

SQL w Spatialite krok po kroku

Aby zaliczyć te zajęcia należy otworzyć w Wordzie lub openoffice pusty plik do edycji tekstu – będzie to kartka z odpowiedziami podlegającymi sprawdzeniu – w pierwszej linii tego pliku proszę wpisać swoje imię i nazwisko.

Następnie należy w bazie danych wykonywać kolejne ćwiczenia tworząc i aktualizując swoją bazę danych. W momencie natrafienia w tekście na polecenia **Zadanie do kartki z odpowiedziami:** należy samodzielnie wykonać zadanie a jego przebieg opisać w pliku kartka z odpowiedziami – proszę tam przekopiować swoje zapytania oraz wpisywać komentarze. Uwaga odpowiedzi zamieszczone przez studenta w tym pliku oraz treść bazy danych stanowią podstawę zaliczenia zajęć bazy danych. Po zrealizowaniu całego konspektu baza i kartka z odpowiedziami musi zostać zgłoszona prowadzącemu do oceny.

1. Utwórz bazę danych

Uruchom program **spatialite_gui.exe** W menu wybierz **Files > Creating a New (empty) SQLiteDB** Wskaż katalog w którym znajduje się twoja baza aby w nim umieścić plik bazy danych o nazwie na przykład db1. – narzędzie GUI utworzy plik na dysku i podłączy od niego bazę.

2. Tworzenie, usuwanie tabel bazy danych

SQLite obsługuje następujące instrukcje Data Definition Language (DDL): CREATE, ALTER TABLE, DROP.

W SQLite, instrukcja CREATE służy do tworzenia tabel, indeksów, widoków i wyzwalaczy.

ALTER TABLE umożliwia zmianę struktury tabeli. Instrukcja DROP usuwa tabelę, indeksy, widoki i wyzwalacze.

Aby utworzyć tabelę, podajemy nazwę tabeli i nazwy jej kolumn. Każda kolumna może być jednym z tych rodzajów danych:

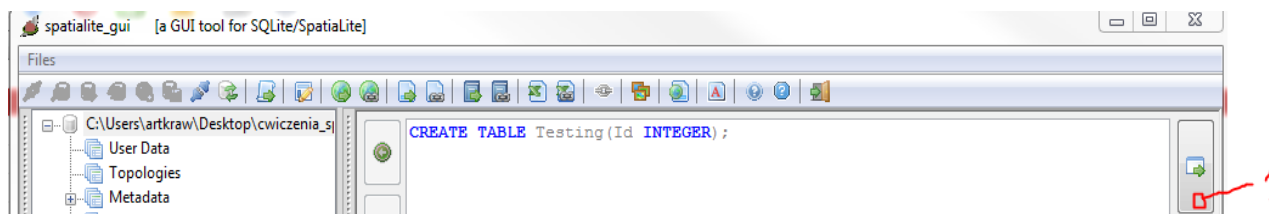
NULL - wartość jest wartością pustą, INTEGER - liczba całkowita ze znakiem, REAL – wartość rzeczywista zmiennoprzecinkowa, TEXT - ciąg znaków, BLOB – **B**inary **L**arge **O**bject, binarny fragment (obiekt) pliku danych.

CREATE

Utwórz tabelę o nazwie Testing z jedną kolumną o nazwie ID która będzie typu liczba całkowita:

CREATE TABLE Testing(Id **INTEGER**, Opis **TEXT**);

Zapytanie piszemy w oknie edycji, a jego wykonanie zatwierdzamy klawiszem oznaczonym na rysunku poniżej jako klawisz nr. 1

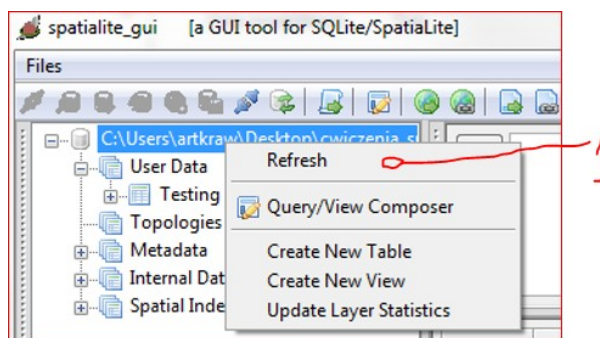


Rys. 1 Przycisk uruchamiania zapytania do bazy danych

Po wykonaniu zapytania w oknie poniżej otrzymujemy informację zwrrotną:

