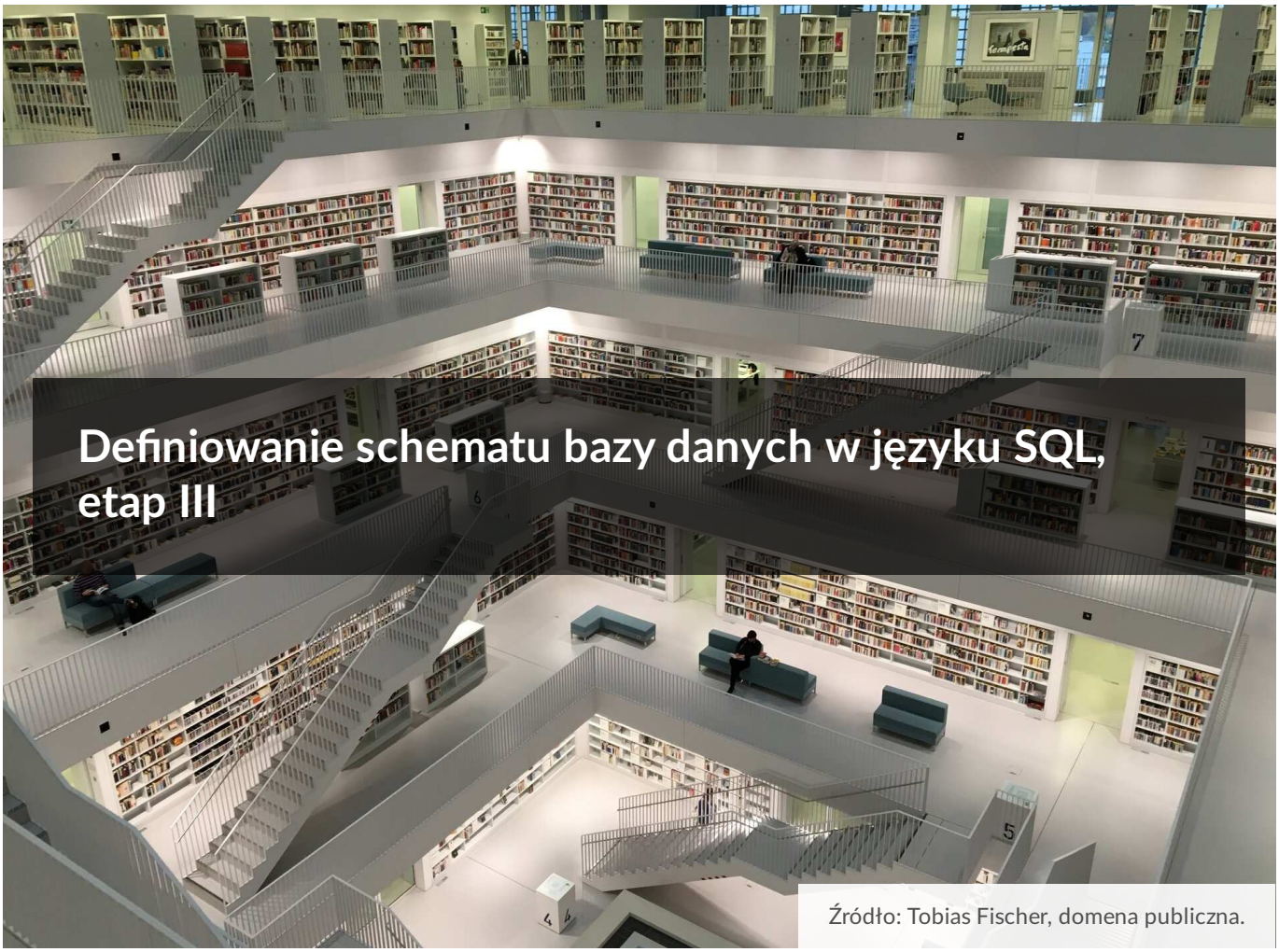
Definiowanie schematu bazy danych w języku SQL, etap III



Definiowanie schematu bazy danych w języku SQL, etap III

Źródło: Tobias Fischer, domena publiczna.

