Objektovo orientované programovanie

Učiteľ:
Ing. Jozef Wagner, PhD.

Učebnica:
https://oop.wagjo.com/

OPG
Teória 27

1. Použitie databáz v Jave
2. JDBC
3. SQLite

Prečo databázy v Jave?

Databázy patria medzi najdôležitejšie technológie pri vývoji moderných aplikácií. Umožňujú trvalé uchovávanie údajov, ich efektívne vyhľadávanie, úpravu a správu

- Dlhodobé uloženie dát (aj po reštarte aplikácie)
- Súbežný prístup viacerých používateľov/procesov
- ACID vlastnosti
- Výkonné dotazovanie a reporting

ACID

- **Atomicita** - Viaceré zmeny v databáza sa vykonajú ako jedna tranzakcia, buď sa vykonajú všetky, alebo ani jedna
- **Konzistencia** - Zmeny v databáze neporušia databázový invariant, napr. nepovolenie duplicitných hodnôt
- **Izolácia** - Pri súčasnom prístupe k databáze od viacerých užívateľov títo nebudú nikdy vidieť nedokončené zmeny od ostatných
- **Durability** - Po úspešnom prevedení zmeny sa táto zmena zachová aj pri reštarte aleo chybe systému

Relačné databázy

- Údaje sú organizované do tabuliek pozostávajúcich z riadkov a stĺpcov.
- Každý riadok predstavuje jeden záznam a každý stĺpec určitý atribút záznamu.
- Medzi najznámejšie relačné databázy patria: PostgreSQL, SQLite, MySQL / MariaDB, Oracle Database a Microsoft SQL Server

id	nazov	autor	rok
1	Java Programovanie	Ján Novák	2023
2	Databázy	Peter Horváth	2022

SQL

Vytvorenie tabuľky

```
CREATE TABLE kniha (
    id INTEGER PRIMARY KEY,
    nazov TEXT,
    autor TEXT,
    rok INTEGER
)
```

SQL

Vloženie údajov

```
INSERT INTO kniha(nazov, autor, rok)
VALUES ('Java Programovanie', 'Ján Novák', 2023);
```

Čítanie údajov

```
SELECT * FROM kniha;
```

SQL

Aktualizácia údajov

```
UPDATE kniha
SET rok = 2024
WHERE id = 1;
```

Odstránenie údajov

```
DELETE FROM kniha WHERE id = 1;
```

Spôsoby prístupu k databázam

- **JDBC** - nízkoúrovňový, plná kontrola nad SQL
- **ORM** (Object-Relational Mapping) - automatické mapovanie Java objektov na tabuľky
- Vyššie abstrakcie - Spring Data JPA, jOOQ, Querydsl (nad ORM)

JDBC

- Predstavuje vrstvu medzi Java aplikáciou a databázovým serverom, vďaka ktorej môže programátor používať rovnaké rozhranie pre rôzne databázové systémy.
- JDBC driver je teda knižnica implementujúca JDBC rozhranie pre konkrétnu databázu. O pripojenie ku konkrétnej databáza sa v JDBC stará trieda `DriverManager`.

Trieda `Connection`

Objekt triedy `Connection` reprezentuje otvorené spojenie s databázou.

Príklad vytvorenia spojenia so SQLite:

```
Connection conn =
    DriverManager.getConnection("jdbc:sqlite:knihy.db");
```

Po úspešnom vytvorení spojenia môže aplikácia vykonávať SQL príkazy.

Trieda `Statement`

Objekt triedy `Statement` slúži na vykonávanie jednoduchých SQL príkazov.

```
Statement stmt = conn.createStatement();
```

```
ResultSet rs =
    stmt.executeQuery("SELECT * FROM kniha");
```

Používa sa najmä v jednoduchých príkladoch a pri SQL príkazoch bez parametrov.

Trieda `PreparedStatement`

`PreparedStatement` je rozšírením triedy `Statement`. Výhody:

- vyššia bezpečnosť,
- ochrana proti SQL Injection (little bobby tables),
- lepší výkon pri opakovanom vykonávaní,
- automatické ošetrenie (escapovanie) špeciálnych znakov.

Trieda `PreparedStatement`

```
String sql = "SELECT * FROM kniha WHERE id = ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, 5);
ResultSet rs = pstmt.executeQuery();
```

Vo väčšine reálnych aplikácií sa odporúča používať práve `PreparedStatement`.

Trieda `ResultSet`

Objekt triedy ResultSet obsahuje výsledok SQL dotazu.

```
while (rs.next()) {
    int id = rs.getInt("id");
    String nazov = rs.getString("nazov");

    System.out.println(id + " " + nazov);
}
```

SQLite

Populárna embedovaná relačná databáza. Nepotrebujeme žiaden samostatný server, ale databáza beží priamo v našom programe.

Celá databáza je jeden súbor na disku.

Oblasti použitia: Desktopové aplikácie, Mobilné aplikácie, IoT / embedded systémy

Nie je vhodná v týchto prípadoch: Vysoká konkurencia zápisov, veľké dáta a komplexná analytika alebo Distribuované systémy s viacerými nodmi