         self.cursor = None
53 
54     ### Avaa tietokanta tai luo uusi tietokanta.
55     def open(self, db_filename):
56         try:
57             conn = sqlite3.connect(db_filename)
58             self.conn = conn
59             self.cursor = self.conn.cursor()
60             self.conn.isolation_level = None
61         except sqlite3.Error:
62             return False
63         return self.conn, self.cursor
64 
65     ### Luo taulut tietokantaan.
66     def create_tables(self, printable=True):
67         self.cursor.execute("CREATE TABLE IF NOT EXISTS Places (id ↩
               INTEGER PRIMARY KEY, name TEXT)")
68         self.cursor.execute("CREATE TABLE IF NOT EXISTS Customers ↩
               (id INTEGER PRIMARY KEY, name TEXT)")
69         self.cursor.execute("CREATE TABLE IF NOT EXISTS Packages (id ↩
               INTEGER PRIMARY KEY, code TEXT, customer_id INTEGER)")
70         self.cursor.execute("CREATE TABLE IF NOT EXISTS Events (id ↩
               INTEGER PRIMARY KEY, package_id INTEGER, place_id ↩
               INTEGER, description TEXT, time TEXT)")
71         if printable:
72             print("Tietokanta luotu")
73 
74     ### Luo indeksit.
75     def create_indexes(self):
76         self.cursor.execute("CREATE INDEX id_customer ON Packages ↩
               (customer_id)")
77         self.cursor.execute("CREATE INDEX id_package ON Events ↩
               (package_id)")
78         self.cursor.execute("CREATE INDEX id_place ON Events ↩
               (place_id)")
79 
80     ### Tehokkuustestin vaiheet 1-4.
81     def insert_all(self, places, customers, packages, events, ↩
```

```
            start_time):
82          self.cursor.execute("BEGIN")
83          self.cursor.executemany("INSERT INTO Places VALUES(?,?)", ↩
               places)
84          new_time = get_time(start_time, "1")
85          self.cursor.executemany("INSERT INTO Customers VALUES(?,?)", ↩
               customers)
86          new_time = get_time(new_time, "2")
87          self.cursor.executemany("INSERT INTO Packages ↩
               VALUES(?,?,?)", packages)
88          new_time = get_time(new_time, "3")
89          self.cursor.executemany("INSERT INTO Events ↩
               VALUES(?,?,?,?,?)", events)
90          get_time(new_time, "4", return_time=False)
91          self.cursor.execute("COMMIT")
92
93      ### Tehokkuustestin vaiheet 5-6.
94      def select_count(self, count, table, phase, start_time):
95          sql = (
96              "SELECT COUNT(id) FROM Packages WHERE customer_id=?" if ↩
                   phase == 5 else
97              "SELECT COUNT(id) FROM Events WHERE package_id=?"
98          )
99          i = 1
100         while (i <= count):
101             self.cursor.execute(sql, [random.choice(table)[0]])
102             i += 1
103         get_time(start_time, phase, return_time=False)
104
105
106 class Place(object):
107     def __init__(self, cursor, name, idx=None):
108         self.cursor = cursor
109         self.name = name
110         self.idx = idx
111
112     # Hae paikan id tietokannasta.
113     def get_idx(self):
114         self.cursor.execute("SELECT id FROM Places WHERE name=?", ↩
               [self.name])
115         self.idx = self.cursor.fetchone()
```

```
116         return self.idx
117 
118     # Lisää paikka tietokantaan, jos ei ole siellä ennestään.
119     def add(self):
120         if self.idx is None:
121             self.cursor.execute("INSERT INTO Places (name) VALUES ↩
                    (?)", [self.name])
122             print("Paikka lisätty")
123         else:
124             print("VIRHE: Paikka on jo olemassa")
125 
126     # Hae paikan tapahtumien määrä tiettynä päivänä.
127     def get_events_count(self, date):
128         self.cursor.execute("SELECT COUNT(*) FROM Events LEFT JOIN ↩
                Places ON Events.place_id = Places.id WHERE Events.time ↩
                LIKE ? AND Places.id=?", ['%'+date+'%', self.idx[0]])
129         return self.cursor.fetchone()
130 
131 
132 class Customer(object):
133     def __init__(self, cursor, name, idx=None):
134         self.cursor = cursor
135         self.name = name
136         self.idx = idx
137 
138     # Hae asiakkaan id tietokannasta.
139     def get_idx(self):
140         self.cursor.execute("SELECT id FROM Customers WHERE name=?", ↩
                [self.name])
141         self.idx = self.cursor.fetchone()
142         return self.idx
143 
144     # Lisää asiakas tietokantaan, jos ei ole siellä ennestään.
145     def add(self):
146         if self.idx is None:
147             self.cursor.execute("INSERT INTO Customers (name) VALUES ↩
                    (?)", [self.name])
148             print("Asiakas lisätty")
149         else:
150             print("VIRHE: Asiakas on jo olemassa")
151 
```