SQL query returned an empty ResultSet, This is not an error

to jest potwierdzenie wykonania operacji, która nie zwróciła żadnych rekordów ale operacja została wykonana prawidłowo. Nie każde zapytanie jest przygotowane po to aby zwracać zwracać rekordy. Aby zobaczyć nazwę naszej tabeli w User Data trzeba kliknąć prawym przyciskiem myszki na nazwę ścieżki do pliku bazy i wykonać polecenie z menu – Refresh (zobacz na rysunek nr 2.).

SQL, Spatialite personalna baza danych
kategoria: SQL_Databases

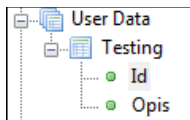
Rys. 2 Opcja Menu podłączonej bazy danych odświeżająca odczyt

Wtedy w sekcji USER DATA pojawi się tabela Testing. Utwórzmy tabelę Testing2, która jest identyczna co do struktury (ilość kolumn i ich nazwy będą identyczne) z Tabelą Testing.

Wykorzystamy do tego celu zapytanie:

```
CREATE TABLE Testing2 AS SELECT * FROM Testing;
```

W zapytaniu tym definicję tabel – nazwę kolumn jak i typ danych tych kolumn zostały podstawione za pomocą instrukcji AS. Najpierw bowiem zostało wykonane zapytanie SELECT * FROM – jego wynik został podstawiony do głównego zapytania CREATE TABLE. Na zakończenie przejrzyj kolumny tabeli Testing2 i Testing.



Rys. 3 Podgląd nazw kolumn, rozwinięcie węzła gałęzi tabeli Testing

DROP

Instrukcja DROP służy do usuwania tabeli z bazy danych. Usuń tabelę Testing2:

```
DROP TABLE Testing2;
```

Wykonajmy to zapytanie jeszcze raz – dostaniemy wtedy komunikat o błędzie.

W języku SQL możemy używać również instrukcji warunkowych typowych dla zwykłych języków programowania. Aby uniknąć wyświetlenia błędu można użyć instrukcji warunkowej:

```
DROP TABLE IF EXISTS Testing2;
```

Zapytanie to można wytłumaczyć następująco:

Jeśli tabela Testing2 istnieje wtedy ją usuń. Jeśli nie istnieje wtedy nie wykonuj zapytania DROP.

Zapytanie to usunie więc tabelę tylko i wyłącznie wtedy gdy ona będzie istniała w bazie. Niestety komunikat potwierdzający wykonanie operacji zawsze wyświetla się ten sam.

ALTER TABLE

SQLite (a tym samym Spatialite) obsługuje dość ograniczony (w stosunku do standardu SQL92) podzbiór instrukcji ALTER TABLE. To zapytanie w SQLite pozwala użytkownikowi zmienić nazwę tabeli lub dodać nową kolumnę do istniejącej tabeli. Nie jest możliwe w celu zmiany nazwy kolumny jej usunięcie i dodanie nowej. Natomiast jest możliwa zmiana nazwy kolumny.

Utwórzmy tabelę Names z dwoma kolumnami Id i Name:

```
CREATE TABLE Names(Id INTEGER, Name TEXT);
```

Teraz zmienimy jej nazwę z Names na Surname używając zapytania:

```
ALTER TABLE Names RENAME TO Surname;
```

Dodajmy nową kolumnę email (typu tekst) do tabeli Surname:

```
ALTER TABLE Surname ADD COLUMN Email TEXT;
```

Odśwież widok drzewa obiektów tabeli bazy danych i sprawdź jakie kolumny posiada tabela Surname. Podsumuj swoją wiedzę na temat poznanych instrukcji SQL.

3. Tworzenie, modyfikowanie i usuwanie danych w tabelach bazy danych

INSERT

- instrukcja ta jest częścią języka Data Manipulation Language (DML). Służy do wstawiania wartości danych do pól w tabelach. W SQL zakładamy, że do tabeli dane wprowadzamy wierszami (rekordami). Ta komenda tworzy nowy wiersz w bazie danych.

