VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

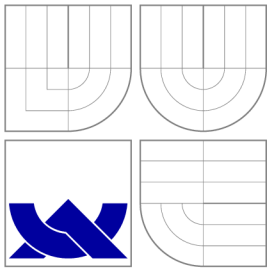
TEMPORÁLNÍ ROZŠÍŘENÍ PRO POSTGRESQL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

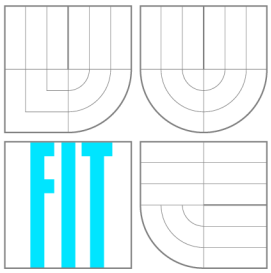
AUTOR PRÁCE
AUTHOR

Bc. RADEK JELÍNEK

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TEMPORÁLNÍ ROZŠÍŘENÍ PRO POSTGRESQL

A TEMPORAL EXTENSION FOR POSTGRESQL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADEK JELÍNEK

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. MAREK RYCHLÝ, Ph.D.

BRNO 2015

Abstrakt

Tato práce se zabývá temporálním rozšířením databázového systému PostgreSQL. Čtenář se tu seznámí se stručným úvodem do temporálních databází, databázovým systémem PostgreSQL, návrhem rozšíření pro PostgreSQL a konkrétní implementací doplněnou příklady. Jsou tu uvedeny i používané temporální databázové systémy a využití temporálních databází v praxi.

Abstract

This thesis is focused on PostgreSQL database system. You can find here introducing to temporal databases, database system PostgreSQL, proposal of temporal extension for PostgreSQL and implementation chapter with examples of using this extension. In this thesis is also using temporal database systems and use temporal databases in practise.

Klíčová slova

temporální databáze, čas platnosti, transakční čas, bitemporální tabulky, postgres, postgresql

Keywords

temporal database, valid time, transaction time, bitemporal tables, postgres, postgresql

Citace

Radek Jelínek: Temporální rozšíření pro PostgreSQL, diplomová práce, Brno, FIT VUT v Brně, 2015

Temporální rozšíření pro PostgreSQL

Prohlášení

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

.....
Radek Jelínek
26. května 2015

Poděkování

Děkuji RNDr. Marku Rychlému, Ph.D. za odborné vedení práce a cenné rady.

Obsah

Kapitola 1

Úvod

Temporální databáze jsou velmi užitečným prostředkem v oblasti databází. Využití temporálních databází může růst s rozvojem umělé inteligence, kdy můžeme analyzovat a zpracovávat data napříč časem. Umělá inteligence se využívá, mimo jiné, k dolování dat z databází, kde mít k dispozici minulost je jistě výhodou. Je mnoho možností využití temporálního databázového systému v různých oblastech. Zároveň tato oblast s sebou přináší z pohledu návrhu celého systému spousty zajímavých problémů, které nemusí být na první pohled zjevné.

Databázový systém PostgreSQL je hojně využíván a využívá jej například Yahoo!, OpenStreetMap, Skype a mnozí další. Jeho velkou výhodou a přínosem je, že je vydán pod open source licencí a je tudíž možno jej volně upravovat. Na PostgreSQL existuje několik přídavných modulů, které rozšiřují funkčnost databázového systému například o nové funkce, datové typy a podobně.

Cílem tohoto dokumentu je spojit open source projekt databázového systému PostgreSQL s temporálními databázemi a vytvořit tak temporální rozšíření právě pro PostgreSQL. V současné době existuje několik rozšíření PostgreSQL o temporální prvky jako je například modul přidávající datový typ `PERIOD`, ale neexistuje kompletní nebo alespoň částečné řešení podpory temporálních databází. PostgreSQL se stále aktivně vyvíjí a zasahovat do konkrétní verze a přepisovat ji pro přijímání temporálního SQL jazyka není moc vhodné. Cílem této práce je tedy navrhnout a implementovat temporální rozšíření tak, aby bylo funkční i na dalších verzí PostgreSQL a mohlo být dále rozšiřováno (poskytnuto jako open source).

1.1 Využití temporálních databází

Využití temporálních databází nalezneme v několika oblastech lidské činnosti některé aplikace temporálních databází můžeme vidět v následujícím výčtu.

