VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

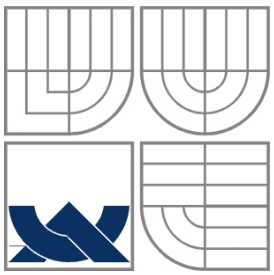
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TSQL2 INTERPRET NAD RELAČNÍ DATABÁZÍ

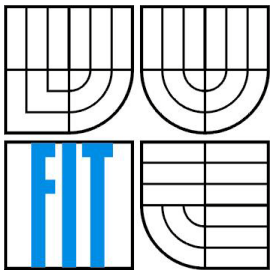
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. JIŘÍ TOMEK
AUTHOR

BRNO 2009



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 RELAČNÍ DATABÁZÍ

PROCESSOR OF TSQL2 ON A RELATIONAL DATABASE SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. JIŘÍ TOMEK

VEDOUCÍ PRÁCE
SUPERVISOR

MGR. MAREK RYCHLÝ

BRNO 2009

Abstrakt

Tato práce se zabývá návrhem a implementací interpretu jazyka TSQL2 pro překlad do SQL. Stručně seznamuje čtenáře s pojmem temporální databáze a přestavuje jazyk TSQL2. Jsou zde také popsány existující implementace temporálních databází a je zhodnocena jejich praktická použitelnost pro správu temporálních dat. Hlavní částí práce je potom popis návrhu a implementace překladače jazyka TSQL2. Výsledkem práce je funkční interpret jazyka TSQL2 implementovaný v jazyce Java jako nadstavba nad ovladač JDBC.

Klíčová slova

databáze, SQL, temporální databáze, TSQL2, Java, JDBC, TimeDB, ChronoLog, interpret, čas platnosti, čas transakce, překladač

Abstract

This thesis is focused on design and implementation of TSQL2 language interpreter for SQL. It briefly introduces temporal database and language TSQL2. Currently available implementations of temporal database systems are introduced and their practical usability for temporal data is reviewed. Main part of this thesis covers design and implementation of TSQL2 interpreter. A working TSQL2 interpreter implemented in Java as JDBC adapter is the result of this work.

Keywords

database, SQL, temporal database, TSQL2, Java, JDBC, TimeDB, ChornoLog, interpreter, valid time, transaction time, compiler

Citace

Tomek Jiří: TSQL2 interpret nad relační databází. Brno, 2009, diplomová práce, FIT VUT v Brně.

TSQL2 interpret nad relační databází

Prohlášení

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

……
Jiří Tomek
19.5.2009

Obsah

1 Úvod

Tato práce se zabývá problematikou temporálních databází, konkrétně specifikací TSQL2. Výsledkem práce je funkční interpret podmnožiny jazyka TSQL2 pro použití nad relační databází.

V dnešní době jsou nejrozšířenějším typem databází databáze relační, které zachycují jeden současný stav dat v nich uložených. Pokud potřebuje aplikace pracovat s více verzemi uložených dat, zpracovávat historii těchto dat nebo jenom evidovat časové údaje, při použití relační databáze musí všechny tyto operace provádět sama a do databáze ukládat již upravená data. Pro evidenci historie tak musí například ke každému záznamu uvádět časové razítko a při výběru dat se na tato časová razítka odkazovat. Pro databázový stroj se ovšem stále jedná o statická aktuální data a nepřiřazuje jim žádnou speciální časovou sémantiku.

Tento nedostatek řeší právě temporální databáze. Pokud vyžaduje aplikace práci s časovým charakterem ukládaných dat, řekne pouze databázi, že daná data jsou temporálního charakteru a o zbytek se již postará přímo databáze. Aplikace tak nemusí data předem upravovat a při dotazech pouze specifikuje o jaký časový úsek nebo okamžik se zajímá a databáze již automaticky poskytne požadovaná data. Aplikační logika se tím může výrazně zjednodušit a aplikace je tak celkově méně náročná.

Tato kapitola práce stručně seznámí čtenáře s pojmy relační a temporální databáze a jazykem SQL.

Druhá kapitola obsahuje přehled dnes dostupných implementací temporálních databází, ať už se jedná o samostatné produkty nebo pouze o rozšířenou funkčnost jiných produktů. Tyto implementace jsou zde představeny a je zhodnocena jejich použitelnost pro práci s temporálními daty.

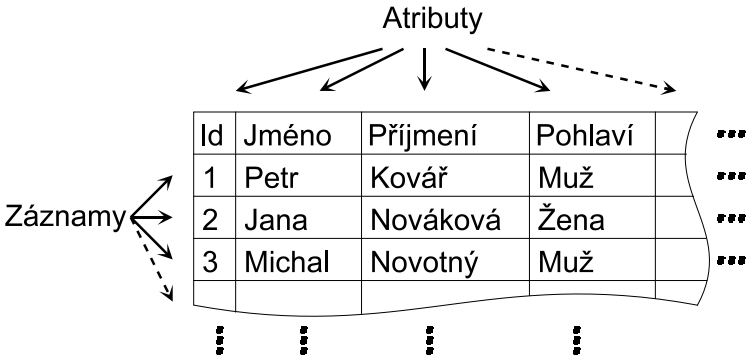
Ve třetí kapitole je představena specifikace TSQL2, která byla vyvinuta jako rozšíření běžného SQL pro práci s temporálními daty.

Čtvrtá kapitola se zabývá návrhem interpretu jazyka TSQL2 za použití čistého SQL. Jsou zde představeny problémy spojené s temporálními daty a jejich ukládání v relační databázi a jsou zde rozebrány přístupy k řešení těchto problémů. Dále se tato kapitola zabývá návrhem samotného interpretu z hlediska programovacího jazyka Java i z hlediska jazyka SQL. Jsou zde rozebrány způsoby překladů a transformací temporálních dotazů na netemporální pro použití v relační databázi.

V páté kapitole je popsána samotná implementace překladače v jazyce Java, je představena struktura překladače a jsou rozebrány a popsány jednotlivé části systému.

1.1 Relační databáze

Relační databáze je druh databáze, ve které jsou data uchovávána jako n-tice **statických** hodnot definovaných atributů. Strukturu uložení těchto dat lze chápat jako tabulky, kdy sloupce tabulky představují jednotlivé atributy a řádky tabulky jsou n-tice hodnot těchto atributů.



Obrázek 1 - Struktura relační databáze

Data uchovávaná v relační databázi mohou mít formu celých čísel, desetinných čísel, řetězců, skupin bytů případně dalších definovaných typů podle typu relační databáze. Komplexnější datové typy je třeba pro uložení v relační databázi převést na základní datové typy databáze a pohled na tato data jako na složitější datové typy musí zajistit aplikace. Data je možné do relační databáze vkládat, upravovat existující data v databázi, mazat data, provádět s daty aritmetické operace, aplikovat na data některé agregační funkce a podobně.

Relační databáze poskytuje data pouze ve stavu, v jakém se v danou chvíli v databázi nacházejí. Samotný systém řízení báze dat není schopen poskytovat pohled na data v závislosti na čase. Není tedy možné zjistit stav dat v určitý okamžik v minulosti nebo provádět operace nad daty, která nejsou aktuální. Z pohledu databáze jsou všechna uložená data aktuální a platná v daný okamžik.

Pokud potřebuje aplikace pracovat s daty, která mají temporální charakter, musí tato data před uložením nejprve převést do formy statických dat, kdy temporální prvky dat převede na běžné atributy a až poté je může uložit do databáze. Naopak při výběru dat z databáze dostane aplikace pouze statická data, kdy temporální prvky z pohledu aplikace jsou pouze další atributy záznamu bez speciální sémantiky. Také dotazy, které mají zohlednit temporální charakter uložených dat, musí aplikace vytvářet sama a databáze je vykonává bez ohledu na sémantický význam atributů.

Z těchto skutečností plyne, že temporální pohled na data musí řešit aplikace a tím se stává složitější. Pro práci s temporálními daty by tedy byl vhodnější jiný typ databáze, který by temporální charakter dat bral v potaz a výše uvedené operace prováděl přímo.

1.2 Temporální databáze

Temporální databáze je druh databáze, která mimo statických dat dokáže uchovávat také temporální data bez ztráty jejich temporálního charakteru. Databáze obsahuje časové údaje označované jako **valid-time** (čas platnosti dat vzhledem k reálnému času) a **transaction-time** (čas kdy byla data přítomna v databázi). Pokud temporální databáze obsahuje oba tyto časy, nazývá se také **bitemporální**.

1.2.1 Temporální data

Temporální data jsou data, která obsahují časovou složku. Při ukládání takovýchto dat do relační databáze je třeba tuto časovou složku modelovat jako další atributy v n-tici a pro relační databázi nemá tato časová složka žádný speciální význam. Naproti tomu u temporální databáze jsou časové složky záznamu definovány zvlášť a databáze s nimi pracuje odděleně od běžných atributů n-tice.

U temporálních dat je možné evidovat dva typy časové informace. Prvním typem je čas platnosti dat. Tento čas určuje, v jakém okamžiku nebo v jaké době byla data platná vzhledem k reálnému světu. Jako příklad vezměme údaje o zaměstnancích v tabulce se sloupci Jméno, Pozice a Plat.

Tabulka 1 – Tabulka s časem platnosti

Jméno	**Pozice**	**Plat**	**VALID**
Petr	programátor	25000	1.5.2007 – 10.10.2008
Martin	grafik	25000	12.3.2007 – NOW
Petr	programátor	30000	11.10.2008 – NOW

Z této tabulky je možné vyčíst, že zaměstnanec Petr dostal k 11.10.2008 přidáno na 30000 Kč a tento plat má dodnes.

Sloupec VALID obsahuje časový údaj o tom, kdy byla daná kombinace ostatních atributů platná v reálném světě a je udržován temporální databází na základě časových údajů poskytnutých aplikací nebo uživatelem. Při vkládání dat může uživatel specifikovat čas platnosti pro tato data a databáze automaticky nastaví časové údaje pro daný záznam. Pokud se jedná o úpravu dat, databáze automaticky rozdělí časový interval mezi původní a novou hodnotu záznamu.

Pokud by se jednalo o data v relační databází, musel by sloupec VALID vytvořit uživatel nebo aplikace a bylo by třeba hodnoty v tomto sloupci explicitně nastavovat.

Někdy je třeba evidovat nejen čas platnosti dat, ale potřebujeme mít přehled o tom, kdy se s daty v databázi manipulovalo. Když se například provádí opravy chybných hodnot nebo se mažou záznamy, může být potřeba zpětně zjistit, kdy ke změně nebo smazání došlo. Proto je v temporálních

databázích k dispozici ještě druhý typ času a tím je čas transakce. Tento čas určuje, kdy byla daná data přítomna v databázi. Pokud vezmeme předchozí příklad dostaneme například následující tabulku.

Tabulka 2 - Tabulka s časem platnosti a s časem transakce

Jméno	**Pozice**	**Plat**	**VALID**	**TRANSACTION**
Petr	programátor	25000	1.5.2007 – 10.10.2008	8.1.2008 – 1.10.2008
Martin	grafik	25000	12.3.2007 – NOW	8.1.2008 – until changed
Petr	programátor	30000	11.10.2008 – NOW	1.10.2008 – until changed

Z časů transakce jednotlivých záznamů můžeme zjistit, že i když Petr a Martin pracovali ve firmě již v roce 2007, záznamy v databázi byly vytvořeny až 8.1.2008. Navíc záznam o navýšení platu pro Petra byl vložen již 1.10 2008, i když k reálnému navýšení platu došlo až 11.10.2008. O údržbu časů transakce se stará databáze transparentně a uživatel nemá možnost tento čas nastavovat. Může se pouze dotazovat na jeho hodnotu.

1.2.2 Využití temporálních databází

Temporální databáze nacházejí využití v mnoha odvětvích lidské činnosti, kde je vhodné nebo dokonce nutné uchovávat časové a historické informace o datech. Jedná se například o:

- finanční data – Bankovní ústavy evidují klientské účty. Stavy účtu a prováděné transakce jsou temporálními daty u kterých je nutné udržovat časové údaje. Musí být možné prohlížet historii stavu účtu nebo výpisy transakcí za určité časové období.
- medicínská data – Údaje o pacientech jako prodělané nemoci, průběh teploty pacienta nebo dávkování léků jsou temporálními daty u kterých je třeba udržovat informaci o časech.
- katastrální data – Je třeba evidovat vlastníky pozemků, přepisy, změny katastrálních map. Všechny tyto údaje jsou vázány na data a časy.
- meteorologická data – Teplota, tlak vzduchu, vlhkost vzduchu se mění v závislosti na čase a je třeba tyto údaje nějak uchovávat.

1.3 Jazyk SQL

SQL (Structured Query Language) je databázový jazyk určený k definici a manipulaci s daty v relačních databázových systémech a pro správu relačních databázových systémů.

Protože relační databáze neobsahují funkce pro práci s temporálními daty, neobsahuje ani jazyk SQL v základní verzi větší podporu práce s časovými údaji. V omezené míře lze s časem pomocí SQL pracovat při použití datových typů jako DATE nebo TIME, ale tyto typy jsou využity pouze jako typy atributů a případný temporální charakter dat musí udržovat aplikace.

1.3.1 Historie

První verze jazyka SQL byla vyvinuta firmou IBM na začátku sedmdesátých let dvacátého století pro manipulaci s daty v jejich databázovém systému System R. Původní název jazyka byl Sequel (Structured English Query Language). [WikiSql]

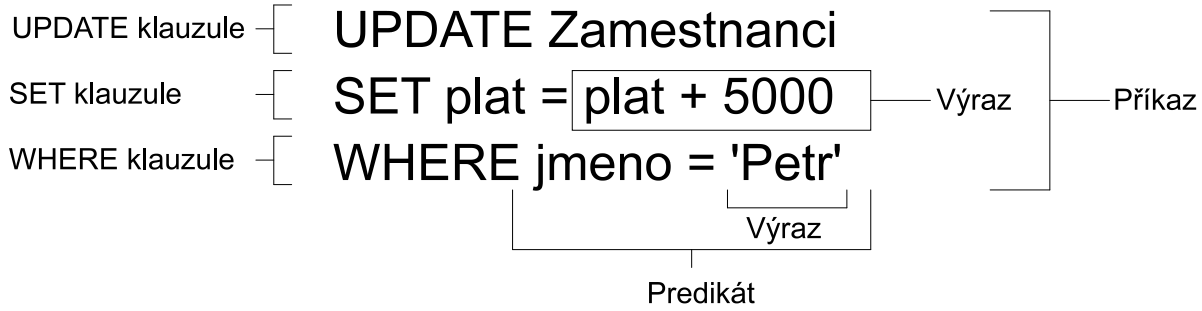
- V roce 1979 vyvinula firma Relational Software (dnes Oracle) první komerční verzi jazyka SQL. [Ora1]
- V roce 1986 je jazyk SQL standardizován organizacemi ANSI a ISO a vzniká tak standard **SQL-86**. Roku 1989 byl ke standardu přidán dodatek o integritě.
- V roce 1992 byl vydán nový standard **SQL-92**, který zavádí datové typy pro práci s časem DATE, TIME, TIMESTAMP, INTERVAL. Z tohoto standardu vychází temporální rozšíření TSQL2.
- Další důležitou verzí byla verze **SQL:1999**, která zavádí databázové triggery, regulární výrazy a některé objektově-relační prvky.
- **SQL:2003** přinesla podporu XML a některé rysy pro podporu OLAP.
- **SQL:2006** upravuje pravidla pro ukládání a manipulaci s daty ve formátu XML.

1.3.2 Struktura

Jazyk SQL se skládá z několika elementů.

- **Příkazy** – Příkaz je řetězec, jehož provedení může modifikovat data v databázi, řídit běh programu, kontrolovat transakce, spojení a další.
- **Dotazy** – Dotaz je druh příkazu, který neovlivňuje databázi, ale slouží pro získání určitých dat z databáze podle definovaných kritérií.
- **Výrazy** – Výraz je řetězec, který se vyhodnotí jako skalární hodnota nebo jako tabulka databáze.
- **Predikáty** – Predikát obsahuje určitou podmínku, která je vyhodnocena jako true, false nebo null a výsledek predikátu se používá v příkazech nebo dotazech pro ovlivnění výsledku.

- **Klauzule** – Klauzule je část příkazu nebo dotazu, která může být i volitelná. Příkazy a dotazy se mohou skládat z více takovýchto klauzuli.



Obrázek 2 - Struktura SQL

2 Existující implementace temporálních databází

Tato kapitola se zabývá existujícími implementacemi temporálních databází a systémy, které temporální databáze podporují. Obsahuje popis jednotlivých řešení a jejich zhodnocení z hlediska použitelnosti pro práci s temporálními daty.

2.1 TimeDB

TimeDB je bitemporální relační databázový systém podporující dotazovací jazyk, DDL, DML i integritní omezení a poskytující temporální rozšíření nad relační databází. Systém je psaný v jazyce Java, a funguje jako nadstavba nad JDBC na principu překladače jazyka ATSQL2 [SBJS96vt][SBJS96tt] na SQL-92. Primárně byl vytvořen pro systém Oracle, ale díky vazbě na JDBC je možné jej používat s různými databázovými systémy. TimeDB poskytuje v jazyce Java vlastní rozhraní TDBCI pro temporální dotazování.

2.1.1 Historie

První verze TimeDB byla vytvořena na Swiss Federal Institute of Technology Zürich jako součást Ph. D. disertační práce Andreasem Steinerem [ST98]. Byla vytvořena v jazyce SCIStus Prolog a později upravena pro SWI Prolog.

Druhá verze byla vytvořena zcela od základu v jazyce Java a využívá rozhraní JDBC. V době psaní této práce je poslední verzí verze 2.2 pro Java 1.4, která byla testována pro databázové systémy IBM Cloudscape 10 a Oracle 10g.

2.1.2 Jazyk ATSQL2

Jazyk ATSQL2 [SBJS96vt][SBJS96tt] navrhli Michael Böehlen, Christian Jensen, Richard Snodgrass a Andreas Steiner jako rozšíření jazyka TSQL2. Jazyk vznikl integrací tří různých přístupů do jednoho. Jedná se o:

- TSQL2 – Temporální rozšíření nad SQL-92 navržené v roce 1994 [TSQL2] [TSQL2Spec]
- ChronoLog – temporální deduktivní databázový systém [BO94]
- Bitemporal ChronoSQL – bitemporální dotazovací jazyk vytvořený Anreasem Steinerem

2.1.3 Forma dotazů

Vzhledem k tomu, že TimeDB využívá jazyk ATSQL2, je syntaxe dotazů prakticky totožná s běžným SQL. Existují zde celkem 3 režimy pro dotazování nad databází. Pro ukázku předpokládejme následující tabulku ZAMESTNANCI.

Tabulka 3 - Ukázková tabulka temporálních dat

VALID	Id	Jmeno	Plat
[2005/05/21-∞)	123	PAVEL	25000
[2005/05/21-2007/01/10)	124	MICHAL	23000
[2007/01/10-∞)	124	MICHAL	30000

Snapshot – Snapshot režim pracuje s temporální databází tak, jako by to byla netemporální databáze obsahující pouze právě aktuální data. Pokud tedy provedeme dotaz na výši platu, vrátí databáze výsledek z aktuálních dat.

```
SELECT Id
FROM ZAMESTNANCI
WHERE Plat < 28000
```

Pokud uvažujeme datum položení dotazu například 2.1.2009, tento dotaz vrátí pouze jeden záznam s Id 123, protože jediný kdo má ve chvíli položení dotazu plat menší než 28000 je Pavel.

Tabulka 4 - Výsledek snapshot dotazu

Id	Jmeno	Plat
123	PAVEL	25000

Sequenced – Režim sequenced používá pro vykonání dotazu temporální charakter záznamů a vrací všechny odpovídající záznamy i s jejich časovými údaji. Je možné určit, která časová informace nás zajímá pomocí klíčových slov **VALID** a **TRANSACTION**. Provedeme tedy výše uvedený dotaz v režimu **VALID**.

```
VALID SELECT Id
FROM ZAMESTNANCI
WHERE Plat < 28000
```

Tento dotaz už vrátí i záznam Michala z minulosti a navíc přidá i časovou informaci

Tabulka 5 - Výsledek sequenced dotazu

VALID	Id	Jmeno	Plat
[2005/05/21-∞)	123	PAVEL	25000
[2005/05/21-2007/01/10)	124	MICHAL	23000

Nonsequenced – V režimu nonsequenced se také pracuje s temporálním charakterem záznamů, ale narozdíl od režimu sequenced se časové údaje VALID a TRANSACTION chovají jako další atributy a je tedy možné s nimi takto pracovat. Můžeme tedy například zjistit, kterým zaměstnancům se měnil záznam.

```
NONSEQUENCED VALID
SELECT BEGIN(VALID(a)) AS Kdy, a.Jmeno
FROM ZAMESTNANCI a, ZAMESTNANCI b
WHERE a.Id = b.Id
  AND VALID(a) MEETS VALID(b)
```

Tento dotaz vrátí záznamy zaměstnanců, u kterých došlo k rozdělení platného času a tedy k nějaké změně.

Tabulka 6 - Výsledek nonsequenced dotazu

Kdy	Jmeno
2007/01/10	MICHAL

2.1.4 Zhodnocení

TimeDB je dnes pravděpodobně nejúplnější implementací temporální databáze pro běžné použití. Díky použití technologie Java a JDBC je snadno přenositelný jak na různé operační systémy, tak na různé databázové systémy. Poslední verze vyšla ovšem v roce 2005 a zdá se, že vývoj již nepokračuje. Stávající verze nepodporuje například příkaz UPDATE a vzhledem k tomu, že se jedná o uzavřenou implementaci a nejsou dostupné zdrojové kódy, není možné opravit některé existující chyby. Přesto se jedná o nejpoužitelnější systém pro praktické použití, pokud potřebujeme implementaci temporální databáze.

2.2 ChronoLog

ChronoLog je temporální deduktivní databázový systém založený na jazyce Prolog navržený v disertační práci Michaela Böhlena Managing Temporal Knowledge in Deductive Databases [BO94]. Je vytvořen jako nadstavba nad komerční relační databázový systém Oracle.

2.2.1 Forma dotazů

Systém poskytuje dva způsoby dotazování nad daty. Protože základem je jazyk Prolog, je možné provádět dotazy pomocí logických predikátů. Tímto způsobem je možné definovat relace, data a následně provádět nad těmito daty dotazy.

Druhou možností je použití jazyka ChronoSQL, který má syntaxi podobnou jazyku SQL, ale v současné době je možné jej v systému ChronoLog použít pouze na dotazování. K definici dat je třeba používat pouze logické predikáty.

Jako příklad uveďme tabulku zaměstnanců a jejich platů:

```
create employee(empNr int,
                name str,
                address str)@period time.

create salary(empNr int, amount num)@period time.
```

Vložíme data zaměstnance Tom.

```
assert employee(123,'Tom','Tucson')@[1993/3/1-1994/10/1).
assert salary(123,3800)@[1993/3-1994/1).
commit.
```

Pokud chceme nyní vybrat z databáze informace o zaměstnanci Tom i s časovým údajem, můžeme použít následující dotaz:

```
?- { employee(EmpNr,'Tom',Address) & salary(EmpNr,Amount)}@Period.
```

To samé lze provést za použití jazyka ChronoSQL. Ve výchozím stavu je ovšem systém v režimu ChronoLog a je tedy třeba nejprve přepnout do režimu ChronoSQL:

```
chronoSQL.
```

Nyní je možné použít ChronoSQL.

```
REDUCE Period OF
  SELECT e.empNr EmpNr, e.address Address, s.amount Amount
  FROM   employee e, salary s
  WHERE  e.EmpNr=s.EmpNr
  AND    e.Name='Tom'.
```

2.2.2 Zhodnocení

V současné době je ChronoLog k dispozici jako samostatný balík obsahující konzoli, grafické rozhraní a emulátor databáze Oracle. Je možné provádět příkazy nad temporální databází pomocí dodané konzole nebo grafického rozhraní. Bohužel neexistuje žádná knihovna pro použití v některém

programovacím jazyce a jedná se tak pouze o demonstrační implementaci pro akademické použití. Pro běžné použití v aplikacích se tedy tento systém zatím nehodí.

2.3 TimeIT

TimeIT – Time-Integrated Testbed – je systém pro testování a vyhodnocování efektivity algoritmů pro zpracování temporálních dotazů. Systém obsahuje generátor temporálních databází a podpůrné nástroje pro testování a vyhodnocování navržených algoritmů na těchto databázích.

Hlavním účelem TimeIT je poskytnutí robustního prostředí pro návrh a testování algoritmů pro temporální databáze. Nejedná se tedy o přímou implementaci temporální databáze použitelnou v produkčním prostředí, ale hodí se při návrhu a vývoji implementace temporální databáze pro testování efektivity navrženého řešení.

2.4 Oracle

Databázový systém Oracle obsahuje operace pro práci s temporálními daty jak na bázi transakčního času, tak na bázi času platnosti. Tyto operace ovšem nejsou plnohodnotnými operacemi temporální databáze.

2.4.1 Podpora transakčního času

Databáze Oracle obsahují určitou formu podpory pro transakční časy. Tato podpora je přítomná ve formě tzv. flashback dotazů pro výběr dat ve tvaru

```
SELECT … AS OF …
```

Tento dotaz je schopen z databáze vybrat data, která se v ní nacházela v konkrétní čas v minulosti. Jedná se tedy o výběr dat z historie databáze. Využití této funkce je několik.

- Obnovení dat po jejich neúmyslné nebo chybné modifikaci nebo smazání.
- Rozdílové porovnání aktuálních dat vůči jejich stavu v minulosti.
- Zjištění stavu konkrétní položky v konkrétní čas v minulosti, například stav účtu.
- Zjednodušení návrhu aplikací, které se tak nemusejí starat o historii dat.

Pokud máme v databází tabulku se stavy účtů a chceme zjistit, kolik bylo na účtu číslo 123456789 peněz dne 3.5.2007 v 15:30, můžeme použít následující dotaz.

```
SELECT balance FROM accounts AS OF TIMESTAMP TO_TIMESTAMP('2007-05-03
15:30:00', 'YYYY-MM-DD HH:MI:SS') WHERE number = '123456789';
```

Podpora flashback dotazů není ovšem ve výchozím stavu aktivní a je třeba nejprve provést několik přípravných kroků k její aktivaci. Je třeba například vytvořit tabulkový prostor o dostatečné velikosti pro ukládání historie změn, přiřadit oprávnění pro provádění flashback dotazů a další.

Oproti podpoře transakčního času u temporálních databází má ovšem tento přístup podstatné omezení v tom, že jediné co dokáže, je vybírat historická data. Nelze porovnávat transakční časy a pracovat s nimi jako s atributy.

2.4.2 Podpora času platnosti

Systém Oracle umožňuje ukládat k záznamům v tabulce časový údaj určující platnost daného záznamu vzhledem k reálnému času. Narozdíl od temporálních databází je ale třeba tuto funkčnost u každé tabulky explicitně povolit. V tabulce se potom vytvoří další atribut, který bude uchovávat interval s platným časem záznamu.

Aktivace podpory temporálních dat a následná práce s těmito daty jsou ovšem prováděny pomocí uložených procedur a ne pomocí přímého SQL.

Vytvoření obyčejné tabulky zaměstnanců a platů

```
CREATE TABLE employees (
  name VARCHAR2(16) PRIMARY KEY,
  salary NUMBER
);
```

Aktivace verzování pro podporu valid-time atributů

```
EXECUTE DBMS_WM.EnableVersioning ('employees', 'VIEW_WO_OVERWRITE',
FALSE, TRUE);

INSERT INTO employees VALUES(
  'Petr',
  30000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-1990', 'MM-DD-YYYY'),
                  TO_DATE('01-01-2005', 'MM-DD-YYYY'))
);
INSERT INTO employees VALUES(
  'Lukáš',
  40000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-2000', 'MM-DD-YYYY'),
DBMS_WM.UNTIL_CHANGED)
);

INSERT INTO employees VALUES(
  'Martin',
  20000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-2003', 'MM-DD-YYYY'),
                  TO_DATE('12-31-9999', 'MM-DD-YYYY'))
);

COMMIT;
```

Nastavení platného času na "vždy" (všimněte si, že neexistují konstanty typu FOREVER jako v TSQL2)

```
EXECUTE DBMS_WM.SetValidTime(TO_DATE('01-01-1900', 'MM-DD-YYYY'),
  TO_DATE('01-01-9999', 'MM-DD-YYYY'));
```

Upravení platu pro Martina od 1.1.2003 – napřed je třeba upravit platný interval.

```
EXECUTE DBMS_WM.SetValidTime(TO_DATE('01-01-2003', 'MM-DD-YYYY'),
DBMS_WM.UNTIL_CHANGED);
```

Skutečné upravení hodnoty platu. Protože se předem upravil interval platnosti, je změna platná od 1.1.2003.

```
  UPDATE employees SET salary = 45000 WHERE name = 'Martin';
```

Zakázání verzování. Tabulka si ovšem stále zachová atribut s platným časem.

```
COMMIT;
EXECUTE DBMS_WM.DisableVersioning ('employees');
```

Jak je vidět z příkladu výše, je práce s temporálními daty v systému Oracle na úrovni databáze poněkud komplikovaná a nejedná se o přímou transparentní podporu temporálních dat, jako je tomu u jiných implementací.

