Język SQL (Structured Query Language)

Podstawowe informacje oraz rys historyczny.

Strukturalny język zapytań SQL to skrót od *Structured Query Language* tj. „język zapytań strukturalnych".

Język **SQL** opracowano w latach siedemdziesiątych w firmie IBM. Język w tej wersji, nazywany był **Sequel**. Język **Sequel** rozwinął się od tego czasu i jego nazwa została zmieniona na **SQL**.

Język SQL w zasadzie jest implementacją działań wewnętrznych relacji. Działania te to:

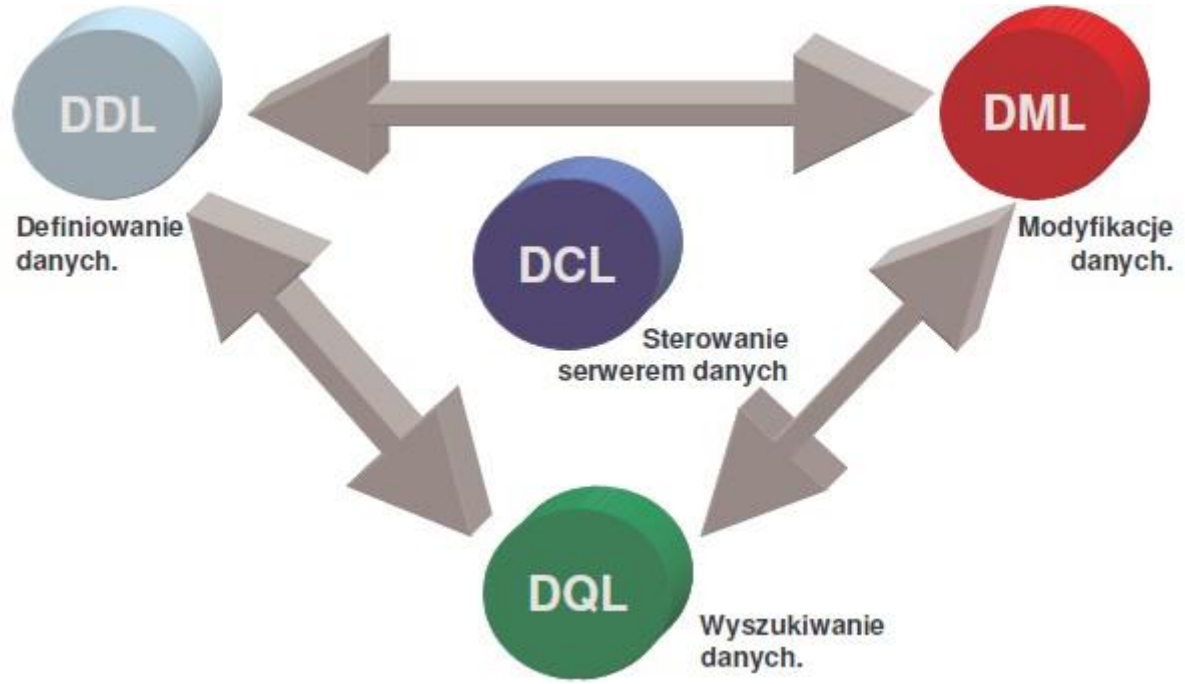
SELEKCJA pobieranie wybranych wierszy z relacji, które spełniają zadany warunek
PROJEKCJA operacja pobrania wskazanych kolumn relacji wierszy
ILOCZYN KARTEZJAŃSKI wynik połączenia każdy z każdym wierszy z dwóch relacji
ZŁĄCZENIE połączenie dwóch relacji poprzez zadane kryterium wiersze z obu relacji
SUMA RELACJI wszystkie wiersze z obu relacji
ILOCZYN RELACJI (CZĘŚĆ WSPÓLNA) wiersze wspólne dla obu relacji
RÓŻNICA RELACJI -wiersze, które występują w jednej, a nie występują w drugiej relacji

W **1986 roku**, SQL stał się oficjalnym standardem, wspieranym przez Międzynarodową Organizację Normalizacyjną - ISO i jej członka, Amerykański Narodowy Instytut Normalizacji - ANSI. Ten standard SQL został nazwany SQL-86 (SQL-1).

Rozszerzeniem standardu SQL-86 był SQL-89.