- Finanční aplikace - uchovávání historie dat na burze
- Pojišťovací aplikace - která politika byla platná v určitém období
- Rezervační systém - kdy je který pokoj v hotelu rezervován
- Medicínské záznamy - záznamy o pacientech
- Rozhodovací systémy - plánování budoucnosti

Další oblasti použití bychom našli velmi snadno, použití temporálních dat si lze představit v mnoha odvětvích lidské činnosti. [2]

Kapitola 2

Základní pojmy

2.1 Netemporální databáze

Komerční DBMS (Database management system) jako je Oracle, Sysbase a další jsou určena k ukládání obrovského množství dat, která jsou obvykle považována, v daném okamžiku za platná. Data z minulosti nebo budoucnosti nejsou v těchto databázích uložena. Daty z minulosti myslíme ta data, která byla do databáze uložena dříve a později byla aktualizována nebo smazána. Stará data jsou obvykle přepsána novými údaji. Budoucností myslíme ta data, která budou platná, až v budoucnosti (v současné době platná nejsou).

DBMS ukládá tato data v relacích (tabulkách). Relační databáze obvykle obsahují množinu tabulek. Každá tabulka obsahuje n-tice (řádky) a atributy (sloupce).

Objektový přístup pracuje s databázemi, které obsahují množinu kolekcí. Každá kolekce obsahuje množinu objektů se stejnými atributy. Můžeme se setkat i s databázovými systémy, které jsou objektově relační, takovým systémem je PostgreSQL. Tento systém kombinuje relační přístup s objektovým.[2]

2.1.1 Snímková tabulka

Jedná se o běžnou tabulku v SQL databázi, která není temporální. Není tedy možné s ní pracovat jako s temporální tabulkou. [16]

2.2 Temporální databáze

Fakta v temporálních databázích jsou ukládány rozdílně oproti faktům v netemporálních databázích. K faktům v temporálních databázích je přidělena časová perioda (interval), která vyjadřuje platnost faktu nebo jeho existenci v databázi (v závislosti na typu temporální tabulky). Oproti netemporálním databázím je zde možné udržovat data, která nejsou platná v daném okamžiku, ale byla platná v minulosti nebo se teprve platnými stanou. Toto chování je možné díky přiřazení časové periody n-ticím. Časová perioda se označuje pojmem čas platnosti (popsáno v kapitole 2.5) nebo transakční čas (popsáno v kapitole 2.6). [2]

2.3 Temporální DBMS

Netemporální DBMS podporující temporální datové typy nelze označovat za temporální DBMS. Abychom mohli označit DBMS za temporální, tak by měl podporovat jazyk pro de-

finici temporálních dat (TDDL - Temporal Data Definition Language), jazyk pro manipulaci s temporálními daty (TDML - Temporal Data Manipulation Language), jazyk pro dotazování temporálních dat (TQL - Temporal Query Language) a temporální omezení (jako je temporální referenční integrita). Některé temporální DBMS jako například TimeDB přidávají nová klíčová slova SQL jako `VALIDTIME` a `TRANSACTIONTIME`. Pro ilustraci si uvedeme příklady jazyku SQL/Temporal používaného v TimeDB: [2]

- TDDL - vytvoříme tabulku zaměstnanci s časem platnosti (klíčové slovo `VALIDTIME`) i transakčním časem (klíčové slovo `TRANSACTIONTIME`) viz. příklad 1

```
CREATE TABLE Zaměstnanci (
      Id INTEGER,
      Jméno CHAR(50),
      Pozice CHAR(40),
      Plat INTEGER
) AS VALIDTIME AND TRANSACTIONTIME
```

Příklad 1: vytvoření tabulky

- TDML - v příkladě 2 vidíme vkládání záznamů s jiným časem platnosti, i když se jedná o stejného zaměstnance, údaje o jeho působení ve firmě jsou v čase různé

```
VALIDTIME PERIOD '1985-1990'
INSERT INTO Zaměstnanci VALUES (10, 'Karel', 'Vývoj', 11000);

VALIDTIME PERIOD '1990-1993'
INSERT INTO Zaměstnanci VALUES (10, 'Karel', 'Prodej', 11000);

VALIDTIME PERIOD '1993-forever'
INSERT INTO Zaměstnanci VALUES (10, 'Karel', 'Prodej', 12000);
```

Příklad 2: vložení do tabulky

- TQL - v příkladě 3 vidíme 3 dotazy, kde první dotaz vrátí historii zaměstnanců, druhý dotaz zobrazí, kdy byly n-tice vloženy do databáze a poslední dotaz vrátí kombinaci předchozích dotazů

```
VALIDTIME
SELECT * FROM Zaměstnanci;

TRANSACTIONTIME
SELECT * FROM Zaměstnanci;

VALIDTIME AND TRANSACTIONTIME
SELECT * FROM Zaměstnanci;
```

