Programowanie aplikacji w Java

Maciej Gowin

Zjazd 5 - dzień 1

Linki

Opis

https://maciejgowin.github.io/wsb-java/

Kod źródłowy przykładów oraz zadań

https://github.com/MaciejGowin/wsb-programowanie-aplikacji-java

Zapis danych

Do tej pory w przykładach opieraliśmy się na danych przechowywanych w pamięci programu. Co za tym idzie - po zakończeniu programu dane te były usuwane.

Najprostszym sposobem na przechowanie ich do dalszego użytku jest zapisanie ich do pliku. Działanie to mogłoby składać się z następujących kroków:

- przy uruchomieniu programu wczytaj dane z pliku
- zapisz zmiany przed zakończeniem programu lub podczas każdorazowej zmiany stanu

Programowanie: przykład 51

Odczytywanie oraz zapisywanie danych podczas działania programu.

```
package pl.wbs.programowaniejava.maciejgowin.przyklad51;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws Exception {
        PersonData titleData = new PersonData("/tmp/data");

        Scanner input = new Scanner(System.in);
        while(true) {
            System.out.print("Podaj imię i nazwisko (lub [w]ypisz, [z]akończ): ");
            String value = input.nextLine();

            if (value.equals("w")) {
                System.out.println(titleData.getPersons());
            } else if (value.equals("z")) {
                break;
            } else {
                if (!titleData.addPerson(value)) {
                    System.out.println("Nie poprawny format: " + value);
                }
            }
        }
    }
}
```

Zapis danych

Dane możemy zapisywać w ustalonym przez nas formacie. Java wprowadza mechanizm serializacji pozwalający na zapis bajtowy obiektów do pliku przy użyciu interfejsu `Serializable`.

Przez **serializację** rozumiemy konwersję obiektów do zewnętrznego formatu, który może zostać zapisany. Proces odwrotny nazywamy **deserializacją**.

W naszym przykładzie użyliśmy serializacji do reprezentacji tekstowej zapisanej w pliku tekstowym. Każdy obiekt reprezentowany był przez jedną linię w pliku, gdzie kolejne własności obiektu rozdzielone były przecinkiem.

Zapis danych: CSV

Format danych tekstowych, w którym każda linia definiuje rekord, a wartości rekordu przedzielone są przecinkiem, określamy mianem formatu **CSV (ang. comma-separated values).**

```
firstName,lastName
Jan,Nowak
Andrzej,Kowalski
```

Zapis danych: JSON

Innym popularnym formatem zapisu rekordów jest format **JSON (ang. JavaScript Object Notation),** który pozwala na rozszerzenie zapisu poprzez definicję pełnych zagnieżdżonych własności.

```
[
    {
        "firstName": "Jan",
        "lastName": "Nowak"
    },
    {
        "firstName": "Andrzej",
        "lastName": "Kowalski"
    }
]
```

Baza danych

Baza danych to zbiór danych zapisanych w z góry określonym formacie oraz przy pomocy jasno zdefiniowanych reguł.

Baza danych pozwala na odseparowanie sposobu przechowywania, wyszukiwania oraz ogólnopojętego zarządzania danymi od działania programu.

Zarządzaniem baz danych zajmują się programy DBMS (ang. database management systems).

Baza danych

Bazy danych pozwalają na odseparowanie logiki programu od warstwy przechowywania oraz zarządzania danymi. Program komunikuje się z systemem bazodanowym poprzez jego interfejs pozwalający na wykonywanie prostych operacji typu CRUD, jak i bardziej zaawansowane działania.

Zazwyczaj system DBMS udostępnia interfejs do zarządzania bazą z perspektywy administratora.

CRUD (ang. create-read-update-delete) określenie dla zbioru operacji dodawania, odczytywania, uaktualniania oraz usuwania danych.

Rodzaje baz danych

Do najczęściej spotykanych rodzajów należą relacyjne oraz nie-relacyjne bazy danych.

Relacyjne bazy danych

Opierają się na relacjach (tabelach) oraz powiązaniach pomiędzy nimi. Definiują wewnętrzne języki stanowiące interfejs do operacji na danych oraz ich zarządzania.

Nierelacyjne bazy danych

Opierają się na danych przechowywanych w formie pozbawionej relacji pomiędzy przechowywanymi rekordami. Dane są zapisywane najczęściej w modelu klucz-wartość, gdzie wartość nie ma jednoznacznie zdefiniowanej struktury.

Relacyjne bazy dany

Przy opisie relacyjnych baz danych będziemy używać intuicyjnego opisu zagadnień z pominięciem definicji formalnej.