Często podczas pracy z danymi tworzymy – za pomocą różnych narzędzi – bazy, w bazach zaś tabele, które wypełniamy informacjami. Podstawowymi narzędziami są programy działające w wierszu poleceń, zapewniane przez twórców systemów bazodanowych.
W wielu zadaniach, zwłaszcza wymagających automatyzacji, warto wykorzystać skrypty napisane w wybranym języku programowania. W tym materiale pokażemy, w jaki sposób możemy zarządzać bazą danych SQLite3 z wykorzystaniem języka Python.

- Definiowanie schematu bazy danych w języku SQL, etap I,
- Definiowanie schematu bazy danych w języku SQL, etap II,
- Definiowanie schematu bazy danych w języku SQL, etap IV.

Twoje cele

- Utworzysz bazę danych za pomocą skryptu Pythona.
- Napiszesz skrypt w języku Python odczytujący i wykonujący zapytania SQL, zapisane w pliku.
- Zaimplementujesz w języku Python funkcję dodającą do bazy dane z plików CSV.

Przeczytaj

Schemat bazy danych

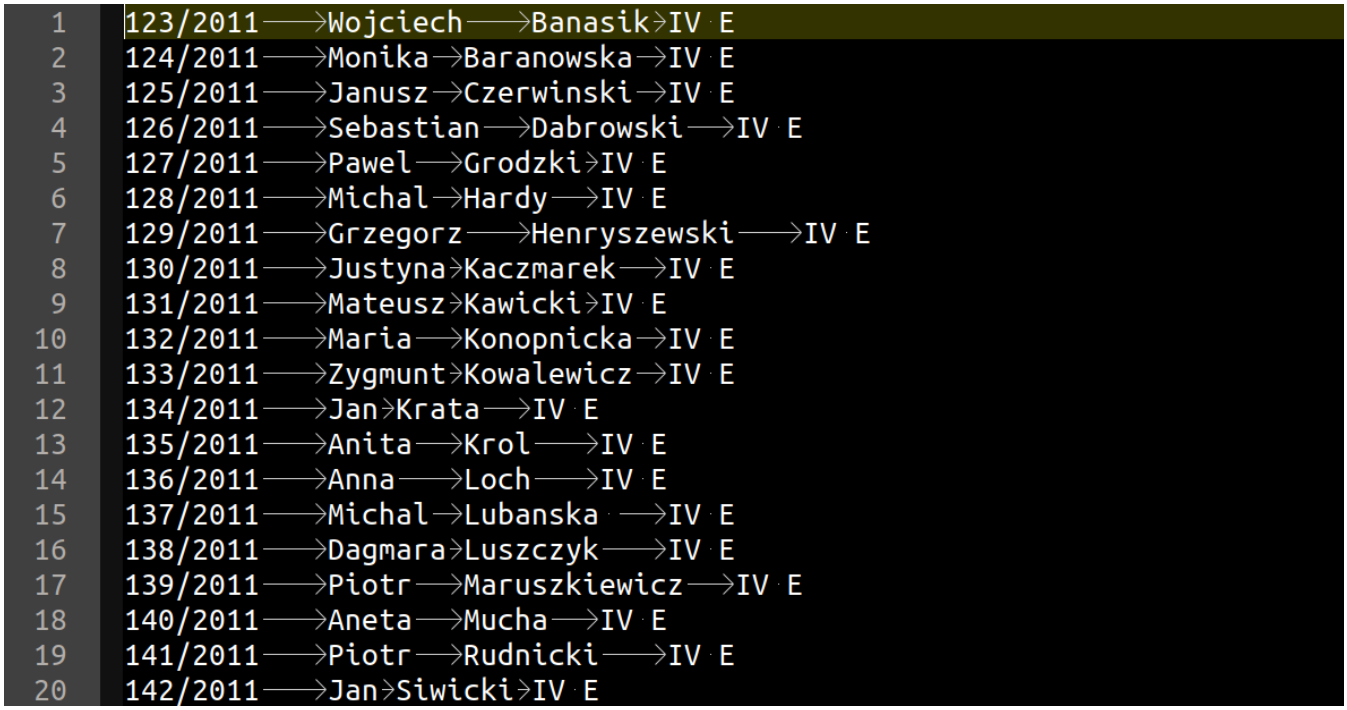
Na początku pobieramy pliki w formacie CSV z danymi i zapisujemy w katalogu `uczniowie_baza`.