Příklad 3: výběr z tabulky

2.4 Uživatelský čas

Uživatelský čas se nevyskytuje pouze v temporálních databázích, ale setkáváme se s ním velmi hojně i v relačních databázích. Je obvykle vyjádřen datovými typy `DATE`, `TIME` a `TIMESTAMP`. V některých databázových systémech se můžeme setkat i s datovým typem

`INTERVAL`. Datový typ `DATE` obvykle nese datum s přesností na jeden den, typ `TIME` zase nese informaci o čase v jednotlivých dnech a jeho přesnost je v případě PostgreSQL jedna milisekunda. Časové razítko (`TIMESTAMP`) kombinuje datový typ `DATE` a typ `TIME` a určuje tedy přesné datum a čas. Speciálním datovým typem je potom datový typ `INTERVAL`, který udává časovou délku v předem definované přesnosti (roky, měsíce, dny, . . . , ale i kombinovaně jako roky a měsíce, dny a minuty a podobně) [5].

2.5 Čas platnosti

Čas platnosti (Valid-Time) je časové období, ve kterém je záznam v databázi platný. Je to tedy atribut udávající, v kterém intervalu platí daný záznam tabulky. Může a často se stává, že nevíme do kdy bude daný fakt platný, například nevíme, kterého dne přestane platit česká koruna, proto musíme mít možnost definovat čas platnosti od libovolného data navždy. V případě, že bude česká koruna nahrazena například eurem, pak již budeme znát datum, do kterého koruna platila. Používá se tedy v historii zachycujících databázích [2].

2.6 Transakční čas

Záznamy v relační tabulce jsou v průběhu času vkládány, editovány nebo mazány. Transakční čas vyjadřuje období, po které je fakt přítomen v databázi. Při editaci záznamu je ukončena platnost původního záznamu a je vytvořen nový záznam. Tento typ času nedefinuje uživatel, ale databáze samotná. Používá se v rollback databázích [2] (databáze, kde je možné vrátit se k předešlému stavu dat).

2.7 Referenční integrita

V kombinaci s tabulkami s časem platnosti, s tabulkami s transakčním časem, bitemporálními tabulky a běžnými databázovými tabulkami jsme nuceni řešit různé problémy s referenční integritou. V následujících bodech si probereme jednotlivé problémy a jejich řešení.

2.7.1 Nekorektní interval času platnosti

Ukazuje-li tabulka s časem platnosti do jiné tabulky s časem platnosti a uživatel se pokusí vložit do odkazující tabulky záznam, který nemá čas platnosti spadající do času platnosti odkazované tabulky. Nastává situace, kdy odkazující záznam je platný v čase, ve kterém není platný odkazovaný záznam. Vhodným řešením tohoto problému je zakázání těchto operací, aby nemohlo dojít k odkazování na nespecifikovanou hodnotu.[16]

2.7.2 Rozdělení záznamu v odkazované tabulce s časem platnosti

Tabulka s časem platnosti ukazuje do tabulky s časem platnosti a nastane rozdělení záznamu v odkazované tabulce na 2 a rozdělí se i interval na nové dva intervaly. Vhodným řešením je vytvořit tabulku referencí tak, že záznam z odkazující tabulky bude odkazovat na dva nové záznamy v odkazované tabulce.[16]

2.7.3 Spojení záznamů ve stavové tabulce

Existují dva záznamy v odkazující tabulce s časem platnosti se stejnými intervaly a každý z nich ukazuje do tabulky s časem platnosti. Dva odkazované záznamy mají stejný obsah včetně času platnosti. Proto dojde ke spojení odkazovaných záznamů. Vhodným řešením je úprava v tabulce referencí (uvedné v podsekci 2.7.2) tak, aby záznamy ukazovaly na jeden spojený záznam.[16]

2.7.4 Odkazovaná transakční tabulka

Snímková tabulka nebo tabulka s časem platnosti odkazuje do tabulky s transakčním časem. V tabulce s transakčním časem je ukončen záznam platnosti a odkazující tabulka ukazuje na neexistující záznam. Možností je zakázat tabulce bez transakčního času ukazovat do tabulky s transakčním časem nebo druhou možností je záznamy z odkazující tabulky fyzicky smazat. [16]