Pojawiła się potrzeba określenia standardu ściślejszego. Powinien on jednocześnie obejmować nowe elementy, nieujęte do tej pory w języku. Tak powstał w 1992 roku SQL2 (SQL-92).

SQL-92 jest ponad pięć razy obszerniejszy od pierwszej wersji.



SQL w standardzie SQL-92 jest podjęzykiem danych, co oznacza, że jest on wykorzystywany wyłącznie do komunikacji z bazą danych. Nie posiada on cech pozwalających na tworzenie kompletnych programów.

SQL w standardzie SQL-92 jest językiem nieproceduralnym (deklaratywnym), określa on jedynie to, co ma być zrobione, nie podaje sposobu rozwiązania. Ten sposób pozwala na dużą elastyczność twórcom baz danych.

Przykład: Wydając polecenie:

```
SELECT imie, nazwisko
FROM Osoba
WHERE imie = 'Julia'
```

deklaruję, że z tabeli Osoba chcę wybrać krotki, w których pole 'imie' ma wartość 'Julia', nie precyzując wcale sposobu (procedury) wykonania. System zarządzania baz danych sam dokona interpretacji polecenia i ustali optymalny sposób jego wykonania.

Język SQL jest językiem interpretowanym.

SQL_99 – (rdzenna specyfikacja + dodatkowe pakiety) Standard języka SQL został uzupełniony w 1999 roku, tak, że stał się on w pełni językiem, w którym można programować.

SQL3, SQL4 – rozszerzenia z 2004 r.

Składnia języka SQL w standardzie SQL2.

Język SQL w wersji SQL2 zawiera następujące składowe:

- **DDL**- Data Definition Language (ang. język definiowania danych). Umożliwia definiowanie struktury danych przechowywanych w bazie, czyli tworzenie schematu implementacyjnego.
- **DML** – Data Manipulation Language (ang. język manipulowania danymi). Umożliwia wypełnianie, modyfikowanie i usuwanie informacji z baz danych.
- **DCL** – Data Control Language (ang. język sterowania danymi). Umożliwia sterowanie transakcjami.

Podział ten okazał się niewystarczający i nieprecyzyjny, dlatego w SQL3 wyróżnia się siedem klas instrukcji:

Podstawy języka SQL — Klasy poleceń SQL99

KLASA	Opis	Przykłady
Obsługa połączenia.	Sterowanie sesją - połączeniem klienta z serwerem.	CONNECT, DISCONNECT
Strukturalne	Sterowanie wykonywaniem zbiorów poleceń.	CALL, RETURN
Przetwarzanie danych	Wyszukiwanie danych połączone z modyfikowaniem nietrwałym lub modyfikowanie trwałe.	SELECT, INSERT, UPDATE, DELETE
Diagnostyka systemu	Wyświetlanie informacji o kondycji i stanie pracy serwera, błędach i sytuacjach wyjątkowych.	GET DIAGNOSTICS, SHOW ERRORS
Obsługa schematów	Modyfikowanie struktur przechowujących dane w serwerze.	CREATE, ALTER, DROP
Obsługa sesji	Modyfikowanie parametrów sterujących procesem przetwarzania informacji w sesji klienta	SET
Obsługa transakcji	Synchronizowanie współpracy wielu użytkowników i dzielenie procesów przetwarzania danych na jednostki elementarne.	SET TRANSACTION, COMMIT, ROLLBACK

1. Connection Statements (CONNECT, DISCONNECT)
2. Control Statement (CALL, RETURN)
3. Data Statement (SELECT, INSERT, UPDATE, DELETE)
4. Diagnostics Statement GET DIAGNOSTICS, TRY – CATCH, RAISERROR
5. Schema Statement (CREATE, ALTER, DROP)
6. Session Statement (SET)

7. Transaction Statement (START, BEGIN, COMMIT, ROLLBACK)

W języku SQL przyjmuje się najczęściej następujące reguły:

- Polecenia SQL nie rozróżniają „wielkości" liter.
- Polecenie SQL może być wprowadzane w jednej lub wielu liniach.
- Słowa kluczowe nie mogą być dzielone pomiędzy linie ani skracane.
- Klauzule (fragment polecenia SQL, pełniącego wyodrębnione funkcje) powinny być umieszczane w osobnych liniach dla przejrzystości i ułatwienia edycji.
- Akapity (wcięcia) powinny być używane dla poprawienia czytelności kodu.

