Databáze, dokumentový model

Dokumentový model a MongoDB

Tomáš Urbanec

Katedra informatiky, Univerzita Palackého v Olomouci

ZS 2025/2026

Úvod

- Relační databáze jako dominantní model desítky let.
- Výborně se hodí pro:
  - dobře strukturovaná data
  - stabilní schéma
- Dnešní aplikace mají často jiné požadavky:
  - rychlé změny schématu
  - obecně různá schémata
  - obrovské objemy dat (Big Data)
  - horizontální škálování
  - data nemusí být dokonale strukturovaná
- Potřeba **alternativ k SQL (relačnímu modelu)**

NoSQL databáze

- **NoSQL = „Not Only SQL“**
- Nejde o jeden model - jde o **skupinu databází**:
  - dokumentové,
  - klíč–hodnota,
  - grafové,
  - sloupcové,
  - “in-memory”,
  - ...
- Typické vlastnosti:
  - flexibilní schéma / často bez schématu
  - horizontální škálování (sharding)
  - důraz na výkon a nízkou odezvu
  - často opouštějí ACID
  - jiný přístup ke konzistenci

Motivace pro NoSQL

- Relační model naráží na limity při
  - ukládání velkých a různorodých struktur dat
  - rychlých změnách schématu
  - masivním paralelním zatížení
- Aplikace pracují s jinými typy dat
  - JSON,
  - vnořenými strukturami obecně,
  - grafová data,
  - klíč - hodnota,
  - ...
- Cílem NoSQL není nahradit SQL, ale zvolit vhodný model pro daný typ dat.

Dokumentové databáze

- Data se ukládají jako **dokumenty** (strukturované objekty).
- Nejčastěji ve formátu:
  - JSON (JavaScript Object Notation; znáte JSON?)
  - BSON (binární JSON - optimalizace výkonu; a znáte BSON?)
- Dokumenty obsahují hodnoty, vnořené dokumenty, seznamy
- Dokumenty mají variabilní strukturu
- Výhody
  - data přirozeně odpovídají realitě
  - nepovinné schéma
  - související data jsou uložená pohromadě, tj. žádné JOINy
- Nevýhody
  - duplicitní data napříč dokumenty (úložiště, konzistence, aktualizace)
  - slabší konzistence
  - chybějící pevné schéma může být i nevýhoda

Vsuvka - JSON

- Datový formát používaný k ukládání a přenosu strukturovaných dat.
- Každá hodnota v JSONu může být:
  - **objekt** (množina dvojic *klíč : hodnota*, v {})
  - **pole** (uspořádaný seznam hodnot; v [])
  - **řetězec**
  - **číslo**
  - **boolean** (`true` / `false`)
  - **null**

```
{
  "jmeno": "Tomáš Fuk",
  "adresy": [
    { "id": 1, "mesto": "Olomouc", "psc": "77900", "trvala": true},
    { "id": 2, "mesto": "Brno", "psc": "60200", "trvala": false}
  ]
}
```

Vsuvka - BSON

- je binární formát používaný MongoDB pro ukládání dokumentů.
- efektivnější ukládání než čistý JSON
- **rychlejší parsování** (kompaktní binární reprezentace)
- umožňuje **typy, které JSON nemá**. Např.
  - `ObjectId` (automatické primární klíče)
  - `Date` (časová značka)
  - `Binary` (binární data)
  - rozlišení `Int32` a `Int64`
  - `Decimal128` (přesné desetinné hodnoty)
- tj. BSON = JSON + nové typy + rychlost
  - ideální pro databázi

MongoDB

- Nejrozšířenější dokumentová databáze, www.mongodb.com
- Vlastnosti
  - Ukládá dokumenty v BSONu
  - Kolekce dokumentů - “tabulka” bez pevného schématu
- Podporuje
  - dotazy,
  - indexy,
  - agregace,
  - horizontálního škálování (sharding)
- Velmi populární
- Existují i jiné
  - CouchDB,
  - Firebase firestore,
  - RethinkDB,
  - ...