Relacyjne bazy dany: relacje

W relacyjnych bazach danych do opisu danych używamy relacji. Relacja jest reprezentowana przez tabelę. Każda z relacji (tabel) zawiera:

- unikatową nazwę,
- nagłówek,
- dane.

Relacyjne bazy dany: relacje

Nagłówek definiuje zbiór nazw przechowywanych atrybutów wraz z ich typami - podobnymi do tych znanych z języków programowania. Odnosząc to do tabeli, każdy z atrybutów opisuje kolumnę wraz z jednoznacznym typem przechowywanych elementów.

```
| Id | First name (string) | Last name (string) | Date of birth (date) | Height (integer) | Weight (double) |
+----+---------------------+--------------------+----------------------+------------------+-----------------|
| 1  | Jan                 | Nowak              | 1986                 | 180              | 85.5            |
| 2  | Andrzej             | Kowalski           | 1991                 | 175              | 79.1            |
```

Relacyjne bazy dany: relacje

Dane są definiowane poprzez zbiór krotek, czyli wierszy tabeli. Przypisują one wartości dla danych atrybutów.

Id	First name (string)	Last name (string)	Date of birth (date)	Height (integer)	Weight (double)
1	Jan	Nowak	1986	180	85.5
2	Andrzej	Kowalski	1991	175	79.1

Relacyjne bazy dany: klucz główny

Dane reprezentowane są przez zbiór krotek (wierszy). Z natury zbiór jest nieuporządkowany, przez co kolejność dodawania oraz przechowywania wartości nie jest istotna.

Do identyfikacji krotek używamy **klucza głównego**. Jest on unikalnym identyfikatorem relacji. Definiowany jest na podstawie jednego lub więcej atrybutów.

```
| Id | Name                        | City                  | Established |
+----+-----------------------------+-----------------------+-------------+
| 1  | BETARD SPARTA Wrocław       | Wrocław               | 1950        |
| 2  | Stal Gorzów Wielkopolski    | Gorzów Wielkopolski   | 1947        |
| 3  | FOGO Unii Leszno            | Leszno                | 1938        |
```

Relacyjne bazy dany: klucz obcy

Relacje definiują również **klucz obcy**. Podobnie do klucza głównego jest on opisany przez atrybuty. Ma on na celu opis zależności pomiędzy relacjami oraz nałożenie restrykcji zapewniających spójność danych.

```
| Id | First name | Last name | Team Id |
+----+------------+-----------+---------|
| 1  | Maciej     | Janowski  | 1       |
| 2  | Tai        | Woffinden | 1       |
| 3  | Bartosz    | Zmarzlik  | 2       |
```

Relacyjne bazy dany: operacje

Relacje (tabele) są ze sobą powiązane. Zależności te mogą służyć do przeszukiwania danych. Relacyjne bazy danych wprowadzają również zestawy operacji do zarządzania danymi oraz ich przeszukiwania.

Zdefiniowane są one za pomocą wewnętrznych języków programowania, zwyczajowo **SQL (ang. Structured Query Language)**.

> Przykład zapytań:
> Wyszukaj wszystkich graczy danego zespołu.
> Wyszukaj wszystkich graczy gdzie imię gracza to Andrzej.

Relacyjne bazy dany: pojęcia

Pojęcia znane z teorii relacyjnych baz danych często są zastępowane terminologią znaną z języka SQL. Do tej pory użyliśmy już kilku odniesień, które przytaczane były zamiennie.

SQL	**Relacyjne bazy danych**
tabela (ang. table)	relacja (ang. relation)
wiersz (ang. row)	krotka, rekord (ang. tuple, record)
kolumna (ang. column)	atrybut, pole (ang. attribute, field)

Relacyjne bazy dany: relacja one-to-one

Relacja one-to-one (jeden do jednego) pomiędzy tabelami A i B występuje wtedy, gdy każdemu rekordowi z tabeli A jest przyporządkowany nie więcej niż jeden rekord z tabeli B i na odwrót.

```
| countryId | name    |
+-----------+---------+
| 1         | Poland  |
| 2         | Germany |
```

```
| capitalId | name   | countryId |
+-----------+--------+-----------+
| 1         | Warsaw | 1         |
| 2         | Berlin | 2         |
```

Relacyjne bazy dany: relacja one-to-many

Relacja one-to-many (jeden do wielu) pomiędzy tabelami A i B występuje wtedy, gdy danemu rekordowi z tabeli A przyporządkowanych może być wiele rekordów z tabeli B, natomiast pojedynczemu rekordowi z tabeli B jest przyporządkowany co najwyżej jeden rekord z tabeli A.

```
| countryId | name    |
+-----------+---------+
| 1         | Poland  |
| 2         | Germany |
```

```
| cityId    | name    | countryId |
+-----------+---------+-----------+
| 1         | Warsaw  | 1         |
| 2         | Berlin  | 2         |
| 2         | Wrocław | 1         |
```

Relacyjne bazy dany: relacja many-to-many

Relacja many-to-many (wiele do wielu) pomiędzy tabelami A i B występuje wtedy, gdy danemu rekordowi z tabeli A przyporządkowanych może być wiele rekordów z tabeli B i na odwrót.

