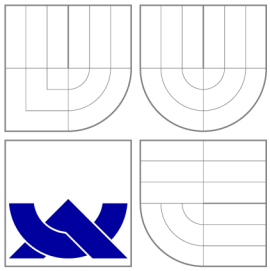
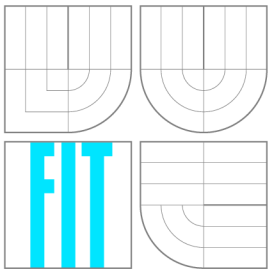


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

APLIKACE PRO DEMONSTRACI DOTAZOVACÍCH JAZYKŮ TEMPORÁLNÍCH DATABÁZÍ

A DEMO OF QUERY LANGUAGES FOR TEMPORAL DATABASES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. FILIP FEKIAČ

AUTHOR

VEDOUCÍ PRÁCE — RNDr. RYCHLÝ MAREK, Ph.D.

SUPERVISOR

BRNO 2015

Abstrakt

Cílem této práce je vytvorit aplikaci pro prepojení odlišních prístupu k temporalním datam. Hlavní myšlenkou je umožniť užívateli práci s použitím jazyka TSQL2 nebo ATSQL bez nutnosti zásahu užívatele do relačního databázového systému. Rovnež tak je součastí práce implementace rozhraní JDBC a demonstrační aplikace pro testovaní temporálních dotazu.

Abstract

The aim of this work is to create an application for interconnect diferrences among temporal databases. The main idea of this project is allow user work with languages TSQL2 and ATSQL without user intervention into relation database system. This work also contain implementation of interface JDBC and demonstration application for testing temporal queries.

Klíčová slova

temporální databáze, JDBC, TSQL2, ATSQL, TimeDB, Validtime

Keywords

temporal database, JDBC, TSQL2, ATSQL, TimeDB, Validtime

Citace

Filip Fekiač: Aplikace pro demonstraci dotazovacích jazyků temporálních databází, diplomová práce, Brno, FIT VUT v Brně, 2015

Aplikace pro demonstraci dotazovacích jazyků temporálních databází

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana RNDr. Mareka Rychlého Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Filip Fekiač

27. května 2015

Poděkování

Chcel by som poďakovať môjmu vedúcemu RNDr. Marekovi Rychlému Ph.D. za jeho čas a odbornú pomoc pri vypracovávaní tejto práce.

Obsah

Kapitola 1

Úvod

V súčasnej dobe je ukladanie a zbieranie informácií veľmi cenná oblasť informačných technológií. Mnoho spoločností alebo vedeckých skupín experimentuje s odlišnými formami ukladania dát. Snažia sa vytvoriť systémy bez čisto relačného spôsobu alebo dodaním rozšírení do tohoto spôsobu reprezentácie informácií. Hlavným dôvodom týchto experimentov je zvýšenie sily ukladaných dát a obohatenie informácií ukladaných bežným spôsobom. Rovnako tak môže byť motiváciou rastúci počet dát a nutnosť ich rýchleho spracovania na natívnej úrovni samotným databázovým serverom mimo aplikácie. Týmto prístupom sa eliminujú nepotrebné prenosy zo servera do aplikácie, čím sa samozrejme aj znižuje riziko odchytenia týchto prenosov a zvyšuje sa bezpečnosť celej architektúry aplikácie. V súčasnosti narastá motivácia na eliminovanie chýb programátora, čo sa dá dosiahnuť hlavne minimalizáciou príkazov zadávaných človekom. Toto všetko vedie k vývoju nových jazykov pre komunikáciu s databázovými serverami, ktoré už v sebe zahŕňajú rozšírenú podporu konkrétneho databázového servera. Veľké firmy, ktoré majú decentralizovanú infraštruktúru a ich hlavným príjmom je práca s dátami na celosvetovej úrovni sa snažia o preklenutie problému decentralizácie nasadzovaním rôznych post-relačných databázových systémov. Tieto majú samozrejme svoje obmedzenia. Nakoľko väčšina spomenutých technológií je ešte stále vo vývoji aj práca s nimi sa neustále mení. Súčasná doba so zvýšeným tlakom na maximalizáciu efektov všetkých procesov prináša nutnosť pre programátorov učiť sa nové postupy, aj keď tie predchádzajúce boli pre ich prácu dostatočné a už sú na ne zvyknutí, prípadne sú schopní s nimi pracovať rýchlejšie, efektívnejšie a kvalitnejšie. Problémom však je, keď sa spoločnosť rozhodne nasadiť inú technológiu z dôvodu prospechu väčšej skupine ľudí, už sa často nepozerá na potreby jednotlivca, ktorý je nútený prispôsobiť sa. S nástupom alternatívnych technológií sa začínajú objavovať aj nástroje pre ich spájanie a elimináciu rozdielov jednotlivých prístupov spoločne so zachovaním ich funkcionality a výhod, ktoré ponúkajú. Rovnako tak aj cieľom tejto diplomovej práce je vytvorenie nástroja odburávajúceho rozdiely medzi rôznymi temporálnymi databázovými systémami a ich dopytovacími jazykmi.

1.1 Štruktúra dokumentu

Dokument je vystavaný v logickom poradí tak ako prebiehal vývoj celej aplikácie. V kapitole 2, sú popísané teoretické znalosti potrebné k vývoju aplikácie, ale aj špecifické informácie potrebné k samotnému implementovaniu. Kapitola 3 obsahuje návrh vyvíjanej aplikácie s príslušnými diagramami, ktoré bolo potrebné vytvoriť pre kvalitnejšiu implementáciu a prípadné budúce rozšírenia. Kľúčové prvky z implementácie, ako napríklad spôsob komunikácie s databázou, výstavbu architektúry a riešenie problémov, ktoré nastali pri implementácií popisuje kapitola 4. Následný postup pri testovaní, určenie sady testovacích dát, popis testovania a vyhodnocovania výsledkov meraní opisuje kapitola 5. V závere dokumentu v kapitole 6 sú zhrnuté všetky prínosy aplikácie a navrhované rozšírenia pre prípadný budúci vývoj.

Kapitola 2

Analýza problematiky

Pri každej inžinierskej práci je potrebné vytvoriť životný cyklus vývoja a rovnako je to potrebné spraviť aj pri vývoji softvéru. Dodržiavanie takto navrhnutého postupu pre vopred stanovený požadovaný výsledok by malo zvýšiť pravdepodobnosť kvalitného produktu. Problémom je, že sa jedná o oblasť neustále sa meniacu a mnoho projektov vzniká a zaniká preto je na to potrebné prispôsobiť aj samotný návrh aplikácie. Pri analýze je nutné stanoviť kľúčové prvky aplikácie a načrtnúť prípadné alternatívy jeho vývoja. Je nevyhnutné preskúmať oblasť temporálnych databáz, problémy s nimi spojené. Získať teoretické znalosti pracujúce na pozadí temporálnych dopytov. Následne je potrebné analyzovať existujúce riešenia a z nich vybrať vhodných kandidátov pre demonštráciu implementácie. Je nutné naštudovať rozhranie JDBC a jeho implementáciu pre správny vývoj ovládača.

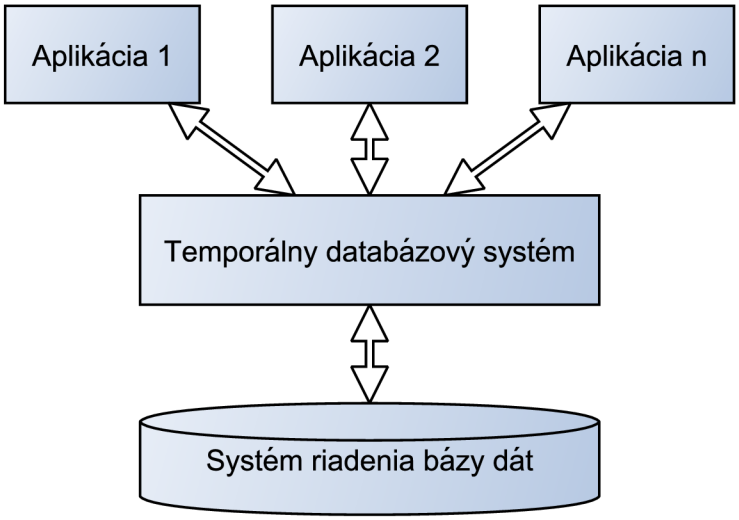
2.1 Temporálne databázy

Temporálne databázy sa snažia reálnejšie napodobniť skutočný svet kedy samotná entita existuje len určitý čas. Takéto správanie je veľmi bežné a vyskytujúce sa takmer v každej oblasti, či už ide o finančníctvo kde je možné temporálnym záznamom modelovať históriu finančných prostriedkov na účte, prípadne v medicíne kde môže obsahovať časovo obmedzený záznam liekov, ktoré pacient užíva. Takýchto príkladov je možné nájsť veľmi veľa, čo svedčí o tom, že oblasť temporálnych databáz má zmysel a preto je vhodné pokračovať vo vývoji.

Keďže sme spomenuli, že táto oblasť je stále vo vývoji, ale jej samotná aplikácia už existuje je vhodné poznamenať akým spôsobom sa tento problém rieši. Temporálna databáza je v princípe samostatnou databázou. Natívna implementácia podpory práci s časom sa do už existujúceho relačného alebo nerelačného databázového systému pridáva ako nadstavba. Pôvodný databázový systém reprezentuje časový údaj ako bežne uloženú informáciu bez jej hlbšieho pochopenia. Takáto rozšírená databáza umožňuje vytváranie jednoduchších dopy-

tov na server čím znižuje pravdepodobnosť programátorskej chyby. Rovnako tak podporuje štandardné operácie s dátami ako vytváranie, čítanie, upravovanie a mazanie jednoduchším dotazom bez nutnosti aplikačnej podpory. Z toho vyplýva, že temporálnu databázu je možné nahradiť implementáciou v samotnej klientskej aplikácií, prípadne zložitejším dopytom na server.

Z implementačného hľadiska sa najčastejšie temporálne databázové systémy vytvárajú ako nadstavba nad existujúci systém riadenia bázy dát. V takomto prípade sa umiestni implementácia temporálnych dopytov medzi aplikáciu a existujúci SRDB[1]. Temporálna nadstavba sa stáva prekladačom medzi formátom rozhrania oproti aplikácií a formátom rozhrania voči databáze. Schéma architektúry takéhoto systému je zobrazená na obrázku 2.1.

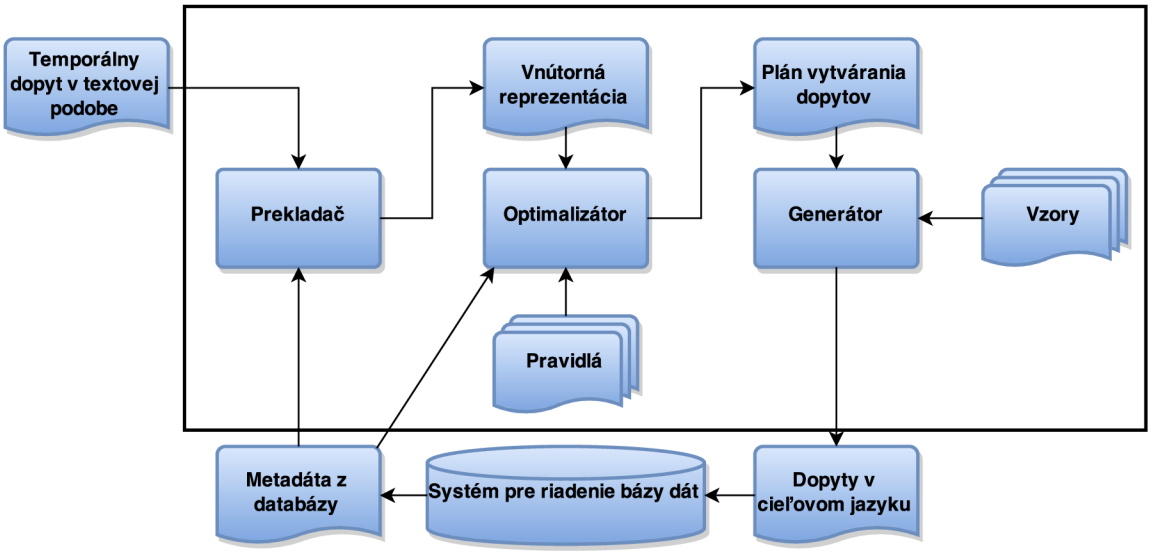


Obr. 2.1: Architektúra práce s temporálnou databázou

Vnútorná štruktúra takejto medzivrstvy sa nápadne podobá na prekladač jazyka 2.2 [16]. Prvou dôležitou časťou systému je preklad do internej reprezentácie. Táto komponenta reprezentuje bežný prekladač jazyka riadený syntaktickým analyzátorom. Syntaktický analyzátor požaduje od lexikálneho analyzátora tokeny. Tieto vyhľadáva vo vstupnom textovom temporálnom dopyte. Súčasne s týmto kontroluje syntaktický analyzátor správnosť postupnosti tokenov a zároveň požaduje od sémantického analyzátora vykonávanie sémantických kontrol. Rozdielom voči bežnému prekladaču je, že táto prvá komponenta očakáva na vstupe metadáta od relačného databázového systému. Tieto metadáta obsahujú názvy a typy definovaných identifikátorov tabuliek, riadkov, funkcií, procedúr, integritných obmedzení ... Výsledok takéhoto prekladu je posunutý do ďalšej komponenty, ktorá má za úlohu

[1]Systém riadenia bázy dát

vykonať optimalizáciu. V tomto prípade sa nejedná o optimalizáciu SQL dopytu, ktorá sa implicitne vykonáva v cieľovom systéme pre riadenie bázy dát a preto ju nie je potrebné vykonávať v tejto fáze. Hlavnou úlohou tejto komponenty je vytvorenie plánu ako sa získajú temporálne dáta. Jej vstupom je kolekcia pravidiel, ktoré obsahujú preddefinované vzory pre tvorbu plánu. Na základe vytvoreného plánu je možné v poslednej komponente generovať dopyt v cieľovom jazyku. Najčastejšie ním je SQL. Pre generovanie je vhodné použiť rovnako externú reprezentáciu vzorov prekladu, nakoľko výmenou týchto vzorov je možné generovať kód pre iný databázový systém bez nutnosti zmeny v iných častiach aplikácie.



Obr. 2.2: Implementácia temporálnej medzivrstvy

2.2 Teoretický model

Ako mnoho vecí aj temporálne databázy sú založené na existujúcom matematickom modeli. Kedže sa výskumu ukladania časovo závislých dát venovalo mnoho skupín poznáme aj veľké množstvo exaktných modelov na popis abstraktného dopytovacieho jazyka. Tieto modely sú zavislé na použitej reprezentácií času.

2.2.1 Reprezentácia času

Čas je spojitá fyzikálna veličina avšak pre jej reprezentáciu v temporálnom databázovom systéme ju môžeme definovať ako niekoľko typov časových údajov. Pri konkrétnej implementácií databázového systému musíme riešiť problém ukladania spojitej informácie do diskrétneho priestoru. Toto je možné vyriešiť zavedením vhodnej granularity, ktorou definujeme po akú úroveň presnosti chceme informáciu ukladať. V prípade zvolenia vysokej

granularity môžeme prísť o presnosť pri jednotlivých časových okamžikoch a tým sa môže stať, že sa nám spoja dva odlišné časové body do jedného. Opačným prípadom je keď zvolíme príliš nízku granularitu zbytočne zaťažíme databázový systém s nepodstatnými informáciami a rovnako tak ani výsledky, ktoré získame nemusia byť presne ako očakávame. V tomto prípade by sa nejednalo o chybné spracovanie dopytu ale o mylné vnímanie rovnosti časového údaja. Príkladom takého omylu je keď človek očakáva pri bežných úkonoch, že operácie vykonané v rovnakej sekunde sa stali naraz ale ak je granularita nastavená na milisekundy je nutné vykonávať už intervalové porovnávanie namiesto porovnávania na zhodu. Podľa autorov [22] je možné reprezentáciu času rozdeliť na 4 rôzne elementy. Tieto sú:

Časový bod Jedná sa o časové razítko reprezentujúce jeden okamih na celej časovej osi. Pre časové body vieme definovať reláciu čiastočného usporiadania nasledovne:

$$I = \langle P, \leq_t \rangle, P = p_1, p_2, \cdots, p_n$$

Kde I je jednorozmerná lineárne zoradená temporálna doména, P je množina časových bodov a $\leq_t$ reprezentuje reláciu čiastočného usporiadania. Z teórie usporiadanie vieme, že táto relácia musí byť reflexívna, antisymetrická a tranzitívna.

Časový interval Jedná sa o jeden spojitý časový úsek s jedným počiatočným a jedným koncovým bodom. Na základe predchádzajúcej definície usporiadania časových bodov vieme definovať 4 typy intervalov.

$$\begin{aligned} [p_1, p_2] \quad & p_1 \leq_t t \leq_t p_2 \\ [p_1, p_2) \quad & p_1 \leq_t t <_t p_2 \\ (p_1, p_2] \quad & p_1 <_t t \leq_t p_2 \\ (p_1, p_2) \quad & p_1 <_t t <_t p_2 \end{aligned}$$

Kde p_1 je počiatočný časový bod intervalu, p_2 je koncový časový bod intervalu a t sú všetky body spĺňajúce reláciu usporiadania. Tieto body tvoria spojitý interval.

Časové rozpätie Jedná sa o neprerušený časový úsek bez počiatočného a koncového bodu. Vyjadruje dĺžku trvania.

Časová množina Ide o zjednotenie rôznych časových bodov a intervalov.

2.3 Význam času

Netemporálne databázy vnímajú čas ako bežnú dátovú jednotku bez pochopenia jej sémantického významu. Čas je pre temporálne databázy prioritnou veličinou s ktorou pracujú.