Plik `uczniowie.csv`
Plik o rozmiarze 22.47 KB w języku polskim

Plik `przedmioty.csv`
Plik o rozmiarze 729.00 B w języku polskim

Plik `oceny.csv`
Plik o rozmiarze 385.12 KB w języku polskim

Pliki otwieramy w edytorze tekstu, który ma możliwość pokazywania białych znaków – w ten sposób łatwiej nam będzie ustalić, jakim separatorem oddzielone są wartości pól w wierszach.



Źródło: Contentplus.pl Sp. z o.o., tylko do użytku edukacyjnego.

Na zrzucie widać, że w pliku `uczniowie.csv` separatorem jest znak tabulacji, a dane składają się z identyfikatora ucznia, imienia i nazwiska oraz klasy.

Instrukcja SQL, która utworzy nam odpowiednią tabelę, może wyglądać następująco:

Przykład 1

```
DROP TABLE IF EXISTS uczniowie;
CREATE TABLE uczniowie (
```

```
    id CHAR(8) PRIMARY KEY,
    imie VARCHAR(20) NOT NULL CHECK(imie <> ''),
    nazwisko VARCHAR(30) NOT NULL CHECK(nazwisko <> ''),
    klasa CHAR(5) DEFAULT ''
);
```

Na uwagę zasługuję fakt, że identyfikator ucznia, czyli klucz główny tabeli, nie jest liczbą całkowitą, ale 8-znakowym ciągiem znaków. Ograniczenia nałożone na pola `imie` i `nazwisko` nie dopuszczają braku wartości oraz pustych ciągów znaków. Dla pola zdefiniowano wartość domyślną – pusty ciąg znaków.

> Klauzula `DROP TABLE IF EXISTS` usuwa tabelę o podanej nazwie w bazie, jeżeli wcześniej już istniała. Używamy jej, aby można było wielokrotnie, bez przeszkód, testować skrypty.

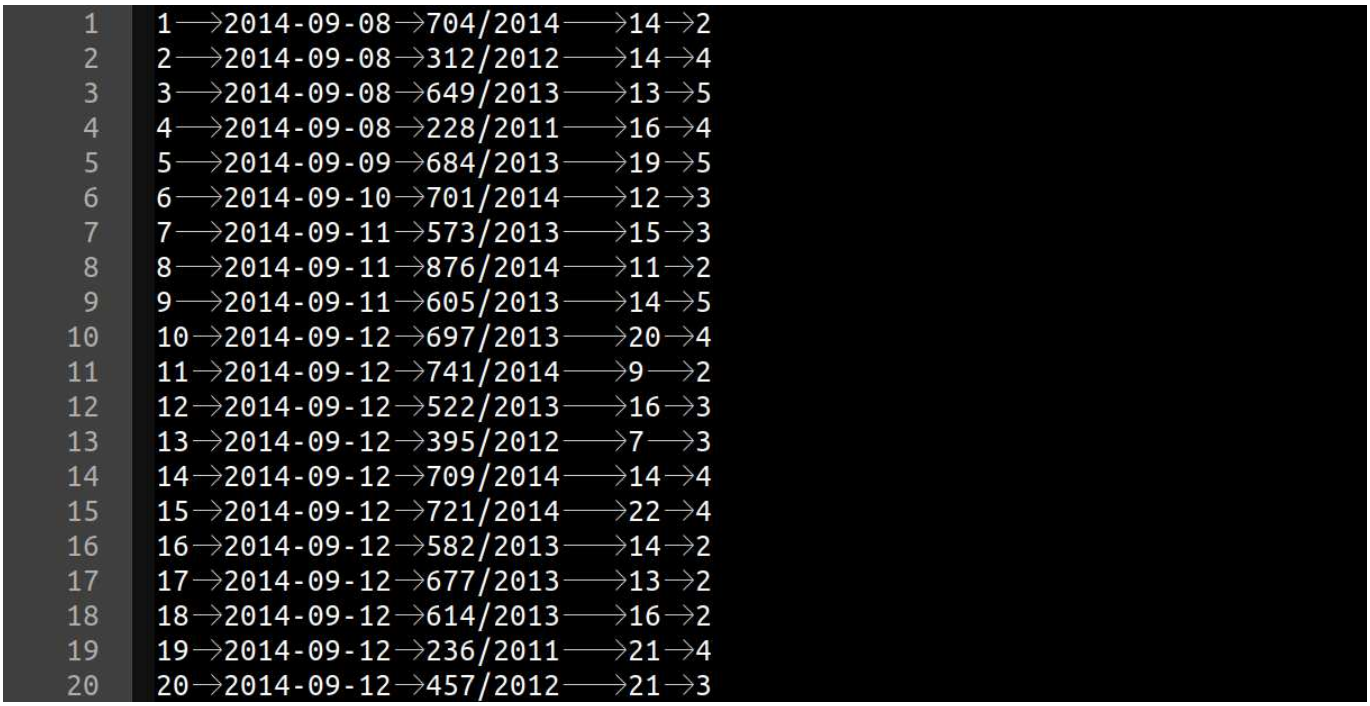
```
1,administracja bazami danych
2,administracja sieciowymi systemami operacyjnymi
3,biologia
4,chemia
5,diagnostyka i naprawa urzadzen techniki komputerowej
6,edukacja dla bezpieczenstwa
7,fizyka
8,geografia
9,historia
10,historia i spoleczenstwo - przedmiot uzupelniajacy
11,informatyka
12,j.angielski
13,j.niemiecki
14,j.polski
15,jezyk angielski zawodowy w branzy informatycznej
16,matematyka
17,podstawy przedsiebiorczosci
18,programowanie aplikacji internetowych
19,projektowanie i montaz lokalnych sieci komputerowych
20,sieci komputerowe
```