2.7.5 Odkazující transakční tabulka

Tabulka s transakčním časem ukazuje do snímkové tabulky nebo tabulky s časem platnosti. Dojde k fyzickému smazání záznamu v odkazované tabulce a v odkazující tabulce se ukončí transakční čas, ale zůstane odkaz na neexistující záznam. Řešením je tuto možnost zakázat a povolit pouze reference z tabulek s transakčním časem pouze do tabulek s transakčním časem.[16]

2.7.6 Odkazující tabulka má méně časových hledisek

Snímková tabulka ukazuje do tabulky s časem platnosti. Nastane situace, kdy skončí čas platnosti v odkazované tabulce a je vytvořen nový záznam. Snímková tabulka tedy musí ukazovat na dva záznamy. Vždy když má odkazující tabulka méně časových hledisek než odkazovaná nastává stejná situace. Řešením je zavedením tabulky referencí jako v bodech 2.7.2 a 2.7.3, čímž bude odkazující záznam ukazovat na dva.[16]

Kapitola 3

PostgreSQL

PostgreSQL je open-source objektově relační databázový systém podporující širokou škálu operačních systémů mezi nimiž je Windows, Linux, UNIX i Mac OS X. Plně podporuje ACID (atomičnost, konzistenci, izolovanost, trvalost). Podporuje většinu datových typů SQL:2008 mezi nimiž jsou i datové typy `DATE`, `INTERVAL` a `TIMESTAMP`. Má nativní programovací rozhraní i pro programovací jazyky Java a C/C++. Nejen díky nativní programovacímu rozhraní, ale i díky vlastnímu PL/pgSQL velmi podobnému Oracle PL/SQL, je PostgreSQL velmi přizpůsobitelné. Triggery a uložené procedury mohou být napsány v jazyce C a nahrány do databáze jako knihovna, což umožňuje vysokou flexibilitu. PostgreSQL zahrnuje také framework, který umožňuje vývojářům definovat a vytvořit vlastní datové typy podporující funkce a operátory, které definují jejich chování. [5]

3.1 Historie

PostgreSQL originálně nazvané jen Postgres, bylo vytvořeno profesorem Michaelem Stonebrakerem na University of California at Berkeley. Postgres se začal vyvíjet v roce 1986 jako nástupce projektu Ingres odtud také vznikl jeho název (Post + Ingres). Ingres byl vyvíjen od roku 1977 do roku 1985 podle klasického RDBMS (Relational database management system - databázový systém založený na relačním modelu [4]) konceptu. Vývoj Postgres byl od roku 1986 až do roku 1994 zaměřen na výzkum v oblasti objektově relačního modelu dat.

Výzkumný tým profesora Stonebrakera za 8 let vývoje představil mimo jiné i procedury, rozšiřitelné datové typy s indexy a objektově relační koncept. V roce 1995 byl dvěma studenty nahrazen Postgres' POSTQUEL (jazyk používaný v Postgres) za rozšířenou podmnožinu jazyka SQL a byl nazván Postgres95. O rok později se stal z Postgres95 open source projekt. [5]

3.2 Architektura PostgreSQL

3.2.1 Postmaster

Je víceuživatelský databázový server. Jeho životní cyklus začíná inicializací základních subsystémů, obnovení databáze do konzistentního stavu přes XLOG, připojení se k segmentu sdílené paměti (System V IPC) včetně inicializace sdílených datových struktur. Dále je proces rozdělen (je proveden `fork`) na *autovacuum launcher* (periodické spouštění čistících úloh), *stats daemon* (přes UDP sbírá statistiky běhu), *bgwriter* (zapisuje obsah vyrovnávací paměti na disk a vytváří checkpointy) a *syslogger* (sbírá a zapisuje do souboru logy ostatních

procesů). Nakonec proběhne navázání na TCP soket a naslouchání příchozího spojení. Pro každé nové spojení je vytvořen *backend* a je periodicky kontrolováno, zda procesy potomků jsou živé a v případě potřeby je provedeno jejich oživení nebo nahrazení.