Stworzymy nową tabelę Books, w której utworzymy kolumnę Id, która stanie się kluczem głównym tej tabeli oraz wpisujemy wartość domyślną w kolumnie numer ISBN książki.

```
CREATE TABLE Books(Id INTEGER PRIMARY KEY, Title TEXT, Author TEXT, ISBN TEXT DEFAULT 'not available');
```

Proszę odświeżyć widok obiektów bazy danych i rozwinąć gałąź Book. Kolumna Id posiada inną ikonę oznaczającą klucz główny tabeli Books.

Wprowadźmy 1 książkę – Lwa Tołstoja „War and Peace”. Zapytanie będzie wyglądało następująco:

```
INSERT INTO Books(Id, Title, Author, ISBN) VALUES(1, 'War and Peace', 'Lew Tolstoj', '978-0345472403');
```

To jest klasyczny INSERT języka SQL. Mamy określone wszystkie nazwy kolumn po nazwie tabeli (lista kolumn) i wszystkie wartości po słowie kluczowym VALUES (lista wartości dla kolumn). Uwaga1 porządek listy kolumn musi być idealnie zgodny z porządkiem listy wartości. Uwaga2 porządek listy kolumn może być całkowicie różny od kolejności kolumn w tabeli w bazie danych. Dodajmy nasz pierwszy wiersz do tabeli.

Dodajmy drugi wiersz – zauważ nie ma na liście kolumny ID ani wartości dla niej!

```
INSERT INTO Books(Title, Author, ISBN) VALUES('The Brothers Karamazov', 'Fiodor Dostojewski', '978-0486437910');
```

Baza dopisała wartość klucza głównego samodzielnie. Ponieważ Klucz Główny tabeli nie może być pusty.

Oglądnij treść tabeli wpisując zapytanie: `select * from books;`

A teraz 3 cecha zapytania insert into:

```
INSERT INTO Books VALUES(3, 'Crime and Punishment', 'Fiodor Dostoejewski', '978-1840224306');
```

W tym zapytaniu, nie określono nazwy kolumn po nazwie tabeli. W takim przypadku, trzeba dostarczyć wszystkie wartości dla wszystkich kolumn w tabeli.

Sprawdź zapytaniem Select treść wierszy tabeli Books.

Co dzieje się gdy nie podamy wszystkich wartości ? przeanalizujmy to zapytanie:

```
INSERT INTO Books(Id, Title) VALUES(4, 'Paradise Lost');
```

ZAUWAŻ Brakuje tutaj wartości dla dwóch ostatnich kolumn – zostaną one wypełnione wartościami domyślnymi.

Oglądnij treść tabeli wpisując zapytanie: `select * from books;`

Dowiadujemy się o brakujących wartościach. Autorem pozycji Paradise Lost jest John Milton a nr ISBN jest 978-0486442877. Możemy więc uzupełnić zapytanie:

```
INSERT INTO Books VALUES(4, 'Paradise Lost', 'John Milton', '978-0486442877');
```

Niestety otrzymamy błąd o treści: 'PRIMARY KEY must be unique'.

WNIOSEK Instrukcja INSERT INTO nie może służyć do aktualizacji danych.

1 Rozwiązanie.

Musimy użyć modyfikatora zapytania – instrukcji .

```
INSERT OR REPLACE INTO Books VALUES(4, 'Paradise Lost', 'John Milton', '978-0486442877');
```

Wtedy instrukcja zostanie wykonana prawidłowo.

2 Rozwiązanie.

to instrukcja UPDATE, którą omówimy w następnej części konspektu.

Powtórzmy teraz komendy z języka DCL

Najpierw tworzenie kopii tabeli bazy danych. Utwórzmy kopię tabeli Books o nazwie Books2.

```
CREATE TABLE Books2 AS SELECT * FROM Books;
```

Zapytaniem SQL – Select sprawdź jak wygląda tabela Books2.