Źródło: Contentplus.pl Sp. z o.o., tylko do użytku edukacyjnego.

Separatorem w pliku `przedmioty.csv` jest przecinek, wiersze zawierają tylko dwie informacje: identyfikator oraz nazwę przedmiotu. Dla tych danych utworzymy prostą tabelę słownikową, za pomocą polecenia:

> **Przykład 2**

```
DROP TABLE IF EXISTS przedmioty;
CREATE TABLE przedmioty (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nazwa VARCHAR(60) NOT NULL CHECK(nazwa <> '')
);
```



Źródło: Contentplus.pl Sp. z o.o., tylko do użytku edukacyjnego.

W pliku `oceny.csv` separatorem danych ponownie jest znak tabulacji, plik zawiera następujące informacje: identyfikator oceny, datę otrzymania, identyfikator ucznia, identyfikator przedmiotu oraz ocenę. Z tego wynika, że odpowiednia klauzula SQL powinna utworzyć nie tylko pola do przechowywania danych, ale również relacje łączące tabelę ocen z tabelami `uczniowie` i `przedmioty`.

Przykład 3

```sql
DROP TABLE IF EXISTS oceny;
CREATE TABLE oceny (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    data DATE NOT NULL,
    id_ucznia CHAR(8) NOT NULL,
    id_przedm INTEGER,
    ocena DECIMAL(3,2) NOT NULL CHECK(ocena < 7),
    FOREIGN KEY (id_ucznia) REFERENCES uczniowie(id)
    ON DELETE CASCADE,
    FOREIGN KEY (id_przedm) REFERENCES przedmioty(id)
    ON DELETE SET NULL
);
```

Definiując pola, w których przechowywane będą wartości z innych tabel, to znaczy `id_ucznia` oraz `id_przedm`, pamiętać musimy, aby miały takie same typy danych, jak odpowiadające im pola w tabelach źródłowych.

Do tworzenia relacji używamy klauzul `FOREIGN KEY...REFERENCES...` umieszczonych na końcu definicji pól. W nawiasach okrągłych podajemy nazwę klucza obcego, a następnie

nazwę tabeli źródłowej i klucza głównego w nawiasach okrągłych.

Na koniec wszystkie podane klauzule zapisujemy w pliku `uczniowie.sql`, w katalogu o nazwie `uczniowie_baza`.

