VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

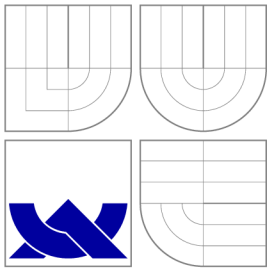
TSQL2 INTERPRET NAD POST-RELAČNÍMI DATABÁZEMI V ORACLE DATABASE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

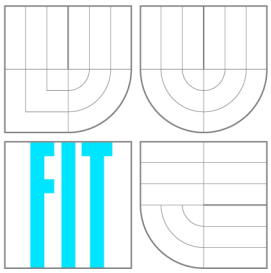
AUTOR PRÁCE
AUTHOR

Bc. JAN SZKANDERA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TSQL2 INTERPRET NAD POST-RELAČNÍMI DATABÁZEMI V ORACLE DATABASE

PROCESSOR OF TSQL2 ON POST-RELATIONAL DATABASES IN ORACLE DATABASE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN SZKANDERA

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. MAREK RYCHLÝ, Ph.D.

BRNO 2011

TSQL2 interpret nad post-relačními databázemi v Oracle Database

Prohlášení

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

.....
Jan Szkandera
26. května 2011

Poděkování

Na tomto místě bych rád poděkoval panu Mgr. Marku Rychlému, Ph.D. za odbornou pomoc, kterou mi poskytoval v průběhu řešení této diplomové práce.
Své přítelkyni děkuji za opravu gramatických chyb a za trpělivost, kterou měla v době, kdy jsem tuto práci řešil.

Abstrakt

Středem zájmu této práce jsou temporální databáze a jejich multimediální a prostorové rozšíření. V úvodu práce jsou shrnuty výsledky výzkumu v oblasti temporálních databází, jsou představeny základní koncepty jazyka TSQL2 a postrelační rozšíření databáze Oracle. Stěžejní částí práce je návrh interpretu jako vrstvy mezi uživatelskou aplikací a relační databází. Práce se dále zabývá kontrolou integritních omezení v temporální databázi a navrhuje její realizaci. Výstupem práce je funkční interpret jazyka TSQL2, který je schopný ukládat postrelační data.

Abstract

This thesis focuses on temporal databases and their multimedia and spatial extensions. The introduction of this work summarizes results in the area of research of temporal databases - key concepts of a TSQL2 language and post-relational extension of Oracle database are introduced. Main part of the thesis is design of an interpreter as a layer between user application and relational database. In the next part of the thesis control of integrity constraints in temporal databases are discussed. Result of this work is functional interpreter of TSQL2 language able to store post-relational data.

Klíčová slova

TSQL2, postrelační databáze, temporální databáze, entitní integrita, referenční integrita, JDBC, Oracle Multimedia, Oracle Spatial

Keywords

TSQL2, post-relational database, temporal database, entity integrity, referential integrity, JDBC, Oracle Multimedia, Oracle Spatial

Citace

Jan Szkandera: *TSQL2 interpret nad post-relačními databázemi v Oracle Database*, diplomová práce, Brno, FIT VUT v Brně, 2011.

„OMNIA HUMANA
BREVIA ET CADUCA SUNT
ET INFINITI TEMPORIS
NULLAM PARTEM OCCUPANTIA.“

„Všechno lidské
je krátkodobé a prchavé
a nezaujímá žádnou část
nekonečného času.“

– Seneca (Ad Marc. 21,1)

Obsah

Kapitola 1

Úvod

Výzkum v oblasti temporálních databází probíhá už přes třicet let. Ukazuje se, že chování temporálních dat se značně liší od dat relačních. Aplikací, ve kterých je využíváno tabulek s časovými hledisky, je velké množství – stačí zmínit finančnictví či lékařství, kde je potřeba uchovávat časově proměnná data.

Možnost skladování multimediálních dat je již v současnosti vyřešena a existují komerční databázové systémy s implementací této podpory. Také prostorová data je dnes již možné efektivně uchovávat a spravovat. Oblasti, ve kterých by bylo možné využít kombinaci temporálních a multimediálních nebo kombinaci temporálních a prostorových dat, jsou například výše zmíněné lékařství nebo katastrální data.

Důvodů proč zavést podporu temporálních dat je tedy velké množství, přesto však stále neexistuje temporální rozšíření SQL – nejpopulárnějšího, a v současné době již standardu, mezi databázovými dotazovacími jazyky a temporální dotazy jsou v čisté formě SQL zbytečně složité.

Problém nedostatečné podpory temporálních dotazů a obecně temporálních dat se pokusil vyřešit Richard T. Snodgrass a jeho spolupracovníci vytvořením jazyka TSQL2. Principy tohoto temporálního jazyka měly být využity v temporálním rozšíření SQL (tzv. SQL/Temporal), ovšem tento projekt je v současnosti zastaven.

V této práci se budeme zabývat právě temporálními databázemi a podporou postrelačních rozšíření uvedených výše. Cílem je vytvoření interpretu jazyka TSQL2, lépe řečeno jeho podmnožiny, který by byl schopen skladovat multimediální a prostorová data ve spolupráci s relační databází.

Kapitola 2 uvede čtenáře do problematiky temporálních databází. Jsou zde zmíněny důležité pojmy z této oblasti, vlastnosti temporálních dat. Je zde stručně popsán jazyk TSQL2 a zpracován přehled současných implementací temporálních databází.

V kapitole 3 jsou představena dvě rozšíření databázového systému Oracle – Multimedia pro skladování multimediálních dat a Spatial pro uchovávání dat prostorových. Rozšíření jsou popsána z hlediska potřeb této práce.

Kapitola 4 slouží k nastínění motivace, která autora vedla k vypracování této diplomové práce. Dále se kapitola zmiňuje o tématech, která podle názoru autora nejsou v současné době dostatečně řešena a těmito tématy se zabývají zbývající kapitoly.

Jednou z nejzásadnějších v této práci je kapitola 5, ve které se pokusíme navrhnout řešení témat zmíněných v předchozí kapitole. Je popsán návrh temporální vrstvy pracující nad relační databází a jejím postrelačním rozšířením. Důležitý je návrh překladu a interpretace konstrukcí jazyka TSQL2, který je popsán v podkapitole 5.3.

Tématem podpory entitní a referenční integrity v temporální databázi se zabývá kapitola 6. Jsou zde rozebrány problémy, které s integritními omezeními souvisí a je zde také navrženo řešení těchto problémů. Kontrola entitní integrity je ve výsledném interpretu implementována. Tato kapitola shrnuje autorovy poznatky, které nejsou zadáním vyžadovány, avšak mají v databázi velmi důležitou roli.

Popis implementace výsledného interpretu lze nalézt v kapitole 7, ve které jsou rozebrány detaily týkající se realizace syntaktické analýzy v interpretu. Dále je zde výčet datových typů a konstant pro realizaci temporálního rozšíření v relační databázi (podkapitola 7.2). Analogicky k podkapitole 5.3, která se zabývá návrhem překladu příkazů jazyka TSQL2 do jazyka SQL, je uvedena podkapitola 7.3, která se tímto problémem zabývá z implementačního pohledu. Na konci kapitoly je rozebrána implementace tříd rozhraní JDBC a v podkapitole 7.5 je věnováno místo problému mazání záznamu z tzv. transakčních tabulek.

V kapitole 8 jsou popsány testy, které byly provedeny pro ověření správnosti návrhu a implementace. Navíc je zde představen nástroj, který byl vytvořen speciálně pro realizaci těchto testů.

Poslední kapitola 9 shrnuje výsledky této diplomové práce a obsahuje popis autorůva přínosu k tématu temporálních databází a jejich postrelačních rozšíření. Kapitola se dále věnuje možným budoucím rozšířením implementovaného interpretu.

Kapitola 2

Temporální databáze

Název kapitoly jmenuje jeden ze dvou problémů, kterým se budeme v této práci věnovat (druhým z nich je skladování multimediálních a prostorových dat v temporální databázi). Pro laika může být překvapující, kolik problémů přináší zavedení časových tabulek do relační databáze. Hlavními problémy temporálních databází je například nutnost použít jiný přístup při používání primárního klíče, složité dotazy či kontrola referenční integrity. Tyto a další problémy budou řešeny v následujících kapitolách.

Úkolem této kapitoly je uvést čtenáře do problematiky. To znamená, seznámit ho s důležitými pojmy a s charakteristickými rysy temporálních dat. Přestaven bude jazyk TSQL2, který je s oblastí temporálních databází spojen. Na konci kaptioly může čtenář nalézt stručný výčet současných implementací temporálních databází.

2.1 Důležité pojmy

V této podkapitole bude popsán pojem temporální databáze a časová hlediska, která obvykle temporální databáze podporují.

2.1.1 Pojem temporální databáze

Než začneme popisovat charakter chování dat v temporální databázi, je dobré si tento pojem blíže specifikovat.

Podle konsenzu [5] předních výzkumníků zabývajících se tímto tématem je temporální databáze definována jako *databáze, která podporuje nějaké hledisko času, jiné než uživatelsky definovaný čas* (viz dále). Tato databáze, na rozdíl od obvyklé relační databáze, rozumí logice temporálních dat a nad těmito daty je definován speciální dotazovací jazyk jako např. TSQL2.

Obvyklými časovými hledisky v temporálních databázích jsou čas platnosti a transakční čas. Všechny zde uvedené časy budou popsány v následujících podkapitolách.

2.1.2 Uživatelský čas

V relační databázi nalezneme tzv. uživatelsky definovaný čas (viz User-defined Time v [5]). Jde o čas, který může být uživatelem definován jako atribut tabulky. S takovýmto časem se pracuje stejně jako s jinými datovými typy a není nad ním definována žádná speciální konstrukce dotazovacího jazyka. Setkáváme se s ním v atributech jako datum narození či datum splatnosti. Uživatelský čas nalezneme jak v relačních, tak i v temporálních databázích.

2.1.3 Čas platnosti a transakční čas

Další dvě časová hlediska, která již nejsou v běžných relačních databázích podporována, jsou čas platnosti a transakční čas.

Čas platnosti

Čas platnosti (angl. Valid Time) je ve sborníku [9] definován autory R. T. Snodgrassem a C. S. Jensenem takto:

> *Čas platnosti daného faktu je doba, po kterou platí, že fakt je pravdivý v modelované realitě. [9, překlad]*

Jinými slovy se jedná o atribut v tabulce, který říká, ve kterém časovém intervalu řádek tabulky platí. Takovýto čas jistě předpokládáme v temporální databázi. Jeho hodnoty zpravidla zadává sám uživatel databáze.

Transakční čas

Definice transakčního času ve sborníku [9] zní takto (autoři opět R. T. Snodgrass a C. S. Jensen):

> *Transakční čas daného faktu je čas, kdy je fakt přítomen v databázi. [9, překlad]*

Jde o atribut, který zaznamenává, kdy byl který řádek tabulky vložen, změněn či smazán. Je to obecně interval v čase. Nemůže odkazovat do budoucnosti ani nemůžou být jeho dříve zadané hodnoty měněny. Hodnoty tohoto času jsou zpravidla zadávány samotnou databází a uživatel tyto záznamy využívá nejčastěji pouze pro kontrolu či zálohování.

2.2 Vlastnosti temporálních dat

Temporální data jsou data, která jsou zasazená do jistého okamžiku či časového intervalu a mají význam právě jen v něm. Tato data jsou uchovávána v temporálních databázích. Obyčejné netemporální databáze nechápou logiku uchovávaných dat v časovém kontextu. Data pro ně v databázi buď jsou, nebo ne.

Postupem času byl odborníky přijat názor, že by temporální databáze měly podporovat kromě uživatelsky definovaného času ještě další dvě časová hlediska, a to čas platnosti a transakční čas (viz podkapitola 2.1). Objevily se i diskuze o dalších hlediscích jako čas rozhodnutí nebo čas události, ovšem bylo zjištěno, že je lze spolehlivě nahradit časem platnosti [9].

Temporální charakter dat přináší do databáze řadu problémů jako:

- nutnost řešení integritních omezení odlišným způsobem než v relačních databázích;
- implementace další logiky v databázi;
- zvýšené požadavky temporálních databází na paměť a další hardwarové prostředky.

I přes výše uvedené problémy existují fungující implementace temporálních databází, z nichž některé budou představeny v podkapitole 2.4.

Další vlastnost temporálních dat souvisí s transakčním časem. Z textu podkapitoly 2.1.3 plyne, že tabulky, které podporují transakční čas, uchovávají i záznamy, které byly logicky smazány, tzn., byl pouze definován konec intervalu času platnosti. Záznamy v takových tabulkách tedy není možné bez použití jiného explicitního přístupu fyzicky smazat, a databáze tak může mít neúnosné požadavky na paměťové prostředky.

V temporálním jazyce TSQL2 je tento problém řešen principem tzv. vacuumingu (viz podkapitola 2.3.3).

2.3 Jazyk TSQL2

V této podkapitole bude popsán jazyk TSQL2 (zkratka z *Temporal Structured Query Language*). Jde o temporální dotazovací jazyk, který byl navržen komisí předních výzkumníku v oblasti temporálních databází v čele s Richardem T. Snodgrassem. Vznikl v roce 1994.

Některé vlastnosti jazyka TSQL2 měly být začleněny do standardu SQL3 pod označením SQL/Temporal [10], tento projekt byl však v roce 2001 zastaven. Nicméně některé společnosti jako Oracle, IBM či Teradata převzaly tuto myšlenku a podporují koncepty a konstrukce SQL/Temporal ve svých databázových systémech. Více o implementacích TSQL2 dále v podkapitole 2.4.

2.3.1 SQL versus TSQL2

TSQL2 je temporální rozšíření dotazovacího jazyka SQL, jehož cílem je splňovat dopřednou kompatibilitu vzhledem k SQL. Důvodů proč toto rozšíření vzniklo je několik, daly by se však sdružit pod jeden, a to je slabá podpora práce s časem v SQL.

Ve specifikaci jazyka TSQL2 [6], bylo popsáno několik cílů, kterých se jeho autoři snažili zavedením TSQL2 dosáhnout. Zde jsou uvedeny ty, které řeší z pohledu autorů nedostatky SQL:

- Datový typ `TIMESTAMP` by neměl být omezen v rozsahu ani v přesnosti.
- Podpora uživatelsky definovaného času by měla zahrnovat datový typ `PERIOD`.
- Podpora lokalizace, tzn., podpora i jiných kalendářů než pouze gregoriánský a různých formátů zápisu časových údajů.
- Zjednodušení dotazů nad tabulkami podporujícími temporální data (autoři specifikace TSQL2 zjistili, že některé dotazy nad temporálními daty jsou v SQL/Temporal (viz dále v této podkapitole) až třikrát kratší než dotazy v SQL [9, TSQL2])
- Podpora nových časových hledisek, které bude moci uživatel volitelně využívat.
- Agregační funkce definované v SQL by měly mít své temporální analogie.

Kromě těchto cílů byly při vývoji TSQL2 stanoveny i jiné (viz [9, TSQL2, Design Goal for TSQL2]), pro tuto práci jsou však již méně podstatné.

2.3.2 Typy tabulek v TSQL2

Podle toho, které časové hledisko tabulky definované pomocí TSQL2 podporují, můžeme tyto tabulky rozdělit na tabulky snímkové, tabulky s časy platnosti, transakční tabulky a tabulky bitemporální.

Snímková tabulka

Tuto tabulku si můžeme představit jako obyčejnou tabulku definovanou v SQL. Neobsahuje žádné časové hledisko kromě uživatelského času a z časového pohledu jsou data uložená v takovéto tabulce platná v kterýkoli časový okamžik.

Tabulka s časem platnosti

Pokud tabulka obsahuje hledisko času platnosti, pak ji nazveme tabulkou s časem platnosti. V TSQL2 jsou dva typy tabulek s časem platnosti [6]:

- stavové – záznamy v této tabulce jsou platné v časových intervalech;
- událostní – záznamy jsou platné v časových okamžicích (viz *time instants* v [6]).

Každá n-tice má v tabulce s časem platnosti svou časovou značku (viz timestamp v [6]) ať už se jedná o stavovou či událostní tabulku. Tabulka může mít specifikován rozsah, přesnost a také možnost časové neurčitosti, kdy je známo, že n-tice v nějaký časový okamžik či interval byla nebo bude platná, ale není známo ve který.

Tabulka s transakčním časem

Tabulky, v nichž je zaznamenáván transakční čas, budeme ve zbytku práce nazývat tabulkami transakčními nebo tabulkami s transakčním časem. Časová značka v řádku (n-tici) tabulky značí, že daný řádek byl přítomen v databázi v daný okamžik nebo časový interval.

Rozdíl mezi tabulkami s časem platnosti a transakčním časem je v tom, že můžeme vznést dotaz, kdy byl daný řádek platný v reálném světě nad tabulkou s časem platnosti, ovšem tentýž dotaz nelze vznést nad tabulkou s transakčním časem. Podobně se můžeme dotázat, kdy byl v databázi přítomen daný řádek tabulky s transakčním časem, ale nikoliv tabulky s časem platnosti.

Bitemporální tabulka

Bitemporální tabulka zaznamenává jak čas platnosti, tak i transakční čas. Můžeme definovat dva druhy bitemporálních tabulek:

- bitemporální stavová;
- bitemporální událostní.

podle toho jaký typ času platnosti je v tabulce zaznamenáván.

Největším přínosem jazyka TSQL2 je čas platnosti ve stavových tabulkách, proto je na něj ve zbytku práce kladen největší důraz a příklady, které zde budou uvedeny, využívaji pro demonstraci právě těchto tabulek.

2.3.3 Nové datové typy a syntaktické konstrukce

TSQL2 definuje několik nových datových typů a syntaktických konstrukcí pro podporu práce s temporálními daty. My zde uvedeme pouze ty, které jsou důležité pro tuto práci. Pro přesný popis syntaxe odkážeme čtenáře na specifikaci jazyka [6].

