



Databázové systémy

# Normální formy, referenční integrita, návrh databáze, interakce s ostatními jazyky

Petr Krajča



Katedra informatiky  
Univerzita Palackého v Olomouci

SCHOOL	DEAN	DEPT	HEAD	ID	COURSE	YEAR
SCI	Adams	AF	Black	7	QOPT1	2012
SCI	Adams	AF	Black	8	LASR1	2012
SCI	Adams	AF	Black	8	LASR1	2013
SCI	Adams	CS	Davis	3	ALMA1	2012
SCI	Adams	CS	Davis	3	ALMA1	2013
SCI	Adams	CS	Davis	6	DATA1	2012
SCI	Adams	CS	Davis	6	DATA1	2013
SCI	Adams	CS	Davis	6	PAPR1	2012

## Problémy

- **redundance dat** (zbytečná duplikace hodnot)
- **anomálie spojená s výmazem dat** (výmaz kurzů katedry „odstraní vedoucího“)
- **anomálie spojená s aktualizací hodnot** (změna vedoucího katedry na víc místech)



*A table is in first normal form (1NF)—equivalently, such a table is normalized—if and only if it's a direct and faithful representation of some relation. — C.J.Date*

## První normální forma (1NF)

Tabulka je v první normální formě, pokud splňuje všechny následující podmínky:

- 1 neuvažuje se **žádné uspořádání řádků** (shora-dolů)
- 2 neuvažuje se **žádné uspořádání sloupců** (zleva-doprava)
- 3 v tabulce se nevyskytují **duplicitní řádky**
- 4 v každém vnitřní poli tabulky je **právě jedna hodnota daného typu**
- 5 všechny atributy jsou **regulární** (neobsahují žádnou skrytou informaci, která je dostupná pouze prostřednictvím speciálních funkcí)



Mějme relační schéma  $R$ , teorii  $\Gamma$  a množinu klíčů  $K_1, \dots, K_n$  nad  $R$  vzhledem k  $\Gamma$ . Pak relačním schéma  $R$  je ve **druhé normální formě** (2.NF) vzhledem k  $\Gamma$ , jestliže

1 je v 1. NF,

2 pokud pro každý atribut  $y \in R$  takový, že  $y \notin K_i$  pro všechna  $i = 1, \dots, n$ , platí

$$\Gamma \not\models K \Rightarrow \{y\},$$

pro všechny  $K \subset K_i$ , kde  $i = 1, \dots, n$ .

### Poznámka

- Jinými slovy: Žádný atribut, který není součástí klíče, není závislý na části některého klíče.
- Jinak: Každý atribut, který není součástí klíče, je závislý na všech klíčích.

INVOICE	DATE	ITEM	PRICE
2014001	2014-01-30	Apples	123
2014001	2014-01-30	Bananas	100
2014001	2014-01-30	Coconut	345
2014002	2014-02-20	Apples	300
2014002	2014-02-20	Bananas	500

## Funkční závislosti

- $\{INVOICE\} \Rightarrow \{DATE\}$
- $\{INVOICE, ITEM\} \Rightarrow \{PRICE\}$

INVOICE	DATE
2014001	2014-01-30
2014002	2014-02-20

⊗

INVOICE	ITEM	PRICE
2014001	Apples	123
2014001	Bananas	100
2014001	Coconut	345
2014002	Apples	300
2014002	Bananas	500



Mějme relační schéma  $R$  a teorii  $\Gamma$ . Pak  $R$  je ve **třetí normální formě** vzhledem k  $\Gamma$ , pokud  $R$  je v 2.NF vzhledem k  $\Gamma$  a žádný neklíčový atribut  $y \in R$  není tranzitivně závislý na nějakém nadklíči relačního schématu  $R$ .

Funkční závislost  $A \Rightarrow C$  v teorii  $\Gamma$  je tranzitivní, pokud existuje  $B$  takové, že  $\Gamma \models A \Rightarrow B$  a  $\Gamma \models B \Rightarrow C$ , přičemž  $\Gamma \not\models B \Rightarrow A$ .

Jinými slovy: Atribut  $C$  je tranzitivně závislý na  $A$ , pokud existuje  $B$  takové, že  $A \Rightarrow B$  a  $B \Rightarrow C$ .