2.4.3 Zhodnocení

Pokud je požadována pouze základní funkčnost pro ukládání historie dat, je možné využít flashback dotazy a není nutné nasazovat jiné řešení temporální databáze. V případě potřeby pracovat s časy platnosti je již situace komplikovanější díky nutnosti používání uložených procedur. Pokud se jedná pouze o ne příliš rozsáhlé použití, dala by se tato funkce využít. Ovšem v případě potřeby plné podpory temporální databáze je vhodné použít jiné řešení, například TimeDB jako nadstavbu nad databázi Oracle.

3 Jazyk TSQL2

Tato kapitola se zabývá popisem specifikace a jazyka TSQL2. Obsahuje stručnou historii a základní koncepty použité při návrhu specifikace TSQL2.

3.1 Popis a stručná historie

Jazyk TSQL2 je temporálním rozšířením jazyka SQL-92 které bylo vydáno v srpnu 1994.

Vytvoření specifikace jazyka TSQL2 předcházelo bezmála 15 let výzkumů na poli temporálních databází. Během těchto let vzniklo několik různých modelů a jazyků pro temporální databáze, ale žádný z nich se neprosadil natolik, aby se stal standardem. Tento fakt pouze bránil rozvoji v této oblasti a proto byla v roce 1993 založena komise ve složení Richard T. Snodgrass, Ilsoo Ahn, Gad Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Christian S. Jensen, Ramez Elmasri, Fabio Grandi, Wolfgang Käfer, Nick Kline, Krishna Kulkarni, Ting Y. Cliff Leung, Nikos Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo, a Surynarayana M. Sripada.

Cílem této komise bylo vytvořit jednotnou specifikaci modelu a dotazovacího jazyka pro temporální databáze. Předběžný návrh specifikace TSQL2 byl vytvořen na začátku roku 1994 a po několika úpravách byla konečná verze specifikace vydána v srpnu 1994.

Při návrhu TSQL2 bylo definováno několik bodů, které mělo rozšíření TSQL2 splňovat.

- TSQL2 nemá rozlišovat mezi identickými a ekvivalentními snímky databáze
- podpora pouze jednoho rozměru platného času (valid-time)
- platný čas musí podporovat časy v minulosti i v budoucnosti
- časové údaje nemají být omezeny rozsahem ani přesností
- TSQL2 má být konzistentní, plně dopředně kompatibilní rozšíření SQL-92
- TSQL2 má umožnit změnu struktury tabulky na libovolné množině atributů
- TSQL2 má umožnit flexibilní temporální projekce, ale syntaxe má jasně ukazovat, když budou použity nestandardní typy projekcí
- operace v TSQL2 nemají spoléhat na žádné explicitní vlastnosti atributů
- podpora temporálních operací má být volitelná na úrovni tabulky
- uživatelsky definovaná podpora času má obsahovat okamžiky, intervaly a doby trvání
- existující agregační funkce mají mít ekvivalenty v TSQL2
- časové operace mají podporovat více druhů kalendářů a jazykových specifik
- má být možné odvozovat temporální a netemporální tabulky z jiných temporálních a netemporálních tabulek
- tabulky v TSQL2 mají být implementovatelné v první normální formě

- TSQL2 musí obsahovat efektivně implementovatelnou algebru, která umožní optimalizace a bude rozšířením standardní algebry
- jazykový datový model má umožňovat více reprezentací datových modelů

3.2 Základní koncepty

V této podkapitole jsou popsány hlavní koncepty specifikace a jazyka TSQL2. Většina konceptů je přiblížena pomocí srovnání se standardem SQL-92 a popisuje rozšíření tohoto standardu pro použití s temporálními daty.

3.2.1 Časová ontologie

Časová osa v TSQL2 je na obou koncích omezená. Definice času neobsahuje informaci o tom, zda je čas modelován diskrétně nebo spojitě a neumožňuje z uživatelského hlediska toto zjistit. Namísto toho definuje pojem *okamžik (instant),* který je podle definice menší, než jeden *chronon,* což je nejmenší jednotka času, kterou lze na dané implementaci přesně namodelovat. Tím pádem je možné určit hodnotu okamžiku pouze přibližně.

Při práci s daty se ovšem čas prezentuje jako diskrétní veličina, která je reprezentována uživatelským nebo výchozím měřítkem. Tímto měřítkem mohou být například sekundy, minuty, dny, roky.

Okamžik je díky tomuto přístupu modelován jako dvojice "časové razítko + měřítko", tedy například 1 den, 2 roky apod. Kromě okamžiku je možné modelovat i interval, který je reprezentován dvěma okamžiky. Jeden okamžik určuje začátek intervalu, druhý potom jeho konec. Interval může být uzavřený, otevřený nebo částečně uzavřený, podle toho, jestli své hraniční okamžiky obsahuje nebo ne.

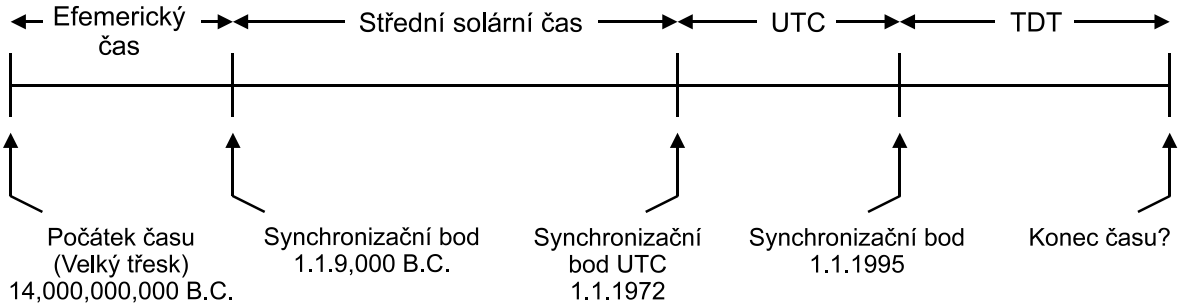
Poslední formou časové informace je doba trvání, která je určena hodnotou a měřítkem, ale nemá konkrétně určený čas výskytu. Obsahuje pouze jedinou informaci, kterou je doba, po kterou daný jev trvá nebo trval.

3.2.2 Základní hodiny

V případě temporálních databází jsou čas a jeho sémantika velmi důležitým prvkem modelu. Standard SQL-92, ze kterého TSQL2 vychází, definuje interně časové údaje jako počet sekund ve formátu UTC. Formát UTC je ovšem začal používat v roce 1958 a je založen na slunečním čase. Díky tomu jej není možné používat pro určení časových okamžiků v dávné minulosti a není proto vhodný jako univerzální časový formát pro temporální databáze.

TSQL2 obsahuje koncept základních hodin, které přiřazují jednotlivým časovým údajům konkrétní sémantický význam a nejsou vázány na žádný konkrétní kalendářový systém. Základní

hodiny rozdělují časovou osu (obrázek) na několik sousedících úseků. Každý z těchto úseků používá samostatné hodiny pro určení času. Hranice úseků se nazývají *synchronizační body*.



Obrázek 3 - Časová osa podle TSQL2

Prvním úsekem na časové ose je interval od velkého třesku do 1. ledna 9000 BC. V tomto úseku je běží základní hodiny podle Efemerických tabulek. Jedná se o astronomické tabulky obsahující pozice planet a hvězd a podle těchto pozic je možné určovat čas.

Druhý úsek od 1. ledna 9000 BC do 1. ledna 1972 využívá pro základní hodiny střední solární čas. Zde se jedná o měření času na základě polohy Slunce.

Do 1. ledna 1972 přechází základní hodiny na čas UTC, protože právě 1. ledna 1972 byl čas UTC synchronizován s atomovými hodinami a byl přijat systém přestupných sekund. Podle standardu TSQL2 měly základní hodiny modelu běžet podle UTC do 23:59:59 31. prosince 1994, kdy měla být přidána přestupná sekunda Od tohoto okamžiku mají základní hodiny v modelu TSQL2 běžet podle Terrestrial Time (dříve TDT, dnes TT). Jedná se nový astronomický standard měření času na Zemi s ohledem na relativitu.

3.2.3 Datové typy

Standard SQL-92 definuje časové typy DATE, TIME, TIMESTAMP a INTERVAL. TSQL2 přidává k těmto typům datový typ PERIOD, který je definován časem začátku a konce a přesností. Přesnost je možné definovat jako číslo, které určuje počet desetinných míst, nebo jako interval (například týden), který určuje, s jakou přesností je doba určena. TSQL2 také rozšiřuje některé operátory pro práci s časovými údaji, jako je porovnávání.

Druhým novým datovým typem, který TSQL2 zavádí je typ SURROGATE. Jedná se o identifikátor, který může být pouze vygenerován a porovnáván na shodu. Uživatel jej nemůže zobrazit ani přímo nastavit, může pouze určit, že má mít daný záznam novou hodnotu tohoto typu, nebo naopak stejnou hodnotu jako jiný záznam. Použití tohoto typu se tedy nabízí jako jedinečný identifikátor objektů, o který se stará systém a nikoliv uživatel.

Práce s typem SURROGATE probíhá následovně.

```
CREATE TABLE Zamestnanci (
      Id SURROGATE,
      Jmeno CHAR,
      Plat INT)
AS VALID STATE
INSERT INTO Zamestnanci VALUES (NEW, 'Petr', 30000)
VALID PERIOD '2005/10/02 – 2006/01/21';
INSERT INTO Zamestnanci VALUES (NEW, 'Honza', 32000)
VALID PERIOD '2005/10/02 – 2006/01/21';
```

Klíčové slovo NEW zde říká, že systém má vygenerovat novou nepoužitou hodnotu a tuto hodnotu použít jako Id. Pokud nyní budeme chtít změnit plat zaměstnance Petr, můžeme to provést následovně.

```
INSERT INTO Zamestnanci
      SELECT (Zamestnanci.Id, 'Petr', 35000)
      VALID PERIOD '2006/01/22 – 2008/12/17')
      FROM Zamestnanci
      WHERE Zamestnanci.Jmeno = 'Petr'
```

Tento příkaz vloží nový záznam o zaměstnanci Petr s novou hodnotou platu, ale zkopíruje hodnotu Id typu SURROGATE ze starého záznamu zaměstnance Petr.

3.2.4 Časové osy

Model TSQL2 obsahuje celkem 3 základní časové osy. Jedná se o čas platnosti, transakční čas a uživatelsky definovaný čas.

Transakční čas je definován jako čas uložení nebo změny záznamu v databázi vzhledem k hodinám databázového stroje. Počátek transakčního času je určen časem vytvoření databáze – `initiation` – a žádná hodnota transakčního času pro danou databázi nemůže být menší než je tento počáteční čas. Konec transakčního času je určen okamžikem následující změny – `until changed` – a žádný záznam nemůže mít větší hodnotu tohoto času. Záznam, který má konec transakčního času nastaven na `until changed` je poslední aktuální verze daného záznamu, protože z významu `until changed` plyne, že nemůže existovat žádná aktuálnější verze.

Čas platnosti a uživatelsky definovaný čas mají dvě speciální hodnoty a těmi jsou počátek – `beginning` – a "navždy" – `forever`. Ty jsou nejmenší respektive největší hodnotou na celé časové ose.

Čas platnosti a uživatelsky definovaný čas mohou být *časově neurčité*. To znamená, že je známo, že událost uložená v temporální databázi se stala, ale není známo přesně kdy.

Temporální hodnoty ukládané v databázi mohou být mimo konkrétního určení času určeny také relativně k aktuálnímu okamžiku. Je možné například zapsat výraz "now – 1 hour", který se v 12:34 interpretuje jako 11:34. Uživatel může při ukládání takového výrazu určit, zda se má uložit relativní nebo interpretovaná hodnota.

3.2.5 Agregační funkce

Stávající agregační funkce standardu SQL-92 jsou upraveny tak, aby přijímaly jako argumenty temporální data a aby bylo možné definovat seskupování pro agregaci také dělením časové osy, tzv. *temporal grouping*. Během výpočtů agregačních funkcí mhou být některé argumenty váženy vzhledem k jejich trvání.

K existujícím agregačním funkcím je přidána i nová funkce s názvem `RISING`, která vrací nejdelší interval po který sledovaná hodnota roste.

3.2.6 Tabulky s časy platnosti

SQL-92 podporuje pouze *snímkové (snapshot)* tabulky. Tyto jsou v TSQL2 zachovány, ale navíc jsou přidány *stavové (state)* tabulky. Ve stavové tabulce je každé n-tici hodnot přiřazen *temporální prvek*, kterým je množina intervalů. Vezměme za příklad tabulku Zamestnanci, která obsahuje sloupce Jméno, Plat a Vedoucí. Tato tabulka obsahuje n-tici (Petr, 25000, Martin). Časové razítko temporálního prvku této n-tice bude obsahovat všechny souvislé nesousedící časové intervaly, kdy zaměstnanec Petr vydělával 25000 Kč a jako vedoucího měl Martina. Ostatní kombinace hodnot budou v jiných n-ticích a budou mít opět vlastní temporální prvky. Časové razítko je implicitně přiřazeno každé n-tici, ale z pohledu uživatele se nejedná o další sloupec tabulky. Rozsah, přesnost a neurčitost těchto časových razítek může být určena uživatelem při vytváření tabulky.

Druhým novým typem tabulky, kterou TSQL2 přináší je tabulka *událostí (event)*. V tomto typu tabulky je každé n-tici místo množiny intervalů přiřazena množina okamžiků, které určují, kdy k dané události došlo. Tento typ tabulky se tedy hodí pro ukládání dat o jednorázových událostech, u kterých je důležitý pouze čas výskytu. Jako příklad můžeme uvést tabulku FotbalovaUtkani, která obsahuje sloupce Domaci, Hoste, Skore. V této tabulce bude například záznam (Sparta, Brno, 0:1). Temporální prvek této n-tice bude obsahovat data a časy utkání mezi Spartou a Brnem, kdy byl výsledek zápasu 0:1 a hrálo se na Spartě. Pokud bylo takovýchto utkání více, temporální prvek bude obsahovat všechny tyto časové údaje.

3.2.7 Tabulky s transakčními časy a bitemporální tabulky

Nezávisle na časech platnosti může být v tabulce zaznamenáván i transakční čas. Transakční čas je temporální prvek asociovaný se záznamem, který říká, kdy byl daný záznam přítomen v databázi vzhledem k reálnému času databázového serveru. Pokud byl například 12.6.2008 vložen do tabulky

záznam a později byl například 5.10.2008 vymazán, bude transakční čas tohoto záznamu obsahovat interval 12.6.2008 – 5.10.2008.

Transakční časy jsou na rozdíl od časů platnosti konkrétně určené a jejich přesnost je závislá na implementaci dané temporální databáze.

Bitemporální tabulka je taková tabulka, které obsahuje čas platnosti i transakční čas dohromady Přitom nezáleží, zda se z hlediska času platnosti jedná o tabulky stavů nebo událostí.

3.2.8 Definice schématu databáze

Základní příkazy SQL-92 CREATE TABLE a ALTER jsou rozšířeny o možnost zadání vlastností vztahujících se k temporálnímu charakteru dat. Jedná se o určení časů platnosti a transakčních časů spolu s jejich přesností a rozsahem.

3.2.9 Restrukturalizace

Pokud je při dotazu typu SELECT vybrána pouze podmnožina ze sloupců v tabulce, TSQL2 dovoluje sloučit temporální prvky u těch n-tic, které mají shodné hodnoty v této podmnožině atributů a vrátit tyto n-tice jako jedinou, která obsahuje sjednocení dílčích temporální prvků. Jako příklad uveďme tabulku Zamestnanci, která obsahuje sloupce Jméno, Plat a Vedoucí. Pokud jsou v tabulce například záznamy (Petr, 25000, Martin) a (Petr, 30000, Martin) a chceme vybrat pouze hodnoty Jméno a Vedoucí, sloučí se ve výsledku výše uvedené záznamy do jednoho (Petr, Martin) a výsledný záznam bude obsahovat temporální prvek složený z temporálních prvků obou původních záznamů.

3.2.10 Temporální selekce

Čas platnosti i transakční čas mohou vystupovat v klauzuli WHERE v predikátech. Pro čas platnosti je možné použít VALID() aplikovaný na název tabulky. Pro transakční čas je možné použít TRANSACTION(). Operátory byly zároveň rozšířeny tak, aby mohly pracovat s temporálními argumenty.

3.2.11 Čištění starých záznamů (vacuuming)

Z principu temporální databáze vyplývá, že pokud je provedena změna záznamu, jedná se na úrovni databáze o vložení nové verze záznamu a pouhou úpravu temporálního prvku u záznamu původního. Podobně smazání záznamu na logické úrovni pouze vede na úpravu temporálního prvku na úrovni fyzické. Díky tomu počet fyzických záznamů v databázi může pouze růst. Jedná-li se o rozsáhlou databázi, může takto časem narůstat do velkých objemů a je tedy vhodné umožnit odstranění již nepotřebných dat. K tomuto účelu slouží funkce čištění (vacuuming). Je možné určit, že data, která jsou starší než daný okamžik se mají fyzicky odstranit z databáze. Tím se velikost databáze dá udržovat v rozumných mezích, ovšem nenávratně se tak ztrácí část historických dat.

3.3 Kompatibilita s SQL-92

Všechny funkce TSQL2 jsou pouhým rozšířením SQL-92. Ukládání a práce s temporálními prvky jsou na nižší úrovni řešeny pouze za použití SQL-92, s databází a tabulkami je stále možné pracovat pomocí příkazů SQL-92. TSQL2 sice zavádí rozšířenou funkčnost některých konstrukcí a operátorů, ale výchozí chování a hodnoty jsou řešeny s ohledem na převoditelnost do běžného SQL-92.

Veškeré platné výrazy a příkazy SQL-92 jsou plně funkční i v TSQL2 a kompatibilita směrem vzhůru je tak plně zajištěna.

4 Návrh interpretu TSQL2

Tato kapitola se zabývá návrhem interpretu jazyka TSQL2 za použití standardního SQL-92. Cílem této snahy je vytvoření překladače TSQL2 do SQL a následné použití tohoto překladače nad běžnou relační databází. Tím by se mělo docílit přidání funkčnosti temporální databáze do libovolné relační databáze.

Tato snaha vede k řešení několika problémů, které můžeme rozdělit do dvou hlavních skupin. První skupinou jsou problémy spojené s reprezentací temporálních dat v relační databázi za použití dostupných prostředků této databáze. Sem patří například reprezentace časů platnosti a transakčních časů, způsob uložení nových metadat k tabulkám, které mají podporovat temporální data nebo řešení integritních omezení vzniklých výskytem temporálních dat.

Druhou skupinou problémů jsou problémy spojené s návrhem samotného interpretu. Zde je třeba řešit efektivní vyjádření temporálních výrazů pomocí SQL nebo převod relačních dat uložených v relační databázi zpět do temporální podoby, ve které byla data do systému ukládána.

4.1 Reprezentace temporálních dat v relační databázi

První část návrhu interpretu TSQL2 musí vyřešit způsob, jak budou temporální data v relační databázi vlastně ukládána. Specifikace TSQL2 definuje podporu temporálních dat na úrovni jednotlivých tabulek databáze. Každá tabulka v databázi tedy může mít rozdílnou úroveň temporální podpory a některé tabulky vůbec temporální data podporovat nemusejí. Je tedy třeba nějakým způsobem ke stávajícím metadatům tabulek databáze přidat další položky, které budou specifikovat právě temporální vlastnosti každé tabulky.

Pokud má tabulka databáze podporovat temporální data, je třeba u řádků takové tabulky udržovat temporální prvky s informacemi o případných časech platnosti a transakčních časech. Tato časová razítka musejí mít volitelnou přesnost a rozsah a musejí pokrývat celý požadovaný prostor. Navíc musejí být časy platnosti a transakční časy vzájemně nezávislé a musí být možno je samostatně měnit nebo i přidávat či odebírat, bez ovlivnění druhého časového údaje.

4.1.1 Uložení metadat k tabulkám

Specifikace TSQL2 definuje celkem tři základní druhy tabulek, které dohromady mohou tvořit šest kombinací. Kromě snímkové (snapshot) tabulky, která neobsahuje podporu temporálních dat, vyžadují všechny ostatní kombinace několik přidaných údajů, které popisují jejich nastavení. Navíc

v případě, kdy se vyskytují v databázi i jiné než snímkové tabulky je třeba i u snímkových tabulek udržovat informaci, že se jedná právě o snímkové tabulky.

Pro ukládán těchto dat v relační databázi je možné použít dva hlavní způsoby.

4.1.1.1 Úprava informačního schématu

Každá SQL databáze obsahuje tzv. informační schéma, ve kterém jsou uložena metadata k databázi a jednotlivým tabulkám. U tabulek se jedná například o název tabulky, typ tabulky, čas vytvoření a další. Tyto informace se uchovávají v tabulce informačního schématu s názvem TABLES. Přidáním několika atributů do této tabulky by tak bylo možné rozlišovat mezi snímkovými a temporálními tabulkami. Hlavní údaje, které je třeba u každé tabulky temporální databáze evidovat, jsou podpora času platnosti, podpora transakčního času a případné maximální povolené staří záznamů pro udržování velikosti databáze v rozumných mezích. Tyto informace přidáme do informačního schématu následujícími příkazy.

```
ALTER TABLE TABLES ADD COLUMN
      VALID_TIME CHARACTER_DATA
      CONSTRAINT VALID_TIME_CHECK
            CHECK (VALID_TIME IN ('STATE', 'EVENT', 'NONE'));
ALTER TABLE TABLES ADD COLUMN
      TRANSACTION_TIME CHARACTER_DATA
      CONSTRAINT TRANSACTION_TIME_CHECK
            CHECK (TRANSACTION_TIME IN ('STATE', 'NONE'));
ALTER TABLE TABLES ADD COLUMN
      VACUUM_CUT-OFF TIMESTAMP;
```

Sloupec VALID_TIME obsahuje informaci o podpoře času platnosti pro danou tabulku. Pokud je hodnota NONE, neobsahuje tabulka žádné časy platnosti. Pokud je hodnota STATE, obsahuje tabulka časy platnosti ve formě intervalů trvání pro jednotlivé záznamy. V případě použití hodnoty EVENT jsou jednotlivé záznamy definovány pouze pomocí jednoho okamžiku výskytu.

Sloupec TRANSACTION_TIME obsahuje naproti tomu informaci o podpoře transakčního času pro danou tabulku. Zde jsou pouze dvě možné hodnoty a to hodnota NONE, kdy tabulka nepodporuje transakční čas, nebo hodnota STATE, kdy je transakční čas určen intervalem, kdy byl daný záznam logicky přítomen v databázi.

Sloupec VACUUM_CUT-OFF se použije v případě, kdy je požadováno automatické odmazávání záznamů starších než určený okamžik. Pokud je zde definována hodnota, jsou průběžně všechny záznamy starší než je hodnota v tomto sloupci fyzicky odmazávány z dané tabulky.

Výše uvedená úprava informačního schématu tedy umožňuje za použití prostředků relační databáze u tabulek udržovat informace o jejich temporálním charakteru. Tyto informace nám ovšem

pouze říkají, jestli daná tabulka podporuje některý druh temporálních dat, nebo ne. Pokud ale tabulka temporální data podporuje, jsou potřeba ještě další informace, které musejí obsahovat údaje o měřítku platného času, o jeho přesnosti a další. Naproti tomu další údaje o transakčním čase již potřeba nejsou, protože přesnost i měřítko transakčního času jsou určeny implementací. Je proto vhodné přidat do informačního schématu novou tabulku, která bude obsahovat metadata pro tabulky platného času. Struktura této nové tabulky by měla podle specifikace TSQL2 vypadat takto.

```
CREATE TABLE TEMPORAL_SPEC {
        TABLE_NAME              CHARACTER_DATA,
        VALID_SCALE             INTERVAL,
        SCALE_GRANULARITY       CHARACTER_DATA,
        VALID_PRECISION         INTERVAL,
        PRECISION_GRANULARITY   CHARACTER_DATA,
        DISTRIBUTION            CHARACTER_DATA,
        GENERAL                 CHARACTER_DATA,
        DEFAULT_EVENT           NONSTANDARD GENERAL INDETERMINATE
                                TIMESTAMP,
        DEFAULT_STATE           NONSTANDARD GENERAL INDETERMINATE
                                PERIOD,
        CONSTRAINT              TEMPORAL_SPEC_PRIMARY_KEY
                                PRIMARY_KEY (TABLE NAME),
        CONSTRAINT              DISTRIBUTION_CHECK
                    CHECK (DISTRIBUTION IN ('STANDARD', 'NONSTANDARD'))
        CONSTRAINT              GENERAL_CHECK
                    CHECK (GENERAL IN ('NONGENERAL', 'GENERAL'))
}
```

Výše uvedená tabulka informačního schématu obsahuje všechny potřebné údaje o temporálních tabulkách databáze. Jedná se hlavně o rozsah, měřítko a přesnost času platnosti dané tabulky a výchozí hodnoty při nezadání temporální informace.

Zde zmíněná úprava informačního schématu má ovšem několik zásadních omezení. Prvním z nich je fakt, že tato úprava může být podle použitého systému platná pro celý databázový systém. Jedná se tedy o globální změnu. To by nemusel být problém v případě, že máme vlastní databázový server a víme, že je to takto pro naše použití nejlepší. Pokud ale máme databázi na nějakém společném databázovém serveru – například webhosting – znamenalo by vytvoření jedné temporální databáze zásah do celého systému.

Druhým problémem jsou potřebná oprávnění pro úpravu informačního schématu. Uživatel běžně takovéto operace s databází provádět nemůže a neměl by tedy možnost vytvoření temporální databáze.

Tyto problémy řeší druhý možný přístup k ukládání metadat k tabulkám.

4.1.1.2 Vytvoření vlastních pomocných tabulek

Druhou možností, jak ukládat potřebná metadata k tabulkám databáze je vytvoření vlastních pomocných tabulek v rámci konkrétní databáze. K tomuto úkonu nepotřebuje uživatel žádná speciální oprávnění mimo oprávnění k vytváření běžných tabulek.

Díky tomu, že pomocné tabulky jsou vytvořeny v dané databázi nebo uživatelském prostoru, neovlivňují nijak ostatní databáze na serveru a je možné tento přístup použít například i na webhostingu.

Co se týče struktury pomocné tabulky, pro potřeby interpretu popisovaného v této práci byla zvolena následující struktura.

```
CREATE TABLE "_TEMPORAL_SPEC" (
     TABLE_NAME VARCHAR(128) NOT NULL,
     VALID_TIME VARCHAR(5) NOT NULL,
     VALID_TIME_SCALE VARCHAR(6) NOT NULL,
     TRANSACTION_TIME VARCHAR(5) NOT NULL,
     VACUUM_CUTOFF NUMBER(20) NOT NULL,
     VACUUM_CUTOFF_RELATIVE NUMBER(1) NOT NULL,
     PRIMARY KEY (TABLE_NAME),
     CONSTRAINT VALID_TIME_CHECK CHECK
             (VALID_TIME IN ('STATE', 'EVENT', 'NONE')),
     CONSTRAINT TRANSACTION_TIME_CHECK CHECK
             (TRANSACTION_TIME IN ('STATE', 'NONE')),
     CONSTRAINT VALID_TIME_SCALE_CHECK CHECK
             (VALID_TIME_SCALE IN ('SECOND', 'MINUTE', 'HOUR', 'DAY',
                                   'MONTH', 'YEAR'))
);
```

Tato tabulka bude obsahovat záznamy pro všechny běžné tabulky v dané databázi. Každý záznam bude obsahovat název dané tabulky, podporu času platnosti (none, event, state), podporu času transakce (none, state), měřítko času platnosti a bod pro odmazávání záznamů.

Protože se v tomto případě jedná o běžnou tabulku databáze, je třeba ošetřit možnosti přístupu k této tabulce ze strany uživatelů. Tuto kontrolu je možné provádět přímo v navrženém interpretu, který jednoduše danou pomocnou tabulku "skryje" tím, že ji bude filtrovat z dotazů a přímé dotazy na danou tabulku vrátí s chybou, že daná tabulka neexistuje.

Aby nedocházelo k častým konfliktům s uživatelskými tabulkami, je vhodné zvolit pojmenování tabulky začínající znakem _. Jako výchozí byl tedy zvolen název `_TEMPORAL_SPEC`, ale tento název je možné v případě potřeby při použití interpretu programově změnit.

Vzhledem k nevýhodám přímé modifikace informačního schématu byla pro tento projekt zvolena možnost vytvoření vlastních pomocných tabulek.