Tworzenie bazy i tabel w języku Python

Język Python ułatwia obsługę baz danych dzięki standardowym modułom, takim jak np. `sqlite3` czy `mysql`. Zacznijmy od kodu, który umożliwi nawiązanie połączenia z bazą oraz wykonywanie na niej operacji. W pliku `uczniowie.py`, który umieszczamy w katalogu `uczniowie_baza`, wpisujemy początkowy kod:

Przykład 4

```
import sqlite3, os


def main(args):
    baza = 'uczniowie.db'
    con = sqlite3.connect(baza)  # połączenie z bazą
    cur = con.cursor()  # utworzenie obiektu kursora

    utworz_tabele(con, 'uczniowie.sql')  # tworzenie tabel
    zbadaj_baze(cur)  # sprawdzenie schematu bazy

    con.commit()  # zatwierdzenie zmian w bazie
    con.close()  # zamknięcie połączenia z bazą
    return 0


if __name__ == '__main__':
    import sys
    sys.exit(main(sys.argv))
```

Baza danych SQLite3 zawarta jest w jednym pliku o nazwie `uczniowie.db`. Metoda `connect()` łączy się z bazą i zwraca obiekt połączenia. Ma on następujące metody:

- `cursor()` – tworzy obiekt kursora, który pozwala na wykonywanie zapytań i odczytywanie ich wyników,
- `commit()` – zatwierdza zmiany w bazie,
- `close()` – zamyka połączenie z bazą.

W funkcji głównej umieściliśmy wywołanie funkcji `utworz_tabele()` oraz `zbadaj_baze()`, których kod (przedstawiony w **Przykładzie 5**) umieszczamy przed funkcją główną.

> **Przykład 5**

```
def zbadaj_baze(cur):
    cur.execute("SELECT * FROM sqlite_master;")
    for t in cur.fetchall():
        print(t[4])


def utworz_tabele(cur, plik_sql):
    if os.path.isfile(plik_sql):
        with open(plik_sql, newline='', encoding='utf-8') as sq
            cur.executescript(sql.read())
    else:
        print("Brak pliku SQL na dysku!")
```

W funkcji `utworz_tabele()` sprawdzamy, czy podany plik zawierający instrukcje SQL istnieje na dysku i jeżeli tak, otwieramy go przy użyciu instrukcji `with open() as`, dzięki czemu zostanie on automatycznie zamknięty. Treść pliku odczytujemy za pomocą metody `read()`, która zwraca ciąg znaków, w tym wypadku polecenia SQL. Ostatecznie przekazujemy je do wykonania metodzie `executescript()`.

Zadaniem funkcji `zbadaj_baze()` jest odczytanie wewnętrznej tabeli bazy SQLite3 o nazwie `sqlite_master`, która przechowuje schemat bazy. Jeżeli po wykonaniu skryptu zobaczymy klauzule SQL, które posłużyły do utworzenia tabel, będzie to znaczyło, że baza utworzona została poprawnie.

> **Ćwiczenie 1**
>
> Przetestuj działanie przygotowanego skryptu, uruchamiając go w wierszu poleceń lub z poziomu wybranego edytora. Skrypt możesz uruchamiać wielokrotnie, aby wyeliminować ewentualne błędy.

Dodawanie danych w języku Python

Do wprowadzenia danych wykorzystamy podane na wstępie pliki w formacie CSV oraz drugi skrypt języka Python. W katalogu z poprzednimi plikami zapisujemy plik o nazwie `uczniowie_dane.py`, którego początkowa zawartość jest identyczna jak w poprzednim skrypcie, poza dodatkowym importem oraz wywoływaną funkcją.

> **Przykład 6**

```
import sqlite3
import os
import csv


def dodaj_dane(cur, tabela, sep):
    plik_csv = tabela + '.csv'
    if not os.path.isfile(plik_csv):
        print(f"Brak pliku {plik_csv} na dysku!")
        return False

    dane = []  # lista rekordów
    with open(plik_csv, newline='', encoding='utf-8') as plik::
        for wiersz in csv.reader(plik, delimiter=sep):
            dane.append(wiersz)
    for rekord in dane:
        print(rekord)


def main(args):
    baza = 'uczniowie.db'
    con = sqlite3.connect(baza)  # połączenie z bazą
    cur = con.cursor()  # utworzenie obiektu kursora

    dodaj_dane(cur, 'uczniowie', '\t')  # dodawanie danych

    con.commit()  # zatwierdzenie zmian w bazie
    con.close()  # zamknięcie połączenia z bazą
    return 0


if __name__ == '__main__':
    import sys
    sys.exit(main(sys.argv))
```