MongoDB - náhled modelu

- Dokument = množina atributů a jejich hodnot.
- Atribut může mít
  - atomickou hodnotu
  - kolekci dokumentů (vnoření)
  - seznam hodnot
- Kolekce = množina dokumentů se stejným (nebo podobným) schématem.
- Schéma
  - může být definováno, ale **není povinné**
- Model přímo odpovídá JSON/BSON objektům

Dokumentový model

Atributy a schémata

- **Atribut** - řetězec písmen a podtržítka.
  - **Atomické atributy** - atributy s spočetnou doménou atomických hodnot D_y
  - **Kolekční atributy** - atributy jejichž hodnota je kolekce
- **(Dokumentové) schéma hloubky** k
  - Prázdná množina je schéma hloubky 0.
  - Je-li schéma S tvaru $\{\langle y_1, S_1\rangle, \ldots, \langle y_n, S_n\rangle\}$, kde maximální hloubka schématu S_i atributu y_i je k, pak S je schéma hloubky $k+1$.
- **Dokumentové schéma** - dokumentové schéma S hloubky k pro nějaké $k \in \mathbb{N}$.
- **Atributy schématu** - $Y(S) = \{y \mid \langle y, S'\rangle \in S\}$
- Pro zápis konkrétního atributu schématu používáme tečkovou notaci
  - např. produkt.cena pro $\{\langle kupec, \emptyset\rangle, \langle produkt, \{\langle cena, \emptyset\rangle\}\rangle\}$

(Příklady)

Dokumenty a kolekce

- **Komponenta** - dvojice $\langle y, d \rangle$, kde
  - y je atomický atribut a d hodnota z jeho domény
    - ⋆ **atomická komponenta**
  - y je kolekční atribut se schématem S a d je kolekce nad S
    - ⋆ **kolekční komponenta**
- **Dokument nad** S - množina $c = \{\langle y_1, d_1 \rangle, \ldots, \langle y_n, d_n \rangle\}$ komponent, kde
  - c je zobrazení s doménou $Y(S)$
  - pro každý kolekční atribut $y \in S$ je $c(y)$ kolekce nad $S(y)$
- **Kolekce nad** S - konečná množina dokumentů nad S
- **Doc(S)** - množina všech dokumentů nad S
- **Vlastnost (dokumentů) nad** S - výroková forma $V(c)$, kde $c \in Doc(S)$
- **Určující vlastnost kolekce** C **nad** S
  - vlastnost $V(c)$ platící právě pro všechny dokumenty $c \in C$.
  - tj. $C = \{c \in Doc(S) \mid V(c)\}$

(Příklady)

Přejmenování

- **Přejmenování atributů** S_1 **na** S_2 je bijekce $h\colon Y(S_1) \to Y(S_2)$, která zachovává typy a kde $S_1(y) = S_2(h(y))$ pro každý $y \in Y(S_1)$
- **Přejmenování dokumentu** c **nad** S_1 **podle** h, přejmenování atributů S_1 na S_2, je dokument nad S_2

$$\rho_h(c) = h^{-1} \circ c = \{\langle h(y), d\rangle \mid \langle y, d\rangle \in c\}$$

- **Přejmenování kolekce** C **nad** S_1 **podle** h, přejmenování atributů S_1 na S_2, je kolekce nad S_2

$$\rho_h(C) = \{\rho_h(c) \mid c \in C\}$$

(Příklady)

Kolekční proměnná

- Atribut `_id` dokumentu nazýváme **identifikátor**.
- **Kolekční proměnná** je proměnná, jejíž hodnotou je kolekce nad schématem S obsahujícím atribut `_id`, kde všechny dokumenty mají unikátní hodnotu `_id`.
- Kolekční proměnná má svou **určující vlastnost**.
- Přípustné hodnoty kolekční proměnné jsou právě kolekce mající její určující vlastnost.
- Často se mluví jen o kolekci - kontext určuje, jestli jde o proměnnou nebo její hodnotu.

