

Raport z przedmiotu Techniki Internetowe i Bazy Danych

30.11.2024

Osoby w projekcie: Katarzyna Cichecka, Karolina Bialecka, Sebastian Bober

Wprowadzenie danych do bazy

Wprowadzono przygotowane dane do bazy za pomocą zapytań w MySQL na podstawie wcześniej przygotowanych tabel i relacji między nimi. Na przykład:

```
INSERT INTO rekiny (nazwa_naukowa, nazwa_potoczna, rodzina, dlugosc, waga, stan_zagrozenia, dieta_id)
VALUES ('Carcharodon carcharias', 'Żarłacz biały', 'Lamnidae', 6.0, 1100, 'Vulnerable', 1),
('Galeocerdo cuvier', 'Żarłacz tygrysi', 'Carcharhinidae', 5.5, 900, 'Near Threatened', 2),
('Rhincodon typus', 'Rekin wielorybi', 'Rhincodontidae', 12.0, 21000, 'Endangered', 3),
('Isurus oxyrinchus', 'Żarłacz mako', 'Lamnidae', 4.0, 600, 'Endangered', 4),
('Sphyrna mokarran', 'Rekin młot', 'Sphyrnidae', 4.3, 500, 'Critically Endangered', 5),
('Hexanchus griseus', 'Rekin sześcioczabowy', 'Hexanchidae', 5.0, 800, 'Least Concern', 6),
('Pristiophorus cirratus', 'Rekin piłonos', 'Pristiophoridae', 1.5, 18, 'Least Concern', 7),
('Cetorhinus maximus', 'Rekin baskijski', 'Cetorhinidae', 10.0, 19000, 'Vulnerable', 8),
('Somniosus microcephalus', 'Rekin grenlandzki', 'Somniosidae', 7.0, 1000, 'Near Threatened', 9),
('Alopias vulpinus', 'Rekin lis', 'Alopiidae', 6.1, 500, 'Vulnerable', 10);
```

Wstępna prezentacja

Dane są możliwe do wyświetlenia na dwa sposoby:

Bazowo w MySQL, gdzie za pomocą zapytań jesteśmy w stanie wyświetlić dane w dodanych tabelach:

SELECT * FROM rekiny;

	id	nazwa_naukowa	nazwa_potoczna	rodzina	dlugosc	waga	stan_zagrozenia	dieta_id
▶	1	Carcharodon carcharias	Żarłacz biały	Lamnidae	6	1100	Vulnerable	1
	2	Galeocerdo cuvier	Żarłacz tygrysi	Carcharhinidae	5.5	900	Near Threatened	2
	3	Rhincodon typus	Rekin wielorybi	Rhincodontidae	12	21000	Endangered	3
	4	Isurus oxyrinchus	Żarłacz mako	Lamnidae	4	600	Endangered	4
	5	Sphyrna mokarran	Rekin młot	Sphyrnidae	4.3	500	Critically Endangered	5
	6	Hexanchus griseus	Rekin sześcioczabowy	Hexanchidae	5	800	Least Concern	6
	7	Pristiophorus cirratus	Rekin piłonos	Pristiophoridae	1.5	18	Least Concern	7
	8	Cetorhinus maximus	Rekin baskijski	Cetorhinidae	10	19000	Vulnerable	8
	9	Somniosus microcephalus	Rekin grenlandzki	Somniosidae	7	1000	Near Threatened	9
*	10	Alopias vulpinus	Rekin lis	Alopiidae	6.1	500	Vulnerable	10
*		NULL	NULL	NULL	NULL	NULL	NULL	NULL

W prostej stronie stworzonej w technologii, która będzie wykorzystywana w dalszej części projektu, czyli w Flasku na pythonie:

rekiny

id	nazwa_naukowa	nazwa_potoczna	rodzina	dlugosc	waga	stan_zagrozenia	dieta_id
1	Carcharodon carcharias	Żarłacz biały	Lamnidae	6.0	1100.0	Vulnerable	1
2	Galeocerdo cuvier	Żarłacz tygrysi	Carcharhinidae	5.5	900.0	Near Threatened	2
3	Rhincodon typus	Rekin wielorybi	Rhincodontidae	12.0	21000.0	Endangered	3
4	Isurus oxyrinchus	Żarłacz mako	Lamnidae	4.0	600.0	Endangered	4
5	Sphyrna mokarran	Rekin młot	Sphyrnidae	4.3	500.0	Critically Endangered	5
6	Hexanchus griseus	Rekin sześciozabowy	Hexanchidae	5.0	800.0	Least Concern	6
7	Pristiophorus cirratus	Rekin piłonos	Pristiophoridae	1.5	18.0	Least Concern	7
8	Cetorhinus maximus	Rekin baskijski	Cetorhinidae	10.0	19000.0	Vulnerable	8
9	Somniosus microcephalus	Rekin grenlandzki	Somniosidae	7.0	1000.0	Near Threatened	9
10	Alopias vulpinus	Rekin lis	Alopiidae	6.1	500.0	Vulnerable	10