4.1.2 Typ SURROGATE

TSQL2 definuje nový datový typ a tím je typ SURROGATE. Jedná se o speciální typ, který je určen hodnotou a podporuje pouze operace porovnání a vytvoření nové hodnoty. Uživatel nemůže sloupec tohoto typu nastavovat na libovolnou hodnotu, může pouze požadovat vytvoření nové nepoužité hodnoty, nebo porovnat hodnotu ve sloupci s jinou hodnotou stejného typu. Jedná se tedy o ideální typ pro identifikaci temporálních dat o který se stará databáze. Je proto nutné, aby byly někde udržovány informace o tom, která tabulka obsahuje takovýto sloupec. Vzhledem k tomu, že jsme zavrhli modifikaci informačního schématu, vytvoříme v databázi další pomocnou tabulku pro ukládání informací o typech SUROGATE.

```
CREATE TABLE _SURROGATE {
      TABLE_NAME   VARCHAR(128),
      COLUMN_NAME  VARCHAR(128),
      NEXT_VALUE   NUMBER(20),
      PRIMARY_KEY (TABLE_NAME, COLUMN_NAME)
}
```

V této tabulce bude udržován seznam tabulek a sloupců typu SURROGATE. Skutečný typ sloupce SURROGATE v cílové tabulce může být jakýkoliv, ale jako vhodný se jeví číselný typ s dostatečně velkým rozsahem pro identifikaci velkého počtu záznamů. V tabulce _SURROGATE je potom ke každému záznamu čítač, který vždy obsahuje následující volnou hodnotu pro daný sloupec.

Když je vložen do cílové tabulky nový záznam a požaduje novou hodnotu typu SURROGATE, vezme se aktuální volná hodnota z této pomocné tabulky a čítač se o jedničku zvýší.

4.1.3 Vytvoření temporální tabulky

Po modifikaci informačního schématu uvedené v předchozí podkapitole je možné vytvářet tabulky s podporou pro temporální data. Specifikace TSQL2 definuje rozšíření příkazu CREATE TABLE následovně.

```
<table_definition> ::=
      CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE <table_name>
            <table_elements>
            [<temporal_definition>]
            [<vacuuming_definition>]
      [ ON COMMIT { DELETE|PRESERVE } ROWS ]

<temporal_definition> ::=
      AS VALID { STATE | EVENT } [<timestamp_precision>]
            [ AND TRANSACTION ]
      | AS TRANSACTION

<vacuuming_definition> ::=
      VACUUM <datetime_value_expression>
```

Jsou přidány definice času platnosti a transakčního času a definice času pro čištění tabulky od starých záznamů. Pokud je tabulka definována jako VALID STATE, jsou všem řádkům tabulky přiřazeny intervaly platnosti s definovanou přesností a měřítkem. Pokud je tabulka definována jako VALID EVENT, jsou všem řádkům tabulky přiřazeny okamžiky definované přesnosti a měřítka. Pokud je tabulka definována jako TRANSACTION, jsou všem řádkům tabulky přiřazeny intervaly transakčního času, ale přesnost je závislá na implementaci a není možné ji definovat při vytváření tabulky. Pokud je tabulka definována jako VALID i TRANSACTION, jsou řádkům přiřazeny dva časové údaje.

Definice VACUUM pro odmazávání starých záznamů z tabulky je povolena pouze pokud obsahuje tabulka časy transakce, protože definice VACUUM se vztahuje právě k transakčním časům. Není-li klauzule VACUUM specifikována, přiřadí se automaticky hodnota VACUUM TIMESTAMP CURRENT_TIMESTAMP. To znamená, že čas pro odmazávání starých záznamů se nastaví na čas vytvoření tabulky a tím pádem se nikdy žádný záznam neodmaže.

Vezměme tedy následující příkaz pro vytvoření tabulky zaměstnanců.

```
CREATE TABLE Zamestnanci (
     Id             SURROGATE NOT NULL,
     Jmeno          CHARACTER (32) NOT NULL,
     Plat           NUMBER (10),
     Pohlavi        CHARACTER (1),
     DatumNarozeni  DATE,
     PRIMARY KEY (Jmeno),
)
AS VALID STATE DAY AND TRANSACTION
```

Výše uvedený příkaz je třeba přeložit do čistého SQL a vykonat jej jako několik příkazů SQL. Překlad by mohl vypadat nějak takto.

```
CREATE TABLE Zamestnanci (
     Id                  NUMBER (20) NOT NULL,
     Jmeno               CHARACTER (32) NOT NULL,
     Plat                NUMBER (10),
     Pohlavi             CHARACTER (1),
     DatumNarozeni       DATE,
     _vts                NUMBER (8),
     _vte                NUMBER (8),
     _tts                TIMESTAMP,
     _tte                TIMESTAMP,
     PRIMARY KEY (Jmeno)
);

INSERT INTO _TEMPORAL_SPEC (
     table_name,
     valid_time,
     valid_scale,
     transaction_time,
     vacuum_cutoff)
VALUES (
     'Zamestnanci',
     'STATE',
     'DAY',
     'STATE',
     CURRENT_TIMESTAMP
);

INSERT INTO _SURROGATE VALUES ('Zamestnanci', 'Id', 1);
```

Výše uvedené příkazy jsou již standardními příkazy jazyka SQL-92. První příkaz vytvoří standardním způsobem tabulku Zamestnanci. Protože tabulka má obsahovat podporu časů platnosti a transakčních časů, jsou přidány sloupce `_vts`, `_vte`, `_tts` a `_tte` pro uložení času platnosti a transakčního času pro každý řádek. Význam těchto sloupců je popsán v podkapitolách 4.1.4.1 a 4.1.4.2. Navíc je nahrazen typ SURROGATE typem `NUMBER`, který bude reprezentovat identifikátor.

Druhý příkaz přidá záznam právě vytvořené tabulky do tabulky metadat a nastaví příznaky podpory času platnosti a transakčního času na hodnotu STATE. Tím je dáno, že tabulka je bitemporální a podporuje čas platnosti na úrovni intervalů. Transakční čas je vždy typu STATE.

Zároveň je nastavena hodnota pro odmazávání starých záznamů na aktuální datum a čas, takže žádný záznam nebude automaticky odmazán, protože nemůže být starší než je čas vytvoření tabulky. Výchozím měřítkem času platnosti je sekunda. Toto měřítko je dáno implementací.

Poslední příkaz vytvoří záznam o typu SURROGATE pro sloupec Id v tabulce Zamestnanci. Tento záznam potom říká, že sloupec Id je z temporálního pohledu typu SURROGATE a je možné je pouze porovnávat na shodu nebo nastavovat na obecně novou hodnotu.

4.1.4 Ukládání časových údajů k temporálním datům

V předchozí podkapitole byl popsán způsob přidání temporální podpory do tabulky pomocí prostředků poskytovaných SQL. Teď už je tedy možné vytvořit tabulku s podporou temporálních dat, ale zatím není jasné, jak ke konkrétním datům evidovat temporální charakter. Podle typu tabulky může být třeba ke každému záznamu evidovat až dva temporální údaje. Jedná se o čas platnosti a čas transakce. Pokud se jedná o snímkovou tabulku, není třeba evidovat žádný temporální prvek, protože data v takové tabulce nejsou temporálního charakteru.

4.1.4.1 Čas transakce

Je-li tabulka definovaná s podporou transakčního času pomocí klauzule AS TRANSACTION, je třeba u každého záznamu v tabulce evidovat interval, během kterého byl daný záznam logicky přítomen v databázi. Standard SQL-92 sice definuje datový typ interval, ale pouze jako blíže neurčenou dobu trvání. Pro čas transakce ovšem potřebujeme evidovat datum a čas vložení záznamu a datum a čas jeho vymazání. Je tedy třeba použít dvou hodnot, které dohromady plní funkci časově definovaného intervalu.

Asi nejjednodušším a zároveň nejefektivnějším řešení je přidání dalších dvou sloupců do tabulky, která má podporovat časy transakce. Sloupce můžeme nazvat například `_tts` a `_tte`. Pro univerzální záznam času napříč různými databázemi se jako vhodný jeví čas ve formátu unix timestamp. Tento formát udává čas jako počet sekund od 1.1.1970. Vzhledem k tomu, že v době psaní této práce je rok 2009, je nemožné, aby byl čas transakce někdy před datem 1.1.1970.

Všechny běžné databázové systémy jsou schopny ukládat minimálně 32-bitová čísla, která mají v bezznaménkovém formátu rozsah 0 – 4294967295. Pokud předpokládáme, že průměrný rok má 31536000 (365 dní * 24 hodin * 60 minut * 60 sekund), bude tento rozsah stačit minimálně do roku 2100. Většina databázových systémů navíc dokáže ukládat čísla v rozsahu 64-bitů. V tomto případě je při použití formátu unix timestamp možno pracovat s daty zhruba 500 miliard let do budoucnosti, což je již určitě dostatečný rozsah.

Základní přesnost je 1 sekunda a tato přesnost je ve většině případů dostatečná pro potřebu transakčních časů.

Při vložení záznamu do databáze je třeba do sloupce `_tts` uložit aktuální datum a čas. Otázkou ovšem je, jakou hodnotu uložit do sloupce `_tte`, aby dostatečně jasně reprezentovala fakt,

že daný záznam je stále v databázi. Bylo by možné uložit nejmenší možnou hodnotu, ovšem v tom případě by v případě řazení záznamů podle `_tte` došlo k tomu, že aktuální záznamy by byly na začátku posloupnosti místo na konci.

Druhou možností by bylo nastavit hodnotu na NULL. Zde se ovšem objevuje problém plynoucí z charakteru hodnoty NULL a to je vliv této hodnoty na vyhodnocování výrazů. Hodnotu NULL není možné logicky porovnávat a většina výrazů se vyhodnocuje opět jako NULL nebo false.

Další možností by mohlo být přidání dalšího sloupce, který by obsahoval pouze příznak, zda je transakční čas ukončený nebo ne a sloupec `_tte` by mohl mít výchozí hodnotu. Zde je ovšem opět problém s porovnáváním, které by muselo obsahovat podmínky zahrnující hodnotu tohoto přidaného sloupce.

Jako nejvhodnější se tedy jeví nastavení hodnoty sloupce `_tte` na dostatečně velkou hodnotu, která prakticky nemůže nastat. V tomto případě je ovšem třeba se zamyslet nad pojmem "dostatečně velká hodnota", aby nedošlo za několik let k problému, že ona dostatečně velká hodnota již nastala. Není tedy vhodné použít například datum 1.1.2100, protože 91 let nemusí stačit. Zvolíme tedy například hodnotu 1.1.5000, která prakticky jistě nikdy během použití této databáze nenastane. Navíc při řazení záznamů podle času smazání budou nesmazané záznamy umístěny v posloupnosti na správném místě.

Při logickém smazání záznamu stačí potom pouze změnit hodnotu ve sloupci `_tte` na aktuální datum a čas a záznam bude z pohledu databáze logicky vymazán.

Ukažme na příkladu tabulky Zamestnanci uvedeném v předchozí kapitole práci s časy transakce. Nejprve vložíme nový záznam a později tento záznam smažeme.

```
INSERT INTO Zamestnanci (Id, Jmeno, Plat, Pohlavi, DatumNarozeni)
VALUES (NEW, 'Petr Novak', 25000, 'M', '1980-02-13');

DELETE FROM Zamestnanci WHERE Jmeno = 'Petr Novak';
```

Hodnota NEW ve sloupci Id je speciální hodnota pro typ SURROGATE, která říká, že má být vygenerována nová dosud nepoužitá hodnota a ta má být přiřazena do Id. Výše uvedené příkazy se tedy pro podporu transakčního času přeloží na následující příkazy.

```
INSERT INTO Zamestnanci (Id, Jmeno, Plat, Pohlavi, DatumNarozeni,
                         _tts, _tte)
VALUES (NEW, 'Petr Novak', 25000, 'M', '1980-02-13',
        CURRENT_TIMESTAMP, TO_TIMESTAMP('1.1.5000', 'DD.MM.YYYY'));

UPDATE Zamestnanci SET _tte = CURRENT_TIMESTAMP
WHERE Jmeno = 'Petr Novak';
```

Jak je vidět, smazání záznamu v temporální databázi vede pouze na úpravu transakčního času a záznam tak fyzicky zůstává v databázi.

4.1.4.2 Čas platnosti

Je-li tabulka definována s podporou času platnosti pomocí klauzule AS VALID, je třeba u každého záznamu evidovat čas platnosti daného záznamu. Na rozdíl od času transakce, který může být definován pouze jako interval, čas platnosti může být interval nebo okamžik.

Interval

Pokud je čas platnosti definován pomocí AS VALID STATE, musí být u každého záznamu evidován interval jeho platnosti. Podobně jako u času transakce je nejvhodnějším přístupem k uložení intervalu uložení jeho začátku a konce zvlášť do dvou sloupců. Tyto sloupce nazveme třeba `_vts` a `_vte`.

Sloupce `_vts` a `_vte` můžeme podobně jako u transakčního času definovat jako 64-bitová čísla a čas ukládat jako unix timestamp. V tomto případě je implementačně omezen rozsah časové osy platnosti od 1.1.0001 – 31.12.9999. Protože formát unix timestamp ukládá počet sekund od 1.1.1970, je pro data před tímto bodem nutné uložit zápornou hodnotu, která se od tohoto bodu odečte.

Stejně jako u času transakce je třeba vyřešit způsob ukládání konce času platnosti, pokud daný záznam platí "navždy". Protože časová osa kalendáře použitého pro danou tabulku je ohraničená, je možné použít relativní číselnou hodnotu, která se nachází až za koncem časové osy kalendáře. Má-li časová osa výše uvedený rozsah, pro uložení hodnoty "navždy" je možné použít například hodnotu pro 1.1.10000 00:00:00.

Vezměme tabulku Zamestnanci z kapitoly 4.1.3. Pokud chceme do tabulky vložit záznam, který má čas platnosti od 5.3.2008 do "navždy", můžeme to provést následujícím příkazem TSQL2.

```
INSERT INTO Zamestnanci (Id, Jmeno, Plat, Pohlavi, DatumNarozeni)
VALUES (NEW, 'Petr Novak', 25000, 'M', '1980-02-13')
VALID PERIOD [2008-03-05 – forever];
```

Pro určení platnosti dat "navždy" se používá v TSQL2 klíčové slovo `forever`. Pro zjednodušení nyní pomineme přítomnost času transakce v tabulce Zamestnanci a ukážeme pouze překlad do SQL s ohledem na čas platnosti.

```
INSERT INTO Zamestnanci (Id, Jmeno, Plat, Pohlavi, DatumNarozeni,
_vts, _vte)
VALUES (NEW, 'Petr Novak', 25000, 'M', '1980-02-13', 1204671600,
253402297200);
```

V příkazu výše došlo k převodu data 5.3.2008 na počet sekund mezi 1.1.1970 a 5.3.2008 a tato hodnota se uložila jako počátek času platnosti. Konec času platnosti není definován, proto se vložila hodnota za koncem časové osy, v tomto případě tedy počet sekund od 1.1.1970 do 1.1.10000.

Okamžik

Pokud je čas platnosti definován pomocí `AS VALID EVENT`, ukládá se ke každému záznamu pouze jedna časová informace, která určuje, kdy k dané události došlo. Pro uložení této informace postačí sloupec `_vts` a způsob uložení může být stejný jako v případě intervalu. Opět se uloží počet jednotek na časové ose od počátku dané časové osy.

4.2 Překlad příkazů TSQL2 na SQL

Tato podkapitola se zabývá rozborem způsobu překladu různých typů dotazů a příkazů v jazyce TSQL2 na čisté SQL. Každý typ dotazu nebo příkazu je různým způsobem ovlivněn úrovní podpory temporálních dat danou tabulkou. Navíc je nutné většinu příkazů přeložit na několik dílčích příkazů v SQL pro simulaci temporálních operací.

4.2.1 CREATE TABLE

Příkaz pro vytvoření tabulky je jedním z těch jednodušších pro překlad. Syntaxe v TSQL2 je prakticky shodná s SQL, pouze jsou dodány některé volitelné části pro specifikaci temporální podpory tabulky. Podrobněji je rozšíření popsáno v **Chyba! Nenalezen zdroj odkazů.**.

Základním principem při překladu tohoto příkazu je získání definice temporální podpory pro vytvářenou tabulku. Po zjištění temporální podpory víme, jaké atributy bude třeba k datům tabulky přidat (`_vts`, `_vte`, `_tts`, `_tte`). Je také třeba zjistit, zda obsahuje tabulka sloupce typu SURROGATE.

Dále je třeba vytvořit pro novou tabulku záznam v tabulce metadat. To se může provést jednoduchým příkazem `INSERT`, který vloží záznam obsahující název tabulky a temporální podporu do tabulky metadat. Následně se vygeneruje upravený příkaz `CREATE TABLE`, do kterého se doplní definice pro pomocné atributy temporálních dat. Pokud obsahuje definice sloupce typu SURROGATE, je ještě nutné vložit záznamy pro tyto sloupce do pomocné tabulky.

Výsledkem překladu tedy budou dva nebo více příkazů – jeden pro vložení metadat, druhý pro samotné vytvoření tabulky a zbylé pro vytvoření záznamů v pomocné tabulce `_SURROGATE`.

4.2.1.1 Vacuuming

Definice tabulky může obsahovat také část týkající se procesu zvaného "vacuuming", česky by se dalo přeložit jako "odsávání". Jedná se o fyzické odmazávání starých záznamů z tabulky, která podporuje transakční čas. V normálním případě jsou data v tabulce s transakčním časem zachována

i po smazání. To ovšem může vést k neúměrnému růstu tabulek databáze. Při použití funkce vacuuming dochází při každém přístupu k tabulce o odmazávání starých neplatných záznamů podle zadaného času.

Při vytvoření tabulky je tak možné zadat pomocí klauzule `VACUUM` časový údaj, podle kterého se budou staré neplatné záznamy postupně mazat. Tento časový údaj je možné specifikovat přímým zadáním data a času, ale tato definice nemá příliš praktické využití. Užitečnější možností je zadat čas jako relativní hodnotu vzhledem k aktuálnímu okamžiku. Příklad takového příkazu pro vytvoření tabulky může vypadat třeba takto.

```
CREATE TABLE tabulka (
  id INT PRIMARY KEY,
  ...)
AS TRANSACTION
VACUUM NOBIND(DATE NOW-14 DAY)
```

Všimněte si především použití klauzule `NOBIND()`. Tato klauzule říká překladači, že zadaný relativní časový údaj nemá být při překladu přeložen na absolutní hodnotu, ale má být uložen v relativní formě. Díky tomu se při každém přístupu k tabulce bere čas pro vacuuming jako aktuální čas mínus 14 dní.

Pokud by se zadal čas bez použití `NOBIND()`, překladač by při vykonání příkazu nahradil čas konkrétní hodnotou, takže čas pro odmazávání by byl po celou dobu životnosti tabulky nastaven na datum vytvoření mínus 14 dní.

4.2.1.2 Problém primárního klíče

Pro tabulky s podporou temporálních dat nastává jeden problém ohledně primárních klíčů. V běžné tabulce je primární klíč jedinečný a nemohou se vyskytnout dva záznamy se shodným primárním klíčem. V temporální tabulce ovšem může dojít k situaci, kdy je stejný primární klíč přítomen v tabulce několikrát pro různé časy platnosti nebo transakce.

Z logického hlediska je to v pořádku, protože v jeden časový okamžik je platný pouze jeden tento klíč. Z hlediska databáze je to ovšem problém, protože databáze o temporální povaze dat nic neví.

Řešením tohoto problému je rozšíření primárního klíče o atribut času platnosti a času transakce. Překladač tedy musí z překládaného příkazu zjistit, jak je definován primární klíč a podle temporální podpory k tomuto primárnímu klíči přidat atributy času platnosti a času transakce. Tím se zajistí jedinečnost primárního klíče i z pohledu databáze. Databáze potom ale může povolit nekonzistentní primární klíče z logického pohledu, například když se budou času platnosti překrývat. V jeden okamžik v překrývajícím se intervalu potom budou dva shodné primární klíče, ale pro databázi bude

vše v pořádku, protože se budou lišit hranice intervalů. Proto se musí provádět manuální kontrola konzistence, která je ovšem řešena při překladu příkazů INSERT a UPDATE.

4.2.1.3 Příklad

```
TSQL2: CREATE TABLE Zamestnanci (
         id SURROGATE PRIMARY KEY,
         jmeno VARCHAR(32) NOT NULL
       ) AS VALID STATE;

SQL:   CREATE TABLE Zamestnanci (
         id                  NUMBER(20),
         jmeno               VARCHAR (32) NOT NULL,
         _vts                NUMBER (20),
         _vte                NUMBER (20),
         PRIMARY KEY (id, _vts)
       );

       INSERT INTO _TEMPORAL_SPEC (
         table_name,
         valid_time,
         valid_scale,
         transaction_time,
         vacuum_cutoff)
       VALUES (
         'Zamestnanci',
         'STATE',
         'SECOND',
         'NONE',
         CURRENT_TIMESTAMP
       );

       INSERT INTO _SURROGATE VALUES ('Zamestnanci', 'id', 1);
```

4.2.2 DROP TABLE

Pro odstranění jednou vytvořené tabulky slouží stejně jako v SQL příkaz DROP TABLE. Protože všechny tabulky vytvořené pomocí příkazů TSQL2 mají vytvořený záznam v tabulce metadat, je třeba i obyčejný příkaz ke smazání tabulky přeložit z TSQL2 do SQL.

Samotný příkaz DROP TABLE může zůstat ve stejné podobě, ve které je zadán, ale je nutné k němu přidat ještě příkazy pro smazání záznamů z tabulky metadata a případných dalších míst.

Překladač tedy musí z příkazu získat název mazané tabulky a vygenerovat navíc příkazy DELETE, které smažou navázaná metadata.

Výsledkem překladu tak budou celkem tři příkazy – smazání samotné tabulky, smazání metadat z tabulky _TEMPORAL_SPEC a nakonec smazání případných záznamů o sloupcích SURROGATE.

4.2.2.1 Příklad

```
TSQL2: DROP TABLE Zamestnanci;

SQL:   DROP TABLE Zamestnanci;
       DELETE FROM _TEMPORAL_SPEC WHERE table_name = 'Zamestnanci';
       DELETE FROM _SURROGATE WHERE table_name = 'Zamestnanci';
```

4.2.3 INSERT

Příkaz pro vložení záznamů do tabulky je v jazyce TSQL2 prakticky stejný, jako v SQL, pouze je možné specifikovat čas platnosti vkládaných záznamů. Obecný tvar příkazu může mít dvě podoby.

```
INSERT INTO tabulka (sloupce)
VALUES (hodnoty)
VALID PERIOD|EVENT definice_casu_platnosti
```

nebo

```
INSERT INTO tabulka (sloupce)
(SELECT sloupce FROM tabulka WHERE podminka)
VALID PERIOD|EVENT definice_casu_platnosti
```

Hlavní úloha při překladu spočívá v tomto případě v definici času platnosti. Překladač tedy ze zadaného příkazu vybere definici času platnosti a nejprve ověří, jestli daná tabulka čas platnosti vůbec podporuje. Pokud tabulka čas platnosti nepodporuje, ale i přesto je zadán, musí překlad skončit chybou. Naopak pokud tabulka čas platnosti podporuje, ale v příkazu není zadán, bere se čas aktuální. V případě stavové tabulky se navíc bere konec platnosti jako neuzavřený.

Pokud podporuje tabulka čas transakce, musí se ještě přidat časové razítko aktuálního času do času transakce, jako počátek existence záznamu.

Protože jsou možné dva základní tvary příkazu INSERT, je třeba postupovat dvěma rozdílnými způsoby.

4.2.3.1 INSERT s definovanými hodnotami

Pokud se jedná o první případ příkazu, tedy se zadanými hodnotami v části VALUES, je situace jednodušší.

Je třeba do příkazu přidat časové hodnoty pro časy platnosti a transakce jako další atributy. Pokud je v příkazu definovaná část `(sloupce)`, přidají se atributy času platnosti a transakce k seznamu sloupců a hodnoty pro tyto atributy se připojí k seznamu hodnot.

Problém nastává v okamžiku, kdy není část definice sloupců pro vložení v příkazu zahrnuta. V tomto případě musí překladač nejprve získat z databáze seznam sloupců pro danou tabulku. Do tohoto seznamu přidá atributy času platnosti a transakce a dále musí projít seznam hodnot pro vložení, a pokud některá hodnota pro některý sloupec chybí, doplní výchozí hodnotu. Následně připojí hodnoty času platnosti a transakce k seznamu doplněných hodnot.

Pokud by překladač nenačetl seznam sloupců tabulky a pouze by připojil časy, mohlo by dojít k chybnému přiřazení hodnot. Pro příklad mějme tabulku `Zamestnanci` se sloupci `id, jmeno, plat`. Vezměme následující příkaz.

```
INSERT INTO Zamestnanci
VALUES (1, 'Petr');
```

Databáze automaticky doplní položku `plat` na výchozí hodnotu. Kdyby překladač nedoplnil seznam sloupců a výchozí hodnoty, ale pouze přidal čas platnosti, byl by výsledný příkaz například takovýto.

```
INSERT INTO Zamestnanci
VALUES (1, 'Petr', 1234567890, 9999999999);
```

Teď se ale hodnota začátku času platnosti ocitla na místě platu a výsledek příkazu je tedy zcela chybný. Pokud ale překladač provede přidání seznamu sloupců, je výsledek v pořádku.

```
INSERT INTO Zamestnanci(id, jmeno, plat, _vts, _vte)
VALUES (1, 'Petr', DEFAULT, 1234567890, 9999999999);
```

Na místo platu se doplní výchozí hodnota a časy platnosti se přiřadí ke správným sloupcům.

Pokud jsou některé sloupce typu SURROGATE, není možné jim podle definice TSQL2 přiřazovat konkrétní hodnoty. Překladač tedy musí pohlídat, že při vkládání záznamů bude použito klíčové slovo `NEW` pro vygenerování nové hodnoty pro daný sloupec, nebo že bude přiřazena existující hodnota z jiného záznamu.

4.2.3.2 INSERT s výběrem pomocí SELECT

Druhým tvarem příkazu `INSERT` je tvar s poddotazem `SELECT`, který slouží k výběru nových hodnot pro vkládaný záznam. V tomto případě je překlad o něco komplikovanější. Hodnoty pro vložení nejsou v příkazu definovány přímo, ale je použit dotaz `SELECT`, který opět může být v TSQL2. Není tedy možné jednoduše z příkazu extrahovat seznam hodnot a pouze doplnit časové údaje.

Překladač musí provést překlad po částech. Nejprve zkontroluje, jestli jsou definované sloupce pro vložení. Pokud ne, musí překladač stejně jako v prvním případě vytvořit seznam sloupců tabulky,

aby byly vkládané hodnoty jednoznačně přiřazené správným sloupcům. V druhém kroku se překlad příkazu INSERT mění na překlad příkazu SELECT. Překladač musí vytvořit pomocný překladač pro dotazy typu SELECT a pomocí tohoto překladače musí přeložit vnitřní SELECT do podoby SQL příkazu. Problémem ovšem je, jak dostat časové údaje pro vložení do dotazu, když nevkládáme přímo hodnoty, ale pouze výsledky jiného dotazu. Řešením je předat tyto časové údaje překladači dotazu SELECT, který je doplní do seznamu vybraných hodnot a tyto hodnoty se tak dostanou tam, kde je potřebujeme mít. Bližší popis překladu příkazu SELECT se nachází v kapitole 4.2.6.

Pro ukázku mějme tabulku Zamestnanci z předchozích příkladů a vložme záznam zaměstnance Martin, který má mít stejný plat jako Petr a pracuje ve firmě od 1.1.2000.

```
INSERT INTO Zamestnanci
SELECT 2, 'Martin', plat FROM Zamestnanci WHERE jmeno='Petr'
VALID PERIOD [2000-01-01 – FOREVER];
```

Jak bylo řečeno, musí se doplnit seznam sloupců a vnitřní dotaz se musí také přeložit. Výsledek potom bude vypadat takto.

```
INSERT INTO Zamestnanci(id, jmeno, plat, _vts, _vte)
SELECT 2, 'Martin', plat, 946681200, 9999999999 FROM Zamestnanci
WHERE jmeno='Petr';
```

Jak je vidět, časové údaje o času platnosti se vložily do vnitřního dotazu SELECT a jsou vráceny jako konkrétní čísla. Tím se vloží všechny hodnoty správně a příkaz INSERT je tedy opět v pořádku.

Podobně jako u vkládání konkrétních hodnot je třeba provádět kontrolu sloupců typu SURROGATE. Z poddotazu je tak možné pouze přiřazovat hodnoty již existujících sloupců a není možné zadávat hodnoty vlastní.

4.2.4 DELETE

TSQL2 příkaz DELETE FROM pro smazání záznamů z tabulky má prakticky stejnou syntaxi jako příkaz SQL, pouze je možné při mazání specifikovat čas platnosti, ve kterém má mazání proběhnout. Vzhledem k charakteru temporální databáze je ovšem většina logických mazání záznamů pouze aktualizací časových údajů v databázi.