Dokážu mu priradiť význam, interpretovať ho, ale stále umožňujú vytvorenie časovej položky bez pridelenia významu. Snodgrass vo svojej publikácií rozdelil časové záznamy do troch kategórií:[21]

Záznam s transakčným časom Tento záznam vyjadruje kedy bola vykonaná operácia s dátovou položkou. Najbežnejšou reprezentáciou takéhoto typu času sú dva časové údaje. Jeden reprezentuje časové razítko vloženia položky a druhý reprezentuje časové razítko úpravy alebo vymazania položky. Označenie časového údaja s koncovým transakčným časom môže byť na začiatku buď pomocou špeciálnej hodnoty NULL. Ak však toto cieľový databázový server nepodporuje je možné použiť maximálny uložiteľný dátum, ktorý nebude reálne dosiahnutý za celú životnosť systému. Z tohoto vyplýva, že položky sa z databázy nevymažú úplne ale len logicky, čo je možné dosiahnuť doplnením správneho formátu dátumu do stĺpca s koncovým časovým razítkom. Tým sa označí položka za vymazanú. V prípade, že je potrebné vykonať úpravu v nejakom zázname, je označený ako keby bol vymazaný a je vytvorený nový záznam s počiatočným časovým razítkom úpravy.

Transakčný čas sa v praktických aplikáciách používa ak je potrebné vytvoriť históriu zmien položky v databázovom systéme. Často je doplnený identifikátorom označujúcim osobu, ktorá vykonala danú zmenu. V prípade problémov je možné na základe takýchto záznamov vyvodzovať zodpovednosť, prípadne zostavovať aktivitu jednotlivých používateľov systému v čase. Rozšírením využitia transakčného času môže byť implementované aj zabezpečenie integritnými obmedzeniami. Príkladom takéhoto obmedzenia je povolenie vykonávať zmeny napríklad len v pracovnom čase. Týmto sa zvýši teoretické zabezpečenie systému a pracujúci personál dokáže eliminovať prípadné neoprávnené zmeny.

Mnohokrát sa označuje termínom transakčný čas aj jeden stĺpec dátovej položky, kde sa ukladá časové razítko pri vložení alebo úprave. Toto však nie je z hľadiska temporálnych databáz validné a jedná sa len o rozšírenie konkrétneho databázového systému, čo však nemožno označovať za temporálnu databázu.

Záznam s časom platnosti Takýto záznam najčastejšie vyjadruje ohraničenú platnosť danej dátovej položky. Rovnako ako pri transakčnom čase ide o dva časové údaje ohraničujúce jednu spojitú periódu. Podobným spôsobom sa modeluje aj neobmedzená platnosť do budúcnosti ale rovnako tak aj do minulosti. Na rozdiel od transakčného času, tieto údaje môžu byť zadané pri vkladaní alebo neskôr upravené. Môžu obsahovať budúci alebo minulý časový údaj, vzhľadom k času vkladania. Pri dopytoch na položky s časom platnosti sú tri významné skupiny požiadaviek. Prvou skupinou sú požiadavky na aktuálny stav, kde sa do výsledku dostanú len aktuálne platné po-

ložky. Druhou skupinou sú sekvenčné požiadavky zobrazujúce dáta, ktoré boli platné v konkrétne určenom čase. Poslednou skupinou sú požiadavky nesekvenčné, ktorých výsledkom je zobrazenie záznamov bez toho, že by systém bral do úvahy časovú platnosť uvedenú pri záznamoch. V tomto prípade sa na tieto stĺpce s časom platnosti pozerá ako na bežné používateľom definované dáta a nepriraďuje sa im žiadna špeciálna sémantika.

Pri tomto type časového záznamu je zložitejšie vykonávanie operácií vkladania, mazania a úpravy. Táto zložitosť plynie z nutnosti vyriešenia niekoľko prípadov prekrytia samotným databázovým systémom.

V reálnom nasadení tento typ času reprezentuje meniace sa vlastnosti skutočne existujúceho prvku a uchováva celú jeho históriu zachytenú do databázového systému.

Bitemporálny záznam Pod týmto označením si môžeme predstaviť spojenie dvoch predchádzajúcich typov časových záznamov. Pri takomto type záznamu dátová položka obsahuje čas platnosti reprezentujúci históriu konkrétnej reality. Ďalej obsahuje aj transakčný čas rerezentujúci vykonávanie zmien v samotnej databáze. V takomto prípade samotná dátová položka uchováva štyri časové údaje, dôsledkom čoho je väčší možný počet kombinácií. Toto môže spôsobiť veľký nárast počtu položiek v databáze reprezentujúcich jednu položku v reálnom svete.

2.4 Dopytovacie jazyky

Počas histórie temporálnych databáz bolo mnoho pokusov definovať vhodný jazyk pre prácu s takýmito údajmi. Mnoho jazykov však bolo len v teoretickej rovine a mali značné obmedzenia pre konkrétnu implementáciu, na ktorú boli navrhnuté. Kedže prvotne sa využívala sila temporálnych databáz hlavne v deduktívnych systémoch aj známi zástupcovia špecifických temporálnych jazykov sú priamim rozšírením Prologu. Medzi prvé temporálne dopytovacie jazyky je možné zaradiť TempLog[7], ktorý vznikol ako demonštrácia časového modelu definovaného v Hornových klavzulách. Ďalším známym príkladom takéhoto jazyka je Chronolog[18], z ktorého neskôr čerpali aj iné už bežnejšie používané jazyky. Nakoľko temporálne databázy využívajú najčastejšie na pozadí pre ukladanie dát štandardný relačný model, tak sa ako najvhodnejšie javí použitie už existujúceho jazyka pre komunikáciu s databázou a jeho rozšírenie o prácu s časovo premenlivými údajmi. Keďže relačný model má definovaný štandard SQL, tento bol použitý aj pri definovaní jeho odnoží pre temporálne databázy. Umožnil pridaním rozširujúcich prvkov vznik dvoch veľmi známych jazykov v oblasti temporálnych databáz a to ATSQL a TSQL2.

2.4.1 SQL-92

Jedná sa o štandard dopytovacieho jazyka nad relačným databázovým modelom, ktorý bol špecifikovaným už v roku 1970 pánom Coddom[12]. Nie je určený pre temporálne databázové systémy nakoľko neprideľuje časovej dimenzii špeciálnu sémantiku, aj keď umožňuje ukladanie časových údajov a následne vykonávanie zložitejších dopytov nad tabuľkami, čím sa stáva vhodným adeptom na to aby bol základom pre ostatné jazyky pracujúce s temporálnymi dátami uloženými v relačnej podobe. Používateľ dokáže vhodnými zloženými dopytmi modelovať vykonanie temporálnej operácie aj mimo temporálny databázový systém.

2.4.2 TSQL2

Jedná sa o temporálne rozšírenie pre dopytovací jazyk SQL-92. Prvotný pokus o definovanie tohoto štandardu bol v roku 1992 pod správou Richarda Snodgrassa. Jednalo sa však ešte len o pokus sformovania výskumnej skupiny, ktorá by sa tejto problematike venovala. Samotná definícia jazyka bola niekoľko krát modifikovaná kým sa dostala do finálnej podoby v roku 1994[20].

Významné rozšírenie priniesol v oblasti definovania integritných obmedzení, kde definoval nové kľúčové slovo a súčasne aj dátový typ `SURROGATE`. Pomocou neho dokáže programátor databázovému systému povedať, že aktuálne vkladaný záznam obsahuje nový ešte neexistujúci prvok a preto je potrebné mu vygenerovať unikátny identifikátor. Ďalším dátovým typom, ktorým jazyk rozšíril SQL-92 štandard je typ PERIOD. Tento typ sa používa na definovanie časových úsekov. Pozostáva z dvoch jednoduchých časových razítok, kde jedno ohraničuje začiatok a druhé koniec daného intervalu. Navyše je možné pri tomto dátovom type definovať jeho presnosť, čo v podstate reprezentuje granularitu času.

Jazyk definuje tri typy uložených časov. Prvým je používateľom definovaný čas, tento je mimo správu temporálny databázový systém a môže ich byť v jednom zázname niekoľko. Je vhodné použiť ho keď potrebujeme vyjadriť nejakú špecifický časový stav, ktorý sa nedá prideliť k iným typom časov, ktoré sú spravované systémom. Ďalším typom už v réžií temporálnej databázy je čas platnosti. Tento čas vyjadruje okamih alebo obdobie, počas ktorého bol daný záznam platný. Pri doteraz spomenutých časoch je podstatné si uvedomiť, že ich hodnoty sú neobmedzené a časová os, ktorú vyjadrujú môže byť nekonečná. Na základe toho bolo potrebné definovať nové kľúčové slová, ktoré by umožnili vyjadrenie tejto vlastnosti. Preto pre vyjadrenie začiatku úseku od prvopočiatku je možné pomocou slova `beginning` a vyjadrenie konca navždy pomocou slova `forever`. Pre tretí typ času, čo je transakčný čas, nie je možné tieto kľúčové slová použiť nakoľko jeho časová os je obmedzená. Počiatočná hodnota, ktorú môže nadobúdať je čas vytvorenia tabuľky a koncová hodnota je čas vykonania zmeny. V prípade, že zmena nebola ešte vykonaná, tak koncová položka časového úseku je označená kľúčovým slovom `until changed`[20]. Na základe existencie

týchto časov systém rozoznáva medzi nasledujúcimi typmi tabuliek:

Snímková Bežná tabuľka bez časového záznamu spravovaného temporálnym systémom. Nie je vylúčené, že táto tabuľka bude obsahovať používateľom definovaný čas.

S udalostným časom platnosti Tabuľka so záznamom času platnosti kde tento čas má význam jedného okamihu na časovej osi.

So stavovým časom platnosti Tabuľka so záznamom času s významom ohraničenej dĺžky trvania.

S transakčným časom Tabuľka uchováva transakčný čas.

S bitemporálnym udalostným časom Tabuľka uchováva transakčný čas a aj čas platnosti s významom jedného okamihu.

S bitemporálnym stavovým časom Tabuľka obsahuje transakčný čas spolu s časom platnosti s nenulovým trvaním.

Ďalším potrebným rozšírením je možnosť špecifikácie odsávania starých záznamov. Je možné ho definovať pomocou kľúčového slova `VACUUM`. Následne je potrebné zadať časový výraz pre pravidelné vymazávanie zastaralých dát. Odsávanie je však možné definovať len nad tabuľkou, ktorá obsahuje čas platnosti.[15]

Tento jazyk podporuje všetky štandardné DDL[2] a DML[3] operácie. Príkladom sú nasledujúce príkazy zapísane v TSQL2

2.4.3 ATSQL2

Jazyk získal názov skrátením pomenovania Applied TSQL2. Už teda z názvu je možné identifikovať, že predlohou pre jeho vznik bolo vyššie spomenutý jazyk TSQL2. Druhým jazykom, ktorý ho ovplyvnil je Chronolog, čo je logický temporálny jazyk používaný v rovnako pomenovaj nadstavbe Prologu.[11] Jeho princípom je vopred obmedziť oblasť, s ktorou sa pracuje pomocou stavových modifikátorov, majúcich význam predikátu v logickom jazyku, a v rámci nich až pracovať so záznamami, ktoré spĺňajú obaľujúci predikát. Vďaka tomuto prístupu je jazyk kompatibilný s SQL-92 a teda môžeme s databázou jednoducho komunikovať týmto jazykom. V tomto prípade sa správa ako keby v nej boli uložené len aktuálne záznamy a čas platnosti je automaticky nastavený na aktuálny. V prípade, že chce používateľ využiť operácie poskytnuté temporálnym systémom sú mu k dispozícií nasledujúce stavové modifikátory:

[2] Data Definition Language - jazyk pre definíciu dát
[3] Data Manipulation Language - jazyk pre manipuláciu s dátami

VALIDTIME Jedná sa o sekvenčný modifikátor, ktorý vyhodnocuje daný dopyt v rámci všetkých možných stavov databázy s tým, že berie do úvahy čas platnosti jednotlivých záznamov.

VALIDTIME period Jedná sa tiež o sekvenčný modifikátor avšak obmedzuje svoje vyhodnocovanie len na definované časové okno.

NONSEQUENCED VALIDTIME Tento modifikátor definuje nesekvenčné správanie databázy, čo znamená že pri dopytoch sa časový záznam stáva bežným dátovým záznamom bez sémantiky.

NONSEQUENCED VALIDTIME period Rovnako aj tento modifikátor definuje nesekvenčné správanie dopytu avšak predtým sú z databázy vybraté len záznamy ktoré spĺňajú definované časové okno. S týmito vybranými položkami sa však následne už pracuje bez ohľadu na časový záznam.

2.5 Java

Java je objektovo orientovaný programovací jazyk so syntaxou podobnou jazyku C preberajúci prácu s objektami z jazyka C++. Dôvodom vytvorenia nového jazyka bolo ekonomické hľadisko. Dovtedy používané jazyky potrebovali kompilátor pre preklad do strojového kódu špecifického pre konkrétny procesor. Java tento problém odbúrala nakoľko jej kód prekladaný do Bytekódu[4], ktorý je interpretovaný virtuálnym strojom JVM[5]. [19] V súčasnosti sa tento programovací jazyk teší veľkej obľube u programátorov hlavne pre svoju nezávislosť na cieľovej platforme.[1]

Keďže sa jedná o plnohodnotný programovací jazyk jeho súčasťou musí byť aj možnosť práce s trvalo uloženými dátami. Java má pre toto hneď tri spôsoby ako trvalo ukladať potrebné informácie bez nutnosti reimplementácie programu, čím zachováva svoju ideu prenositeľnosti bez zásahu programátora:[2]

Java DB Ide sa o implementáciu relačného databázového systému Apache Derby, ktorý je priamo podporovaný v JDK[6]. Podporuje SQL dopyty ale aj volania JDBC. Môže byť priamo súčasťou vyvíjanej aplikácie. Podporou tohoto databázového systému sa spoločnosť Oracle snaží dosiahnuť jednotnosť práce s databázou. Obmedzením tohoto prístupu sú špecifickejšie operácie, ktoré táto databáza neposkytuje.

Java Data Objects [7] Tento prístup pre uchovávanie dát vznikol na základe mnohých

[4] optimalizovaná skupina inštrukcií
[5] Java Virtual Machine
[6] Java development kit
[7] JDO

požiadaviek na jednoduché perzistentné ukladanie objektov vytvorených v programe. Umožňuje intuitívnu manipuláciu s celým objektom bez nutnosti ďalších podporných operácií na napĺňanie objektu. Takáto implementácia je prenositeľná vďaka popisu dát v metadátach a vďaka tomu, že podporuje mnoho spôsobov ukladania. Najčastejšie sa používa v aplikáciách pre Java EE[8]. Nevýhodou je, že tento prístup podporuje len prácu s objektami a nepodporuje napríklad ich sémantické chápanie samotnou databázou. A rovnako ako Java DB nedisponuje rozšíreniami jednotlivých aplikačných serverov.

Java Database Connectivity [9] Jedná sa o najvšeobecnejší prístup k trvalému ukladaniu dát. Umožňuje pripojenie na rôzne databázové servery, úložisko dát prípadne na bežný súbor. Nad nimi vykonáva dopyty zadané jednotným rozhraním. Jeho výhodou je množstvo operácií, ktoré sa dajú vykonávať a rovnako aj to, že dokáže sprostredkovať rozšírenia databázového servera do samotnej aplikácie. Nevýhodou je pracnejšia implementácia v aplikácií nakoľko je nutné mnoho vecí definovať manuálne a rovnako je potrebné definovať pripojenie k databáze. Z tohoto dôvodu je tu možnosť zanesenia väčšieho množstva chýb.

Nakoľko sa jedná o objektovo orientovaný jazyk je možné jednoducho využiť objektové návrhové vzory, tieto zjednodušujú a zprehľadňujú implementáciu, prípadne do implementácie prinášajú špecifické vlastnosti, ktorými konkrétne vzory disponujú. Pre túto prácu by mohli byť vhodné nasledujúce návrhové vzory:

Abstraktná továreň Jedná sa o návrhový vzor z kategórie vzorov týkajúcich sa vytvárania objektov. Umožňuje definovať spoločné rozhranie pre vytváranie objektov, ktoré implementujú ostatné továrne. Tieto továrne následne vytvárajúce už konkrétne objekty. Architektúra tohoto návrhového vzoru umožňuje pridávanie nových tovární bez nutnosti zmien v už existujúcich. Používateľ systému sa nemusí starať o to aký objekt sa vytvorí, nakoľko to je v réžií samotnej továrne. Týmto prístupom umožňuje oddelenie spôsobu implementácie od používania, čím sa stáva kód udržatelnejším.[13]

Posluchač Tento návrhový vzor sa zaraďuje do kategórie vzorov týkajúcich sa správania. Je vhodné ho použiť v grafických rozhraniach v prípade, že chceme udržať oddelenosť definície vzhľadu a správania aplikácie. Jeho princípom je vytvorenie rozhrania, ktoré poskytuje prístup k implementácií reakcie na udalosť. Grafický objekt registruje a udržiava zoznam objektov, ktoré reagujú na jeho zmenu. V prípade, že zmena nastane je spustená akčná metóda vo všetkých objektoch zo zonamu.[13]

[8] Java Enterprise Edition
[9] JDBC

Adaptér Jedná sa o návrhový vzor, ktorý poskytuje svoje rozhranie inej komponente, či už z dôvodu veľkej premenlivosti rozhrania danej komponeny alebo z dôvodu spojenia viacerých komponent do jedného rozhrania. Patrí do skupiny rozhraní týkajúcich sa štruktúry programu. Jeho cieľom je sprehľadniť kód aplikácie a nemá žiadne funkčné účely. Princípom fungovania je, že sa v konštruktori vytvorí objekt adaptovanej triedy a následne v metódach rozhrania sa volajú metódy pre rôzne adaptované triedy uložené v atribútoch adaptéra.