Do realizacji relacji wiele do wielu konieczna jest tabela pomocnicza łącząca rekord.

Relacyjne bazy dany: relacja many-to-many

```
| countryId | name    |
+-----------+---------+
| 1         | Poland  |
| 2         | Germany |
```

```
| riverId | name      |
+-----------+---------+
| 1         | Oder    |
| 2         | Vistula |
```

```
| countryId | riverId |
+-----------+---------+
| 1         | 1       |
| 1         | 2       |
| 2         | 1       |
```

Relacyjne bazy dany: implementacje

Istnieje wiele implementacji relacyjnych baz danych. Do najpopularniejszych systemów zarządzania relacyjną bazą danych należą:

- Systemy otwarte
  - MySQL
  - PostgreSQL
  - SQLite
- Systemy zamknięte
  - Microsoft SQL Server
  - Oracle Database
  - IBM DB2

Relacyjne bazy dany: implementacje

Podstawowe koncepcje tych systemów są zbieżne. Zrozumienie działania jednego pozwala na dość łatwe zrozumienie podstaw kolejnego.

Większość z systemów opiera się na standardowym języku zapytań SQL, który jest doprecyzowany dla każdej z implementacji.

Relacyjne bazy danych: modele działania systemów

Większość systemów działa w modelu klient-serwer. W modelu tym system bazodanowy uruchomiony jest na serwerze. Połączenie z systemem odbywa się przy pomocy interfejsu sieciowego i jest inicjalizowane przez klienta.

Przykład: MySQL, PostgreSQL

Istnieją też implementacje działające w modelu opierającym się na wbudowanej bazie danych w program końcowy działający w kontekście lokalnym dla programu klienckiego.

Przykład: SQLite

MySQL

MySQL jest systemem zarządzania bazami danych opartym na otwartym oprogramowaniu.

Dane zorganizowane są w bazach danych, które składają się z jednej lub więcej tabeli. Tabele mogą być połączone relacjami, które pozwalają na strukturyzowanie danych.

MySQL opera się na silniku SQL pozwalającym nie tylko na operacje na danych, ale również na zarządzanie dostępem do danych poprzez mechanizmy autentykacji oraz autoryzacji.

MySQL

MySQL (w wersji otwartej) jest dostępny pod adresem https://dev.mysql.com/downloads/.

Instalacja dostarcza 2 podstawowe programy uruchomieniowe:

- **mysqld**: serwer bazodanowy, który odpowiedzialny jest za zarządzanie plikami danych oraz dostępem do nich.
- **mysql**: klient bazodanowy, który pozwala na połączenie z bazą danych oraz wykonywanie zapytań SQL.

Choć system dostarcza aplikację kliencką pozwalającą na dostęp do baz danych, aplikacje częściej używają bibliotek, które pozwalają na dostęp do baz danych z poziomu programu operującego na nich.

MySQL Workbench

Rozszerzeniem systemu MySQL jest narzędzie MySQL Workbench umożliwiające wizualizację danych, ich modelowanie, konfigurację serwera, zarządzanie użytkownikami i wiele więcej. Wszystkie te operacje dostępne są z poziomu graficznego interfejsu użytkownika (GUI).

MySQL Workbench jest dostępny pod adresem https://www.mysql.com/products/workbench/.

MySQL: podstawy

MySQL pozwala na stworzenie wielu baz danych. W każdym z przypadków baza danych będzie składała się z tabel, relacji między nimi, dodatkowych ustawień oraz zależności.

Istotne jest odpowiednie nazewnictwo używane podczas tworzenia zasobów takich jak: baza danych, tabela czy kolumny. Będziemy kierowali się wskazówkami znanymi z języka Java:

- używamy jedynie liter alfabetu łacińskiego oraz cyfr (przy czym cyfra nie może rozpoczynać identyfikatora),
- nie używamy spacji, zastępujemy ją znakiem podkreślnika,
- system rozpoznaje wielkość znaków,
- nazewnictwo powinno być jednoznaczne, klarowne oraz spójne.

MySQL: podstawy

Przed stworzeniem bazy zwyczajowo opisujemy problem oraz staramy się ją zaprojektować. W dalszej części przejdziemy przez cały proces tworzenia bazy, wprowadzając kolejne pojęcia oraz instrukcje znane z języka SQL, które wspierane są przez MySQL.