Podle úrovně podpory temporálních dat pro danou tabulku je nutné překlad příkazu DELETE rozdělit do několika skupin. Každý druh temporální tabulky totiž vyžaduje jiný přístup k mazání záznamů a není možné vytvořit obecný univerzální model pro všechny typy tabulek.

4.2.4.1 Snapshot tabulka

Nejjednodušším případem pro mazání záznamů jsou snapshot tabulky. Tyto tabulky nepodporují temporální data na žádné úrovni a je tedy možné mazat záznamy přímo. Příkaz DELETE se v tomto

případě nemusí vůbec upravovat, pouze je nutné kontrolovat, jestli příkaz neobsahuje specifikaci času platnosti. V takovém případě skončí překlad chybou, protože snímková tabulka čas platnosti nepodporuje.

Výsledek vykonání příkazu `DELETE` na snapshot tabulku tedy fyzicky odstraní mazaná data.

4.2.4.2 Transakční tabulka

Druhým případem tabulky v temporální databázi je tabulka s podporou času transakce. V takové tabulce jsou všechny záznamy opatřeny časovým údajem, od kdy do kdy jsou logicky přítomné v tabulce. Fyzická přítomnost těchto záznamů je ovšem až na speciální případy (vacuuming) stálá.

Příkaz `DELETE` na transakční tabulku tedy nesmí žádné záznamy fyzicky odstranit. Místo toho je třeba TSQL2 příkaz `DELETE` přeložit na SQL příkaz `UPDATE`, který pouze vhodně upraví časy transakce pro záznamy v tabulce.

Mějme transakční tabulku `Zamestnanci`, která obsahuje následující data.

Tabulka 7 - Příklad transakční tabulky pro mazání dat

Id	Jmeno	Plat	TRANSACTION
1	Petr	22000	1.3.2000 12:25:35 – until changed
2	Martin	25000	6.8.2002 10:12:05 – until changed
3	Jana	18000	9.9.2005 15:23:31 – 5.5.2008 13:22:14

Nyní na tuto tabulku provedeme příkaz `DELETE`. Jako čas provedení příkazu bereme 12.5.2009 8:21:35.

```
DELETE FROM Zamestnanci;
```

Tento příkaz se musí přeložit na příkaz `UPDATE`, který ukončí transakční časy záznamů v tabulce.

```
UPDATE Zamestnanci SET _tte = 1242109295
WHERE _tte = 'until changed';
```

V příkazu je již nahrazen časový údaj formátem unix timestamp. Příkaz provede logické smazání záznamů z tabulky tím, že ukončí jejich transakční čas. Jak je ovšem vidět v příkazu `UPDATE`, ukončí se pouze transakční čas záznamů, které stále platí. Záznam s id=3 se tak nijak nezmění, protože ten již byl ukončen před vykonáním příkazu.

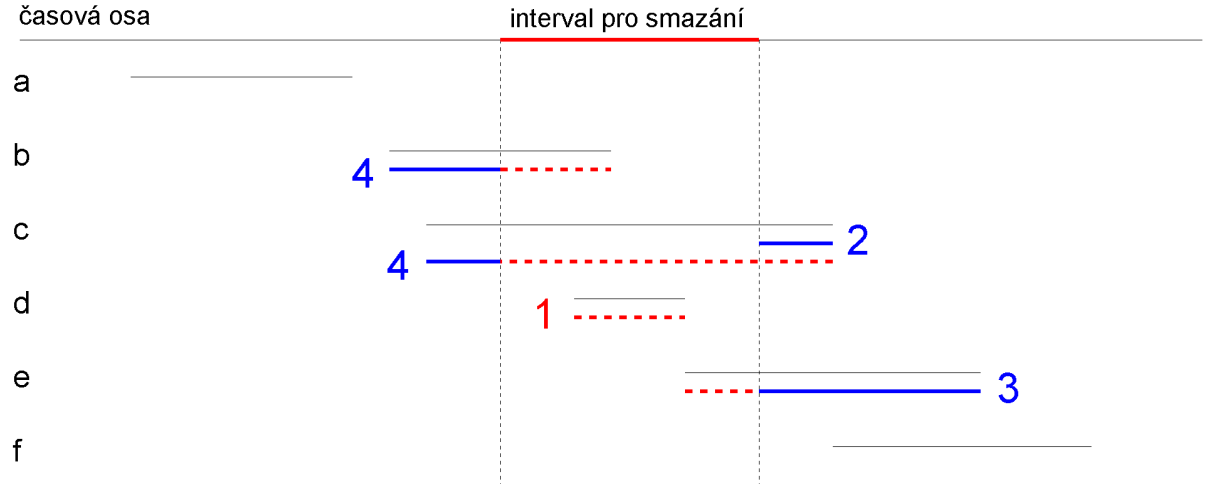
Tabulka 8 - Transakční tabulka po logickém vymazání dat

Id	Jmeno	Plat	TRANSACTION
1	Petr	22000	1.3.2000 12:25:35 – 12.5.2009 8:21:35
2	Martin	25000	6.8.2002 10:12:05 – 12.5.2009 8:21:35
3	Jana	18000	9.9.2005 15:23:31 – 5.5.2008 13:22:14

4.2.4.3 Stavová tabulka

Dalším případem tabulky je tabulka s podporou času platnosti. U takovéto tabulky je už možné provádět mazání i s určením, pro který časový úsek se má mazání provést. Pokud je zadána klauzule `VALID`, musí překladač načíst časové údaje od kdy do kdy trvá interval pro smazání dat. Pokud není klauzule `VALID` zadána, bere se při mazání interval od aktuálního okamžiku dále.

Při mazání ze stavové tabulky je třeba řešit všechny možnosti, jak mohou být záznamy v tabulce vzhledem k intervalu pro mazání umístěny. Tuto situaci ilustruje následující obrázek.



Obrázek 4 - Mazání z tabulky s časem platnosti

Na obrázku je v horní části zobrazena časová osa a na ní je vyznačen interval, ve kterém se má provést smazání záznamů. Jsou zde vidět všechny možnosti, ve kterých mohou být záznamy vzhledem k intervalu mazání. Záznam může být kompletně před tímto intervalem (a), může začínat před a končit uvnitř (b), může začínat před a končit po (c), může být celý uvnitř (d), může začínat uvnitř a končit po (e) a nakonec může být celý po intervalu (f).

Aby byly správně upraveny všechny záznamy, skládá se mazání celkem ze čtyř kroků.

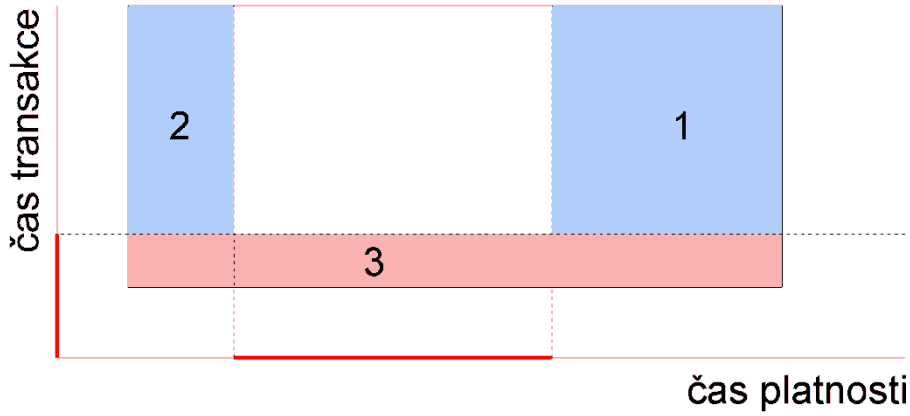
1. Všechny záznamy, které jsou kompletně uvnitř intervalu pro smazání jsou rovnou smazány. Tabulka nepodporuje čas transakce, takže je mazání možné.
2. Záznamy, které přesahují interval pro smazání na obou stranách, jsou zduplikovány a čas platnosti kopie je nastaven na úsek zbývající po mazaném intervalu.
3. Záznamy, které začínají uvnitř záznamu a končí po něm, jsou upraveny tak, že jejich počátek času platnosti je posunut na konec mazaného intervalu.
4. Záznamy začínající před mazaným intervalem a končící uvnitř jsou upraveny tak, že jejich konec času platnosti je posunut na začátek mazaného intervalu.

Jak je vidět na obrázku, všechny části uvnitř mazaného intervalu jsou takto odstraněny a zůstanou pouze části vně tohoto intervalu. I pokud není při mazání čas platnosti zadán a bere se čas od okamžiku mazání dále, výše uvedený postup funguje stejně.

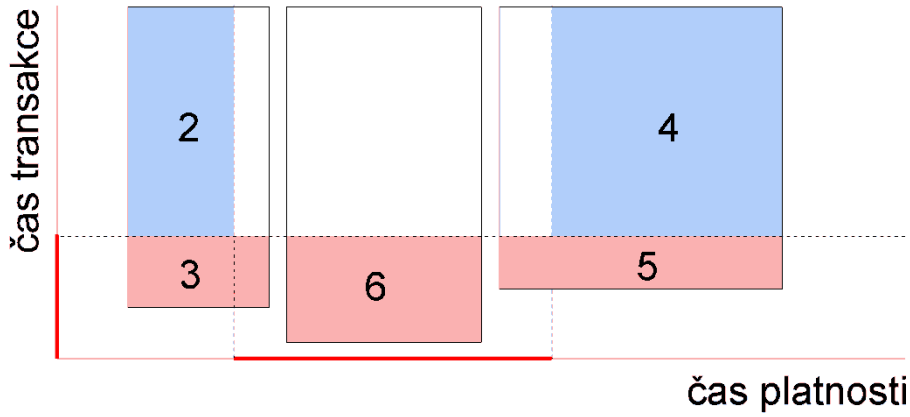
4.2.4.4 Bitemporální tabulka

Posledním a nejkomplikovanějším typem tabulky pro mazání dat je tabulka s podporou času platnosti i času transakce. V této tabulce není možné žádné záznamy fyzicky mazat, pouze se mohou upravovat časové intervaly časů platnosti a transakce. Stejně jako u předchozího typu je možné zadat interval času platnosti pro mazání a pokud není zadán, bere se interval od okamžiku mazání dále. Čas transakce není možné při mazání přímo specifikovat, bere se automaticky okamžik mazání.

Pro ilustraci průběhu mazání poslouží opět nejlépe obrázek.



Obrázek 5 - Mazání z bitemporální tabulky 1



Obrázek 6 - Mazání z bitemporální tabulky 2

Aby byly správně upraveny všechny záznamy, skládá se mazání celkem ze šesti kroků.

1. Záznamy, které jsou stále transakčně platné a obsahují celý mazaný interval se zduplikují tak, že se u kopie nastaví čas transakce od aktuálního okamžiku a čas platnosti se ořízne koncem mazaného intervalu.
2. Záznamy, které jsou stále transakčně platné, začínají před mazaným intervalem a končí v průběhu intervalu nebo po něm se zduplikují tak, že se u kopie nastaví čas transakce od aktuálního okamžiku a čas platnosti se ořízne počátkem mazaného intervalu.
3. Záznamy, které jsou stále transakčně platné, začínají před mazaným intervalem a končí v průběhu intervalu nebo po něm (tedy původní záznamy z bodu 2 před duplikací) se

upraví tak, že se ukončí jejich transakční čas. Tím se uchová původní podoba záznamu v historii transakcí.

4. Záznamy, které jsou stále transakčně platné, začínají před koncem mazaného intervalu a končí po něm se zduplikují tak, že se u kopie nastaví čas transakce od aktuálního okamžiku a čas platnosti se ořízne koncem mazaného intervalu. Tento krok vypadá na první pohled stejně jako krok 1, ale je zde rozdíl v tom, že se berou záznamy, které pouze zasahují do mazaného intervalu, ale neobsahují jej celý. Proto není potřeba je dělit na více částí jako u záznamů obsahujících celý interval.
5. Záznamy, které jsou stále transakčně platné, začínají před mazaným intervalem a končí v průběhu intervalu nebo po něm (tedy původní záznamy z bodu 4 před duplikací) se upraví tak, že se ukončí jejich transakční čas. Tím se uchová původní podoba záznamu v historii transakcí.
6. Nakonec se vezmou záznamy, které jsou stále transakčně platné a jsou kompletně obsažené v mazaném intervalu. U těchto záznamů se pouze ukončí transakční čas a tím se uchovají v historii transakcí.

Jak je vidět z obrázků, původní verze záznamů (červená barva) jsou zachovány v historii (pod dělicí čárou času transakce). Dále jsou vytvořeny nové verze záznamů (nad dělicí čárou času transakce modrou barvou) a tyto nové části jsou již pouze mimo smazaný interval.

4.2.5 UPDATE

TSQL2 příkaz UPDATE pro aktualizaci záznamů v tabulce má prakticky stejnou syntaxi jako příkaz SQL, pouze je možné při aktualizaci specifikovat čas platnosti, ve kterém má aktualizace proběhnout.

Podle úrovně podpory temporálních dat pro danou tabulku je nutné překlad příkazu UPDATE rozdělit do několika skupin. Každý druh temporální tabulky totiž vyžaduje jiný přístup k aktualizaci záznamů a není možné vytvořit obecný univerzální model pro všechny typy tabulek.

4.2.5.1 Snapshot tabulka

Nejjednodušším případem pro aktualizaci záznamů jsou snapshot tabulky. Tyto tabulky nepodporují temporální data na žádné úrovni a je tedy možné aktualizovat záznamy přímo. Příkaz UPDATE se v tomto případě nemusí vůbec upravovat, pouze je nutné kontrolovat, jestli příkaz neobsahuje specifikaci času platnosti. V takovém případě skončí překlad chybou, protože snímková tabulka čas platnosti nepodporuje.

Výsledek vykonání příkazu UPDATE na snapshot tabulku tedy pouze aktualizuje požadovaná data v tabulce.

4.2.5.2 Transakční tabulka

Druhým případem tabulky v temporální databázi je tabulka s podporou času transakce. V takové tabulce jsou všechny záznamy opatřeny časovým údajem, od kdy do kdy jsou logicky přítomné v tabulce. Jakákoliv změna v transakční tabulce tedy musí proběhnout tak, aby byla zachována historie dat v tabulce.

Příkaz `UPDATE` tedy musí v tomto případě proběhnout ve dvou krocích.

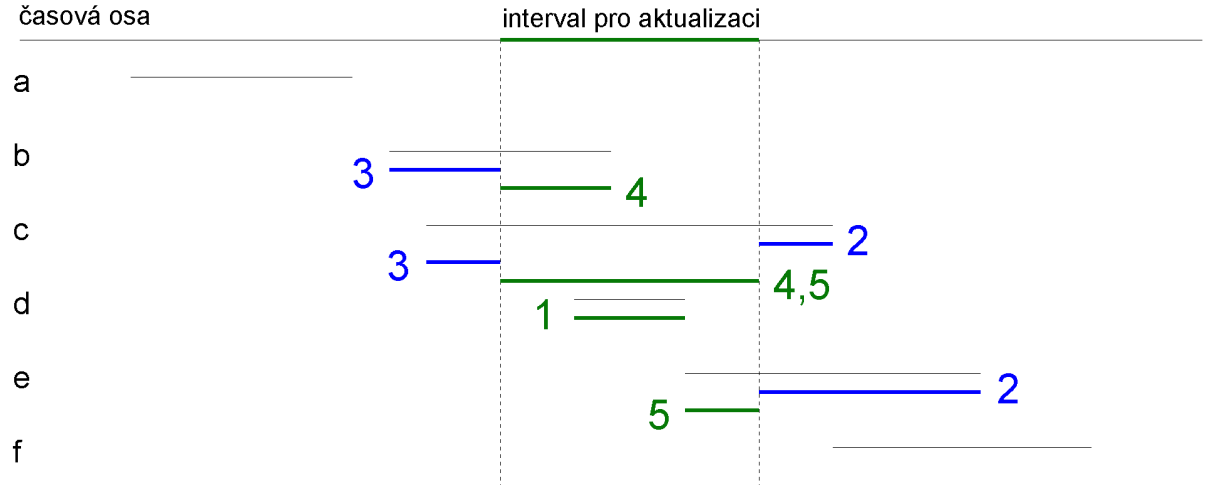
1. Záznamy, které jsou stále transakčně platné a mají se aktualizovat se upraví tak, že se ukončí jejich transakční čas. Tím se tyto záznamy zachovají v historii.
2. Nyní se vezmou tyto původní uložené záznamy a vloží se znovu s aktualizovanými hodnotami tak, že se transakční časy těchto nových záznamů nastaví od aktuálního okamžiku dále.

Tímto postupem máme v tabulce jak původní záznamy, které ale již neplatí, tak nové verze záznamů, které jsou aktuální.

4.2.5.3 Stavová tabulka

Dalším případem tabulky je tabulka s podporou času platnosti. U takovéto tabulky je už možné provádět aktualizaci i s určením, pro který časový úsek se má aktualizace provést. Pokud je zadána klauzule `VALID`, musí překladač načíst časové údaje od kdy do kdy trvá interval pro aktualizaci dat. Pokud není klauzule `VALID` zadána, bere se při aktualizaci interval od aktuálního okamžiku dále.

Při aktualizaci dat ve stavové tabulce je třeba řešit všechny možnosti, jak mohou být záznamy v tabulce vzhledem k intervalu pro aktualizaci umístěny. Tuto situaci ilustruje následující obrázek.



Obrázek 7 - Aktualizace tabulky s časem platnosti

Na obrázku je v horní části zobrazena časová osa a na ní je vyznačen interval, ve kterém se má provést aktualizace záznamů. Jsou zde vidět všechny možnosti, ve kterých mohou být záznamy vzhledem k intervalu aktualizace. Záznam může být kompletně před tímto intervalem (a), může začínat před a končit uvnitř (b), může začínat před a končit po (c), může být celý uvnitř (d), může začínat uvnitř a končit po (e) a nakonec může být celý po intervalu (f).

Aby byly správně upraveny všechny záznamy, skládá se aktualizace celkem z pěti kroků.

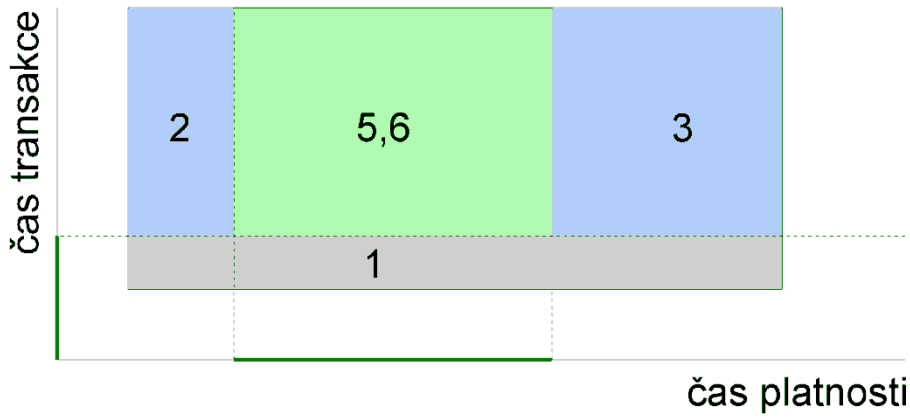
1. Všechny záznamy, které jsou kompletně uvnitř intervalu pro aktualizaci jsou rovnou upraveny do výsledné podoby. Tabulka nepodporuje čas transakce, takže je přímá úprava možná.
2. Záznamy, které začínají před koncem intervalu pro aktualizaci a končí po něm, jsou zduplikovány a čas platnosti kopie je nastaven na úsek zbývající po aktualizovaném intervalu. Hodnoty těchto nových záznamů zůstávají původní, protože tato část leží mimo požadovaný interval.
3. Záznamy, které začínají před daným intervalem a končí někdy po začátku tohoto intervalu jsou také zduplikovány a jejich čas platnosti je ukončen počátkem intervalu. Jejich hodnoty jsou opět původní, protože leží mimo interval.
4. Záznamy, které začínají před daným intervalem a končí někdy po začátku tohoto intervalu (tedy původní záznamy z bodu 3) jsou aktualizovány a počátek jejich času platnosti je nastaven na počátek aktualizovaného intervalu.
5. Záznamy, které začínají před koncem intervalu a končí po tomto intervalu (tedy původní záznamy z bodu 2) jsou aktualizovány a konec jejich času platnosti je nastaven na konec aktualizovaného intervalu.

Jak je vidět na obrázku, všechny části uvnitř aktualizovaného intervalu jsou takto upraveny a části mimo tento interval zůstávají beze změny. I pokud není při aktualizaci čas platnosti zadán a bere se čas od okamžiku aktualizace dále, výše uvedený postup funguje stejně.

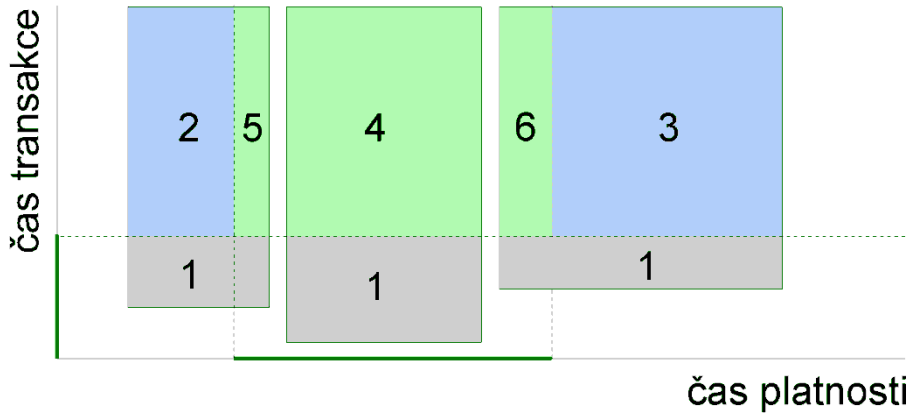
4.2.5.4 Bitemporální tabulka

Posledním a nejkomplikovanějším typem tabulky pro aktualizaci dat je tabulka s podporou času platnosti i času transakce. V této tabulce je nutné veškeré záznamy před provedením změny uchovat v historii v jejich původní podobě. Stejně jako u předchozího typu je možné zadat interval času platnosti pro aktualizaci a pokud není zadán, bere se interval od okamžiku aktualizace dále. Čas transakce není možné při aktualizaci přímo specifikovat, bere se automaticky okamžik aktualizace.

Pro ilustraci průběhu aktualizace poslouží opět nejlépe obrázky.



Obrázek 8 - Aktualizace bitemporální tabulky 1



Obrázek 9 - Aktualizace bitemporální tabulky 2

Aby byly správně upraveny všechny záznamy, skládá se aktualizace celkem ze šesti kroků.

1. Záznamy, které jsou stále transakčně platné a překrývají se s aktualizovaným intervalem, se zduplikují tak, že se u kopie ořízne čas transakce aktuálním časem. Tím se vytvoří v historii původní verze upravovaných záznamů.
2. Záznamy, které jsou stále transakčně platné, začínají před daným intervalem a končí v průběhu intervalu nebo po něm se zduplikují tak, že se u kopie nastaví čas transakce od aktuálního okamžiku a čas platnosti se ořízne počátkem daného intervalu.
3. Záznamy, které jsou stále transakčně platné, začínají před koncem intervalu a končí po něm, se zduplikují tak, že se u kopie nastaví čas transakce od aktuálního okamžiku a čas platnosti se ořízne koncem daného intervalu.
4. Záznamy, které jsou stále transakčně platné a jsou celé obsaženy v daném intervalu se přímo upraví na nové hodnoty a jejich transakční čas se nastaví od aktuálního okamžiku.
5. Záznamy, které jsou stále transakčně platné, začínají před mazaným intervalem a končí v průběhu intervalu nebo po něm (tedy původní záznamy z bodu 2 před duplikací) se aktualizují, jejich transakční čas se nastaví na aktuální okamžik a počátek jejich času platnosti se nastaví na počátek intervalu.

6. Nakonec se vezmou záznamy, které jsou stále transakčně platné, začínají před koncem daného intervalu a končí po něm (tedy původní záznamy z bodu 3 před duplikací) se aktualizují, jejich transakční čas se nastaví na aktuální okamžik a konec jejich času platnosti se nastaví na konec intervalu.

Jak je vidět z obrázků, původní verze záznamů (šedá barva) jsou zachovány v historii (pod dělicí čárou času transakce). Jsou vytvořeny úseky záznamů s původními hodnotami a aktuálním transakčním časem (modrá barva). A nakonec jsou vytvořeny aktualizované verze záznamů uvnitř daného intervalu (zelená barva).

4.2.6 SELECT

Nejčastěji používaným a také co do možné složitosti nejkomplexnějším dotazem je dotaz typu `SELECT`. Jazyk TSQL2 upravuje syntaxi tohoto dotazu nejvíce ze všech ostatních typů uvedených výše. Podrobnější popis rozšířené syntaxe je možné nalézt v příloze 1, tato podkapitola je zaměřena na popis způsobu překladu dotazu do SQL.

Obecný formát dotazu `SELECT` je následující.

```
SELECT <select_list> FROM <tables> WHERE <conditions>
```

Vzhledem k možné komplexnosti tohoto typu dotazu je vhodné provádět překlad po jednotlivých částech uvedených výše. V následujících podkapitolách budou tedy postupně představeny jednotlivé kroky zpracování a překladu dotazu.

4.2.6.1 Select list

První částí dotazu, která je překladačem zpracována je tzv. **select list**. Select list je čárkou oddělený seznam hodnot, které se mají vybrat jako výsledek dotazu. Může se jednat o konkrétní číselné nebo řetězcové hodnoty, sloupce tabulek, výrazy a další.

Jazyk TSQL2 přidává do skupiny možných objektů pro výběr v select listu možnost vybírat časové údaje temporálních dat. Jedná se o čas platnosti záznamu, který se definuje pomocí funkce `VALID(tabulka)`, a čas transakce záznamu, který se definuje pomocí funkce `TRANSACTION(tabulka)`. Při zadání výběru času platnosti nebo transakce jsou tyto hodnoty vráceny jako intervaly, případně jako konkrétní okamžik.

Překladač tedy postupně prochází jednotlivé položky ze seznamu k výběru. Pokud narazí na funkci `VALID()`, nahradí tuto položku dvojicí položek, kterými vybere počátek a konec času platnosti daného záznamu. Pokud narazí překladač na funkci `TRANSACTION()`, nahradí tuto položku dvojicí položek pro výběr začátku a konce času transakce daného záznamu.

Výše uvedené funkce pro získání časových údajů mohou být navíc pomocí funkce `CAST()` převedeny na jiné měřítko času. V případě že se nachází ve výběru funkce `CAST()`, provede

překladač nahrazení této funkce matematickým výrazem, který vrátí výsledek v požadovaném měřítku.

4.2.6.2 FROM <tables>

V běžném SQL může být v části `FROM` uveden seznam tabulek pro výběr, volitelně s jejich aliasy, nebo zde mohou být další dotazy `SELECT` pro výběr z výsledků těchto dotazů.

Jazyk TSQL2 přidává k těmto možnostem další položku a tou jsou tabulky omezené na konkrétní sloupce. Formát zápisu je v tomto případě následující.

```
nazev_tabulky(sloupec1, sloupec2, ...)
```

Tento zápis říká, že se mají data vybírat z tabulky, ve které jsou všechny záznamy sloučeny pouze podle hodnot zadaných sloupců. Tato funkce slouží k tomu, aby bylo možno vybírat nejdelší spojité časové úseky pro sledované hodnoty a tyto úseky nebyly rozděleny změnami jiných hodnot. Nejlépe se to ukáže na následujícím příkladu.

Mějme tabulku `Zamestnanci`, ve které jsou sloupce `id`, `jmeno` a `plat` a která podporuje časy platnosti. V této tabulce jsou záznamy o změnách platů zaměstnanců po dobu jejich působení ve firmě. V této tabulce budou následující data.

Tabulka 9 - Příklad na slévání časů platnosti - zadání

Id	Jmeno	Plat	VALID
1	Petr	20000	1.1.2000 – 8.6.2002
1	Petr	25000	8.6.2002 – 9.12.2005
2	Martin	23000	18.1.2001 – 16.3.2003
2	Martin	28000	16.3.2003 – 25.7.2006

Nyní chceme z této tabulky zjistit, od kdy do kdy pracovali zaměstnanci v dané firmě. Pokud se pokusíme vybrat požadovaná data pomocí standardního formátu SQL dotazu, nedostaneme příliš užitečné informace a data budeme muset ještě aplikačně zpracovat.

```
SELECT jmeno FROM Zamestnanci
```

Tento dotaz nám totiž vrátí následující výsledek.