Datový typ surrogate

Byl zaveden typ *surrogate*. Jedná se o unikátní identifikátor, který může být porovnán na rovnost, ale jeho skutečná hodnota je pro uživatele skrytá.

Jeho užitečnou vlastností je, že je možné snadno identifikovat různé záznamy (řádky), které jsou spolu v relaci, ale mají časově proměnlivý charakter. Nejedná se o nahrazení klíčů v databázi, ale naopak o vyjádření vztahu dvou různých záznamů v tabulce.

Datové typy pro uchování časových údajů

V TSQL2 jsou využity všechny časové typy jako v SQL. Je přidán nový datový typ *period* pro definici času. S jeho pomocí je možné specifikovat časový interval s přesně určeným začátkem a koncem. Dále je možné nastavit rozsah možných hodnot a přesnost (př. přesnost na dny).

Nové syntaktické prvky

TSQL2 zavedlo nové konstrukce při definici tabulek. Příkazy `CREATE` a `ALTER` byly upraveny tak, aby tabulky mohly podporovat nová časová hlediska. Stejně tak musela být do příkazů manipulujících s řádky tabulky přidána funkčnost, aby mohly být operace s řádky prováděny v časovém kontextu. Nejvíce nových prvků zavádí TSQL2 do databázových dotazů, při jejichž návrhu byl kladen důraz na co možná největší jednoduchost.

Více je možné se o syntaxi jazyka dozvědět v [6].

Vacuuming

V příkazu `CREATE` definice tabulek je, kromě výše uvedených prvků, přidán také prvek umožňující podporu principu tzv. vacuumingu. Tento princip byl přidán kvůli charakteristickému chování temporálních databází, které by jinak neměly možnost fyzicky odstraňovat data z tabulek s podporou transakčního času, viz podkapitola 2.2. Pomocí klauzule *VACUUM* je možné nastavit dobu, od které mohou byt neplatná data z databáze fyzicky smazána.

2.3.4 Shrnutí

TSQL2 je koncept, který byl navržen s ohledem na řadu faktorů souvisejících s temporálním charakterem dat. Implementací tohoto konceptu by vznikl velmi silný nástroj pro práci s temporálními daty. V současnosti však neexistuje implementace, která by využila všech prvků tohoto jazyka. V následující podkapitole uvedeme několik současných databází podporujících práci s temporálními daty.

2.4 Současné temporální databáze

V této podkapitole popíšeme některé databázové systémy, které implementují prostředky pro práci s temporálními tabulkami. Všechny zde uvedené databáze využívají více či méně principů jazyka TSQL2 stručně popsaného v předchozí podkapitole.

Velmi dobře zpracovaný přehled stavu implementací databázových systému s podporou temporálních dat k roku 1995 je možné nalézt v článku [1]. Rovněž v práci [8] lze nalézt obdobný přehled, jaký zde uvádíme my.

2.4.1 ChronoLog

ChronoLog je deduktivní temporální databází, která funguje jako vrstva nad komerčním databázovým systémem Oracle. Byl napsán v jazyce Prolog. Podporuje všechna časová hlediska, která byla popsána výše.

Dotazy lze provádět jak ve formě logických klauzulí (jak jsou známy v jazyce Prolog), tak i pomocí tzv. ChronoSQL – jazyka na bázi SQL.

Jazyk a databáze byly vytvořeny v rámci disertační práce Michaela Böhlena na univerzitě ETH v Curychu.

2.4.2 TimeDB

TimeDB je front-end nadstavbou nad relační databází Oracle napsanou v jazyce Java. Jedná se o překladač z jazyka ATSQL2 do jazyka SQL-92. Jazyk ATSQL2 je kombinací tří přístupů:

- TSQL2;
- ChronoLog [2];
- Bitemporal ChornoSQL – bitemporální dotazovací jazyk [7].

ATSQL2 překládá dotaz v tomto jazyce (dotazy jsou podobné těm v TSQL2) na sérii dotazů SQL. Původně byla tato nadstavba napsána v jazyce Prolog, později byla přepsána do jazyka Java ve formě uzavřeného kódu. Implementace TimeDB vznikla jako disertační práce Andrease Steinera [7]. Na jejím vývoji se podíleli R. T. Snodgrass, C. S. Jensen a M. Böhlen – lidé, kteří se zasadili o výzkum temporálních databází. Poslední verze má označení 2.2 a vyšla v roce 2005.

2.4.3 Temporální podpora v Oracle Database

V *Oracle Database* verze 10g i 11g lze nalézt podporu temporálních tabulek. V jisté formě jsou podporována obě časová hlediska, tzn. jak čas platnosti, tak i transakční čas. Tato funkčnost však není taková, jakou bychom očekávali od temporální databáze – práce s temporálními daty není dostatečně automatizována a dotazy se tvoří složitě.

Flashback dotazy

Pomocí *flashback* dotazů lze v Oracle Database docílit podpory transakčního času. Primárním úkolem flashback podpory je zálohování dat, tzn. jejich ochrana proti uživatelské nebo aplikační chybě, popř. zjednodušení aplikací pracujících s historickými daty.

Od verze 10g může uživatel či administrátor databáze použít flashback dotazu ke kontrole všech verzí (tzn. všech modifikací) řádků v intervalu vymezeném dvěma časovými údaji. Podporu FLASHBACK je třeba nejprve povolit, není totiž implicitně povolena. Nejde však o transakční čas, jak jsme ho definovali v podkapitole 2.1, protože nedovoluje dotazy, kterými by bylo možné spojit dvě tabulky s využitím časového hlediska.

Uvedeme příklad dotazu flashback SELECT s klauzulí AS OF. Mějme tabulku uživatelů, u které je povolená FLASHBACK podpora a chceme zjistit email uživatele Novák ke dni 8. ledna 2011:

```
SELECT email FROM users
  AS OF TIMESTAMP
  TO_TIMESTAMP('2011-01-08␣00:00:00', 'YYYY-MM-DD␣HH:MI:SS')
  WHERE user_name = 'Novák';
```

Workspace Manager

Kromě výše uvedených flashback dotazů je v Oracle možno využít nástroj *Workspace Manager*. S jeho pomocí je možné tabulkám přidat časová hlediska užívaná v temporálních databázích. Lze definovat tabulky jak s jedním časovým hlediskem, tak i tabulky bitemporální.

Transakční čas je podporován u tzv. „version-enabled“ tabulek a je možné podobně jako u flasback dotazů prohlížet historické záznamy, popř. vracet verze tabulky k určitému datu. Aby tabulky s transakčním časem nespotřebovaly příliš paměťových prostředků, je využit mechanismus úložných bodů (angl. savepoints) – staré záznamy v tabulce jsou ukládány pouze do časového okamžiku specifikovaného tímto úložným bodem.

Podpora času platnosti je opět umožněna u version-enabled tabulek. Práce s tímto časem i s časem transakčním je v Oracle Database poměrně komplikovaná. Příklady použití lze nalézt v [13] a [8].

Dále je pomocí něj definován datový typ period, lze provádět dotazy, které berou v úvahu časové hledisko, je řešena referenční integrita a další vlastnosti navrhované v SQL/Temporal.

2.4.4 Teradata Temporal

Společnost Teradata nabízí ke své nejnovější databázi verze 13 volitelnou součást zvanou *Teradata Temporal*. Tato databáze podporuje několik principů užitých při návrhu TSQL2:

- časové hledisko integritních omezení a dotazů;
- dopředná kompatibilita;
- datový typ period;
- podpora času platnosti a transakčního času.

Teradata se snaží o podporu rozšíření SQL/Temporal ve formě jaká byla navrhována organizací ISO předtím než byl tento projekt zastaven.

2.4.5 Další temporální databáze

Do výše uvedeného přehledu se nevešly některé komerčně či na akademické půdě vyvíjené databáze, které může čtenář hledat pod těmito názvy (stručný přehled jejich vlastností lze nalézt v [1]):

- ARCADIA
- Calenda
- HDBMS
- TempCASE
- TempIS
- TIMEMULTICAL
- T-REQUIEM
- T-squared DBMS
- VT-SQL

Podporu tabulek s časy platnosti a transakčními časy nabízí také firma IBM ve své *DB2* pro operační systém z/OS, temporální možností jsou v ní obdobné jako v Teradata Temporal.

2.4.6 TSQL2lib

Jako poslední temporální databázi uvádíme tu, na kterou tato diplomová práce do jisté míry navazuje. Jedná se o interpret jazyka TSQL2 na bázi JDBC ovladače, který vznikl jako diplomová práce Ing. Jiřího Tomka na FIT VUT v Brně [8] v roce 2009.

Interpret pracuje nad relační databází, tzn. dotazy z jazyka TSQL2 jsou překládány do SQL. Primárně byl program vyvíjen pro databázový systém Oracle, ale testován je i pro MySQL. Rozhraní JDBC umožňuje přistupovat v jazyce Java k temporální databázi prakticky stejně jako ke kterékoli relační.

Databáze podporuje čas platnosti i transakční čas tak, že definice temporálních tabulek a práce s nimi je ve shodě s tím, jak bylo definováno v jazyce TSQL2 včetně tabulek bitemporálních. Je možnost využít datového typu surrogate i period. Metadata časové logiky tabulek jsou uchovány ve dvou tabulkách – `_TEMPORAL_SPEC` pro informace o tom jaké časové údaje tabulka uchovává a `_SURROGATE` pro metadata ke sloupcům typu surrogate.

V této diplomové práci se budeme snažit implementovat interpret, který podporuje stejné temporální komponenty jako TSQL2lib a navíc je ještě rozšíří o podporu multimediálních a prostorových dat. Důležitým tématem této práce bude rozvinutí diskuze podpory temporálních integritních omezení, která byla v práci [8] započata.

Kapitola 3

Postrelační rozšíření Oracle Database

Ústředním tématem této práce je implementace temporální databáze schopné skladovat multimediální a prostorová data. Samotná temporální databáze bude pracovat nad komerčním databázovým systémem Oracle.

Oracle Database poskytuje podporu pro ukládání jak multimediálních, tak i prostorových dat. Rozšíření, která dovolují skladování tohoto druhu dat se nazývají *Oracle Multimedia* a *Oracle Spatial*.

V této kapitole popíšeme obě výše uvedená postrelační rozšíření. Zdrojem informací pro tuto kapitolu byly stránky dokumentace k produktům firmy Oracle [14] a [15].

3.1 Oracle Multimedia

Oracle Multimedia je funkce databázového systému Oracle, umožňující ukládání správu a zpřístupnění obrázků, audia, videa, dokumentů a dalších heterogenních formátů dat. V dřívějších verzích Oracle Database bylo toto rozšíření známo pod názvem *Oracle interMedia*, ve verzi 11g bylo přejmenováno na Oracle Multimedia.

3.1.1 Datové typy

Oracle Multimedia zavádí několik nových objektových typů, z nichž nejdůležitější ve smyslu této diplomové práce jsou tyto:

- ORDAudio – pro skladování audio dat;
- ORDDoc – pro skladování multimediálních dat různých typů včetně textu, obrázků, audia či videa;
- ORDImage – pro skladování obrázků;
- ORDSource – pro skladování velké škály typů multimediálních dat;
- ORDVideo – pro skladování video dat.

Kromě výše uvedených pěti typů je možné ukládat multimediální data také jako tzv. *BLOB* (zkr. z Binary Large Object) pro ukládání binárních dat nebo *CLOB* (zkr. z Character Large Object) pro ukládání dat tvořených znaky. Lze také využít speciální typ BFILE, který slouží jako reference odkazující na soubor, jehož data jsou uložena mimo databázový systém.

Dále jsou zavedeny datové typy pro podporu práce s medicínským obsahem tzv. DICOM (Digital Imaging and Communications in Medicine).

Datové typy s prefixem ORD využívají pro samotné skladování dat typ BLOB, ale uloženému obsahu přidávají sémantiku specifickou pro každý multimediální typ. Díky metodám těchto objektových typů je možné přistupovat k metadatům multimediálního obsahu.

Výše jmenované typy s předponou ORD jsou typy Oracle, kromě nich však v Oracle Database existuje podpora datových typů a operátorů standardu *SQL/MM Still image* [12]. Tento standard definuje jaké informace o obrázku mají být uloženy a metody pro jeho zpracování. Nejdůležitějším objektovým typem tohoto rozšíření standardního SQL je *SI_StillImage*, který zapouzdřuje samotný obrázek. Jsou implementovány i objektové typy určené pro porovnávání obrázků na základě různých parametrů např. podle histogramů nebo podle průměrné barvy.

Ve verzi 10g byl používán pro porovnávání obrázků objekt typu ORDImageSignature ve verzi 11g se však již neobjevuje.

3.1.2 Možnosti správy multimediálního obsahu

V této podkapitole uvedu možnosti nahrávání, stahování multimediálních dat a jejich správy. Pro tyto operace s daty je možné využít různých prostředků např.: procedurálního rozšíření *PL/SQL*, nástroje *SQL*Loader*, rozhraní *OCI* (zkr. z Oracle Callable Interface) pro přístup k Oracle Database na nízké úrovni pomocí jazyka C nebo ovladače *Oracle JDBC* pro jazyk Java. Práci s multimediálními daty pomocí ovladače JDBC popíši blíže v následující podkapitole 3.1.3.

Definice schématu tabulky

Tabulky uchovávající multimediální data jsou definovány stejně jako tabulky s jakýmkoliv jiným obsah. Př.: definice tabulky s obrázky a audiem:

```
CREATE TABLE media_table (
  mid NUMBER,
  img ORDSYS.ORDIMAGE,
  audio ORDSYS.ORDAUDIO
);
```

Nahrávání a stahování

Jak už bylo popsáno výše, nahrávání a stahování dat je umožněno více prostředky. Pokud chceme data pouze nahrát a uchovávat v databázi je možné využít utility SQL*Loader, kromě nahrávání tabulkových dat ze souborů CSV je možné také nahrávat a stahovat data ze sloupců s multimediálním obsahem. SQL*Loader vyžaduje vytvoření souboru s popisem postupu při nahrávání dat.

Další možností je využití PL/SQL. Chceme-li pomocí tohoto jazyka nahrát multimediální obsah, musíme mít soubor s tímto obsahem přítomný v operačním systému s databází (tento přístup vyžaduje oprávnění pro vytváření složek v databázi a přidělování jim práv uživatelů) nebo na něj musí odkazovat nějaké URL. Výhodou je možnost volání metod při vkládání dat – např. generování náhledů pro obrázky.

V jazyce C je pro nahrávání a správu využíváno nízko–úrovňové rozhraní OCI.

Operace nad multimediálními daty

Oracle Mutimedia podporuje většinu populárních formátů multimediálních dat ať už to jsou obrazové formáty jako JFIF (JPEG), PNGF, video formáty jako MPEG či zvukové jako 3GP nebo WAV. Navíc je možné definovat formáty vlastní.

Nad výše popsanými objektovými datovými typy je možné v PL/SQL volat jejich metody. Každý typ má své vlastní metody různé funkcionality. Lze například získat formát dat, použitý typ komprese, ale i nastavovat délku audio/video záznamu, rotovat obrázek o zadaný úhel a další operace s multimediálními daty.

Příklad získání a výpisu hodnoty atributu *fileFormat* obrázku z tabulky *media_table* definované výše v PL/SQL [14, převzato a přízpůsobeno]:

```
DECLARE
  image ORDSYS.ORDImage;
  file_format VARCHAR2(4000);
BEGIN
  SELECT img INTO image FROM media_table mt
    WHERE mt.mid = 12;
    file_format := image.getFileFormat();
    DBMS_OUTPUT.PUT_LINE('File format is ' || file_format);
  COMMIT;
END;
```

3.1.3 Podpora v jazyce Java

Oracle poskytuje rozhraní *Oracle Multimedia Java API* s implementacemi tříd objektů shodných s těmi v Oracle Database. Jsou tedy k dispozici Java třídy *OrdAudio*, *OrdDoc*, *OrdImage* i *OrdVideo*. Všechny tyto Java třídy implementují škálu metod pro získání, popř. nastavení atributů objektů stejných jako vlastní objektové typy přímo v Oracle Database. Všechny metody těchto tříd, kromě metod pro čtení a zápis, pracují na úrovni aplikace, tzn., nevyžadují připojení k databázi nebo přístup k souborům operačního systému. Ve všech případech jde o proxy třídy, pracující s databázovými objekty.

Pro čtení a zápis jsou v tomto rozhraní pro každou třídu implementovány metody čtení a zápisu do databáze (pro objekty typu BLOB nebo BFILE) a metody pro čtení a zápis souborů zadaných referencí URL.

Pro upload a download objektů do/z tabulek databáze se využívá standardního JDBC rozhraní pracujícího s Oracle objektovými typy. Upload je umožněn třídou *OraclePreparedStatement*, která metodou *setORAData* nastaví objekt pro nahrání do databáze. K downloadu je pak použito třídy *OracleResultSet*, která dovoluje pomocí metody *getORAData* získat požadovaný objekt.