Dialekty SQL

Do najpopularniejszych dialektów języka SQL należą:

1. T- SQL – dialekt związany z SQL Server i Sybase
2. PL/SQL – dialekt stosowany w serwerach firmy Oracle
3. PL/pgSQL – PostgreSQL
4. SQL PL – serwery bazodanowe firmy IBM

Każda firma dąży do przekonania użytkownika, że zaimplementowany dialekt jest najbardziej odpowiedni i optymalny. Czy tak jest przekonają się Państwo w trakcie nauki.

W bieżącym wykładzie ograniczymy się do omawiania klasycznego standardu SQL2. W dalszych wykładach rozszerzymy informacje o SQL3 i SQL4.

Typy danych atrybutów i dziedziny wartości w standardzie SQL2

Podstawowe typy danych: **typy numeryczne, ciągi znakowe, ciągi bitowe, wartości logiczne, data i czas.**

- **Numeryczne –** INTEGER (INT), FLOAT lub REAL, DOUBLE PRECISSION, DECIMAL(i, j), DEC(i, j), NUMERIC(i, j)
- **Ciągi znaków –** CHAR(n), CHARACTER(n), VARCHAR(n), VARYING(n), CHARACTER VARYING(n)
- **Ciągi bitowe –** BIT(n), BIT VARYING(n)
- **Logiczny –** TRUE, FALSE
- **Data –** DATE format ('YYYY-MM-DD')
- **Czas –** TIME format 'HH:MM:SS', TIME(i), i – precyzja ułamka sekund, zadeklarowanie dodatkowych miejsc dziesiętnych, TIME WITH TIME ZONE – sześć dodatkowych pozycji dla +HH:SS (+13:00 –12:59)
- **Znacznik czasowy –** TIMESTAMP (obejmuje zarówno pola DATE, TIME jak i opcjalnie TIMEZONE, np. '2008-10-07 09:12:47 09:00', INTERVAL – przedział czasowy

Wartość NULL – logika trójwartościowa

Postulaty Codda wymagają możliwości przetwarzania wartości nieznanej "**NULL**", która jest różna od zera lub pustego ciągu znaków. Porównanie wartości NULL z dowolną inną wartością daje wartość nieznaną (Unknown), a nie prawdę lub fałsz. Trzeba pamiętać o tej **logice trójwartościowej**, konstruując zapytania do bazy danych.

1. Wynikiem wszystkich operacji zawierających NULL jest wartość NULL.
2. SELECT NULL/0, 'jan' + NULL, 5 + NULL, 12*NULL;
3. Dwie wartości NULL nie mogą być porównane (nie są równe ani różne od siebie)
4. Porównanie NULL z inną wartością daje zawsze wartość nieznaną

Notacja Backusa – Naura (BNF)

Do zapisu składni poleceń SQL będziemy używać notacji BNF).

- Wielkie litery – słowa kluczowe danego polecenia
- Małe litery – słowa definiowane przez użytkownika
- Pionowa kreska – oznacza możliwość wyboru (alternatywa)
- Nawiasy klamrowe – element obligatoryjny (musi wystąpić)
- Nawiasy kwadratowe – element opcjalny (może wystąpić)
- Nawiasy okrągłe – możliwość wielokrotnego powtarzania elementu

Wykaz podstawowych poleceń języka SQL w standardzie SQL92

DDL - tworzenie, usuwanie oraz zmiana schematów tabel.

Polecenie	*Typ*	*Opis*
CREATE SCHEMA	DDL	Tworzy schemat bazy danych
CREATE TABLE	DDL	Tworzy tabelę i definiuje jej kolumny oraz alokację przestrzeni dla danych
CREATE VIEW	DDL	Definiuje widok dla jednej lub większej liczby tabel lub innych widoków
CREATE INDEX	DDL	Tworzy indeks dla tabeli
CREATE DOMAIN	DDL	Tworzy domenę
ALTER TABLE	DDL	Dodaje kolumnę do tabeli, redefiniuje kolumnę w istniejącej tabeli lub redef. ilość miejsca zarezerwowaną dla danych
ALTER DOMAIN	DDL	Modyfikuje domenę
DROP obiekt **SCHEMA** **DOMAIN** **TABLE** **VIEW** **INDEX**	DDL	Usuwa indeks, sekwencje, tablicę, widok lub inny obiekt
RENAME obiekt	DDL	Zmienia nazwę tabeli, widoku lub innego obiektu
SET TRANSACTION	DDL	Zaznacza aktualną transakcję jako read-only (tylko do odczytu).