```
152     # Hae asiakkaan paketit.
153     def get_packages(self):
154         self.cursor.execute("SELECT Packages.code FROM Customers ↩
                JOIN Packages ON Customers.id = Packages.customer_id ↩
                WHERE Packages.customer_id=?", [self.idx[0]])
155         return self.cursor.fetchall()
156
157
158 class Package(object):
159     def __init__(self, cursor, code, customer_idx=None, idx=None):
160         self.cursor = cursor
161         self.code = code
162         self.customer_idx = customer_idx
163         self.idx = idx
164
165     # Hae paketin id tietokannasta.
166     def get_idx(self):
167         self.cursor.execute("SELECT id FROM Packages WHERE code=?", ↩
                [self.code])
168         self.idx = self.cursor.fetchone()
169         return self.idx
170
171     # Lisää paketti tietokantaan, jos asiakas löytyy.
172     def add(self):
173         if self.customer_idx is not None:
174             self.cursor.execute("INSERT INTO Packages (code, ↩
                    customer_id) VALUES (?,?)", [self.code, ↩
                    self.customer_idx[0]])
175             print("Paketti lisätty")
176         else:
177             print("VIRHE: Asiakasta ei ole olemassa")
178
179     # Hae paketin tapahtumat.
180     def get_events(self):
181         self.cursor.execute("SELECT Events.time, Places.name, ↩
                Events.description FROM Events JOIN Places ON ↩
                Events.place_id = Places.id WHERE Events.package_id=?", ↩
                [self.idx[0]])
182         return self.cursor.fetchall()
183
184     # Hae paketin tapahtumien määrä.
```

```
185     def get_events_count(self):
186         self.cursor.execute("SELECT COUNT(Packages.id) FROM Events ↩
               JOIN Packages ON Events.package_id = Packages.id WHERE ↩
               Events.package_id=?", [self.idx[0]])
187         return self.cursor.fetchone()
188 
189 
190 class Event(object):
191     def __init__(self, cursor, package_idx, place_idx, description, ↩
           idx=None):
192         self.cursor = cursor
193         self.package_idx = package_idx
194         self.place_idx = place_idx
195         self.description = description
196         self.idx = idx
197         self.time = datetime.datetime.now().strftime("%d.%m.%Y %H:%M")
198 
199     # Lisää tapahtuma tietokantaan.
200     def add(self):
201         self.cursor.execute("INSERT INTO Events (package_id, ↩
               place_id, description, time) VALUES (?,?,?,?)", ↩
               [self.package_idx[0], self.place_idx[0], ↩
               self.description, self.time])
202         print("Tapahtuma lisätty")
203 
204 
205 if __name__ == "__main__":
206     print("""
207     Komennot:
208     0. Poistu.
209     1. Luo tietokanta.
210     2. Lisää uusi paikka.
211     3. Lisää uusi asiakas.
212     4. Lisää uusi paketti.
213     5. Lisää uusi tapahtuma.
214     6. Hae kaikki paketin tapahtumat seurantakoodin perusteella.
215     7. Hae kaikki asiakkaan paketit ja niihin liittyvien ↩
           tapahtumien määrä.
216     8. Hae annetusta paikasta tapahtumien määrä tiettynä päivänä.
217     9. Suorita tietokannan tehokkuustesti
218     """)
```

```
219     db_filename = "tietokanta.db"
220     test_db_filename = "testitietokanta.db"
221     db = DB()
222     test_db = DB()
223     command = ""
224 
225     while command != "0":
226         print("-----------------------\nValitse toiminto (0-9):")
227         command = input()
228 
229         # Luo tietokanta.
230         if command == "1":
231             conn, c = db.open(db_filename)
232             db.create_tables()
233 
234         # Tehokkuustesti.
235         elif command == "9":
236             # Valmistele listat tehokkuustestiä varten (tätä osaa ei ↩
                    lasketa mukaan suoritusaikaan).
237             print("Luodaan listoja...")
238             if os.path.exists("places.pkl"):
239                 places = open_pickle("places.pkl")
240                 customers = open_pickle("customers.pkl")
241                 packages = open_pickle("packages.pkl")
242                 events = open_pickle("events.pkl")
243             else:
244                 places = [(i, "P" + str(i)) for i in range(1, 1001)]
245                 customers = [(i, "A" + str(i)) for i in range(1, 1001)]
246                 packages = [[i, "K" + str(i), ↩
                        random.choice(customers)[0]] for i in range(1, 1001)]
247                 events = [[i, random.choice(packages)[0], ↩
                        random.choice(places)[0], "kuvaus", ↩
                        datetime.datetime.now().strftime("%d.%m.%Y %H:%M")] ↩
                        for i in range(1, 1000001)]
248                 build_pickle(places, "places.pkl")
249                 build_pickle(customers, "customers.pkl")
250                 build_pickle(packages, "packages.pkl")
251                 build_pickle(events, "events.pkl")
252 
253             # Aloita varsinainen testi.
254             print("Suoritetaan tehokkuustestiä...")
```