2.6 JDBC

Je aplikačné rozhranie pre nezávislú komunikáciu medzi rôznymi databázovými serverami pomocou programovacieho jazyka Java. Obaľuje SQL dotazy do volania funkcií, čím dosiahne univerzálny spôsob zápisu pre rôzne databázové systémy. Toto rozhranie má tri základné funkcie:

1. Pripojenie k databáze
2. Odoslanie SQL dopytu
3. Spracovanie výsledkov

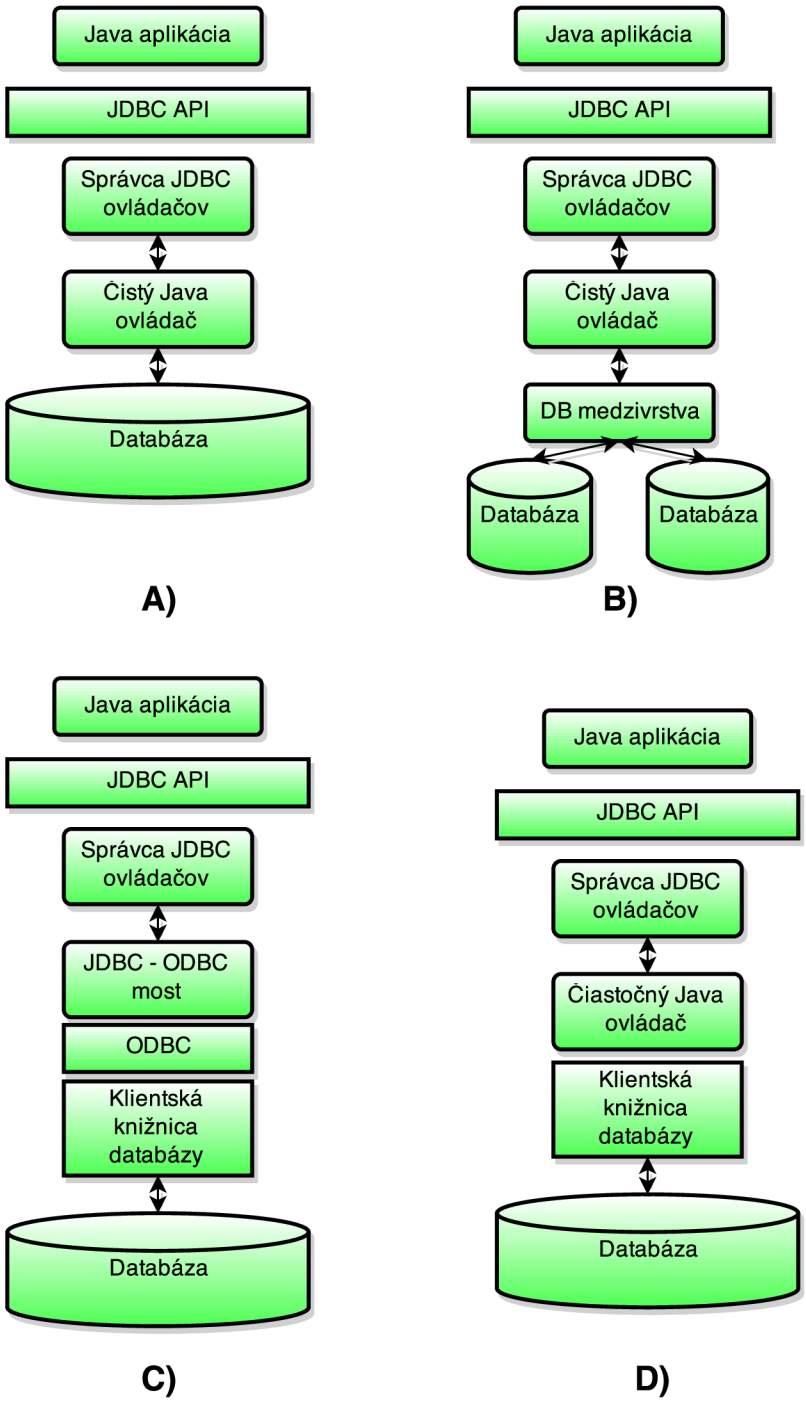
Jeho výhodou je, že nepotrebuje žiadnu inštaláciu nakoľko rozhranie je súčasťou jazyka Java a pre jeho implementáciu je nutné len pripojenie vhodnej knižnice na základe použitej architektúry.

2.6.1 Architektúra

Rozhranie JDBC definuje štyri odlišné architektúry napojenia na databázu.[3] Tieto architektúry sú:

Čistý Java ovládač Ide o ovládač, ktorý prekladá JDBC požiadavky priamo do protokolu používaného systémom pre správu bázy dát. Diagram tejto architektúry je zobrazený na obrázku 2.3 pod písmenom A. V tomto prípade je ovládač najčastejšie napísaný v jazyku Java a je súčasťou samotného JVM. Výhodou takéhoto prístupu je rýchlosť, nakoľko neexistuje žiadna medzivrstva a JVM má prehľad o všetkých dopytoch čím môže vykonávať optimalizácie. Nevýhodou je, že takáto implementácia je veľmi závislá na databáze, ku ktorej sa pripája a preto sa musí pre odlišnosti v komunikácií preprogramovať celý ovládač.

Čistý Java ovládač pre databázovú medzivrstvu V preberanej problematike ovládač prekladá JDBC požiadavky do protokolu nezávislého na použitej databáze, ktorému



Obr. 2.3: Architektúry JDBC ovládačov. Jednotlivé diagramy zobrazujú nasledujúce architektúry: A) Čistý Java ovládač, B) Čistý Java ovládač pre databázovú medzivrstvu, C) JDBC - ODBC most a D) Natívny API ovládač.[3]

rozumie medzivrstva. Diagram tejto architektúry môžete nájsť na obrázku 2.3 pod písmenkom B. Najčastejšie je medzivrstva implementovaná ako server a nie je súčasťou klienta. Medzivrstva prekladá prichádzajúce dopyty do požiadaviek, ktorým rozumejú samotné databázy. Môže implementovať podporu niekoľkých databáz s tým, že navonok komunikuje jednotným spôsobom. Výhodou takéhoto prístupu je, že pri zmene systému pre riadenie bázy dát nie je potrebné meniť kód aplikácie. Stačí implementovať len odlišné rozhranie medzivrstvy voči databáze. Rovnako tak môže medzivrstva poskytovať rozšírenie funkcionality samotného databázového systému. Nevýhodou ale je, že je nutné vykonávať preklad dopytov, čo je samozrejme réžia navyše.

JDBC - ODBC most Tento ovládač poskytuje prístup k databáze pomocou ODBC[10] ovládača. ODBC je jednotné rozhranie pre prístup k databáze. Pre jeho prevod do jazyka Java je ale potrebné ho obaliť do JDBC ovládača.[8] Architektúra takéhoto systému je zobrazená na obrázku 2.3 pod písmenom C. Volanie JDBC metódy je preložené na ODBC volanie, ktoré je dialektom SQL. Následne sa z ODBC ovládača využívajú klientské knižnice pre preklad na komunikáciu s jednotlivými databázovými systémami. Hlavnou výhodou takéhoto prístupu je veľké množstvo databáz podporujúcich prístup cez ODBC. Nevýhody sú rovnako ako v predchádzajúcom ovládači zvýšená réžia nutná na preklad dopytov. Spoločnosť Oracle, správca jazyka Java, neodporúča používanie tohoto ovládača, nakoľko jeho implementácia je vydaná spoločnosťou Sun v uzatvorenej podobe, preto je nemožné ju ďalej vyvíjať. Na základe týchto dôvodov bolo učinené rozhodnutie od verzie JDK 8 nepodporovať túto architektúru.[4]

Natívny API ovládač Pri tomto ovládači sa volanie JDBC metódy prekladaná na volanie klientskej API[11] daného systému pre riadenie bázy dát. Výhodou je vyššia rýchlosť nakoľko odpadá réžia prekladu. Avšak takáto implementácia má nevýhodu v tom, že je neprenositeľná ak je pre rôzne databázy odlišné API. Rovnako tak je podmienkou mať na klientskej strane nainštalované knižnice pre konkrétnu databázu ale hlavnou nevýhodou takéhoto prístupu je, že nie všetky databázové systémy disponujú klientskou knižnicou. Architektúra tohoto systému je na obrázku 2.3 pod písmenom D.

2.6.2 Rozhranie

Samotné rozhranie JDBC API je súčasťou jazyka Java. Jeho implementácia je definovaná v dvoch balíčkoch[5]

`java.sql` API pre prístup a spracovanie dát zo zdroja dát. Tento balíček implementuje základné klientské operácie. Tieto operácie je možné rozdeliť do niekoľkých skupín:

[10] Open Database Connectivity
[11] aplikačné rozhranie

[5]

Vytváranie spojenia Spojenie sa vytvára pomocou triedy `DriverManager`, ktorá definuje spojenie s ovládačom. V tejto skupine sa nachádza aj rozhranie `Driver`, ktoré definuje API pre registráciu a pripojenie JDBC ovládača. Toto rozhranie sa však používa hlavne so samotnou triedou `DriverManager`.

Odoslanie SQL príkazu Keďže SQL podporuje mnoho rôznych dopytov je nutné, aby v tejto skupine bolo viac rôznych tried. Najzákladnejšou je trieda `Statement`, ktorá sa používa pre odoslanie jednoduchého SQL príkazu. Pre zložitejšie parametrické príkazy je trieda `PreparedStatement`. Nakoľko SQL podporuje aj volanie procedúr a funkcií je nutná trieda obsluhujúca túto aktivitu a to je `CallableStatement`. Okrem práce s dátami je potrebné mať aj riadiace triedy, cez ktoré je možné nastaviť rôzne parametre. Pre pripojenie sa k databáze, nastavenie parametrov a vytvorenie potrebných objektov sa používa trieda *Connection*. Pre prácu s transakciami a vytváranie bodov návratu sa využíva trieda `Savepoint`.

Získanie výsledku z dopytu Rozhranie definujúce výsledok zadaného dotazu je špecifikované v triede `ResultSet`.

Mapovanie z SQL typov na reprezentáciu v Jave V tejto skupine je niekoľko tried, ktoré reprezentujú SQL dátové typy v samotnom jazyku Java. Jedná sa napríklad o triedu `Array` pre reprezentáciu SQL polí, `Date` pre reprezentáciu dátumu, `Ref` pre reprezentáciu odkazu v SQL a mnoho iných tried.

Mapovanie z SQL typov na užívateľom definované triedy Táto skupina tried umožňuje užívateľovi zvýšiť abstrakciu tým, že obalí základné dátové typy vlastnou triedou. Preto aby zapúzdrenie bolo možné, je nutné rozhranie `SQLData`, ktoré špecifikuje metódy pre mapovanie na užívateľský dátový typ ako inštanciu triedy. Navyše je potrebné definovať spôsob ako sa tento dátový typ číta a zapisuje. Metódy potrebné pre načítavanie hodnôt z dátového toku špecifikuje rozhranie `SQLInput` a metódy pre zápis hodnôt do dátového toku sú špecifikované v rozhraní `SQLOutput`.

Metadáta Skupina obsahuje rozhrania pre ukladanie rozširujúcich informácií. Rozhranie `DatabaseMetaData` uchováva informácie o používanej databáze. S vykonávaním SQL dopytov sú spojené rozširujúce informácie. Konkrétne pri vrátení výsledku dopytu sa informácie o stĺpcoch ukladajú do rozhrania `ResultSetMetaData`. Ak bol SQL dopyt parametrizovaný, informácie o parametroch sa uložia do rozhrania `ParameterMetaData`.

Výnimky Táto skupina definuje výnimky, ktoré môžu nastať pri práci s JDBC. Najčastejšou a najstriktnejšou výnimkou je `SQLException`, ktorá znamená, že nebolo možné pristúpiť k dátam z akéhokoľvek dôvodu. Jemnejšou výnimkou, ktorá môže nastať je `SQLWarning`. V prípade, že je prekročená maximálna dĺžka dát je generovaná výnimka `DataTruncation`. Keďže môže SQL obsahovať hromadnú prácu s dátami a chyba nastane len v niektorých príkazoch je to vhodné oddeliť od bežnej chyby, preto je definovaná výnimka `BatchUpdateException`.

[14]

`javax.sql` Jedná sa o rozširujúce funkcie závislé na použitom zdroji dát. Reprezentuje API pre serverovo špecifické spracovanie dát a prístup k nim. Je neoddeliteľnou súčasťou Java EE avšak nemusí byť implementované pre Java SE[12]. Podobne ako pri `java.sql` aj tu je možné rozdeliť triedy rozhrania do nasledujúcich skupín:

Rozhranie pre pripojenie Definuje alternatívne rozhranie `DataSource` pre pripojenie k zdroju dát. Jeho hlavnou výhodou je, že pridáva nové operácie a umožňuje zmeniť parametre spojenia bez nutnosti zmeny programu.

Spájanie dopytov Vytvára sa medzivrstva, ktorá spravuje dopyty odoslané na zdroj dát. Daná medzivrstva má prehľad o celej komunikácií, preto ju dokáže optimalizovať tým, že nevytvára nové komunikačné kanály, ale recykluje už otvorené a tým šetrí veľkú réžiu. Takýmto spôsobom môžu byť spájané spojenia na zdroj dát. Pre vytvorenie takéhoto spojenia sa využíva trieda `ConnectionPoolDataSource`, ktorá vracia objekt rozhrania `PooledConnection`. Pre toto je potrebné registrovať reakciu na udalosť triedy `ConnectionEventListener`, ktorá je vyvolaná pri ukončení spojenia a je jej zaslaný objekt `ConnectionEvent`. V prípade, že je podporované spájanie príkazov podobným spôsobom je možné registrovať reakciu na udalosť `StatementEventListener`, ktorá je vyvolaná pri vykonaní príkazu a je jej zaslaný objekt `StatementEvent`.

Distribuované transakcie Rozhrania tejto skupiny umožňujú vďaka medzivrstve zasielanie požiadaviek na niekoľko zdrojov dát v rámci jednej transakcie. Tieto triedy sa nepoužívajú v aplikácií ale implementuje ich samotné `DataSource` rozhranie.

RowSet Jedná sa o rozhranie, ktoré pridáva podporu pre JavaBeans™.

Java API je možné používať, len ak je k aplikácií pripojená implementácia rozhrania určenú pre konkrétny používaný databázový systém.

[12] Java Standard Edition

2.7 Existujúce implementácie

Ako bolo spomenuté vyššie najčastejšie sa jedná pri temporálnych databázach o akademické verzie, ktoré sú stále vo vývoji alebo sú už zastaralé. Druhou alternatívou sú zase proprietárne riešenia jednotlivých vydavateľov databázových systémov, ktoré ale často nerešpektujú štandardy pri dotazovaní, prípadne majú len čiastočnú podporu jednotlivých operácií.

2.7.1 TimeDB

Jedná sa o najrozsiahlejší databázový systém rozumejúci štandardným ATSQL2 dopytom. Umožňuje napojenie na niekoľko relačných databáz nakoľko je implementovaný v jazyku Java a používa JDBC rozhranie. Dokáže vykonávať dopyty, vytvárať objekty, mazať záznamy a definovať integritné obmedzenia.[6] Obrovskou nevýhodou tohoto systému je už ukončený vývoj a uzavretosť zdrojových kódov. V minulosti však projekt prešiel značným vývojom nakoľko jeho začiatky boli ako Ph.D. práca Andreasa Steinera. Implementovaný bol pôvodne v jazyku Prolog a umožňoval len pripojenie na databázu Oracle. Prechod na jazyk Java a definovanie temporálneho rozhrania bolo inšpirované prechodom na komerčnú verziu. Voči štandardu ATSQL2 TimeDB nepodporuje zadávanie nekonštantných údajov do obmedzenia intervalu. Veľkým obmedzením tohoto rozšírenia je, že od verzie 2 nepodporuje transakčný čas.

2.7.2 tsql2lib

Táto databázová medzivrstva vznikla ako súčasť diplomovej práce na Fakulte informačních technológií Vysokého učení technického v Brne. Pre komunikáciu využíva TSQL2. Je implementovaná v jazyku Java pre zabezpečenie dostatočnej prenositeľnosti. Z podobného dôvodu je implementovaná ako knižnica, ktorá sa pribalí k implementovanej aplikácií ako JDBC ovládač.[23] Rovnako však je potrebné pripojiť aj JDBC ovládač databázového systému, na ktorý sa chceme pripojiť. Následne prebieha komunikácia aplikácie cez JDBC ovládač tsq2lib a táto knižnica komunikuje s ovládačom databázy už bežným relačným spôsobom. Takéto riešenie je veľmi vhodné nakoľko sa využívajú len štandardizované operácie JDBC a odstraňuje sa závislosť na použitom systéme pre správu bázy dát. Samozrejme to so sebou prináša mierne spomalenie pre zvýšenú réžiu dvojitého prekladu dopytu.

2.7.3 ValidTime Oracle 12c

ValidTime je proprietárne riešenie problému práce s časom v databázach spoločnosti Oracle. Toto rozšírenie je priamo implementované v databázovom systéme od verzie 12. Na prácu

s časom využíva svoje rozšírenie jazyka SQL, ktoré nie je nijakým spôsobom štandardizované. Má len obmedzenú prácu s časom nakoľko podporuje len záznamy s časom platnosti.