Tabulka 10 - Příklad na slévání - chybný výsledek

Jmeno	VALID
Petr	1.1.2000 – 8.6.2002
Petr	8.6.2002 – 9.12.2005
Martin	18.1.2001 – 16.3.2003
Martin	16.3.2003 – 25.7.2006

Databáze v tomto případě prostě provedla pouze jednoduchou projekci a vrátila hodnotu sloupce `jmeno` pro všechny řádky spolu s časy platnosti těchto řádků. Toto chování ovšem upravuje

právě zmíněná nová možnost zadání tabulky pro výběr. Pokud upravíme dotaz s využitím této nové syntaxe, dostaneme rovnou požadované výsledky.

```
SELECT jmeno FROM Zamestnanci(jmeno)
```

Překladač v tomto případě provede sloučení řádků pouze na základě hodnoty ve sloupci `jmeno` a výsledek dotazu tak přímo řekne, od kdy do kdy daný člověk ve firmě pracoval.

Tabulka 11 - Příklad na slévání - správný výsledek

Jmeno	VALID
Petr	1.1.2000 – 9.12.2005
Martin	18.1.2001 – 25.7.2006

Nyní jsou výsledkem pouze dva řádky, pro každého zaměstnance jeden, a časy platnosti těchto řádků jsou spojením původních dílčích časů platnosti.

Výše uvedený příklad tedy objasnil význam nové formy zápisu tabulek pro výběr. Nyní tedy můžeme rozebrat, jak vlastně překladač tento nový zápis zpracuje.

Jakmile narazí překladač při čtení tabulek pro výběr na tuto novou formu zápisu, musí nejprve vybrat z této položky jméno zdrojové tabulky a jednotlivé sloupce, přes které se má spojení provést. Následně je třeba vybrat všechny záznamy ze zdrojové tabulky a seřadit je podle uvedených sloupců a časů platnosti. Potom překladač postupně jednotlivé záznamy projde a ty které může, sloučí do jednoho s rozšířeným časem platnosti. Jakmile má všechny záznamy takto sloučené, vytvoří dočasnou databázovou tabulku, do které tyto záznamy uloží. Do seznamu tabulek k výběru vloží na místo původního prvku jméno právě vytvořené dočasné tabulky.

Tento postup zajisté není příliš efektivní, protože dochází k výběru dat a jejich opětovnému vkládání, navíc s vytvořením dočasné tabulky. Bohužel ale není možnost provést slévání záznamů přímo v databázi a je třeba tuto operaci provádět programově.

4.2.6.3 WHERE <conditions>

Asi nejvíce rozšíření přináší jazyk TSQL2 do části `WHERE` pro omezení výběru dotazu. Nově je možné pracovat v podmínkách s hodnotami časů platnosti i transakce a je navíc přidáno i několik nových predikátů vztahujících se právě k časovým údajům.

Stejně jako v případě výběru hodnot je možné v podmínkách používat klauzule `VALID(tabulka)` a `TRANSACTION(tabulka)`, kdy jsou tyto výrazy nahrazeny časy platnosti a transakce pro daný záznam.

Dále je možné používat v podmínkách novou konstrukci `PERIOD [od – do]`, která představuje časový úsek s danými hranicemi. Tento časový úsek potom může být porovnáván s časy platnosti nebo transakce a lze tak vytvářet další způsoby omezení výsledných záznamů.

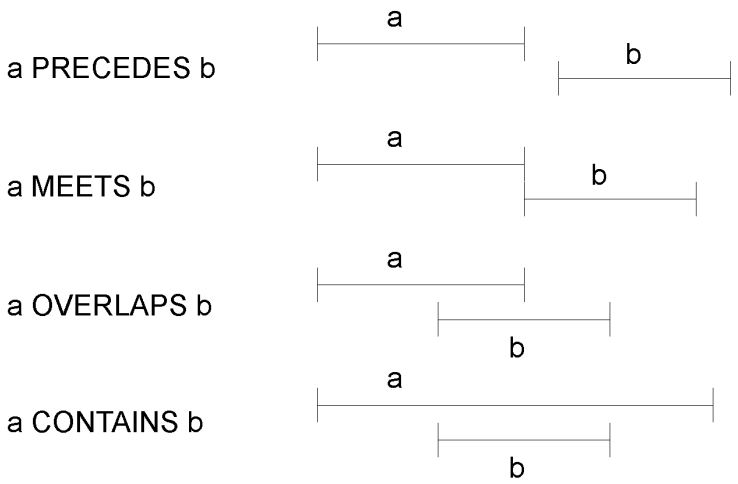
Při překladu jednotlivých podmínek a výrazů provádí překladač nejprve ověření, zda obsahuje daný výraz nějakou temporální složku. Pokud se výraz skládá pouze z jednoduchých netemporálních položek, není třeba nic překládat a může se výraz použít tak jak je. Pokud ale obsahuje výraz temporální položky, je třeba převést tyto položky na jejich SQL reprezentace.

V takovém případě překladač výraz rozloží na levou a pravou stranu a každou z nich postupně přeloží stejným způsobem. Pokud se na levé straně nachází například výraz `VALID(tabulka)`, překladač jej přeloží na dvě hodnoty reprezentující počátek a konec času platnosti daného záznamu v dané tabulce. Podobně výraz `PERIOD [od - do]` se přeloží jako začátek a konec intervalu.

Potom co překladač takto převede levou i pravou stranu, provede s výsledky operaci podle typu výrazu. Například pro typ výrazu `PRECEDES`, který říká, že levá strana musí předcházet pravé straně, je výsledný výraz ve tvaru

```
konec_intervalu_leve_strany < zacatek_intervalu_prave_strany.
```

Výraz typu PRECEDES je jeden z nově zavedených predikátů pracujících s intervaly. Jednotlivé nové predikáty jsou potom uvedeny v následujícím obrázku.



Obrázek 10 - Nové predikáty v TSQL2

- **a PRECEDES b** – vrátí True pokud interval **a** předchází intervalu **b**
- **a MEETS b –** vrátí True pokud interval **b** navazuje na interval **a**
- **a OVERLAPS b** – vrátí True pokud se intervaly **a** a **b** překrývají
- **a CONTAINS b** – vrátí True pokud interval **a** kompletně obsahuje interval **b**

Kromě uvedených možností výrazů v podmínkách umožňuje samozřejmě TSQL2 použít ve výrazu i poddotaz typu `SELECT`. Taková dotaz může vypadat například následovně.

```
SELECT jmeno FROM Zamestnanci Z1 WHERE id = ANY (
  SELECT id FROM Zamestnanci Z2 WHERE ...)
```

Pokud tedy překladač narazí ve výrazu na poddotaz, musí tento poddotaz nejprve přeložit. To provede jednoduše vytvořením nového překladače pro dotazy typu `SELECT`, který použije na

poddotaz. Takto je tedy možné, že se bude překlad provádět rekurzivně několikrát, podle struktury dotazu.

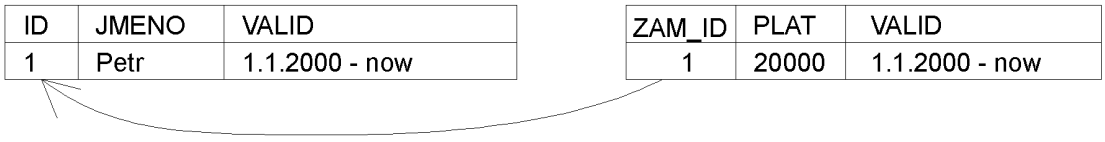
Pokud má některá z tabulek, ze kterých se budou vybírat data, nastavenu podporu času transakce, musí se ještě do omezujících podmínek dotazu přidat pro každou takovou tabulku podmínka, aby se vybíraly pouze záznamy, které mají platný čas transakce.

4.3 Referenční integrita v temporální databázi

Největší komplikací kterou temporální data přinášejí do relační databáze je problém udržování referenční integrity. V běžné relační databázi funguje referenční integrita pomocí cizích klíčů relativně jednoduše. Pokud jeden záznam tabulky odkazuje pomocí cizího klíče do jiné tabulky, odkazuje na jeden konkrétní záznam. Pokud je tento záznam smazán, mohou být jednoduše smazány i závislé záznamy. Protože všechna data v databázi se považují za aktuálně platná, nemusí se při referenční integritě řešit platnost těchto vazeb.

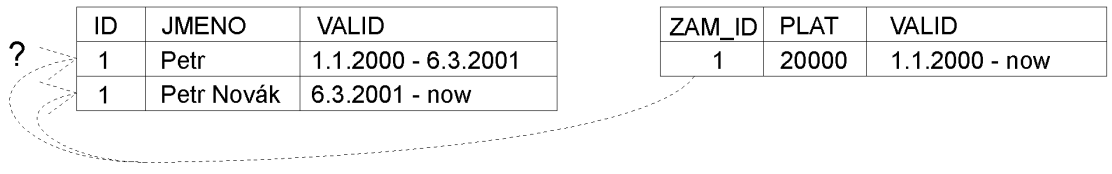
V temporální databázi je ovšem situace zcela odlišná. Na rozdíl od relační databáze může mít jeden a ten samý záznam v databázi několik různých verzí, každou platnou v jiný časový okamžik nebo úsek. Pro řešení referenční integrity to přináší komplikaci v podobě možné nejednoznačnosti odkazů mezi daty.

Představme si dvě tabulky s podporou časů platnosti. V těchto tabulkách jsou v každé po jednom záznamu a tyto záznamy jsou navzájem svázány cizím klíčem.



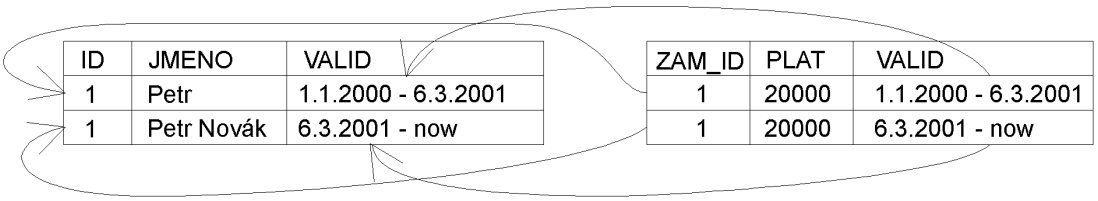
Obrázek 11 - Cizí klíče pro dva záznamy

Dokud jsou oba záznamy platné po stejnou dobu a jsou přítomné v tabulkách pouze jednou, je vše v pořádku. Nyní se ale provede aktualizace dat v jednom ze záznamů a tato operace povede k rozdělení záznamu na dva, každý s jiným časem platnosti. V tuto chvíli je ovšem problém, jak svázat tyto dva záznamy s druhou tabulkou, kde je záznam pouze jeden.



Obrázek 12 - Nejednoznačnost odkazů

Řešením je vzít záznam z druhé tabulky, tento záznam rozdělit na dva záznamy podle prvního záznamu a nastavit odkazy mezi odpovídajícími variantami. To ovšem přináší nový problém. Primární klíč, podle kterého se původně odkazovalo, je teď v tabulce dvakrát. Podle čeho se tedy má odkazovat nyní? V tomto případě se musí použít složený primární klíč, který bude obsahovat i čas platnosti.



ID	JMENO	VALID
1	Petr	1.1.2000 - 6.3.2001
1	Petr Novák	6.3.2001 - now

ZAM_ID	PLAT	VALID
1	20000	1.1.2000 - 6.3.2001
1	20000	6.3.2001 - now

Obrázek 13 - Správné odkazy na cizí klíče po úpravě

Jak je vidět, i taková jednoduchá operace, jako je aktualizace hodnoty jednoho záznamu, vede v případě temporálních dat k řešení složitého problému. V následujících podkapitolách bude tedy rozebráno řešení problému referenční integrity pro různé situace.

4.3.1 Snímková tabulka – snímková tabulka

Pokud jsou obě tabulky snímkové bez podpory temporálních dat, je situace jednoduchá. Hlídání referenční integrity může provádět samotná relační databáze, protože data v tabulkách jsou jednoznačná a stále platná.

4.3.2 Snímková tabulka – stavová tabulka

Pokud máme odkazy z tabulky s časy platnosti (odkazující) do snímkové tabulky (odkazovaná), řešíme dvě varianty. V prvním případě se jedná o úpravu nebo smazání dat ze snímkové tabulky.

V případě úpravy dat se nic neděje a referenční integrita je neporušena, samozřejmě pokud se neupravuje hodnota primárního klíče. V takovém případě se musejí aktualizovat odkazy z druhé tabulky.

V případě smazání dat ze snímkové tabulky musí ovšem dojít i ke smazání dat ve stavové tabulce. Data by se ovšem neměla mazat fyzicky, ale měl by být pouze ukončen jejich čas platnosti. Co ale udělat s odkazem na již neexistující záznam ve snímkové tabulce? Pokud ponecháme odkaz tak jak je, bude odkazovat na neexistující klíč a poruší tedy referenční integritu. Pokud odkaz zrušíme, bude referenční integrita v pořádku, ale z hlediska historie záznamu se ztratí informace, že v době svojí platnosti někam odkazoval.

Druhá varianta je úprava nebo smazání dat ze stavové tabulky. V takovém případě může dojít k rozdělení záznamu na více verzí. Každá tato verze ale může normálně odkazovat do snímkové tabulky, takže referenční integrita je v pořádku.

Pro spolehlivé řešení výše uvedených problémů by tedy nemělo být možné odkazovat mezi snímkovou a stavovou tabulkou, ale snímková tabulka by měla být změněna na stavovou. Potom by se mohly souměrně upravovat časy platnosti v obou tabulkách a vazby by zůstaly v pořádku.

Pokud máme odkazy ze snímkové tabulky (odkazující) do tabulky s časy platnosti (odkazovaná), řešíme opačné problémy. Co dělat, když se ve stavové tabulce upraví záznam a rozdělí se na dva? Logicky by se měl odkaz ze snímkové tabulky upravit, aby odkazoval na aktuálně platný záznam. Jenže snímková tabulka nebere v úvahu časové hledisko, takže by měl odkaz správně ukazovat na všechny varianty ve stavové tabulce. To ovšem není v případě jednoho záznamu možné. Opět by tedy nemělo být možné odkazovat ze snímkové do stavové tabulky.

4.3.3 Snímková tabulka – transakční tabulka

Dvojice snímkové a transakční tabulky řeší prakticky stejný problém jako dvojice snímkové a stavové tabulky. Opět dochází ve všech případech buď k porušení referenční integrity. Nebo k porušení logických vazeb.

4.3.4 Snímková tabulka – bitemporální tabulka

V případě snímkové a bitemporální tabulky se situace komplikuje ještě o druhou časovou osu v bitemporální tabulce a opět není spolehlivě řešitelná.

4.3.5 Stavová tabulka – stavová tabulka

V případě dvojice stavových tabulek je situace o poznání příznivější. V obou tabulkách je možné mít více verzí záznamů pro různé časy platnosti. Pokud se tedy některý záznam upraví nebo smaže a upraví se tak časy platnosti, mohou se stejným způsobem upravit i závislé záznamy.

Při aktualizaci se záznam rozdělí na dva nebo více úseků platnosti, tak se rozdělí i odkazované záznamy. V případě smazání odkazujícího záznamu se pouze ukončí jeho platnost a tak se ukončí platnost i odkazovaných záznamů. Stačí tedy "zrcadlit" změny v časech platnosti mezi záznamy.

Vzhledem k tomu, že řádek ve stavové tabulce je identifikován primárním klíčem, který obsahuje i čas platnosti, je nutné provádět odkazování pomocí složeného cizího klíče, který bude také zahrnovat cílový čas platnosti záznamu.

4.3.6 Stavová tabulka – bitemporální tabulka

Pokud se podíváme na referenční integritu mezi stavovou a bitemporální tabulkou, dostaneme se k řešení podobného problému, jako v případě dvojice snímkové a stavové tabulky. Opět je v tomto případě na vině rozdíl v počtu časových os pro obě tabulky. Stavová tabulka obsahuje pouze jednu

časovou osu, zatímco bitemporální tabulka má časové osy dvě. Jak jsme vysvětlili v předchozí podkapitole, není velký problém udržovat referenční integritu mezi časy platnosti.

V případě bitemporální tabulky je zde ovšem navíc čas transakce. Díky tomu může být pro jeden záznam s jedním intervalem času platnosti přítomno v tabulce více verzí s různými časy transakce. Tato situace vede opět k problému, jak vést odkazy na tyto různé verze nebo z nich, aby byla zachována referenční integrita i logické vztahy mezi záznamy.

Ideální řešení opět neexistuje a odkazy mezi stavovou a bitemporální tabulkou by tedy správně neměly být možné.

4.3.7 Bitemporální tabulka – bitemporální tabulka

Pro dvě bitemporální tabulky je situace podobná jako pro dvě stavové. Obě tabulky v tomto případě obsahují dvě časové osy a všechny operace provedené na jedné tabulce je tak možné zrcadlit i na druhé tabulce. Tím je možné udržovat referenční integritu při změnách v jedné i v druhé tabulce.

4.3.8 Výsledek návrhu

Vzhledem k výše popsaným skutečnostem jsem se rozhodl v této fázi vývoje podporu pro referenční integritu do produktu nezahrnout. Teoreticky by bylo možné zavést podporu referenční integrity pouze pro shodné typy tabulek, ale i to by byl relativně komplikovaný úkol a vzhledem ke složitosti ostatních částí implementace považuji za rozumné, raději vytvořit produkt s plně implementovanými ostatními funkcemi, než zahrnout pouze částečnou podporu referenční integrity potenciálně na úkor ostatní funkčnosti.

5 Implementace

Tato kapitola obsahuje hlavní část práce a tou je popis samotné implementace interpretu jazyka TSQL2. V první části kapitoly je čtenář seznámen s použitým prostředím a jazykem implementace. Dále následuje popis parseru jazyka TSQL2, který je použit pro zpracování vstupních příkazů před jejich překladem pomocí samotné aplikace. Ve třetí části kapitoly je specifikována výsledná gramatika TSQL2 implementovaná v tomto produktu. Dále je rozebrán způsob překladu příkazů pomocí dílčích překladačů.

5.1 Jazyk implementace a použité databáze

Pro implementaci překladače je použit jazyk Java verze 1.6. Jazyk Java byl zvolen hlavně z důvodu široké podpory napříč operačními systémy a také z důvodu široké podpory databází za použití standardizovaného rozhraní JDBC. Vytvořený interpret tedy implementuje právě rozhraní JDBC verze 4.0. Je tedy možné nahradit v existujících aplikacích pouze s drobnými úpravami stávající adaptér JDBC za tento interpret a přidat tak do aplikace podporu temporálních databází.

Při vývoji interpretu byla primárně použita databáze Oracle Database 10g Express Edition Release 10.2.0.1.0. Mimo této databáze byl interpret testován i na databázi MySQL 5.0.

5.2 Parser jazyka TSQL2

Hlavním úkolem interpretu vytvořeného v této práci je překlad příkazů jazyka TSQL2 na standardní SQL příkazy. Vstupem interpretu jsou pouze řetězce příkazů TSQL2 a tyto řetězce není možné ve většině případů přeložit do SQL za použití jednoduchých transformací, jako jsou nahrazení řetězce a podobné, přímo dostupné operace v jazyce Java. Naopak je ve většině případů nutné získat sémantický význam řetězce příkazu a podle tohoto významu vytvořit i několik jiných příkazů SQL.

Vzhledem k výše uvedené skutečnosti je nutné provést nejprve syntaktickou analýzu vstupních řetězců a vytvořit syntaktický strom reprezentující daný příkaz. Tento strom navíc musí být ve vhodném tvaru, který je možné jednoduše zpracovávat programově v jazyce Java.

5.2.1 Možnost vlastní implementace

První možností řešení bylo pustit se do psaní vlastního parseru jazyka TSQL2 od základů. Výhodou tohoto přístupu by mohl být potenciálně více specializovaný a optimalizovaný parser vyvinutý s ohledem na specifika jazyka TSQL2. Následují ovšem už pouze nevýhody. Vlastní implementace by byla velice náročná na čas i prostředky. Navíc bez předchozí praxe ve vývoji podobných nástrojů

by byl výsledek pravděpodobně pomalý a zbytečně složitý. Tuto možnost jsem se tedy rozhodl opustit a místo toho jsem zvolil variantu druhou.

5.2.2 JavaCC a JJTree

Druhou možností bylo vybrat některý z mnoha existujících nástrojů pro generování překladačů. Z existujících nástrojů pro jazyk Java můžeme zmínit například Grammatica, Byacc/J, COCO/R, ANTLR nebo JavaCC. Pro použití v tomto projektu jsem zvolil právě posledně jmenovaný nástroj JavaCC (Java Compiler Compiler).

JavaCC [JavaCC] je open-source nástroj, který ze zadané LL(k) gramatiky vygeneruje parser pro danou gramatiku. Výsledný parser je čistě v jazyce Java a je tedy možné jej přímo použít v aplikaci psané v jazyce Java. Součástí balíku JavaCC je navíc nástroj JJTree, který slouží pro jednoduché vytváření generátoru syntaktických stromů podle zadané gramatiky.

Vstupem těchto nástrojů je soubor s definicí požadované gramatiky. Syntaxe pro zápis gramatiky je velmi podobná syntaxi jazyka Java, pouze přidává některé specifické prvky nutné pro podrobnější popis parseru. V souboru je třeba nadefinovat vlastní třídu parseru, která potom bude přeložena na běžnou Java třídu. Dále se definují jednotlivé neterminály a terminály, ze kterých se gramatika skládá.

Tento soubor s gramatikou se poté předzpracuje pomocí nástroje JJTree, který do gramatiky přidá prvky nutné pro vygenerování syntaktického stromu. Následně se tento mezivýstup zpracuje již přímo nástrojem JavaCC a výstupem této operace je několik tříd v jazyce Java, které reprezentují požadovaný parser.

Získané třídy stačí pouze vložit do projektu a je možné parser používat přímo v aplikaci. Použití je velice jednoduché. Stačí pouze předat parseru vstupní řetězec, ze kterého chceme vygenerovat syntaktický strom. Pokud vstup neodpovídá zadané gramatice, vyhodí parser výjimku s popisem chyby. Pokud je vstup v pořádku vrátí praser kořen syntaktického stromu. Z tohoto kořene je potom možné celý strom postupně procházet a zpracovávat. Kořen stromu je také objekt třídy v jazyce Java, vše je tedy řešeno standardními nástroji jazyka.

5.2.2.1 Příklad použití parseru v kódu

```
import tsql2lib.TSQL2Adapter;
import tsql2lib.parser.SimpleNode;

...
/**
 * Vrati strukturu stromu zadaného korenem jako řetězec.
 *
 * @param node Korenovy uzel ke zpracovani
 * @param level Uroven zanoreni
 * @return Retezec reprezentujici strukturu stromu.
 */
private String eval(SimpleNode node, int level) {
  int numChildren = node.jjtGetNumChildren();
  // predsadime mezery podle zanoreni
  String prefix = "";
  for (int i = 0; i < level; i++) {
    prefix += "  ";
  }

  String result = prefix + node.toString() + "\n";
  if (numChildren > 0) {
    for (int i = 0; i < numChildren; i++) {
      result += prefix + eval((SimpleNode)node.jjtGetChild(i), level+1);
    }
  }

  return result;
}

...

// vytvoříme nový parser
TSQL2Parser parser = new TSQL2Parser();
// předáme vstupní řetězec a získáme kořen stromu
SimpleNode treeRoot = parser.parse("SELECT id, jmeno FROM studenti WHERE
obor = 'informační systémy'");
// vypíšeme strom
if ((treeRoot != null) && (treeRoot.jjtGetNumChildren() > 0)) {
  System.out.print(eval((SimpleNode) treeRoot.jjtGetChild(0), 0));
}
```

Výše uvedený kus kódu vytvoří parser za použití třídy `TSQL2Parser` vygenerované pomocí JavaCC. Poté předá ke zpracování řetězec a pro získaný kořen syntaktického stromu zavolá metodu `eval()`, která vypíše strukturu syntaktického stromu.

Výsledný strom pro výše uvedený příkaz potom vypadá takto:

```
QueryStatement
  SelectStatement
      SelectWithoutOrder
          SelectList
              SelectItem
                  TableColumn
                      ObjectName
                          :id
              SelectItem
                  TableColumn
                      ObjectName
                          :jmeno
          FromClause
              FromItem
                  TableReference
                      ObjectName
                          :studenti
          WhereClause
              SQLExpression
                  SQLRelopExpression
                      SQLPrimaryExpression
                          TableColumn
                              ObjectName
                                  :obor
                      Relop
                          :=
                      SQLPrimaryExpression
                          TableColumn
                              ObjectName
                                  :'informační systémy'
```

5.3 Gramatika TSQL2

Jazyk TSQL2 je rozšířením jazyka SQL-92. Gramatika jazyka TSQL2 tedy vychází z gramatiky SQL a pouze upravuje nebo rozšiřuje tuto základní gramatiku.

Jako základ v této práci byla použita gramatika SQL [SQLGRAMMAR] pro JJTree. Do této gramatiky bylo přidáno rozšíření jazyka TSQL2 a byly mírně upraveny některé již přítomné definice z důvodu lepší struktury výsledného syntaktického stromu.

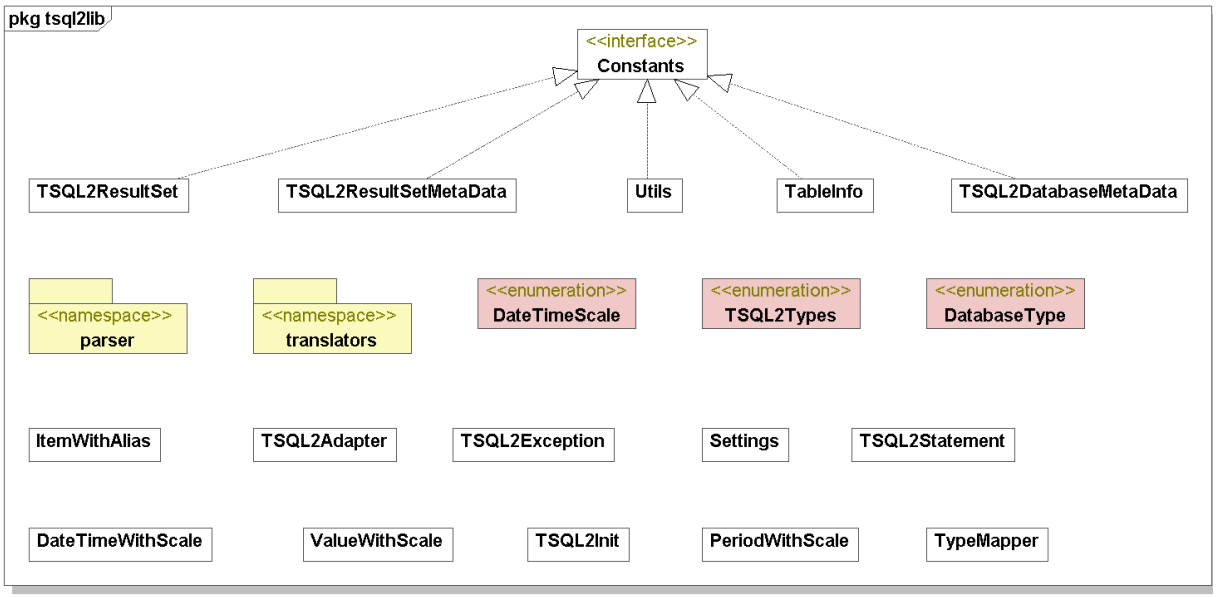
Kompletní výsledná gramatika pro překladač je uvedena v příloze 1.

5.4 Části překladače

Implementace překladače TSQL2 se skládá z tří hlavních částí – balíčků – v jazyce Java. Jedná se o **tsql2lib**, **tsql2lib.parser** a **tsql2lib.tranlators**. Každý z těchto balíčků sdružuje jednu z logických částí překladače. Tyto části jsou implementace JDBC API, parser jazyka TSQL2 a překladač příkazů TSQL2 na SQL.

5.4.1 Balíček tsql2lib

Balíček **tsql2** je hlavní částí překladače určenou pro použití v aplikacích. V tomto balíčku jsou implementovány třídy rozhraní JDBC.



Obrázek 14 - Struktura balíčku tsql2lib

5.4.1.1 TSQL2Adapter

Třída `TSQL2Adapter` je hlavní třída interpretu, která implementuje rozhraní `Connection` a slouží pro připojení k běžné relační databázi. Při použití této třídy je tak k relační databázi přidána podpora temporálních dat pomocí překladů příkazů TSQL2 na SQL.

Při vytvoření instance třídy `TSQL2Adapter` je nutné poskytnout třídě instanci rozhraní `Connection`, která představuje běžné připojení k dané databázi. TSQL2Adapter toto připojení převezme a obalí jej vlastními implementacemi metod rozhraní `Connection`. Následně je možné instanci třídy TSQL2Adapter používat jako běžné připojení JDBC pro přístup k databázi.

Příklad připojení k databázi Oracle 10g Express Edition

```
import oracle.jdbc.pool.OracleDataSource;
import tsql2lib.TSQL2Adapter;
...
String url = "jdbc:oracle:thin:tsql2/tsql2@//192.168.1.8:1521/xexdb";
OracleDataSource ods = new OracleDataSource();
ods.setURL(url);
Connection connection = new TSQL2Adapter(ods.getConnection());
```

5.4.1.2 TSQL2Statement

Třída reprezentující příkaz v jazyce TSQL2 je třída `TSQL2Statement`. Podle modelu JDBC se třídy příkazů získávají voláním metody `Connection.createStatement()`. Instanci třídy `TSQL2Statement` je tedy možné získat voláním této metody na objekt třídy `TSQL2Adapter`.