Příklad načtení obrázku z tabulky *media_table* v jazyce Java s využitím ovladačů Oracle:

```
private void retrieveImg(Connection conn){
    String query = "select␣img␣from␣media_table␣where␣mid␣=␣12";
    PreparedStatement pstmt = conn.prepareStatement(query);

    OracleResultSet rset = (OracleResultSet) pstmt.executeQuery();

    OrdImage imgProxy = null;
    if (rset.next()) {

        imgProxy = (OrdImage) rset.getORAData(
                                  "img",
                                  OrdImage.getORADataFactory());
        int height = imgProxy.getHeight();
        int width = imgProxy.getWidth();
        System.out.println("Height:␣" + height);
        System.out.println("Width:␣" + width);
        imgProxy.getDataInFile("img.gif");
    }
}
```

3.2 Oracle Spatial

Druhým rozšířením, které popíši v této podkapitole je Oracle Spatial. Tato funkce umožňuje ukládání prostorových dat. Oracle Spatial je pod tímto názvem prodáván od verze 8 jako volitelná součást Oracle Database. V současnosti (verze 11g) je možné ho zakoupit v edici Enterprise. Jádro Oracle Spatial nese název *Oracle Locator* a je nabízeno ve všech edicích Oracle Database 11g, jedná se o stejný produkt s omezenou funkčností. Oracle Spatial je složen z následujících komponent:

- schématu MDSYS, které předepisuje ukládání, syntaxi a sémantiku podporovaných geometrických datových typů;
- indexovací mechanizmus pro prostorová data;
- sadu funkcí a operátorů pro provádění dotazů nad prostorovými daty, prostorové spojování tabulek a další prostorové analytické operace;
- administrativní utility.

3.2.1 Datové typy

Oracle uchovává prostorová data společně s obyčejnými relačními daty v tabulkách, jak je známe z relačních databází, využívá tzv. objektově–relační model dat. Nejdůležitějším typem atributů zavedeným pro uchovávání prostorového obsahu ve smyslu této diplomové práce je `MDSYS.SDO_GEOMETRY`.

Datový typ `MDSYS.SDO_GEOMETRY` je tvořen pěti atributy:

- SDO_GTYPE – tento atribut říká, jakého typu jsou uložená data, může to být např.: bod, úsečka, polygon a další;
- SDO_SRID – udává v jakém souřadném systému je geometrický útvar definován (souřadné systémy jsou uchovány v tabulce `MDSYS.CS_SRS`), pokud je hodnota tohoto atributu null, pak jde o lokální souřadnice tedy souřadnice, které nejsou vztaženy k žádné geografické reprezentaci Země;
- SDO_POINT – pokud je hodnota atributů SDO_ELEM_INFO a SDO_ORDINATES nastavena na null bere se tento atribut jako definice geometrie bodu;
- SDO_ELEM_INFO – tento atribut říká, jakým způsobem mají být interpretovány hodnoty atributu SDO_ORDINATES;
- SDO_ORDINATES – je definován jako pole s proměnnou délkou typu NUMBER, ve kterém je uložena informace o hranicích objektu. Hodnoty v tomto poli jsou řazeny dle dimenze. Př.: {X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4, X1, Y1, Z1}.

K tomu, aby nad záznamy tabulky mohly být vykonávány některé prostorově orientované funkce (např. funkce `SDO_AREA` pro výpočet plochy), musí být ještě navíc o sloupci typu `SDO_GEOMETRY` veden záznam v prostorové meta-tabulce `MDSYS.USER_SDO_GEOM_METADATA`. Pro efektivní práci s prostorovými daty je také nutné vytvořit nad vloženými záznamy prostorový index.

Dále Oracle Spatial implementuje některé operátory specifikované ve standardu *SQL/MM Part 3: Spatial* [11]. Jde např. metody objektu `SDO_GEOMETRY` `GET_WKT` a `GET_WKB`, které mají svou obdobu u objektu `ST_GEOMETRY` ve zmíněném standardu jako `ST_AsText` resp. `ST_AsBinary`. Tyto metody slouží k textové resp. binární reprezentaci gemetrického objektu ve formátu *WKT* (Well-Known Text) resp. *WKB* (Well-Known Binary) – oba jsou definovány v [11]. Objekt `SDO_GEOMETRY` lze také vytvořit pomocí konstruktoru, který má jako parametr textovou reprezentaci geometrie ve formátu WKT nebo WKB.

3.2.2 Podpora v jazyce Java

Pro jazyk Java je k dispozici rozhraní *Oracle Spatial Java API*. Jedná se o sadu rozhraní a tříd, které podporují operace a datové typy definované v Oracle Spatial. Je alternativou k práci s prostorovými daty v jazyce PL/SQL – až na výjimky jsou v tomto rozhraní definovány stejné třídy jako v PL/SQL.

Je zde implementována třída JGeometry, která mapuje typ `MDSYS.SDO_GEOMETRY` a zprostředkovává základní funkce pro přístup ke geometrickým datům uloženým v databázi.

Dále jsou tímto rozhraním definovány balíčky pro podporu síťového a topologického datového modelu (viz [15]).

Kapitola 4

Motivace

Tato kapitola slouží ke shrnutí témat, kterým se autor chce dále věnovat v této diplomové práci. Jejím úkolem je seznámit čtenáře s východisky návrhu a následnou implementací. Kapitola zároveň může sloužit jako upřesnění očekávaného přínosu k výzkumu v oblasti. Přínos je pak na závěr shrnut v kapitole 9.

V podkapitole 4.1 jsou popsána témata, která nejsou v současných temporálních databázích dořešena a mají význam pro tuto diplomovou práci. Následující podkapitola 4.2 navazuje očekávaným přínosem autora k řešení popsaných témat.

4.1 Některá nedořešená témata

Oblast temporálních databází je zkoumána už od 80. let 20. století. V roce 1995 byl navržen jazyk TSQL2, který však nenašel širšího uplatnění v praxi. Bylo zvažováno vytvoření temporálního rozšíření jazyka SQL – tzv. SQL/Temporal, které však nakonec nebylo zavedeno a tento projekt byl zastaven v roce 2003.

V kapitole 2 je popsán stav výzkumu v oblasti temporálních databází. Zde uvádíme témata, která nejsou podle našeho názoru v této oblasti úplně vyřešena:

- absence nových implementací temporálních databází;
- implementace konstrukcí navržených v TSQL2;
- správa multimediálních dat v temporální databázi;
- správa prostorových dat v temporální databázi;
- kontrola referenční integrity v temporální databázi nad databází realční.

4.2 Očekávaný přínos

V rámci této práce se pokusíme vyřešit či navrhnout řešení témat popsaných v předchozí podkapitole. V následujících kapitolách je představen návrh a implementace interpretu podmnožiny jazyka TSQL2, který umožňuje ukládání multimediálních a prostorových dat.

Dále je nad rámec zadání diskutována podpora integritních omezení v temporální databázi. Podpora kontroly entitního integritního omezení je ve zmiňovaném interpretu TSQL2 implementována.

Kapitola 5

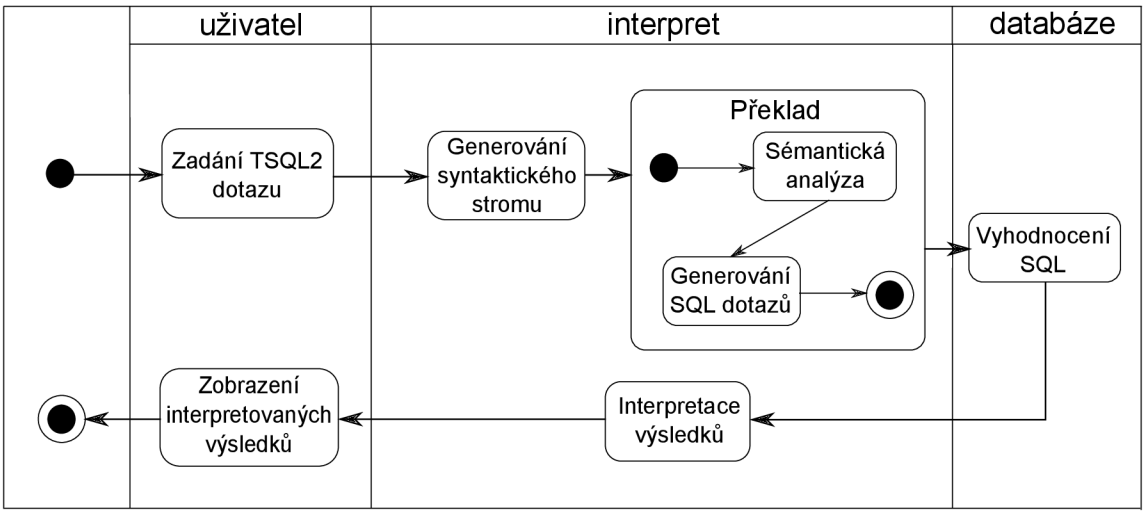
Návrh

V této kapitole uvedeme návrh interpretu. V první podkapitole čtenáři objasníme architekturu a požadavky na interpret z obecného pohledu. V další podkapitole 5.2 budeme pokračovat návrhem uchovávání temporálních metadat, která vychází z práce Ing. Jiřího Tomka [8]. Návrh kontroly referenční integrity, která však není součástí interpretu, je zpracován v podkapitole 6.2.

5.1 Architektura

Hlavním výstupem diplomové práce, je funkční interpret jazyka TSQL2 s podporou multimediálních a prostorových dat. Systém bude postaven nad relační databází, která podporuje práci právě s multimediálními a prostorovými daty.

Dotaz v TSQL2 tedy bude muset být navrhovaným interpretem překládán na SQL dotazy. Praxe v oblasti překladačů ukázala, že je vhodné při syntaktické analýze budovat syntaktický strom, díky čemuž je následná analýza překládaného řetězce výrazně ulehčena. V tomto návrhu budeme předpokládat použití syntaktického stromu. Zjednodušenou architekturu celého navrhovaného sytému (tj. interpretu spolupracujícího s relační databází) je možné vidět na obrázku 5.1.



Obrázek 5.1: UML diagram aktivit architektury navrhovaného interpretu

5.2 Temporální metadata

Vzhledem k tomu, že temporální tabulky budou v námi navrhovaném interpretu uloženy v relační databázi, je nutné udržovat i informaci o temporálním charakteru tabulek – společně s tabulkami s daty uživatele v databázi. Toho může být dosaženo dvěma způsoby.

5.2.1 Změna informačního schématu

První možností je změna informačního schématu v relační databázi. V metadatech o všech tabulkách v systému se přidají nové sloupce vypovídající o tom, o jakou tabulku se z temporálního hlediska jedná (např. platnostní-stavová, transakční, bitemporální-událostní, atd.). Kromě těchto hodnot je ještě u platnostních tabulek nutné uchovávat informaci o měřítku a přesnosti času platnosti. To lze vyřešit např. zavedením tabulky s metadaty, ve které jsou zaneseny bližší informace o čase platnosti.

S tímto přístupem jsou spojeny dva problémy. Prvním je, že změna informačního schématu se projeví ve všech schématech databáze a druhým problémem je, že uživatel při instalaci temporálního rozšíření může mít omezená práva pro tuto operaci. Oba problémy řeší druhý způsob, viz další podkapitola.

5.2.2 Zavedení meta tabulek

Možným řešením problémů spojených se způsobem navrženým v předchozí podkapitole je definování nových tabulek s metadaty o temporalitě tabulek uživatelských. Takto se vyřeší problém s ovlivněním všech schémat v databázovém systému i omezená uživatelská práva pro úpravu informačního schématu. Ve zbytku této podkapitoly popíšeme právě tento přístup a bude použit i v navrhovaném interpretu.

Pro definici základních temporálních vlastností uživatelských tabulek vystačí dvě metatabulky – tabulka `_TEMPORAL_SPEC` pro uchování informace o temporálním charakteru uživatelských tabulek a `_SURROGATE` pro uchování následujících hodnot atributů typu surrogate.

`_TEMPORAL_SPEC` informuje o typu tabulky, měřítku a přesnosti času platnosti a dále je v ní uložena informace o čase, který určuje dobu, po kterou jsou v tabulce ukládány záznamy s ukončeným transakčním časem. K tabulce `_TEMPORAL_SPEC` není možné přistupovat s využitím interpretu, dotazy na její obsah jsou filtrovány a vráceny s chybou.

Kromě informací o temporalitě tabulky je nutné uchovávat v uživatelských temporálních tabulkách také informaci o čase (ať už čase platnosti nebo transakčním čase) záznamu. Při fyzickém vytváření tabulky v databázi bude tato tabulka tvořena kromě uživatelských sloupců také dvěma novými sloupci s informací o časovém hledisku daného záznamu. U tabulky s časem platnosti budou zavedeny sloupce `_VTS` a `_VTE` pro uchování začátku a konce tohoto času. Budou mít celočíselný datový typ a preciozitu danou specifikací v `_TEMPORAL_SPEC`. Pro transakční čas budou zavedeny sloupce `_TTS` a `_TTE`, které budou mít přesnost danou implementací relační databáze a datový typ `TIMESTAMP`. Příklad bitemporální tabulky, jak by byla uložena v databázi, viz tabulka 5.1.

ID	PANOVNIK	_VTS	_VTE	_TTS	_TTE
1	Karel IV.	1002003	2003620	11.7.2010 12:32:24	FOREVER
2	Václav IV.	2003621	3673230	11.7.2010 12:36:43	FOREVER

Tabulka 5.1: Příklad bitemporální tabulky v relační databázi, která byla vytvořena pomocí interpretu tsql2lib.

5.2.3 Shrnutí

Návrh uchovávání temporálních metadat podle Ing. Tomka je podle našeho názoru správný a v této práci se jím budeme řídit. Jestliže má temporální databáze nad databází relační podporovat kontrolu referenční integrity, pak takový systém musí uchovávat i metadata spojená s právě zmíněnou kontrolou. Návrh kontroly referenční integrity v temporální databázi je rozebrán v kapitole 6.

5.3 Překlad a interpretace konstrukcí TSQL2

TSQL2 je navržen jako plně dopředně kompatibilní rozšíření jazyka SQL. Jeho výhodou je, jak již bylo zmíněno v podkapitole 2.3, zjednodušení práce s časovými údaji v databázi. Tato výhoda však s sebou nese nutnost zavedení nových syntaktických konstrukcí do databázových příkazů. V této podkapitole, která je jednou ze zásadních z hlediska přínosu práce, popíšeme některé nově zavedené konstrukce a bude navrženo řešení jejich následné interpretace v relační databázi.

5.3.1 Nové datové typy

Než bude popsána interpretace příkazů, bude dobré se seznámit s novými datovými typy, které TSQL2 přináší. Těmito datovými typy a jejich interpretací v databázi jsou všechny databázové příkazy ovlivněny.

Surrogate

Atribut uchovávající expresi vztahu mezi dvěma záznamy ve stejné tabulce je v TSQL2 definován klíčovým slovem `SURROGATE`. Hodnoty ukládány v tomto atributu musí být jednoznačně odlišitelné pro záznamy, které spolu v relaci nejsou. Z hlediska uložení hodnot tohoto atributu v databázi je možné použít celočíselný datový typ. Dále je nutné uchovávat informaci o již použitých hodnotách. Vhodným přístupem je použít přidělování nových hodnot jako inkrement hodnoty naposledy použité. Potom už stačí uchovávat informaci pouze o hodnotě, která má být použita jako následující. Pro skladování této informace slouží výše popsaná tabulka `_SURROGATE`.

Period

Atribut s datovým typem period je možné definovat v TSQL2 explicitně. Větší přínos má však jeho použití ve stavových a transakčních tabulkách, kde je čas záznamu uchováván ve sloupci tohoto typu. Period je časový interval s pevně určenými hranicemi – tedy dvěma časovými okamžiky.

Tento datový typ je velkou výhodou temporálních databází, ale při jejich implementaci nad databázemi relačními přináší nejvíce problémů. Zásadní skutečností je, že záznam, který obsahuje sloupec s tímto datovým typem, nereprezentuje jeden řádek tabulky, ale nekonečně mnoho řádků – každý pro jeden časový okamžik v intervalu, který period vymezuje. Tato skutečnost přináší problémy jak při manipulaci s řádky, tak i v dotazech nad temporálními tabulkami.

Pro uložení tohoto typu atributu není v databázovém systému Oracle ekvivalentní datový typ. Řešením je jeho uložení ve dvou sloupcích, které určují jeho dvě hranice intervalu.

Otázkou však zůstává, jaký datový typ pro tyto dva sloupce zvolit. Transakčnímu času vyhovuje v Oracle podporovaný typ `TIMESTAMP`, který je pro transakční záznamy definován v dostatečném rozmezí (od roku 4713 před Kr. do 9999 po Kr.). Pro čas platnosti je však toto rozmezí nedostatečné – pro většinu aplikací temporální databáze je pro uložení tohoto času výhodnější zvolit celočíselný datový typ s dostatečnou kardinalitou, který udává počet časových jednotek uplynulých od pevně stanoveného okamžiku (pozn.: tyto časové jednotky jsou nazývány *chronony* a jsou dále nedělitelné).

Instant

Druhým datovým typem po typu period, který slouží pro ukládání časových údajů, je typ instant. Tento datový typ nezavádí TSQL2, ale naopak většina relačních databází jej podporuje. Temporální databáze však má na tento typ větší nároky z hlediska použitelného časového rozmezí a také z hlediska přesnosti, kterou je schopen vyjádřit.

Instant lze definovat v tabulce explicitně, ale stejně jako u typu period jeho síla především v definici času platnosti událostních tabulek. Pro reprezentaci typu instant v databázi lze analogicky s typem period zvolit celočíselný typ. Pro uložení hodnot atributu instant nám postačí jeden sloupec.

5.3.2 Data definition language

TSQL2 přináší pět nových typů tabulek (viz podkapitola 2.3) a tři nové výše uvedené datové typy. Tyto nové vlastnosti se projeví mimo jiné také při interpretaci definice a rušení tabulek.

Příkaz CREATE

Překlad TSQL2 příkazu CREATE se skládá ze tří až čtyř kroků:

1. Vytvoření dotazu pro definici tabulky – při definici tabulky je potřeba ve výsledném dotazu, který bude tabulku vytvářet, přeložit klíčová slova nově definovaných datových typů na datové typy, kterými jsou v relační databázi reprezentovány.

2. Vložení záznamů o sloupcích typu surrogate – je nutné vložit záznamy o sloupcích s tímto typem do tabulky `_SURROGATE`, pokud nově definovaná tabulka takové atributy má.