Vnitřní procesorová komunikace probíhá především pomocí sdílené paměti jsou využívány, ale i signály, semafory a roury. *Stat collector* využívá UDP na *loopback* rozhraní. Výhodou tohoto systému je ochrana adres paměti (je obtížnější, aby jeden proces způsobil pád celého DBMS) a další výhodou je, že IPC a modifikace sdílených dat je přímá. Tento způsob má i své nevýhody, mezi ně patří statická velikost sdíleného prostoru, která je určena při startu a další je, že některé sdílené operace se provádí neefektivně (například vyhodnocení jednoduchého dotazu několika procesory). [11]

3.2.2 Backend

Postmaster přijme připojení a vytvoří nový *backend* se kterým probíhá komunikace s klientem. Nyní probíhá *frontend/backend* protokol, který se sestává z autentizace klienta, *Simple query protocol*, který se stará o přijetí, vykonání a navrácení výsledku dotazu. Po odpojení klienta *backend* končí. [11]

3.3 Průběh zpracování SQL dotazu

3.3.1 Parser

Skládá se z lexikální analýzy a parsování. Lexikální analýza provede rozdělení dotazu na sekvenci tokenů za pomoci nástroje GNU Flex. Parser sestaví ze sekvence tokenů abstraktní syntaktický strom (AST - abstract syntax tree) pomocí nástroje GNU Bison. Elementy abstraktního syntaktického stromu se nazývají *parse nodes*. Výstupem je seznam *parse nodes*, který odpovídá gramatice. [11]

3.3.2 Sémantická analýza

Výstup *parseru* je pouze AST a sémantická analýza musí být provedena v dalším kroku. Nejdříve se ověří existence odkazovaných schémat, tabulek a sloupců. Krokem číslo dva je kontrola konzistence typů použitých ve výrazu a na závěr se provede obecná kontrola chyb, které je nemožné detekovat nebo jsou jen velmi těžko detekovatelné v parseru. [11]

3.3.3 Rewriter a planner

Syntaktická analýza produkuje *query* (dotaz), který je reprezentován AST s dodatečnými informacemi. *Rewriter* aplikuje přepisovací pravidla včetně definic pohledů. Vstupem je tedy dotaz a výstupem je žádný nebo více dotazů (*Queries*). *Planner* potom vezme jednotlivé dotazy a vytvoří plán, který popisuje jak má být dotaz vykonán. Výstupem je *query plan*, který popisuje jednotlivé operace daného dotazu. Plánovač je potřebný pouze pro dotazy, které je možné optimalizovat. Příkazy které je možné optimalizovat jsou `INSERT`, `DELETE`, `SELECT` a `UPDATE`. Výstupem plánovače je *plan tree*, jehož jednotlivé uzly popisují fyzické operace jako může být spojení dvou relací, řazení, aplikace predikátu, prohledání indexu a další. [11]

3.3.4 Executor

Plan tree získaný z předchozího kroku popisuje datový tok mezi operacemi. *Executor* vezme tento *plan tree* a jeho rekurzivním prováděním získá požadované množiny řádků. Pro ilustraci si představme, že kořenovým uzlem je uzel `MergeJoin`. Před tím než může být samotné spojení dokončeno musejí být vybrány řádky. Řádky se vybírají v potomcích kořenového uzlu. *Executor* provede rekurzivní zavolání nad potomky současného uzlu (nejdříve na levý uzel). Nyní se stává vrcholový uzel levý uzel kořenu a ten například může znamenat `Sort`. Potomek uzlu `Sort` nám musí navrátit řádky k seřazení a může to být například uzel `SeqScan` reprezentující aktuální čtení tabulky. Vykonáním tohoto uzlu zapříčiní získání řádku a navrácení jej rodičovskému uzlu. Uzel `Sort` bude získávat postupně od svého potomka jednotlivé řádky, dokud nezíská všechny (uzel `SeqScan` navrátí `NULL` namísto řádku). Nyní už uzlu `Sort` nebrání nic v tom, aby řádky nemohly být seřazeny a navráceny ve správném pořadí. Seřazené řádky jsou v uzlu uloženy pro pozdější použití. Kořenový uzel (`MergeJoin`) si vyžádá řádek od pravého potomka, zjistí zda může být spojen s řádkem od levého potomka a spojený řádek vrátí volajícímu. V případě, že dva řádky nelze spojit požádá některého potomka o další řádek. Po vyčerpání všech potomků navrací `NULL`.

