Začal by som s úvodom do tejto problematiky. Dnešné databázy používajú predovšetkým binárne formáty, pretože umožňujú efektívne ukladanie, vyhľadávanie a manipuláciu s údajmi. Hoci existujú textové databázy, v porovnaní s binárnymi databázami majú obmedzenia z hľadiska výkonu, indexovania a škálovateľnosti. Textové databázy sú jednoduchšie, prenosné a dobre spolupracujú so systémami riadenia verzií, ale sú pomalšie, chýba im indexovanie a majú problémy s veľkými objemami údajov.

Existujú však databázy, ktoré používajú formáty JSON a XML, avšak typicky údaje nie sú uložené v týchto formátoch na súborovom systéme. Mojou úlohou bolo vytvoriť si JDBC ovládač, ktorý využíva súborový systém na uloženie a prístup k údajom na súborovom systéme, v formáte, ktorý je čitateľné pre človeka.

V analytickej časti som sa venoval oblastiam, ako je parsovanie SQL príkazov, spracovanie a uloženie údajov, ľudsky čitateľné formáty, a transakcie v rôznych databázových systémoch.

Výsledkom mojej práce je JDBC ovládač, ktorý parsuje SQL príkazy pomocou bezkontextovej gramatiky, spracováva údaje v pamäti a podporuje dva najpoužívanejšie formáty JSON a XML, do ktorých perzistuje údaje. Ovládač používa na overovanie integrity údajov schémy JSON Schema a XSD. Aby bolo jasné, v našom prípade je databáza priečinok, v ktorej sú tabuľky uložené v súboroch XML alebo JSON. Metadáta o tabuľkách sú uložené v schémach. Ako nástroj na verzovanie databázy sme implementovali GIT, ktorý možno voliteľne použiť v ovládači. Viac o tom budem hovoriť neskôr.

Na obrázke vidíte high level komponent diagram, ktorý znázorňuje tie najdôležitejšie komponenty ovládača.

Implementáciu sme rozdelili na nasledujúce moduly:

• Statement parser - je zodpovedný za parsovanie SQL príkazov a vytváranie DTO z nich

• Statement DTO Cache – je zodpovedný na uloženie parsovaných SQL príkazov do pamäti

• Statement service manager – zahŕňa všetky servisy, ktoré riešia zmeny v pamäte

• Semantic validator – je zodpovedný za validáciu príkazov z hľadiska sémantiky

• Reader – je zodpovedný za čítanie údajov

• Writer – je zodpovedný za zápis údajov

• Transaction manager – je zodpovedný za spracovanie transakcií

• Database manager – je zodpovedný za vytvorenie novej databázy a načítanie existujúcej databázy

• Schema validator – je zodpovedný za validáciu tabuľky voči schéme

Pomocou rozdelenia sme dosiahli izoláciu modulov, nezávislosť medzi nimi.

Parsovanie SQL príkazov sme riešili pomocou nástroja ANTLR, ktorý dokáže vygenerovať parser z danej gramatiky. Ovládač používa odvodenú syntax SQLite, inými slovami, implementovali sme menšiu množinu gramatiky SQLite. Na obrázku je znázornené, ako ANTLR vygeneroval strom parsovania z príkazu SQL. Zo stromu extrahujeme tieto údaje, vytvoríme z nich objekt prenosu údajov a odošleme ho na spracovanie.

Ovládač využíva princíp databáza v pamäti, čo znamená, že údaje budeme spracovávať v pamäti, kým sa nezapíšu na disk. Tabuľky sa načítavajú do pamäte len v prípade potreby, takzvaným "lenivým" spôsobom. Je dôležité spomenúť, že tento spôsob spracovania údajov nie je optimálny, ak máme veľmi veľkú databázu, nad ktorou treba iterovať, pretože to znamená, že do pamäte treba načítať celú databázu. Tento princíp spracovania údajov nám umožnil rýchle spracovanie údajov pre stredne veľkú databázu. Na zlepšenie tohto problému by sa v budúcnosti mohol implementovať mechanizmus stránkovania.