2.7.4 TigerDB

Jedná sa o akademickú verziu bitemporálneho databázového systému. Vznikol na univerzite Aalborgv Dánsku. Pre písanie SQL dopytov využíva ATSQL2. Je veľmi podobný systému TimeDB. Súčasťou systému je konzola na zadávanie príkazov a ich interpretáciu. Logika prevodu temporálnych dopytov je implementovaná v jazyku Prolog avšak zložitejšie výpočetné operácie a práca s databázou sú napísané v jazyku C++. Je postavený na temporálnom deduktívnom systéme ChronoLog.[17] Problémom tohto systému je, že bol ukončený jeho vývoj pre prílišné napojenie na konkrétnu verziu databázového systému Oracle.

2.7.5 TempDB

TempDB vznikol ako alternatíva k TimeDB. Rovnako sa jedná o akademický projekt vytvorený výskumnou skupinou na univerzite Sun Yat-Sen. Podobne je implementovaný v jazyku Java avšak cieľovým databázovým systémom je MySQL. Ako hlavnú myšlienku si projekt nového temporálneho databázového systému kládol zjednodušiť temporálne operácie voči používanému relačnému databázovému systému. Nakoľko pri veľkom počte používateľov a dopytov zasielaných na server TimeDB neúmerne zaťažoval server, bolo potrebné vykonávať optimalizáciu dopytov, či už v rámci spracovania v medzivrstve alebo na serveri. Hlavnou myšlienkou optimalizácie bolo, že najčastejšie je potrebné pristúpiť ku dátumu s aktuálnou platnosťou prípadne akokoľvek súvisiacej položke s aktuálnym časom. Rovnako ako TimeDB aj TempDB má implementované GUI rozhranie zahrňujúce konzolu pre zadávanie príkazov. Navyše má rozhranie prepracovanejšie podporné zobrazovanie dát v databáze. Problémom tohoto projektu však je, že zanikol a už nie je možné ani získať jeho zdrojové kódy či inštalačné súbory. [22]

2.7.6 ChronoLog

Nejde o temporálny databázový systém v pravom slova zmysle, ale o temporálny deduktívny systém.[9] Pre svoju komunikáciu s používateľom používa rovnomenný temporálny deduktívny jazyk Chronolog, ktorý je súčasťou jazyka ChronoSQL[18]. Používa sa ako nadstavba na databázový systém Oracle. Nakoľko sa jedná o deduktívny model vhodnou voľbou programovacieho jazyka bol SWI Prolog. Problémom je, že jeho vývoj bol už ukončený nakoľko autor chcel vytvoriť len demonštratívnu aplikáciu s grafickou konzolou avšak nie je implementované žiadne aplikačné rozhranie.[10] Z tohoto dôvodu ho nie je možné použiť vo vlastnej aplikácií. Rovnako je problémom aj úzka väzba na staršie verzie databázového systému Oracle, ktoré už nie sú v súčasnosti podporované.

2.7.7 Ostatné

S trvalým ukladaním časových údajov sa v minulosti snažilo mnoho ľudí a výskumných skupín. Výsledkom ich snaženia sú však hlavne nedokončené verzie prípadne verzie s obmedzenou podporou. Mnoho projektov je uzavretých a neposkytujú žiadne rozhranie pre použitie vo vlastnej aplikácií preto sú určené len na konkrétny účel alebo na demonštrovanie operácií s časom. K väčšine z nich dokonca už neexistujú ani zdrojové kódy prípadne binárne súbory. Príkladmi takýchto temporálnych projektov sú: [10]

ARCADIA Objektovo orientovaná temporálna databáza určená na medicínske účely. Zadávanie dopytov pomocou GCH-OSQL[13].

Calanda Systém pre správu času pri finančných transakciách.

Historical Database Management System Rozšírenie relačného modelu o uchovávanie historických dát. Dopyty zadávané pomocou HSQL[14].

Temporal DataBase Management System Temporálny systém podporujúci základnú temporálnu relačnú algebru. Históriu ukladá použitím denormalizácie, ktorá povolí zložené hodnoty ako obsah bunky.

TempCASE Nástroj pre modelovanie temporálnych aplikácií. Vstupom sú temporálne modely, nad ktorými sa vykonávajú zadané operácie.

TempIS Temporálne rozšírenie pre databázu Ingres používajúce dopyty vo formáte TQuel[15]

TimeIT Nástroj určený pre testovanie temporálnych aplikácií. Zobrazuje vykonané zmeny a meria výkonnosť systému.

TimeMultiCal Projekt rozširuje databázový systém o podporu pre mnoho rôznych kalendárov a definuje rôzne časové konštanty používané vo svete.

T-REQUIEM Temporálne rozšírenie pre databázový systém Requiem. Používa podmnožinu TSQL2 dopytov.

T-squared DBMS Temporálny relačný databázový systém podporujúci množstvo typov časových údajov. Definuje komunikačné rozhranie pre dopyty v relačnej algebre, SQL a TSQL[16]

VT-SQL Konzolová nadstavba pre databázu Ingres. Vykonáva preklad temporálnych požiadaviek a dáta ukladá v databáze Ingres.

[13] GCH-OSQL = Granular Clinical History - Object SQL
[14] HSQL = Historical SQL
[15] TQuel = temporálny dopytovací jazyk
[16] TSQL = Temporálne SQL

Kapitola 3

Návrh aplikácie

Cieľom práce je vytvorenie aplikačnej medzivrstvy, ktorá umožní spojiť viacero temporálnych databázových systémov v jednom databázovom systéme tak, aby toto spojenie bolo používateľovi odtienené. Umožňuje využitie toho temporálneho systému, ktorý je vhodnejší na danú úlohu. Na nasledujúcu úlohu môže byť vhodnejší iný databázový systém, preto používateľ jednoducho prepne na tento a používa ho. Všetky zmeny musia byť dostupné pre každý databázový systém a uložené údaje musia byť konzistentné. Aplikácia má podporovať TimeDB, tsql2lib a TempDB. Počas teoretickej prípravy sa však zistilo, že TempDB už je pozastavený a nebolo možné získať jeho binárne súbory či zdrojové kódy ani po priamom kontaktovaní autorov tohoto temporálneho systému. Z tohto dôvodu bola ako alternatíva zvolená technológia od spoločnosti Oracle umožňujúca prácu s časom platnosti Oracle Validtime. Je však vhodné, aby implementovaný JDBC ovládač podporoval v budúcnosti jednoduché pridanie aj iného temporálneho systému.

3.1 Rozdiely v metadátach

Všetky temporálne databázy používajú pre svoju prácu dodatočné údaje, ktoré si ukladajú v pomocných tabuľkách alebo rôznych interných štruktúrach. Pre ich spojenie je potrebné, aby tieto metadáta boli vnímané všetkými systémami rovnako, čím sa zabezpečí vzájomná konzistencia.

3.1.1 tsql2lib

Temporálny systém tsql2lib využíva dve pomocné tabuľky `_TEMPORAL_SPEC` a `_SURROGATE`. Prvá tabuľka bližšie popísaná v 3.1 uchováva informáciu o vytvorených používateľských tabuľkách. Medzi dôležité údaje uchovávané v tejto tabuľke patrí definovaný typ časového záznamu a spôsob odsávania starých údajov. Druhá tabuľka popísaná v 3.2 uchováva počítadlá pre vytvárané objekty. Každý stĺpec typu `SURROGATE` tu má definované aké číslo bude

_TEMPORAL_SPEC		
table_name	VARCHAR	Názov tabuľky
valid_time	NONE|EVENT|STATE	Definuje typ prítomnosti časovej platnosti. Táto sa v tabuľke nemusí nachádzať vôbec, prípadne ak sa tam nachádza môže byť ako udalosť alebo stav.
valid_time_scale	UNDEFINED|SECOND|MINUTE|HOUR|DAY|MONTH|YEAR	Definícia granularity údaja reprezentujúceho čas platnosti. Udáva presnosť zobrazovaných a porovnávaných časových záznamov.
transaction_time	NONE|STATE	Definuje prítomnosť transakčného času. Tento sa v tabuľke môže ale aj nemusí nachádzať.
vacuum_cutoff	NUMBER	
vacuum_cutoff_relative	TRUE|FALSE	Definuje či je odsávanie údajov zadané relatívnou formou alebo priamo plánované na konkrétny časový okamih.

Tab. 3.1: Štruktúra a popis tabuľky _TEMPORAL_SPEC v temporálnom databázovom systéme tsql2lib

_SURROGATE		
table_name	VARCHAR	Názov tabuľky.
column_name	VARCHAR	Názov stĺpca v tabuľke, ktorý je typu SURROGATE.
next_value	NUMBER	Nasledujúca hodnota, ktorá sa bude prideľovať novovytvorenému objektu pre daný stĺpec.

Tab. 3.2: Štruktúra a popis tabuľky _SURROGATE v temporálnom databázovom systéme tsql2lib

priradené pri nasledujúcom vytvorení objektu. Samotné časové údaje sú ukladané priamo v používateľom vytvorenej tabuľke ako nový stĺpec. Pre uchovávanie času platnosti sa vytvoria stĺpce _VTS a _VTE. Prvý uchováva začiatok platnosti a druhý uchováva koniec platnosti daného záznamu. Pre uchovanie transakčného času sa zase vytvoria stĺpce _TTS a _TTE s podobným významom ako stĺpce času platnosti. Formát ukladaného času reprezentuje počet sekúnd, ktoré uplynuli od polnoci 01.01.1970.

3.1.2 TimeDB

Temporálny systém TimeDB využíva viac pomocných tabuliek. Hlavnou tabuľkou, kde sa uchovávajú informácie o type časového záznamu je `TABLE_TYPES`. Jej štruktúra je rozpísaná v tabuľke 3.3. Informácia o konkrétnych stĺpcoch používateľskej tabuľky sú uchovávané

TABLE_TYPES		
`table_name`	`VARCHAR`	Názov tabuľky
`table_type`	`0|1`	Definuje typ tabuľky, či sa jedná o pohľad alebo reálnu tabuľku
`time_type`	`validtime| transactiontime| snapshot`	Typ tabuľky z časového pohľadu. Môže sa jednať o tabuľku s časom platnosti alebo s transakčným časom, prípadne bez akejkoľvek spravovanej temporálnej informácie

Tab. 3.3: Štruktúra a popis tabuľky `TABLE_TYPES` v temporálnom databázovom systéme TimeDB

v pomocnej tabuľke `TABLE_VIEW_SCHEMES` a jej detailnejší popis je zobrazený v tabuľke 3.4. Tieto dve tabuľky sú pre systém TimeDB veľmi dôležité lebo ak nejaký záznam o existencii tabuľky alebo stĺpca sa v nich nenachádza systém ho nevidí a nedokáže pracovať s dopytom, ktorý by prípadne smeroval na takú tabuľku čo nemá vytvorený záznam. Toto je zapríčinené tým, že TimeDB nedokáže pracovať so zabudovanou databázovou schémou a preto si musí vytvárať svoju. Pri dopyte sa potom prehľadávajú tieto tabuľky a hľadá sa zhoda v názvoch. Kedže SQL nie je citlivý na veľkosti písmen temporálna medzivrstva prekladá všetky znaky v dopyte na malé a až následne ich porovnáva s databázou. Z tohoto dôvodu je nutné aby názvy boli ukladané tiež bez veľkých písmen lebo inak by to spôsobilo nenájdenie záznamu.

Keďže temporálny systém TimeDB umožňuje aj definovanie integritných obmedzení sú pre toto určené pomocné tabuľky `KEYS` pre uloženie kľúčov a `ASSERTIONS` pre definovanie podmienok, ktoré musia byť splnené. Ďalšou podporovanou operáciou je definovanie temporálnych pohľadov, ktorých vytváracie dopyty sú uložené v tabuľke `VIEWS`. Tieto pomocné tabuľky však nie sú až také významné nakoľko ostatné systémy nemajú podporu takýchto rozšírených operácií. Keďže TimeDB umožňuje len čas platnosti je podobne ako pri tsql2lib ukladaný v užívateľom definovanej tabuľke pod stĺpcami `vts_timeDB` a `vte_timeDB`, ktoré majú dátový typ `NUMBER`. Časový údaj ukladaný v týchto stĺpcoch je reprezentovaný ako počet sekúnd od polnoci 01.01.0000. Neberie sa do úvahy žiadny kalendárny systém a počíta sa, že každý jeden mesiac má 31 dní. Z tohoto dôvodu je nutné vytvoriť na aplikačnej úrovni logický filter kontrolujúci platnosť dátumu lebo technicky je možné uložiť aj neplatný dátum. Nie je možné špecifikovať žiadnu granaularitu ukladaných údajov a všetky časové

TABLE_VIEW_SCHEMES		
table_name	VARCHAR	Názov tabuľky
column_name	VARCHAR	Názov stĺpca v tabuľke
data_type	number|numeric| longint|integer| smallint|float| double|real| interval|period| date|char|varchar	Dátový typ stĺpca. Podporované sú len vymenované dátové typy a do tejto tabuľky sa ich názvy ukladajú ako reťazce.
nullable	NULLABLE|NOT_NULL	Označuje či daný stĺpec tabuľky môže nadobúdať nedefinovanú hodnotu null alebo nie a teda musí niesť nejakú hodnotu.
column_length	NUMBER	Definuje veľkosť dátového typu ak to má zmysel, ak to zmysel nemá je nastavená hodnota -1. Príkladom kedy má zmysel definovať dĺžku dátového typu je reťazec.
column_id	NUMBER	Definuje index/poradie stĺpca v tabuľke

Tab. 3.4: Štruktúra a popis tabuľky TABLE_VIEW_SCHEMES v temporálnom databázovom systéme TimeDB

údaje sú s presnosťou na sekundy.

3.1.3 ValidTime

Temporálny systém ValidTime od spoločnosti Oracle nevyužíva špeciálne doplňujúce tabuľky, ale len svoje interné štruktúry. Umožňuje definíciu len času platnosti pomocou príkazu PERIOD. Tento príkaz môže obsahovať vyjadrené alebo nevyjadrené stĺpce pre uchovanie časových údajov. Ak za príkazom PERIOD nasleduje definícia stĺpcov tieto sa stávajú začiatočným a koncovým bodom časového intervalu. Navyše sú dostupné aj ako bežné stĺpce uchovávajúce časový údaj. Ak používateľ nezadá názvy stĺpcov databázový systém si vytvorí svoje vlastné. Názov týchto stĺpcov sa odvíja od zadaného názvu časovej dimenzie. Vytvoria sa teda tri nové stĺpce s nasledujúcimi názvami:

názov_času jedná sa o definíciu temporálnej dimenzie a jeho dátový typ je NUMBER

názov_času_start jedná sa o začiatočný bod intervalu času platnosti a jeho dátový typ je TIMESTAMP(6) WITH TIME ZONE

názov_času_end jedná sa o koncový bod intervalu času platnosti a jeho dátový typ je

```
TIMESTAMP(6) WITH TIME ZONE
```

V prípade, že tieto stĺpce nie sú definované priamo majú nastavený príznak skrytosti v systémovej tabuľke `user_tab_cols`. Toto obmedzenie vyradí stĺpce z toho, aby sa dostali do výsledku pri použití príkazu `SELECT` so všeobecným identifikátorom. Je možné k ním pristupovať alebo ich upravovať pri zadaní ich presného názvu.

Pre získavanie uložených dát príkazom `SELECT` umožňuje prácu s časom buď cez zabudovaný systém `DBMS_FLASHBACK_ARCHIVE`. Toto riešenie nás však nezaujíma, nakoľko takýmto spôsobom sa vyradia až do dalšieho zadania časového intervalu všetky záznamy, ktoré nespĺňajú definované podmienky. Preto by nebolo možné prepínať medzi rôznymi systémami nakoľko by bolo potrebné dať Oracle databázový systém pred komunikáciou s inou temporálnou implementáciou do pôvodného stavu. Keďže na toto sa spoliehať nemôžeme je vhodné obmedzeniť prácu s časom platnosti len na priame dopyty obsahujúce časové podmienky príkazom `PERIOD`, ktorý nepozmeňuje správanie celej databázy. Pre podporu tohoto systému je potrebné mať nainštalovanú edíciu Enterprise databázového systému Oracle 12c a súčasne mať v nej rozšírenie `Sample Schemas`. Ak sa používa len štandardná edícia nefunguje vyberanie na základe času platnosti aj napriek tomu, že všetky metadáta sa ukladajú správne.

3.2 Alternatívy spojenia databáz

Jednou z možností je vytvorenie samostatnej databázovej schémy pre každý jeden temporálny systém. Pri tomto riešení by bola každá vytvorená používateľská tabuľka zrkadlená pre všetky podporované temporálne databázy. V tomto prípade by ale bol veľký počet redundantných dát hlavne ak by temporálny údaj bol nepomerne malý voči ukladaným dátam. Rovnako by bolo však potrebné riešiť zachovanie konzistentnosti údajov medzi kópiami tabuľky. Toto zachovanie konzistencie by bolo možné vykonať prekladom ATSQL dopytu do TSQL dopytu a naopak. Problémom by však bolo, že spomínané systémy obsahujú mnoho nedokončených častí, čo by mohlo viesť na neúspech operácie pri jednej tabuľke, avšak vykonanie zmeny pri druhej tabuľke a tým by nastala vzájomná nekonzistencia údajov. Rovnako by mohlo byť problémom aj nemožnosť preloženia niektorých dopytov do podoby druhého jazyka, pretože nepodporujú tie isté operácie. Často krát má používateľ databázového systému dostupnú len jednu schému, v ktorej pracuje. Toto by sa muselo vyriešiť tak, že by sa jednotlivé zrkadlené tabuľky vytvárali v rovnakej databázovej schéme pridaním identifikátora k menu tabuľky. Takýto prístup by však odbúral výhodu nezmenšovania menného priestoru. Výhodou riešenia cez zrkadlenie tabuliek by bolo, že implementované spojenie by nebolo závislé na konkrétnych metadátach používaných temporálnymi systémami a preto by bolo možné kedykoľvek preimplementovať daný temporálny systém prípadne jednoducho

doplniť iný. Rovnako by sa týmto riešením ani nezmenšovala doména použiteľných mien pre stĺpce a tabuľky nakoľko použitie samostatnej schémy by vytvorilo nový menný priestor.