Wykonanie strony:

Po zapoznaniu się z dokumentacją biblioteki Flask stworzyliśmy prostą stronę, która wyświetla bardzo uproszczony HTML:

```
1  from flask import Flask  
2  
3  app = Flask(__name__)  
4  
5  @app.route("/")  
6  def hello_world():  
7      return "<p>rekinki</p>"  
8
```

Wynik na stronie:



rekinki

Następnie przeczytano o połączeniu z bazą MySQL w pythonie i napisano “connection string'a” umożliwiającego połączenie z bazą. Utworzono słownik z danymi koniecznymi do połączenia, takimi jak: nazwa użytkownika, hasło, host, baza danych, typ zapisu. Następnie utworzono połączenie za pomocą łącznika “pymysql”, do którego przekazano wcześniej utworzony słownik.

```
6  # Konfiguracja połączenia z bazą MySQL
7 db_config = {
8     'user': 'root',
9     'password': '2203',
10    'host': 'localhost',
11    'database': 'rekinki',
12    'charset': 'utf8mb4'
13 }
14 connection = pymysql.connect(**db_config)
```

Mając działające połączenie z bazą umożliwiono odczyt danych. Aby to osiągnąć należało stworzyć funkcję, która wykonuje zapytanie SQL'owe odczytujące dane z przekazanej tabeli, a następnie za pomocą kurwora dane zostają przekazane do wyjścia w postaci krotki (kolumny,wiersze).

```
6  # Konfiguracja połączenia z bazą MySQL
7 db_config = {
8     'user': 'root',
9     'password': '2203',
10    'host': 'localhost',
11    'database': 'rekinki',
12    'charset': 'utf8mb4'
13 }
14
15 # Funkcja do pobierania danych z tabeli
16 def fetch_data(table_name):
17     connection = pymysql.connect(**db_config)
18     try:
19         with connection.cursor() as cursor:
20             cursor.execute(f"SELECT * FROM {table_name}")
21             columns = [desc[0] for desc in cursor.description] # Nazwy kolumn
22             rows = cursor.fetchall() # Wszystkie wiersze
23             return columns, rows
24     finally:
25         connection.close()
```

Flask pozwala na tworzenie prostych stron internetowych w zmodyfikowanym języku HTML z opcją wstrzykiwania kodu pythonowego na podstawie przekazanych do wzorca danych.

Przygotowany został wzorzec HTML'owy który wyświetla dane odczytywanych tabel w prosty i czytelny sposób:

```
1 1<!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Dane z bazy rekinów</title>
6 </head>
7 <body>
8     <h1>Tabele w bazie danych</h1>
9     {%- for table_name, table_data in tables.items() %}<br>
10    <h2>{{ table_name }}</h2>
11    <table border="1" cellpadding="5" cellspacing="0">
12        <thead>
13            <tr>
14                {%- for column in table_data.columns %}<br>
15                    <th>{{ column }}</th>
16                {%- endfor %}<br>
17            </tr>
18        </thead>
19        <tbody>
20            {%- for row in table_data.rows %}<br>
21            <tr>
22                {%- for cell in row %}<br>
23                    <td>{{ cell }}</td>
24                {%- endfor %}<br>
25            </tr>
26            {%- endfor %}<br>
27        </tbody>
28    </table>
29    {%- endfor %}<br>
30 </body>
31 </html>
```

Aby wyświetlić ten wzorzec na stronie należy zmodyfikować metodę renderującą stronę tak, aby najpierw gromadziła dane a następnie przekazywała je do wyświetlenia przed wczytaniem docelowej strony:

```
27 @app.route('/')
28 def show_tables():
29     # Nazwy tabel w bazie danych
30     table_names = [
31         "siedliska", "diety", "zagrozenia", "naukowcy",
32         "badania", "rekiny", "rekiny_siedliska",
33         "rekiny_zagrozenia", "rekiny_badania"
34     ]
35
36     # Pobieranie danych z każdej tabeli
37     tables = {}
38     for table in table_names:
39         columns, rows = fetch_data(table)
40         tables[table] = {"columns": columns, "rows": rows}
41
42     # Renderowanie HTML
43     return render_template("tabelki.html", tables=tables)
```

Flask na pythonie, dlaczego taki wybór?

Ze względu na wcześniejsze zapoznanie grupy z pythonem, zdecydowaliśmy się skorzystać właśnie z tego języka. Po krótkiej analizie doszliśmy do dwóch potencjalnych rozwiązań – Flask lub Django. Django jest skomplikowanym zbiorem bibliotek służących do tworzenia wielowarstwowych aplikacji webowych, za to Flask do prostych aplikacji – takich jak nasza.