Databázové systémy

Integrácia programovacích jazykov s databázovými systémami

2026 Lukáš Miňo

Motivácia

- **Problém:** Databáza je len úložisko dát, aplikácia len spracováva dáta.
- **Objektovo-relačný nesúlad (Impedance Mismatch):**
  - **Programovací jazyk (imperatívny prístup):** Dáta berie ako objekty, polia alebo prúdy a krok za krokom ich prechádza (napr. cez cykly a podmienky), mení ich stav v pamäti a reaguje na udalosti. Hovoríte **ako** má dáta spracovať (**how to do things**).
  - **RDBMS (deklaratívny prístup):** Dáta vníma ako relačné tabuľky (množiny) a spracúva ich hromadne naraz. Cez SQL poviete iba **čo** chcete získať, nie akým postupom to má urobiť (**what to do**).
- **Cieľ:** Vytvoriť bezpečný a efektívny komunikačný kanál na spracovanie používateľských vstupov (formulárov) a dynamické generovanie webových aplikácií.

Vrstvy komunikácie

- **Aplikačná vrstva:** Biznis logika programu a spracovanie formulárov (PHP, Python, Java, C#, Node.js, R, ...).
- **Abstraktné rozhranie (API):** Štandardizuje prístup nezávisle od konkrétnej databázy (PDO, DB-API 2.0, JDBC, DBI).
- **Databázový ovládač (Driver):** Prekladá volania do natívneho binárneho protokolu databázy (napr. MySQL Wire Protocol).

SQL Injection je **kritická** bezpečnostná hrozba

- **Princíp útoku:** Priame spájanie vstupu z formulára (napr. hodnota z GET/POST) do SQL textu umožní útočníkovi zmeniť štruktúru dopytu.
- **Zraniteľný kód:**

"SELECT * FROM user WHERE username = '" + username + "' AND password = '" + password + "'";

- **Podvrhnutý vstup:**
  - **Username:** admin' -- (v MySQL nezabudnúť na medzeru za dvoma pomlčkami)
  - **Password:** cokolvek
- **Výsledný SQL dopyt, ktorý dostane databáza:**

SELECT * FROM user WHERE username = 'admin' -- ' AND password = 'cokolvek'

- **Podvrhnutý vstup:**
  - **Username:** ' OR 1 = 1 -- (v MySQL nezabudnúť na medzeru za dvoma pomlčkami)
  - **Password:** cokolvek
- **Výsledný SQL dopyt, ktorý dostane databáza:**

SELECT * FROM user WHERE username = ' ' OR 1 = 1 -- ' AND password = 'cokolvek'

SQL Injection je **kritická** bezpečnostná hrozba

- **Prepared Statements:**
  - **Ochrana proti SQL injection (Bezpečnosť):** Vstupné dáta od používateľa sa nikdy priamo nespájajú do textu SQL príkazu. Databáza ich berie striktne ako čistú hodnotu, nie ako spustiteľný kód, čím úplne znemožňuje útočníkovi podvrhnúť vlastný SQL príkaz.
  - **Vyšší výkon (Rýchlosť):** Databázový systém (RDBMS) analyzuje, skompiluje a optimalizuje plán vykonania dotazu iba raz. Ak ten istý dotaz spúšťate opakovane s inými dátami (napr. v cykle), databáza už len dosadzuje nové hodnoty do hotovej šablóny.

```
String sql = "SELECT meno, priezvisko FROM Osoba WHERE pohlavie = ?";
try (Connection conn = DriverManager.getConnection(URL, USER, PASS);
   PreparedStatement pstmt = conn.prepareStatement(sql)) {
   pstmt.setString(1, pohlavie);
   ResultSet rs = pstmt.executeQuery();
   while (rs.next()) {
      html
         .append("<li>")
         .append(rs.getString("meno"))
         .append(" ")
         .append(rs.getString("priezvisko"))
         .append("</li>");
   }
}
```

SQL Injection vs. Cross-site scripting (XSS)

- **Častá chyba študentov:** Použijú Prepared Statement (ochránia databázu pred SQL Injection pri spracovaní formulára), ale dáta vypíšu priamo do HTML bez escapovania.
- **Útok Stored XSS:** Ak útočník uloží do stĺpca meno reťazec **<script>alert('Hack')</script>**, databáza ho bezpečne uloží, no pri zobrazení na webe sa kód spustí v prehliadači obete.
- **Pravidlo: Parametrizácia na vstupe** do DB a **HTML escapovanie** (htmlspecialchars, Jinja2, escapeHtml) **na výstupe** do prehliadača.

Prečo prihlasovacie údaje nesmú byť súčasťou zdrojového kódu?

- **Riziko „Hardcoded Credentials“:**
  - Priame zapísanie mena a hesla do kódu (napr. `$pass = "TajneHeslo123"`) vedie k ich úniku pri zdieľaní kódu alebo nahratí do verzovacieho systému (Git / GitHub).
  - VEREJNÝ REPOZITÁR = OKAMŽITÝ ÚTOK (automatizované boty skenujú GitHub v reálnom čase a hľadajú heslá a API kľúče).
- **Riešenie: Oddelenie konfigurácie (12-Factor App):**
  - Konfigurácia sa presúva do externého súboru (.env, config.properties, .Renviron), ktorý sa načíta do **premenných prostredia (Environment Variables)** pri štarte aplikácie.

Prečo prihlasovacie údaje nesmú byť súčasťou zdrojového kódu?

