Unix Motion

Zápisnica 30.9.2016

Jaroslav Ištok
Katarína Fabianová
Dušan Suja
Jerguš Adamec

September 30, 2016

ewkhfwe

1 Stručný popis zadania

Kompletné zadanie projektu možno nájsť na nasledujúcej stránke: `http://dai.fmph.uniba.sk/~simko/db2/projekt.html`

- Aplikácia musí obsahovať CRUD operácie, t.j. vytvorenie, zobrazenie, aktualizovanie a zmazanie, pre vybrané množiny entit.
- Aplikácia nesmie obsahovať výlučne CRUD operácie.
- Aplikácia musí obsahovať zaujímavú doménovú logiku, ako napr. úročenie peňazí na bankových účtoch na konci každého mesiaca, prevod peňazí medzi účtami, aktualizovanie počtu produktov pri vytvorení novej objednávky a pod.
- Aplikácia musí produkovať zaujímavé reporty (správy, štatistiky) ako napr. počet otvorených účtov za jednotlivé štvrťroky a pod. Po vypnutí aplikácie nemôže dôjsť k strate údajov. Tie sa musia ukladať do PostgreSQL databázy.
- Aplikácia musí vhodne využívať transakcie.
- Aplikácia musí vhodne využívať indexy a optimalizované SQL dopyty.
- Obe verzie aplikácie musia implementovať tú istú funkcionalitu.

2 Použité technológie a prostriedky

Pri implementácii projektu som použil nasledujúce technológie:

2.1 Java

Multiplatformový jazyk, ktorý beží na virtuálnom stroji. Patrí medzi najpoužívanejšie programátorské jazyky. Je staticky typovaný, vhodný na veľké projekty, má čistú syntax a bohatý ecosystém knižníc. Používa automatickú správu pamäte. Množstvo podobných aplikácii, je naprogramovaných v Jave, čo je jeden z hlavných dôvodov, prečo som si zvolil práve Javu.

2.2 InteliJIDEA

IDE od firmy JetBrains. Patrí medzi v súčastnosti najpokročilejšie IDE pre Javu. Uľahčuje prácu s databázou, vie kontrolovať syntax SQL dopytov priamo v kóde.

2.3 JPA

JPA, teda Java Persistence API, je rozhranie, ktoré je súčasťou JavaEE (Enterprise edition). Hovorí o tom, ako má vyzerať implementácia data mapperu v Jave. Existuje niekoľko implementácii tohto rozhrania. Napríklad: Eclipse Link alebo Hibernate. V mojej aplikácii som si zvolil Eclipse Link implementáciu. Umožňuje ORM (objektovo relačné mapovanie),

2.4 PostgreSQL

PostgreSQL je voľne šíriteľný objektovo-relačný databázový systém (systém riadenia báz dát), uvoľnený pod flexibilnou licenciou BSD. Ponúka alternatívu k ostatným voľne šíriteľným databázovým systémom (ako sú MySQL, Firebird, MaxDB a iné), ako aj k proprietárnym (akými sú napr. Oracle, DB2 od IBM či Microsoft SQL Server). Podľa mnohých databázových odborníkov je v súčasnosti PostgreSQL najvyspelejší a najsofistikovanejší voľne šíriteľný systém riadenia báz dát. Tento systém mi poskytol všetku potrebnú funkcionalitu na implementáciu aplikácie.

2.5 GIT

Git je distribuovaný systém riadenia revízií, ktorý vytvoril Linus Torvalds. Nemal by sa zamieňať s programom GIT (GNU Interactive Tools), správcom súborov na spôsob programu Norton Commander, ktorý vytvorili Tudor Hulubei a Andrei Pitis. Návrh Gitu bol inšpirovaný systémami BitKeeper a Monotone. Git bol pôvodne navrhnutý len ako nízkoúrovňový engine, ktorý by ostatní mohli využiť na napísanie vlastných používateľských rozhraní ako sú Cogito alebo StGIT. Avšak základný Git projekt sa odvtedy pretransformoval do kompletného systému na správu revízií, ktorý sa dá používať priamo. Git je teraz používaný v niekoľkých dôležitých softvérových projektoch, z ktorých najznámejším je jadro Linuxu. V súčastnosti je to jeden z najpoužívanejších verziovacích systémov, medzi najužitočnejšie vlastnosti patrí vetvenie jednotlivých revízii. Git ponúka všetko potrebné, čo som potreboval pri implementácii projektu.

2.6 Visual Paradigm

Profesionálny program na dátové modelovanie. Používal som ho na vytvorenie entitno relačného modelu tabuliek a následne aj na vytvorenie modelu výslednej databázy, ktorý vznikou tranformáciou entitno relačného modelu.