DML - język manipulacji danymi

Polecenie	*Typ*	*Opis*
SELECT	DML	Wykonuje zapytanie. Wybiera wiersze i kolumny z jednej lub kilku tabel
INSERT	DML	Dodaje nowy wiersz (lub wiersze) do tabeli lub widoku
UPDATE	DML	Zmienia dane w tabeli
DELETE	DML	Usuwa wszystkie lub wyróżnione wiersze z tabeli
COMMIT	DML	Kończy transakcję i na stałe zapisuje zmiany
ROLLBACK	DML	Wycofuje zmiany od początku transakcji lub zaznaczonego punktu.
SAVEPOINT	DML	Zaznacza punkt, do którego możliwe jest wykonanie rozkazu ROLLBACK

DCL - Kontrola dostępu do bazy danych

Polecenie	*Typ*	*Opis*
GRANT	DML	Nadawanie praw użytkownikowi
REVOKE	DML	Cofanie nadanych uprawnień

Przykład (na początek): Baza danych „Biblioteka”.

Jako obiekt do demonstracji struktury poleceń SQL stwórzmy bardzo prostą bazę danych o nazwie Biblioteka. Niech w bazie tej znajdą się trzy relacje: *Czytelnicy, Ksiazki, Wypozyczenia.*

Tabela *Czytelnicy* zawiera informacje o czytelnikach i wygląda następująco:

nr_czyt	nazwisko	imie	Adres
1	Kowalski	Jan	Bilgoraj Sloneczna 7
2	Michonski	Pawel	Bilgoraj Ladna 18
3	Bil	Ewa	Biłgoraj Armii Ludowej 7
4	Konopka	Maria	

Rys. Tabela Czytelnicy

Tabela książki to nic innego jak wykaz istniejących w bibliotece książek

Nr_ks	Tytul	Autor
2104	Okulary	J. Tuwim
2002	W pustyni i w puszczy	H. Sienkiewicz
2030	Rosyjska ruletka	G. Milton

Rys Tabela Ksiazki

Trzecia tabela *Wypozyczenia* gromadzi informacje o wypożyczeniach (czytelnikach i książkach, które wypożyczyli). Powinniśmy mieć tu informację o czytelniku, informację o wypożyczonej książce oraz dacie wypożyczenia i zwrotu. Aby nie stwarzać redundancji danych zastosowane zostały klucze obce (wyp_ks i wyp_czyt), odwołujące się do odpowiednich krotek tabel *Czytelnicy* i *Ksiazki*.

wyp_ks	wyp_czyt	data_w	data_z
2002	2	2004-02-02	2004-02-17
2104	3	2004-02-01	2004-01-02
2002	1	2004-03-01	

Rys. Tabela Wypozyczenia

Schemat przykładowej bazy może wyglądać następująco:

Biblioteka

```
Czytelnicy(nr czyt, nazwisko, imie, adres, …);
```

```
Ksiazki(nr_ks, tytul, autor, …);
Wypozyczenia(wyp_ks, wyp_czyt,data_w, data_z, …);
```

Schemat uzupełniamy odpowiednimi typami danych oraz potrzebnymi ograniczeniami.

Tworzenie tabel

```
TABLE Czytelnicy(
nr_czyt INTEGER NOT NULL,
nazwisko VARCHAR (35),
imie VARCHAR(30),
adres VARCHAR(200),
PRIMARY KEY (nr_czyt));

TABLE Ksiazki(
nr_ks INTEGER NOT NUll,
tytul VARCHAR(200),
autor VARCHAR(150),
PRIMARY KEY (nr_ks));


TABLE Wypozyczenia(
wyp_ks INT,
wyp_czyt INT,
data_w DATE NOT NULL,
data_z DATE NULL,
PRIMARY KEY(wyp_ks,wyp_czyt,data_w),
FOREIGN KEY(wyp_ks) REFERENCES ksiazki(nr_ks),
FOREIGN KEY(wyp_czyt) REFERENCES czytelnicy(nr_czyt));
```

Tak przedstawiony schemat pozwoli nam w sposób prosty napisać polecenia SQL.

Jak tworzymy bazę danych?