`TSQL2Statement` implementuje rozhraní `Statement` a je tedy transparentně použitelná jako běžná třída příkazu JDBC. Všechny příkazy předané k vykonání pomocí této třídy jsou transparentně přeloženy z jazyka TSQL2 do SQL a vykonány pomocí interního objektu rozhraní `Statement`. Tento interní objekt získá třída při svém vytváření od třídy JDBC `Connection` obsažené v instanci TSQL2Adapter.

```
// predpokladame, ze connection je instanci TSQL2Adapter
Statement statement = connection.createStatement();
```

5.4.1.3 TSQL2ResultSet

Po vykonání příkazu na databázi pomocí instance rozhraní `Statement` jsou podle JDBC vráceny případné výsledky v instanci rozhraní `ResutSet`. Protože existující implementace rozhraní `ResultSet` nejsou uzpůsobeny k práci s výsledky z temporální databáze, je vytvořena třída `TSQL2ResultSet`, která implementuje právě rozhraní `ResultSet` a je použita pro reprezentaci výsledků z TSQL2 dotazů.

Databázové tabulky pro uložení temporálních dat jsou v relační databázi doplněny o některé speciální atributy, které slouží pro uložení potřebných doplňkových dat k temporálním datům, ale tyto atributy mají zůstat před uživatelem skryty. Navíc je třeba v některých případech z těchto atributů vytvořit pro uživatele ve skutečnosti neexistující atributy, jako jsou intervaly časů platnosti a podobně.

Pokud by se použila implementace `ResultSet` dodávaná s použitým rozhraním pro danou databázi, nebylo by možné kontrolovat vrácené výsledky, ani je nijak upravovat. Proto je vytvořena třída `TSQL2ResultSet`, která výsledky vrácené z databáze upraví do výsledné podoby předložené uživateli.

Tyto úpravy zahrnují skrývání systémových atributů použitých k uložení doplňkových dat k temporálním záznamům, převod interní reprezentace časů platnosti na uživatelsky použitelný formát a podobně.

Protože ale potřebuje třída sama získat data z databáze nějakým univerzálním způsobem, obsahuje pro interní potřeby objekt rozhraní `ResultSet` pro konkrétní databázi, který získá při vytvoření z objektu `Statement`. Tento interní objekt kompletně obaluje a využívá jeho metod pro práci s výsledky dotazu.

```
// predpokladame, ze objekt statement je instanci TSQL2Statement
ResultSet results = statement.executeQuery("SELECT * FROM Zamestnanci");
while (results.next()) {
  // zpracuj vysledek
  ...
}
```

5.4.1.4 TSQL2ResultSetMetaData

Při práci s výsledky dotazů na databázi je v aplikaci často nutné zjistit počty a typy sloupců. Navíc musí i samotný objekt `ResultSet` při přístupu k získaným datům provádět různé kontroly a převody, jako je převod názvu sloupce na index pro získání jeho hodnoty.

Jak bylo zmíněno u popisu `TSQL2ResultSet`, obsahují tabulky temporální databáze navíc různé pomocné atributy. Pokud by se použila implementace `ResultSetMetaData` dodávaná s databází, mohl by uživatel získávat informace o těchto atributech a navíc by tyto atributy ovlivňovaly hodnoty, jako jsou počty vrácených atributů.

Je proto vytvořena třída `TSQL2ResultSetMetaData`, která implementuje rozhraní `ResultSetMetaData`. Pokud požaduje aplikace nebo uživatel informace o výsledcích dotazu, jako je počet sloupců výsledku nebo typy sloupců, použije k tomu právě instanci této třídy, kterou získá pomocí standardního postupu voláním metody `ResultSet.getMedaData()` na výsledek dotazu.

Hlavní úloho této třídy je úprava skutečných metadat výsledku. Je třeba upravit počet vrácených sloupců ve výsledku, aby se skryly pomocné atributy temporálních tabulek. Dále provádí třída přemapování indexů a názvů sloupců výsledku pro zabránění přístupu k pomocným atributům.

Stejně jako výše uvedené třídy používá tato třída interně původní objekt `ResultSetMetaData` pro danou databázi, pomocí kterého přistupuje k originálním metadatům.

```
// predpokladame, ze results je instanci TSQL2ResultSet
ResultSetMetaData meta = results.getMetaData();
// vypis nazvy a typy sloupcu
for (int i = 0; i < meta.getColumnCount(); i++) {
  System.out.print(meta.getColumnName(i) + " – " + meta.getColumnType(i));
}
```

5.4.1.5 TSQL2DatabaseMetaData

Velmi často je při překladu dotazů TSQL2 nutné získávat informace o tabulkách v databázi. Jedná se hlavně o podpory času platnosti a transakce, sloupce typu SURROGATE a další temporální údaje.

Třída `TSQL2DatabaseMetaData` poskytuje rozhraní pro získání těchto informací jednoduchým a jednotným způsobem kdekoliv v aplikaci. Implementuje vzor Singleton, takže je všude k dispozici identická instance třídy, nezávisle na ostatních částech aplikace.

Jedinou metodou této třídy je metoda `getMetaData(String table_name)`, která vrací jako výsledek instanci třídy `TableInfo`. Třída `TableInfo` slučuje všechny potřebné informace o tabulce a slouží tak k jednotnému přístupu k těmto informacím.

Protože k volání metody `getMetaData()` pro konkrétní tabulku dochází vždy, když se provádí s danou tabulkou nějaká operace nebo se nad tabulkou provádí dotaz, je tato metoda místem, ve kterém probíhá zároveň případné odmazávání záznamů (vacuuming). Při každém volání metody se provede kontrola, zda tabulka podporuje čas transakce. Tato kontrola nepřináší žádnou režii navíc, protože tuto informaci stejně musí metoda vrátit volajícímu kódu, který ji potřebuje. Pokud tedy tabulka podporuje čas transakce, vezmou se časové údaje pro vacuuming a provede se odpovídající příkaz ke smazání starých neplatných záznamů. Jak je popsáno v kapitole 4.2.1.1, může být časový údaj pro vacuuming nastaven absolutně nebo relativně. Podle toho se tedy případně pokaždé dopočítává reálný čas pro odmazávání.

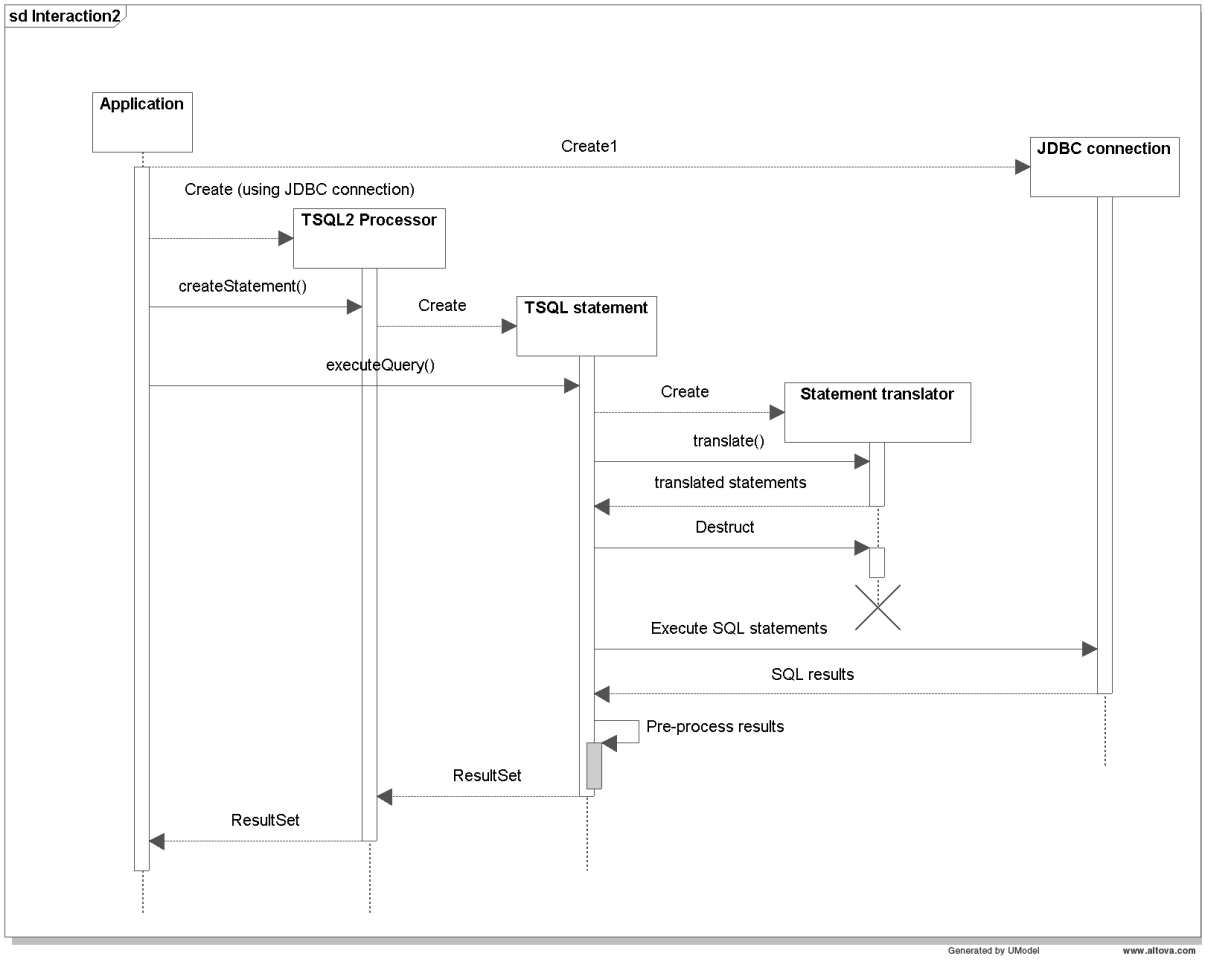
5.4.1.6 Další třídy balíčku

Mimo výše popsaných hlavních tříd jsou v balíčku obsaženy i další pomocné třídy. Jedná se o třídy pro nastavení překladače, inicializaci, překlad datových typů a podobně.

- **Init** – Tato třída slouží k inicializaci nastavení prostředí překladače před jeho použitím. Zde se provádí rozpoznání použitého databázového systému a nastavení systému na tento konkrétní typ databáze. Dále se ověří, zda je daná databáze již připravena na podporu temporálních dat (obsahuje pomocné tabulky) a pokud není, jsou automaticky vytvořeny potřebné pomocné tabulky pro překladač.
- **Settings** – V této třídě se nachází definice názvů pomocných tabulek a sloupců. Při inicializaci systému jsou v této třídě také nastaveny konkrétní hodnoty lišící se podle typu použité databáze. Jedná se například o znaky pro escape identifikátorů nebo řetězců.

- **TypeMapper** – V této třídě je definováno mapování obecných datových typů používaných překladačem například pro vytvoření pomocných tabulek a sloupců. Jedná se o typy pro 32-bitové celé číslo, 64-bitové celé číslo, řetězec s proměnnou délkou a další. Tato třída obsahuje pole mapující tyto obecné typy na konkrétní datové typy podporovaných databázových systémů. Překladač potom místo použití konkrétního typu používá tuto třídu a obecné typy, které se potom přeloží pro daný databázový systém.
- **Utils** – Zde se nachází soubor statických metod pro obecné použití. Jedná se o metody pro escapování identifikátorů a řetězců nebo převody data a času na řetězcovou reprezentaci a zpět.
- **Constants** – Jedná se o rozhraní obsahující pouze definice konstant používaných v aplikaci. Většina tříd potom implementuje toto rozhraní a má tak přímý přístup ke všem potřebným konstantám.

5.4.1.7 Zjednodušená ukázka vykonání dotazu



Obrázek 15 – Zjednodušený cyklus práce s dotazem a výsledky

5.4.2 Balíček tsql2lib.parser

Balíček **tsql2lib.parser** obsahuje hlavně vygenerované třídy parseru jazyka TSQL2 pomocí nástroje JavaCC. Vygenerovaný parser ovšem není ještě v ideální podobě pro přímé použití v aplikaci. Například neobsahuje přímo možnost provést analýzu řetězce uloženého jako `String`, takže není možné zapsat například

```
parser.parse("SELECT * FROM Zamestnanci");
```

, ale je nutné převést řetězec na objekt rozhraní Reader nebo InputStream. Všude v aplikaci by tak bylo nutné psát

```
parser.parse(new StringReader("SELECT * FROM Zamestnanci"));
```

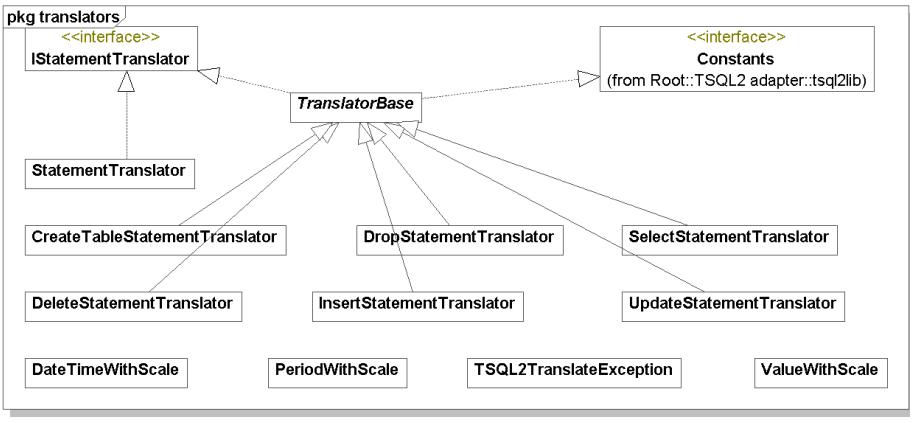
Dále je zvlášť pro potřeby ladění aplikací vhodné poskytnout například možnost výpisu syntaktického stromu. Z těchto důvodů se nepoužívá parser v programovém kódu přímo, ale je vytvořena třída `TSQL2ParserAdapter`, která slouží jako rozhraní k praseru.

Tato třída obaluje parser a přidává možnost přímého zpracování řetězců a výpisy struktury syntaktického stromu.

Zbylé třídy obsažené v tomto balíčku jsou generované pomocí JavaCC a nebudou zde proto blíže popisovány.

5.4.3 Balíček tsql2lib.translators

Nejsložitější částí interpretu jazyka TSQL2 je překlad příkazů TSQL2 na čisté SQL. K tomuto úkolu slouží právě třídy obsažené v balíčku **tsql2lib.translators**.



Obrázek 16 - Struktura balíčku tsql2lib.translators

Při návrhu interpretu bylo nutné vyřešit problém, jakým způsobem provádět překlad příkazů, aby byl výsledný programový kód dobře strukturovaný a udržovatelný, ale zároveň logicky uspořádaný a snadno použitelný. Proto jsem zvolil pro překlad příkazů řešení pomocí tzv. **Statement translators**.

Jedná se o samostatné třídy pro překlad různých typů příkazů TSQL2, kdy každá takováto třída je určena k překladu konkrétního typu příkazu. Byly tak vytvořeny třídy **SelectStatementTranslator, InsertStatementTranslator, DeleteStatementTranslator, ...**

Všechny tyto třídy jsou odvozeny od společné nadtřídy `TranslatorBase`, která obsahuje pomocné metody a vlastnosti využívané ve všech překladačích.

5.4.3.1 StatementTranslator

Vstupní třídou pro překlad příkazů je třída `SatementTranslator`. Tato třída není jako jediná skutečným překladačem, ale slouží pouze jako rozhraní mezi vyšší vrstvou interpretu a samotnými překladači.

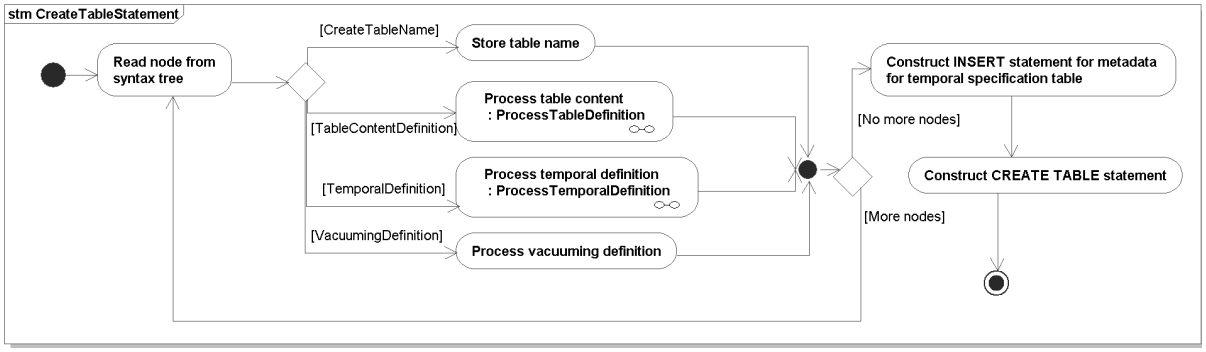
Všechny příkazy určené k překladu jsou z interpretu předány ke zpracování třídě `StatementTranslator`. Tato třída pouze rozpozná, o jaký typ příkazu se jedná a předá tento příkaz k překladu konkrétnímu překladači pro konkrétní typ příkazu. Potom převezme výsledky překladu od daného překladače a tyto výsledky vrátí interpretu jako výsledek překladu.

Příklad použití translatoru v kódu

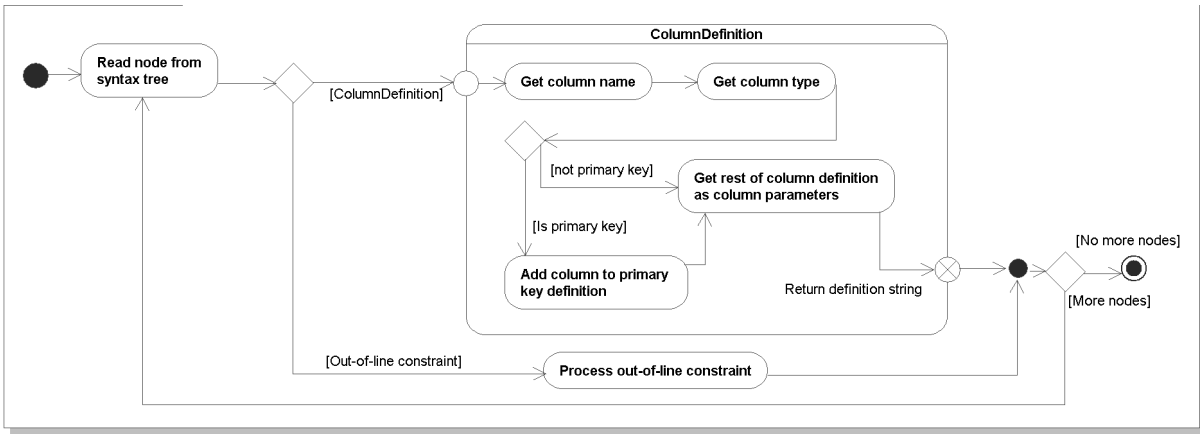
```
// connection je instance TSQL2Adapter
StatementTranslator  translator = new StatementTranslator(connection);
// parser je instanci TSQL2Parser
String[] statements = translator.translate(parser.parse(query));
for (String statement : statements) {
  stmt.execute(statement);
}
// uvolneni docasnych prostredku
translator.clear();
```

5.4.3.2 CreateTableStatementTranslator

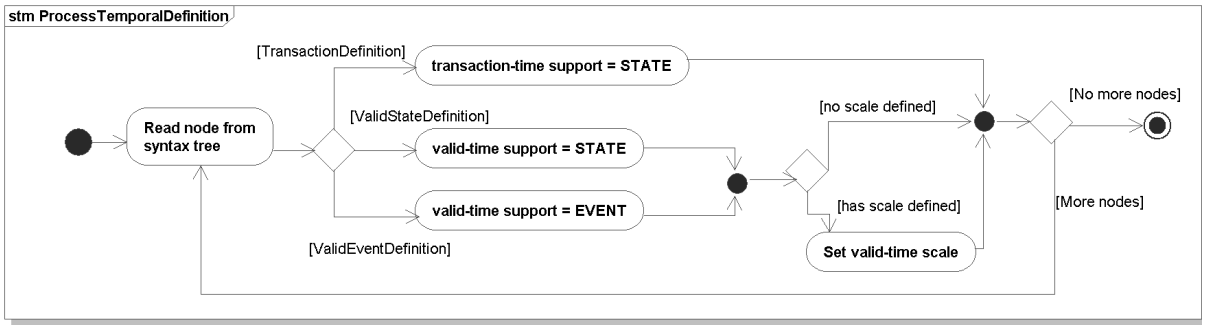
Pro překlad TSQL2 příkazů typu CREATE TABLE slouží třída překladače s názvem `CreateTableStatementTranslator`. Tato třída vezme na vstupu syntaktický strom příkazu pro vytvoření tabulky, tento příkaz analyzuje a vygeneruje odpovídající SQL příkazy pro vytvoření relační tabulky simulující podporu temporálních dat.



Obrázek 17 - Schéma činnosti překladu CREATE TABLE



Obrázek 18 - Schéma zpracování definice sloupců tabulky



Obrázek 19 - Schéma zpracování definice temporální podpory tabulky

Nejprve se přečte z příkazu název vytvářené tabulky. Poté se postupně čtou definice pro jednotlivé sloupce tabulky, a pokud se narazí na primární klíč, přidá se tento sloupec do pomocného pole pro vygenerování primárního klíče tabulky. Pokud následuje definice temporální podpory, analyzuje se a podle zadání se nastaví podpora pro čas platnosti a čas transakce.

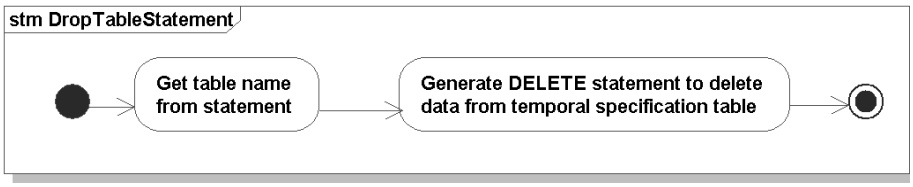
Po zpracování celého příkazu následuje generování výsledných SQL příkazů. Nejprve se sestaví SQL příkaz pro vytvoření záznamu o tabulce do tabulky metadat, která se používá po uložení informací o temporálních tabulkách. Je třeba uložit název tabulky, úroveň podpory času platnosti a transakce, měřítko času platnosti a další nutné informace.

Následně se vygeneruje vlastní SQL příkaz pro vytvoření tabulky. Podle požadované temporální podpory je třeba k uživatelsky definovaným sloupcům přidat ještě sloupce pomocné pro uložení temporálních časových informací. Zároveň je nutné případně modifikovat definici primárního klíče, aby obsahovala i přidané pomocné sloupce.

Výsledná dvojice příkazů je vrácena jako výsledek překladu. Jeden příkaz v jazyce TSQL2 pro vytvoření tabulky je tedy vždy přeložen na dva příkazy SQL.

5.4.3.3 DropStatementTranslator

Pro překlad TSQL2 příkazů typu DROP TABLE slouží třída `DropStatementTranslator`. Tato třída vezme na vstupu syntaktický strom příkazu pro smazání tabulky, tento příkaz analyzuje a vygeneruje odpovídající SQL příkazy pro smazání relační tabulky.



Obrázek 20 - Schéma činnosti překladu DROP TABLE

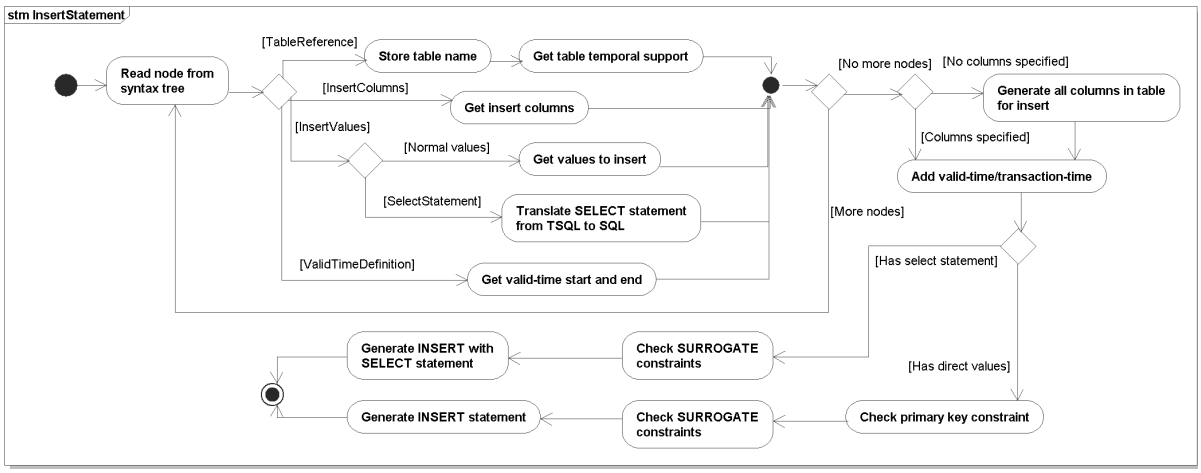
Z příkazu se přečte název tabulky pro smazání. Poté se vygeneruje příkaz pro smazání záznamu o tabulce z tabulky metadat. Každá tabulka v temporální databázi totiž obsahuje záznam v tabulce metadat a při smazání tabulky musí být zároveň smazán tento záznam.

Samotný příkaz pro smazání dané tabulky se nemusí nijak upravovat a použije se, jak byl zadán.

Jako výsledek je tedy vrácena dvojice příkazů, jeden pro smazání metadat a jeden pro smazání tabulky.

5.4.3.4 InsertStatementTranslator

Pro překlad TSQL2 příkazů typu INSERT slouží třída `InsertStatementTranslator`. Tato třída vezme na vstupu syntaktický strom příkazu pro vložení dat do tabulky, tento příkaz analyzuje a vygeneruje odpovídající SQL příkazy pro vložení dat.



Obrázek 21 - Schéma činnosti překladu INSERT

Překladač nejprve načte název vkládané tabulky a zjistí si pro danou tabulku podporu temporálních dat. Pokud jsou definovány sloupce pro vkládání, načte tyto sloupce do interního pole sloupců. Následně se liší postup podle typu příkazu `INSERT`. Pokud obsahuje příkaz hodnoty zadané pomocí `VALUES`, překladač načte tyto hodnoty a uloží si je do interního pole hodnot. Pokud se jedná o příkaz s poddotazem typu `SELECT`, uloží si překladač tento poddotaz k pozdějšímu zpracování, ke kterému bude potřebovat ještě další údaje.

Jako poslední část vstupu je definice platnosti vkládaných dat. Pokud je zadána, uloží si překladač čas platnosti do vnitřní proměnné. Pokud není čas platnosti zadán, použije se čas od aktuálního okamžiku dále.

Nyní je celý vstup zpracován a následuje krok generování výsledného příkazu.

Pokud nebyly zadány sloupce pro vkládání, vygeneruje překladač seznam všech sloupců dané tabulky a uloží jej do interního pole sloupců. Poté přidá v závislosti na podpoře temporálních dat k těmto sloupcům pomocné sloupce pro uložení času platnosti a transakce.

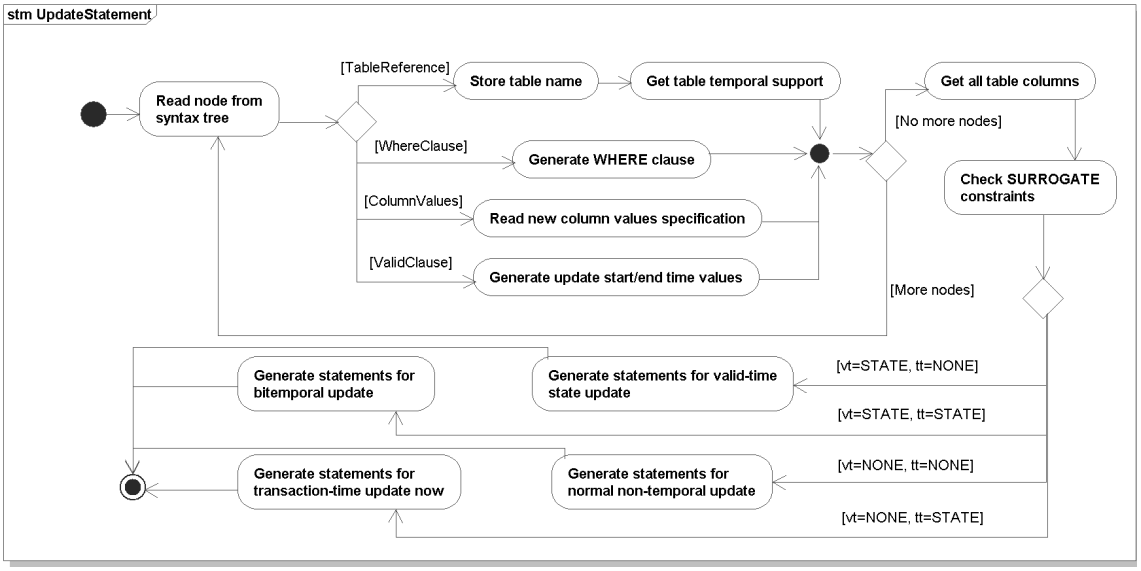
Další krok závisí na tom, zda jsou zadány hodnoty přímo nebo pomocí dotazu `SELECT`. Pokud jsou zadány přímo, provede překladač kontrolu integrity primárního klíče tabulky. Tato kontrola probíhá tak, že se ověří, zda v tabulce neexistuje záznam se stejným primárním klíčem, který by nějakým způsobem překrýval časem platnosti čas platnosti vkládaného záznamu.