2.7 Scene Builder

Program na vytvaranie GUI aplikacii v Jave. Zjednodušuje vytváranie GUI, pretože je možné ho jendoducho naklikať, namiesto zložitého písania statického kódu.

2.8 JavaFX

GUI framework v Jave. Existuje v ňom množstvo rôznych prístupov ako vytvárať GUI aplikácie, ja som si vhľadom na použitie Scene Buildera zvolil FXML súbory a MVC prístup, ktorý je vhodný na vytváranie zložitejších GUI aplikácii.

2.9 JavaDoc

Nástroj na vygenerovanie dokumentácie k projektu z dokumentačných komentárov.

3 Popis riešenia

- Začal som návrhom databázy. Navrhol som jednotlivé entity, a vytvoril som z nich entitno relačný model. Tento model som počas riešenia projektu veľa krát menil a upravoval. Vystihuje pekne vzťahy medzi jednotlivými entitami a zobrazuje aké dátové atribúty budú mať výsledné tabuľky, ktoré vzniknú transformáciou na databázový model.
- Druhým krokom bola tranformácia entitno relačného modelu na databázový model. Je to automatizovaná tranformácia, preto to nebol problém. Základ transformácie je vhodné pridanie cudzích kľúčov do jednotlivých tabuliek, ktoré reprezentujú entity z relačného modelu, na základe kardinality vzťahov.
- Pokračoval som vytvorením projektu a spojazdnením všetkých potrebných vecí, najmä JPA, ktoré nebolo jednoduché spojazdniť. Potom som si vytvoril package pre obe verzie aplikácie a začal som s implementáciou. Celý projekt si pre svoje vlastné potreby verzujem v GIT-e.

Vytvoril som si "data row gateways" a JPA entity triedy pre všetky tabuľky v databáze. Samotnú databázu som v podstate vytváral na serveri počas vytvárania Entít pre JPA.

- Keď som mal základ projektu hotový pustil som sa do návrhu GUI. Najskôr som si na papier navrhol rozloženie prvkov na jednotlivych oknach. V aplikácii je niekoľko okien. Hlavné okno, okno na pridávanie nového lieku, okno na pridávanie, okno v ktorom sa budú zobrazovať statistiky atd. Všetky tieto okna som si navrhol v Scene Builderi a následne som ich vložil do mojej aplikácie a vytvoril k nim potrebné controllery.

- Ďalej som pokračoval v programovaní logiky aplikácie. Pridával som potrebné metódy a pomaly som začal pridávať funkcionalitu do samotného GUI.

- Ďalším krokom bolo vytvoriť vhodné indexy a triggery do databázy. Pozrel som sa, ktoré operácie používam vo svojej aplikácii načastejšie a tie som sa snažil optimalizovať. Triggery som pridal na niektoré špeciálne udalosti, napríklad, ak sa updatuje cena nejakeho lieku, tak sa povodna cena vlozi zaznamena do tabulky, ktora uchovava historiu cien daneho lieku. Každý mesiac sa spustí trigger, ktorý náhodne zmení maržu lekárne, čiže je zisk, čo sa premietne do cin všetkých liekov. (Taká mini simulácia reálneho sveta). Jedným s posledných krokov bolo vytvoriť create scripty pre celú databázu vrátane vzorky testovacích dát a všetkých indexov a triggerov a uložiť ich do súborov, tak aby boli spustiteľné na ľubovoľnom serveri, na ktorom beží PostgreSQL server a ktore vytvoria pozadovanu databzu.

- Na záver som vygeneroval pomocou JavaDoc nástroja dokumentáciu k celému projektu, ktorú som písal v priebežne počas programovania.

- Nakoniec som vytvoril jednoduchú webovú stránku s linkami na stiahnutie projektu a dokumentácie. Pri jej tvorbe som použil šablónu bootstrap.

4 Veci, ktoré som sa pri navrhovani tohto projektu naučil

- Praca s databazou v Jave

- Množstvo návrhových vzorov, ktoré budem potrebovať v praxiô

- JPA Api a princípy fungovania data mapperu, výhody a nevýhody objektovo orientovaného prístupu.
- Transaction script a výhody a navýhody tohto prístupu.

5 Záver

Na tomto projekte som sa naučil množstvo užitočných vecí, ktoré budem môcť priamo využiť v praxi. Je to napríklad Java Persistence API, ktoré je používané vo veľkom množstve aplikácii. Osvojil som si aj niektoré návrhové vzory, ako napríklad Connection Pool, Proxy, a iné. Kedže som projekt musel implementovať dvomi rôznymi spôsobmi, vedel som ich na konci porovnať a zistil som aké výhody a nevýhody majú jednotlivé prístupy.