#### Poznámky:

- Neklíčové atributy závisí pouze na klíči.
- Možná interpretace: Každý neklíčový atribut musí reprezentovat pouze fakt o klíči, celém klíči a ničem jiném.

ITEM	DEPT_ID	DEPT_NAME	PRICE
1	1	books	123
2	2	clothes	100
3	1	books	345
4	1	books	300
5	2	clothes	500

## Funkční závislosti

- $\{ITEM\} \Rightarrow \{DEPT\_ID\}$
- $\{DEPT\_ID\} \Rightarrow \{DEPT\_NAME\}$
- tranzitivní závislost  $\{ITEM\} \Rightarrow \{DEPT\_ID\} \Rightarrow \{DEPT\_NAME\}$

ITEM	DEPT_ID	PRICE
1	1	123
2	2	100
3	1	345
4	1	300
5	2	500



DEPT_ID	DEPT_NAME
1	books
2	clothes



- Uvažujme relační schéma  $\{ICO\_DODAVATELE, NAZEV\_DODAVATELE, PSC, MESTO\}$
- s funkčními závislostmi:  $\{ICO\_DODAVATELE\} \Rightarrow \{NAZEV\_DODAVATELE\}$ ,  
 $\{ICO\_DODAVATELE\} \Rightarrow \{MESTO\}$ ,  $\{ICO\_DODAVATELE\} \Rightarrow \{PSC\}$ ,  
 $\{NAZEV\_DODAVATELE\} \Rightarrow \{ICO\_DODAVATELE\}$ ,  $\{NAZEV\_DODAVATELE\} \Rightarrow \{MESTO\}$ ,  
 $\{NAZEV\_DODAVATELE\} \Rightarrow \{PSC\}$ ,
- kandidátními klíči jsou  $\{ICO\_DODAVATELE\}$ ,  $\{NAZEV\_DODAVATELE\}$
- problém: odstraněním řádků s konkrétním dodavatelem ztratíme informaci o jeho ICO a městě
- možná dekompozice  $\{ICO\_DODAVATELE, NAZEV\_DODAVATELE\}$  a  $\{ICO\_DODAVATELE, PSC, MESTO\}$



Mějme relační schéma  $R$  a teorii  $\Gamma$ . Pak  $R$  je v BCNF vzhledem k  $\Gamma$ , pokud pro každou netriviální  $A \Rightarrow B \in \Gamma$  platí, že  $\Gamma \models A \Rightarrow R$ .

## Poznámka:

- Možná interpretace: Atribut musí reprezentovat pouze fakt o klíči, celém klíči, a ničem jiném.

## Normalizace pomocí dekompozice

Pokud není  $R$  v BCNF vzhledem k  $\Gamma$ , pak

- 1 vezmeme netriviální  $A \Rightarrow B \in \Gamma$  takovou, že  $\Gamma \not\models A \Rightarrow R$
- 2 položíme  $R_1 = A \cup B$  a  $\Gamma_1 = \{C \cap R_1 \Rightarrow D \cap R_1 \mid C \cap R_1 \Rightarrow D \in \Gamma\}$
- 3 položíme  $R_2 = A \cup (R \setminus B)$  a  $\Gamma_2 = \{C \cap R_2 \Rightarrow D \cap R_2 \mid C \cap R_2 \Rightarrow D \in \Gamma\}$
- 4 proces se pokusíme opakovat pro  $R_1$  a  $\Gamma_1$ , pokud není v BCNF a analogicky pro  $R_2$  a  $\Gamma_2$

## Poznámka:

- každé schéma ve 3NF je zároveň v BCNF; obrácené tvrzení neplatí
- BCNF nemusí být dosažitelná

Uvažujme relační schéma:

$$R = \{\text{SCHOOL, DEAN, DEPT, HEAD, ID, COURSE, YEAR}\}$$

a teorii  $\Gamma$  popisující závislosti mezi atributy:

$$\begin{aligned}\Gamma = \{ & \{\text{DEPT}\} \Rightarrow \{\text{HEAD, SCHOOL, DEAN}\}, \\ & \{\text{SCHOOL}\} \Rightarrow \{\text{DEAN}\}, \\ & \{\text{COURSE, YEAR}\} \Rightarrow \{\text{ID}\}, \\ & \{\text{ID, YEAR}\} \Rightarrow \{\text{DEPT}\} \}.\end{aligned}$$