Zadanie 1. do kartki z odpowiedziami: Na czym polega dokładnie działanie instrukcji CREATE TABLE

SQL, Spatialite personalna baza danych

kategoria: SQL_Databases

powiązanej z aliasem zapytania AS SELECT w kontekście instrukcji Create opisanej na drugiej stronie niniejszego konspektu (tworzenie Testing2).

Usuwanie danych

Słowo kluczowe DELETE służy do usuwania danych z tabel. Po pierwsze, chcemy, usunąć jeden wiersz z tabeli. Będziemy korzystać z tabeli Books2, którą stworzyliśmy wcześniej.

Możemy usunąć wiersz z ID 1.

```
DELETE FROM Books2 WHERE Id=1;
```

Zapytaniem SQL – Select sprawdź jak wygląda tabela Books2. A teraz usuń wszystkie wiersze w tabeli Books2.

```
DELETE FROM Books2;
```

Zapytaniem SQL – Select sprawdź jak wygląda tabela Books2.

Samodzielnie usuń tabelę Books2.

Aktualizacja danych

Słowo kluczowe UPDATE służy do zmiany wartości kolumn w wybranych wierszach tabeli.

Chcemy zmienić "Lew Tolstoj" do "Lew Nikolajewicz Tolstoj" w naszej tabeli Books. Poniższa instrukcja pokazuje jak tego dokonać:

```
UPDATE Books SET Author='Lew Nikolajewicz Tolstoj' WHERE Id=1;
```

Zapytaniem SQL – Select sprawdź jak wygląda wiersz Id tabeli Books.

```
SELECT * FROM Books WHERE Id=1;
```

Zadanie 2. do kartki z odpowiedziami: Do tabeli Books dodaj kolumnę Price (język kreacji danych)

Uwaga zastosuj prawidłowy typ danych, ponieważ ceny mogą mieć również wartości groszy. Wprowadź fikcyjne wartości kosztów tych książek (zapytanie aktualizujące dane – które lepsze UPDATE czy Replace?). Samodzielnie ułóż i wykonaj własne zapytanie dodające nowy wiersz do tej tabeli (Books).

4. Zapytanie Select, wybieranie danych z tabel

Samodzielnie utwórz tabelę o strukturze i wartościach danych tak jak wskazano poniżej:

Id	Name	Cost
1	Audi	52642
2	Mercedes	57127
3	Skoda	9000
4	Volvo	29000
5	Bentley	350000
6	Moskwicz	1000
7	Volvo	41400
8	Volkswagen	31600
9	Vauxhall	24800

Wyświetlanie danych z tabeli

Wykonaj zapytanie wyświetlające całą zawartość tabeli :

```
SELECT * FROM Cars;
```

Zauważ, że nazwy kolumn zgodne są ze zdefiniowanymi w zapytaniu Create Table. Wykonaj teraz zapytanie wyświetlające wybrane kolumny tabeli:

```
SELECT Name, Cost FROM Cars;
```

W wyniku zapytania SELECT nie chcemy aby była używana nazwa kolumny Cost i chcemy by w wyniku zapytania zamiast Cost wyświetliła się nazwa kolumny Cena – zakładamy w tym celu alias za pomocą instrukcji AS:

```
SELECT Name, Cost AS Cena FROM Cars;
```

Ograniczanie zakresu wyświetlanych danych

SQL, Spatialite personalna baza danych

kategoria: SQL_Databases

Możemy użyć klauzuli LIMIT, aby ograniczyć ilość danych zwracanych przez instrukcję Select.

```
SELECT * FROM Cars LIMIT 4;
```

Sprawdź i opisz jakie restrykcje powodują pozostałe parametry instrukcji LIMIT:

```
SELECT * FROM Cars LIMIT 2, 4;
```

```
SELECT * FROM Cars LIMIT 4 OFFSET 3;
```

Zadanie 3. do kartki z odpowiedziami: Na podstawie wyników tych zapytań wyjaśnij opcje (warianty) instrukcji Limit. Poproś zadawanie pytań z innymi wartościami instrukcji Limit, które uzasadnią twoją odpowiedź.