Druhou alternatívou je implementovanie spojenia databáz v konkrétnom použitom databázovom systéme pomocou spúšťačov[1]. V tomto prípade je potrebné sa pozerať na uchovávanie údajov v metadátach pre časové záznamy a na základe tohoto vykonať spojenie. Takéto riešenie by odbúralo redundanciu, nakoľko by v systéme existovala len jedna reálna tabuľka pre jednu užívateľom definovanú tabuľku. Problémy by však mohli nastať pri nových verziách temporálneho systému. Ak by sa zmenili používané metadáta bolo by potrebné zmeniť aj definované spojenie medzi nimi. Spomenuté systémy, ktoré potrebujú ukladať metadáta v štruktúrach definovaných ich tvorcami neprešli takouto zmenou už niekoľko rokov a ani takáto zmena nie je plánovaná. Pri systéme Oracle ValidTime je pravdepodobné, že nastanú nejaké zmeny keďže sa jedná o novú nadstavbu. Tieto zmeny by sa ale nemali dotknúť samotnej implementácie, nakoľko metadáta sú uložené v používateľom definovaných stĺpcoch. Na základe predchádzajúceho zhodnotenia je toto riešenie vhodné na výslednú implementáciu.

Treťou alternatívou implementácie je definovanie databázových pohľadov nad jednotlivými tabuľkami čo by umožnilo pohľad na časové stĺpce pre každý databázový systém pod iným menom. Takéto riešenie odbúra v maximálnej miere redundantnosť uchovávaných údajov avšak je použiteľné len ak by mali všetky databázové systémy rovnakú reprezentáciu času. V našom prípade toto neplatí nakoľko každý systém používa iný internú reprezentáciu času a je potrebné vykonávať medzi nimi prepočet časovej hodnoty. Tento prepočet by bolo možné definovať aj pre databázový pohľad avšak už by nebolo možné vykonávať modifikačné a vkladacie operácie nakoľko takýto pohľad by bol matematicky vypočítaným a nie priamym. Ďalšou nevýhodou je, že by sa na úrovni tabuliek vytváralo mnoho nových názvov, ktoré by mohli prípadne kolidovať s existujúcimi tabuľkami. Toto by musel používateľ brať na zreteľ pri zadávaní dopytov.

3.3 Navrhované spojenie

Na základe porovnania výhod a nevýhod jednotlivých možností z predchádzajúcej sekcie je vhodné zvoliť druhú popisovanú variantu. V tomto prípade je potrebné len základné parsovanie zadávaných príkazov a ich rozdelenie do nasledovných skupín:

Vytváranie tabuľky Tento typ dopytu, `CREATE TABLE`, je nevyhnutné rozparsovať priamo v aplikácií a na základe získaných údajov je potrebné pridať metadáta aj do alternatívnych temporálnych databáz. Vo vytvorenej tabuľke je potrebné pridať stĺpce

[1]Database trigger - jedná sa o kus kódu, najčastejšie v PL/SQL, ktorý sa spúšťa automaticky v databázovom systéme pri vopred definovanej udalosti

pre ukladanie času alternatívnych databáz.

Mazanie tabuľky Pri dopyte typu `DROP TABLE` je potrebné zabezpečiť aby sa vymazali aj všetky záznamy v metadátach iného temporálneho systému. Pri tejto operácií sa automaticky vymažú aj databázové spúšťače pridelené k danej tabuľke čiže toto už nie je potrebné riešiť v správe aplikácie.

Manipulácia s dátami Do tejto kategórie spadajú príkazy `INSERT` a `UPDATE`. Tieto typy príkazov nevyžadujú v aplikačnej časti žiadne špeciálne operácie, nakoľko sa spracovávajú priamo v databáze pomocou spúšťačov. Je potrebné udržiavať rovnakú časovú informáciu pre všetky databázové systémy s tým, že sa berie do úvahy odlišná reprezentácia uloženého času.

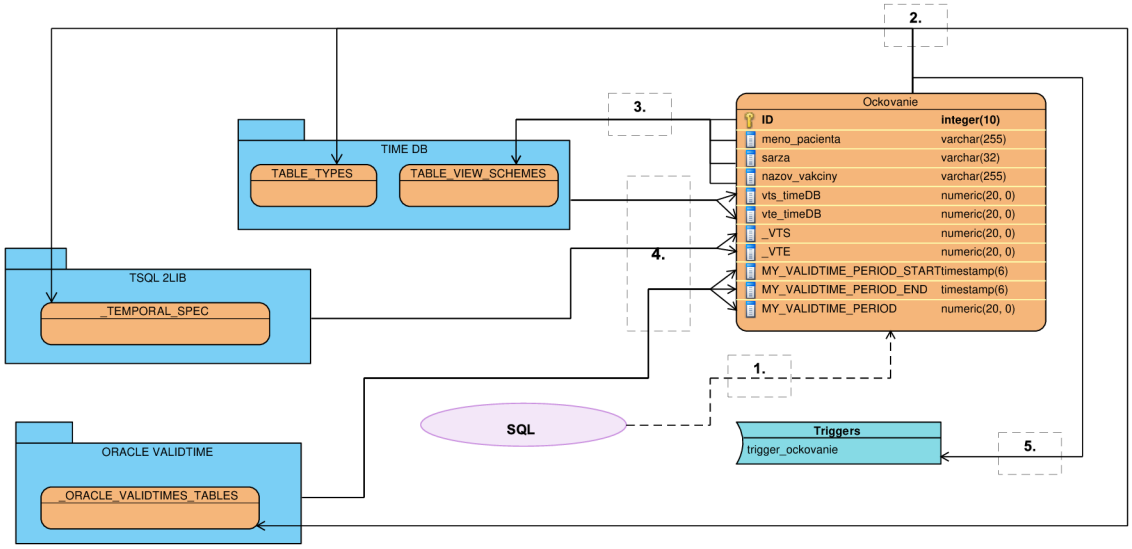
Získavanie výsledku Pri volaní príkazu `SELECT` sa správajú doplňujúce stĺpce alternatívneho databázového systému ako bežné dátové položky. Pre úplné odtienenie jednotlivých systémov je však vhodné tieto odstrániť na aplikačnej úrovni priamo zo získaného výsledku. Prípadne iným spôsobom zabezpečiť aby sa do tohoto výsledku vôbec nedostali.

Mazanie záznamu Operácia `DELETE` sa v systéme s časom platnosti preloží na skupinu operácii `UPDATE`, `INSERT` a `DELETE`. Temporálne mazanie môže vytvoriť v existujúcom časovom intervale platnosti záznamu dieru a tento sa tak rozpadne na dva nové záznamy, prípadne sa vymaže celý pôvodný záznam ak je vymazaná oblasť väčšia ako čas platnosti.

Odsávanie Táto operácia má význam reálneho vymazania záznamu z tabuľky. Nie je ju potrebné v systéme žiadnym spôsobom ošetrovať nakoľko sa s odstraňovaným záznamom vymažú aj záznamy v stĺpcoch spravovaných inými temporálnymi databázami.

Pre zjednotenie pohľadu na temporálne údaje zo všetkých systémov je potrebné identifikovať rozdiely a následne ich elimináciu implementovať do aplikačnej časti JDBC ovládača. Pri identifikácií rozdielov je vhodné použiť vyššie spomenuté rozdelenie dopytov a takýmto spôsobom aj implementovať triedy pre ich následné spracovanie. Aplikácia preto musí každé spustenie príkazu zanalyzovať a dopyt zaradiť do jednej z piatich kategórií, aby ho mohla následne vhodne spracovať.

Najzložitejším je dopyt pre vytváranie tabuľky, nakoľko musí vytvoriť všetko od začiatku. Postup je znázornený na obrázku 3.1 a mal by pozostávať z niekoľkých krokov, aby sa umožnila prehľadnosť a udržatelnosť celého JDBC ovládača. V prvom kroku je potrebné identifikovať názov novo vznikajúcej tabuľky aby sa bolo možné na ňu ďalej odkazovať. Následne je potrebné zistiť typ danej tabuľky, čo znamená že sa zaradí do jednej z časových kategórií. Môže sa jednať o tabuľku s časom platnosti, s transakčným časom, so žiadnym



Obr. 3.1: Schéma spracovania dopytu pre vytvorenie tabuľky. Postupnosť operácií je nasledujúca: 1. Vytvorenie tabuľky v databáze s používateľom definovanými položkami — 2. Vytvorenie záznamov v tabuľkách metadát pre jednotlivé temporálne databázové systémy — 3. Vytvorenie záznamu pre každý používateľom definovaný stĺpec v TimeDB — 4. Pridanie stĺpcov pre uschovanie času každým systémom — 5. Vytvorenie databázových spúšťačov pre prepočet pri úprave a vkladaní záznamov

spravovaným časovým záznamom alebo s obidvomi typmi časov. Pre potreby temporálneho systému TimeDB je ešte navyše nutné identifikovať dátové typy a ich presnosť všetkých stĺpcov v tabuľke. Keď už sú všetky údaje určené je možné urobiť zmeny v samotnej databáze. Tieto zmeny zahŕňajú vytvorenie doplnkových stĺpcov pre ukladanie všetkých potrebných časov pre každý ďalší databázový systém. Ďalej je potrebné vytvoriť všetky potrebné záznamy v metadátach jednotlivých databázových systémoch. Konkrétne sú to nasledovné údaje:

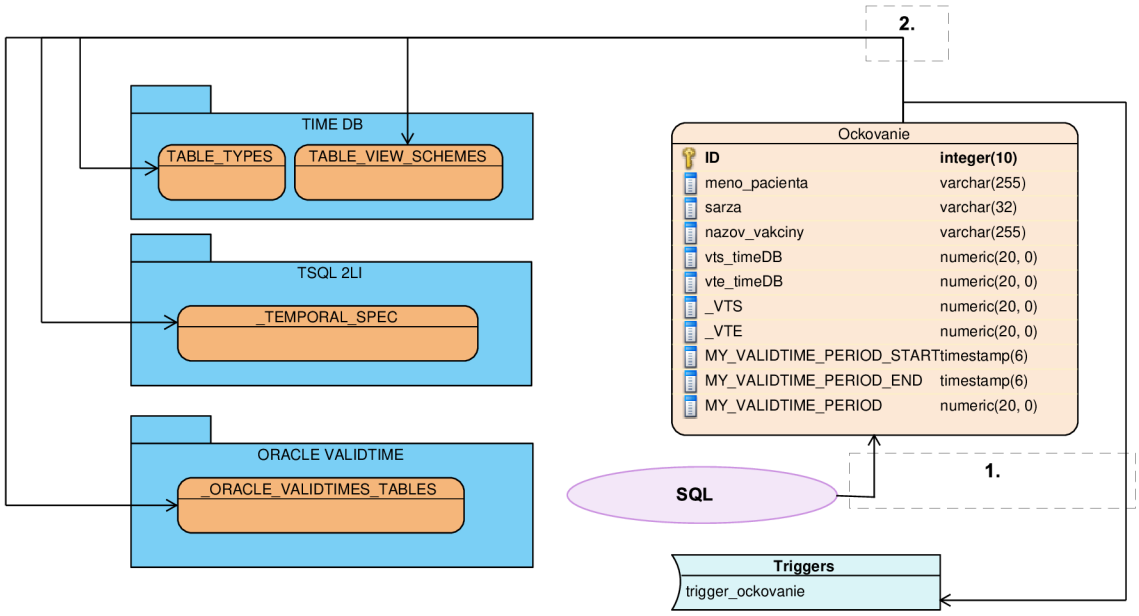
tsql2lib Zapísanie záznamu do tabuľky `_TEMPORAL_SPEC` so všetkými potrebnými údajmi, ktoré bolo možné identifikovať. Ostatné implementované systémy nepodporujú definíciu odsávania zastaralých dát preto je potrebné priradiť hodnotu vypnutia tejto funcie ak je tabuľka vytváraná z iného systému. Nie je však vhodné túto funkcionalitu plne ignorovať v celom systéme nakoľko sa môže stať, že v budúcnosti pribudne jej podpora v inom systéme a pri jej ignorácií by bolo nutné preimplementovať celý systém. Ďalšou vlastnosťou čo nie je podporovaná v ostatných databázach je definícia typu `SURROGATE`, avšak rovnako nie je vhodné ju ignorovať ale je potrebné pripraviť aspoň všetky operácie pre jej nasadenie.

TimeDB Zapísanie záznamu do tabuľky `TABLE_TYPES` s príslušným typom novovytvorenej

tabuľky. Následne je potrebné vykonať ešte uloženie záznamu pre každý používateľom definovaný stĺpec v tabuľke `TABLE_VIEW_SCHEMES`. Do tejto tabuľky nie je vhodné dávať záznamy pre stĺpce iných temporálnych systémov, nakoľko tým sa zaistí to aby neboli programovo viditeľné.

Oracle ValidTime Kedže táto tabuľka nemá žiadne verejne prístupné metadáta je potrebné špecifikovať vlastnú tabuľku pre ich uloženie. Do tejto tabuľky je potrebné ukladať názvy stĺpcov uchovávajúcich čas platnosti pre každú tabuľku s takýmto typom času, nakoľko neskôr tieto stĺpce nie je možné žiadnym bežným spôsobom identifikovať.

Opakom vytvárania tabuľky je jej zahodenie znázornené na obrázku 3.2. V tomto prí-



Obr. 3.2: Schéma spracovania dopytu pre vymazanie tabuľky. Postupnosť operácií je nasledujúca: 1. Odstránenie používateľom definovanej tabuľky a databázového spúšťača napojeného na túto tabuľku — 2. Vymazanie záznamov pre daný názov tabuľky v každej tabuľke určenej pre ukladanie metadát

pade je potrebné rovnako zistiť názov tabuľky, s ktorou sa pracuje a následne je možné vykonať zahodenie celej tabuľky. Týmto krokom sa odstránia metadáta uložené priamo v tabuľke, ale aj všetky databázové štruktúry, ktoré sú priamo napojené na tabuľku. V ďalšom kroku je potrebné vykonať operácie mazania v pomocných tabuľkách pre všetky záznamy spojené s vymazávanou tabuľkou. Konkrétne sa jedná o potrebu odstránenie záznamov s rovnakým názvom v stĺpci `table_name` ako je názov vymazávanej tabuľky v nasledujúcich tabuľkách pre jednotlivé databázové systémy:

tsql2lib `_TEMPORAL_SPEC` a `_SURROGATE`

TimeDB `TABLE_TYPES` a `TABLE_VIEW_SCHEMES`

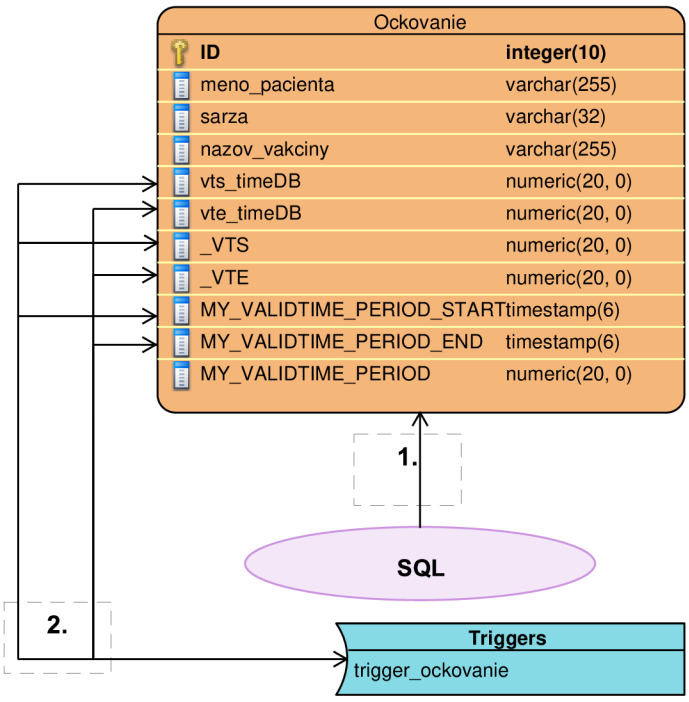
Oracle ValidTime `_ORACLE_VALIDTIME_TABLES`

Po vytvorení tabuľky je potrebné ju vedieť aj naplniť dátami tak, aby každý databázový systém videl tieto dáta rovnako. Na toto je možné použiť databázové spúšťače, ktoré budú nastavené na udalosť vkladania a upravovania záznamov v tabuľke. Tieto spúšťače je najvhodnejšie vytvoriť pri vytváraní celej tabuľky, nakoľko v tom okamihu poznáme všetky potrebné údaje a nemusíme ich opätovne analyzovať z už existujúcej tabuľky. Rovnako tak vieme okamžite odchytiť prípadné prichádzajúce dáta a správne ich spracovať. Spúšťač by mal obsahovať PL/SQL kód ktorý by bol závyslí na použitých databázových systémoch. V tomto kóde by sa overovalo pre aký databázový systém používateľ zadal dáta a následne by sa na základe toho vypočítali dáta pre všetky ostatné systémy. Rovnako musí byť kód informovaný o aký typ teporálnej tabuľky sa jedná aby vedel obslúžiť všetky potrebné časy. Vytvorený spúšťač nemusí brať ohľad na používateľské dáta, nakoľko tieto nie sú pre temporálny systém podstatné a sú zdieľané bez akejkoľvek zmeny. Tento spúšťač bez nutnosti akéhokoľvek zásahu dokáže spracovávať aj operáciu temporálneho vymazania záznamu z databázy. Grafické znázornenie tohoto postupu je možné vidieť na obrázku 3.3. Pre spojenie jednotlivých systémov musí definovať nasledujúce operácie pre každý implementovaný temporálny systém:

tsql2lib Kedže každý systém má odlišnú reprezentáciu dát je potrebné na začiatku vykonať prepočet vstupných časov. Následne je potrebné pomocou vypočítaných časových údajov zachovať konzistenciu vo všetkých spravovaných temporálnych stĺpcoch tabuľky priradených tomuto systému. Pre tabuľku s podporou času platnosti sú to stĺpce `_VTS` a `_VTE`. Pre tabuľku s podporou transakčného času sú to stĺpce `_TTS` a `_TTE`. Ak je tabuľka bitemporálna, je potrebné pracovať so všetkými štyrmi stĺpcami.