Schéma  $R$  není v BCNF vzhledem k  $\Gamma$ , protože (například):

- $\{\text{DEPT}\} \Rightarrow \{\text{HEAD, SCHOOL, DEAN}\} \in \Gamma$ , ale  $[\{\text{DEPT}\}]_{\Gamma} = \{\text{SCHOOL, DEAN, DEPT, HEAD}\} \neq R$ , to jest  $\Gamma \not\models \{\text{DEPT}\} \Rightarrow R$ , nebo:
- $\{\text{SCHOOL}\} \Rightarrow \{\text{DEAN}\} \in \Gamma$ , ale  $[\{\text{SCHOOL}\}]_{\Gamma} = \{\text{SCHOOL, DEAN}\} \neq R$ , to jest  $\Gamma \not\models \{\text{SCHOOL}\} \Rightarrow R$ .

$R = \{\text{SCHOOL, DEAN, DEPT, HEAD, ID, COURSE, YEAR}\}$

$\Gamma = \{\{\text{DEPT}\} \Rightarrow \{\text{HEAD, SCHOOL, DEAN}\}, \dots\}$  (viz předchozí příklad)

■  $R_1 = \{\text{SCHOOL, DEAN, DEPT, HEAD}\}$

$\Gamma_1 = \{\{\text{DEPT}\} \Rightarrow \{\text{HEAD, SCHOOL, DEAN}\}, \{\text{SCHOOL}\} \Rightarrow \{\text{DEAN}\}\}$

■  $R_{11} = \{\text{SCHOOL, DEAN}\}$

$\Gamma_{11} = \{\{\text{SCHOOL}\} \Rightarrow \{\text{DEAN}\}\}$

■  $R_{12} = \{\text{SCHOOL, DEPT, HEAD}\}$

$\Gamma_{12} = \{\{\text{DEPT}\} \Rightarrow \{\text{HEAD, SCHOOL}\}, \{\text{SCHOOL}\} \Rightarrow \{\}\}$

■  $R_2 = \{\text{DEPT, ID, COURSE, YEAR}\}$

$\Gamma_2 = \{\{\text{DEPT}\} \Rightarrow \{\}, \{\text{COURSE, YEAR}\} \Rightarrow \{\text{ID}\}, \{\text{ID, YEAR}\} \Rightarrow \{\text{DEPT}\}\}$

■  $R_{21} = \{\text{DEPT, ID, YEAR}\}$

$\Gamma_{21} = \{\{\text{DEPT}\} \Rightarrow \{\}, \{\text{ID, YEAR}\} \Rightarrow \{\text{DEPT}\}\}$

■  $R_{22} = \{\text{ID, COURSE, YEAR}\}$

$\Gamma_{22} = \{\{\text{COURSE, YEAR}\} \Rightarrow \{\text{ID}\}, \{\text{ID, YEAR}\} \Rightarrow \{\}\}$

SCHOOL	DEAN	DEPT	HEAD	ID	COURSE	YEAR
SCI	Adams	AF	Black	7	QOPT1	2012
SCI	Adams	AF	Black	8	LASR1	2012
SCI	Adams	AF	Black	8	LASR1	2013
SCI	Adams	CS	Davis	3	ALMA1	2012
SCI	Adams	CS	Davis	3	ALMA1	2013
SCI	Adams	CS	Davis	6	DATA1	2012
SCI	Adams	CS	Davis	6	DATA1	2013
SCI	Adams	CS	Davis	6	PAPR1	2012

$$=$$

SCHOOL	DEAN
SCI	Adams

$$\bowtie$$

SCHOOL	DEPT	HEAD
SCI	AF	Black
SCI	CS	Davis

$$\bowtie$$

DEPT	ID	YEAR
AF	7	2012
AF	8	2012
AF	8	2013
CS	3	2012
CS	3	2013
CS	6	2012
CS	6	2013

$$\bowtie$$

ID	COURSE	YEAR
7	QOPT1	2012
8	LASR1	2012
8	LASR1	2013
3	ALMA1	2012
3	ALMA1	2013
6	DATA1	2012
6	DATA1	2013
6	PAPR1	2012



- Hodnoty atributů jedné relace se musí nacházet jako hodnoty (jiných) atributů jiné relace.