Funkcja `dodaj_dane()` otrzymuje jako argumenty kursor, nazwę tabeli i separator danych. Na początku funkcji tworzymy nazwę pliku z danymi, dodając do nazwy tabeli rozszerzenie `.csv`. Jeżeli pliku nie ma na dysku, wypisujemy komunikat o jego braku, w przeciwnym razie otwieramy plik, używając wspominanej instrukcji `with open() as`.

Do obsługi plików CSV Python dysponuje dedykowanym modułem `csv`, który zaimportowaliśmy na początku pliku. Funkcja `reader()` rozbija na części każdy

przeczytany wiersz pliku, używając podanego separatora. Części zwraca jako listę wartości dla danego rekordu. Kolejne rekordy zapisujemy w liście `dane`. Na końcu w pętli drukujemy zapisane rekordy.

> **Ćwiczenie 2**
>
> Uruchom skrypt, zwróć uwagę na wypisane listy wartości.

Obiekt kursora oferuje kilka metod pozwalających dodawać dane do tabel. Zacznijmy od najprostszej wersji:

> **Przykład 7**

```
1 cur.execute("INSERT INTO uczniowie VALUES ('123/2011', 'Wojciec
```

Metoda `execute()` pozwala wykonać jedno zapytanie SQL na raz. W przedstawionej wersji wartości pól zostały podane bezpośrednio jako ciągi znaków. W praktyce ze względów bezpieczeństwa częściej wykorzystuje się wersję zawierającą tak zwane zastępniki (ang. *placeholders*), czyli umowne znaki, które zostają zastąpione przez wartości podane w tupli lub liście. Prześledźmy przykłady.

> **Przykład 8**
>
> Dodanie jednego rekordu do tabeli z wykorzystaniem zapytania z zastępnikami:

```
1 cur.execute("INSERT INTO uczniowie VALUES (?, ?, ?, ?)", ('123/
```

> **Przykład 9**
>
> Dodanie wielu rekordów do tabeli w pętli z wykorzystaniem zapytania z zastępnikami:

```
1 for rekord in dane:
2     cur.execute("INSERT INTO uczniowie VALUES (?, ?, ?, ?)", re
```

W naszym przypadku, kiedy chcemy dodać do tabel wiele rekordów na raz, skorzystamy z metody `executemany()`, która wymaga wspomnianych zastępników w zapytaniu SQL oraz listy (lub tupli) zawierającej wartości rekordów:

> **Przykład 10**

```
1 dane = [
2     ['123/2011', 'Wojciech', 'Banasik', 'IV E'],
3     ['124/2011', 'Monika', 'Baranowska', 'IV E'],
4 ]
```

```
5 cur.executemany("INSERT INTO uczniowie VALUES (?, ?, ?, ?)", da
```

Ćwiczenie 3

Omówioną metodę wykorzystamy w naszym skrypcie. Do funkcji `dodaj_dane()` dopisz podany kod. Ze względu na czytelność wyników działania skryptu, w funkcji `dodaj_dane()` wstaw znaki komentarza przed pętlą `for` i przed instrukcją wypisującą odczytane rekordy.

```
    zastepniki = ','.join(['?'] * len(dane[0]))
    zapytanie = 'INSERT INTO ' + tabela + ' VALUES(' + zastepni
    cur.executemany(zapytanie, dane)
    return cur.rowcount
```

Instrukcja `zastepniki = ','.join(['?'] * len(dane[0]))` tworzy ciąg znakowy zastępników. Wyrażenie `['?'] * len(dane[0])` zwraca listę zawierającą tyle znaków zapytania, ile wartości ma pierwszy dodawany rekord. Metoda `join()` zwraca elementy listy połączone przecinkiem, np. `'?,?,?,?'`, jeżeli dodawany rekord zawiera 4 pola (wartości).