Porządkowanie wyświetlanych danych

Uporządkowanie kolejności danych w tabeli Cars ze względu na cenę przez instrukcję ORDER BY:

```
SELECT * FROM Cars ORDER BY Cost;
```

Domyślnym porządkiem jest porządek rosnący ASC – zmienimy porządek na malejący DESC.

```
SELECT Name, Cost FROM Cars ORDER BY Cost DESC;
```

Wyświetlanie danych ograniczonych warunkiem – tzw. zapytanie „wybierające” dane

Warunkowe wybieranie danych polega na użyciu instrukcji WHERE. Wybierz auto które posiada identyfikator nr 6

```
SELECT * FROM Cars WHERE id=6;
```

Wybierz auto po nazwie jego marki:

```
SELECT * FROM Cars WHERE Name='Volvo';
```

Wybierz wszystkie auta których Name (nazwa) zaczyna się od litery M:

```
SELECT * FROM Cars WHERE Name LIKE 'M%';
```

Wybierz wszystkie auta których Cost (cena) jest mniejsza od 30 000:

```
SELECT * FROM Cars WHERE Cost < 30000;
```

UWAGA na GISowym marginesie

„Zapytanie SELECT jest absolutnie najważniejszym zapytaniem wykorzystywanym w systemach GIS.”

Zadanie 4. do kartki z odpowiedziami:

1. Zadać zapytanie – wyświetl samochody, które są droższe niż 20000 w kolejności od najdroższego do najtańszego.
2. Zadać zapytanie – wyświetl tylko kolumnę Cost jako Cena samochodu, które są tańsze niż 40000 w kolejności od najtańszego do najdroższego.
3. Zadać zapytanie – wyświetl tylko dwie kolumny Name i Cost jako Nazwa i Cena z tabeli Cars, których nazwa zaczyna się na literę V, w kolejności od najdroższego do najtańszego.

Grupowanie danych

Klauzula GROUP BY jest używana do łączenia rekordów bazy danych o identycznych wartościach w jednym rekordzie. Stosuje się je często z funkcjami agregacji.

Chcemy dowiedzieć się sumę kosztów każdej z marek. Zwróć uwagę na markę volvo.

```
SELECT sum(Cost) AS Price, Name FROM Cars GROUP BY Name;
```

5. Łączenie tabel - wewnętrzne

W tej części będziemy uczyć się łączenia tabel w SQL. Prawdziwa siła i korzyści z relacyjnych baz danych pochodzą z tabel łączonych. Klauzula JOIN łączy rekordy z dwóch lub więcej tabel w bazie danych. Istnieją dwa rodzaje łączy: wewnętrzne i zewnętrzne (INNER and OUTER.). W tej części ćwiczeń, będziemy pracować z tabelami: Customers i Reservations. Samodzielnie utwórz tabelę Customers:

CustomerId	Name
1	Paul Novak
2	Terry Neils
3	Jack Fonda

4 Tom Willis

Oraz tabelę Reservations:

Id	CustomerId	Day
1	1	2009-22-11
2	2	2009-28-11
3	2	2009-29-11
4	1	2009-29-11
5	3	2009-02-12

Łączenia wewnętrzne INNER JOIN

Łączenie wewnętrzne jest najczęstszym typem łączenia. Jest to domyślne łączenie, które wybiera tylko te rekordy z tabel bazy danych, które mają pasujące wartości. Mamy trzy rodzaje połączeń wewnętrznych: INNER JOIN, NATURAL INNER JOIN i CROSS INNER JOIN. Uwaga słowo kluczowe INNER może być pominięte w trakcie wykonywania zapytań.

```
SELECT Name, Day FROM Customers AS C JOIN Reservations AS R ON C.CustomerId=R.CustomerId;
```

Użyliśmy tu aliasów R i C dla tabel Reservations i Customers. Natomiast użycie kropki oznacza wskazanie atrybutu w zapytaniu wielotabelowym. Tak aby baza wiedziała z której tabeli ma wybrać kolumnę CustomerId. Analogiczne zapytanie bez instrukcji JOIN:

```
SELECT Name, Day FROM Customers, Reservations WHERE Customers.CustomerId = Reservations.CustomerId;
```

Łączenie NATURAL INNER JOIN

NATURAL INNER JOIN automatycznie wykorzystuje wszystkie pasujące nazwy kolumn, które można dołączyć. W naszych tabelach, mamy kolumnę o nazwie customerId w obu tabelach.

```
SELECT Name, Day FROM Customers NATURAL JOIN Reservations;
```

Dostajemy tą samą odpowiedź ale pytanie jest znacznie krótsze.