Následně musí překladač ověřit, zda nejsou v tabulce sloupce typu SURROGATE a pokud jsou, ověří, zda jim není přiřazována konkrétní hodnota. Sloupce typu SURROGATE totiž mohou být pouze generovány klíčovým slovem `NEW`. Je-li pro některý takový sloupec požadována nová hodnota, musí překladač ještě vygenerovat tuto hodnotu a nahradit jí hodnotu pro vložení. Pokud je vše v pořádku, vygeneruje se výsledný příkaz a překlad končí.

V případě, že jsou hodnoty zadány jako `SELECT`, má tento dotaz překladač uložen. Nyní překladač vytvoří instanci třídy `SelectStatementTranslator` a pomocí této třídy přeloží poddotaz `SELECT` do SQL. K překladu ovšem specifikuje, pro jaký čas platnosti se má překlad provést, aby vrácená data obsahovala správné hodnoty pro nově vytvářený záznam. Navíc musí ověřit, zda nejsou v poddotazu na místě pro případné SURROGATE sloupce uvedeny konkrétní hodnoty. Povoleny jsou pouze jiné sloupce. Výsledkem překladu je potom příkaz `INSERT` s přeloženým poddotazem `SELECT`.

5.4.3.5 UpdateStatementTranslator

Pro překlad TSQL2 příkazů typu `UPDATE` slouží třída `UpdateStatementTranslator`. Tato třída vezme na vstupu syntaktický strom příkazu pro aktualizaci dat tabulky, tento příkaz analyzuje a vygeneruje odpovídající SQL příkazy pro aktualizaci dat.



Obrázek 22 - Schéma činnosti překladu UPDATE

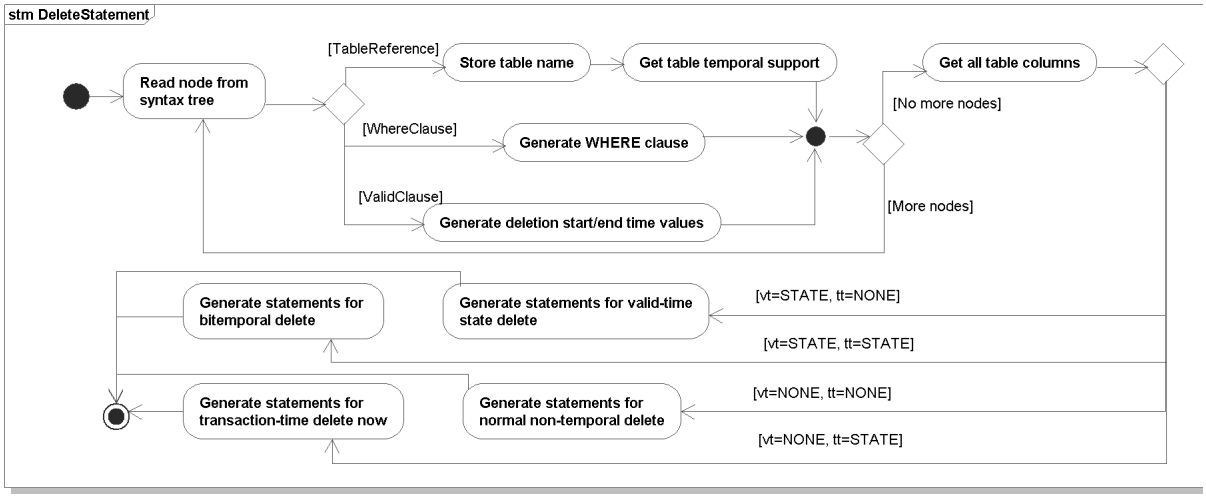
Překladač nejprve načte název aktualizované tabulky a zjistí si pro danou tabulku podporu temporálních dat. Následně načte ze vstupního stromu definice aktualizovaných sloupců a nové hodnoty pro tyto sloupce. Tyto údaje si uloží do interního pole. Načte si případnou definici času platnosti a uloží si začátek a konec intervalu pro aktualizaci. Nakonec přečte případné omezující podmínky.

Po načtení celého vstupu načte překladač seznam všech sloupců aktualizované tabulky. Tyto sloupce budou potřeba při generování některých příkazů pro aktualizaci. Poté provede ověření, zda není pro sloupce typu SURROGATE nastavena konkrétní hodnota a pokud je pro sloupce SURROGATE požadována hodnota nová, překladač ji vygeneruje a nastaví pro aktualizaci.

Následuje fáze generování výsledných příkazů. Jak je uvedeno v kapitole 4.2.5, je třeba vygenerovat pro různé druhy tabulek různé příkazy pro aktualizaci. Při generování příkazů `INSERT` se právě využije seznam všech sloupců tabulky, aby se vložila všechna potřebná data. Překladač tedy vygeneruje sekvenci příkazů popsaných v kapitole 4.2.5. Pokud se tyto příkazy provedou, jsou data v tabulce aktualizována podle původního příkazu `UPDATE`.

5.4.3.6 DeleteStatementTranslator

Pro překlad TSQL2 příkazů typu `DELETE` slouží třída `DeleteStatementTranslator`. Tato třída vezme na vstupu syntaktický strom příkazu pro smazání dat z tabulky, tento příkaz analyzuje a vygeneruje odpovídající SQL příkazy pro smazání dat.



Obrázek 23 - Schéma činnosti překladu DELETE

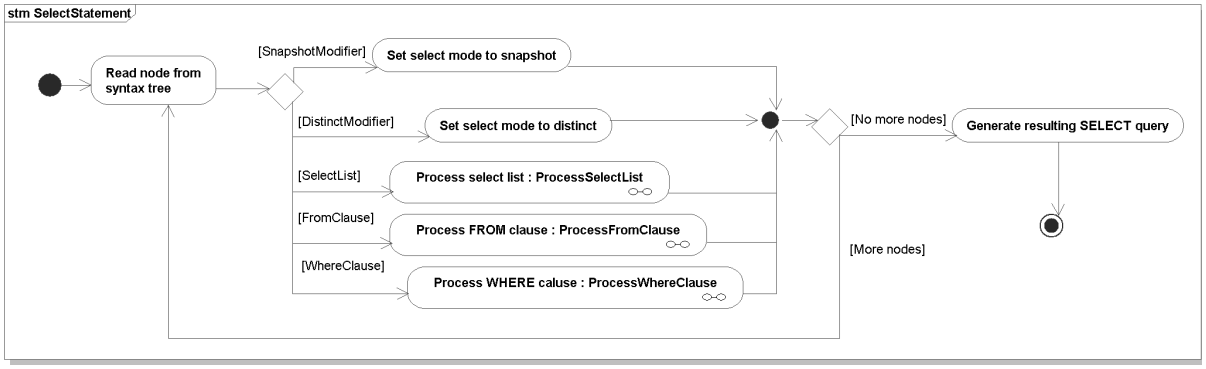
Překladač nejprve načte název tabulky pro mazání dat a zjistí si pro danou tabulku podporu temporálních dat. Přečte si případné omezující podmínky. Nakonec si načte případnou definici času platnosti a uloží si začátek a konec intervalu pro smazání.

Po načtení celého vstupu následuje fáze generování výsledných příkazů. Nejprve se ovšem musí získat všechny sloupce aktualizované tabulky. Tyto sloupce budou potřeba při generování některých příkazů pro mazání.

Jak je uvedeno v kapitole 4.2.4, je třeba vygenerovat pro různé druhy tabulek různé příkazy pro smazání. Při generování příkazů `INSERT` se právě využije seznam všech sloupců tabulky, aby se vložila všechna potřebná data. Překladač tedy vygeneruje sekvenci příkazů popsaných v kapitole 4.2.4. Pokud se tyto příkazy provedou, jsou data v tabulce smazána podle původního příkazu `DELETE`.

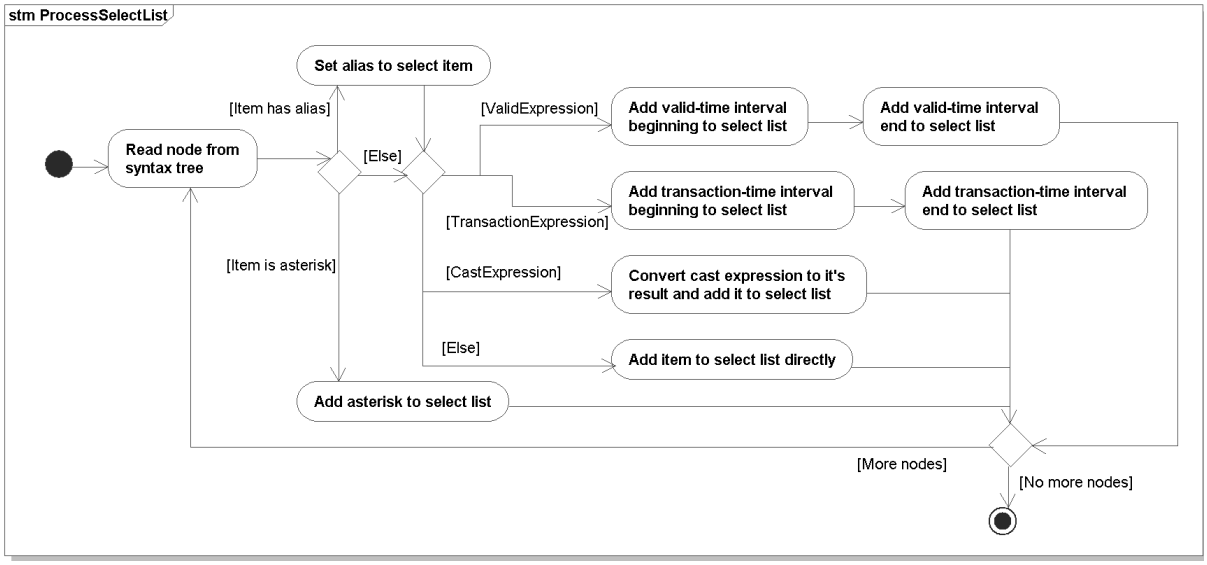
5.4.3.7 SelectStatementTranslator

Pro překlad TSQL2 příkazů typu `SELECT` slouží třída `SelectStatementTranslator`. Tato třída vezme na vstupu syntaktický strom dotazu pro výběr dat z databáze, tento příkaz analyzuje a vygeneruje odpovídající SQL příkaz pro výběr dat.



Obrázek 24 - Schéma činnosti překladu SELECT

Vzhledem ke komplexnosti dotazů typu `SELECT` je tato třída překladače nejkomplikovanější. Jak je vidět na obrázku výše, skládá se překlad dotazu z několika hlavních částí. Nejprve se načtou případné modifikátory `SNAPSHOT` a `DISTINCT`. Modifikátor `SNAPSHOT` potlačí výběr časů platnosti, modifikátor `DISTINCT` potom zajišťuje jedinečnost výsledků. Následně se načte a zpracuje seznam pro výběr, poté se zpracuje klauzule `FROM` a nakonec se zpracují podmínky v klauzuli `WHERE`.



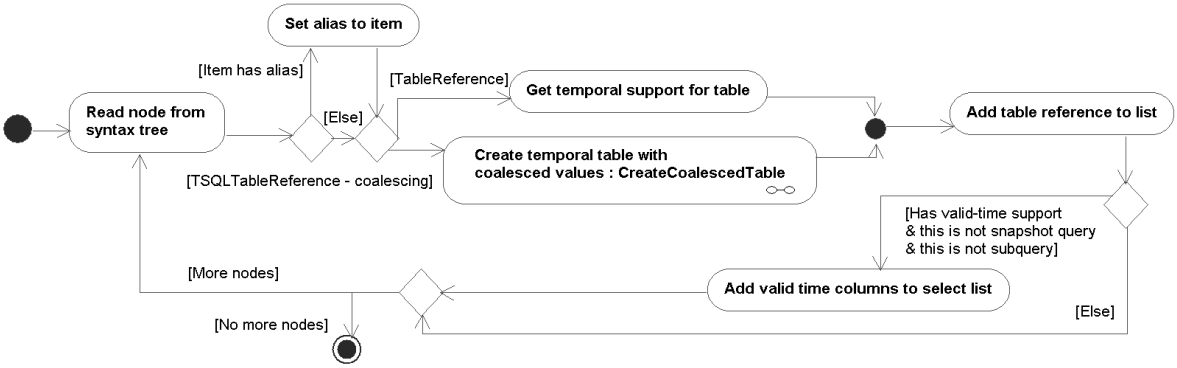
Obrázek 25 - Schéma zpracování seznamu pro výběr u dotazu SELECT

Zpracování seznamu pro výběr probíhá tak, jak je znázorněno na obrázku výše. Překladač postupně čte jednotlivé prvky seznamu. Pokud má prvek definován alias, uloží si překladač tento alias, aby ho mohl k prvku později přiřadit. Poté podle typu prvku provádí potřebné transformace.

- Pokud je prvek typu `VALID(tabulka)`, vygeneruje překladač do seznamu pro výběr dva prvky. Jeden pro výběr počátku času platnosti, druhý pro výběr konce času platnosti. Tyto prvky jsou v tabulce uloženy jako hodnoty sloupců, takže se přidají jména těchto sloupců. Navíc ještě přidá překladač do seznamu pro výběr řetězcové hodnoty, které obsahují název

zdrojové tabulky a případný alias. Tyto hodnoty se následně využijí při prezentaci výsledků.

- Pokud je prvek typu `TRANSACTION(tabulka)`, vygeneruje překladač do seznamu pro výběr dva prvky. Jeden pro výběr počátku času transakce, druhý pro výběr konce času transakce. Tyto prvky jsou v tabulce uloženy jako hodnoty sloupců, takže se přidají jména těchto sloupců. Navíc ještě přidá překladač do seznamu pro výběr řetězcové hodnoty, které obsahují název zdrojové tabulky a případný alias. Tyto hodnoty se následně využijí při prezentaci výsledků.
- Pokud je prvek typu temporálního přetypování `CAST()`, je u tohoto přetypování uvedeno měřítko času, na které se má daný časový údaj převést. Překladač tedy vygeneruje matematický výraz, který převede daný časový údaj na požadované měřítko. Například výraz `CAST(VALID(tabulka) AS INTERVAL DAY)` se převede na `ROUND((tabulka."_VTE" - tabulka."_VTS") / 86400)`, hodnota 86400 zde reprezentuje počet sekund v jednom dni. Výraz tedy znamená, že se vezme délka intervalu času platnosti, který je ve výchozím stavu v sekundách, a tato délka se podělí jedním dnem také v sekundách. Po zaokrouhlení dostáváme počet dní v daném časovém intervalu.
- Pokud je prvek typu `INTERSECT`, překladač vezme počátky a konce intervalů pro průnik a pomocí SQL funkcí `GREATEST()` a `LEAST()` vygeneruje výrazy, které vyberou průnik zadaných intervalů. Při zadání konkrétního intervalu klíčovým slovem `PERIOD` se použijí konkrétní hodnoty, při použití `VALID` se dosadí sloupce s časem platnosti.
- Pokud je prvek hvězdička `"*"`, znamená to, že se nají vybrat všechny sloupce. V této fázi je tento prvek přidán k seznamu pro výběr jako hvězdička, ale později bude ještě upraven.
- Zbylé typy prvků jsou přidány do seznamu pro výběr přímo.

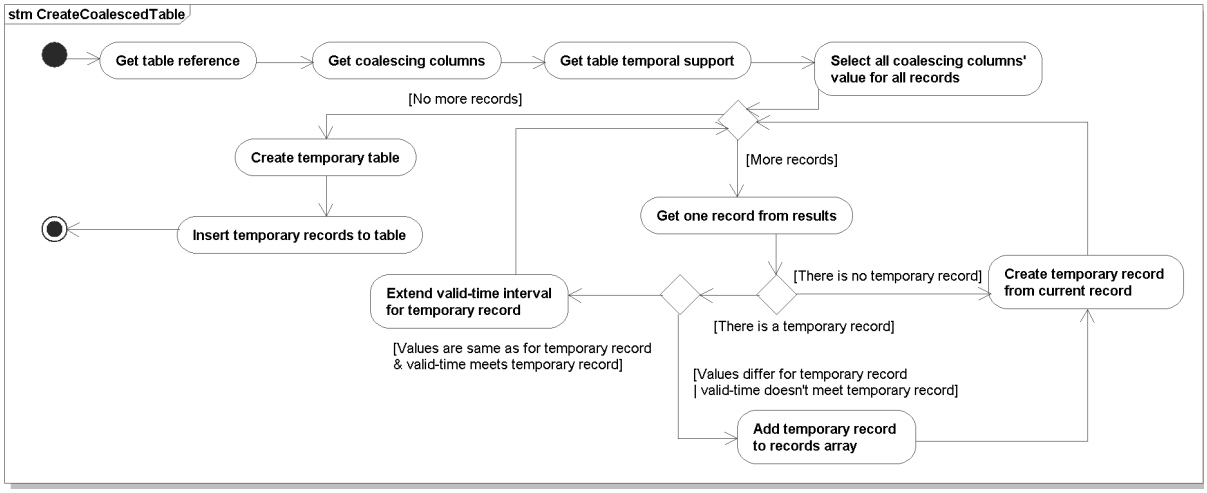


Obrázek 26 - Schéma zpracování klauzule FROM u dotazu SELECT

Po zpracování seznamu prvků pro výběr následuje fáze zpracování klauzule `FROM`, která je uvedena na obrázku výše. Překladač postupně načítá prvky v klauzuli `FROM`. Pokud má prvek

definovaný alias, uloží překladač tento alias k prvku. Potom zkontroluje, o jaký typ prvku se jedná. Pokud se jedná o obyčejnou tabulku, přidá tuto tabulku i s případným liasem do seznamu zdrojových tabulek. Pokud podporuje tabulka časy platnosti a zároveň se nejedná o poddotaz ani o snímkový dotaz, přidá překladač do seznamu pro výběr ještě sloupce s časy platnosti, ze kterých po výběru dat sestaví interval času platnosti daného záznamu.

Pokud se ovšem jedná o tabulku zapsanou syntaxí pro slévání záznamů, která byla zavedena v TSQL2, musí překladač tuto tabulku nejprve vygenerovat. Tato speciální syntaxe pro slévání záznamů v tabulce má tvar `Tabulka(sloupec1, sloupec2, ...)` a znamená, že se mají záznamy v tabulce sloučit podle hodnot zadaných sloupců tak, aby tvořily co nejdelší úseky časů platnosti. Podrobněji je celý postup popsán v kapitole 4.2.6.2.



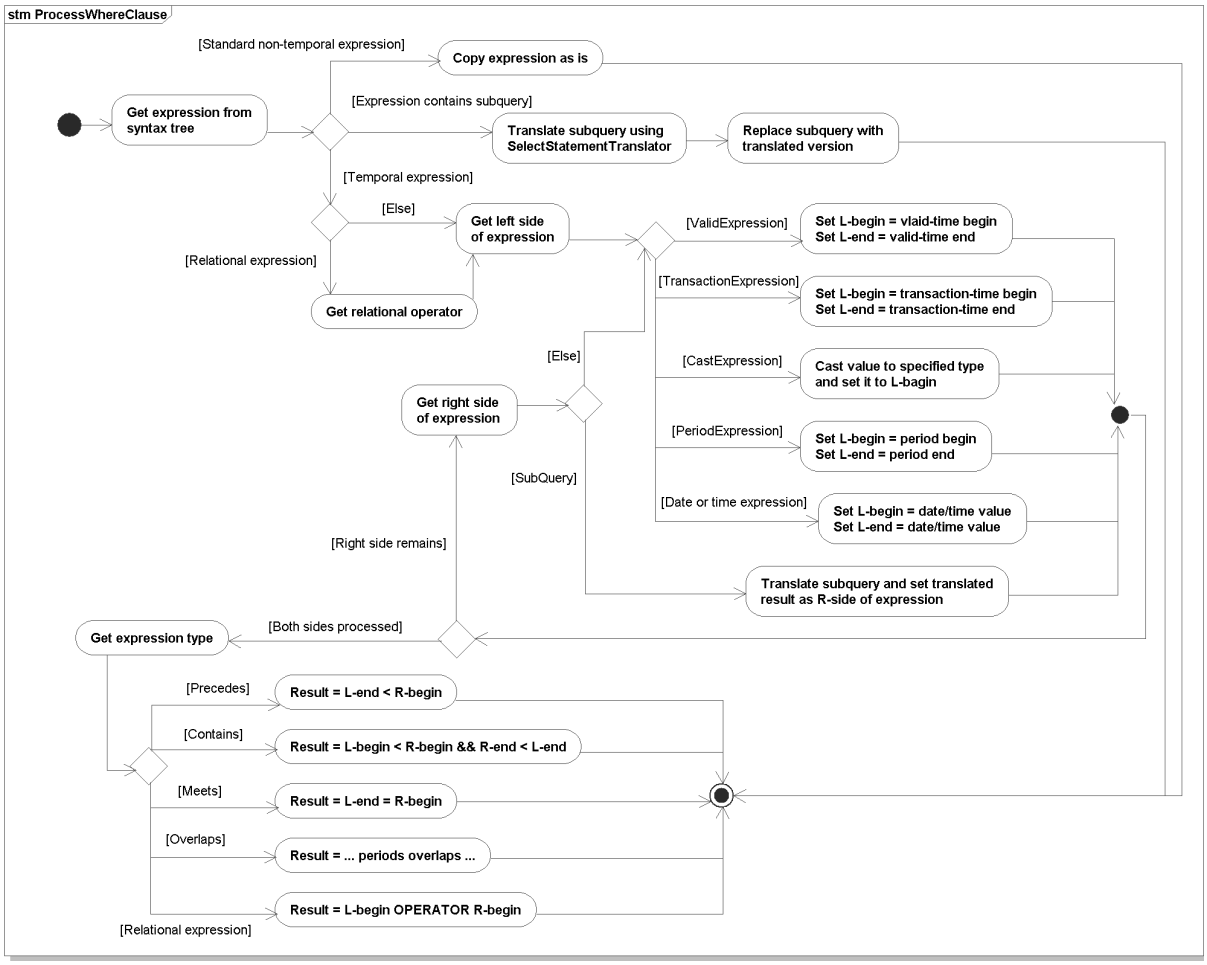
Obrázek 27 - Schéma vytvoření slévané tabulky

Překladač tedy načte název zdrojové tabulky a seznam sloupců, nad kterými se má slévání provést. Poté vybere ze zdrojové tabulky všechny záznamy a hodnoty pro dané sloupce, spolu s časy platnosti a tyto hodnoty vybírá seřazené podle jednotlivých sloupců a časů platnosti.

Z prvního záznamu si vytvoří pomocný záznam, do kterého uloží hodnoty sloupců a interval času platnosti. Poté čte postupně další záznamy a porovnává jejich hodnoty s uloženým pomocným záznamem. Pokud se všechny hodnoty shodují a časy platnosti na sebe navazují, přidá čas platnosti k času platnosti pomocného záznamu a tím prodlouží celkový čas platnosti. Tato operace se nazývá slévání času platnosti.

Jakmile narazí překladač na záznam, který se v některé hodnotě liší od pomocného záznamu, nebo který na pomocný záznam nenavazuje v čase platnosti, uloží pomocný záznam jako maximální nalezený výsledek pro dané hodnoty sloupců do výsledného pole záznamů. Potom do pomocného záznamu přiřadí aktuální záznam, který se liší, a pokračuje v porovnávání. Tímto způsobem se vygenerují všechny maximální slévané záznamy.

Po dokončení slévání vytvoří překladač v databázi dočasnou tabulku a do této tabulky uloží výsledné záznamy. Jméno tabulky zařadí do pole zdrojových tabulek pro výběr. Navíc si jméno této tabulky uloží do vnitřní ho pole a při úklidu po provedení dotazu tuto tabulku smaže.



Obrázek 28 - Schéma zpracování podmínky v klauzuli WHERE

Poslední fází překladu dotazu typu `SELECT` je zpracování podmínek omezujících výběr v klauzuli `WHERE`. Překladač postupně načítá výrazy ze syntaktického stromu a zpracovává je tak, jak je znázorněno na obrázku výše.

Pokud se jedná o výraz, který neobsahuje temporální prvky ani poddotazy, je tento výraz přímo zkopírován a nemusí se překládat.

Pokud obsahuje výraz poddotaz typu `SELECT`, tento poddotaz se rekurzivně přeloží pomocí instance třídy `SelectStatementTranslator` a výsledek překladu se uloží do výstupního řetězce podmínek.

Pokud výraz obsahuje temporální prvek, rozloží se na levou a pravou stranu a případný operátor. Napřed se vezme levá strana výrazu a ověří se, co za druh výrazu představuje. Podle typu se potom vygenerují odpovídající hodnoty do pomocných proměnných L-begin a L-end. Do těchto proměnných se uloží hranice intervalu, který představuje levá strana výrazu. Pokud je levá strana

výrazu pouze jeden časový údaj, uloží se do obou těchto proměnných. Poté se provede stejná operace s pravou stranou výrazu a výsledek se uloží do R-begin a R-end. Je zde ovšem jedna výjimka, kterou je možnost poddotazu na pravé straně výrazu. Pokud se na pravé straně objeví poddotaz, je nejprve přeložen a přeložený dotaz je celý uložen do R-begin.

Následně se vyhodnotí typ výrazu. Podle typu výrazu jsou z hodnot L-begin, L-end, R-begin a R-end sestaveny následující výrazy.

- PrecedesExpression: `L-end < R-begin`
- ContainsExpression: `L-begin <= R-begin AND R-end <= L-end`
- MeetsExpression: `L-end = R-begin`
- OverlapsExpression: `(L-begin <= R-begin AND R-begin < L-end) OR (R-begin <=L-begin AND L-begin < R-end)`
- Jinak: `L-begin OPERATOR R-begin`

První čtyři výrazy jsou predikáty uvedené v TSQL2 a jejich popis můžete nelézt v kapitole 4.2.6.3. Poslední výraz prostě vezme zadaný relační operátor a porovná jím levou a pravou stranu.

Nakonec má překladač k dispozici všechny potřebné údaje pro vygenerování odpovídajícího SQL dotazu, který vybere požadovaná data. Postupně se tedy generuje seznam hodnot pro výběr, kde se musí například provést přidání pomocných aliasů k některým sloupcům pro následné zpracování výsledku. Připojí se jednotlivé tabulky a poddotazy do klauzule `FROM` a nakonec se složí podmínka v části `WHERE`. Výsledný dotaz je většinou řádově složitější, než původní dotaz zapsaný v TSQL2, protože podpora temporálních dat se musí různě složitě simulovat.

Příklad takového překladu může vypadat takto.

TSQL2:
```
SELECT id, jmeno, TRANSACTION(z) AS cas_transakce
FROM Zamestnanci AS z
WHERE VALID(z) OVERLAPS DATE '2005-05-24'
```

SQL:
```
SELECT id, jmeno, z."_TTS" AS "_E_TTS", 'z', 'cas_transakce',
       z."_TTE" AS "_E_TTE", z."_VTS" AS "_I_VTS",
       'Zamestnanci', 'z', z."_VTE" AS "_I_VTE"
FROM Zamestnanci z
WHERE ((z."_VTS" <= 1116885600 AND 1116885600 < z."_VTE")
    OR (1116885600 <= z."_VTS" AND z."_VTS" < 1116885600))
    AND "_TTE" > 1242243437
```

Dotaz má vybrat hodnoty ID, jména a časů transakce pro záznamy zaměstnanců, které byly platné dne 24.5.2005. Jak je vidět na dotazu přeloženém do SQL, obsahuje výsledek pouze běžné

operace jazyka SQL. Ve výsledku ovšem vybírá přesně požadovaná data. Výsledek výběru je samozřejmě nutné před předáním aplikaci ještě upravit. Jedná se hlavně o vytvoření časových úseků z jednotlivých sloupců a odstranění pomocných dat z výsledku. Nakonec dostane aplikace například následující výsledky.