Ostatnia instrukcja `return cur.rowcount` zwraca liczbę zmodyfikowanych rekordów, co możemy wykorzystać jako potwierdzenie, że operacja się udała.

Ćwiczenie 4

W funkcji głównej zmienimy i wstawimy wywołania funkcji `dodaj_dane()`:

```
    cur.execute('PRAGMA FOREIGN_KEYS = ON')
    print(dodaj_dane(cur, 'uczniowie', '\t'))  # dodawanie dany
    print(dodaj_dane(cur, 'przedmioty', ','))  # dodawanie dany
    print(dodaj_dane(cur, 'oceny', '\t'))  # dodawanie danych
```

W przedstawionym kodzie warto zauważyć, że aby poprawnie dodać dane do tabel, musimy pamiętać o podawaniu odpowiedniego separatora danych oraz o kolejności dodawania danych – przy założeniu, że baza sprawdza zdefiniowane ograniczenia.

W bazach SQLite3 ograniczenia `FOREIGN KEY` są domyślnie wyłączone, włączamy je więc, wykonując zapytanie `PRAGMA FOREIGN_KEYS = ON`. Tabela `oceny` zawiera klucze obce, czyli pola `id_ucznia` i `id_przedm`, w których wartości muszą odpowiadać wartościom z tabel i pól powiązanych relacjami, a więc dane powinny zostać dodane najpierw do tabel `uczniowie` i `przedmioty`.

> **Ćwiczenie 5**
>
> Uruchom skrypt `uczniowie_dane.py`.

Słownik

> **format CSV**
>
> standard zapisywania informacji z bazy danych w pliku tekstowym, w którym pola każdego rekordu oddzielone separatorem (np. przecinkiem) zapisywane są w kolejnych wierszach
> **kursor**
>
> (ang. *cursor*) obiekt pozwalający komunikować się z bazą danych, czyli wykonywać zapytania SQL, a także umożliwiający pobieranie kolejnych rekordów, zwróconych przez zapytania
> **tabela słownikowa**
>
> tabela zawierająca listę wartości, które dopuszczalne są w polu innej tabeli powiązanej relacją

Prezentacja multimedialna

Polecenie 1

Zapoznaj się z prezentacją multimedialną. Stwórz relacyjną bazę danych, przeznaczoną dla dwóch użytkowników, z których każdy będzie miał inne uprawnienia. Wykorzystaj dane zawarte w załączonych plikach.

Plik `osoby.csv`
Plik o rozmiarze 11.42 KB w języku polskim

Plik `obiekty.csv`
Plik o rozmiarze 130.00 B w języku polskim

Plik `zajecia.csv`
Plik o rozmiarze 1.09 KB w języku polskim

Plik `wejscia.csv`
Plik o rozmiarze 43.09 KB w języku polskim

Polecenie 2

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Zaznacz wszystkie poprawne odpowiedzi. Dane, które chcemy zaimportować do tabel w bazie danych, mogą być zapisane:

- [ ] W innej bazie i tabeli.
- [ ] W plikach arkusza kalkulacyjnego.
- [ ] W plikach z rozszerzeniem `.sql`.
- [ ] W plikach w formacie CSV.

Ćwiczenie 2

Zaznacz poprawną odpowiedź. Plik w formacie CSV to:

- ( ) Plik z instrukcjami SQL.
- ( ) Plik sformatowany w specjalnym edytorze.
- ( ) Plik tekstowy zawierające wiersze, w których wartości oddzielone są znakiem separatora.
- ( ) Binarny obraz bazy danych.

Ćwiczenie 3

Zaznacz poprawną odpowiedź. Klauzula `DROP TABLE IF EXISTS`:

- ◯ Nie jest poprawną klauzulą SQL.
- ◯ Zapowiada zmianę struktury tabeli.
- ◯ Usuwa tabelę o podanej nazwie w bazie, jeżeli wcześniej istniała.
- ◯ Wykonuje kopię podanej tabeli.

Ćwiczenie 4

Uzupełnij poprawnie zdanie.