(Příklady)

Základy práce s kolekcemi v MongoDB

Nástroje

- MongoDB běží na stejném serveru jako PostgreSQL
- Port 27017
- Uživatelské jméno i heslo je stejné, jako (výchozí) do PostgreSQL a systému
- Můžete používat `mongosh` (Mongo shell), `MongoDB Compass` (grafická aplikace), či doplňky do oblíbeného IDE
- Lze provozovat i lokálně (viz materiály dr. Laštovičky)

Kolekční proměnné / kolekce

- Není nutné předem deklarovat kolekci ani její schéma
- Schéma je ve výchozím stavu **dynamické** (přizpůsobuje se vloženým datům)
- Kolekce vznikne **až při prvním vložení dat**
- Pokud kolekce neexistuje, dotaz na ni vrátí chybu nebo prázdný výstup podle operace

Vkládání do kolekce

- Základní příkaz `insertMany`
  - Nastaví hodnotu kolekční proměnné `kolekce` na sjednocení původní hodnoty `kolekce` s kolekce `kolekce2`
  - Původní hodnota `kolekce` a kolekce `kolekce2` musí být disjunktní, jinak chyba (duplicita `_id`)

  ```
  db.kolekce.insertMany(kolekce2)
  ```
- Existuje i `insertOne` pro přidání jednoho dokumentu.
- Nedodáte-li `_id` bude vygenerováno.
- Jen `insert` je zastaralý, nepoužívejte jej.

(Ukázka)

Kolekční výraz

- Výraz, jehož hodnota je kolekce
- Základní kolekční výraz `find`
  - vrátí hodnotu existující kolekční proměnné `kolekce`
  - všimněte si, že výsledek přísně vzato není JSON (zjednodušení výstupu)

```
db.kolekce.find()
```

(Ukázka)

Mazání kolekce

```
db.kolekce.drop()
```

- Nastaví hodnotu kolekční proměnné `kolekce` na prázdnou kolekci

(Ukázka)

Podmínka

- **Podmínka**
  - je výraz, u kterého umíme rozhodnout, zda jej nějaký dokument splňuje
  - dokument c *splňuje podmínku* θ, pokud je tvrzení „θ platí pro c" pravdivé
- Říkáme, že podmínka θ je *nad schématem* S, pokud pro každý dokument c nad S umíme rozhodnout, zda c splňuje θ
- Základní struktura podmínek odpovídá dobře známé struktuře formulí výrokové logiky
  - atomické podmínky
  - konjunkce a disjunkce podmínek
  - negace podmínky (v MongoDB omezená)

(Příklady)

Atomická podmínka na rovnost

- Nechť
  - y je atomický atribut
  - $d \in D_y$ je hodnota z domény atributu y
- Pak
  - výraz $y = d$ je podmínka nad $\{y\}$
  - dokument c splňuje $y = d$, právě když $c(y) = d$
- JSON zápis:
  - základní tvar:
    ```
    { "y": { "$eq": d }}
    ```
  - zkratka:
    ```
    { "y": d }
    ```

(Ukázky)

Atomická podmínka na nerovnost

- Nechť:
  - y je atomický atribut
  - $d \in D_y$ je hodnota z domény atributu y
- Pak
  - výraz $y \neq d$ je podmínka nad $\{y\}$
  - dokument c splňuje $y \neq d$, pokud $c(y) \neq d$
- JSON zápis:
  - základní tvar

```
{ "y": { "$ne": d }}
```

(Ukázka)

Logické spojky – konjunkce

- Nechť $P_1, \ldots, P_n$ jsou podmínky nad S, $n > 0$
- Pak
  - $P_1 \wedge \cdots \wedge P_n$ je podmínka nad S
  - dokument ji splňuje, *právě když* splňuje každou P_i
- JSON zápis:

```
{
  "$and": [  P1, ..., Pn ]
}
```

(Ukázka)

Konjunkce - zkratka

- I konjunkci podmínek lze zapsat úsporněji.

```
{
  "y1": e1,
  ...,
  "yn": en
}
```

je zkratka za

```
{ "$and": [ { "y1": e1 }, ..., { "yn": en } ] }
```

(Ukázka)

Logické spojky – disjunkce

- Nechť $P_1, \ldots, P_n$ jsou podmínky nad S, $n > 0$
- Pak
  - $P_1 \vee \cdots \vee P_n$ je podmínka nad S
  - dokument ji splňuje, pokud splňuje *alespoň jednu* z P_i
- JSON zápis:

```
{
  "$or": [ P1, ..., Pn ]
}
```

(Ukázka)