3. Vložení záznamů do meta-tabulky `_TEMPORAL_SPEC` – tento krok je nutný pří definici tabulek s podporou nově zavedených časových hledisek.

4. Vacuuming – posledním krokem, který však nemusí být proveden vždy, je zajištění procesu mazání z tabulek podporujících transakční čas.

K tomu, aby mohly být provedeny poslední dva kroky, je nutné znát ještě informace o typu tabulky, popř. o čase, kdy má docházet k mazání z transakční tabulky. Tyto informace v TSQL2 poskytují dva výrazy, které jsou v příkazu uváděny za definicí obsahu tabulky [6]. Jde o tzv. *temporal definition*, kde je uvedeno, jaký typ času tabulka podporuje a tzv. *vacuum definition*, která může být součástí příkazů definujících transakční tabulky [6]. Vacuum definition udává, po jaké době mají být záznamy, jejichž transakční čas byl ukončen, z tabulky odstraněny.

Samostatnou kapitolou je definice primárního klíče v temporální tabulce. Tomuto tématu se blíže věnuje následující kapitola 6.

Příkaz DROP

V případě příkazu `DROP` je situace poměrně jednoduchá. Při rušení tabulky je nutné smazat také záznamy týkající se této tabulky z meta-tabulek `_SURROGATE` a `_TEMPORAL_SPEC`. Popřípadě zrušit jiné databázové prostředky použité pro práci s tabulkou.

5.3.3 Data manipulation language

Také u příkazů manipulujících s daty se v TSQL2 objevují nové syntaktické konstrukce a logika jejích provádění je svázána s časem. U příkazů `INSERT`, `UPDATE` a `DELETE` se objevuje nová klauzule uvozená klíčovým slovem `VALID`. Účelem této klauzule je vymezit časový interval, v rámci kterého se s daty v tabulce manipuluje.

Nejmarkantnější odlišnost mezi jednotlivými příkazy je v počtu příkazů jazyka SQL, na které jsou přeloženy. Zatímco u `INSERT` je tento příkaz přeložen u všech typů tabulek na jeden příkaz v jazyce SQL, u typu `UPDATE` je v bitemporální stavové tabulce nutné provést až sedm příkazů jazyka SQL.

Problém příkazů DML (konkrétně `UPDATE` a `DELETE`), který je potřeba řešit, je slévání záznamů u stavových tabulek.

Záznamy, které mají všechny atributy shodné a jejichž čas platnosti na sebe navazuje, je možné spojit do jednoho, protože z pohledu temporální databáze není potřeba, aby byly odděleny. U relačních dat lze shodu dvou záznamů ověřit databázovým dotazem.

Z logického pohledu je možné slévání vynechat, ale z hlediska optimalizace výkonu databáze je slévání nezbytně nutné. Bez řešení slévání záznamů ve stavové tabulce by data v této tabulce byla velmi redundantní. Neustálým prováděním příkazů `UPDATE` a `INSERT` by se totiž záznamy neúnosně parcializovaly.

Velkou komplikaci do procesu slévání vnášejí postrelační data. Porovnání dvou položek multimediálního či prostorového atributu nelze provést způsobem stejným jako u dat relačních. Pro slévání záznamů v tabulkách s postrelačním obsahem by bylo nutné zvolit jiný způsob než porovnání databázovým dotazem.

Následuje popis interpretace jednotlivých příkazů.

Klauzule VALID

Než popíšeme jednotlivé příkazy, bude dobré si představit klauzuli `VALID`. Tato klauzule rozšiřuje syntaxi a mění sémantiku příkazů manipulujících se záznamy v tabulkách s časem platnosti.

Tělo této klauzule tvoří buď výrazy v TSQL2 určující časový okamžik (výrazy s klíčovým slovem `DATE`, `TIME` a `TIMESTAMP`), nebo výraz uvozený slovem `PERIOD` vymezující časový interval pro manipulaci. Za klíčovým slovem, které říká, o jaký typ časového údaje půjde, následuje jeden (okamžik) nebo dva (interval) časové údaje. Časový údaj může být specifikován buď pevným datem, nebo konstrukcí s klíčovým slovem `NOW` a přičtením či odečtením čísla, které udává relativní posun od času zadání příkazu. U tohoto způsobu zadání času modifikace je možné za časovým údajem uvést měřítko (klíčová slova `HOUR`, `DAY`,...), aby bylo zřejmé v jakých jednotkách se má od současného času odečítat či přičítat.

INSERT

Z hlediska překladu je tento příkaz nejjednodušším typem manipulujícím s daty. Po určení časového okamžiku v událostní tabulce nebo intervalu v tabulce stavové (určí se pomocí klauzule `VALID`) je konstrukce SQL dotazu přímočará.

Překlad je vždy tvořen jedním SQL příkazem `INSERT`. Nově vkládané hodnoty musí být doplněny o sloupce s časem platnosti, popř. transakčním časem – podle toho, o jaký typ tabulky se jedná. Hodnota počátku transakčního času záznamu musí být dána časem vložení do databáze. Konec transakčního času u nově vloženého záznamu je roven konstantě `FOREVER` – tedy není ukončen (viz [6]).

U atributů typu surrogate je potřeba dbát na to, aby nebyla vkládána hodnota přímo. U těchto atributů může být nová hodnota vkládána pouze s použitím klíčového slova `NEW`.

UPDATE

Příkaz `UPDATE` je co do složitosti překladu mezi příkazy manipulujícími s daty nejnáročnější. Počet příkazů jazyka SQL, na které je příkaz překládán, se liší podle typu času, který daná tabulka podporuje. Náročnost překladu tohoto příkazu je vnesena tabulkami s časovými hledisky ukládanými pomocí datového typu period – tzn. transakční, stavové a bitemporální. S datovým typem period se totiž významně mění chování dotazu `UPDATE`.

Zatímco v relační databázi po provedení tohoto příkazu je vždy zachován počet záznamů v tabulce, v databázi temporální toto neplatí. Na záznamy v ní je totiž třeba nahlížet v časovém kontextu – platnost či logická přítomnost záznamu je dána intervalem. Záznam může v jenom okamžiku platit, ale v jiném ne.

Pokud je příkaz `UPDATE` prováděn ve **stavové** tabulce, pak je prováděn v určitém časovém intervalu. Se záznamy, které do tohoto intervalu nespadají, nebude nijak manipulováno. Hodnoty záznamů, které leží uvnitř intervalu jsou měněny (pozn.: vyhovují-li výrazům v klauzuli `WHERE`). V těchto dvou případech je situace jednoduchá a není potřeba nijak měnit počet záznamů v tabulce. Problémy však způsobují záznamy, které do časového intervalu spadají jen částečně. Tyto záznamy je potřeba rozdělit na dva, resp. na tři (podle toho zda záznam časově přesahuje interval pro úpravu na jednom nebo na obou koncích) a upravovat pouze ty záznamy, které spadají do intervalu pro úpravu.

V tabulkách podporujících transakční časové hledisko je potřeba každý nově upravovaný záznam před úpravou duplikovat a duplikovanému ukončit transakční čas. Transakční čas všech záznamů musí zůstat konzistentní, aby bylo možné se v každém okamžiku vrátit k potřebnému stavu databáze. V bitemporální stavové tabulce jsou možné dvě řešení tohoto problému. Ukončit transakční čas je možné:

1. buď celým záznamům, tedy i těm částem, které nezasahují do intervalu úpravy;
2. nebo jen těm částem záznamů, které jsou uvnitř intervalu úpravy.

První způsob vyžaduje vytvoření nových záznamů s transakčním časem, který začíná v době úpravy. Druhý způsob počítá s novými záznamy, jejichž transakční čas začíná v okamžiku transakčního začátku původního záznamu.

Z hlediska zálohování není důležité, který typ ukončování záznamu bude zvolen. Jedná se spíše o filozofickou otázku, zda je lepší zachovávat co nejdéle nepřerušené intervaly času platnosti nebo transakčního času.

Celou situaci nejlépe ilustruje příklad. Mějme bitemporální tabulku – viz tabulka 5.2. Záznam v této tabulce bude upraven 1. 6. 2011 v rozmezí času platnosti od roku 1350 do roku 1360. Výsledek úpravy prvním způsobem můžete vidět v tabulce 5.3. Výsledek úpravy druhým způsobem je uveden v tabulce 5.4.

ID	PANOVNIK	Čas platnosti	Transakční čas
1	Karel IV.	1346 - 1378	17. 5. 2011 - FOREVER

Tabulka 5.2: Bitemporální tabulka před úpravou

ID	PANOVNIK	Čas platnosti	Transakční čas
1	Karel IV.	1346 - 1378	17. 5. 2011 - 1. 6. 2011
1	Karel IV.	1346 - 1350	1. 6. 2011 - FOREVER
1	null	1350 - 1360	1. 6. 2011 - FOREVER
1	Karel IV.	1360 - 1378	1. 6. 2011 - FOREVER

Tabulka 5.3: Bitemporální tabulka po úpravě prvním způsobem

ID	PANOVNIK	Čas platnosti	Transakční čas
1	Karel IV.	1350 - 1360	17. 5. 2011 - 1. 6. 2011
1	Karel IV.	1346 - 1350	17. 5. 2011 - FOREVER
1	null	1350 - 1360	1. 6. 2011 - FOREVER
1	Karel IV.	1360 - 1378	17. 5. 2011 - FOREVER

Tabulka 5.4: Bitemporální tabulka po úpravě druhým způsobem

DELETE

Příkaz `DELETE` se z hlediska překladu v mnohém podobá příkazu `UPDATE`. Opět je potřeba řešit mazání u tabulek s časovými hledisky definovanými typem period analogicky jako u příkazu `UPDATE` s tím rozdílem, že není potřeba brát v úvahu úpravu starých hodnot.

U tabulek s transakčním časem nejsou záznamy fyzicky z tabulky vymazány, ale pouze je ukončen jejich transakční čas. U bitemporálních stavových tabulek nastává při mazání opět analogická situace jako v případě `UPDATE`, kdy je možné řešit ukončení transakčního času částečně zasahujících záznamů do intervalu mazání dvěma obdobnými způsoby.

5.3.4 Databázový dotaz

Nejvíce nových syntaktických prvků vnáší jazyk TSQL2 do databázových dotazů (`SELECT`), proto je překlad dotazů nejsložitější. Temporální výrazy se objevují v seznamu pro výběr sloupců, ve `FROM` klauzuli pro výběr tabulek, ale nejvíce nových výrazů TSQL2 vnáší do klauzule `WHERE` pro omezení výběru řádků.

Výběr sloupců

V seznamu výběru sloupců počítáme v návrhu se třemi novými funkcemi:

1. Zadáním funkce `VALID` se jménem tabulky jako parametrem při definici výběru sloupce je možné vybrat do výsledku dotazu sloupec s časem platnosti tabulky, pokud takový čas tabulka podporuje. Sloupec s časem platnosti je implicitně vybírán při výběru všech sloupců (výraz `*`).

2. Funkce `TRANSACTION` se jménem tabulky jako parametrem zahrne do výsledku dotazu sloupec s transakčním časem záznamu. Sloupec s transakčním časem není implicitně při výběru všech sloupců (výraz `*`) vybírán. Pokud tato funkce není přítomna v seznamu sloupců pro výběr, nejsou do výsledku dotazu zahrnuty transakčně ukončené řádky.

3. Měřítko sloupců s časy platnosti nebo s transakčními časy lze měnit použitím funkce `CAST`.

Při interpretaci nových funkcí ve výběru sloupců v SQL bude v přeloženém dotazu nutné v případě transakční a stavové tabulky zahrnout při výběru dva sloupce. Uživatel bude k těmto dvěma sloupcům přistupovat jako k jednomu.

Slévání záznamů

Jak již bylo zmíněno při popisu interpretace datového typu period, řádek tabulky se sloupcem period (např. stavové tabulky), nereprezentuje pouze jeden záznam, ale obecně nekonečně mnoho záznamů – jeden pro každý časový okamžik, vymezený intervalem typu period. Vlivem příkazu `UPDATE` může dojít k rozdělení záznamu v relační databázi na dva, přitom však na sebe tyto záznamy časově stále navazují. Pokud z takové tabulky vybereme sloupec, ve kterém se tyto záznamy neliší, dostaneme dva záznamy se shodnými hodnotami a navazujícími časy platnosti.

Ve `FROM` výběru tabulek přibyl tzv. výběr sléváním řádků tabulky (angl. coalesced rows). Syntakticky se zapisuje jako jméno platnostní tabulky se seznamem jmen sloupců v závorce. Takto deklarovaná tabulka má „slité“ řádky podle specifikovaných hodnot sloupců – tzn. řádky z původní tabulky, které se neliší ve specifikovaných sloupcích a jejichž čas platnosti na sebe navazuje, jsou reprezentovány jedním řádkem a odpovídajícím časovým intervalem. Příklad dotazu se zmíněnou konstrukcí ve `FROM` klauzuli:

```
SELECT * FROM panovnici(id, panovnik);
```

Pokud předchozí dotaz provedeme nad tabulkou 5.5, pak výsledkem bude tabulka 5.6.

ID	PANOVNIK	TITUL	Čas platnosti
1	Karel IV.	král	1346 - 1355
1	Karel IV.	císař	1355 - 1378

Tabulka 5.5: Tabulka panovnici k příkladu slévání záznamů

Interpretaci slévání záznamů se práce blíže věnuje v podkapitole 7.3.3.

ID	PANOVNIK	Čas platnosti
1	Karel IV.	1346 - 1378

Tabulka 5.6: Výsledek dotazu nad tabulkou panovnici

Omezení výsledků dotazu

U klauzule `WHERE`, omezující počet výsledků dotazu, v návrhu počítáme se čtyřmi relačními operátory TSQL2, jejichž operandy jsou typu period:

- `PRECEDES` – výraz s tímto operátorem je pravdivý, jestliže interval času prvního operandu předchází intervalu druhého operandu;
- `OVERLAPS` – výraz je pravdivý, jestliže čas prvního operandu začíná dříve než čas druhého operandu a končí taktéž dříve než konec druhého operandu;
- `CONSTAINS` – výraz je pravdivý, jestliže časový interval prvního operandu obsahuje interval druhého operandu;
- `MEETS` – výraz je pravdivý, jestliže čas prvního operandu navazuje na čas druhého operandu.

Jako operand je možné zadat `VALID` funkci nebo výraz `PERIOD`, oba výrazy jsou popsány výše.

5.4 Podpora multimediálních a prostorových dat

Multimediální i prostorová rozšíření relační databáze přinášejí mnoho nových syntaktických konstrukcí, ať už jde o nové datové typy, funkce, metody objektů, nebo konstruktory. Popíšeme zde chování obou typů dat v navrhovaném systému z hlediska sémantické analýzy a navrhneme řešení problému, který je specifický pro multimediální data, a to jsou proxy proměnné.

5.4.1 Sémantická analýza

Z popisu rozšíření relační databáze v kapitole 3 plyne, že multimediální a prostorová data jsou ukládána stejně jako data relační, tedy ve sloupcích tabulek a je s nimi zacházeno rovněž jako s relačními daty. TSQL2 kromě dvou nových typů (period a surrogate) žádné odlišné datové typy než SQL nemá.

Z hlediska syntaktické analýzy je potřeba v syntaktickém analyzátoru definovat pravidla pro přijímání řetězců, které mají v příkazech konstrukce postrelačního charakteru. Jde především o funkce, které pracují s multimediálními a prostorovými daty.

Sémantika dat typická pro postrelační data kontrolována být nemusí. Pokud bude v interpretu TSQL2 realizována sémantická analýza částí řetězců s multimediálními a prostorovými konstrukcemi, pak bude tato analýza v implementovaném systému prováděna dvakrát – jednou při překladu z TSQL2 do SQL a podruhé v relační databázi. Když si uvědomíme, že části dotazů, které budou obsahovat multimediální a prostorová data, nebude interpret nijak překládat, zjistíme, že jedna ze sémantických analýz těchto částí dotazů je zbytečná.

Lepší řešení tohoto problému je umožnit v interpretu vynechání kontroly u těch částí dotazů, které jsou tvořeny konstrukcemi stejnými jako v SQL a tuto kontrolu ponechat pouze na relační databázi.

K řešení navrženému v předchozím odstavci uvedeme příklad dotazu, který byl převzat z dokumentace k produktu Oracle Spatial [15]:

```
SELECT i.highway
FROM geod_cities c, geod_interstates i
WHERE c.city = 'Tampa'
 AND sdo_within_distance (
      i.geom, c.location,
      'distance=15␣unit=mile') = 'TRUE';
```

Postrelační část tohoto databázového dotazu v klauzuli `WHERE`, tedy

```
sdo_within_distance (
   i.geom, c.location,
   'distance=15␣unit=mile')
```

se nemusí nijak překládat ani kontrolovat a může se rovnou začlenit do klauzule `WHERE` výsledného dotazu v SQL. Podobný postup je uplatňován při práci s daty relačními, které není třeba překládat.

5.4.2 Multimediální proxy objekty

Druhý problém souvisí pouze s multimediálními daty. Práce s těmito daty je specifická v tom, že objem dat uložených v jednom záznamu je zpravidla mnohem větší, než je tomu u dat relačních nebo prostorových.

Nástroje, které s multimediálními daty pracují, se snaží počet přenosů do/z databáze omezit na minimum – hlavně kvůli velkému objemu přenesených dat. Oracle poskytuje pro práci s multimediálními daty ovladač s tzv. *proxy třídami*. Při práci s nimi je nejprve nutné získat referenci na databázový objekt pomocí metody *getORAData* třídy *OracleResultSet*.