Zakładamy, że mamy odpowiednie konto w SZBD z odpowiednimi uprawnieniami nadanymi przez administratora.

Logujemy się więc do bazy danych (Oracle, MySQL, SQL Server itp.) i aby utworzyć bazę danych musimy przy pomocy poleceń języka SQL wydać polecenia utworzenia odpowiednich obiektów, tj.

1. Utwórz bazę danych - **CREATE SCHEMA**
2. Utwórz tabele bazy danych - **CREATE TABLE**
3. Wprowadź dane do tabel - **INSERT INTO**

Tworzenie bazy danych

W systemach wielodostępnych prawo tworzenia bazy danych posiada administrator (DBA).

W każdym SZBD istnieje katalog utrzymujący zbiór schematów baz danych. **Schemat** obejmuje zespół obiektów bazy danych takich jak tabele, perspektywy, dziedziny, asercje. Wszystkie obiekty w schemacie mają tego samego właściciela i szereg wspólnych wartości domyślnych. Polecenie definiujące schemat ma następującą postać:

CREATE SCHEMA [Nazwa AUTHORIZATION NazwaWlasciciela]

Przykład:

CREATE SCHEMA SqlTest AUTHORIZATION Lojewski;

Standard ISO definiuje także, że powinno być możliwe określenie w tym poleceniu zakresu uprawnień dla użytkowników schematu. Szczegóły zależne są od implementacji. Schemat usuwamy poleceniem:

DROP SCHEMA Nazwa [RESTRICT | CASCADE]

RESTRICT – opcja domyślna, zadziała, jeśli schemat jest pusty,
CASCADE – kaskadowo usuwa wszystkie obiekty związane ze schematem.

UWAGA: Polecenia CREATE i DROP SCHEMA nie są implementowane w wielu SZBD, ze względu na możliwe negatywne skutki użycia.
W MySQL istnieje zaimplementowane polecenie CREATE DATABASE nazwa;

Tworzenie tabel

Gdy tworzymy bazę danych tworzymy odpowiednio tabele zawierające informacje o danych obiektach. Aby utworzyć tabelę należy użyć polecenia CREATE TABLE.

Składnia najprostszego polecenia tworzenia tabeli wygląda następująco:

```
CREATE TABLE nazwa tabeli
(nazwa kolumny typ danych [ograniczenie kolumny] [DEFAULT
domyślna wartość][, ...]);
```

Parametry w powyższym poleceniu to:

- nazwa tabeli – nazwa tworzonej tabeli
- DEFAULT– wartość domyślna
- Typ danych – typ i długość danych dla kolumny

Widać, że tworzenie tabeli polega na definiowaniu jej kolumn. Dla każdej kolumny należy określić nazwę, typ danych i długość (w zależności od typu) oraz tzw. ograniczenia integralnościowe np. czy jest dozwolone istnienie wartości pustej (NULL).

Przykład 1:

```
CREATE TABLE Czytelnicy (
nr_czyt    INTEGER NOT NULL,
nazwisko   varchar (35) ,
imie       varchar (30),
adres      varchar (255) );
```

Powyżej przedstawiono deklarację utworzenia tabeli, która odpowiada schematowi relacji *Czytelnicy*. Pierwszy atrybut *nr_czyt* jest liczbą całkowitą typu integer, wartość ta nie może być pusta. Kolejne trzy atrybuty *nazwisko, imie i adres* są zadeklarowane jako ciągi znaków o zmiennej długości.

Wstawianie danych do relacji.

Dodawanie nowych wierszy do tabeli odbywa się za pomocą polecenia INSERT.

Składnia tego polecenia wygląda następująco:

```
INSERT INTO nazwa tablicy [ (nazwa kolumny,..) ]
VALUES (lista wartości);
```

Parametry:

- nazwa kolumny – lista kolumn wypełnianej tabeli (opcjalnie)
- lista wartości – ujęta w nawiasy lista wartości, po jednej wartości dla każdego atrybutu z listy: (nazwa kolumny,..)

Za pomocą tej składni tylko jeden wiersz naraz może być wstawiony. Lista kolumn nie jest wymagana w klauzuli INSERT. Jednak, jeśli lista kolumn nie zostanie użyta, wartości muszą być podane według domyślnej kolejności i obowiązkowo dla wszystkich kolumn.