MySQL: system uprawnień

MySQL dostarcza również pełny system uprawnień pozwalający na zarządzanie tym, kto ma dostęp do danych zasobów oraz kto może wykonywać dane operacje.

Istnieją 4 poziomy uprawnień:

- globalny: obowiązują dla wszystkich istniejących baz,
- baza danych: obowiązują w kontekście danej bazy danych,
- tabela: obowiązują w kontekście danej tabeli,
- kolumna: obowiązują w kontekście danej kolumny tabeli.

MySQL: system uprawnień

Dodatkowo uprawnienia są pogrupowane w 3 typy:

- użytkowników,
- administratorów,
- specjalne.

Powinniśmy stosować zasada najmniejszego uprzywilejowani (ang. principle of least privilege), przypisując tylko uprawniania konieczne dla danego użytkownika.

MySQL: system uprawnień

Typ	Poziomy	Uprawnienia
użytkownik	tabela, kolumna	SELECT, INSERT, UPDATE
użytkownik	tabela	DELETE, INDEX, ALTER
użytkownik	baza danych, tabela	CREATE, DROP

MySQL: system uprawnień

Typ	Poziomy	Uprawnienia
administrator	-	CRETE TEMPORARY TABLES, FILE, LOCK TABLES, PROCESS, RELOAD, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, SHUTDOWN, SUPER
specjalne	-	ALL, USAGE

MySQL w praktyce: definicja problemu

Stwórz bazę reprezentującą dane przechowywane przez prosty system rezerwacji lotów pozwalająca na przechowywanie:

- klientów posiadających imię, nazwisko, data urodzenia (opcjonalne),
- lotów definiujących: lotnisko wylotu oraz przylotu, data i czas wylotu oraz przylotu,
- rezerwacji dla klienta na danych lot wraz z ceną,
- lotnisk opisujących nazwę lotniska oraz jego położenie (szerokość oraz długość geograficzna).

MySQL w praktyce: diagram tabel oraz relacji

Tabela	Atrybuty
customers	id (int), first_name (string), last_name (string), date_of_birth (date)
flights	id (int), departure_airport_code (string), arrival_airport_code (string), departure_date_time (datetime), arrival_date_time (datetime)
bookings	id, customer_id (int), flight_id (int), price (double)
airports	code (string), name (string), latitude (double), longitude (double)

MySQL w praktyce: tworzenie/usuwanie bazy danych

Tworzenie bazy danych:

```
CREATE DATABASE booking_system DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Usuwanie bazy danych:

```
DROP DATABASE booking_system;
```

MySQL w praktyce: tworzenie użytkownika

Stworzenie użytkownika, który ma dostęp tylko do wcześniej utworzonej bazy danych.

```
CREATE USER 'booking' IDENTIFIED WITH mysql_native_password BY 'pass';
```

Aktualizacja użytkownika poprzez zmianę jego hasła:

```
ALTER USER 'booking' IDENTIFIED WITH mysql_native_password BY 'pass123!';
```

Dodanie wszystkich uprawnień do wcześniej utworzonej bazy.

```
GRANT ALL
ON booking_system.*
TO 'booking';
```

MySQL w praktyce: tworzenie użytkownika

Aby odebrać dostęp, moglibyśmy użyć polecenia:

```
REVOKE ALL
ON booking_system.*
FROM 'booking';
```

A następnie dodać konkretne uprawnienia:

```
GRANT INSERT, UPDATE, DELETE, SELECT
ON booking_system.*
TO 'booking';
```

MySQL w praktyce: tworzenie tabel

W pierwszej kolejności musimy poinstruować system, że będziemy pracować z konkretną bazą danych:

```
USE booking_system;
```

MySQL w praktyce: tworzenie tabel

Aby utworzyć poprzednio opisane tabele, użyjemy poleceń:

```
CREATE TABLE customers
(
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    date_of_birth DATE
);
```

```
CREATE TABLE flights
(
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    departure_airport_code VARCHAR(3) NOT NULL,
    arrival_airport_code VARCHAR(3) NOT NULL,
    departure_date_time DATETIME NOT NULL,
    arrival_date_time DATETIME NOT NULL
);
```

MySQL w praktyce: tworzenie tabel

Aby utworzyć poprzednio opisane tabele, użyjemy poleceń:

```
CREATE TABLE bookings
(
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    customer_id INT UNSIGNED NOT NULL,
    flight_id INT UNSIGNED NOT NULL,
    price DECIMAL(10,2) NOT NULL DEFAULT 0.0
);
```

```
CREATE TABLE airports
(
    code VARCHAR(3) NOT NULL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    latitude DOUBLE NOT NULL,
    longitude DOUBLE NOT NULL
);
```

MySQL w praktyce: tworzenie tabel

Aby sprawdzić wszystkie istniejące tabele:

```
SHOW TABLES;
```

Aby sprawdzić definicję danej tabeli:

```
DESCRIBE customers;
```

MySQL w praktyce: modyfikacja tabel

Na podstawie istniejącej tabeli możemy zmienić definicję istniejącej już kolumny.