Z tohoto důvodu bude muset třída implementující rozhraní *ResultSet* ve výsledném interpretu implementovat nejen toto rozhraní, ale také rozhraní *OracleResultSet*, které je poskytováno JDBC ovladačem Oracle. Použitím ovladače JDBC jak při realizaci rozhraní klient-interpret, tak i rozhraní interpret-databáze, se práce s multimediálními daty (a nejen s nimi) značně zjednoduší.

Současně je ovšem potřeba zdůraznit, že multimediální data vnášejí do systému nutnost implementace tzv. předpřipraveného příkazu (v JDBC třídy *PreparedStatement* nebo *CallableStatement*). Tím se však značně stíží překlad některých typů dotazů. Více se práce tomuto tématu věnuje v podkapitole 7.4.2.

Kapitola 6

Temporální integritní omezení

V této kapitole představíme problémy, které souvisí s podporou kontroly integritních omezení v temporální databázi. Jmenovitě se budeme zabývat kontrolou entitní a referenční integrity. Z pohledu času platnosti je toto téma rozebráno v publikaci [3]. My se v následujícím textu budeme tímto tématem zabývat méně formálně a budeme na něj nahlížet z pohledu praktické implementace v relační databázi.

6.1 Entitní integrita

V relační databázi jsou primárními klíči unikátní hodnoty. Jde tedy o sloupec jehož hodnoty jednoznačně identifikují řádky tabulky. Většinou jsou číselného nebo řetězcového typu.

V temporální databázi při kontrole zdali záznamy neporušují entitní integritu, nestačí brát v úvahu pouze sloupce primárního klíče. Dva záznamy se shodnými hodnotami primárního klíče se v temporální tabulce mohou vyskytovat, jestliže se jejich intervaly času (ať už času platnosti, nebo transakčního čas) neprotínají.

Řešení tohoto problému při interpretaci událostní tabulky v relační databázi je triviální – stačí pouze rozšířit primární klíč o sloupec s časem platnosti. Tímto způsobem nemůže nastat situace, že by dva záznamy měly shodný primární klíč ve stejném časovém okamžiku.

V temporální databázi problémy působí záznamy s časy, které jsou reprezentovány datovým typem period (viz předchozí kapitola 5). Implementace kontroly entitní integrity u takových tabulek v relační databázi má dvě úskalí:

1. je nutné zajistit, aby dva záznamy neporušovaly entitní integritu relační databáze;
2. je nutná kontrola temporální entitní integrity.

První problém je možné jednoduše vyřešit tím, že primární klíč specifikovaný při definici tabulky rozšíříme o sloupec či sloupce času platnosti či transakčního času. Navíc je ale potřeba dát pozor, aby entitní integrita relační databáze nebyla porušena při úpravě či mazání záznamů. Příkaz těchto operací v TSQL2 je překládán na sekvenci příkazů jazyka SQL a při špatně zvoleném pořadí jejich vykonávání může dojít k přechodnému porušení entitní integrity, která by však byla obnovena v případě provedení sekvence celé. Relační databáze ale provedení příkazu porušujícího integritu nepovolí.

Při zajišťování temporální entitní integrity je potřeba dbát na to, aby v žádném časovém okamžiku nebyly platné dva záznamy se stejným primárním klíčem. Dodržení entitní integrity v temporální tabulce lze ověřit databázovým dotazem (viz dále). Tento dotaz je nutné spouštět po provedení příkazů `INSERT` a `UPDATE`, které potenciálně mohou způsobit porušení

integrity. Nejprve je však nutné vypnout automatické provádění změn v databázi (tzv. *autocommit*), aby mohla být v případě porušení integrity provedena operace `ROLLBACK`, která obnoví původní stav databáze. Jestliže byla operace `ROLLBACK` provedena, pak je nutné dat uživateli o této vyjímečné situaci vědět.

Zde uvádíme příklad dotazu SQL nad stavovou tabulkou `ZAMESTNANCI`, který ověří, zda záznamy splňují entitní integritní omezení. Tabulka má primární klíč daný sloupcem `ID` a struktura této tabulky se řídí strukturou navrženou v kapitole 5. Dotaz:

```
SELECT COUNT(id)
  FROM ZAMESTNANCI z1, ZAMESTNANCI z2
  WHERE
      z1.id = z2.id
    AND
      z1.ROWID <> z2.ROWID
    AND (
        z1."_VTS" <= z2."_VTS"
        AND z1."_VTE" > z2."_VTS"
    );
```

Jestliže je výsledek dotazu roven nule, pak tabulka entitní omezení splňuje, protože žádné dva záznamy nesplňují podmínky porušení specifikované v dotaze. V opačném případě by podle navrhovaného řešení došlo k operaci `ROLLBACK`.

V dotazu byl použit pseudo atribut `ROWID` typický pro databázi Oracle, který jednoznačně identifikuje řádek v tabulce. Pokud bychom výraz porovnávající tyto atributy vynechali, pak by byl výsledek dotazu nad tabulkou neporušující entitní integritu roven počtu záznamů tabulky.

Pokud bychom chtěli zkonstruovat dotaz bez použití `ROWID`, pak bychom mohli využít obyčejné relační entitní integrity. Dotaz by pak mohl vypadat následovně:

```
SELECT COUNT(id)
  FROM ZAMESTNANCI z1, ZAMESTNANCI z2
  WHERE
      z1.id = z2.id
    AND (
       ( z1."_VTS" < z2."_VTS"
         AND z1."_VTE" > z2."_VTS"
       ) OR (
         z1."_VTS" = z2."_VTS"
         AND z1."_VTE" <> z2."_VTE"
       )
    );
```

V prvním dotaze jsme použili atributu `ROWID`, abychom z výsledku odebrali shodné záznamy. V druhém dotaze činíme totéž náhradou neostré nerovnosti za ostrou (viz levý operand operace `OR`) za cenu vyřazení záznamů se shodným počátečním časem, ale různým koncovým, které do výsledku však započítat chceme. Tento problém je řešen pravým operandem operace `OR`. Poslední variantou porušení entitní integrity, která může nastat, je výskyt dvou odlišných záznamů se shodnými časy platnosti. Tomu ovšem zabraňuje kontrola entitní integrity, prováděná relační databází. Je potřeba si uvědomit, že primární klíč tabulky je v relační databázi tvořen trojicí (`ID`, `_VTS`, `_VTE`) a případnou duplicitu tohoto klíče kontroluje samotná relační databáze.

6.2 Referenční integrita

V práci Ing. Tomka [8] byla diskutována podpora kontroly referenční integrity. My v této podkapitole tuto diskuzi rozvedeme a navrhneme několik možných řešení jak z hlediska obecného návrhu kontroly referenční integrity, tak i z hlediska použitých prostředků pro její realizaci.

Kontrola referenční integrity v relačních databázích ověřuje existenci cizích klíčů v odkazovaných tabulkách. Schází-li v odkazované tabulce záznam s obrazem cizího klíče, pak není referenční integrita dodržena. K porušení referenční integrity může dojít dvěma způsoby:

1. při mazání z odkazované tabulky je smazán záznam, na který bylo odkazováno z odkazující tabulky;
2. při vkládání do odkazující tabulky je vložen záznam, jehož cizí klíč nemá obraz v tabulce odkazované.

Řešení v relační databázi je triviální:

1. mazání záznamů buď není povoleno, nebo jsou společně se záznamy z odkazované tabulky mazány i záznamy tabulky z odkazující;
2. záznamy, jejichž cizí klíče nemají svůj obraz v odkazované tabulce, není možné do odkazující tabulky vložit.

Téměř všechny relační databázové systémy mají kontrolu referenční integrity implementovánu.

V temporální databázi jsou největší problémy, stejně jako u entitní integrity, způsobeny záznamy s časovými údaji uloženými pomocí typu period. U tabulek podporujících temporální referenční integritu nelze připustit, aby v jednom časovém okamžiku záznam odkazoval na záznam jiný, který v tomtéž okamžiku neplatí (u času platnosti) nebo není přítomen v databázi (u transakčního času). Během manipulace s daty prostřednictvím příkazů DDL nastává řada situací, kdy dochází k porušení referenční integrity. Příklady těchto situací uvedu v následující podkapitole a v podkapitole další navrhnu jejich řešení.

6.2.1 Příklady porušení referenční integrity

Tato podkapitola je rozčleněna na řadu situací, kdy dochází k porušení referenční integrity v temporální databázi. Každá situace je označena číslem a v následující podkapitole může čtenář nalézt řešení těchto situací pod stejným číslem.

Situace 1. Nekorektní interval času platnosti

Uvažujme situaci, kdy stavová tabulka odkazuje do stavové tabulky. Uživatel se pokusí vložit záznam do odkazující tabulky, jehož interval vymezený časem platnosti nespadá do intervalu záznamu, na který odkazuje. Odkazující záznam je tedy platný i v čase, kdy není platný odkazovaný záznam.

Situace 2. Rozdělení záznamu v odkazované stavové tabulce

Záznam ze stavové tabulky odkazuje zase na záznam ze stavové tabulky. Záznam v odkazované tabulce je rozdělen na 2, přičemž se rozdělí interval času platnosti na nové dva.

Situace 3. Spojení záznamů ve stavové tabulce

Dva záznamy ze stavové tabulky odkazují na dva záznamy (každý na jeden) z jiné stavové tabulky přičemž intervaly času platnosti na sebe v obou případech navazují. Odkazované záznamy mají stejný obsah. Dojde ke sloučení záznamů v odkazované tabulce z důvodu úspory místa v databázi.

Situace 4. Odkazovaná transakční tabulka

Snímková tabulka odkazuje do tabulky s transakčním časem. V tabulce s transakčním časem je ukončen transakční čas záznamu (záznam byl logicky smazán z databáze). Záznamy ze snímkové tabulky teď odkazují na záznam, který již logicky není v databázi.

Stejný problém nastává, pokud na transakční tabulku odkazuje tabulka s časem platnosti.

Situace 5. Odkazující transakční tabulka

Tabulka s transakčním časem odkazuje do snímkové tabulky. Dojde ke smazání záznamu ve snímkové tabulce. Záznamy by měly být z transakční tabulky logicky smazány, fyzicky však v databázi zůstanou, odkazují ale na neexistující záznam ve snímkové tabulce.

Tento problém se projeví i tehdy, když transakční tabulka odkazuje do tabulky stavové.

Situace 6. Odkazující tabulka má méně časových hledisek

Snímková tabulka odkazuje do tabulky s časem platnosti. Skončí čas platnosti záznamu v odkazované tabulce a vytvoří se nový záznam s jiným časem. Záznam ve snímkové tabulce, který před rozdělením odkazoval pouze na jeden řádek, teď musí odkazovat na dva. Kardinalita vztahu těchto dvou tabulek se změní z 1:N na M:N.

Tato situace nastává u všech typů referencí, kdy odkazující tabulka má méně časových hledisek než tabulka odkazovaná. Další podobné situace tedy nebudeme uvádět.

6.2.2 Řešení situací porušujících referenční integritu

V této podkapitole jsou uvedeny návrhy na řešení problémů referenční integrity, které byly popsány výše.

Řešení 1. Nekorektní interval času platnosti

1. Z temporální sémantiky plyne, že nejlepším řešením je zakázání takové operace. Logicky totiž není korektní, když interval času platnosti odkazujícího záznamu přesahuje hranice intervalu odkazovaného. Odkazující záznam totiž částí intervalu odkazuje na nespecifikovanou hodnotu.

2. Jiným řešením by mohlo být oříznutí intervalu tak, aby zachoval referenční integritu. Bez toho, aby se dala tato skutečnost na vědomí uživateli, to ovšem není příliš vhodné.

Řešení 2. Rozdělení záznamu v odkazované stavové tabulce

1. Tento problém je možné řešit tak, že dojde k rozdělení záznamů i v odkazující tabulce, ve shodě s rozdělením v odkazované.

2. Druhým možným řešením je zavést další tabulku (dále ji budeme nazývat *tabulka referencí*), která bude uchovávat informaci o referenčním vztahu tak, že záznam v odkazující tabulce bude odkazovat na oba dva nové záznamy.

Oba přístupy se zdají být vhodné, v obou případech dochází k nárůstu objemu dat v databázi. Přičemž, pokud uvážíme, že na odkazující tabulku může být také odkazováno, zjistíme, že tato operace se v tomto případě bude šířit v databázi na další tabulky. První navržené řešení tedy bude muset využít rekurze ke kontrole integrity u tabulek odkazujících na odkazující tabulku.

Druhý způsob řešení nepotřebuje využívat rekurzivního rozdělování záznamů. Využití tabulky referencí odráží tu skutečnost, že řádek v temporální tabulce nereprezentuje pouze jeden záznam, ale nekonečně mnoho záznamů – pro každý časový okamžik vymezený časovým intervalem jeden.

Řešení 3. Spojení záznamů v odkazované stavové tabulce

1. První řešení by mohlo být takové, že se zkontrolují záznamy v odkazující tabulce, a pokud by si odpovídaly, došlo by ke spojení. V opačném případě by bylo zakázáno spojení v odkazované tabulce. Zde by stejně jako u problému 2. došlo k rekurzivnímu šíření spojování záznamů na odkazující tabulky.

2. Jiným řešením by bylo využití tabulky referencí uvedené u problému 2. Záznamy v odkazované tabulce by mohly být spojeny bez komplikací a u záznamů z odkazující tabulky by došlo k úpravě reference v tabulce referencí.

Druhé řešení se zdá být vhodnější, protože umožňuje spojení záznamů i tehdy, když to u prvního řešení nelze.

Řešení 4. Odkazovaná transakční tabulka

Uvědomme si, že tabulky uchovávající transakční čas mohou tohoto časového hlediska mimo jiné využívat i k zálohování. Je tedy otázkou, jestli je nutné uchovávat i záznamy z tabulky, která na transakční tabulku odkazuje.

1. Pokud připustíme, že je to nutné, pak jediným možným řešením je umožnit odkazovat na transakční tabulku pouze tabulkou s transakčním časem. Tak je to řešeno i u transakční podpory v Oracle Database.

2. Pokud odkazující záznamy není nutné uchovávat, pak je problém vyřešen a záznamy ze snímkové tabulky se fyzicky smažou z databáze.

Řešení 5. Odkazující transakční tabulka

Zde stojíme před opačným typem porušením referenční integrity než u přechozího problému. Navíc řešíme problém neplatného odkazu z transakční tabulky u záznamu, který byl logicky smazán.

Zde je tedy jediným možným řešením umožnit odkazování z transakční tabulky pouze do tabulky transakční.

Řešení 6. Odkazující tabulka má méně časových hledisek

1. Už z popisu kardinality vztahu plyne, že řešením tohoto problému je zavedení tabulky referencí. Odkazující záznam, který odkazoval původně na jeden, bude pomocí tabulky referencí odkazovat na záznamy dva.
2. Druhým možným řešením by bylo umožnění referenčních vztahů pouze mezi tabulkami se stejnými časovými hledisky.

První řešení se zdá být obecnější a navíc umožňuje odkazování mezi tabulkami s různými časovými hledisky (až na tabulky transakční, u nichž, jak se zdá, bude nutné omezit kontrolu referenční integrity, viz řešení situace 4. a 5.).

6.2.3 Shrnutí řešení problémů

Nejvhodnější navrhovaná řešení by mohla být shrnuta do těchto bodů:

- zakázání vkládání záznamů, jejichž interval času platnosti přesahuje interval času platnosti odkazovaných záznamů;
- použití tabulek referencí;
- povolení odkazování z transakční tabulky pouze do tabulky s transakčním časem;
- povolení odkazování do tabulky s transakčním časem pouze z tabulky transakční.

Jak je možné usoudit z této podkapitoly, referenční integrita v temporální databázi není zdaleka tak triviální, jak je tomu u databáze pouze s relačními daty. Je nutno brát v úvahu časová hlediska, která mají své logické vazby, jež relační databáze neřeší.

6.2.4 Prostředky kontroly

V této podkapitole popíši, jakými prostředky by měla být zajištěna kontrola referenční integrity v temporální databázi. Navrhnu dva přístupy, a to buď kontrolovat integritu přímo interpretem při vkládání, rušení či změně záznamu, anebo použít prostředků databázového systému jako jsou triggery nebo uložené procedury.

Kontrola při interpretaci

U tohoto typu kontroly bude probíhat ověřování, zda záznam neporušuje pravidla referenční integrity, ještě před tím než bude do databáze vůbec vkládán. Tuto kontrolu provede samotný interpret.

Pokud bychom využili pouze tohoto přístupu, museli bychom definovat další tabulku s metadaty, která by uchovávala informaci o referenčních vztazích. Záznam v této tabulce by obsahoval jména tabulek a sloupců mezi nimiž referenční vztah je.

Kontrola v databázi

Jestliže při implementaci systému využiji tento přístup, pak interpret vytvoří při definici tabulek trigger nebo uloženou proceduru a tyto prostředky se budou starat o kontrolu referenční integrity nad právě definovanou tabulkou.

Vkládání, úprava či mazání z databáze by pak probíhalo tak, že by při každém z těchto příkazů byl spuštěn trigger (automaticky databázovým systémem) nebo uložená procedura (interpretem či databázovým triggerem), která by ověřila zachování referenční integrity.

Shrnutí

Dá se očekávat, že některé podmínky referenční integrity budou snáze řešitelné interpretem a jiné naopak databází, při implementaci je možné využít kombinace obou přístupů.

Triggery lze velmi elegantně vyřešit problém 1. z předchozí podkapitoly. Při vkládání záznamu se pouze ověří, zda spadá interval odkazujícího záznamu do intervalu odkazovaného. Pokud tomu tak nebude, pak trigger vyvolá databázovou výjimku.

Naopak pro zajišťování složitějších operací s tabulkou referencí bude možná výhodnější použít kontrolu pomocí samotného interpretu.

Kapitola 7

Implementace

Hlavním výstupem této diplomové práce je funkční interpret jazyka TSQL2. Tato kapitola popisuje proces implementace tohoto interpretu a zabývá se problémy, se kterými jsme se při implementaci setkali.