Mějme relační proměnné  $\mathbb{R}$  typu  $R$  a  $\mathbb{S}$  typu  $S$ . Referenční integritní omezení je výraz ve tvaru  $\rho_f(\pi_{R'}(\mathbb{R})) \subseteq \pi_{S'}(\mathbb{S})$ , kde  $R' \subseteq R$  a  $S' \subseteq S$ .

Omezení  $\rho_f(\pi_{R'}(\mathbb{R})) \subseteq \pi_{S'}(\mathbb{S})$  je *splněno* v  $D$ , pokud pro každou  $r \in \mathbb{R}^D$  platí, že existuje  $s \in \mathbb{S}^D$  tak, že  $r(f(y)) = s(y)$  pro každý atribut  $y \in S'$ .

## Příklady

$\mathbb{S}$  – studenti,  $\mathbb{P}$  – předměty,  $\mathbb{Z}$  – student má zapsaný předmět

- $\rho_{\text{ID} \leftarrow \text{STUDENT\_ID}}(\pi_{\text{STUDENT\_ID}}(\mathbb{Z})) \subseteq \pi_{\text{ID}}(\mathbb{S})$
- $\rho_{\text{ID} \leftarrow \text{COURSE\_ID}}(\pi_{\text{COURSE\_ID}}(\mathbb{Z})) \subseteq \pi_{\text{ID}}(\mathbb{P})$



- SQL (částečně) implementuje pomocí cizích klíčů
- pro  $\rho_f(\pi_{R'}(\mathbb{R})) \subseteq \pi_{S'}(\mathbb{S})$  je
  - $R'$  je **cizí klíč** relační proměnné  $\mathbb{R}$ , který se odkazuje na atributy  $S'$  proměnné  $\mathbb{S}$
  - nutný předpoklad:  $S'$  musí být klíč v  $\mathbb{S}$  (PRIMARY KEY nebo UNIQUE)

## SQL:

- jednoatributový cizí klíč (sloupcové omezení):  
REFERENCES jméno-tabulky ( atribut )
- víceatributový cizí klíč (omezení v rámci celé tabulky):  
FOREIGN KEY (r-atribut1, r-atribut2 , ...)  
REFERENCES jméno-tabulky (s-atribut1, s-atribut2, ...)

```
CREATE TABLE student (  
  id NUMERIC NOT NULL PRIMARY KEY,  
  name VARCHAR NOT NULL,  
  major VARCHAR NOT NULL);
```

```
CREATE TABLE course (  
  name VARCHAR NOT NULL,  
  ver NUMERIC NOT NULL,  
  PRIMARY KEY (name, ver));
```

```
CREATE TABLE enrolled (  
  student_id NUMERIC NOT NULL REFERENCES student (id),  
  c_name VARCHAR NOT NULL,  
  c_ver NUMERIC NOT NULL,  
  FOREIGN KEY (c_name, c_ver) REFERENCES course (name, ver),  
  year NUMERIC NOT NULL,  
  PRIMARY KEY (student_id, c_name, c_ver, year));
```



## Předpoklad

- je dáno  $\rho_{y \leftarrow x}(\pi_{\{x\}}(\mathbb{R})) \subseteq \pi_{\{y\}}(\mathbb{S})$
- modifikace proměnné  $\mathbb{R}$ , které **končí chybou**:
  - vložení hodnoty  $x$  do  $\mathbb{R}$ , která se nenachází mezi hodnotami  $y$  z  $\mathbb{S}$   
příklad: vložení výsledku zkoušky pro ID, které nepatří žádnému studentovi
  - jako v předchozím případě, pro UPDATE  
příklad: snaha modifikovat ID na hodnotu, která nepatří žádnému studentovi
- modifikace proměnné  $\mathbb{S}$ , kdy je možné **specifikovat chování**
  - pokus smazat z  $\mathbb{S}$   $n$ -tici  $s$ , kde  $s(y)$  se stále používá v  $\mathbb{R}$   
příklad: smazání studenta, který má stále záznamy o zkoušce
  - jako v předchozím případě, pro UPDATE místo DELETE  
příklad: modifikace ID studenta, který má záznamy o zkoušce