Náš prístup k ukladaniu údajov spočíva v ukladaní údajov do pamäte, kým sa neodovzdajú. To nám umožňuje uchovávať nezapisované údaje v objektoch a zapísať ich všetky naraz, keď je transakcia komitovaná. V prípade, že je potrebné použiť metódu rollback, údaje sa môžu jednoducho znovu načítať z perzistovaných súborov do pamäte. Existuje však riziko poškodenia údajov, preto sme sa rozhodli implementovať git ako nástroj na verzovanie databázy, aby sme mohli databázu vrátiť do platného bodu.

Rozhodli sme sa pre úroveň izolácie transakcií "READ COMMITTED", aby sme zabezpečili konzistenciu medzi súbežnými operáciami. Toto nemenné nastavenie zabezpečuje, že všetky údaje načítané zo systému už boli zapísané do súborov, čím predchádza potenciálnym nekonzistentnostiam spôsobeným súbežnými transakciami. Aby sme ešte viac posilnili konzistenciu nášho ovládača a zabránili konfliktom z viacvláknových transakcií, rozhodli sme sa implementovať pesimistické zamykanie ako náš preferovaný mechanizmus zamykania. Ten zahŕňa získanie zámkov na údaje pred povolením prístupu, čím sa zabezpečí, že údaje môže v danom čase meniť len jedno vlákno.

Na overenie funkčnosti ovládača sme použili niekoľko metód. Po prvé, pre každú zložku nášho ovládača sme vytvorili jednotkové testy, ktoré testovali správnosť danej časti kódu. Testy boli napísané v testovacom rámci JUnit5 v kombinácii s Mockito. V jednotkových testoch sme dosiahli pokrytie kódu na úrovni 85 percent pre ovládač a 95 percent pre parser. Potom sme náš ovládač porovnali z hľadiska výkonu s ovládačom SQLite, pričom sme merali celkový čas behu operácií CRUD, v ktorom sa nám v niektorých prípadoch podarilo prekonať SQLite. Tieto testy nám tiež ukázali, že GIT nám síce prináša tieto výhody, ale má veľký negatívny výkonnostný odpor.   
Ako integračný test sme vytvorili jednoduchú aplikáciu, do ktorej sme importovali náš ovládač, ktorý by som teraz rád ukázal.

V aplikácií je možné vykonávať jednoduché CRUD operácie, respektíve vytvoriť si novú úlohu, označiť ju ako hotové alebo ju vymazať.

Ako návrhy na zlepšenie vidím rozšírenie syntaxu a funkcionality ovládača. Bolo by tiež možné implementovať iné ľudsky čitateľné formáty ako napríklad YAML alebo CSV. Testy nám preukázali na to, že git má veľký negatívny vplyv na výkon, preto by bolo možné nájsť inú alternatívu na verzovanie databázy. Ak rozhodneme zachovať git ako nástroj, bolo by možné implementovať podporu vzdialených repozitárov. Tiež by bolo možné implementovať algoritmus na riešenie stránkovania v ovládači.

1. Bolo by možné implementovať algoritmus stránkovania viacerým spôsobom. Prvé je, ak by sme rozdelili jednu tabuľku na viaceré súbory. Aktuálne v ovládači jedna tabuľka je jeden súbor. Mohli by sme rozdeliť tabuľku podľa počtu záznamov v jednej tabuľke. Týmto spôsobom by sme nemuseli načítať celú tabuľku.
2. Súčasná implementácia nepodporuje databázové procedúre ako CallableStatement. Všetky metódy, ktoré neboli implementované z JDBC API vychadzujú SQLFeatureNotSupportedException, ktorá označuje, že metóda nie je podporovaná ovládačom. Napriek tomu, bolo by možné rozšíriť parser, aby podporoval takéto operácie, ktoré by sme potom museli implementovať na strane ovládača.
3. To závisí od toho, do akej miery zasahujeme do repozitára. Vymazaním repozitára, respektíve git priečinka nám vyskočí výnimka v programe, a manuálne by sme museli znovu inicializovať repozitár, aby to bolo možné používať. Ak do git repozátára zasahujeme počas behu programu, tak zase môže stať, že údaje sa už nachádzajú v pamäti. V tomto prípade nám nevznikne žiadny problém, lebo súbor sa vždy prepíše pri odovzdaní zmien. Súčasná implementácia nepodporuje vzdialený git repozitár, čiže takzvané git konflikty sa nemôžu stať, lebo tiež by bolo zabezpečené, že pri viacvláknovom používaní iba jedno vlákno má prístup k repozitáru, čiže celý proces je synchronizovaný.