Programovacím jazykem implementace je zadáním vyžadovaný jazyk Java. Pro realizaci programové části této diplomové práce skýtá použití tohoto jazyka několik zde uvedených výhod:

- Program vyvíjený v tomto jazyce se díky nezávislosti virtuálního stroje na použitém operačním systému automaticky stává multiplatformním.
- Existuje ovladač pro komunikaci s databází Oracle.
- Existující implementace jiných knihoven a nástrojů využitelných při vývoji interpretu (např. *JavaCC* – viz dále) obecně jakéhokoliv jazyka.

Výsledná implementace interpretu je rozdělena do tří balíčků *oratelib.SQLGenerator*, *oratelib.tsql2parser* a *oratelib*. Těmto balíčkům odpovídají podkapitoly pořadě 7.1, 7.3 a 7.4.

7.1 Syntaktická analýza

Z hlediska posloupnosti operací, které je potřeba v interpretu provést při zpracování příkazu v jazyce TSQL2, je prvním krokem syntaktická analýza tohoto příkazu. Je zapotřebí nejen zkontrolovat syntaktickou správnost příkazu, ale navíc je, z pohledu dalších kroků překladu, velmi cenným výstupem syntaktického analyzátoru tzv. syntaktický strom.

Programování syntaktického analyzátoru jazyka, jehož gramatiku známe, je v informatice běžně řešený problém. Syntaktický analyzátor je ve své podstatě zásobníkovým automatem a algoritmus převodu bezkontextové gramatiky, přesněji LL(k) gramatiky, kterou je generován také jazyk TSQL2, je znám.

Protože jde o algoritmicky řešitelný problém, existují nástroje schopné vygenerovat ze zadané gramatiky již hotový syntaktický analyzátor. Takovým nástrojem je i *JavaCC*, který byl zvolen pro implementaci syntaktického analyzátoru navrženého interpretu.

JavaCC je konzolová aplikace, která vygeneruje z gramatiky zadané v souboru, analyzátor v jazyce Java. S využitím další aplikace *JJTree* je možné před spuštěním JavaCC ke gramatice přidat konstrukce jazyka Java tak, aby výsledný analyzátor při analýze generoval snadno použitelný syntaktický strom. Oba programy jsou open source a jsou distribuovány v jednom balíčku pod BSD licencí.

7.2 Reprezentace temporálních komponent v Oracle

Podle návrhu uvedeného v kapitole 5 byly pro reprezentaci komponent interpretu v databázi použity tyto datové typy a konstanty:

- čas platnosti je uložen ve sloupcích typu `NUMBER(20)`, čísla uložena v těchto sloupcích označují počet milisekund uběhlých od půlnoci 1. ledna 1970;
- transakční čas je uložen ve dvou sloupcích datového typu `TIMESTAMP`;
- sloupce typu surrogate jsou uloženy pomocí datového typu `NUMBER`;
- konstanta `FOREVER`, u času platnosti shora ohraničující možný časový interval pro definici platnosti záznamu, je nastavena na hodnotu `MAX_VALUE` specifikovanou ve standardní třídě Long jazyka Java;
- konstanta `BEGINING`, ohraničující tento interval zdola, je nastavena na hodnotu `MIN_VALUE` třídy Long;
- konstanta `FOREVER`, u transakčního času definující transakčně neukončený záznam, má hodnotu nastavenu na 1. ledna 5000 v reprezentaci datového typu `TIMESTAMP`.

Třída *Calendar* umožňuje ukládání časových údajů v měřítku milisekund od stanovené epochy. Jak již bylo zmíněno výše, my jsme jako epochou zvolili počátek roku 1970. Třída *Calendar* používá pro ukládání počtu milisekund datového typu *Long*, který má maximální hodnotu definovánu jako $2^64/2$. Této hodnotě odpovídá cca. 290 mil. let. My v interpretu využíváme pro operace s časem právě třídu *Calendar* a proto jsme omezili možný rozsah času platnosti na 250 mil. let před naším letopočtem až 250 mil. let našeho letopočtu.

7.3 Překlad příkazů jazyka TSQL2

S novými datovými typy a syntaktickými konstrukcemi jazyka TSQL2 byl čtenář seznámen v podkapitole 5.3, tato podkapitola popisuje implementaci nových vlastností, které TSQL2 definuje.

Překlad ze syntaktického stromu příkazu v jazyce TSQL2 na obecně sérii příkazů jazyka SQL je implementován v balíčku *SQLGenerator*. K samotnému překladači je možné přistoupit pomocí třídy *StatementTranslator*, ve které dochází k delegaci překladu na třídy specializované na jednotlivé příkazy. Stěžejní metodou třídy *StatementTranslator* je metoda *translate*, která jako parametr přijímá syntaktický strom příkazu a jako výsledek vrací objekt třídy *SQLStatements*, což je zapouzdření příkazů jazyka SQL, které jsou v další fázi předány relační databázi.

Všechny třídy překladu příkazů jazyka TSQL2 dědí od třídy *CommonStatementTranslator*, ve které jsou definovány metody pro extrakci informací o čase z TSQL2 výrazu `PERIOD` a výrazů pro definici časového okamžiku.

Pro snadnější práci při překladu příkazu nad již existujícími tabulkami byla vytvořená třída *TableInfo*. Objekt této třídy načítá data o podpoře nových časových hledisek a názvy sloupců tabulky. Dále schraňuje informace o sloupcích typu surrogate a další data potřebná při překladu jednotlivých příkazů jazyka TSQL2. Inicializace všech informací o tabulce je zapouzdřená metodou *init*, která jako parametr přebírá název tabulky, o níž mají být informace zjištěny.

Další text podkapitoly je rozdělen podle typů příkazů jazyka TSQL2.

7.3.1 Data definition language

O překlad příkazů definujících data se starají tři třídy:

- *CreateTranslator* – definice tabulek;
- *DropTranslator* – rušení tabulek;
- *IndexDefinitionTranslator* – definice a rušení indexu nad sloupci.

Při překladu příkazu definice tabulky v TSQL2 je nejdůležitějším příkazem SQL `CREATE TABLE`, který vytvoří tabulku v relační databázi. Pokud se jedná o temporální tabulku podporující alespoň jedno nově zavedené časové hledisko, je navíc potřeba vložit záznam o temporalitě tabulky do meta-tabulky `_TEMPORAL_SPEC`. Pakliže se v tabulce vyskytují surrogate sloupce, je nutné vložit záznam o každém sloupci do tabulky `_SURROGATE`. Posledním krokem při definici tabulky je zajištění procesu mazáni záznamů z transakčních tabulek, tomuto problému se práce věnuje v samostatné podkapitole 7.5.

Při rušení tabulky je potřeba provést reverzi kroků provedených při její definici. Výsledkem překladu příkazu pro zrušení tabulky, který je prováděn pomocí třídy *DropTranslator*, je tak opět sada příkazů SQL. Je potřeba zrušit samotnou tabulku, zrušit její záznam v tabulce `_TEMPORAL_SPEC`, zrušit záznamy o jejích surrogate sloupcích v tabulce `_SURROGATE` a také zrušit proces mazání transakčních záznamů (viz podkapitola 7.5).

Pro efektivní přístup k prostorovým datům je potřeba nad těmito daty vytvořit index. O překlad příkazu pro vytvoření, resp. zrušení indexu nad sloupci je v interpretu využívána třída *IndexDefinitionTranslator*. Při definici, resp. rušení indexu nejsou v příkazu zavedeny nové syntaktické prvky, a tak jsou tyto příkazy rovnou předávány v nezměněné podobě.

7.3.2 Data manipulation language

Příkazy manipulující s daty v databázi mají společnou vlastnost v tom, že pracují vždy jen nad jednou tabulkou. Překladač syntaktického stromu tedy zpravidla na začátku překladu vytvoří instanci třídy *TableInfo* a zavolá metodu *init* tohoto objektu.

Dále budeme popisovat implementaci jednotlivých příkazů manipulujících s daty.

INSERT

O překlad příkazu pro vložení záznamu se v implementovaném interpretu stará třída *InsertTranslator*. Překlad tohoto příkazu není z implementačního hlediska nijak složitý. Po inicializaci objektu třídy *TableInfo* je nutné přidat hodnoty sloupců definující časová hlediska. Čas platnosti je možné zjisti z klauzule `VALID`. Transakční čas tohoto nově vloženého záznamu začíná v okamžiku vložení a končí v čase definovaném pomocí TSQL2 konstanty `FOREVER`.

Jestliže tabulka neobsahuje sloupce s hodnotami typu surrogate, pak je výsledkem překladu takového příkazu pouze jeden příkaz v jazyce SQL. Při překladu je nutné brát na vědomí, že hodnoty v tabulce `_SURROGATE` by neměly být upravovány hned při překladu, protože přeložený příkaz nemusí splňovat integritní omezení a při případné nutnosti navrácení stavu databáze pomocí operace `ROLLBACK` by nedošlo k navrácení tabulky `_SURROGATE` do původního stavu.

UPDATE

Překlad příkazu úpravy záznamů obstarává třída *UpdateTranslator*. Implementace překladu tohoto příkazu již není tak přímočará jako u příkazu `INSERT`.

Velikost výsledné množiny SQL příkazů se liší podle toho, jaké časové hledisko tabulka, která je upravována, podporuje – implementace třídy *UpdateTranslator* respektuje tuto rozmanitost tím, že dělí zpracování příkazů právě podle typu tabulky – pro každý typ tabulky je implementovaná jedna metoda, která sestaví potřebnou množinu SQL příkazů.

V návrhu v podkapitole 5.3 byly zmíněny dva způsoby ukončování transakčních záznamů v bitemporálních stavových tabulkách. Interpret podporuje způsob, kdy jsou transakčně ukončeny všechny části záznamů zasahujících do intervalu úpravy.

DELETE

Interpretace překladu mazání z temporálních tabulek je zapouzdřena ve třídě *DeleteTranslator*. Tato třída je v mnohém podobná třídě *UpdateTranslator*, která byla popsána v předchozím textu. Obdobně jako u `UPDATE` je překlad rozdělen podle typu časového hlediska, které tabulka, z níž je mazáno, podporuje. Příkaz `DELETE` je co do implementace překladu jednodušší než příkaz `UPDATE`, protože není potřeba se starat o vkládání nových hodnot.

7.3.3 Databázový dotaz

Daleko nejsložitější z pohledu překladu jednotlivých příkazů podmnožiny jazyka TSQL2 je dotaz `SELECT`. Implementaci tohoto překladu zapouzdřuje třída *SelectTranslator*. V případě že je v dotazu přítomná `WHERE` klauzule, využívá *SelectTranslator* třídy *SelectWhereClauseTranslator*, která se stará o překlad temporálních relačních výrazů ve `WHERE` klauzuli. Výsledkem překladu tohoto příkazu musí být vždy pouze jeden databázový dotaz, aby bylo možné pracovat s jeho výsledky.

V další části této podkapitoly jsou vybrána některá, z pohledu implementace zajímavá, témata.

Implementace slévání řádků

V návrhu v podkapitole 5.3 byl popsán problém tzv. slévání řádků (v definici TSQL2 je použit výraz *coalescing*) v tabulce podporující čas platnosti. Zjednodušeně řečeno jde o spojení některých záznamů podle daných sloupců vzhledem k času platnosti.

Výše popsané prostředky překladu byly implementovány s důrazem na co nejmenší interakci s relační databází. Architektura interpretu navržená v podkapitole 5.1 nepočítá s prováděním příkazu jazyka SQL dříve než po samotném překladu. Tento princip však při slévání řádků nebylo možné dodržet, protože není možné vytvořit jediný dotaz nad tabulkou tak, aby byly řádky „slity“ podle požadovaných sloupců.

V interpretu je slévání implementováno tak, že je nejprve vytvořena dočasná tabulka, která nahradí odpovídající položku ve `FROM` výrazu pro výběr tabulek. Dočasná tabulka je nejprve naplněna hodnotami sloupců, které byly definovány ve výčtu položky ze seznamu `FROM` (viz [6]). Po naplnění jsou záznamy v cyklu spojovány, dokud existují záznamy, které lze „slít“.

Po ukončení práce s výsledky dotazů nad dočasnou tabulkou je nutné tabulku zrušit. V interpretu je tato operace provedena při zavírání instance třídy *ResultSet* metodou *close*. Zde je ovšem problém v tom, že uživatel nemusí vždy metodu *close* použít. Operace je

tedy v interpretu dále pojištěna rušením dočasných tabulek v metodě *finalize*, kterou volá Garbage Collector virtuálního stroje při mazání (angl. garbage collecting) objektu. Ovšem nelze se spoléhat ani na tento způsob rušení dočasných tabulek, protože k zavolání metody *finalize* nemusí vždy dojít [4].

Jako budoucí rozšíření interpretu navrhujeme zavedení nové meta tabulky, která bude schraňovat informace o aktuálně používaných dočasných tabulkách, a případně odstraňovat tabulky, které používány nejsou. Mechanismus by se mohl rozhodovat např. podle kontroly aktuálních sezení. Tento přístup by však vyžadoval dodatečné přidělení oprávnění k dotazům nad některými virtuálními tabulkami systému Oracle.

Meta informace dotazu

Po úspěšném překladu dotazu je potřeba k jeho vyhodnocení předat nejen samotný dotaz, ale také metadata, která popisují samotné výsledky databázového dotazu. Týká se to názvů a pozic sloupců, ty totiž mohou lišit v dotazu TSQL2 a dotazu nad relační databází. Je například potřeba zapouzdřit dva atributy transakčního času v dotazu SQL do jednoho atributu výsledku dotazu TSQL2, totéž se pak týká času platnosti definovaného intervalem.

O zapouzdření informací o výsledcích dotazu se stará třída *SelectMetaInfo*, která pro popis sloupců využívá instance třídy *UserColumnMetaData*.

7.4 Třídy rozhraní JDBC

Komunikace uživatele s interpretem je realizována pomocí rozhraní JDBC. Implementace všech tříd by byla co do rozsahu poměrně náročná, výsledný interpret obsahuje jen některé ze tříd tohoto rozhraní. Tato podkapitola stručně popisuje třídy, které jsou v interpretu realizovány. Podrobně se pak věnuje popisu problémů, které jsou z implementačního hlediska zajímavé.

Každá ze zde uvedených tříd zapouzdřuje svůj ekvivalent JDBC ovladače databázového systému Oracle. Úkolem těchto tříd je interpretovat komunikaci mezi uživatelem a databází.

Hlavní rozhraní, které musí implementovat každý JDBC ovladač, je *Driver*, toto rozhraní realizuje třída *OrateDriver*, pomocí které je možné vytvořit připojení zapouzdřené rozhraním *Connection*. Rozhraní mající na starost připojení implementuje třída *OrateConnection*.

7.4.1 Statement

Pomocí třídy *OrateConnection* lze získat instanci třídy *OrateStatment*, která zapouzdřuje JDBC rozhraní *Statement*. Toto rozhraní plní funkci okamžité realizace příkazů v databázovém jazyce.

Příkazy lze zadávat pomocí metody *execute*, která má jako parametr textovou podobu příkazu. Zmíněná metoda je stěžejní z hlediska interpretace komunikace, protože je v jejím těle proveden jak překlad dotazu z jazyka TSQL2 do SQL, tak i předání přeložených příkazů databázi pomocí JDBC ovladače Oracle. Při provádění příkazů, které manipulují s daty, je v této metodě využito navrženého principu, kdy dochází k návratu ke stavu databáze v případě výskytu jakékoli výjimky či při nedodržení entitní integrity v tabulce (viz kapitola 6). O kontrolu dodržení temporální entitní integrity se stará metoda *isEnitityIntegrityOk*, která pomocí dotazu tuto integritu ověří.

Kromě metody *execute* jsou pro vykonání dotazu implementovány ještě metody *executeQuery*, která vrací výsledek databázového dotazu (viz dále v podkapitole 7.4.3), a *executeUpdate*, která vrací počet upravených záznamů v tabulce.

Metoda *isEntityIntegrityOk* řeší kontrolu entitní integrity popsanou v kapitole 6. Metoda *tryCoalescing* řeší slévání záznamů v tabulkách s relačním obsahem popsané v kapitole 5.3.3.

7.4.2 PreparedStatement

Obdobně jako u třídy *OrateStatement* je možné pomocí třídy *OratePreparedStatement* komunikovat prostřednictvím TSQL2 příkazů s databází Oracle. Tato třída implementuje rozhraní JDBC *PreparedStatement*.

Rozdíl oproti *OrateStatement* je ten, že příkaz je nejprve předán s tzv. substitučními výrazy, za které je následně možné dosazovat hodnoty. Tato vlastnost musela být v interpretu implementována, protože multimediální data nedovolují načtení do databáze jiným prostředkem, než právě pomocí předpřipraveného příkazu.

Tento předpřipravený příkaz však vnáší do překladu příkazů další problém, a to nutnost rozdělovat příkazy v přeložené sadě SQL příkazů na předpřipravené a běžné. Příkaz `UPDATE` je překládán v některých případech na pět příkazů jazyka SQL. Část příkazů slouží pro úpravu hodnot a část je určena k vložení starých hodnot s pozměněným časem. Příkazy, které neupravují hodnoty, neobsahují `SET` klauzuli, a tedy ani substituční výrazy, a proto nemohou být použity jako předpřipravené. Příkazy pro úpravu naopak po přeložení substituční výrazy v klauzuli `SET` obsahují, a proto jsou při vykonávání příkazů zpracovávány jako předpřipravené.

Při provádění příkazů v metodě *execute* musí být předem známo, o jaký z výše jmenovaných typů příkazů se jedná a dodržet správné pořadí spouštění těchto příkazů. V objektu třídy *SQLStatements* bylo proto nutné zavést další mechanizmus pro ukládání sekvence provádění předpřipravených a běžných příkazů.