- NO ACTION implicitní chování – zastavení, nahlášení chyby
- RESTRICT: nahlášení chyby – bez možnosti odložení kontroly (do konce transakce)
- CASCADE: kaskádování – nedojde k chybě, ale změna se propaguje do tabulek, ve kterých je hodnota přítomna jako cizí klíč:
  - při DELETE se smažou odpovídající n-tice
  - při UPDATE se adekvátně změní hodnoty
- SET NULL: smazání hodnoty z n-tic – hodnoty v tabulkách s cizím klíčem budou nedefinované (nebezpečné)
- SET DEFAULT: nastavení na implicitní hodnotu – pokud má atribut dānu implicitní hodnotu pomocí DEFAULT



```
CREATE TABLE enrolled (  
  student_id NUMERIC NOT NULL  
    REFERENCES student (id) ON UPDATE CASCADE ON DELETE CASCADE,  
  course_id NUMERIC NOT NULL  
    REFERENCES course (id) ON DELETE RESTRICT ON UPDATE CASCADE,  
  year NUMERIC NOT NULL,  
  PRIMARY KEY (student_id, course_id, year));
```

```
/* odstrani i radky z tabulky enrolled, kde je student_id = 666 */  
DELETE FROM student WHERE id = 666;
```

```
/* aktualizuje i radky v tabulce enrolled se student_id = 666 */  
UPDATE student SET id = 777 WHERE id = 666;  
DROP TABLE student;           -- chyba  
DROP TABLE student RESTRICT;  -- chyba  
DROP TABLE student CASCADE;   -- odstrani omezeni, ne tabulky
```



- entita – odpovídá objektům reálného světa (osoba, věc)
  - silný entitní typ – existuje nezávisle na jiném entitním typu
  - slabý entitní typ – existence závisí na jiné entitním typu
- vlastnost – popisuje entity (jméno, barva, ...)
- vztah (relace, relationship) – vazba mezi dvěma a více entitami (pracovník-pracoviště)
  - kardinality 1:1, 1:N, M:N

## Vztah k relačnímu modelu

- entity odpovídají relacím
- i vztahy odpovídají relacím (cizí klíče)
- vlastnosti nemusí odpovídat atributům

## Poznámka

- při návrhu DB možné využít ER model společně s ER diagramem (viz skripta)
- možné použít i UML



## Přirozený klíč

- podmnožina atributů
- výhody:
  - již existují v databázi
  - možné ověřit vůči reálné entitě
- nevýhody:
  - může se ukázat, že zvolený klíč není jednoznačný (např. 2 lidé a jedno RČ)
  - může dojít ke změně (např. jména) – nutná propagace změn do cizích klíčů
  - mohou být rozsáhlé – dopad na výkon, psaní složitých dotazů

## Umělý klíč

- přidáný atribut, který není přímo vlastností entity
- uživatel k němu má přístup
- je sám o sobě ověřitelný, přidělen externě (např. ISBN, EAN)
- výhody:
  - efektivnější operace než s přirozeným klíčem
  - neměnné
- nevýhody
  - vyžaduje způsob, jak unikátně přiřadit hodnotu



## Náhradní klíč

- systémem generovaný klíč (často posloupnost přirozených čísel)
- uživatel by jej nikdy neměl vidět (interní součást systému)
- výhody:
  - efektivní provádění dotazů; úspora místa
  - umožňuje pružněji reagovat na nečekané situace (např. dva lidé stejné RČ)
  - neměnné
- nevýhody:
  - možnost zavléct duplicity do přirozených klíčů
  - problém, pokud je potřeba exportovat/sdílet data
  - nezapadá úplně do relačního modelu

## Náhradní klíč v SQL

```
CREATE TABLE foo
(bar INTEGER AUTO_INCREMENT, -- MySQL
 baz SERIAL,                -- PostgreSQL
 qux int IDENTITY(1,1));    -- MSSQL
```



