Verzovanie databázy

Projekt 1

Prednáška 3

Flyway

Motivácia

- Vytvorili sme si anotované entity a nechali sme z nich generovať DB schému (JPA DDL – Data Definition Language)
  - SQL tabuľky bez písania SQL
- Úprava entitnej triedy
  - Asociovaná tabuľka už nemusí byť kompatibilná
  - Aplikácia už bude mať problém
- Počas (raného) vývoja to nie je problém
  - Pôjdem do svojej DB a upravím tabuľku ručne
  - Alebo zmažem celú DB (**docker compose down**) a nechám JPA pregenerovať schému

Motivácia

Čo ak mám aplikáciu už reálne nasadenú?

- JPA generovanie nebude fungovať
- Zmazať produkčnú DB a znova vytvoriť nepripadá do úvahy
- Musím ručne upraviť tabuľku napr. cez ALTER TABLE

Čo ak mám aplikáciu nasadenú 20x?

- Mám niekoľko dev nasadení, stage nasadení, niekoľko produkčných nasadení u klientov
- Musím ručne upraviť každé jedno nasadenie
- Pravdepodnosť pomýlenia sa je vysoká
- Zbláznim sa
- Či?...

Inicializácia DB

Na počiatku bola tabuľka

Nechať Docker inicializovať DB

Na PAZ1c ste videli…

- Mali sme súbor **init.sql**
- Do neho sme skopírovali SQL vygenerovaný cez ER diagram v MySQL Workbenchi
- Nastavili sme MySQL Docker kontajner aby pri prvom spustení načítal tento súbor

```
volumes:
  - ./init.sql:/docker-entrypoint-initdb.d/init.sql
```