Tabulka 12 - Výsledek přeloženého dotazu SELECT

id	jmeno	cas_transakce	VALID
1	Petr	2003-03-15 12:23:34 – NOW	2002-01-01 – 2005-06-08
2	Martin	2002-03-30 14:21:14 – NOW	2004-03-21 - NOW

6 Závěr

V této práci byl popsán princip temporálních databází a jejich rozdíl oproti běžným relačním databázím. Byly zde představeny a zhodnoceny dostupné implementace temporálních databází z hlediska použitelnosti pro produkční nasazení.

Následoval popis specifikace a jazyka TSQL2, který byl vyvinut právě pro podporu temporálních databází jako rozšíření specifikace a jazyka SQL-92.

Hlavní částí této práce je ovšem návrh a implementace interpretu jazyka TSQL2 pro použití nad relační databází. V tomto návrhu je rozebrán způsob reprezentace temporálních dat v relační databázi a jsou zde zmíněny problémy, které se ve vztahu k temporálním datům objevují. Je zde uveden způsob ukládání dalších potřebných metadat k temporálním datům s použitím prostředků relačních databází. Dále je zde představen způsob překladu příkazů jazyka TSQL2 do čistého SQL, pro každý typ základních příkazů a dotazů je zde podrobný popis.

V kapitole o implementaci je představena konkrétní vyvinutá implementace interpretu podmnožiny jazyka TSQL2. Jsou zde popsány jednotlivé části systémy a jeho celková funkčnost a provázanost jednotlivých částí.

Definice TSQL2 obsahuje velké množství úprav proti klasickému SQL. V této práci se podařilo implementovat hlavní část z těchto rozšíření ať už v rámci příkazů jazyka nebo v rámci dotazů pro výběr dat. Stále je však několik oblastí, které se v této práci nepodařilo pokrýt. Jedná se především o referenční integritu, která se v kombinaci s temporálními daty stává velice složitým problémem. Z tohoto důvodu není v implementaci zahrnuta podpora cizích klíčů mezi tabulkami. Dalším chybějícím prvkem je podpora pro uživatelskou definici kalendářů. Implementované řešení pracuje pouze s běžnou časovou osou v rozsahu 1.1.0001 – 31.12.9999 s přesností na sekundy.

Výsledek této práce je volně dostupný komukoliv, kdo projeví zájem o úpravy nebo další vývoj tohoto řešení. Další vývoj by se mohl ubírat právě směrem k implementaci referenční integrity nebo k optimalizaci některých částí interpretu. Zvlášť pro dotazy typu SELECT by se jistě našlo několik možností ke zvýšení výkonu. Také přidání podpory pro vlastní definici kalendářů může být zajímavým rozšířením možností tohoto produktu.

7 Literatura

[BO94] BÖHLEN, Michael. Managing Temporal Knowledge in Deductive Databases. [s.l.], 1994. 174 s. Dizertační práce.

[Ora1] Oracle® Database SQL Reference, 10g Release 2 (10.2), B14200-02, December 2005

[TSQL2] SNODGRASS, Richard T.. TSQL2 Temporal Query Language [online]. - [cit. 2008-12-17]. Dostupný z WWW: <http://www.cs.arizona.edu/~rts/tsql2.html>.

[TSQL2Spec] SNODGRASS, Richard T., et al. TSQL2 Language Specification [online]. 1994 [cit. 2008-12-17]. Dostupný z WWW: <ftp://ftp.cs.arizona.edu/tsql/tsql2/spec.pdf>.

[SBJS96tt] SNODGRASS, Richard T., BÖHLEN, Michael H., JENSEN, Christian S., STEINER, Andreas. Adding Transaction Time to SQL/Temporal. SQL/Temporal Change Proposal. listopad 1996. ANSI X3H2-96-502r2, ISO/IEC JTC1/SC21/WG3 DBL MAD-147r2

[SBJS96vt] SNODGRASS, Richard T., BÖHLEN, Michael H., JENSEN, Christian S., STEINER, Andreas. Adding Valid Time to SQL/Temporal. SQL/Temporal Change Proposal. listopad 1996. ANSI X3H2-96-501r2, ISO/IEC JTC1/SC21/WG3 DBL MAD-146r2, listopad 1996.

[ST98] STEINER, Andreas. A Generalisation Approach to Temporal Data Models and their Implementations. [s.l.], 1998. 163 s. SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH. Dizertační práce. Dostupný z WWW: <http://www.timeconsult.com/Publications/diss.pdf>.

[SN00] SNODGRASS, Richard T. Developing Time-Oriented Database Applications in SQL. San Francisco, California : Morgan Kaufmann Publishers, 2000. 504 s., 1 CD-ROM. ISBN 1-55860-436-7.

[SQLGR] DRAHEIM, Guido. Java - javacc/jjtree problems [online]. 2004-02-01 [cit. 2009-05-10]. Dostupný z WWW: <http://www2.informatik.hu-berlin.de/~draheim/java/jjtree.html>.

[JavaCC] JavaCC : Project home [online]. [2008] [cit. 2009-05-08]. Dostupný z WWW: <https://javacc.dev.java.net/>.

[WikiSql] SQL [online]. 2008 [cit. 2008-12-17]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Sql>.

[WikiTT] Terrestrial Time [online]. 2008 [cit. 2008-12-17]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Terrestrial_Time>.

[WikiUTC] Coordinated Universal Time [online]. 2008 [cit. 2008-12-17]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Coordinated_Universal_Time>.

Seznam příloh

Příloha 1: Gramatika překladače

V této příloze se nachází definice syntaxe jazyka pro překladač. Jedná se o rozšíření jazyka SQL, ale jsou vypuštěny veškeré části jazyka mimo dotazů SELECT a příkazů CREATE TABLE, DROP TABLE, INSERT, UPDATE a DELETE.

1.1 Definice časových údajů

```
datetime_definition:
     {DATE|TIME|TIMESTAMP} {'datetime_string'|now_relative_date}
datetime_string:
     YYYY[-MM[-DD[ HH[:MIN[:SS]]]]]
now_relative_date:
     NOW [{+|-} number [datetime_scale]]
datetime_scale:
     SECOND | MINUTE | HOUR | DAY | MONTH | YEAR
```

Řetězec pro definici data a času může mít podobu konkrétního časového údaje nebo relativního časového údaje vztaženého k aktuálnímu času pomocí klíčového slova `NOW`.

Jednotlivé části data a času je možné zprava vynechat. Význam částí je následující.

- YYYY – rok v rozmezí 1-9999
- MM – měsíc v rozmezí 1-12
- DD – den v rozmezí 1-31
- HH – hodina v rozmezí 0-23
- MIN – minuta v rozmezí 0-59
- SS – sekunda v rozmezí 0-59

Při relativní definici je možné specifikovat měřítko pro relativní posun. Bez definice měřítka se bere jednotka sekunda.

Příklady:

```
DATE '2009-02-15 15:32:45'
```

Datum 15.2.2009 a čas 15:32:45.

```
DATE '2008-12-05'
```

Datum 5.12.2008.

```
DATE '2008-03'
```

Datum březen 2008.

```
DATE NOW-7 DAY
```

Výsledek je aktuální datum mínus 7 dní. Pokud je tedy například 13.4.2009, je výsledek 6.4.2009

```
period_definition:
     PERIOD "[" {datetime_string|now_relative_date} -
                {datetime_string|now_relative_date|FOREVER} "]"
     [datetime_scale]
```

Časový úsek se definuje pomocí klíčového slova PERIOD, počátečního času a koncového času. Časový úsek se považuje za zprava otevřený, tedy koncový časový údaj již v úseku není. Jako konec času může být zadáno klíčové slovo FOREVER, které udává otevřený konec intervalu.

Příklady:

```
     PERIOD [2008-03-26 – 2008-05-30]
```

Časový úsek od 26.3.2008 do 30.5.2008.

```
     PERIOD [2008-03-26 12:34:56 – FOREVER]
```

Časový úsek od 26.3.2008 12:34:56 dále.

```
     PERIOD [NOW – NOW+7 DAY]
```

Časový úsek od aktuálního okamžiku do okamžiku za sedm dní. Pokud je tedy například 3.4.2005, vyhodnotí se tento výraz jako PERIOD [2005-04-03 – 2005-04-10].

```
valid_definition:
     VALID {period_definition | datetime_definition}
```

Definice času platnosti pro příkazy INSERT, UPDATE a DELETE.

1.2 CREATE TABLE

```
CREATE TABLE tbl_name
    (content_definition,...)
    [temporal_options]
    [vacuuming_options]
```

Definice obsahu tabulky content_definition je shodná se standardem SQL, pouze je navíc přidán datový typ SURROGATE do definice sloupce.

```
temporal_options:
     AS VALID [STATE|EVENT] [datetime_scale] [AND TRANSACTION]
|    AS TRANSACTION
```

Definice temporální podpory tabulky se provádí výše uvedeným výrazem. Stavová tabulka se definuje pomocí klíčového slova `VALID`, transakční tabulka pomocí `TRANSACTION` a bitemporální pomocí obou. Pokud není definice vůbec zadána, je tabulka snímková.

```
vacuuming_options:
     VACUUM {datetime_definition | NOBIND(datetime_definition)}
```

Definice času pro odmazávání starých záznamů se provádí klíčovým slovem `VACUUM` a specifikací data a času. Je možné použít i relativní definici data. Pokud se použije klauzule `NOBIND()` v kombinaci s relativním časem, je tento čas při každém přístupu k tabulce přepočítáván podle zadání. Bez klauzule `NOBIND()` je čas vypočítán pouze jednou při vytvoření tabulky.

1.3 DROP TABLE

```
DROP TABLE table_name
```

1.4 INSERT

```
INSERT INTO table_name [(column_name,...)]
  VALUES ({expr|DEFAULT|NEW}
  [valid_definition]
```

nebo

```
INSERT INTO table_name [(column_name,...)]
  SELECT ...
```

Do seznamu hodnot pro vložení je přidáno nové klíčové slovo `NEW` pro specifikaci nové hodnoty pro sloupce typu SURROGATE. Dále je přidána možnost definovat čas platnosti vkládaného záznamu. Pokud není platnost zadána, platí záznam od aktuálního okamžiku dále.

Příklad:

```
     INSERT INTO tabulka VALUES ('Petr')
     VALID PERIOD [2005-02-01 – FOREVER]
```

Záznam se vloží s platností od 1.2.2005, konec platnosti není specifikován.

1.5 UPDATE

```
UPDATE table_name
```

```
SET column_name={expr|NEW} [, column_name={expr|NEW} ...]
[valid_definition]
[WHERE conditions]
```

Jako nová hodnota sloupce typu SURROGATE může být zadáno klíčové slovo NEW pro přiřazení nepoužité hodnoty. Je možné zadat časový úsek pro aktualizaci záznamů. Pokud není časový úsek zadán, bere se aktualizace od aktuálního okamžiku dále.

Příklad:

```
UPDATE tabulka SET jmeno = 'Martin'
VALID PERIOD [2005-06-08 – 2006-03-21]
WHERE id = 1
```

Provede změnu jména u záznamu s ID 1 na jméno Martin, ale pouze v období od 8.6.2005 do 21.3.2006.

1.6 DELETE

```
DELETE FROM table_name
  [WHERE conditions]
  [valid_definition]
```

Je přidána možnost specifikovat časový úsek pro smazání. Pokud není časový úsek zadán, bere se čas od aktuálního okamžiku dále.

1.7 SELECT

```
SELECT [ALL|DISTINCT] [SNAPSHOT] select_list
FROM table_reference [, table reference ...]
[WHERE conditions]

select_list:
    * | column_name | expr | VALID(table_name) | TRANSACTION(table_name)
      | cast_expression | intersect_expression

cast_expression:
    CAST(VALID(table_name) AS INTERVAL number datetime_scale)

intersect_expression:
    INTERSECT({VALID(table_name)|period_definition},
              {VALID(table_name)|period_definition})
```

```
table_reference:
     table_name[(column_name[, column_name ...])] [AS] [alias]
|    select_statement

conditions:
     standard_sql_conditions
|    expr [NOT] PRECEDES expr
|    expr [NOT] CONTAINS expr
|    expr [NOT] MEETS expr
|    expr [NOT] OVERLAPS expr
```

K dotazu typu SELECT je přidána možnost specifikovat výběr v režimu SNAPSHOT, kdy se nepracuje s časy platnosti záznamů.

Přibyla možnost vybrat čas platnosti pomocí klauzule VALID() a čas transakce pomocí klauzule TRANSACTION(). Je možné převést čas platnosti na intervaly dané přesnosti pomocí klauzule CAST(). Tyto klauzule je možné používat jak při výběru hodnot, tak také jako výrazy v podmínkách výběru.

Je možné vybrat průnik dvou intervalů zadaných jako čas platnosti nebo konkrétní perioda pomocí klauzule INTERSECT(). Pokud nemají dané intervaly průnik, výsledkem je hodnota NULL.

Do části FROM přibyla možnost specifikovat k tabulce pro výběr seznam sloupců, podle kterých se provede slévání časů platnosti při výběru.

Do podmínek pro omezení výběru přibyly čtyři nové predikáty pro porovnávání časových úseků.

Příklady:

```
SELECT * FROM tabulka
```

Vybere všechny záznamy z tabulky spolu s jejich časy platnosti.

```
SELECT SNAPSHOT * FROM tabulka
```

Vybere všechny záznamy z tabulky, ale bez jejich časů platnosti.

```
SELECT TRANSACTION(t) FROM tabulka AS t
```

Vybere časy transakce pro všechny záznamy v tabulce.

```
SELECT CAST(VALID(t) AS INTERVAL DAY) FROM tabulka AS t
```

Vybere časy platnosti záznamů v tabulce a převede tyto časy na délky daných intervalů ve dnech.

```
SELECT * FROM tabulka(id, jmeno)
```

Vybere všechny záznamy z tabulky, vzniklé sloučením časů platnosti podle sloupců `id` a `jmeno`.

```
SELECT * FROM tabulka t WHERE VALID(t) PRECEDES DATE '2009-03-04'
```

Vybere všechny záznamy, kterým skončila platnost před datem 4.3.2009.

```
SELECT * FROM tabulka t WHERE CAST(VALID(t) AS INTERVAL YEAR) > 5
```

Vybere všechny záznamy, jejichž délka času platnosti je větší než 5 let.

Příloha 2: Tvorba ukázkové aplikace

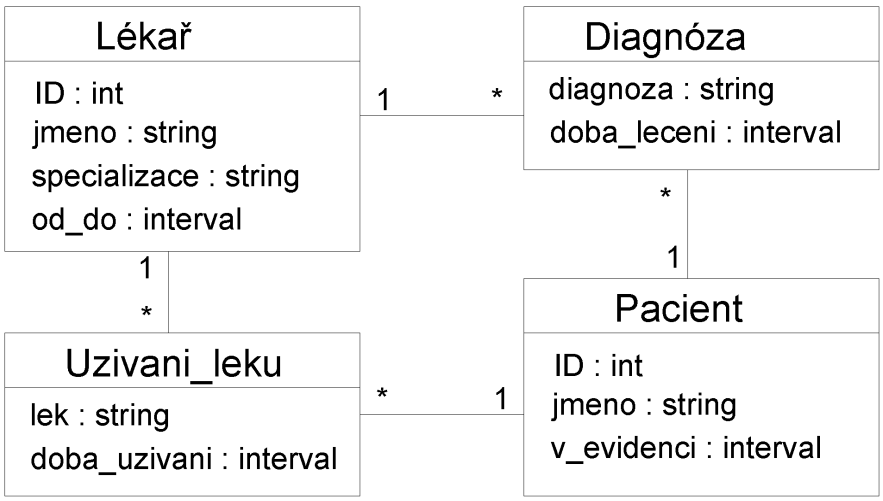
V tomto dokumentu bude popsán příklad použití knihovny na jednoduché ukázkové aplikaci.

Kompletní knihovna tsql2lib je obsažena v souboru `tsql2lib.jar`. Pro použití knihovny je třeba tento soubor přidat do projektu jako externí knihovnu. Od té chvíle je možné používat v aplikaci třídy pro připojení a práci s databází. Knihovna implementuje rozhraní JDBC, takže pro podrobný popis jednotlivých metod doporučuji http://java.sun.com/products/jdbc/overview.html.

2.1 Specifikace aplikace

Ukázková aplikace bude jednoduchá grafická aplikace v jazyce Java, která bude obsahovat ukázky dotazů v jazyce TSQL2 a konzoli pro zadávání vlastních dotazů a příkazů. Jako ukázková data bude sloužit jednoduchá evidence lékařů, pacientů, diagnóz a léků.

Lékař bude mít specializaci a časový údaj, kdy danou specializaci vykonával. Pacient bude mít evidováno jméno a dobu, po kterou je v evidenci. Lékař určuje u pacienta diagnózu, u které se eviduje doba léčení. Lékař pacientovi může předepisovat léky, u kterých se eviduje doba užívání. Datový model je na následujícím obrázku.



2.2 Struktura databáze

Databáze pro uložení výše uvedených dat bude obsahovat čtyři tabulky, které mají následující definice.

```
CREATE TABLE Lekari (
     id INT NOT NULL PRIMARY KEY,
     jmeno VARCHA(32) NOT NULL
     specializace VARCHAR(32) NOT NULL
```

```
) AS VALID STATE DAY

CREATE TABLE Pacienti (
     id INT NOT NULL PRIMARY KEY,
     jmeno VARCHAR(32) NOT NULL
) AS VALID STATE DAY

CREATE TABLE Diagnozy (
     pacient INT NOT NULL,
     diagnoza VARCHAR(32) NOT NULL,
     lekar INT NOT NULL
) AS VALID STATE DAY

CREATE TABLE Leky (
     pacient INT NOT NULL,
     lek VARCHAR(32) NOT NULL,
     lekar INT NOT NULL
) AS VALID STATE DAY
```

2.3 Vzorová data

Tabulka 1 - Lékaři

ID	Jméno	Specializace	Od-do
1	Jan Novák	obvodní lékař	[1995-01-01 - 2000-01-06]
1	Jan Novák	kardio	[2000-01-06 - FOREVER]
2	Pavel Kovář	kožní	[1998-08-06 - FOREVER]
3	Jana Jandová	ORL	[2000-04-01 - FOREVER]
4	Petra Kuncová	obvodní lékař	[2000-06-01 - FOREVER]

Tabulka 2 - Pacienti

ID	Jméno	V evidenci
1	Martin Tlustý	[1995-06-03 - FOREVER]
2	Petr Polák	[2001-07-18 - FOREVER]
3	Marta Novotná	[2002-03-13 - 2005-09-03]
4	Alena Adámková	[2005-06-02 - FOREVER]

Tabulka 3 - Diagnózy

Pacient	Diagnóza	Lékař	Doba léčení
1	chřipka	1	[1995-06-03 - 1995-06-13]
1	angína	1	[1995-06-13 - 1995-06-20]
2	infarkt	2	[2004-06-03 - 2004-08-08]

3	popáleniny	2	[2001-01-02 - 2001-06-06]
4	zánět stř. ucha	3	[2004-12-06 - 2005-01-06]
1	angína	4	[2003-08-03 - 2003-08-17]
3	nachlazení	4	[2004-10-13 - 2004-10-20]
2	omrzliny	2	[2005-01-03 - 2005-02-03]
4	nachlazení	4	[2004-12-10 - 2004-12-20]

Tabulka 4 - Předepsané léky

Pacient	Lék	Lékař	Doba užívání
1	paralen	1	[1995-06-03 - 1995-06-13]
1	penicilin	1	[1995-06-13 - 1995-06-20]
1	augmentin	4	[2003-08-03 - 2003-08-17]
2	aktiferrin	1	[2004-06-03 - FOREVER]
2	avenoc	1	[2005-10-08 - 2006-03-02]
3	betadine	2	[2001-01-02 - 2001-03-04]
3	aciclovir	2	[2001-03-04 - 2001-06-06]
3	paralen	4	[2004-10-13 - 2004-10-20]
2	betadine	2	[2005-01-03 - 2005-02-03]
2	balmandol	2	[2005-01-03 - 2005-01-13]
4	oxafirol	3	[2004-12-06 - 2005-01-06]
4	paralen	4	[2004-12-10 - 2004-12-20]

2.4 Vytvoření aplikace

V této kapitole bude popsáno vytvoření aplikace z hlediska použití knihovny tsql2lib. Nebude zde podrobně popsán postup tvorby GUI a podobné věci, které nesouvisejí s databází.

2.4.1 Načtení knihovny do aplikace

Knihovna `tsql2lib` je dostupná jako jeden archiv `tsql2lib.jar`. V prvním kroku tvorby aplikace tedy připojíme soubor s knihovnou do projektu. Do sekce include v souboru aplikace přidáme následující příkazy.

```
import tsql2lib.TSQL2Adapter;
import tsql2lib.TSQL2Types;
import tsql2lib.TypeMapper;
```

`TSQL2Adapter` je třída pro připojení k databázi, kterou budeme potřebovat pro komunikaci s databází.

Třída `TSQ2Types` obsahuje obecné datové typy nezávislé na databázovém systému a třída `TypeMapper` slouží k mapování těchto obecných typů na konkrétní typy pro danou databázi. Tyto třídy budeme potřebovat při vytváření struktury ukázkové databáze pro zajištění přenositelnosti.

2.4.2 Připojení k databázi

Po spuštění ukázkové aplikace se zobrazí dialog pro zvolení typu databáze a pro zadání řetězce pro připojení k databázi. Podle zvoleného typu – MySQL nebo Oracle – a řetězce pro připojení je třeba vytvořit připojení k databázi. K tomu slouží metoda `connect(String databaseType, String url)`.

```
private void connect(String databaseType, String url) {
     try {
       if (databaseType.equals("MySQL")) {
          Class.forName("com.mysql.jdbc.Driver");
          // vytvorime pripojeni k MySQL a pouzijeme je pro vytvoreni TSQL2
          // pripojeni
           _con = new TSQL2Adapter(DriverManager.getConnection(url,
                                                               "root",
                                                               "root"));
       } else {
          OracleDataSource ods = new OracleDataSource();
           ods.setURL(url);
          // vytvorime pripojeni k Oracle a pouzijeme je pro vytvoreni
          // TSQL2 pripojeni
          _con = new TSQL2Adapter(ods.getConnection());
       }
     } catch (Exception e) {
       JOptionPane.showMessageDialog(this, e.getMessage());
     }
}
```

Podle zadaného typu databáze se v této metodě vytvoří standardní JDBC Connection specifický pro danou databázi. Toto připojení se poté hned použije pro vytvoření instance `TSQL2Adapter`, která se uloží do atributu `_con`. Od této chvíle má aplikace v atributu `_con` k dispozici připojení k databázi s podporou TSQL2. Původní JDBC připojení podle typu databáze aplikace k ničemu nepotřebuje a je pouze uvnitř instance `TSQL2Adapter`.

2.4.3 Vytvoření testovacích dat

Po připojení k databázi je třeba vytvořit testovací data. Aplikace má k tomuto účelu volbu v menu pod položkou `Akce->Inicializovat testovací data`. Po výběru této volby se zavolá metoda, která naplní databázi daty uvedenými v úvodu. Tato metoda vypadá následovně.

```
private void initDatabase() {
     Statement stmt = null;
     try {
       stmt = _con.createStatement();
       // smazani puvodnich tabulek
       try {
         stmt.execute("DROP TABLE Lekari");
       } catch (SQLException e) {}
       // pripadne smazani ostatnich tabulek

       // definice tabulek
       stmt.execute("CREATE TABLE Lekari ( "
       +"id " + TypeMapper.get(TSQL2Types.INT)+ " NOT NULL PRIMARY KEY, "
       +"jmeno " + TypeMapper.get(TSQL2Types.VARCHAR) + "(32) NOT NULL, "
       +"specializace "+TypeMapper.get(TSQL2Types.VARCHAR)+"(32) NOT NULL"
       +" ) AS VALID STATE DAY");
       // zde nasleduji definice dlasich tabulek ...

       // vlozeni dat do tabulek
       stmt.execute("INSERT INTO lekari "
                  + " VALUES (1, 'Jan Novák', 'obvodní lékař') "
                  + " VALID PERIOD [1995-01-01 - 2000-01-06]");

       // zde nasleduje vlozeni zbylych zaznamu ...
     } catch (SQLException e) {
       JOptionPane.showMessageDialog(this, e.getMessage());
     } finally {
       if (stmt != null) {
         try {
           stmt.close();
         } catch (SQLException sqlEx) {}
         stmt = null;
       }
     }
}
```

V metodě se vytvoří instance rozhraní `Statement` z uloženého připojení v atributu `_con`. Toto připojení je typu `TSQL2Statement`, ale implementuje rozhraní `Connection`, takže práce s ním je stejná jako s jakýmkoliv jiným JDBC připojením. Rozdíl spočívá v tom, že vytvořený objekt `Statement` podporuje příkazy TSQL2.

Nejprve se pokusíme vymazat případné předchozí verze tabulek s testovacími daty. Příkaz pro smazání tabulky je standardní SQL příkaz.

Potom vytvoříme tabulky, které jsou popsány v kapitole 2.2. Zde již využijeme syntaxi TSQL2 kdy specifikujeme pomocí klauzule `AS VALID STATE DAY`, že tabulky mají podporovat čas platnosti s přesností na dny. Protože tuto aplikaci vytváříme pro více databází, použijeme v příkazu pro vytvoření tabulky třídu `TypeMapper`, pomocí které nadefinujeme typy sloupců tabulky. Při vykonání příkazu se potom podle typu databáze dosadí konkrétní datový typ.

Nakonec naplníme tabulky testovacími daty. Použijeme příkazy `INSERT` a specifikujeme pro jednotlivé záznamy čas jejich platnosti pomocí klauzule `VALID`.

2.4.4 Dotazy na databázi a zobrazení výsledků

Ukázková aplikace obsahuje několik předdefinovaných ukázkových dotazů v jazyce TSQL2. Mimo to umožňuje zadávat vlastní dotazy. Pro vykonání dotazu a zobrazení výsledku tohoto dotazu slouží metoda `query(String query)`.

```
private void query(String query) {
     Statement stmt = null;
     ResultSet results = null;
     ResultSetMetaData meta = null;

     try {
       // vytvorime statement pro vykonani prikazu
       stmt = _con.createStatement();

       // vykoname prikaz/dotaz
       if (stmt.execute(query)) {
         // nacteme vysledky dotazu
         results = stmt.getResultSet();
         // nacteme metadata vysledku
         meta = results.getMetaData();

         // vytvorime pole s nazvy sloupcu
         int cols = meta.getColumnCount();
         Vector<String> columnNames = new Vector<String>();
```

```
          for (int i = 1; i <= cols; i++) {
            columnNames.add(meta.getColumnLabel(i));
          }

          // vytvorime pole s vysledky dotazu
          Vector<Vector<String>> data = new Vector<Vector<String>>();
          while (results.next()) {
            Vector<String> row = new Vector<String>();
            String str = "";
            for (int i = 1; i <= cols; i++) {
                  // vememe hodnotu sloupce jako řetězec pro zobrazeni
                  str = results.getString(i);
                  row.add(str);
            }
            data.add(row);
          }

          // zobrazime vysledky v tabulce
        } else {
          // vypiseme OK jako uspech provedemi prikazu
        }
      } catch (SQLException e) {
        // vypiseme chybu
      } finally {
        if (results != null) {
          try {
            results.close();
          } catch (SQLException sqlEx) {
          } // ignore
          results = null;
        }
        if (stmt != null) {
          try {
            stmt.close();
          } catch (SQLException sqlEx) {
          } // ignore
          stmt = null;
        }
      }
}
```

V metodě nejprve vytvoříme instanci třídy `Statement` pro vykonání dotazu nebo příkazu. Vykonám dotaz nebo příkaz pomocí metody `execute()` a podle vrácené hodnoty zjistíme, jestli jsou výsledky k zobrazení. Pokud ano, načteme jednotlivé sloupce pomocí metadat výsledku a vygenerujeme tabulku s výsledky.

Jak je vidět, použití knihovny je stejné jako u jakéhokoliv jiného adaptéru JDBC. Z hlediska aplikace se chová `TSQL2Adapter` a další třídy jako běžné JDBC rozhraní k databázi.

2.4.5 Ukončení připojení

Před ukončením aplikace je vhodné ukončit připojení k databázi. Stejně jako u jakéhokoliv jiného připojení se to provede jednoduše zavoláním metody `close()`.

```
private void disconnect() {
     try {
       if ((null != _con) && (_con.isValid(1000))) {
         _con.close();
       }
     } catch (SQLException e) {
       JOptionPane.showMessageDialog(this, e.getMessage());
     }
}
```

2.5 Závěr

Hotová ukázková aplikace je uložena na přiloženém CD. Jak je vidět z popisu uvedeného v této příloze, je použití knihovny `tsql2lib` stejné, jako použití jiného JDBC rozhraní k databázi. Jediný rozdíl je při vytváření připojení, kdy je třeba použít jiné JDBC připojení, které se zabalí do třídy `TSQL2Adapter`.