Složitější dotazy jsou prováděny stejným způsobem, že každý uzel navrací po zavolání další řádek nebo hodnotu `NULL`. Tento mechanismus je prováděn pro všechny základní SQL dotazy (`SELECT`, `INSERT`, `UPDATE` a `DELETE`). Pro `SELECT` je vykonávání stejné jak bylo popsáno výše. Kořenový uzel dotazu `INSERT` vrací řádek, který je vložen do databáze. Pro `UPDATE` je navrácen řádek, který zahrnuje všechny aktualizované řádky včetně TID (*tuple ID* - identifikace fyzického uložení řádku v dané tabulce), tento řádek je poté označen za smazaný a je vytvořen řádek nový s aktualizovanými hodnotami. Dotaz `DELETE` navrací z kořenového uzlu TID a řádky, které jsou takto identifikovány jsou označeny za smazané. [5]

3.4 Ukládání dat

V PostgreSQL jsou tabulky a indexy ukládány do běžných souborů. Každá tabulka nebo index je rozdělen do segmentů o maximálně 1GB. Každý soubor je rozdělen na bloky bajtů (v základním nastavení 8192 batů na blok), ve kterých jsou položky jako n-tice (v případě tabulek) nebo indexační záznamy (v případě indexů) spolu s metadaty. N-tice je jednoznačně identifikována trojicí (OID, číslo bloku, posunutí v bloku). [5]

Kapitola 4

Temporální databáze

Temporální databáze jsou implementovány pro několik komerčních i nekomerčních databázových systémů. Jednotlivé implementace se mnohdy liší jak přístupem, tak práci s temporálními daty. V následujících kapitolách si uvedeme některé implementace temporálních databázových systémů, nejedná se ale o výpis všech systémů, ale spíše o stručný přehled a popis jednotlivých přístupů. Mezi další databázové systémy zabývající se temporálními databázemi jsou například: ARCADIA, Calenda, HDBMS, TempCASE, TempIS, TIME-MULTICAL, VT-SQL, T-REQUIEM, T-squared DBMS [16].

4.1 ChronoLog

Je to temporální deduktivní databázový systém, který je zcela zdarma. Pracuje jako vrstva nad komerčním systémem od Oracle. Temporální požadavky napsané v příkazech ChronoLog jsou kompilovány do sekvence příkazů a následně vykonávány v Oracle databázovém systému. Při vývoji tohoto systému byl kladen velký důraz na uživatelsky přívětivý DML (Data Manipulation Language). Podporuje uživatelský čas, čas platnosti i transakční čas, integritní omezení, slévání, temporální operace jako join, negace, disjunkce, selekce a projekce. [10]

4.2 TimeDB

Jedná se o bitemporální relační databázový systém založený na SQL, který podporuje dotazovací jazyk, DML, DDL (Data Definition Language). Systém byl vyvinut Andreasem Steinerem v rámci jeho disertační práce. Tato práce obsahuje popis temporálního objektově relačního a temporálního objektově orientovaného databázového systému. Druhá verze TimeDB je založená na programovacím jazyce Java, používá JDBC a disponuje API (Application Programming Interface) rozhraním. Podporuje temporální dotazovací jazyk ATSQL2. ATSQL2 je výsledek třech rozdílných přístupů [16]:

- TSQL2 - temporální rozšíření SQL-92
- Chronolog - temporální deduktivní systém
- Bitemporal ChronoSQL - bitemporální dotazovací jazyk

TimeDB běží jako vrstva nad databázovým systémem Oracle. Jazyk ATSQL2 je kompilován do sekvence SQL-92 dotazů. [2]

4.3 Temporální databáze v Oracle 12c

Oracle 12c obsahuje rozšíření zvané *Temporal Validity*. S pomocí tohoto rozšíření lze do tabulky přidat jednu nebo více dimenzí času platnosti. V kombinaci s Flashback Data Archive (možnost navrácení se k nějakému předchozímu stavu databáze) lze vytvořit bitemporální nebo dokonce multi temporální databázi.

Flashback Data Archive umožňuje dát faktu interval po který byl uložen v databázi. Oracle definuje tento interval jako čas $t_1 \epsilon \langle start\_time_1, end\_time_1)$ na který navazuje interval $\langle end\_time_1, end\_time_2)$ příslušící novému faktu. Mezi dvěma fakty neexistuje žádná mezera což způsobuje o něco delší dotazy jako v příkladu 4, kde je zobrazena `WHERE` klauzule dotazu na současný stav. Oracle také používá `NULL` pro $-\infty$ i pro $+\infty$. Tímto mechanismem lze docílit transakčního času i když je jeho primárním účelem zálohování dat.

```
WHERE (vt_start IS NULL OR vt_start <= SYSTIMESTAMP)
      AND (vt_end IS NULL OR vt_end > SYSTIMESTAMP)
```

