Databázové systémy

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

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Motivace pro normalizaci

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)

Opakování: 1. NF

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)

2. NF

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

1. je v 1. NF,
2. pokud pro každý atribut $y \in R$ takový, že $y \notin K_i$ pro všechna $i = 1, \ldots, n$, platí
$$\Gamma \not\models K \Rightarrow \{y\},$$
pro všechny $K \subset K_i$, kde $i = 1, \ldots, n$.

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.

Příklad

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

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

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

$\bowtie$

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

3. NF

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

Funkční závislost $A \Rightarrow C$ v teorii Γ 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 klíčí, celém klíči a ničem jiném.

Příklad

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

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

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

$\bowtie$

DEPT_ID	DEPT_NAME
1	books
2	clothes

Anomálie v 3NF

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

Boyce-Coddova normální forma (BCNF)

Mějme relační schéma R a teorii Γ. Pak R je v BCNF vzhledem k Γ, 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 klíčí, celém klíči, a ničem jiném.

Normalizace pomocí dekompozice

Pokud není R v BCNF vzhledem k Γ, 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 $R_2 = A \cup (R \backslash B)$ 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 Γ_1, pokud není v BCNF a analogicky pro R_2 a Γ_2

Poznámka:

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

Příklad: Schéma, které není v BCNF

Uvažujme relační schéma:

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

a teorii Γ popisující závislosti mezi atributy:

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

Schéma R není v BCNF vzhledem k Γ, protože (například):

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

Příklad: Normalizace pomocí dekompozice

$R = \{\texttt{SCHOOL}, \texttt{DEAN}, \texttt{DEPT}, \texttt{HEAD}, \texttt{ID}, \texttt{COURSE}, \texttt{YEAR}\}$
$\Gamma_= \{\{\texttt{DEPT}\} \Rightarrow \{\texttt{HEAD}, \texttt{SCHOOL}, \texttt{DEAN}\}, \ldots\}$ (viz předchozí příklad)

- $R_1 = \{\texttt{SCHOOL}, \texttt{DEAN}, \texttt{DEPT}, \texttt{HEAD}\}$
  $\Gamma_1 = \{\{\texttt{DEPT}\} \Rightarrow \{\texttt{HEAD}, \texttt{SCHOOL}, \texttt{DEAN}\}, \{\texttt{SCHOOL}\} \Rightarrow \{\texttt{DEAN}\}\}$
  - $R_{11} = \{\texttt{SCHOOL}, \texttt{DEAN}\}$
    $\Gamma_{11} = \{\{\texttt{SCHOOL}\} \Rightarrow \{\texttt{DEAN}\}\}$
  - $R_{12} = \{\texttt{SCHOOL}, \texttt{DEPT}, \texttt{HEAD}\}$
    $\Gamma_{12} = \{\{\texttt{DEPT}\} \Rightarrow \{\texttt{HEAD}, \texttt{SCHOOL}\}, \{\texttt{SCHOOL}\} \Rightarrow \{\}\}$
- $R_2 = \{\texttt{DEPT}, \texttt{ID}, \texttt{COURSE}, \texttt{YEAR}\}$
  $\Gamma_2 = \{\{\texttt{DEPT}\} \Rightarrow \{\}, \{\texttt{COURSE}, \texttt{YEAR}\} \Rightarrow \{\texttt{ID}\}, \{\texttt{ID}, \texttt{YEAR}\} \Rightarrow \{\texttt{DEPT}\}\}$
  - $R_{21} = \{\texttt{DEPT}, \texttt{ID}, \texttt{YEAR}\}$
    $\Gamma_{21} = \{\{\texttt{DEPT}\} \Rightarrow \{\}, \{\texttt{ID}, \texttt{YEAR}\} \Rightarrow \{\texttt{DEPT}\}\}$
  - $R_{22} = \{\texttt{ID}, \texttt{COURSE}, \texttt{YEAR}\}$
    $\Gamma_{22} = \{\{\texttt{COURSE}, \texttt{YEAR}\} \Rightarrow \{\texttt{ID}\}, \{\texttt{ID}, \texttt{YEAR}\} \Rightarrow \{\}\}$

Rozklad

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

Referenční integritní omezení

- 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 $\mathtt{D}$, pokud pro každou $r \in \mathbb{R}^{\mathtt{D}}$ platí, že existuje $s \in \mathbb{S}^{\mathtt{D}}$ 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

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

Referenční integritní omezení v SQL

- 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, ...)
```

Příklad

```
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));
```

Chování cizích klíčů v SQL

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

Možné reakce na porušení referenční integrity (1/2)

- `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 dánu implicitní hodnotu pomocí `DEFAULT`

Možné reakce na porušení referenční integrity (2/2)

```
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
```

Konceptuální modelování, ER model

- 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

Přirozené, umělé a náhradní klíče (1/2)

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

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

Přirozené, umělé a náhradní klíče (2/2)

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,                 -- PostgresSQL
 qux int IDENTITY(1,1));     -- MSSQL
```

Procedurální programování na straně databáze

- 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, PostgresSQL – PL/pgSQL, MSSQL – T-SQL, MySQL – SQL)
- kód je často nepřenositelný mezi DBMS

Triggery

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

Prostředky pro správu databáze (1/3)

- 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;
```

Prostředky pro správu databáze (2/3)

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 opravneni ke vsem tabulkam v mojedb
GRANT ALL PRIVILEGES ON mojedb.* TO 'joe'@'localhost';
-- prideli opravneni provest SELECT nad tabulkou foo
GRANT SELECT PRIVILEGES ON mojedb.foo TO 'joe'@'localhost';
-- odebere opravneni 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;
```

Prostředky pro správu databáze (3/3)

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;`
  PostgresSQL: `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 dávkach (podle nastavení)
- vynucené uvolnění místa:
  MySQL: `OPTIMIZE TABLE foo;`
  PostgresSQL: `VACUUM foo;`

Spolupráce SQL s jinými jazyky

Embedded databáze

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

Architektura Client/Server

- DBMS samostatný proces
- komunikace typicky přes TCP/IP (přip. 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

Spolupráce SQL s Java (JDBC) (1/3)

- 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-parametery>`
- 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)

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

   // provede s timto prikazem dotaz
   stmt.executeQuery("SELECT * FROM employees");

   try (ResultSet results = stmt.getResultSet()) {
      // iteruje pres vsechny radky vysledku
      while (results.next()) {

         // vybere hodnotu v prvnim a druhem sloupci
         // (sloupce jsou indexovany od 1!)
         System.out.println(results.getString(1));
         System.out.println(results.getInt(2));

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

Spolupráce SQL s Java (JDBC) (3/3)

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 INT 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();
```

Spolupráce SQL s C# (ADO.NET)

```
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);
    }
}
```

Spolupráce SQL s PHP (přímý přístup)

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

```php
<?
$con = pg_pconnect("host=localhost dbname=mojedb user=joe password=nbusr123");
if (!$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";
}
?>
```

Spolupráce SQL s PHP (PDO)

- univerzální rozhraní pro různé DB

```php
<?
$db = new PDO('postgresql: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);
?>
```

SQL injection attack

```php
<?
$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`