```
255
256             # Luo erillinen tietokanta tehokkuustestiä varten.
257             test_conn, test_c = test_db.open("testitietokanta.db")
258             test_db.create_tables(printable=False)
259
260             # Vaiheet 1-4:
261             t0 = time.perf_counter()
262             test_db.insert_all(places, customers, packages, events, t0)
263             test_db.create_indexes()
264
265             # Vaiheet 5-6:
266             t1 = time.perf_counter()
267             test_db.select_count(1000, customers, "5", t1)
268             t2 = time.perf_counter()
269             test_db.select_count(1000, packages, "6", t2)
270
271             # Sulje yhteys ja poista tietokanta (tiedosto).
272             test_conn.close()
273             os.remove(test_db_filename)
274
275         # Muut komennot.
276         else:
277             # Tarkista ensin, että tietokanta löytyy.
278             if database_exists(db_filename):
279                 conn, c = db.open(db_filename)
280
281                 if command == "2":
282                     print("Anna paikan nimi:")
283                     name = input()
284                     place = Place(c, name)
285                     place.get_idx()
286                     place.add()
287
288                 elif command == "3":
289                     print("Anna asiakkaan nimi:")
290                     name = input()
291                     customer = Customer(c, name)
292                     customer.get_idx()
293                     customer.add()
294
295                 elif command == "4":
```

```
296                 print("Anna paketin seurantakoodi:")
297                 code = input()
298                 package = Package(c, code)
299                 package_idx = package.get_idx()
300                 # Tarkista, ettei seurantakoodilla ole jo olemassa ↩
                       pakettia.
301                 if package_idx is not None:
302                     print("VIRHE: Seurantakoodilla on jo paketti")
303                 else:
304                     print("Anna asiakkaan nimi:")
305                     name = input()
306                     customer = Customer(c, name)
307                     customer_idx = customer.get_idx()
308                     package = Package(c, code, customer_idx)
309                     package.add()
310 
311             elif command == "5":
312                 print("Anna paketin seurantakoodi:")
313                 code = input()
314                 package = Package(c, code)
315                 package_idx = package.get_idx()
316                 # Tarkista, että seurantakoodi löytyy.
317                 if package_idx is None:
318                     print("VIRHE: Pakettia ei ole olemassa")
319                 else:
320                     print("Anna tapahtuman paikka:")
321                     name = input()
322                     place = Place(c, name)
323                     place_idx = place.get_idx()
324                     # Tarkista, että paikka löytyy.
325                     if place_idx is None:
326                         print("VIRHE: Paikkaa ei ole olemassa")
327                     else:
328                         print("Anna tapahtuman kuvaus:")
329                         descr = input()
330                         event = Event(c, package_idx, place_idx, descr)
331                         event.add()
332 
333             elif command == "6":
334                 print("Anna paketin seurantakoodi:")
335                 code = input()
```

```
336                 package = Package(c, code)
337                 package_idx = package.get_idx()
338                 # Tarkista, että seurantakoodi löytyy.
339                 if package_idx is None:
340                     print("VIRHE: Pakettia ei ole olemassa")
341                 else:
342                     events = package.get_events()
343                     pprint_events(events)
344 
345             elif command == "7":
346                 print("Anna asiakkaan nimi:")
347                 name = input()
348                 customer = Customer(c, name)
349                 customer_idx = customer.get_idx()
350                 # Tarkista, että asiakas löytyy.
351                 if customer_idx is None:
352                     print("VIRHE: Asiakasta ei ole olemassa")
353                 else:
354                     packages = customer.get_packages()
355                     event_counts = []
356                     for p in packages:
357                         package = Package(c, p[0], customer_idx)
358                         package.get_idx()
359                         count = package.get_events_count()
360                         event_counts.append(count)
361                     pprint_events_counts(packages, event_counts)
362 
363             elif command == "8":
364                 print("Anna paikan nimi:")
365                 name = input()
366                 place = Place(c, name)
367                 place_idx = place.get_idx()
368                 # Tarkista, että paikka löytyy.
369                 if place_idx is None:
370                     print("VIRHE: Paikkaa ei ole olemassa")
371                 else:
372                     print("Anna päivämäärä (dd.mm.yyyy):")
373                     date = input()
374                     count = place.get_events_count(date)
375                     print("Tapahtumien määrä:", count[0])
```