```
ALTER TABLE airports
    MODIFY name VARCHAR(200);
```

Możemy też dodać nową kolumnę do tabeli.

```
ALTER TABLE airports
    ADD country VARCHAR(2) AFTER name;
```

A następnie usunąć kolumnę.

```
ALTER TABLE airports
    DROP country;
```

MySQL w praktyce: usuwanie tabel

Aby usunąć tabelę:

```
DROP TABLE customers;
```

MySQL: atrybuty kolumn

Przy definicji kolumn użyte zostały dodatkowe atrybuty:

NOT NULL

Podczas dodawania nowego wiersza dana wartość dla atrybutu nie może być pusta, w przypadku próby dodania pustej wartości zostanie zwrócony błąd. Domyślnie dopuszczalne są wartości **NULL**.

DEFAULT

W przypadku braku zdefiniowanej wartości dla atrybutu, podczas dodawania nowego wiersza użyta zostanie wartość domyślna.

MySQL: atrybuty kolumn

AUTO_INCREMENT

Instrukcja automatycznego generowania klucza działająca na liczbach całkowitych. Działa na kolumnach będących kolumnami indeksowanymi. Kolumna będąca częścią klucza jest automatycznie indeksowana.

Atrybut ten powoduje automatyczne wygenerowanie nowego klucza dla nowego elementu, który będzie wartością klucza poprzedniego powiększoną o 1.

> Jeżeli do tabeli zostaną dodane wiersze o identyfikatorach 1, 2, 3, a następnie usuniemy element 3, kolejnym identyfikatorem będzie dalej 4.

> Jeżeli do tabeli zostanie dodany wiersz z explicite zdefiniowanym indeksem, kolejny identyfikator będzie wygenerowany na podstawie największego dotychczasowego identyfikatora.

MySQL w praktyce: atrybuty kolumn

PRIMARY KEY

Oznaczenie klucza głównego dla tabeli, dla których wartości muszą być unikalne, dzięki czemu możemy odwołać się do konkretnego wiersza. Jeżeli klucz główny składa się tylko z jednej kolumny, może zostać zdefiniowany na poziomie kolumny:

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
```

W przypadku kluczy złożonych używamy zapisu, który odseparowuje definicję klucza od kolumny:

```
PRIMARY KEY(customer_id, booking_id)
```

MySQL: typy danych kolumn

Typy danych przechowywanych są podobne do tych znanych z języków programowania, w tym Java. Rozszerzają one jednak podstawowe typy o ich wersje o z góry zdefiniowanej zajmowanej przestrzeni. Związane jest do z optymalizacją przestrzeni przechowywania danych. Trzeba jednak pamiętać, że typ o danej pojemności nie będzie w stanie przechować informacji wykraczającej poza jego zakres.

Przedstawione zostaną najczęściej używane typy.

MySQL: typy bitowe i numeryczne

Typ	Opis
BIT(M)	wartość bitowa, M opcjonalne
TINYINT(M) (UNSIGNED)	bardzo mała wartość całkowita z zakresu (-128, 127) lub (0, 255)
BOOL, BOOLEAN	synonim dla TINYINT(1)
SMALLINT(M) (UNSIGNED)	mała wartość całkowita z zakresu (-32768, 32767) lub (0, 65535)
MEDIUMINT(M) (UNSIGNED)	średnia wartość całkowita z zakresu (-8388608, 8388607) lub (0, 16777215)

MySQL: typy bitowe i numeryczne całkowite

Typ	Opis
INT(M) (UNSIGNED)	wartość całkowita z zakresu (-2147483648, 2147483647) lub (0, 4294967295)
INTEGER(M) (UNSIGNED)	synonim dla INT
BIGINT(M) (UNSIGNED)	wartość całkowita z zakresu (-9223372036854775808, 9223372036854775807) lub (0, 18446744073709551615)
SERIAL	synonim dla BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE

MySQL: typy numeryczne stałoprzecinkowe

Typ	Opis
DECIMAL(M, D) (UNSIGNED)	wartość stałoprzecinknowa z M cyframi (maksymalnie 65) przed przecinkiem oraz D cyframi po przecinku (maksymalnie 30)
DEC(M, D) (UNSIGNED)	synonim dla DECIMAL
NUMERIC(M, D) (UNSIGNED)	synonim dla DECIMAL
FIXED(M, D) (UNSIGNED)	synonim dla DECIMAL