Omezení, popsané v předchozích dvou odstavcích má kritický vliv na implementaci předpřipraveného příkazu. Realizace této třídy v interpretu totiž zcela neodpovídá požadavkům rozhraní JDBC. Problém je u příkazů upravujících záznamy, které jsou překládány na více příkazů SQL (jedná se o `UPDATE` příkazy upravující stavové a bitemporální tabulky). U těchto příkazu totiž nejsou překládány klauzule `WHERE`, ale jsou rovnou předány jako součásti výsledných dotazů. Objeví-li se tedy v příkazu jazyka TSQL2 substituční výraz v klauzuli `WHERE`, je tento výraz ve výsledku předán všem výsledným příkazům, tedy i těm, které nejsou při vyhodnocování brány jako předpřipravené. To ale znamená, že do substitučních výrazů nebude dosazena hodnota.

Řešením by bylo vyhledání všech substitučních výrazů ve zmíněných příkazech a zavedení nové funkčnosti pro výpočet jejich pozic. Substituční výrazy nelze pouze odstranit z `WHERE` klauzulí ne-předpřipravených příkazů, ale je nutné tyto příkazy brát také jako předpřipravené. Pozice substitučních výrazů v příkazech nemodifikujících nové hodnoty pak budou sníženy o počet substitučních příkazů, které se vyskytují v klauzuli `SET`.

V současné implementaci interpretu je možné předpřipraveného příkazu využívat s příkazem `UPDATE`, pouze pokud jsou substituční výrazy obsaženy v klauzuli `SET`, tím je umožněno načítání multimediálních dat do databáze.

7.4.3 ResultSet

Výsledky databázového dotazu zapouzdřuje v interpretu třída *OrateResultSet*, která implementuje JDBC rozhraní *ResultSet*.

Tato třída obsahuje velkou sadu metod, které vracejí objekty odpovídající výsledkům databázového dotazu. Příkladem takové metody může být *getString*, která z výsledku dotazu vrátí řetězcovou reprezentaci dané položky v tabulce. K tomuto typu metod rozhraní JDBC byly přidány metody *getInstant* a *getPeriod*, které vracejí instanci třídy zapouzdřující informace o čase platnosti či transakčním čase, uloženého v daném sloupci výsledku dotazu (viz dále v podkapitole 7.4.4).

Bylo zvažováno, zda má být časové hledisko ve výsledku dotazu reprezentováno sloupcem, nebo zda čas záznamu ukládat pro čas implicitně – tedy bez použití sloupce. Nakonec bylo rozhodnuto reprezentovat časová hlediska sloupcem, protože by správně počet položek pro výběr sloupců v dotazu měl odpovídat počtu sloupců ve výsledku dotazu. Použitím funkcí `VALID` a `TRANSACTION` při výběru sloupců v databázovém dotazu, se tím správně zvýší počet sloupců ve výsledku dotazu.

V interpretu obecně neplatí, že pozice, resp. názvy sloupců odpovídají pozicím, resp. názvům sloupců v ekvivalentním objektu třídy ResultSet JDBC ovladače Oracle.

ResultSetMetaData

Výsledky databázových dotazů je potřeba správně vyhodnotit. Rozhraní JDBC, které ke správné interpretaci výsledků pomáhá, je rozhraní *ResultSetMetaData*. Implementaci tohoto rozhraní v interpretu je třída *OrateResultSetMetaData*.

Úkolem *OrateResultSetMetaData* je správně popsat vlastnosti sloupců ve výsledku dotazu. Jde například o jména, datové typy a další vlastnosti, které je nutné zapouzdřit. Vlastnosti nově zavedených typů atributů jsou interpretovány. Informace o sloupcích, které nebyly nově zavedeny v tomto interpretu, jsou získávány přímo z ekvivalentní instance třídy ResultSetMetaData JDBC ovladače Oracle.

7.4.4 Třídy zapouzdřující časové údaje

Pro snadnější práci s hodnotami času platnosti a transakčního času byly zavedeny dvě nové třídy *OrateInstant* a *OratePeriod*, které lze z výsledků dotazu získat metodami popsanými v podkapitole 7.4.3.

OrateInstant zapouzdřuje čas platnosti událostní tabulky. *OratePeriod* zapouzdřuje čas platnosti stavové tabulky a transakční čas. Obě třídy vnitřně pracují s časem uloženým v milisekundách od půlnoci 1. 1. 1970.

7.5 Vacuuming transakčních záznamů

U některých tabulek podporujících transakční čas je vyžadováno, aby byly jejich záznamy, které byly transakčně ukončeny před relativně definovanou dobou, automaticky z databáze smazány.

Výsledný interpret tento požadavek implementuje využitím Oracle balíčku *DBMS_JOB*, který umožňuje periodicky spouštět příkazy či uložené procedury v databázovém systému.

Při vytváření transakční tabulky, ve které je požadováno zmíněné mazání starých transakčních záznamů, se vytvoří databázový úkol (překlad z angl. job), který je periodicky

spouštěn v přednastaveném intervalu. Při rušení tabulky s podporou vacuumingu transakčních záznamů je potřeba tento nově zavedený databázový úkol také zrušit. Je proto potřeba uchovat identifikátor tohoto úkolu. Ve výsledném interpretu je tento identifikátor uchováván v atributu tabulky `_TEMPORAL_SPEC`.

Některé transakčně ukončené záznamy se mohou v databázi vyskytovat ještě jistou dobu poté, co již měly být smazány – do příštího spuštění databázového úkolu. To sice nemá vliv z hlediska zálohování transakcí, ale může to mít vliv z hlediska objemu dat. Jako budoucí rozšíření interpretu tedy navrhujeme přidat do interpretem přijímané podmnožiny jazyka TSQL2 ještě možnost specifikovat interval spouštění zmíněného databázového úkolu, popř. použít jiných prostředků pro mazání transakčních záznamů.

Kapitola 8

Testování

Nedílnou součástí procesu vývoje jakéhokoli softwarového produktu je i jeho testování. Řídili jsme se tím i my a provedli jsme několik testů, které ověřily funkčnost implementovaného interpretu.

Pro testování byl vyvinut nástroj – konzolová aplikace – pomocí kterého bylo provedeno několik testů. Tato kapitola popisuje tento nástroj a provedené testy.

8.1 Konzole TSQL2

Konzolová aplikace primárně slouží k jednoduchému zadávání příkazů v jazyce TSQL2, které budou interpretovány databázi. Nástroj využívá pro komunikaci s databází pouze ovladače JDBC, který zapouzdřuje implementovaný interpret.

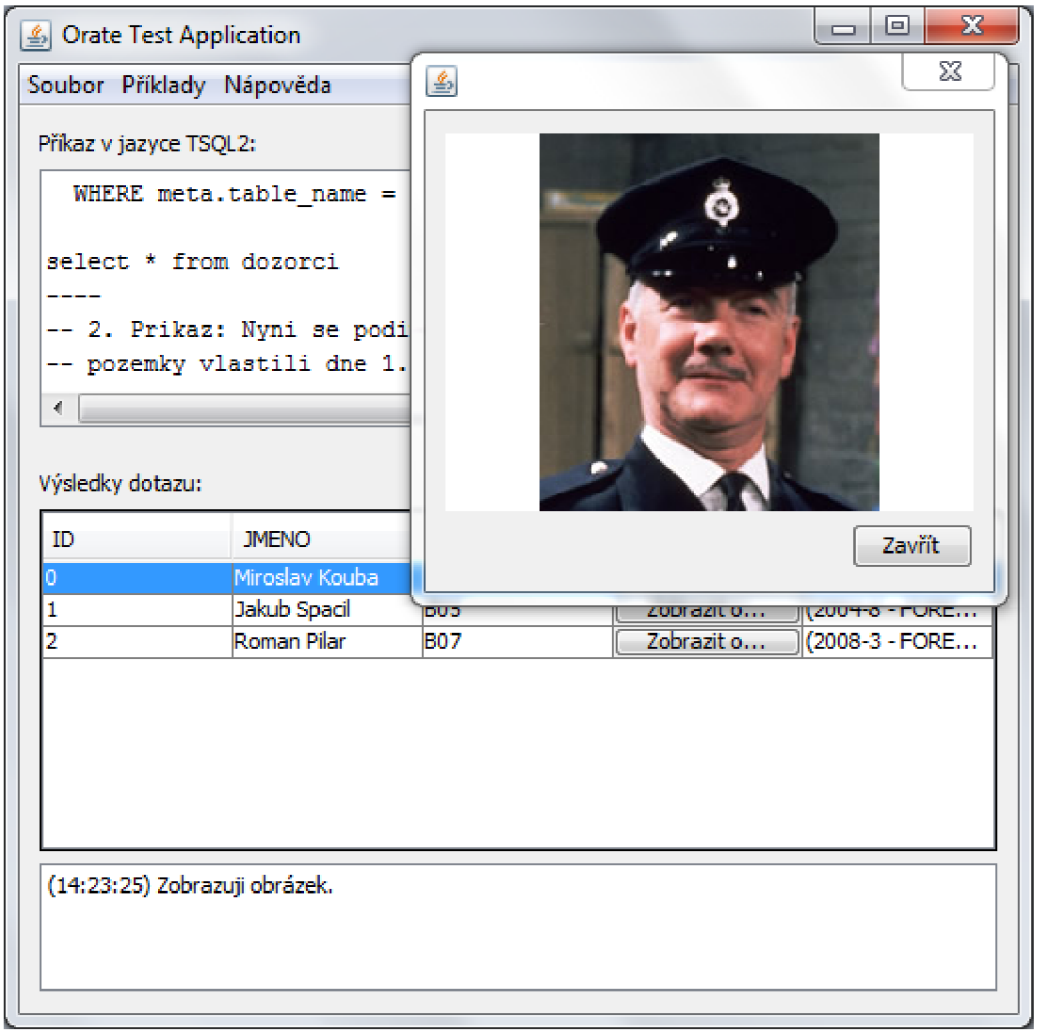
Aplikace poskytuje jednoduché a intuitivní uživatelské rozhraní, jehož prostřednictvím lze po připojení zavést prostředky temporální databáze (tedy meta-tabulky) do databázového systému Oracle. Vstup ve formě příkazů jazyka TSQL2 je možné vkládat pomocí textového panelu. Tento text lze uložit do souboru a později opět nahrát do vstupního okna.

Výsledky databázových dotazů jsou pak zobrazovány pod tímto panelem ve formě tabulky. Speciální funkci má tato tabulka při zobrazování multimediálních a prostorových dat, kdy jsou v buňkách odpovídajících těmto datům tlačítka, pomocí kterých lze zobrazovat obrázky a jednoduché informace o geometrii prostorových dat. Jako zástupce multimediálních dat byly vybrány obrázky pro uživatelsky jednoduchý a názorný způsob jejich reprezentace. O výsledcích všech operací informuje panel umístěný v dolní části okna aplikace.

Jestliže je tato aplikace spuštěna prostřednictvím konzole, pak je možné na standardním výstupu sledovat sekvence SQL příkazů, které jsou ekvivalentní v aplikaci vykonávaným příkazům v TSQL2.

V menu je navíc připravena položka *Příklady*, jejímž účelem je zavést do databáze předpřipravenou sadu dat, které byly využity při testování.

Okno aplikace se zobrazeným obrázkem si můžete prohlédnout na obrázku 8.1.



Obrázek 8.1: Aplikace pro testování ovladače

8.2 Provedené testy

Pomocí výše popsané aplikace byly provedeny sady testů. Jejich úkolem bylo prověřit jak správnost návrhu a implementace temporální podpory, tak i podporu postrelačních dat. Sekundárním účelem testů bylo demonstrovat možnosti praktického využití temporálních databází. Testy operují nad jednoduchými databázemi. První dvě sady testů se řídí smyšleným scénářem – viz podkapitoly.

8.2.1 Multimediální sada testů

Pro tyto skupinu testů jsme vytvořili jednoduchý scénář, ve kterém modelujeme databázi věznice. Věznice má záznamy o dozorcích, kteří v ní pracují, v tabulce `DOZORCI`. Dále je v databázi tabulka `VEZNI`, ve které jsou uloženy záznamy vězňů.

Vězni i dozorci mají u svých záznamů atribut se svou fotografií, a vězni, resp. dozorci jsou umístěni, resp. pracují na určitém bloku, který je dán atributem `BLOK`.

Nad těmito tabulkami byla provedena sada příkazů, jejichž účelem bylo ověřit správnost implementace času platnosti a hlavně možnost ukládání multimediálních dat pomocí výsledného interpretu. Jak již bylo zmíněno výše obrazová data byla zvolena pro jednoduchost a názornost práce s nimi. Multimediální data nelze do databáze vkládat pomocí konzolové aplikace jinak než prostřednictvím automatického vložení dat položky menu.

8.2.2 Sada testů nad prostorovými daty

Tato sada testů modeluje databázi katastrálního úřadu, která je realizována pomocí tří tabulek. Dvě z těchto tabulek – tabulka `UREDNICI` a tabulka `MAPA` – jsou temporální s podporou času platnosti definovaného intervalem (jsou to stavové tabulky).

Tabulka `MAPA` představuje katastrální mapu a záznamy v ní jsou ukládány s přesností na jeden den. Každý záznam modeluje jeden pozemek a jedním z atributů je geometrie pozemku.

Tabulka `UREDNICI` je tabulkou zaměstnanců úřadu. Tyto osoby se starají o vkládání záznamů do tabulky `MAPA`. Úředník na úřadu pracuje po jistou dobu a tuto skutečnost vyjadřuje čas platnosti záznamu v tabulce `UREDNICI`, který je dán s přesností jednoho měsíce.

Poslední tabulkou `VLASTNICI` je standardní snímková tabulka, která uchovává vlastníky pozemků.

Nad těmito tabulkami bylo provedeno několik příkazů tak, aby prověřily správnost implementace času platnosti a zároveň možnost ukládání prostorových dat. Načtení prostorových dat pro testování lze za pomocí tlačítka v menu konzolové aplikace, ale byla ověřena možnost zavedení dat pomocí příkazů v textové podobě.

8.2.3 Sada testů nad relačními daty

Poslední sada testů, která byla provedena pouze nad relačními daty, byla vytvořena za účelem ověření správnosti návrhu a implementace příkazů, které pracují s temporálními tabulkami.

Snahou této sady bylo prověřit zbývající vlastnosti interpretu, které nebyly prověřeny v předešlých testech. Jednalo se o:

- transakční čas;
- čas platnosti v událostních tabulkách;
- bitemporální stavové tabulky;
- bitemporální událostní tabulky.

Pomocí testů jsme ověřili, že návrh i implementace těchto součástí interpretu byl správný.

Kapitola 9

Závěr

Tato práce se zabývá temporálními databázemi a podporou multimediálních a prostorových rozšíření v nich. Hlavní důraz je kladen zejména na využití principů temporálních databází v praktické implementaci.

Práce shrnuje současný stav výzkumu v této oblasti a představuje jazyk TSQL2, který do databázového jazyka SQL zavádí nové konstrukce pro podporu práce s časem. Dále byl zpracován výčet současných implementací databází, které využívají jazyka TSQL2 nebo poznatků získaných při jeho vývoji.

Jsou zde představena dvě postrelační rozšíření databázového systému Oracle, který byl ve zbytku práce použit jako referenční relační databáze. Byly popsány nejdůležitější prvky tohoto systému nutné pro podporu multimediálních a prostorových dat.

Východiskem pro implementaci temporální databáze nad databází relační je návrh interpretu podmnožiny jazyka TSQL2. Byla navržena jeho architektura, reprezentace temporálních dat v databázi, realizace příkazů pracujících s těmito daty a další koncepty nutné pro funkčnost systému. Navržen je i způsob interpretace multimediální a prostorové podpory.

Samostatná kapitola této práce je věnována integritním omezením v temporální databázi. Kontrola těchto omezení je výrazně složitější než v databázích relačních, a to hlavně kvůli zavedení nových časových hledisek, která výrazně mění sémantiku záznamu v temporální tabulce. Jsou diskutovány problémy spojené s kontrolou entitního a referenčního omezení při implementaci temporální databáze nad databází relační a je navrženo řešení těchto problémů. Kontrola temporální entitní integrity byla ve výsledném interpretu implementována.

Dále je zde popsána samotná implementace celého systému, která vychází z výše zmíněného návrhu. Funkční interpret podmnožiny jazyka TSQL2 je hlavním výstupem diplomové práce. V technické zprávě je možné nalézt popis postupu realizace interpretu. Zvláště se věnujeme tématům, která jsou z implementačního hlediska zajímavá. Nejsou opomenuty ani omezení interpretu a problémy, které v průběhu implementace nastaly.

Vývoj interpretu byl završen jeho testováním. Za tímto účelem byl vyvinut nástroj – konzolová aplikace – který správnou funkčnost systému ověřil provedením několika testů.

Oblastí, kde je možné prakticky využít interpret jazyka TSQL2 schopného ukládat multimediální a prostorová data, je velké množství. V úvodu jsme zmínili lékařství či katastrální data. Testy ukázaly možnost využití při evidenci osob.

Širokou oblastí, ve které by se mohly temporální databáze uplatnit, je, zejména kvůli jednoduchosti dotazů nad temporálními daty, implementace datových skladů, které je možné využít pro dolovaní z dat či při statistické analýze.

Výsledná realizace systému je rozsáhlá, avšak nelze říci, že by vývoj v této fázi mohl být ukončen. Stále existuje řada funkcí, které nebyly implementovány, a řada problémů, které nebyly vyřešeny.

Na velkém rozšíření relačních databází pro skladování dat se výraznou měrou podílelo zavedení kontroly referenční integrity. Přestože implementace kontroly temporální referenční integrity je náročnější, jistě by její realizace v interpretu byla potencionálním uživatelům užitečná.