- Následne naša aplikácia pri spustení už mala pripravené tabuľky

JPA DDL generovanie

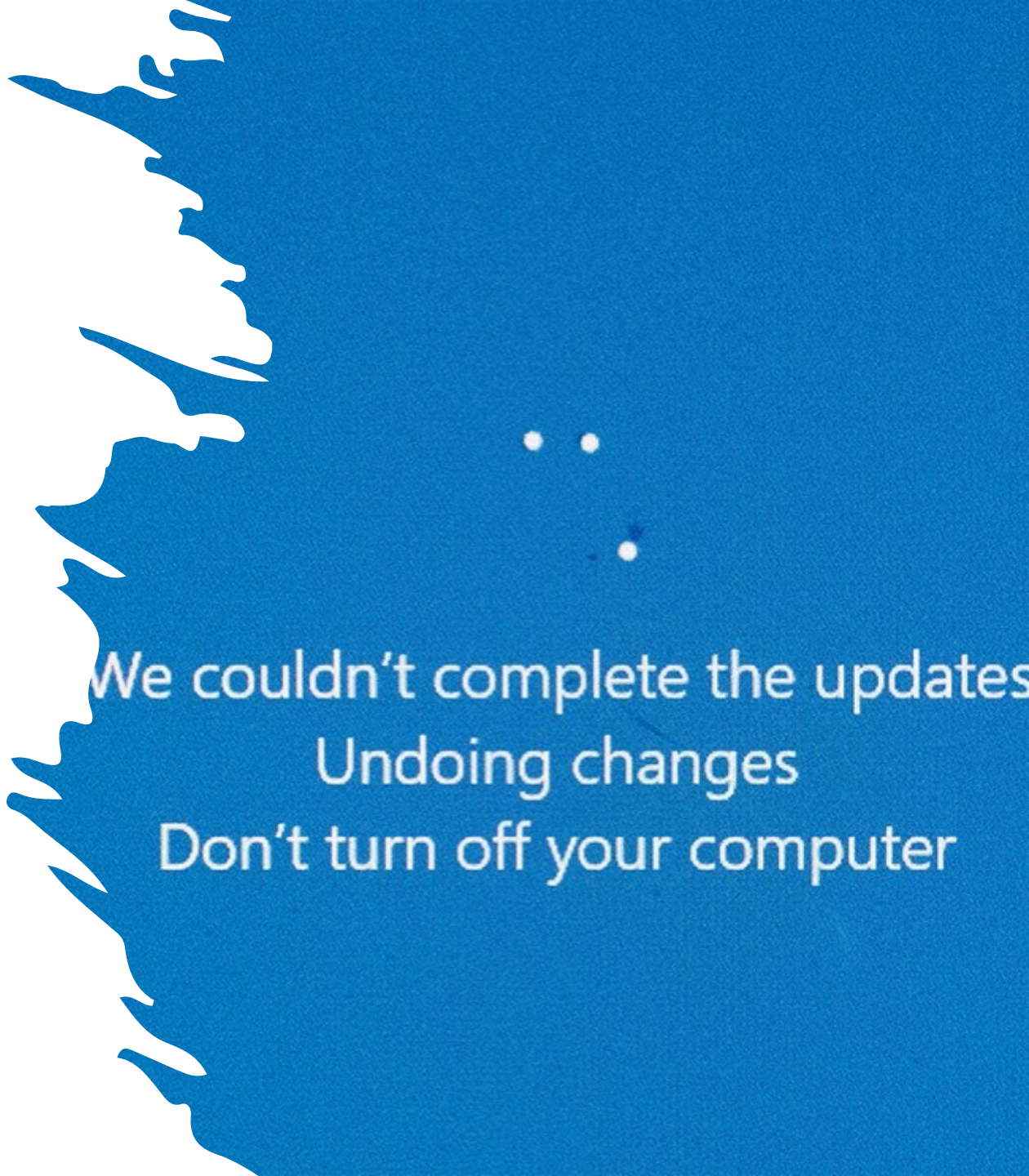
Na minulej prednáške ste videli…

- MySQL kontajner sa spustil prázdny
- Naša aplikácia vygenerovala tabuľky sama
  - Stačili jej anotácie v entitných triedach
  - Nenapísali sme ani riadok SQL

- DDL = Data Definition Language
  - Všetky CREATE príkazy atď.

Aktualizácia DB

Tu končia kariéry…



Príklad – zmena typu stĺpca

Teraz máme

```
@Data
@Entity
public class User {
    ...
    private int sex;
    ...
}
```

Chceme

```
@Data
@Entity
public class User {
    ...
    @Enumerated(EnumType.STRING)
    private Gender gender;
    ...

    public enum Gender {
        MALE,
        FEMALE,
        OTHER,
        SECRET,
    }
}
```

Aktualizácia aplikácie

- Mám server s mojou bežiacou appkou a DB
- Chcem nasadiť novú verziu appky
  - Mám nové entity, upravené iné entity, zmazané ďalšie entity…
  - Potrebujem naraz s aplikáciou aktualizovať aj DB
- (PAZ1c) SQL skripty v Dockeri sú mi na nič
  - Produkčná DB asi nebude bežať v Dockeri spolu s mojou appkou
  - Rieši iba naplnenie prázdnej DB,
    nie modifikáciu existujúcej DB
- (PRO1a) JPA DDL generovanie
  - Zvláda pridávať nové veci do schémy, ale má problém s modifikáciou a mazaním
  - Veľmi rudimentárny nástroj
  - **Nezvláda pracovať s dátami v tabuľkách**

Čo mám robiť?

- Lokálne prostredie
  - Ak appku spúšťam u seba, tak môžem zmazať DB a nechať ju pregenerovať nanovo
  - Mal by som mať ale (polo)automatizované vkladanie testovacích dát

- Produkčné prostredie
  - Len si skúste zmazať DB...
    - V normálnej firme tam programátori ani nemajú prístup

Verzovanie DB

- S vydaním novej verzie aplikácie vydáme aj novú verziu našej databázy (pozor, nie verziu RDBMS)
  - Napr. entrance v1.1
- POZOR, databáza ale obsahuje dáta
  - Nová verzia DB nemení len „SQL zdroják“ (CREATE TABLE...), ale mení aj dáta.
  - Každá nasadená inštancia má iné dáta v tabuľkách
- Dáta potrebujeme „migrovať“ z pôvodnej verzie DB do novej verzie DB
  - Napr. zmena dátového typu stĺpca
  - Napr. rozbitie tabuľky do viacerých tabuliek

Príklad – migrácia databázy

- Potrebujeme **migrovať** databázu na nové entity
- Teda pre spustením novej verzie appky musíme zabezpečiť:

```
ALTER TABLE `users`
   ADD COLUMN `gender` ENUM('MALE', 'FEMALE', 'OTHER‘, 'SECRET');

UPDATE `users` SET `gender` = CASE `sex`
   WHEN 0 THEN 'MALE’
   WHEN 1 THEN 'FEMALE’
   ELSE NULL END;

ALTER TABLE `users` DROP COLUMN `sex`;
```

- Ako to najlepšie urobiť? (Ručne? Haha.)

DB verzovacie nástroje v Jave

Flyway

- Jednoduchší
- Free aj platený
- Píšeme priamo SQL pre danú RDBMS
- Java knižnica, CLI, Maven/Gradle plugin, integrácia so Spring Boot

Liquibase

- … (asi) všetko čo Flyway
- Okrem SQL vieme písať aj JSON a XML
  - Ak potrebujeme podporovať viac RDBMS (MySQL, SQL Server, …)

Čo si vybrať?

- Obe možnosti sú široko používané
- Pre obrie projekty je asi lepší Liquibase
  - Viac fíčur, vo free aj v platenej verzii
  - Verzovanie pre viac RDBMS naraz
- Väčšinou ale stačí Flyway kvôli jednoduchosti
- My si ukážeme Flyway

Konfigurujeme Maven

pom.xml

```
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-mysql</artifactId>
</dependency>
```

Konfigurujeme Spring

src/main/resources/application.properties

```
spring.jpa.generate-ddl=true

spring.flyway.baseline-on-migrate=true
spring.flyway.locations=classpath:db/migration
```