TimeDB Tento databázový systém si uchováva časové údaje v sekundách avšak tieto sa nepočítajú bežným kalendárnym spôsobom ale ako počet sekúnd, ktoré uplynuli od roku nula. Pre prevod kalendárnej informácie je potrebné použiť nasledujúci vzorec:

$$\begin{aligned} reprezentaciaVSekundach =& (roky * 12 * 31 * 24 * 60 * 60) + \\ &+ ((mesiace - 1) * 31 * 24 * 60 * 60) + \\ &+ ((dni - 1) * 24 * 60 * 60) + \\ &+ (hodiny * 60 * 60) + \\ &+ (minuty * 60) + \\ &+ sekundy \end{aligned}$$

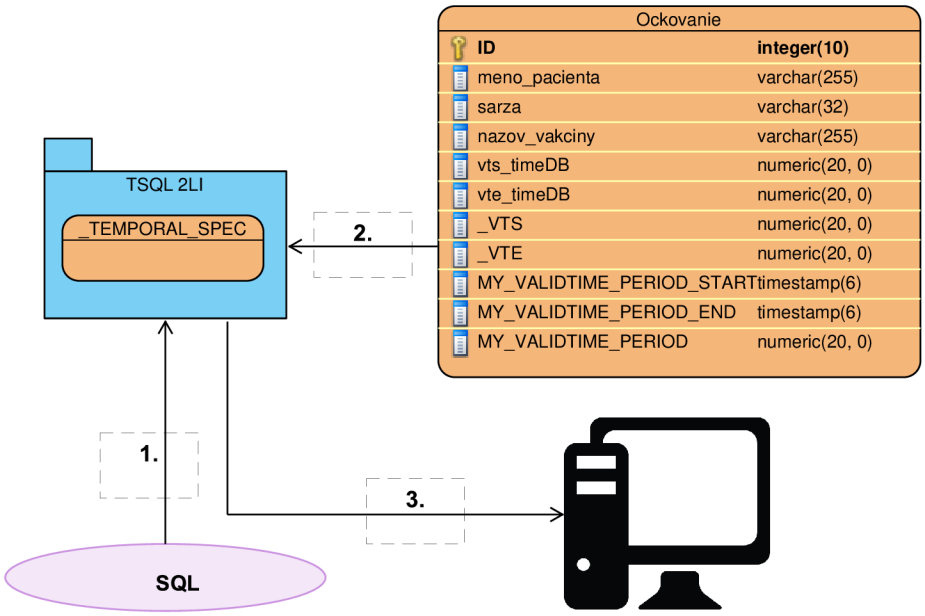


Obr. 3.3: Schéma spracovania dopytu pre vkladanie nových záznamov do tabuľky prípadne ich úprava. Postupnosť operácií je nasledujúca: 1. Vloženie záznamu do tabuľky — 2. Pred samotným zápisom je záznam odchytený databázovým spúšťačom a sú doplnené chýbajúce stĺpce vrátane toho, že sa berie do úvahy rozdielnosť ukladania časov

Prepočítané dáta je opäť potrebné uložiť do správnych stĺpcov. Pre tabuľku s podporou času platnosti sú to stĺpce `vts_timeDB` a `vte_timeDB`. Aj keď podpora transakčného času nie je v systéme implementovaná je vhodné sa pripraviť na jej podporu, nakoľko v zdrojovom kóde je tiež definovaná príprava na transakčné stĺpce `tts_timeDB` a `tte_timeDB`. Rovnako je potrebné pri bitemporálnom type spracovávať všetky štyri stĺpce.

Oracle ValidTime Pri tomto databázovom systéme je potrebné postupovať rovnako ako pri predchádzajúcich avšak rozdiel je v tom, že nemá pevne definované názvy stĺpcov s časom platnosti. Tieto názvy je možné získať z vytvorenej pomocnej tabuľky a následne udržiavať ich stav konzistentný. Pre tabuľky s transakčným časom je možné ho ignorovať nakoľko tento systém nemá žiadnu podporu pre prácu s transakčným časom.

Pravdepodobne najdôležitejšou operáciou pre používateľa je získavanie dát z databázy, ktorá je znázornená na obrázku 3.4. V prípade, že sú tieto dáta správne uložené nie je potrebné vykonávať žiadne dodatočné operácie pre to, aby výsledky čo temporálny databázový systém poskytne, boli správne. Môže sa však stať, že tieto výsledky budú obsahovať



Obr. 3.4: Schéma spracovania dopytu pre získanie údajov z tabuľky. Postupnosť operácií je nasledujúca: 1. Zaslanie požiadavky na spracovanie konkrétnemu databázovému systému — 2. Získanie údajov na základe používateľom definovaného dopytu priamo z databázy — 3. Vrátenie výsledku pre daný dopyt len s údajmi viditeľnými cez databázovú schému

aj údaje z iného temporálneho systému. Takýto výsledok nie je v zásade chybný, nakoľko nadbytočné údaje môže používateľ ignorovať, ale je vhodné ich odstrániť už na aplikačnej úrovni. Toto je potrebné vykonať pre každý systém odlišným spôsobom.

tsql2lib Vhodným spôsobom zamedzenia ovplyvnovania sa medzi jednotlivými systémami je skrytie stĺpcov ukladajúcich metadáta. V takomto prípade nie sú skryté stĺpce dostupné v databázovej schéme a teda temporálny systém nevie o ich existencii. Na základe typu tabuľky však vie aké metadáta by tabuľka mala uchovávať, čiže ich vie na základe presného názvu adresovať aj keď sú skryté. Pri takomto prístupe je nutnosťou aby mali všetky operácie explicitne vyjadrené názvy stĺpcov a nespoliehalo sa len na implicitné priradenie na základe poradia v databáze.

TimeDB Pre tento databázový systém je možné vyjadriť oblasť jeho záujmu už pri vytváraní tabuľky a definovaní jej metadát. Toto je zaručené tým, že ak sa neuvedie názov stĺpca do tabuľky `TABLE_VIEW_SCHEMES`, systém sa k nemu správa ako keby neexistoval. Identifikovanie verejne dostupných stĺpcov je nutné vykonať už pri vytváraní tabuľky na základe porovnania názvu stĺpca s rezervovaným názvom iného temporálneho systému. Problémom môže byť, že tento systém používa implicitné vyjadrenie názvov stĺpcov pri príkaze `INSERT`. Z tohto dôvodu je potrebné vykonať preprogramovanie metódy, ktorá má túto operáciu na starosti tým spôsobom, že sa pridajú reálne

názvy stĺpcov.

Oracle ValidTime V systéme Oracle ValidTime je možné postupovať podobne ako pri tsql2lib a využiť skývanie stĺpcov na databázovej úrovni ich vyradením zo schémy. Metadáta tohoto systému nie je potrebné skrývať nakoľko to je vykonané automaticky. Ak používateľ zadá názvy stĺpcov kde sa majú uložiť hodnoty začiatku a konca času platnosti zostanú tieto viditeľné. Toto správanie je však korektné, nakoľko vyjadruje, že používateľ chce aby tieto stĺpce boli dostupné ako dátové položky.

Pri spracovaní všetkých týchto príkazov sa stanú jednotlivé systémy vzájomne transparentné. Súčasne sa aj navzájom prepoja na pozadí, či už pomocou aplikačnej vrstvy JDBC ovládača alebo pomocou definovaných databázových operácií.

3.4 Grafická konzola

Pre demonštrovanie funkčnosti aplikácie je vhodné vytvoriť grafickú konzolu, ktorá umožní zadávanie temporálnych dopytov. Keďže je nutné zadávanie dopytov pre rôzne systémy, musí táto konzola umožniť používateľovi zvoliť ktorému systému má byť príkaz zaslaný. Rovnako tak musí konzola umožniť zobrazovanie výsledkov jednotlivých dopytov v prehľadnej tabuľkovej podobe. Nakoľko databázové skripty sa väčšinou skladajú z viacerých príkazov a teda môžu mať viac výsledkov je nutné tieto zobraziť v záložkovej podobe. Konzolová aplikácia musí zadaný skript vnútorne rozdeliť na jednotlivé príkazy a tieto následne klasifikovať do jednotlivých typov požiadaviek aby bolo možné určiť či je očakávaný výsledok alebo je len vykonaná zmena v databáze. Konkrétne operácie s databázou môžu byť vykonávané pomocou JDBC ovládača, ktorý je tiež súčasťou tejto diplomovej práce.

Vykonávané skripty sa často opakujú a preto je potrebné, aby používateľ mohol prehliadať históriu ním zadaných príkazov a výsledky, ktoré boli získané z databázy. Keďže uchovávanie histórie môže byť pri existencii väčšieho počtu dopytov pamäťovo náročné je vhodné, aby bolo možné definovať históriu len na obmedzený počet záznamov. V tomto prípade môžu byť uchované len samotné dopyty vo forme textových reťazcov čo je alternatíva, ktorá nepotrebuje až také veľké množstvo pamäte. Druhou možnosťou je uchovávanie celých dopytov vrátane odpovedí. Takýto prístup môže byť pamäťovo náročný ale v konečnom dôsledku dokáže ušetriť čas a výpočetný výkon. Toto je možné dosiahnúť vďaka tomu, že pri analýze údajov používateľ nemusí znovu opakovať požiadavku na databázový server. Stačí keď si prehliadne históriu a vidí v minulosti získané údaje. Vo fáze testovania je potrebné vykonať merania a zhodnotiť aké veľké pamäťové nároky má ukladanie výsledkov.

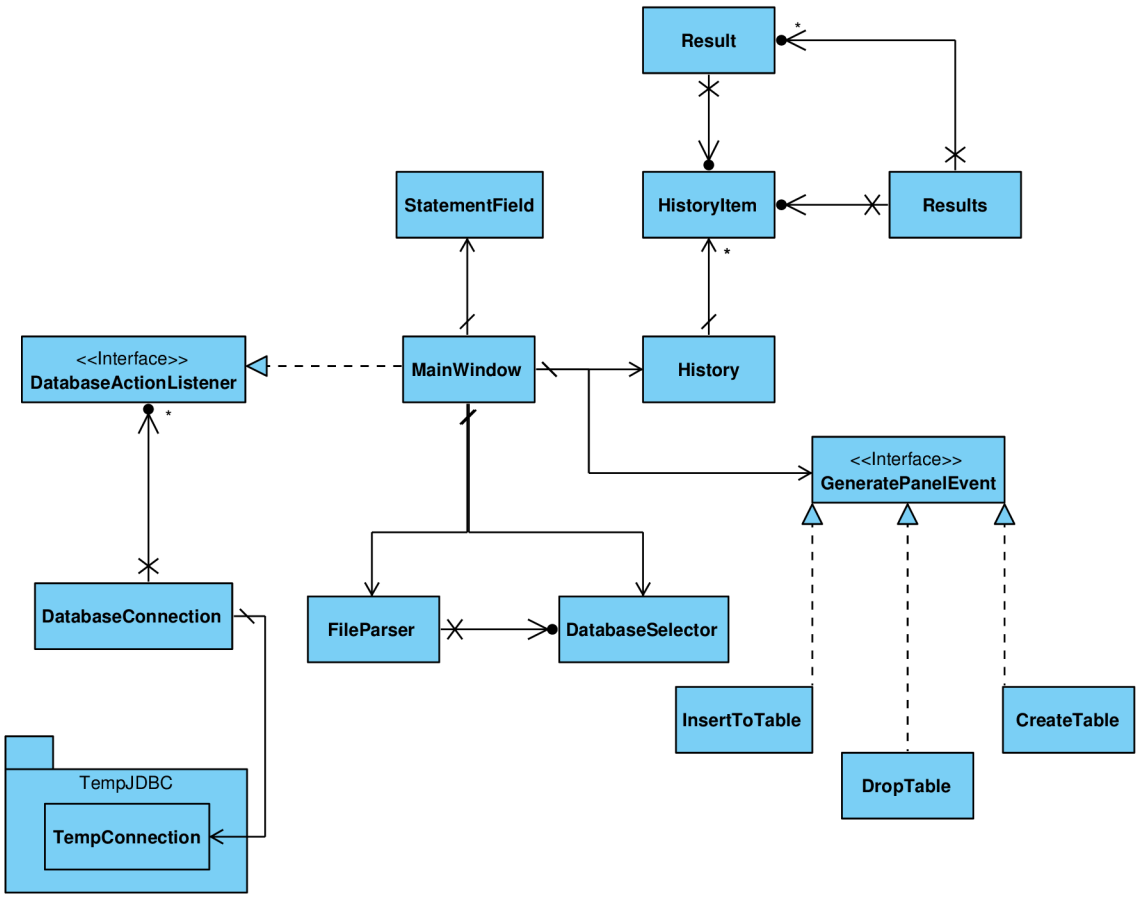
Pre zjednodušenie práce používateľa s aplikáciou je vhodné umožniť mu prerušiť prácu, ktorú vykonáva a uložiť si jej stav na nasledujúce vykonávanie. Z tohoto dôvodu by mala

byť aplikácia schopná uložiť celý dopytovací skript do súboru a neskôr umožniť jeho načítanie. Spolu s dopytmi je však potrebné uchovávať aj typ jazyka v akom sú napísané. V prípade, že by táto informácia chýbala musel by používateľ zvoliť správny databázový systém, nakoľko analýza príkazov a automatické pridelenie databázového systému by mohlo byť nejednoznačné. Informácia o použitom jazyku môže byť uchovaná ako nová súborová prípona napríklad `*.tsql` alebo `*.atsql`. Problémom takéhoto riešenia je, že by vytváralo zbytočne nové koncovky súborov v operačnom systéme, ktoré by boli pre používateľa neznáme a mätúce, aj keď obsahom týchto súborov by bola len rozšírená verzia SQL skriptu. Alternatívou je definovanie vlastného formátu napríklad v XML kde by bol definovaný element obsahujúci informáciu o použitom jazyku a druhý element by niesol samotný text dopytu. Ani toto riešenie sa však nejavý ako najvhodnejšie, lebo súbor sa stane neinterpretovateľným v iných aplikáciách a zníži sa aj jeho prehľadnosť pre používateľa pri otvorení v textovom editore. Ako najvhodnejšia sa javí alternatíva uchovávajúca doplňujúce informácie vo formáte komentára na začiatku súbora. Jej výhodou je, že umožňuje kompatibilitu s inými systémami podporujúcimi spracovanie SQL dopytov, nakoľko pre tieto systémy by to bol obyčajný používateľský komentár zadaný človekom. Tento prístup navyše zachováva aj prehľadnosť výsledného súboru, pretože editory pre SQL dokážu aj naďalej zvýrazňovať syntax SQL jazyka.

Aby mohli so systémom pracovať aj osoby, ktoré nepoznajú syntax jazykov používaných jednotlivými systémami je vhodné vytvoriť jednoduché grafické rozhranie. V tomto by by bolo možné jednotne naklikať parametre dopytu a následne na základe zvoleného systému vygenerovať konkrétny dopyt. Týmto rozšírením sa zvýši aj demonstračný prínos aplikácie.

Návrh konzolovej časti aplikácie je možné vidieť na obrázku 3.5 vo forme doménového diagramu.

Podobný prístup ako pri grafickom rozhraní je potrebné implementovať aj pre textové rozhranie, čo umožní prístup k systému napríklad cez vzdialený terminál alebo v obmedzenom textovom režime. V tomto prípade stačí jednoducho umožniť prepínanie medzi jednotlivými temporálnymi systémami príkazom, ktorý nemá konflikt s akýmkoľvek SQL dopytom. Príkladom takéhoto príkazu je napríklad `OPEN názov_systému`.