Łączenie CROSS INNER JOIN

CROSS INNER JOIN łączy wszystkie rekordy z jednej tabeli ze wszystkimi rekordami z innej tabeli. Ten rodzaj sprzężenia ma małą wartość praktyczną. Jest również nazywany iloczynem kartezjańskim rekordów.

```
SELECT Name, Day FROM Customers CROSS JOIN Reservations;
```

Uwaga to samo można osiągnąć zapytaniem:

```
SELECT Name, Day FROM Customers, Reservations;
```

6. Łączenie tabel - zewnętrzne

Sprzężenie zewnętrzne nie wymaga aby każdy rekord w dwóch połączonych tabelach miał te same wartości. Istnieją trzy typy zewnętrznego łączenia tabel: Łączenia zewnętrzne lewostronne, prawostronne, oraz pełne łączenia. SQLite obsługuje tylko lewostronne łączenia zewnętrzne.

LEFT OUTER JOIN

LEFT OUTER JOIN zwraca wszystkie wartości z lewej tabeli, nawet jeśli nie ma pasującego rekordu z prawej tabeli. W takich wierszach będzie wartość NULL. Innymi słowy, w lewo sprzężenie zewnętrzne zwraca wszystkie wartości z lewej tabeli, plus dopasowane wartości z prawej tabeli. Należy zauważyć, że słowo OUTER może być pominięte.

```
SELECT Name, Day FROM Customers LEFT JOIN Reservations ON Customers.CustomerId = Reservations.CustomerId;
```

Tutaj mamy wszystkich klientów z ich rezerwacjami, oraz jednego klienta, który nie ma rezerwacji. Ma wartość NULL w swoim wierszu.

Istnieje jeszcze wersja Natural Left Outer Join

```
SELECT Name, Day FROM Customers NATURAL LEFT OUTER JOIN Reservations;
```

Instrukcje agregujące

Są to instrukcje matematyczne pozwalające na wykonanie obliczeń oraz agregacji zbiorów danych. Wiele z nich jest dość pożytecznych.

Wybierz wartości maksymalne i minimalne z tabeli Cars:

```
SELECT max(Cost), min(Cost) FROM Cars;
```

Policz ile jest samochodów w tabeli Cars

```
SELECT count(*) AS 'Ilosc aut' FROM Cars;
```

Oblicz średnią cenę samochodu w tabeli Cars

```
SELECT avg(Cost) AS 'Srednia cena auta' FROM Cars;
```

Zadanie 5. do kartki z odpowiedziami: Wykonaj sekwencję zapytań instrukcji agregujących:

1. Oblicz średnią cenę samochodu w tabeli Cars. (zapisz ją na kartce)
2. Zaktualizuj za pomocą UPDATE wartość samochodu Bentley na 0 (zero) (zapisz ją na kartce)
3. Oblicz wartość średnią aut oraz minimalną wartość auta podawaną przez bazę. (zapisz ją na kartce)
4. Zaktualizuj za pomocą UPDATE wartość samochodu Bentley na NULL.
5. Oblicz wartość średnią aut oraz minimalną wartość auta podawaną przez bazę. (zapisz ją na kartce)

Skomentuj wyniki. (po zakończeniu przywróć pierwotną wartość samochodu Bentley)

Kilka instrukcji wewnętrznych

Tak zwane instrukcje wewnętrzne są to instrukcje producenta baz danych wbudowane przez niego, które nie są zdefiniowane w żadnym standardzie SQL. Żadna organizacja standaryzująca nie wprowadziła tych instrukcji do stosowania. Każda baza danych je oczywiście musi posiadać w tej lub innej formie.

Wersja bazy danych:

```
SELECT sqlite_version() AS 'SQLite Version';
```

Wylosuj liczbę:

```
SELECT random() AS Wylosowano;
```

Funkcja random () zwraca pseudolosowych całkowitą między -9223372036854775808 i 9223372036854775807.