- **Pravidlá pre prácu v tíme (Git Workflow):**
  - **.env:** Obsahuje reálne heslá. Je pridaný do súboru **.gitignore** (**nikdy sa nepushuje do Gitu**).
  - **.env.example:** Obsahuje iba prázdnu šablónu kľúčov bez hesiel. Je nahratý v Gite ako **návod pre ostatných vývojárov**.

```
projekt/
├── .env              ← SKUTOČNÉ HESLÁ (v .gitignore, iba lokálne na serveri)
├── .env.example      ← ŠABLÓNA (nahratá na GitHub: DB_PASS=zadaj_heslo)
├── .gitignore        ← Obsahuje riadok: .env
└── app.py            ← Číta údaje cez os.getenv('DB_PASS')
```

Bezpečnosť súboru .env na serveri

- **Riziko verejnej viditeľnosti:** Ak nahráte **.env** do verejného priečinka (**public_html/www**) s právami na čítanie (**0644**), webový server ho pošle každému, kto zadá jeho URL (**https://vas-web.sk/.env**).
- **Presun mimo verejný priečinok:** Uložte **.env** o úroveň vyššie než DocumentRoot (napr. **/home/login/.env**). Z internetu je nedostupný, skript ho načíta cez relatívnu cestu (**/../.env**).
- **Zákaz prístupu v konfigurácii servera:** Ak súbor musí zostať v **public_html**, zakážte jeho zobrazenie cez webový server (vráti 403 Forbidden):
  - **Apache (.htaccess):** <Files ".env"> Require all denied </Files>
  - **Nginx:** location ~ /\. { deny all; return 404; }
- **Súborové práva na úrovni OS (chmod):** Na vlastnom Linux serveri nastavte chmod 0640 (pre skupinu www-data) alebo chmod 0600 (pre Python/Node.js), aby súbor neprečítali ostatní používatelia v systéme.

Výkon, asynchrónnosť a správa transakcií

- **Connection Pooling:** Otváranie TCP spojenia je drahé. Servery (najmä **Java** a **Node.js**) udržiavajú pool otvorených spojení pripravených na okamžité použitie.
- **Synchrónny vs. Asynchrónny prístup:** Kým klasické **PHP** blokuje vlákno počas čakania na databázu, **Node.js** (a asynchrónny **Python**) využíva neblokujúce I/O (async/await).
- **ACID v aplikačnom kóde:** Riadenie transakcií (BEGIN, COMMIT, ROLLBACK) v blokoch **try-catch** pri viacnásobných zápisoch.

Raw SQL vs. ORM (Object-Relational Mapping)

- **Raw SQL:**
  - SELECT meno, priezvisko FROM osoba WHERE pohlavie = ?
  - Plná kontrola nad výkonom, indexmi a zložitými JOIN/UNION operáciami.
- **ORM (Object-Relational Mapping):**
  - napr. Prisma v Node.js

```
const osoby = await prisma.osoba.findMany({
      where: { pohlavie: pohlavie },
      select: { meno: true, priezvisko: true },
});
```

  - napr.SQLAlchemy v Pythone

```
osoby = session.query(Osoba.meno, Osoba.priezvisko)
.filter(Osoba.pohlavie == pohlavie).all()
```

  - Rýchly vývoj, typová bezpečnosť, **ochrana pred SQL Injection** v základe, avšak riziko generovania neefektívnych dopytov (tzv. **N+1 Query Problem**).

Zhrnutie a rady

1. **Bezpečnosť na prvom mieste (Zero Trust k vstupom)**
   - Vždy používajte **Prepared Statements:** Nikdy nespájajte premenné z formulárov ($_GET, request.args) priamo do SQL reťazca.
   - Chráňte aj výstup **(Anti-XSS):** Dáta z databázy pred vykreslením do HTML vždy escapujte (htmlspecialchars, Jinja2, escapeHtml), aj keď boli do DB uložené bezpečne.
2. **Bezpečná konfigurácia**
   - **Žiadne heslá v kóde:** Prihlasovacie údaje držte výhradne v **.env** (alebo **config.properties**, **.Renviron**) a nezabudnite ich pridať do **.gitignore**.
   - **Princíp najmenších oprávnení:** Webová aplikácia sa **nemá** pripájať ako **root**. Vytvorte pre ňu dedikovaného DB používateľa iba s právami, ktoré reálne potrebuje (SELECT, INSERT, UPDATE).
3. **Výkon a životný cyklus spojenia**
   - **Zatvárajte zdroje alebo používajte Pool:** Neuzavreté spojenia vyčerpajú pamäť databázy. Využívajte try-with-resources (**Java**), on.exit() (**R**) alebo **Connection Pooling** (**Node.js, Python**).
   - **Preneste ťažkú prácu na databázu:** Nefiltrujte a netrieďte tisíce riadkov v cykle v PHP/Jave/Pythone – nechajte databázu urobiť WHERE, JOIN a ORDER BY priamo nad indexami.
4. **Odolnosť voči chybám (Error Handling & ACID)**
   - **Transakcie pri viacerých zápisoch:** Ak jedna akcia používateľa mení viac tabuliek naraz, obaľte ju do transakcie s ROLLBACK v bloku catch.
   - **Skryte interné chyby DB:** Surové hlášky typu SQLSTATE[23000]: Integrity constraint violation **logujte** na serveri, no používateľovi **zobrazte zrozumiteľnú správu**.

Tutoriál

- https://www.mysqltutorial.org/php-mysql/
- https://www.mysqltutorial.org/mysql-nodejs/
- https://www.mysqltutorial.org/mysql-jdbc-tutorial/
- https://www.mysqltutorial.org/python-mysql/
- https://www.mysqltutorial.org/perl-mysql/
- ...

Demo

https://s.ics.upjs.sk/~lukasmino/demo/