- SQL deklarativní jazyk
- složité vyjádřit některé operace
- možnost vytvářet vlastní procedury a funkce
- forma abstrakce a optimalizace (data zůstávají v DBMS)  
CREATE PROCEDURE ...  
CREATE FUNCTION ...
- vlastnosti jazyků pro procedurální programování se liší mezi DBMS (Oracle – PL/SQL, PostgreSQL – PL/pgSQL, MSSQL – T-SQL, MySQL – SQL)
- kód je často nepřenositelný mezi DBMS

## Triggery

- procedury provedené při aktualizaci tabulky (INSERT, UPDATE, DELETE)
- buď na úrovni jednotlivých řádků nebo celé tabulky
- možnost číst obsah měněných řádků
- možnost nastavit provedení před nebo po změně



- každý DBMS řeší jinak
- vždy použijte dokumentaci

## Aktualizace tabulek

```
ALTER TABLE foo ADD column_bar int;
```

```
ALTER TABLE foo REMOVE column_bar;
```

```
ALTER TABLE foo ADD CONSTRAINT bar_constraint CHECK (bar > 0);
```

```
...
```

```
DROP TABLE foo;
```

```
DROP VIEW bar;
```



## Správa oprávnění

- uživatelé, role (skupiny)
- db. objekty jsou vlastněny uživateli/rolemi a mohou k nim mít oprávnění

## MySQL

```
CREATE USER 'joe'@'localhost' IDENTIFIED BY 'nbusr123';  
-- prideli vsechna opraveni ke vsem tabulkam v mojedb  
GRANT ALL PRIVILEGES ON mojedb.* TO 'joe'@'localhost';  
-- prideli opraveni provest SELECT nad tabulkou foo  
GRANT SELECT PRIVILEGES ON mojedb.foo TO 'joe'@'localhost';  
-- odebere opraveni mazat zaznamy v tabulce foo  
REVOKE DELETE PRIVILEGES ON mojedb.foo FROM 'joe'@'localhost';
```

## PostgresSQL

```
CREATE USER joe PASSWORD 'nbusr123';  
GRANT ALL PRIVILEGES ON DATABASE mojedb TO joe;  
GRANT SELECT PRIVILEGES ON foo TO joe;  
REVOKE DELETE PRIVILEGES ON foo FROM joe;
```





## Aktualizace statistik

- DBMS si udržuje statistiky o datech (velikost, nejčastější hodnoty) a podle nich vytváří prováděcí plán
- měly by se udržovat průběžně (nemusí vždy odpovídat realitě)
- vynucená aktualizace:  
MySQL: `ANALYZE TABLE foo;`  
PostgreSQL: `ANALYZE foo;`

## Uvolnění místa

- DBMS obvykle data nemaže přímo (pomalá operace), pouze přidá příznak, že daný řádek je odstraněný
- k uvolnění místa by mělo docházet průběžně po dávkách (podle nastavení)
- vynucené uvolnění místa:  
MySQL: `OPTIMIZE TABLE foo;`  
PostgreSQL: `VACUUM foo;`



## Embedded databáze

- knihovna, součást procesu
- přímý přístup, volání funkcí
- často jednovýživatelské
- např. SQLite, Apache Derby, FireBird, SQL Server Compact, ...

## Architektura Client/Server

- DBMS samostatný proces
- komunikace typicky přes TCP/IP (příp. unixové sockety, jiná API)
- víceuživatelský přístup
- možnost škálování
- na straně klienta je vyžadován ovladač pro přístup k DB

## Obecný přístup

- programovací jazyky poskytují standardní prostředky pro přístup k DBMS
- mohou poskytovat přístup k embedded i vzdáleným DBMS
- např. Java – JDBC, C# – ADO.NET, PHP – PDO, vlastní API



- JDBC – standardní rozhraní v Javě
- obvykle je potřeba přibalit k projektu JDBC ovladač (soubor \*.jar dodávaný výrobcem DBMS)
- URL pro připojení k db:  
`jdbc:<ovladac>://<nazev-stroje>:<port>/?<dodatecne-parametry>`

- příklady:

```
jdbc:postgresql://localhost/mojedb
```

```
jdbc:mysql://example.com:3306/foobase
```

- připojení:

```
String connectionURL = "jdbc:postgresql://localhost/mojedb";
```

```
Connection con
```

```
    = DriverManager.getConnection(connectionURL, "joe", "nbusr123");
```