Obr. 3.5: Doménový model GUI

Kapitola 4

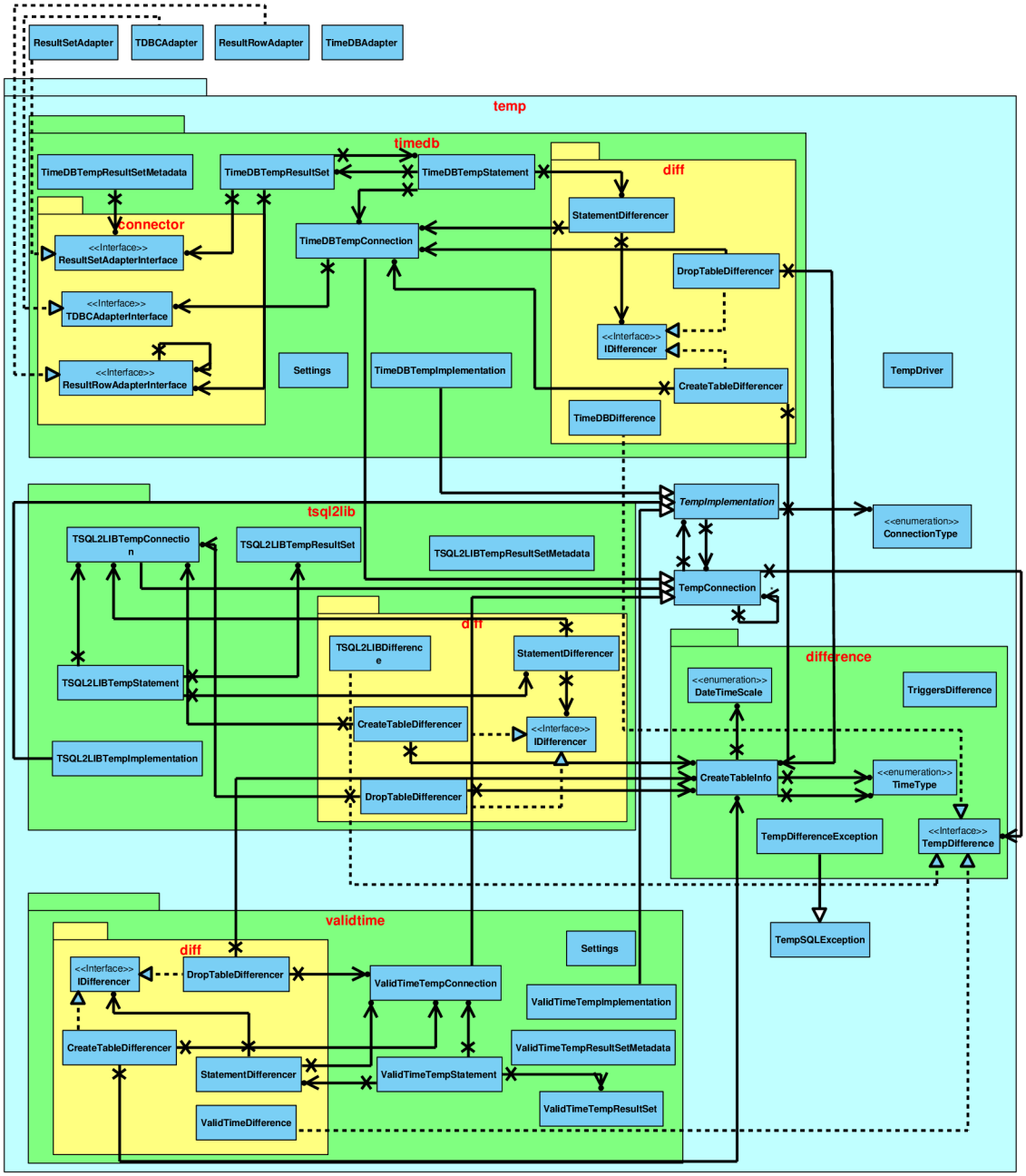
Implementácia

Keďže cieľom práce je vytvoriť JDBC ovládač, ktorý je technológiou pre programovací jazyk Java, bolo jednoznačné, že aj implementácia bude vykonaná v tomto jazyku. Celý proces tvorby bol starostlivo naplánovaný a následne navrhovaný pomocou UML diagramov, čím sa minimalizovali zmeny implementácie za behu len na nutné neočakávateľné úpravy. Návrh práce sa sústredil na jednoduchú udržateľnosť funkčnosti pri zmene podradených temporálnych systémov. Rovnako v návrhu bolo myslené na možnosť doplnenia nového systému bez nutnosti väčších zásahov do už existujúcej implementácie. Tieto podmienky boli prenesené aj do fázy implementácie kde boli reálne vyskúšané. Postup implementácie prebiehal tak, že najskôr bol vytvorený JDBC ovládač pre dva temporálne systémy a dodatočne bola pridaná podpora pre tretí systém. Toto bolo vďaka vhodnej architektúre možné len s minimálnou modifikáciou zdieľaných tried.

4.1 JDBC ovládač

Najväčšou časťou práce je implementácia JDBC ovládača. Jedná sa o jadro projektu vychádzajúceho zo všeobecného návrhu z kapitoly 3. Tento návrh bolo potrebné previesť do implementačného modelu tried na obrázku 4.1. Ako základ systému bolo vhodné zvoliť architektúru abstraktnej továrne pre možnosť pridávania implementácie nových temporálnych systémov pomocou balíčkov implementujúcich definované rozhranie. Toto definované rozhranie sa muselo stotožňovať s rozhraním JDBC a definovať rozširujúce operácie pre prácu s viacerými databázovými systémami.

Implementácia umožňuje dva prístupy k pripojeniu na databázu. Prvým spôsobom je pripojenie pomocou triedy `TempConnection`, kde je potrebné v konštruktore špecifikovať o aký typ pripojenia sa bude jednať. V tomto prípade prichádza na rad abstraktná továreň v podobe triedy `TempImplementation`, ktorá zavolá konkrétnu továreň definovanú v zozname v triede `ConnectionType`. Vytvoreným objektom je pripojenie na daný da-



Obr. 4.1: Diagram tried pre JDBC ovládač

tabázový systém a následne sa už pre vykonávanie operácií používajú triedy definované v konkrétnom balíčku. Druhým spôsobom pripojenia je možnosť inicializovať priamo triedu implementujúcu rozhranie `Connection`, čím sa obchádza vytváranie objektov pomocou továrne. Pri pridaní podpory pre nový systém je potrebné zabezpečiť aby jeho implementácia obsahovala implementáciu už spomenutej abstraktnej triedy `TempImplementation` a `Connection`. Tieto dve triedy je potrebné pridať do zoznamu `ConnectionType`, ktorý za-

bezpečuje vytváranie sekundárnych pripojení pre zachovanie konzistencie zmien pre každý systém. Z tohoto dôvodu musí balíček obsahovať implementáciu reakcie na každý podporovaný typ príkazu. Jednotlivé reakcie vychádzajú z vytvoreného návrhu a sú súčasťou balíčka daného databázového systému. Spoločne sa spracúvajú len dopyty vkladania záznamu, modifikácie a temporálneho zneplatnenia, nakoľko tie sa vykonávajú v databázovom systéme pomocou spúšťača. Preto pri pridaní nového systému je potrebné upraviť aj túto definíciu v triede `TriggersDifference`. Spracúvajúci spúšťač bolo potrebné rozdeliť na dve rôzne časti kde jedna spracováva operácie vkladania a hľadá stĺpec, ktorý obsahuje nenulové dáta, ten potom berie ako referenčný. Úlohou druhej časti je spracovať zmenové operácie tak, že hľadá, ktorý stĺpec nadobudol inú hodnotu tak ten sa stane referenčným pre zmenu údajov v ostatných stĺpcoch.

tsql2lib

Implementácia komunikácie pre tento temporálny databázový systém sa nachádza v balíčku `temp.tsql2lib`. Tento balíček obsahuje všetky potrebné implementácie továrne a databázového konektora. Nachádzajú sa v triedach `TSQL2LIBTempImplementation` a `TSQL2LIBTempConnection`. Súčasťou balíčka sú aj implementácie jednotlivých JDBC rozhraní. Ich úloha je daná definíciou rozhrania, navyše však implementujú volania sekundárnych pripojení s parametrami operácie požadovanej používateľom prevedených do univerzálnej štruktúry `CreateTableInfo`. Operácie potrebné vykonať pri vracaní výsledkov sú implementované v triedach `TSQL2LIBTempResultSet` a `TSQL2LIBTempResultSetMetadata`.

Každý dopyt vstupujúci do systému prechádza cez triedu `StatementDifferencer`, ktorá ho analyzuje a zaradí do príslušnej kategórie na základe návrhu. Po kategorizovaní je pre daný dopyt vytvorený objekt príslušného typu `CreateTableDifferencer` alebo `DropTableDifferencer`. Tu sa dopyt bližšie analyzuje a získajú sa z neho všetky potrebné informácie pre vykonanie operácií v ostatných databázových systémoch. Tieto rozdielove operácie sú definované v triede `TSQL2LIBDifference`.

TimeDB

Implementácia pre podporu tohoto systému sa nachádza v balíčku `temp.timedb`. Rovnako ako tsql2lib obsahuje implementáciu továrne v triede `TimeDBTempConnection` a konektora v triede `TimeDBTempImplementation`. Problémom implementácie tohoto systému je, že nedefinuje štandardné JDBC rozhranie a preto je potrebné všetky operácie ním požadované operácie vyskladať len z obmedzenej škály metód. Pre implementáciu posunu kurzora vo výsledku je nutné pre triedu `TimeDBResultSet` implementovať len pomocou metód `firstRow()` a `nextRow()` z triedy `ResultSetAdapter`. Následné získavanie hodnôt je možné len pomocou metódy `getColumnValue()`, čo je tiež potrebné rozšíriť na čo možno

najviac operácií JDBC rozhrania. Tento systém nemá od tvorcov implementované rozhranie, ktoré by poskytovalo názvy jednotlivých stĺpcov. Toto však JDBC rozhranie požaduje ako základnú funkcionalitu, preto bolo potrebné to do samotného systému TimeDB doprogramovať pridaním metódy `getHeaders()`, ktorá vracia vektor názvov všetkých stĺpcov obsiahnutých vo výsledku. Túto metódu je už možné v implementovanom ovládači upraviť na rôzne metódy JDBC rozhrania.

Ďalším veľkým problémom tohoto systému je, že nedokáže pracovať v akomkoľvek Java balíčku ale len v koreňovom adresári. Toto bolo potrebné eliminovať pridaním priechodných adaptérov, ktoré majú rovnaké rozhranie ako definuje temporálny systém a ich implementácia len opakuje volania metód priamo do reálnej implementácie systému.

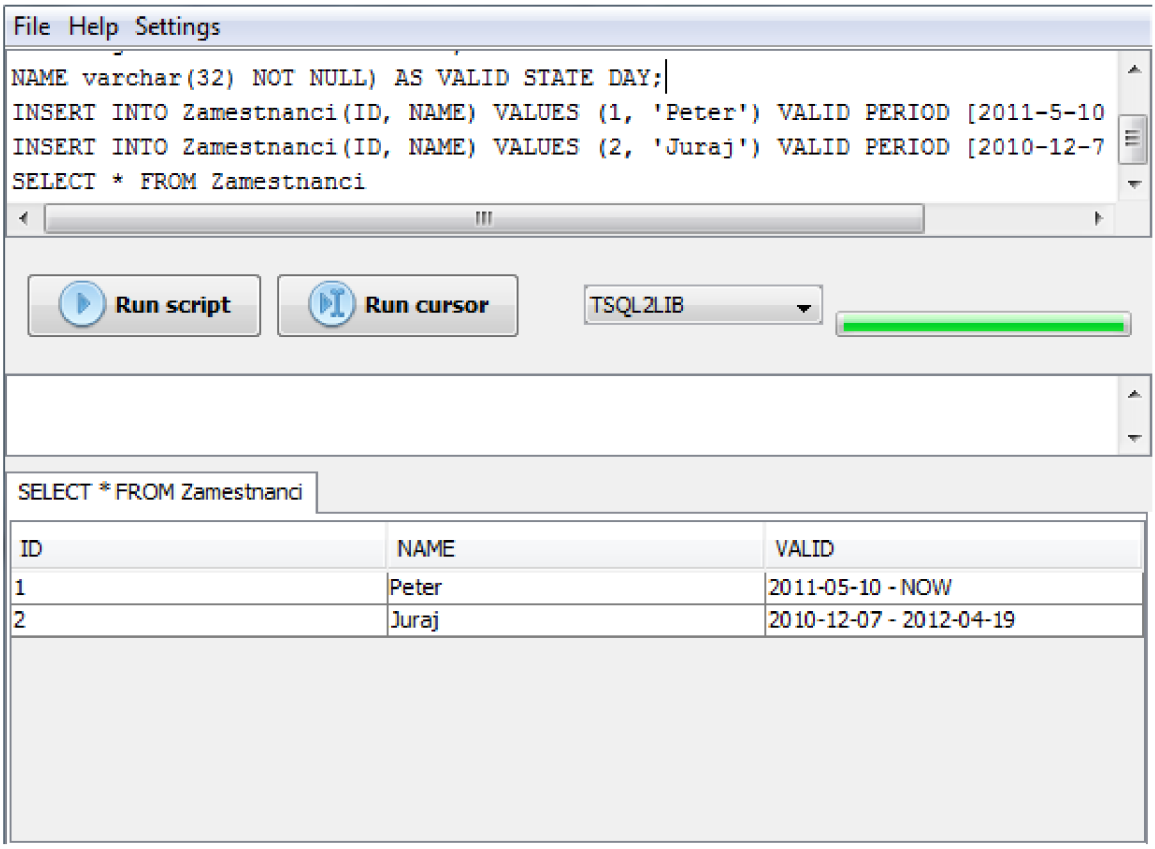
Zvyšné časti implementujúce spracovanie dopytov sú rovnaké ako v systéme tsql2lib.

Oracle ValidTime

Tento systém je implementovaný v balíčku `temp.validtime`. Tiež obsahuje továrenskú triedu `ValidTimeTempConnection` a triedu konektora `ValidtimeTempImplementation`. Nakoľko sa jedná priamo o typ spojenia pomocou JDBC, jednotlivé metódy často volajú len samotnú základnú implementáciu. Rozšírenie spočíva len v prenose používateľom definovaného dopytu aj na iné temporálne systémy. Mnoho operácií je implementovaných podobne ako v tsql2lib. Navyše bolo potrebné dodefinovať pomocnú tabuľku s názvom `_ORACLE_VALIDTIMES_TABLES`, kde sa ukladajú názvy stĺpcov uchovávajúcich čas platnosti. V prípade, že iný systém chce zistiť či je daný stĺpec dátový používateľom definovaný alebo sa jedná o metadáta len jednoducho pristúpi do tejto tabuľky a pozrie sa či tam existuje záznam pre daný stĺpec.

4.2 Grafické rozhranie

Cieľom bolo aj vytvorenie aplikácie demonštrujúcej použiteľnosť navrhnutého ovládača. Táto bola navrhnutá a následne naprogramovaná, aby čo možno v najväčšej miere využívala potenciál vytvoreného ovládača. Umožnuje používateľovi definovať jedno pripojenie k databáze a následne len prepínať medzi všetkými dostupnými temporálnymi databázami. Neimplementuje žiadne špecifiká akéhokoľvek temporálneho systému, ale všetko deleguje na JDBC ovládač. Z podobného dôvodu umožnenia následnej rozšíriteľnosti si pri spustení načíta všetky podporované temporálne systémy aby umožnila intuitívne prepínanie medzi nimi. Preto nie je nutné pri pridaní nového databázového systému do ovládača nijakým spôsobom modifikovať grafickú konzolu. Demonstračná aplikácia sa snaží minimalizovať úsilie, ktoré musí používateľ vytvoriť vhodným umiestnením najčastejšie používaných ovládacích prvkov.



Obr. 4.2: Snímok grafického rozhrania

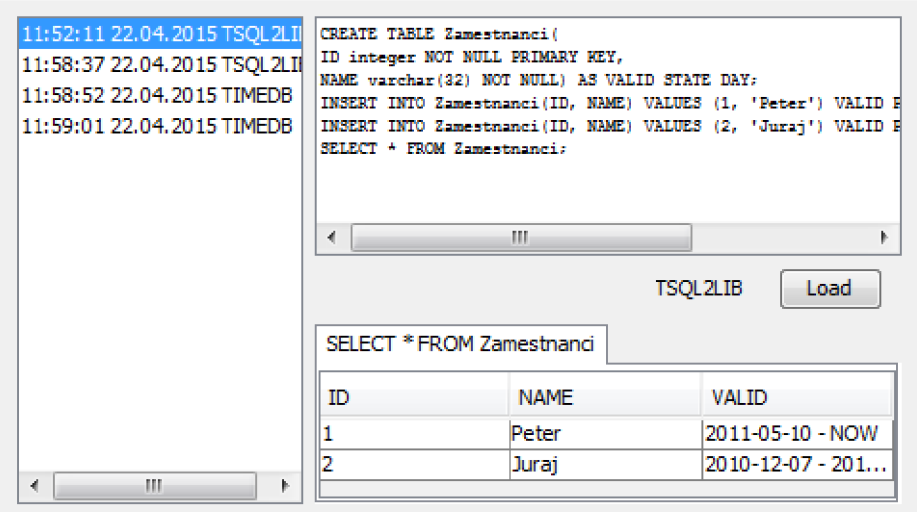
4.2.1 Objektová implementácia

Podľa návrhu na obrázku 3.5, bolo implementovaných niekoľko tried pre spracovanie výsledkov a ich následnú interpretáciu. Najhlavnejšou triedou je `DatabaseConnector`, ktorá vykonáva všetky operácie nad databázou. Všetky operácie, ktoré majú dlhé vykonávanie spracúva v osobytných vláknach a o ich priebehu informuje grafické rozhranie pomocou návrhového vzoru Poslucháč.

Problémom pri implementácií bolo spúšťanie jednotlivých príkazov zadaných používateľom. V tomto prípade bolo potrebné rozdeliť skript na jednotlivé príkazy. Vďaka obmedzeniu podpory jednotlivých temporálnych systémom len na jednoduché operácie bez zanorenia bolo možné použiť k rozdeleniu scriptu regulárny výraz. Tento výraz sa snaží nájsť jednotlivé príkazy oddelené bodkočiarkou. V regulárnom výraze bolo navyše potrebné ošetrenie zadanej bodkočiarky napríklad v textovom reťazci alebo v komentároch kde nenadobúdala význam oddeľovača. Po tom ako je vstupný skript rozdelený na kolekciu jednoduchých príkazov je možné vykonať nad nimi odstránenie príkazov nachádzajúcich sa pred grafickým kurzorom v textovej oblasti a vykonať spustenie len určitej časti skriptu. Tu bolo potrebné vyriešiť problém kedy sa kurzor nachádza uprostred nejakého príkazu aby sa aj tento zahr-

nul do výsledku. Z tohoto dôvodu nebolo možné jednoducho odstrániť celý text umiestnený pred pozíciou kurzora, ale bolo nutné vykonať rozbor celého vstupu a až následné odstránenie príkazov nachádzajúcich sa pred pozíciou označenou používateľom.