Logické spojky - negace

- Pro podmínku θ nad S
  - $\neg\theta$ je podmínka nad S
  - dokument c splňuje $\neg\theta$, právě když *nesplňuje* θ
- Negaci obecně **nezapisujeme přímo** v JSON
  - MongoDB to nepodporuje z důvodů efektivity
  - Používáme ekvivalentní úpravy
  - De Morganovy zákony:
    - $\neg(\theta_1 \wedge \cdots \wedge \theta_n) \equiv \neg\theta_1 \vee \cdots \vee \neg\theta_n$
    - $\neg(\theta_1 \vee \cdots \vee \theta_n) \equiv \neg\theta_1 \wedge \cdots \wedge \neg\theta_n$
  - negace rovnosti / nerovnosti:
    - $\neg(y = d) \equiv y \neq d$
    - $\neg(y \neq d) \equiv y = d$

(Příklad)

(Ukázka)

Ekvivalence podmínek

- Dvě podmínky θ_1, θ_2 nad stejným schématem S jsou **ekvivalentní**, pokud:
  - pro každý dokument c nad S platí:
    - c splňuje θ_1 **právě tehdy, když** c splňuje θ_2
- Značení: $\theta_1 \equiv \theta_2$

(Příklad)

Triviálně pravdivá podmínka

- Podmínka nad $\emptyset$, která je splněna v každém dokumentu
  - značíme 1
  - v JSON ji zapisujeme jako:
    `{}`

Určující vlastnost podmínky

- Každé podmínce θ přiřadíme *určující vlastnost*
  - $V_\theta(c) \equiv$ „dokument c splňuje θ“
- Příklady
  - θ: `produkt = "Televize Tesla"`
  - $V_\theta(c)$: „`c(produkt)` je rovno `"Televize Tesla"`“

Operátory v podmínkách MongoDB

- **Porovnávací operátory**
  - `$eq`, `$ne` - rovnost / nerovnost
  - `$gt`, `$gte`, `$lt`, `$lte` - porovnání
  - `$in`, `$nin` - členství v množině
- **Logické operátory**
  - `$and` - konjunkce
  - `$or` - disjunkce
  - `$nor` - negace disjunkce (NOT OR)
  - `$not` - negace operátoru na jednom poli
- **Elementové operátory**,
- **Operátory pro pole**,
- **Regulární výrazy**,
- ...

Restrikce kolekce

- Nechť C je kolekce a θ je podmínka nad schématem S

$$\sigma_\theta(C) = \{c \in C \mid c \text{ splňuje } \theta\}$$

- Pokud je kolekce C určena vlastností $V_C(c)$, pak $\sigma_\theta(C)$ má vlastnost:
  - „$V_C(c)$ a $V_\theta(c)$“
- Pro každou kolekci C platí:
  - $\sigma_1(C) = C$
- V MongoDB

```
db.kolekce.find(podmínka)
```

(Ukázky)

Projekce

- Nechť
  - c je dokument nad schématem S_1
  - C je kolekce nad schématem S_1
  - $S_2 \subseteq S_1$
- **Projekce dokumentu** c na S_2 je

$$c|_{Y(S_2)} = \{\langle y, d\rangle \in c \mid y \in Y(S_2)\}$$

- **Projekce kolekce** C na S_2 je

$$\pi_{S_2}(C) = \{c|_{Y(S_2)} \mid c \in C\}$$

- V MongoDB

```
db.kolekce.find(podmínka, S2)
```

(Ukázka)

Mazání dokumentů

- V MongoDB lze dokumenty na základě podmínky i mazat
- Nechť
  - `kolekce` je kolekce nad schématem S
  - P je podmínka nad schématem S
- Pak výraz

```
db.kolekce.deleteMany(P)
```

- smaže z kolekce `kolekce` všechny dokumenty pro které platí podmínka P
- nová hodnota $C = \sigma_{\neg\theta}(C)$

(Ukázka)

Součin kolekcí

- Nechť
  - S_1 a S_2 jsou disjunktní schémata
  - c_1 je dokument na S_1
  - c_2 je dokument na S_2
  - C_1 je kolekce na S_1
  - C_2 je kolekce na S_2
- Pak
  - **Spojení dokumentů** c_1 a c_2 je dokument $c_1c_2 = c_1 \cup c_2$ na $S_1 \cup S_2$.
  - **Součin kolekcí** C_1 a C_2 je kolekce $C_1 \times C_2 = \{c_1c_2 \mid c_1 \in C_1, c_2 \in C_2\}$