MySQL: typy numeryczne zmiennoprzecinkowe

Typ	Opis
FLOAT(M, D) (UNSIGNED)	liczba zmiennoprzecinkowa pojedynczej precyzji, gdzie M to ilość cyfr, a D to maksymalna ilość cyfr po przecinki.
FLOAT(p) (UNSIGNED)	liczna zmiennoprzecinkowa, gdzie p reprezentuje precyzję w bitach
DOUBLE(M, D) (UNSIGNED)	liczba zmiennoprzecinkowa podwójnej precyzji, gdzie M to ilość cyfr, a D to maksymalna ilość cyfr po przecinki.

MySQL: typy ciągów znaków

Typ	Opis
CHAR (M)	ciąg znaków o stałej długości M
VARCHAR(M)	ciąg znaków o długości maksymalnej M
TEXT	ciąg znaków
BLOB	ciąg binarny

MySQL: typy daty i czasu

Typ	Opis
DATE	data w formacie `2021-01-01`
TIME	czas w formacie `23:59:59`
DATETIME	data z czasem w formacie '2021-01-01 23:59:59'
TIMESTAMP	data z czasem w formacie '2021-01-01 23:59:59', z automatyczną konwersją do UTC z obecnej strefy czasowej
YEAR	rok w formacie '2021'

MySQL w praktyce: dodawanie wierszy

Aby dodać wiersz do danej tabeli, definiując wszystkie kolumny, użyjemy polecenia:

```
INSERT INTO customers VALUES
    (NULL, 'Andrzej', 'Kowalski', '1987-11-14');
```

Możemy też dodać nowy wiersz z definicją wybranych kolumn:

```
INSERT INTO customers(first_name, last_name, date_of_birth) VALUES
    ('Jan', 'Nowak', '1952-01-12');
```

Lub explicite ustawionymi wartościami dla kolumn:

```
INSERT INTO customers SET
        first_name = 'Karolina',
        last_name = 'Mróz',
        date_of_birth = '1989-03-01';
```

MySQL w praktyce: dodawanie wierszy

Istnieje też opcja dodawania wielu wierszy jednorazowo.

```
INSERT INTO customers(first_name, last_name, date_of_birth) VALUES
    ('Maciej', 'Krab', '1976-07-08'),
    ('Anna', 'Rembiszewska', '1991-12-01'),
    ('Grażyna', 'Łokietek', '1966-05-09');
```

MySQL w praktyce: proste wyszukiwanie wierszy

Aby wyszukać wszystkie wiersze w tabeli, użyjemy polecenia:

```
SELECT * FROM customers;
```

MySQL w praktyce (zadanie): dodaj nowe rekordy do bazy danych

```
Klient 1: Andrzej Kowalski urodzony 1987-11-14
Klient 2: Jan Nowak urodzony 1952-01-12
Klient 3: Karolina Mróz urodzony 1989-03-01
Klient 4: Maciej Krab, urodzony 1976-07-08
Klient 5: Anna Rembiszewska urodzony 1991-12-01
Klient 6: Grażyna Łokietek urodzony 1966-05-09

Lotnisko WRO: Wrocław położony (1.1, 11.11)
Lotnisko DUB: Dublin położony (2.2, 22.22)

Lot 1: z WRO do DUB pomiędzy 2021-01-01 15:00:00 a 2021-01-01 18:00:00
Lot 2: z WRO do GDN pomiędzy 2021-01-01 10:00:00 a 2021-01-01 11:30:00
Lot 3: z GDN do WRO pomiędzy 2021-01-02 21:00:00 a 2021-01-02 23:30:00

Rezerwacja 1: klient 1 zamówił lot 1 w cenie 6.60
Rezerwacja 2: klient 1 zamówił lot 2 w cenie 17.00
Rezerwacja 3: klient 2 zamówił lot 1 w cenie 18.90
Rezerwacja 4: klient 3 zamówił lot 1 w cenie 24.70
Rezerwacja 5: klient 3 zamówił lot 2 w cenie 30.20
Rezerwacja 6: klient 1 zamówił lot 1 w cenie 12.30
```