Aplikácia umožňuje používateľovi uchovávanie vykonaných príkazov bez toho aby si ich musel manuálne ukladať. Tohoto bolo docielené tým, že aplikácia uchováva celý spustený skript a typ databázového systému, na ktorom bol tento skript spustený. Pre demonštráciu a odstránenie nutnosti opätovného spustenia si aplikácia uchováva aj výsledok získaný z databázy. Tento výsledok je samozrejme uchovaný z času vykonania dopytu a preto nemusí byť už aktuálny a je na zvážení používateľa či je relevantný alebo nie. Práve pre prípad nutnosti zopakovania behu aplikácia jednoduchým spôsobom umožní nastavenie pôvodne použitého databázového systému a načítanie príkazu pripraveného na opätovné spustenie. Všetky operácie spojené s uchovávaním histórie obsluhuje trieda `History`, ktorá uchováva všetky operácie v zozname s obmedzenou dĺžkou. Túto dĺžku môže špecifikovať používateľ na základe posúdenia potreby uchovávania histórie a veľkosti dostupnej voľnej operačnej pamäte. V prípade, že sa vykoná nový príkaz najstarší záznam zo začiatku fronty je zahodený a na koniec sa pridá aktuálne vykonaný. Príklad rozhrania pre zobrazovanie histórie je možné vidieť na obrázku 4.3.



Obr. 4.3: Snímok dialógového okna zobrazujúceho históriu používateľovi

Aby bolo možné používateľské skripty spúšťať aj po určitej dobe bolo potrebné implementovať export a import dopytov. Je vhodné aby exportovaný súbor mal validný formát SQL vrátane konkrétneho temporálneho rozšírenia, navyše je však potrebné uchovať informáciu o aký temporálny jazyk sa jedná. V tomto prípade bolo z navrhovaných možností vybraté ukladanie doplňujúcich metadát vo forme SQL komentára na prvom riadku súbora.

Kapitola 5

Testovanie a optimalizácia

Aplikácia bola počas implementácie priebežne testovaná na navrhnutej sade testov. Tieto pomáhali pri odhaľovaní chýb pri vývoji a neskôr ich je možné použiť na demonstračné účely aplikácie. Používaným databázovým systémom bol primárne Oracle 12c pre jeho nevyhnutnosť pri používaní systému Oracle ValidTime a pre možnoť skrývania stĺpcov v tabuľke. Pre iné systémy nemá zmysel aplikáciu testovať, nakoľko temporálny databázový systém tsql2lib primárne podporuje len MySQL, ktorá však neumoňuje úpravu databázovej schémy skytím jednotlivých stĺpcov. V dôsledku tohoto by pre ostatné databázy, ako napríklad Cloudspace a Sybase, zostala podpora len jedným temporálnym systémom TimeDB a v takom prípade už nemá zmysel vykonávať žiadne spájanie.

Súčasťou finálneho testovania aplikácie bolo meranie výkonnosti a pamäťovej náročnosti najzložitejších operácií. Pri JDBC ovládači bolo vykonané meranie vplyvu zaradenia medzivrstvy do behu vykonávania dopytov.

5.1 Výkonnosť JDBC ovládača

Najdôležitejšou otázkou testovania bolo určiť aký veľký vplyv na výpočtový čas má to, že každý príkaz musí byť predspracovaný ďalšou medzivrstvou. Pre minimalizáciu vedľajších vplyvov pri testovaní nebolo použité grafické rozhranie ale napojenie priamo na jednoúčelovú konzolovú aplikáciu so zabudovanými dopytmi. Meranie bolo vykonané opakovane pre každý typ dopytu a následne rozdiely boli zanesené do tabuľky 5.1. Všetky vykonané dopyty boli jednoduché aby sa eliminoval vplyv používateľských dát na testovanie. Bola vytváraná tabuľka s jedným číselným záznamom. Nad ňou boli vykonané operácie vkladania a úpravy a následne aj operácia výberu. Z nameraných rozdielov vo výkonnosti je možné konštatovať, že implementované spojenie ovplyvňuje najčastejšie vykonanávané operácie s manipuláciou dát len minimálnou mierou. Najvačší vplyv na čas behu bol zaznamenaný pri vytváraní tabuľky alebo jej mazaní, čo nie je až taký veľký problém nakoľko tieto operácie sa vykonávajú

len sporadicky.

Typ dopytu	tsql2lib	TimeDB	Oracle ValidTime
Tvorba tabuľky	54	32	69
Odstránenie tabuľky	24	17	27
Vkladanie záznamu	4	4	4
Získanie výsledku	4	6	2

Tab. 5.1: Rozdiel v čase získanom pomocou implementovaného JDBC ovládača a priamo nad daným systémom. Výsledky meraní jednotlivých dopytov sú v ms.

Ďalším dôležitím parametrom pri meraní bolo určenie pamäťových nárokov v databázovom systéme pre každý implementovaný temporálny systém pri zanedbaní používateľom definovaných dát. Toto meranie má opodstatnenie pre určenie nárastu objemu databázy pri použití implementovaného rozšírenia. Výsledky z merania a analýzy sú uvedené v tabuľke 5.2 a je možné z nich určiť, že pri použití všetkých podporovaných systémov sa veľkosť jedného záznamu zväčší až o 54 bytov, čo je pri milóne záznamov redundancia 51,5 MB dát.

Názov databázy	Objem dát
tsql2lib	4 * 2 = 8
TimeDB	8 * 2 = 16
Oracle ValidTime	13 * 2 + 4 = 30

Tab. 5.2: Približný objem dát ukladaný pre jednotlivé temporálne databázové systémy v bytoch pri použití času platnosti.

5.2 Výkonnosť grafickej konzoly

Pri behu grafickej konzoly boli vykonané viaceré merania, ktoré sa neskôr prejavili ako zanedbateľné. Príkladom takéhoto merania bolo posúdenie vplyvu grafickej konzoly na čas behu databázového dopytu. To, že bola komunikácia vedená cez grafické rozhranie nijako nezmenilo, nakoľko priemerné namerané časy sa zhodovali s priemernými časmi nameranými v predchádzajúcej časti 5.1.

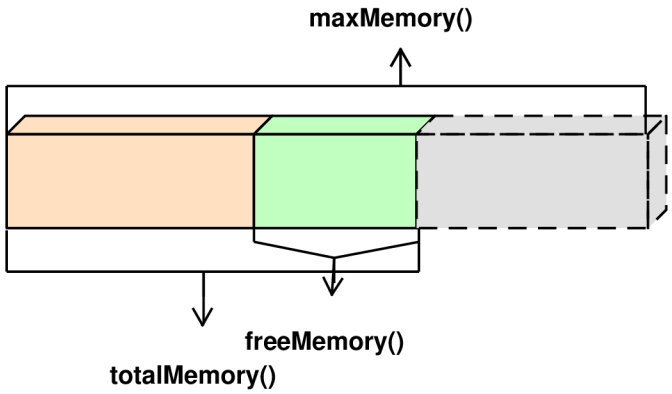
Ako podstatné sa prejavilo meranie pamäťovej náročnosti ukladania histórie jednotlivých spustení skriptu. V tabuľke 5.3 je možné vidieť spotrebu pamäte v porovnaní s veľkosťou výsledku a počtom záznamov v pamäti. Výsledkom merania je konštatovanie, že história má v porovnaní so zvyškom aplikácie značný vplyv na spotrebu pamäte ale táto

nie je až taká enormná v porovnaní s veľkosťou pamäte inštalovanej do súčasných počítačov.

Záznamov/ Jednotková veľkosť	10 riadkov po 5 stĺpcov	100 riadkov po 5 stĺpcov	1000 riadkov po 5 stĺpcov
1 záznam	0,4	1,2	2,3
10 záznamov	3,5	8,1	15,8
100 záznamov	25,2	69,7	94,5

Tab. 5.3: Približná spotreba pamäte v porovnaní s veľkosťou výsledku a veľkosťou histórie v MB.

Samotné meranie spotreby pamäte prebiehalo pomocou metód objektu `Runtime`. Tento definuje tri metódy pre získavanie informácií o pamäti. Sú to metódy `maxMemory()`, ktorá vracia maximálnu dostupnú pamäť pre spustený virtuálny stroj, `totalMemory()`, ktorá vracia veľkosť alokovanej pamäte pre spustený program a `freeMemory()`, ktorej výsledkom je veľkosť voľnej časti v alokovanom priestore. Ich význam je znázornený na obrázku 5.1, z ktorého je možné vyčítať, že na to aby sme získali veľkosť použitej pamäte je potrebné vykonať príkaz `usedMemory = Runtime.getRuntime().totalMemory() - Runtime.getRuntime().freeMemory()`. Takýto výpočet je potrebné z dôvodu, že Java vykonáva doprednú alokáciu priestoru pre jednotlivé objekty a nie až vtedy keď naozaj vzniknú. Pre odstránenie konštantnej spotreby bolo potrebné vykonať meranie spotreby pred spúšťaním skriptom a po ich dokončení, kde výsledkom veľkosť spotrebovanej pamäte je rozdielom týchto dvoch hodnôt.



Obr. 5.1: Rozdelenie typov pamäte v Java Runtime

Kapitola 6

Záver

Táto práca obsahuje prehľad existujúcich temporálnych databázových systémov. Rovnako je jej súčasťou skupina znalostí potrebných pre vývojára pracujúceho s týmito databázami. Keďže program bol napísaný v Jave bolo potrebné vykonať štúdium ohľadom možnosti komunikácie medzi týmto programovacím jazykom a databázovým systémom.

Hlavným výsledkom tejto práce je implementovaný JDBC ovládač správajúci sa ako medzivrstva pre odstránenie rozdielov medzi rôznymi temporálnymi databázovými systémami. Podarilo sa vytvoriť architektúru, ktorá umožňuje jednoduché pridávanie nových systémov bez nutnosti väčších zmien v už existujúcich triedach. Pridanie je možné vykonať dodefinovaním nového balíčka implementujúceho navrhnuté rozhranie a jeho následného pripojenia do zoznamu podporovaných systémov. Aby bolo možné vytvoriť tento ovládač bolo potrebné pre systém TimeDB nanovo doimplementovať podporu JDBC rozhrania, nakoľko pôvodné rozhranie malo len obmedzené operácie.

Najväčším problémom pri vytváraní práce bolo, že sa pri temporálnych databázových systémoch väčšinou jedná o nedokončené výskumné projekty, prípadne ich vývoj bol ukončený a už nie je možné ich použiť. Toto bol problém aj pri vytváraní tejto práce, nakoľko bol ukončený vývoj systému TempDB a už ho nebolo možné použiť. Ako vhodnou, aj keď nie úplnou, náhradou sa stal komerčný systém ValidTime od spoločnosti Oracle. Tento bude pravdepodobne v nových verziáchešte rozširovaný, nakoľko je aktuálne vo svojej prvej verziii, a stane sa plnohodnotným temporálnym databázovým systémom. Problémy zistené ohľadom podpory niektorých funkcií jednotlivých systémov bohužiaľ nebolo možné odstrániť a ani to nebolo cieľom práce. Implementácia zjednotenia sa však snaží pri nefunkčných častiach držať dostupných dokumentácií k týmto systémom. V oblastiach kde nebolo možné doimplementovať podporu sa snaží ovládač pozerať na tento systém ako keby takúto funkcionalitu mal. Dôvodom k takémuto prístupu je možnosť, že by sa v budúcnosti autori databáz rozhodli svoj systém dokončiť. Vhodným príkladom je spojenie dát pre podporu transakčných časov v jednotlivých systémoch.

Práca bola starostlivo otestovaná a bol vyhodnotený jej dopad na výkonnosť databázových operácií.

Dôležitou súčasťou práce je grafická konzola umožňujúca intuitívnu prácu s podporovanými temporálnymi databázami. Pre prístup k rovnakej funkcionalite bez grafického prostredia je možné použiť vyvinutú konzolovú aplikáciu, ktorá funguje len v textovom režime.

6.1 Budúci vývoj

Ako možné pokračovanie v práci navrhujem pridanie nových temporálnych databázových systémov bežiacich nie len na databázových serveroch spoločnosti Oracle, ale napríklad aj na serveri od spoločnosti Microsoft s využitím podpory rozšírenia tempdb. Rovnako je možné implementovať preklad integritných obmedzení, ak by takúto funkcionalitu jednotlivé systémy začali podporovať. Ďalším možným pokračovaním je implementácia modulu, ktorý by mal zadefinované nepodporované operácie pre jednotlivé databázové systémy a prípadne by ich riešil vo svojej vlastnej réžii. Tento modul je ale potrebné implementovať ako zoznam jednotlivých úkonov aby sa dali pri aktualizácii databázového systému jednoducho vypnúť.

Literatura

[1] *Learn About Java Technology*, [online]. 1. 11. 2014 [cit. 2014-11-29]. Dostupné z: http://www.java.com/en/about/

[2] *Java™ SE Technologies - Database*, [online]. 31. 1. 2014 [cit. 2014-11-29]. Dostupné z: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html

[3] *JDBC Overview*, [online]. 13. 5. 2014 [cit. 2014-11-28]. Dostupné z: http://www.oracle.com/technetwork/java/overview-141217.html

[4] *JDBC-ODBC Bridge*, [online]. 2014 [cit. 2014-11-30]. Dostupné z: http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/bridge.html

[5] *Java™ JDBC API*, [online]. 2014 [cit. 2014-11-28]. Dostupné z: http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/

[6] *TimeConsult Software*, [online]. 2014 [cit. 2014-12-14]. Dostupné z: http://www.timeconsult.com/Software

[7] ABADI, M.; Zohar, M.: Temporal logic programming. *Journal Symbolic Computation*, ročník 8: s. 277–295.

[8] ANTAL T., A.: Access to an ORACLE database using JDBC. *Applied Mathemathics and Mechanics*, ročník 3, č. 47, 2004: s. 63–68, ISSN 1221-5872.

[9] BÖHLEN, M. H.: *The Temporal Deductive Database System ChronoLog*. Dizertační práce, Department Informatick, ETH Zürich, 1994.

[10] BÖHLEN, M. H.: Temporal Database System Implementations. *SIGMOD Rec.*, ročník 24, č. 4, Prosinec 1995: s. 53–60, ISSN 0163-5808.

[11] CHOMICKI, J.; TOMAN, D.; BÖHLEN H., M.: Querying ATSQL Databases with Temporal Logic. *ACM Trans. Database Syst.*, ročník 26, č. 2, Červen 2001: s. 145–178, ISSN 0362-5915.

[12] Codd, E. F.: A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, ročník 13, č. 6, Červen 1970: s. 377–387, ISSN 0001-0782.

[13] COOPER W., J.: *The Design Patterns Java Companion*. United States: Addison-Wesley, Oct 1998, 218 s.
Dostupné z: http://software.ucv.ro/~cstoica/MPV/DesignJava.pdf

[14] FISHER, M.; ELLIS, J.; BRUCE, J.: *JDBC™ API Tutorial and Reference*. Addison Wesley, třetí vydání, Červen 2003, ISBN 0-321-17384-8, 1280 s.

[15] JENSEN, C. S.; SNODGRASS, R.: Temporal data management. *Knowledge and Data Engineering*, ročník 11, č. 1, Jan 1999: s. 36–44, ISSN 1041-4347.

[16] KOSTENKO, B.: Temporal Preprocessor: Towards Temporal Applications Development. Moscow, Russia: Colloquium on Databases and Information Systems, 2007, ISSN 1613-0073.

[17] KOUBARAKIS, M.; SELLIS, T. K.; FRANK, A. U.; et al. (editoři): *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, *Lecture Notes in Computer Science*, ročník 2520, Springer, 2003, ISBN 3-540-40552-6.

[18] ORGUN, M. A.; WADGE, W. W.: *Chronolog: A temporal logic programming language and its formal sematics*. Victoria B.C., Canada: Department of Computer Science, University of Victoria, Leden 1988.

[19] SCHILDT, H.: *Java: The Complete Reference*. The McGraw-Hill Companies, 7 vydání, 2006, ISBN 978-0-07-163177-8, 1024 s.

[20] SNODGRASS, R.; AHN, I.; ARIAV, G.; et al.: TSQL2 Language Specification. University of Arizona, Tucson, Sep 1994.
Dostupné z: ftp://ftp.cs.arizona.edu/tsql/tsql2/spec.pdf

[21] SNODGRASS T., R.: *Developing Time-Oriented Database Applications in SQL*. San Francisco: Morgan Kaufmann Publishers, 1999, ISBN 1-55860-436-7, 504 s.
Dostupné z: http://www.cs.arizona.edu/people/rts/tdbbook.pdf

[22] TANG, Y.; YE, X.; TANG, N.: *Temporal Information Processing Technology and Its Application*. Springer Heidelberg Dordrecht London New York, 2011, ISBN 978-3-642-14959-7, 349 s.

[23] TOMEK, J.: TSQL2 interpret nad relační databází. 2009.
Dostupné z: http://www.fit.vutbr.cz/study/DP/DP.php?id=8603&file=t

Příloha A

Obsah CD

Súčasťou práce je aj CD. Popis jeho obsahu je nasledovný:

project.pdf text magisterskej práce

tex-src LaTeXové zdrojové kódy magisterskej práce

src zdrojové kódy implementovaného JDBC ovládača

src-gui zdrojové kódy implemnetovanej grafickej konzoly

src-cli zdrojové kódy implemnetovanej textovej konzoly

doc dokumentácia k aplikácií

test súbory použité pri testovaní