Změna dokumentů

- Nechť:
  - `kolekce` je kolekční proměnná na S
  - P je podmínka na S
  - c je dokument na $S' \subseteq S$
  - `_id` $\notin Y(S')$
- Pak příkaz

```
db.kolekce.updateMany(P, { $set: c })
```

- nastaví hodnotu proměnné `kolekce` na $(\pi_{S-S'}(\sigma_P(C)) \times \{c\}) \cup \sigma_{\neg P}(C)$, kde C je původní hodnota `kolekce`

(Ukázka)

Hluboké dokumenty

- Práce s vnořenými strukturami: dokumenty obsahující kolekce dalších dokumentů.
- Schéma S je **podschéma** schématu S' pokud
  - $Y(S) \subseteq Y(S')$
  - pro všechny atributy $y \in S$ platí $S(y)$ je podschéma $S'(y)$
- Příklad schématu:
  - `{ _id, název, vlastnosti.{výška, váha, barva}, cena }`
- Charakteristické vlastnosti:
  - Produkt `_id` se nazývá `název` a stojí `cena`.
  - Je vysoký `vlastnosti.výška`, těžký `vlastnosti.váha` a má barvu `vlastnosti.barva`.
- Podschémata
  - `{ _id, vlastnosti.{výška, váha} , cena }`
  - `{ _id, vlastnosti.{výška, váha, barva}}`
  - ...

Hluboké podmínky

- Podmínka tvaru $y.(\Theta)$ nad $\{y.S\}$
  - je splněna, pokud v kolekci $c(y)$ existuje dokument splňující Θ.
- Podmínky nad podschématem jsou podmínkami i pro nadřazené schéma.
- Zápis v MongoDB

```
{ "y": { "$elemMatch": podmínka } }
```

(Ukázka)

Negace hluboké podmínky

- Podmínka tvaru $\neg y.(\Theta)$ nad $\{y.S\}$
- Význam
  - dokument nesmí obsahovat žádný prvek splňující Θ.
- Zápis v MongoDB

```
{ "y": { "$not": { "$elemMatch": Podmínka } } }
```

(Ukázka)

Hluboké projekce

- Projekce na podschéma:
  - ponechá atomické atributy,
  - kolekční atributy projektuje rekurzivně.
- **Projekce dokumentu** t nad S_1 na S_2 je dokument $t(S_2)$ definován jako
  - $$\{\langle y, d\rangle \in t \mid y \text{ je atomický atribut}\}$$
    $$\cup$$
    $$\{\langle y, \pi_{S'}(d)\rangle \mid \langle y, d\rangle \in t \text{ a } \langle y, S'\rangle \in S_2 \text{ a } y \text{ je kolekční atribut}\}$$
- **Projekce kolekce** C nad S_1 na podschéma S_2 je kolekce: $\pi_{S_2}(C) = \{t(S_2) \mid t \in C\}$
  - kde $t(S_2)$ je projekce dokumentu t nad S_1 na S_2.

(Příklad)

(Ukázka)

Nastavení hodnoty atributu

- Pro dokument c nad schématem S, atribut $y \in Y(S)$ a hodnotu d
- Výsledkem je dokument

$$c(y/d) = \pi_{S'}(c) \cup \{\langle y, d \rangle\},$$

kde

$$S' = \{\langle y', S'' \rangle \in S \mid y' \neq y\}.$$

(Příklad)

Hluboké změny – $pull

- Odstranění prvků z vybraného kolekčního atributu:
- Příkaz

```
db.kolekce.updateMany(Podmínka1, {
  "$pull": { "y": Podmínka2 }
})
```

nastaví hodnotu kolekční proměnné `kolekce` na

$$\sigma_{\neg\Theta_1}(C) \cup \left\{ c\Big(y/\sigma_{\neg\Theta_2}(c(y))\Big) \;\middle|\; c \in \sigma_{\Theta_1}(C) \right\}.$$

(Ukázka)

Hluboké změny – $push + $each

- Přidání prvků do vybraných kolekčních atributů
- Příkaz

```
db.kolekce.updateMany(Podmínka, {
  "$push": { "y": { "$each": C' } }
})
```

nastaví hodnotu kolekční proměnné `kolekce` na

$$\sigma_{\neg\Theta}(C) \cup \left\{ c\Big(y/(c(y) \cup C')\Big) \;\middle|\; c \in \sigma_{\Theta}(C) \right\}.$$

Kde navíc požadujeme $c(y) \cap C' = \emptyset$.