Kolejność importowania danych do tabel w bazie jest [ ].

uzależniona od uprawnień użytkownika

uzależniona od ograniczeń nałożonych przez relacje | dowolna

Ćwiczenie 5

Zaznacz wszystkie poprawne odpowiedzi. Za pomocą skryptów Pythona można:

- ☐ Tworzyć bazy danych i tabele.
- ☐ Importować dane z plików CSV.
- ☐ Odczytywać dane z bazy.
- ☐ Tylko tworzyć tabele.

Ćwiczenie 6

Zaznacz wszystkie poprawne odpowiedzi. Zdefiniowanie relacji za pomocą języka SQL jest możliwe:

- [ ] Przy użyciu klauzuli `CREATE REFERENCES (klucz_obcy) (klucz_podstawowy)`.
- [ ] Jeżeli łączone pola mają ten sam typ danych.
- [ ] Jeżeli łączone pola mają różne typy danych.
- [ ] Przy użyciu klauzuli `FOREIGN KEY (klucz_obcy) REFERENCES tabela (klucz_podstawowy)`.

Ćwiczenie 7

Zaznacz wszystkie poprawne odpowiedzi. Do wykonywania operacji na bazie danych z poziomu skryptu Pythona niezbędne jest:

- [ ] Posiadanie uprawnień użytkownika `root`.
- [ ] Utworzenie obiektu kursora.
- [ ] Zapisanie poleceń SQL w zewnętrznym pliku.
- [ ] Nawiązanie połączenia z bazą.

Ćwiczenie 8

Zaznacz wszystkie poprawne odpowiedzi. Wskaż metody z modułu `sqlite3`, które pozwalają wykonywać zapytania SQL:

- [ ] `run_sql()`
- [ ] `executemany()`
- [ ] `execute()`
- [ ] `executescript()`

Dla nauczyciela

Autor: Tomasz Jarosz
Przedmiot: Informatyka

Temat: Definiowanie schematu bazy danych w języku SQL, etap III

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.
>
> > Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:
> >
> > > 4) przygotowując opracowania rozwiązań złożonych problemów, posługuje się wybranymi aplikacjami w stopniu zaawansowanym:
> > >
> > > > d) projektuje i tworzy relacyjną bazę złożoną z wielu tabel oraz sieciową aplikację bazodanową dla danych związanych z rozwiązywanym problemem, formułuje kwerendy, tworzy i modyfikuje formularze oraz raporty, stosuje język SQL do wyszukiwania informacji w bazie i do jej modyfikacji, uwzględnia kwestie integralności danych, bezpieczeństwa i ochrony danych w bazie,

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Utworzysz bazę danych za pomocą skryptu Pythona.
- Napiszesz skrypt w języku Python odczytujący i wykonujący zapytania SQL, zapisane w pliku.
- Zaimplementujesz w języku Python funkcję dodającą do bazy dane z plików CSV.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. Chętny lub wybrany uczeń przygotowuje rozwiązanie ćwiczenia nr 1 z sekcji „Przeczytaj". Będzie pełnił rolę eksperta podczas zajęć.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Definiowanie schematu bazy danych w języku SQL, etap III". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Przedstawienie tematu i celów zajęć.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Jeżeli przygotowanie uczniów do lekcji jest niewystarczające, nauczyciel prosi o indywidualne zapoznanie się z treścią zawartą w sekcji „Przeczytaj". Każdy uczestnik zajęć podczas cichego czytania wynotowuje najważniejsze kwestie poruszane w tekście. Chętni lub wybrani uczniowie przedstawiają rozwiązania ćwiczeń z sekcji „Przeczytaj". Pozostali uczniowie weryfikują poprawność rozwiązań lub przedstawiają alternatywne sposoby.
2. **Praca z multimedium**. Uczniowie zapoznają się indywidualnie z treścią sekcji „Prezentacja multimedialna", zapisują problemy i pytania z nią związane. Następuje dyskusja, w trakcie której nauczyciel wyjaśnia niezrozumiałe kroki procedury.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenia nr 1-5. Po wykonaniu każdego z nich następuje omówienie rozwiązania przez nauczyciela.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.

Praca domowa:

1. Uczniowie wykonują ćwiczenia 6-8 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla systemu MySQL 8.0 (lub nowszej wersji).

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać multimedium w sekcji „Prezentacja multimedialna" do przygotowania się do lekcji powtórkowej.