Příklad 4: Oracle WHERE [13]

V databázích Oracle lze tedy využívat temporální rozšíření, ale práce s ním není příliš jednoduchá a vcelku se liší od jiných temporálních databází jako například TimeDB. [16]

4.4 Teradata Temporal

Teradata nabízí kompletní bitemporální řešení. Je tu implementován datový typ `PERIOD`, který umožňuje zadávat periody nebo trvání v čase a umožňuje jejich manipulaci a porovnávání přímo v databázi namísto uživatelského SQL. Transakční čas i čas platnosti je reprezentován právě datovým typem `PERIOD`. Běžná tabulka, které jsou přidány sloupce `ValidTime`, `TransactionTime` nebo oba dva, se stává tabulkou temporální (popř. bitemporální) a databázový systém automaticky sám ví jak s těmito sloupci nakládat (podpora integritních omezení z hlediska času). [6]

4.5 IBM DB2 Universal Database

Je blízká SQL standardu SQL-92 podporující temporální datové typy. Podporované typy jsou `DATE`, `TIME` a `TIMESTAMP`, kde datový typ `TIME` je přesný na sekundy (je uložen jako celé šestimístné číslo). `TIMESTAMP` je přesný na 6 mikrosekund a je uložen jako desetiné číslo. `DATE` je definováno jako osmimístné číslo s intervalem rozsahu deset tisíc let. [15] Je tu i podpora pro transakční čas a čas platnosti. Temporální možnosti jsou podobné možnostem v Teradata Temporal.

Kapitola 5

TSQL2

Je to temporální rozšíření standardu SQL-92. Tento standard TSQL2 modifikuje a rozšiřuje. Má několik požadovaných rysů.

- nemá rozlišovat mezi identitou a ekvivalencí snímků databáze
- měl by podporovat pouze jednu dimenzi času platnosti
- čas platnosti podporuje časy v minulosti i v budoucnosti
- hodnoty časových razítek (`TIMESTAMP`) nejsou omezeny rozsahem ani přesností (SQL-92 omezuje čas na přesnost sekund a do roku 9999 našeho letopočtu)
- TSQL2 má být konzistentní rozšíření SQL-92
- TSQL2 má umožnit rozšíření tabulky na libovolné množině atributů
- TSQL2 by měl umožnit flexibilní temporální projekci, ale jeho syntaxe by měla odhalit pokud byly provedeny nestandardní temporální projekce
- operace v TSQL2 by neměly spoléhat na žádné explicitní vlastnosti atributů
- temporální podpora je volitelná (je to nutné pro zachování SQL-92 kompatibility)
- uživatelsky definovaná podpora času má zahrnovat okamžiky, intervaly a fixní a variabilní rozsahy
- existující agregační funkce by měly mít svoje TSQL2 verze (to je potřebné chcemeli aplikovat agregační funkce na temporální model dat)
- práce s časem by měla podporovat více jazyků a kalendářů (SQL-92 podporuje pouze gregoriánský kalendář a jeden formát času)
- mělo by být možné odvozovat temporální a snímkové tabulky z jiných temporálních a snímkových tabulek
- TSQL2 tabulky by mělo být možno implementovat v první normální formě
- TSQL2 musí mít efektivně implementovanou algebru umožňující optimalizaci a bude vycházet ze standardní algebry (klasické DBMS jsou založeny na standardní algebře)
- jazykový datový model má umožnit více reprezentací datových modelů

5.1 Časová ontologie

Model času v TSQL2 je z obou stran ohraničen. Není možné rozhodnout, zda je čas modelu diskrétní nebo spojitý. Namísto toho se zde setkáváme s pojmem instant (okamžik), který je ještě menší než chronon. Chronon je nejmenší jednotka, kterou může reprezentovat časové razítko, jeho velikost je implementačně závislá. Okamžik lze tedy reprezentovat pouze přibližně. Diskrétní reprezentace času závisí na granularitě (měřítko) časového razítka. Měřítko může být zadáno uživatelem nebo může být výchozí (například výchozí nastavení datového typu `TIMESTAMP` v PostgreSQL je 1 mikrosekunda [5]).