(Ukázka)

Schémata v MongoDB

- V MongoDB implicitně máme dynamické schéma
- Lze ale vynutit, aby kolekce měla schéma dle požadavků situace
- Deklarujeme tzv. JSON schéma
- Typy atributů:
  - `int`
  - `string`
  - `...`
  - `object`
  - `array`
- Syntaxe

```
{ "bsonType": "string" }
```

Schéma dokumentu

- Objektové atributy mají ve schématu následující základní vlastnosti
  - `bsonType` - u objektů vždy `object`
  - `required` - názvy povinných atributů
  - `properties` - popis vnitřní struktury atributů (schémat)
  - existuje mnoho dalších (validace hodnot, komentáře, ...)
- Syntaxe

```
{
  "bsonType": "object",
  "required": [...],
  "properties": { ... }
}
```

(Ukázka)

Kolekční atribut ve schématu

- Kolekční atributy jsou typu `array`
- Klíček `items` popisuje schémata dokumentů v kolekci
- Syntaxe

```
{
  "y": {
   "bsonType": "array",
   "items": S
  }
}
```

(Ukázka)

Validátor schématu

- Chceme-li vytvořit kolekci s vynuceným schématem, použijeme **validátor**.
- Zadáme jej jednoduše požadovaných schématem.
- Syntaxe

```
db.createCollection("role", {
  validator: { $jsonSchema: S }
})
```

(Ukázka)

Roura

- Roura = posloupnost fází (operací), které transformují kolekci dokumentů.
- Každá fáze vezme kolekci a vrátí novou kolekci.
- Silný nástroj pro složité dotazy.
- Umožňuje:
  - restrikci (filtr, `$match`)
  - projekci (výběr polí, `$project`),
  - shlukování (agregaci, `$group`)
  - vyhledání mezi kolekcemi (`$lookup`)
  - `$limit`, `$sort`, `$sample`, `$search`, ...
- Formálněji:
  - Pro posloupnost fází O_i a posloupnost kolekcí C_i nad schématy S_i ($i \in \{1, ..., n\}$),
  - kde platí $C_{i+1} = O_i(C_i)$
  - Použitím roury získáme kolekcí C_{n+1}

Vybrané fáze roury I

1. **Restrikce**

   ```
   { "$match": podmínka }
   ```

2. **Projekce**

   ```
   { "$project": schéma }
   ```

3. **Agregace a shlukování**

```
{
  "$group": {
    "_id": "$atribut1",
    "averageAge": { "$operace": "$atribut2" }
  }
}
```

Vybrané fáze roury 2

4. **Vyhledání / join**

```
{
  "$lookup": {
    "from": "kolekce",
    "localField": "atribut1",
    "foreignField": "atribut2",
    "as": "atribut3"
  }
}
```

Použití roury

- Jednotlivé fáze specifikujeme ve výrazu `aggregate`.
- Vykonají se v uvedeném pořadí

```
db.kolekce.aggregate([ O1, ..., On])
```

(Ukázky)

Shrnutí

- Dokumentové databáze jako zástupci NoSQL přístupu
  - Související data uchovávána v dokumentech
  - Dokumenty jsou strukturované (do hloubky)
  - Základní pojmy dokument a kolekce
  - Nepovinná schémata
- MongoDB jako zástupce dokumentových databází
  - Dokumenty uloženy jako JSON/BSON
  - Bohaté možnosti manipulace s dokumenty a kolekcemi
    - restrikce, projekce
    - vytváření, mazání a úpravy
    - hluboké změny a roury
  - Schémata lze vynutit validátory
- Mnoho směrů k dalšímu čtení
  - Převod ER modelu (viz materiály dr. Laštovičky)
  - Vhodná struktura dokumentové databáze
    - Např. kdy data mít v jednom dokumentu a kdy se raději odkazovat?