## Spolupráce SQL s Java (JDBC) (2/3)



```
// vytvoři db prikaz
try (Statement stmt = con.createStatement()) {

    // provede s tímto příkazem dotaz
    stmt.executeQuery("SELECT * FROM employees");

    try (ResultSet results = stmt.getResultSet()) {
        // iteruje přes všechny řádky výsledku
        while (results.next()) {

            // vybere hodnotu v prvním a druhém sloupci
            // (sloupce jsou indexovány od 1!)
            System.out.println(results.getString(1));
            System.out.println(results.getInt(2));

            // vybere hodnotu podle názvu sloupce
            System.out.println(results.getDouble("salary"));
        }
    }
}
```



## Aktualizace

```
stmt.executeUpdate("UPDATE employee SET salary = salary * 1.1");
```

## Připravené dotazy

- optimalizace zpracování dotazu
- předávání parametrů

```
PreparedStatement empStmt =  
    con.prepareStatement("SELECT * FROM employees WHERE name = ?");
```

```
PreparedStatement insertStmt =  
    con.prepareStatement("INSERT INTO employees (name, age, salary)"  
        + "VALUES (?, ?, ?)");
```

```
empStmt.setString(1, "Alice");  
empStmt.executeQuery();
```

```
insertStmt.setString(1, "Xavier");  
insertStmt.setInt(2, 40);  
insertStmt.setDouble(3, 24000.0);  
insertStmt.executeUpdate();
```



```
string conStr = "Server=localhost;Database=mojedb;User Id=joe; Password=nbusr123"
using (SqlConnection con = new SqlConnection(conStr)) {
    string queryString = "SELECT * FROM employee WHERE name = @name";
    SqlCommand command = new SqlCommand(queryString, con);
    command.Parameters.AddWithValue("@name", "Bob");

    try {
        connection.Open();
        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read()) {
            Console.WriteLine("\t{0}\t{1}\t{2}",
                reader[0], reader[1], reader["salary"]);
        }
        reader.Close();
    } catch (Exception ex) {
        Console.WriteLine(ex.Message);
    }
}
```

- PHP obsahuje API pro přímé připojení k nejběžnějším databázím (MySQL, PostgreSQL, ...)
- možné využít všech vlastností DBMS
- API jsou si podobná ale nekompatibilní
- **nepřenositelné řešení!**

<?

```
$con = pg_pconnect("host=localhost dbname=mojedb user=joe password=nbusr123");  
if (!$con) die("DB error");
```

```
$result = pg_query("SELECT name, salary FROM employees");  
if (!$result) die("Query error");
```

```
while ($row = pg_fetch_row($result)) {  
    echo "Name: {$row[0]}, Salary: {$row[1]}\n";  
}
```

?>

- univerzální rozhraní pro různé DB

<?

```
$db = new PDO('pgsql:host=localhost;dbname=mojedb','joe','nbusr123');  
$result = $db->query('SELECT * FROM employees');
```

```
while($row = $result->fetch(PDO::FETCH_ASSOC)) {  
    echo "Name: {$row['name']}, Salary: {$row['salary']}\n";  
}
```

*// pripravene dotazy*

```
$stmt = $db->prepare('SELECT * FROM employees WHERE name = :name');  
$params = array(':name' => 'Alice');
```

```
$result = $stmt->execute($params);  
$row = $result->fetch(PDO::FETCH_ASSOC);
```

?>



<?

```
$login = $_REQUEST["login"];  
$pasw = sha1($_REQUEST["passwd"]);  
$q = "SELECT * FROM users WHERE (user = '$login' AND pasw = '$pasw')";  
$loggedIn = pg_num_rows(pg_query($q));
```

?>

- pokud je login foo a heslo bar ⇒ funguje dobře
- pokud je login foo' or true) -- a heslo bar ⇒ máme problém
- protože:  

```
SELECT * FROM users WHERE (user = 'foo' or true) -- pass='...')
```
- přes funkce DB se lze dostat k dalším citlivým informacím
- řešení ⇒ pokud to API umožňuje parametrizované dotazy (JDBC, ADO, PDO) nebo důsledná kontrola vstupů
- podobný problém nastane kdekoliv, kde použijeme eval