SQLite-programmering i Python

1 Introduksjon

Bindeleddet mellom Python og SQLite er sqlite3, en modul som baserer seg på Pythons offisielle database-API-spesifikasjon (PEP 249). Fordelen med sistnevnte er at man ved å lære seg å bruke sqlite3-modulen enkelt vil kunne benytte seg av andre databaser fra Python senere, eksempelvis PostgreSQL (via psycopg3) og MySQL (via Connector/Python), ettersom mange av disse implementerer samme API, hvilket gjør bruken tilnærmet lik.

2 Tilkobling

For at Python skal kunne prate med en gitt SQLite-database, må vi først av alt opprette en tilkobling til databasen. Det er så enkelt som å importere sqlite3 og opprette et tilkoblingsobjekt ved hjelp av **connect()**-metoden, hvor vi spesifiserer filnavnet eller filstien til databasen:

```
import sqlite3
con = sqlite3.connect("test.db")
```

Vi kan deretter hente ut et markørobjekt fra tilkoblingsobjektet ved hjelp av **cursor()**-metoden. Markørobjektet kan så brukes til å utføre spørringer mot databasen og hente ut resultater:

```
cursor = con.cursor()
cursor.execute("SELECT * FROM sqlite_master")
```

Når du er fornøyd med det du har utrettet i databasen, er det en god idé å lukke tilkoblingen:

```
con.close()
```

3 Utføre spørringer

Med et aktivt markørobjekt tilgjengelig er det en enkel sak å utføre spørringer mot databasen. Dette gjøres med **execute()**-metoden. Ofte vil man ønske å bruke Python-variabler (f.eks. konstruert fra brukerinput) i spørringene. Her må man dog holde tunga rett i munnen. **Ikke** bruk strenginterpolasjon for å oppnå dette, da det dette åpner for SQL-injeksjonsangrep. (Denne xkcd-stripa er et godt eksempel på problemet.) For å illustrere:

```
# Ikke gjør dette hjemme!
cursor.execute("SELECT * FROM person WHERE navn = '%s'" % navn)
# Gjør heller:
cursor.execute("SELECT * FROM person WHERE navn = ?", (navn))
# Eller, om du vil være veldig ryddig:
cursor.execute("SELECT * FROM person WHERE navn =:navn", {"navn" = navn})
```

For å hente ut resultatet har man noen alternativer:

1. **fetchone()**: hent ut neste resultat i resultatsettet, hvis det finnes. Returnerer en tuppel eller **None** dersom det ikke er flere tupler igjen.

   ```
   cursor.execute("SELECT * FROM person")
   row = cursor.fetchone()
   print("First row from table person:")
   print(row)
   ```

2. **fetchall()**: hent ut hele resultatsettet (eventuelt de gjenværende elementene i settet dersom fetchone() er blitt kalt). Returnerer en liste av tupler eller en tom liste dersom ingen tupler i databasen matcher spørringen.

   ```
   cursor.execute("SELECT * FROM person")
   rows = cursor.fetchall()
   print("All rows in the table person:")
   print(rows)
   ```

3. **fetchmany(i)**: hent ut en liste av **i** tupler fra resultatsettet. Om det er færre rader tilgjengelig, returnerer den bare disse, eventuelt en tom liste hvis det ikke er rader igjen.

   ```
   cursor.execute("SELECT * FROM person")
   rows = cursor.fetchmany(2)
   print("First two rows from table person:")
   print(rows)
   ```

4. Bruk markørobjektet som en iterator, eksempelvis i en for-løkke.

```
for row in cursor.execute("SELECT * FROM person"):
    print(row)
```

4 Opprettelse av tabeller og innsetting av data

Å lage tabeller og sette inn data gjøres på liknende vis, igjen ved hjelp av **execute()**:

```
cursor.execute('''CREATE TABLE person
                  (id INTEGER PRIMARY KEY, name TEXT, birthday TEXT)''')
cursor.execute('''INSERT INTO person VALUES (1, 'Ola Nordmann', '2002-02-02')''')
connection.commit()
connection.close()
```

Legg merke til at vi også kaller **commit()**-metoden til tilkoblingsobjektet for å fullføre transaksjonen.

Lurer du på noe? Python-dokumentasjonen inkluderer en god side om sqlite3-modulen. Især kan kapittelet «Using sqlite3 efficiently» være av interesse. Ellers kan du som alltid spørre på Piazza. Om du lurer på noe, er sjansen stor for at noen andre også gjør det.