Przykład:

```
INSERT INTO Czytelnicy (nr_czyt, nazwisko, imie, adres)
VALUES (1, 'Kowalski', 'Jan', 'Bilgoraj Sloneczna 7');
```

Powyższa instrukcja do listy czytelników dołącza nowego czytelnika. W wyniku wykonania tej instrukcji do relacji

Czytelnicy (nr_czyt, nazwisko, imie, adres),

zostanie dołączona krotka z czteroma wartościami atrybutów wymienionych w liście atrybutów.

Wypełnianie tabel wieloma krotkami

Można również w jednej instrukcji INSERT wprowadzić szereg krotek, które mają zostać wstawione do relacji. W tym celu wykorzystuje się podzapytania. Podzapytanie występuje w miejscu klauzuli VALUES. Liczba kolumn oraz ich typy danych w liście klauzuli INSERT muszą zgadzać się z liczbą i typami danych kolumn w podzapytaniu, np.

Przykład:

```
INSERT INTO ByliCzytelnicy (nr_czyt, nazwisko, imie, adres)
VALUES (SELECT *
                FROM Czytelnicy
                WHERE nr_czyt NOT IN (SELECT wyp_czyt
                                      FROM Wypozyczona)));
```

Usuwanie krotek z relacji.

Usuwanie istniejących wierszy z tabeli odbywa się za pomocą polecenia DELETE.

Składnia polecenia DELETE:

```
DELETE [FROM] nazwa tabeli
[WHERE    warunek];
```

Parametry:

- nazwa tabeli – nazwa tabeli, z której usuwamy krotki
- warunek– identyfikuje wiersze do usunięcia i jest kompozycją składającą się z nazw kolumn, wyrażeń, stałych, podzapytań oraz operatorów

Uwaga: Pominięcie klauzuli WHERE w poleceniu DELETE spowoduje usunięcie wszystkich wierszy tabeli.

Przykład:

```
DELETE FROM Ksiazki
WHERE nr_ks= 2003
AND tytul = 'Ogniem i mieczem'
AND autor = 'H. Sienkiewicz';
```

W powyższym przykładzie z relacji *Ksiazki(nr_ks, tytul, autor)* została usunięta książka o numerze "2003", jeśli jej tytuł brzmi „Ogniem i mieczem”, zaś autor to H. Sienkiewicz.

Zmiany wartości danych w określonych krotkach.

Zmiana wartości całych kolumn w tabeli odbywa się za pomocą polecenia UPDATE.

Składnia polecenia UPDATE:

```
UPDATE nazwa tablicy
SET nazwa kolumny = wartosc [, nazwa kolumny = wartosc,
...]
[WHERE warunek];
```

Parametry:

- nazwa tablicy– nazwa tabeli, którą modyfikujemy
- nazwa kolumny – nazwa kolumny z modyfikowanej tabeli
- wartość– właściwa wartość lub podzapytanie dla kolumny
- warunek – (warunek) identyfikuje wiersze do zmodyfikowania i może zawierać nazwy kolumn, wyrażenia, stałe oraz operatory porównawcze

Instrukcja **UPDATE** zmienia wartości w jednej lub wielu kolumnach w istniejących wierszach tabeli.

Nowe przypisanie zawiera nazwę atrybutu, znak równości oraz wyrażenie. Każde przypisanie jest oddzielone przecinkiem.

W wyniku instrukcji **UPDATE** zostaną wyszukane krotki z tabeli, które spełniają warunek zawarty w klauzuli **WHERE.**

Jeżeli klauzula **WHERE** zostanie pominięta wówczas wszystkie wiersze w tabeli zostaną zmodyfikowane.

Przykład:

```
UPDATE Ksiazki
SET tytul = CONCAT('Wyp. ' , tytul)
WHERE nr_ks IN (SELECT wyp_ks
          FROM Wypozyczona
          WHERE data_z IS NULL);
```

Przykład pokazuje modyfikację relacji *Ksiazki(nr_ks, tytul, autor)*. Modyfikacja ta polega na poprzedzeniu tytułów książek skrótem Wyp., jeśli dana książka jest wypożyczona i nie zwrócona. A zatem warunkiem tego, żeby krotka została zmodyfikowana, jest, aby wartość atrybutu nr_ks (klucz główny) z relacji *Ksiazki* występowała jako wartość atrybutu wyp_ks w relacji *Wypozyczona*, gdzie data_z jest wartością pustą.