Spíšeme si inicializačný SQL skript

- Súčasný stav našej DB prehlasíme za verziu 1
- Vygenerujeme si DDL skript napr. cez IntelliJ IDEA
- V **src/main/resources/db/migration** vytvoríme súbor V1__Init.sql
  - To čo je žltým musíme zadať presne takto
- Skopírujeme DDL do súboru
- Pre každý príkaz CREATE doplníme IF NOT EXISTS

(Voliteľne) Upravíme anotácie v entitách

- Tabuľky už nebudú vytvárané cez JPA
- Je vhodné manuálne nastaviť mapovanie názvov entít a názvov tabuliek
- Teraz to nemusíme robiť, no budúce verzie DB nemusia následovať názvoslovie JPA DDL

  …ktoré je trocha divné a nejde podľa bežnej konvencie
- K entitám dodáme anotáciu @Table, napr. @Table(name = "user")
- Vzťahom @ManyToMany dodáme na strane vlastníka anotáciu @JoinTable, napr.

```
@JoinTable(
    name = "user_card_readers",
    joinColumns = @JoinColumn(name = "users_id"),
    inverseJoinColumns = @JoinColumn(name = "card_readers_id")
)
```

Spravíme migračný skript

- V entitnej triede už máme inš. premennú gender, no v tabuľke ešte stále máme stĺpec sex
- Navyše majú iné dátové typy
- V **src/main/resources/db/migration** vytvoríme súbor V2__Replace_sex_with_gender.sql, alebo V1_1__Replace_sex_with_gender.sql
  - Nová verzia DB musí byť vyššia ako súčasná
- Dáme tam

```
ALTER TABLE `users`
   ADD COLUMN `gender` ENUM('MALE', 'FEMALE', 'OTHER', 'SECRET');

UPDATE `users` SET `gender` = CASE `sex`
   WHEN 0 THEN 'MALE'
   WHEN 1 THEN 'FEMALE'
   ELSE NULL END;

ALTER TABLE `users` DROP COLUMN `sex`;
```

Voilà

- Spring Boot teraz bude automaticky verzovať našu DB
- Pri spustení appky si automaticky
  - Zistí aktuálny stav (verziu) našej DB
  - Spustí všetky migračné skripty s vyššou verziou (podľa názvu skriptu V1, V2)
- Všimnite si novú tabuľku flyway_schema_history
- Tam sú uložené metadáta k verzovaniu a migráciám

Čo ak migrácia zlyhá

- Pr.: V migračnom SQL skripte máme chybu
- Flyway si poznačí, že táto verzia je zlá
- Fixneme chybu v SQL skripte
- Zmažeme riadok z flyway_schema_history, kde success=0
  - Alebo si nainštalujeme Flyway ako CLI appku, alebo ako Maven plugin
  - A použijeme príkaz `flyway repair`
- Opravíme migračný skript a prípadne aj neúplné zmeny v DB
- Spustíme Spring Boot (alebo `flyway migrate`)

Pamätajme

- Pri verzovanej DB pozor na ručné úpravy (nasadenej) DB schémy
  - GUI DB klientov používajte prípadne iba na prezeranie
- Raz **aplikovaný** migračný SQL skript už **nesmieme** meniť
  - Flyway bude plakať a hlásiť chybné stavy
- Ak korektne aplikujeme nový migračný skript a potom sa nám znepáči...
  - Najjednoduchšie je vytvoriť nový skript a starý nechať tak
  - Ak naše zmeny ešte neboli merdžnuté v gite, alebo inak aplikované v DB mimo nášho počítača, tak ešte môžeme vymazať príslušný riadok z flyway_schema_history, ručne vrátiť naše zmeny v DB do pôvodného stavu a prepísať príslušný migračný skript.
    - Ak chceme mať na jeden merge request max 1 nový migračný skript (...lebo šéf povedal)
    - S plateným Flyway je to troška jednoduchšie

Pamätajme

- **Nesmieme** mať **viac** migračných skriptov s **rovnakou** verziou
- Toto sa však často stáva vo väčších projektoch
  - V main vetve máme posledný skript s prefixom V4_1
  - Dvaja ľudia začnú naraz robiť na dvoch rôznych fíčurách
    - Pomenujú si svoje nové skripty V4_2...
  - Každá vetva samostatne ide spustiť, ale ak sa merdžnú do main, tak ten sa pri spustení rozbije
- Existujú stratégie ako tomu zabrániť
  - Tesne pred merdžovaním vetvy do main skontrolujeme, či medzitým nepribudol nový migračný skript s rovnakým prefixom V...
  - Ak áno, tak svoj skript premenujeme a vetvu rýchlo po otestovaní merdžneme (je dobré mať CI s automatickými testami)
  - Alebo budeme ako verzie používať timestampy a povolíme outOfOrder migrácie
    - Ale tam sú zase iné riziká
  - Alebo si naskriptujeme do CI prečíslovanie migračných skriptov
- **Nedávajte** do migračných skriptov **testovacie dáta**
  - Žiadne `INSERT INTO user` atď.
  - Je dovolené vkladať dáta do číselníkov (tabuľka so statickými riadkami – zoznam krajín, miest, kategórie, atď.)

Ďakujem za pozornosť

V.K. volá domov