Dále se v interpretu nepodařilo vyřešit realizaci rozhraní *PreparedStatement* pro příkaz `UPDATE` tak, aby odpovídala standardu JDBC. Realizace třídy implementující toto rozhraní se komplikuje výskytem substitučních výrazů v klauzuli `SET`. Řešení tohoto problému zde bylo navrženo.

Ve výsledném interpretu je řešen tzv. vacuuming transakčních záznamů. Toto řešení však uvažuje přítomnost záznamů v tabulce po delší dobu, než je nutné. Z hlediska funkce interpretu tento rys v pořádku, ovšem z pohledu objemu dat nemusí být vždy optimální. Je tedy otázkou, zda nezvážit jiné řešení.

Z hlediska optimalizace prostorové složitosti databáze by bylo užitečné navrhnout a implementovat řešení slévání záznamů v tabulkách s postrelačním obsahem. Interpret realizovaný v této práci je schopen slévat záznamy pouze s atributy relačních datových typů.

Literatura

[1] BÖHLEN, Michael: Temporal Database System Implementations, *ACM SIGMOD Record*. 1995, roč. 24, č. 4, s. 53-60. ISSN 0163-5808.

[2] BÖHLEN, Michael: *The Temporal Deductive Database System ChronoLog*. Curych, 1994, Dizertační práce, ETH Zürich, Departement Informatik.

[3] BÖHLEN, Michael: *Valid Time Integrity Constraints*. Tucson, 1994, Technická zpráva, University of Arizona, Department of Computer Science.

[4] BLOCH, Joshua: *Effective Java Programming Language Guide*. Addison Wesley, 2001, 272 stran, ISBN: 0-201-31005-8.

[5] JENSEN, Christian S, et al.: A consensus glossary of temporal database concepts, *ACM SIGMOD Record*. 1994, roč. 23, č. 1, s. 52-64, ISSN 0163-5808.

[6] SNODGRASS, Richard T, et al.: TSQL2 language specification, *ACM SIGMOD Record*. 1994, roč. 23, č. 1, s. 65-86, ISSN 0163-5808.

[7] STEINER, Andreas: *A Generalisation Approach to Temporal Data Models and their Implementations*. Curych, 1998, 138 stran, Dizertační práce, Swiss Federal Institute Of Technology.

[8] TOMEK Jiří: *TSQL2 interpret nad relační databází*, Brno, 2009, 79 stran, Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií.

[9] *Encyclopedia of Database Systems*. LIU, Ling a ÖZSU, M. Tamer (Eds.). Springer US, 2009, LXX, 3752 stran, ISBN 978-0-387-39940-9.

[10] *SQL/Temporal*. Melton, J. (ed.), 1996, (ISO/IEC JTC 1/SC 21/WG 3 DBL-MCI-0012.).

[11] *ISO/IEC 13249-3 SQL/MM Part 3: Spatial*. Ženeva : ISO, 2006.

[12] *ISO/IEC 13249-5 SQL/MM Part 5: Still image*. Ženeva : ISO, 2003.

[13] *Oracle* [online]. 2010 [cit. 2010-12-27], Database Workspace Manager Developer's Guide, Dostupné z WWW: ⟨http://download.oracle.com/docs/cd/E11882_01/appdev.112/e11826/toc.htm⟩.

[14] *Oracle* [online]. 2010 [cit. 2010-12-27], Multimedia User's Guide, Dostupné z WWW: ⟨http://download.oracle.com/docs/cd/E11882_01/appdev.112/e10777/toc.htm⟩.

[15] *Oracle* [online]. 2010 [cit. 2010-12-27], Spatial Developer's Guide, Dostupné z WWW: ⟨http://download.oracle.com/docs/cd/E11882_01/appdev.112/e11830/toc.htm⟩.

Implementované konstrukce TSQL2

Pro usnadnění práce s interpretem zde uvádíme ucelený přehled nových gramatických konstrukcí jazyka TSQL2, které byly v interpretu implementovány. Podobný text lze nalézt také v [8, Příloha 1].

Výrazy definující časové údaje

V této části popíšeme konstrukce, které v TSQL2 slouží k definici časových údajů. Jde o výrazy `DATE`, `TIME`, `TIMESTAMP` a `PERIOD`.

Výrazy definující časový okamžik

Pro definici časového okamžiku jsou v interpretu implementovány tři výrazy – `DATE`, `TIME`, `TIMESTAMP`. V implementaci interpretu nezáleží na tom, které z těchto klíčových slov použijete pro definici času okamžiku, výrazy mají stejnou syntaxi (až na klíčové slovo) i sémantiku.

Struktura těchto výrazů je následující:

```
INSTANT_DEFINITION = INSTANT_KEY_WORD {"'" DATETIME_STRING "'"
                                      | NOW_RELATIVE} [SCALE]
INSTANT_KEY_WORD = "DATE" | "TIME" | "TIMESTAMP"
DATETIME_STRING = ["BC"] ROK "-" MESIC "-" DEN ["-" HODINA["-" MINUTA["-" SEKUNDA]]]
SCALE = "YEAR" | "MONTH" | "DAY" | "HOUR" | "MINUTE" | "SECOND" | "MILLISECOND"
NOW_RELATIVE = "NOW" {"+"|"-"} INTEGER [SCALE]
```

Volitelné `BC` před datem udává roky před naším letopočtem. Implicitně je brána hodnota přesnosti času platnosti na milisekundy. Rozmezí časových údajů jsou následující:

- `ROK`: od 1 do 250 mil;
- `MESIC`: od 1 do 12;
- `DEN`: od 1 do 31 (podle měsíce);
- `HODINA`: od 0 do 23;
- `MINUTA`: od 0 do 59;
- `SEKUNDA`: od 0 do 59;

Příklady těchto výrazů:

- `DATE '2000-11-14'`: den 14. 11. 2000;
- `DATE '2000-1-1 23:59:59'`: 23:59:59 dne 1. 1. 2000;
- `DATE NOW+1 DAY`: zítra v tomto čase;
- `DATE NOW-2 HOUR`: před dvěma hodinami;
- `DATE NOW+1 YEAR`: příští rok v tomto čase.

Výraz PERIOD

Výraz `PERIOD` definuje časový úsek pomocí dvou stanovených okamžiků. Struktura tohoto výrazu je následující (některé non-terminály viz výše):

```
PERIOD_EXPR = "PERIOD" "["
        {NOW_RELATIVE|DATETIME_STRING|"BEGINNING"}
        "-" {NOW_RELATIVE|DATETIME_STRING|"FOREVER"}
                    "]" [DATE_SCALE]
```

Konstanta `BEGINNING` udává počátek času platnosti (používá se, jestliže nechceme počátek specifikovat) a konstanta `FOREVER` udává konec času platnosti (pokud nechceme určit konec). Příklady těchto výrazů:

- `PERIOD [2000-11-14 - NOW+1] DAY`: od 14. 11. 2000 do zítřka v tuto dobu;
- `PERIOD [NOW-1 - NOW+1] HOUR`: od uplynulé hodiny do příští hodiny;
- `PERIOD [2011-12-23 - 2011-12-24]`: od 00:00:00 dne 23. 12. 2011 do 00:00:00 24.12.2011;

Klauzule VALID

V příkazech DML pro manipulaci se záznamy se využívá klauzule valid. Tato klauzule má následující strukturu:

```
valid_clause = "VALID" {period_expression|INSTANT_DEFINITION}
```

Příklady příkazů s `VALID` klauzulí:

- `INSERT INTO TAB VALUES (3,'RET') VALID DATE '2000-01-01'`:
  - do stavové tabulky bude vložen záznam s počátkem v 00:00:00 1.1.2000 a koncem daným přesností tabulky;
  - do událostní tabulky bude vložen záznam s časem v 00:00:00 1.1.2000.
- `INSERT INTO TAB VALUES (3,'RET') VALID PERIOD [2000-01-01 - 2001-01-01]`:
  - do stavové tabulky bude vložen záznam s počátkem v 00:00:00 1.1.2000 a koncem v 00:00:00 1.1.2001;
  - při pokusu o provedení tohoto příkazu nad událostní tabulkou bude vyvolána výjimka.

- `INSERT INTO TAB VALUES (3,'RET') VALID PERIOD [now+1 - 2050-01-01] YEAR`:
  - do stavové tabulky bude vložen záznam s počátkem rok od momentálního času (patřičně zaokrouhlený podle přesnosti tabulky) a koncem v 00:00:00 1.1.2050;
  - při pokusu o provedení tohoto příkazu nad událostní tabulkou bude vyvolána výjimka.
- `UPDATE TAB SET ID = 3 VALID DATE NOW+1 DAY WHERE ID = 2`:
  - záznamy stavové tabulky budou upraveny v rozmezí časů od zítřka (stejného času) do pozítří (stejného času);
  - viz stavová tabulka.
- `DELETE TAB WHERE ID = 4 VALID PERIOD [NOW - 2015-02-03 12] HOUR`:
  - záznamy stavové tabulky budou smazány v rozmezí časů od momentálního času do 12:00:00 3. 2. 2015.
  - viz stavová tabulka.

Pokud při práci se **stavovou** tabulkou `VALID` klauzule chybí, pak je příkazem `INSERT` vložen záznam s celým intervalem času platnosti (`BEGINNING` až `FOREVER`, viz konstanty specifikované u výrazu `PERIOD`), u příkazu DELETE jsou smazány všechny záznamy vyhovující klauzuli `WHERE` (je-li přítomná, jinak jsou smazány všechny záznamy), u příkazu `UPDATE` jsou upraveny všechny záznamy vyhovující klauzuli `WHERE` (pokud je přítomna, jinak jsou upraveny všechny záznamy).

Pokud klauzule `VALID` chybí při práci s **událostní** tabulkou, pak se příkazem `INSERT` vloží záznam s aktuálním časem. `UPDATE` a `DELETE` viz stavová tabulka.

Definice tabulky

Příkaz `CREATE` má následující strukturu:

```
create_stmt = "CREATE" "TABLE" (
   {COLUMN_DEF | SURROGATE_DEF}
   ) [TEMPORAL_DEF] [VACUUMING_DEF]
SURROGATE_DEF = IDENTIFICATOR "SURROGATE" [CONSTRAINTS]
TEMPORAL_DEF = "AS" "VALID" "STATE" [SCALE] ["AND" "TRANSACTION"]
TEMPORAL_DEF = "AS" "TRANSACTION"
VACUUMING_DEF = "VACUUM" "NOBIND" "(" INSTANT_KEY_WORD NOW_REALATIVE SCALE ")"
```

Příklady příkazů:

- `CREATE TABLE TAB(id SURROGATE, img ORDIMAGE) AS VALID STATE AND TRANSACTION`: vytvoří bitemporální stavovou tabulku se dvěma sloupci, z nichž sloupec id je typu surrogate;
- `CREATE TABLE TAB(id SURROGATE, img ORDIMAGE) AS VALID STATE AND TRANSACTION VACUUM NOBIND(DATE NOW-2 DAY)`: vytvoří bitemporální stavovou tabulku se dvěma sloupci, z nichž sloupec id je typu surrogate a transakční záznamy, jejichž čas byl ukončen před dobou delší než dva dny, budou z tabulky smazány.

Definice, resp. rušení indexu a rušení tabulky má stejnou syntaxi jako v SQL.

Vkládání hodnot do SURROGATE sloupců

Při vkládání hodnot do sloupce typu `SURROGATE` se na patřičné místo v příkazu insert vloží klíčové slovo `NEW`.

Mějme tabulku definovanou výše, příklad příkazu vložení bude vypadat takto:

- `INSERT INTO TAB VALUES (NEW, ORDIMAGE.INIT())`: vloží záznam do tabulky `TAB` s novou hodnotou sloupce `ID` typu `SURROGATE`.

Databázový dotaz

Databázový dotaz má strukturu:

```
select_stmt = "SELECT" ["DISTINCT"] ["SNAPSHOT"] SELECT_LIST
                "FROM" FROM_LIST
                "WHERE" WHERE_PREDICATE

SELECT_LIST = SELECT_ITEM ("," SELECT_ITEM])*
SELECT_ITEM = VALID_FUNC | TRANSACT_FUNC | CAST_FUNC | INTERSECT | SQL_SELECT_ITEM
VALID_FUNC = "VALID" "(" TABLE_NAME ")"
TRANSACT_FUNC = "TRANSACTION" "(" TABLE_NAME ")"
CAST_FUNC = "CAST" "(" VALID_FUNC "AS" "INTERVAL" SCALE ")"
INTERSECT = "INTERSECT" "(" {VALID_FUNC | PERIOD_EXPR}
                          "," {VALID_FUNC | PERIOD_EXPR} ")"

FROM_LIST = FROM_ITEM ("," FROM_ITEM])*
FROM_ITEM = SQL_TABLE_REF | TSQL2_TABLE_REF
TSQL2_TABLE_REF = TABLE_NAME "(" COLUMN_NAME ("," COLUMN_NAME])* ")"
```

`WHERE_PREDICATE` bude popsán pro přehlednost slovně. V klauzuli `WHERE` se můžou vyskytovat tyto binární relační operátory jazyka TSQL2:

- `[NOT] MEETS`
- `[NOT] OVERLAPS`
- `[NOT] CONTAINS`
- `[NOT] PRECEDES`

funkce operátorů je popsána v kapitole 5. Operandy těchto operátorů mohou být `VALID`, `TRANSACTION` či `CAST` funkce. Predikáty s těmito operátory se zapisují infixově.

Příklady databázového dotazu i ostatních příkazů hledejte v připravených příkladech, které jsou obsahem adresáře `examples` na přiloženém nosiči.

Obsah datového média

V kořenovém adresáři datového média se nacházejí dvě složky:

- `program` – programová část práce;
- `text` – textová část práce.

Složka `program` obsahuje složky s tímto obsahem:

- `doc` – programová dokumentace;
- `examples` – příklady k provedeným testům a další soubory, které testy vyžadují. Soubory s příponou „.orate“ lze načíst jako vstup konzolové aplikace;
- `jar-compiled` – přeložené soubory ve formě jar archivu;
- `lib` – knihovny třetích stran nutné pro chod ovladače, resp. konzolové aplikace;
- `properties` – konfigurační soubor konzolové aplikace;
- `src` – zdrojové soubory programů ovladače a konzolové aplikace.

Po přeložení nástrojem `Ant` se vytvoří další složky:

- `build` – přeložené třídy programů;
- `dist` – přeložené programy ve formě jar archivů.

Ve složce `text` s textovou částí práce se nachází tato technická zpráva ve formátu PDF a zdrojový text této zprávy psaný s využitím šablony pro systém TEX.

Manuál konzolové aplikace

Spuštění

Aplikaci je možné spustit příkazem `java` ve složce `dist` následujícím způsobem:

```
java -jar commandLineApp.jar
```

Menu

Popis položek v menu aplikace.

- Připojení – Po spuštění je potřeba se připojit k databázi. V menu *Soubor* je pro tento úkon položka *Připojit*. Po vybrání této položky je nutné do okna zadat řetězec pro připojení ve tvaru:

  ```
  jdbc:oracle:thin:xjmeno00/heslo@berta.fit.vutbr.cz:1521:STUD
  ```

  Po zadání řetězce pro připojení by se v dolní části okna aplikace měla objevit potvrzující informace.

- Zavedení metadat temporální databáze – Po připojení je před použitím temporální databáze nutné zavést metadata do databáze relační. K tomu v menu *Soubor* slouží položka *Zavést databázi*.

  Po výběru této položky by se měla v dolní části okna objevit informace o této operaci.

- Příklady – V menu *Příklady* lze nalézt sady příkazů, které byly použity při testování. Položky tohoto menu jsou rozděleny do skupin podle konkrétní sady dat. Data lze jak importovat do databáze, tak i zrušit. Po správném načtení dat se ve vstupní části okna objeví průvodní text k těmto příkazům.

- Otevírání a ukládání – V menu *Soubor* lze otevřít dříve uložený text vstupní části aplikace pomocí položky *Otevřít*. Text ze vstupní části okna aplikace je možné uložit položkou *Uložit*. Soubory s uloženým textem mají příponu *.orate*.

Vstup

Funkci vstupu v aplikaci plní editovatelné okno, do kterého je možné zadávat příkazy v jazyce TSQL2. Příkazy je možné zadávat pouze bez ukončujícího středníku.

Vykonání příkazu je možné třemi způsoby:

- Pokud je ve vstupním okně pouze jeden příkaz, pak stačí pro jeho vykonání stisknout tlačítko *Proveď*.
- Jestliže je v okně příkazů více, je možné příkaz, který chceme vykonat, označit do bloku a poté stisknout tlačítko *Proveď*.
- Druhým způsobem, jak provést jeden příkaz z množiny více příkazů, je označit tento příkaz do bloku a provést jej pomocí klávesové zkratky `Ctrl` + `Enter`.

Pro vykonání TSQL2 příkazu musí být aplikace připojena k databázi, ve které jsou zavedena temporální metadata – viz Menu.

Výstup

Výstup aplikace realizuje tabulka, která je naplněna výsledky dotazů spouštěných ze vstupní části aplikace.

Speciální funkci má tabulka při zobrazovaní položek tabulky, které mají prostorový datový `SDO_GEOMETRY` typ nebo typ `ORDIMAGE`. V těchto případech je buňka tabulky nahrazena tlačítkem, po jehož stisknutí se objeví okno s vhodně prezentovaným obsahem dané položky tabulky.

Jako výstupní část aplikace lze brát také panel informující o všech provedených operacích, který je umístěn v dolní části okna.

Pokud byla aplikace spuštěna z konzole, pak je možné na standardním výstupu sledovat příkazy v jazyce SQL, které jsou výsledkem překladu právě spuštěného příkazu v jazyce TSQL2.