MySQL w praktyce (zadanie): dodaj nowe rekordy do bazy danych

```
INSERT INTO `customers` VALUES
    (1, 'Andrzej', 'Kowalski', '1987-11-14'),
    (2, 'Jan', 'Nowak', '1952-01-12'),
    (3, 'Karolina', 'Mróz', '1989-03-01'),
    (4, 'Maciej', 'Krab', '1976-07-08'),
    (5, 'Anna', 'Rembiszewska', '1991-12-01'),
    (6, 'Grażyna', 'Łokietek', '1966-05-09');

INSERT INTO `airports` VALUES
    ('DUB', 'Dublin', 2.2, 22.22),
    ('WRO', 'Wroclaw', 1.1, 11.11);

INSERT INTO `flights` VALUES
    (1, 'WRO', 'DUB', '2021-01-01 15:00:00', '2021-01-01 18:00:00'),
    (2, 'WRO', 'GDN', '2021-01-01 10:00:00', '2021-01-01 11:30:00'),
    (3, 'GDN', 'WRO', '2021-01-02 21:00:00', '2021-01-02 23:30:00');

INSERT INTO `bookings` VALUES
        (1, 1, 1, 6.60),
        (2, 1, 2, 17.00),
        (3, 2, 1, 18.90),
        (4, 3, 1, 24.70),
        (5, 3, 2, 30.20),
        (6, 1, 1, 12.30);
```

MySQL w praktyce: wyszukiwanie

Poznaliśmy proste wyszukiwanie wierszy:

```
SELECT * FROM customers;
```

Możemy ograniczyć kolumny zwrócone:

```
SELECT last_name FROM customers;
```

MySQL w praktyce: wyszukiwanie z kryteriami

Wyszukiwanie możemy ograniczyć według kryteriów wyszukiwanie stosując klauzulę `WHERE` wraz z operatorami porównania:

```
SELECT * FROM flights WHERE id = 3;
SELECT * FROM flights WHERE departure_airport_code = 'WRO';
SELECT * FROM flights WHERE id > 1;
```

Możemy również użyć operatorów logicznych takich jak `AND` czy `OR`:

```
SELECT * FROM flights WHERE departure_airport_code = 'WRO' AND arrival_airport_code = 'DUB';
SELECT * FROM flights WHERE departure_airport_code = 'WRO' OR departure_airport_code = 'GDN';
```

MySQL: operatory porównania

Operator	Przykład	Opis
=	id = 1	wartość równa
>	id > 1	wartość większa
>=	id >= 1	wartość większa lub równa
<	id < 1	wartość mniejsza
<=	id <= 1	wartość mniejsza lub równa
!=, <>	id != 1	wartość nie równa

MySQL: operatory porównania

Operator	Przykład	Opis
IS NOT NULL	date_of_birth IS NOT NULL	wartość nie jest pusta
IS NULL	date_of_birth IS NULL	wartość jest pusta
BETWEEN	price BETWEEN 10.00 AND 20.00	wartość pomiędzy
IN	code IN ('WRO', 'DUB')	wartość w zbiorze wartości
NOT IN	code NOT IN ('WRO', 'DUB')	wartość poza zbiorem wartości

MySQL: operatory porównania

Operator	Przykład	Opis
LIKE	last_name LIKE 'Kow%'	wartość pasująca do wzorca
NOT LIKE	last_name NOT LIKE 'Kow%'	wartość nie pasująca do wzorca
REGEXP	last_name REGEXP '^Kow.*'	wartość pasująca do wzorca

MySQL w praktyce: operatory porównania

```
SELECT * FROM customers WHERE first_name = 'Karolina';
SELECT * FROM bookings WHERE price > 18.90;
SELECT * FROM bookings WHERE price >= 18.90;
SELECT * FROM bookings WHERE price < 18.90;
SELECT * FROM bookings WHERE price <= 18.90;
SELECT * FROM customers WHERE first_name != 'Karolina';
SELECT * FROM customers WHERE date_of_birth IS NOT NULL;
SELECT * FROM customers WHERE date_of_birth IS NULL;
SELECT * FROM bookings WHERE price BETWEEN 10.00 AND 20.00;
SELECT * FROM flights WHERE departure_airport_code IN ('DUB', 'GDN');
SELECT * FROM flights WHERE departure_airport_code NOT IN ('GDN');
SELECT * FROM customers WHERE last_name LIKE 'K%';
SELECT * FROM customers WHERE last_name NOT LIKE 'K%';
SELECT * FROM customers WHERE last_name REGEXP '^K.*';
```

MySQL w praktyce: wyszukiwanie z sortowaniem

Wyniki wyszukiwania mogą zostać posortowane za pomocą `ORDER BY`:

```
SELECT * FROM bookings ORDER BY price;
```

Automatycznie zostało użyte sortowanie naturalne. Możemy nim sterować za pomocą słów kluczowych `ASC` oraz `DESC`.

```
SELECT * FROM bookings ORDER BY price ASC;
SELECT * FROM bookings ORDER BY price DESC;
```

MySQL w praktyce: wyszukiwanie z limitem

Dla dużych tabel możemy użyć limitu zwróconych wierszy. Pozwala to na stronicowanie wyników i odbywa się przy użyciu `LIMIT`.

```
SELECT * FROM customers ORDER BY id LIMIT 3;
```

Możemy też przesunąć początek wyszukiwania o daną wartość.

```
SELECT * FROM customers ORDER BY id LIMIT 2, 3;
```

MySQL: wyszukiwanie z grupowaniem

MySQL dostarcza szereg funkcji wbudowanych w tym funkcje grupujące pozwalające na scalanie wyników takie jak: zliczanie elementów czy obliczanie średniej wartości dla elementów.

Metody te często występują z klauzulą `GROUP BY` pozwalającą na zdefiniowanie kolumny, po której następuje grupowania. Jeżeli chcemy dodatkowo ograniczyć wyniki na podstawie sprawdzenia wartości grupowania, używamy klauzuli `HAVING`.

MySQL: wyszukiwanie z grupowaniem

Funkcja	Opis
COUNT(*)	liczba wierszy
COUNT(k)	liczba wierszy, gdzie wartość kolumny k jest niepusta
COUNT(DISTINCT kolumna)	liczba unikatowych wartości w kolumnie k
SUM(k)	suma wartości dla kolumny k
AVG(k)	średnia arytmetyczna dla kolumny k
MIN(k)	minimalna wartość w kolumnie k
MAX(k)	maksymalna wartość w kolumnie k

MySQL w praktyce: wyszukiwanie z grupowaniem

Aby policzyć ilość klientów:

```
SELECT COUNT(*) FROM customers;
```

Aby sprawdzić sumaryczną cenę wszystkich rezerwacji:

```
SELECT SUM(price) FROM bookings;
```

Aby sprawdzić największą cenę rezerwacji:

```
SELECT MAX(price) FROM bookings;
```

MySQL w praktyce: wyszukiwanie z grupowaniem

Aby policzyć ilość zamówień dla danego lotu, użyjemy grupowania po numerze lotu przy użyciu klauzuli `GROUP BY`.

```
SELECT flight_id, COUNT(*) FROM bookings
    GROUP BY flight_id;
```

Grupowanie możemy rozszerzyć, o sprawdzenie ile rezerwacji dokonał konkretny klient dla każdego z lotów.

```
SELECT customer_id, flight_id, COUNT(*) FROM bookings
    GROUP BY customer_id, flight_id;
```

MySQL w praktyce: wyszukiwanie z grupowaniem

Aby sprawdzić jaką sumaryczną cenę zapłacił każdy z klientów:

```
SELECT customer_id, SUM(price) FROM bookings
    GROUP BY customer_id;
```

Jeżeli teraz chcielibyśmy pobrać klientów, których wydatki przekroczyły daną kwotę użyjemy klauzuli `HAVING`.

```
SELECT customer_id, SUM(price) FROM bookings
    GROUP BY customer_id
    HAVING SUM(price) > 20;
```

Aby uprościć zapis, możemy użyć aliasu dla wartości przy użyciu klauzuli `AS`.

```
SELECT customer_id, SUM(price) AS sum_per_customer FROM bookings
    GROUP BY customer_id
    HAVING sum_per_customer > 20;
```

MySQL w praktyce: wyszukiwanie z aliasami

Do poprawienia czytelności oraz w przypadku kolizji nazw możemy używać aliasów przy użyciu klauzuli `AS`.

```
SELECT date_of_birth AS dob FROM customers;
```

Mogą być też użyte na poziomie tabel.

```
SELECT * FROM customers AS c WHERE c.first_name = 'Karolina';
```

MySQL w praktyce: aktualizacja wierszy

Aktualizacja wierszy odbywa się przy pomocy klauzuli `UPDATE`. Działa ona w połączeniu z klauzulami `WHERE`, `LIMIT` oraz `ORDER BY` choć w praktyce spotykane jest głównie ograniczenie warunkowe:

```
UPDATE customers
    SET first_name = 'Maciej'
    WHERE id = 1;
```

Możemy też zmienić wartości dla wszystkich wierszy bez ograniczeń.

```
UPDATE bookings
    SET price = price * 1.1;
```

MySQL w praktyce: usuwanie wierszy

Usuwanie wierszy odbywa się przy użyciu klauzuli `DELETE`. Działa ona w połączeniu z klauzulami `WHERE`, `LIMIT` oraz `ORDER BY` choć w praktyce spotykane jest głównie ograniczenie warunkowe:

```
DELETE FROM airports WHERE code = 'GDN';
```