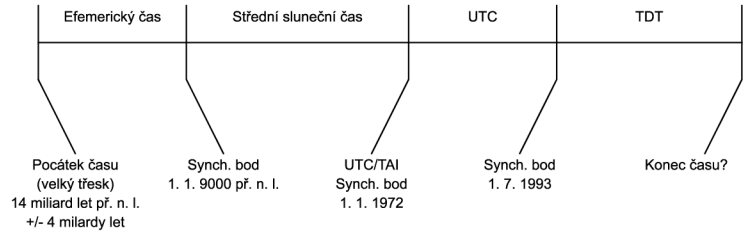
Okamžik je modelován na základě granularity datového typu tedy měsíce, dny, sekundy a podobně. Je možné modelovat i interval pomocí kompozice dvou okamžiků. [14] Intervaly mohou být uzavřené, otevřené nebo částečně otevřené podle toho zda krajní hodnoty do nich spadají nebo ne.

5.2 Základní hodiny

Základní hodiny musí být specifikovány u každého času, který je v databázi uložen. SQL-92 specifikuje, že hodiny jsou udávány v sekundách a jsou uvedeny v UTC (Coordinated Universal Time - koordinovaný světový čas) a je u nich buď implicitně nebo explicitně specifikována časová zóna [1].

UTC je nástupce GMT (Greenwich Mean Time – greenwichský střední čas) a je nezávislý na rotaci země (pokud pomineme dilatace času), protože je založen na atomových hodinách [3].

TSQL2 rozděluje čas na několik period které je možné vidět na obrázku 5.1. Jednotlivé synchronizační body zaznamenávají přechod do nové periody. První perioda začíná počátkem času (velký třesk) a trvá až do půlnoci prvního ledna roku 9000 před naším letopočtem. Toto období se nazývá efemerický (bezvýznamný) čas. Na toto období navazuje období středního slunečního času, který trvá až do půlnoci 1. ledna 1972, od kdy pokračuje UTC. UTC přichází právě v den, kdy byl tento čas synchronizován s atomovými hodinami a byl přijat systém přestupné sekundy (nepravidelná oprava UTC času o jednu sekundu). Konec tohoto systému se datuje k 1. červenci 1993 jednu sekundu před půlnocí. K tomuto novému času může být přidána přestupná sekunda (oznámení o přidání přestupné sekundy organizace International Earth Rotation Service). Od tohoto synchronizačního bodu mají běžet základní hodiny v TDT (Terrestrial Dynamic Time). TDT by mělo platit až do úplného konce času. [14]



Obrázek 5.1: Synchronizační body `TIMESTAMP` [14].

5.3 Datové typy

SQL-92 temporální typy jsou `DATE`, `TIME`, `TIMESTAMP` a `INTERVAL` [1]. TSQL2 zavádí datový typ `PERIOD`, který je určen počátečním a koncovým časem a přesností [16]. Přesnost může být definována pomocí celého čísla (počet dílčích číslic) nebo jemností (například milisekunda). Jsou připraveny operátory pro porovnávání časových razítek s různou přesností. Temporální hodnoty mohou být vkládány nebo získávány v uživatelsky definovaných formátech.

Datový typ `INTERVAL` je velmi rozdílný od datového typu `PERIOD`. Ačkoliv se může na první pohled zdát, že se jedná o vymezení časového období (s počátečním a koncovým časem), pravda je, že časové období popisuje pouze datový typ `PERIOD`. `INTERVAL` slouží k vyjádření délky trvání období. Budeme-li chtít zjistit interval mezi dvěma datovými typy `DATE`, dostaneme jako výsledek konkrétní délku tohoto období (například s přesností na měsíce dostaneme výsledek 5 měsíců).

Novým a důležitým datovým typem v TSQL2 je typ `SURROGATE`. Jedná se o unikátní datový typ, který může být porovnán na shodu, ale jeho hodnota nemůže být zobrazena uživateli. `SURROGATE` je tedy "čistá" identita, která nepopisuje vlastnost. Tento typ je užitečný pro identifikaci objektů s časově variabilními atributy. [14]

5.4 Časové osy

V TSQL2 jsou podporovány 3 časové osy. Uživatelský čas, transakční čas a čas platnosti. Temporální hodnoty je možné porovnat napříč časovými osami s příslušnou přesností. Transakční čas je ohraničen počátkem, kdy byla databáze vytvořena až do doby její změny. Uživatelský čas a čas platnosti mají dvě speciální hodnoty a to začátek ($-\infty$) a věčnost ($+\infty$), kde tyto hodnoty jsou maximy při porovnávání. Transakční čas má také definovánu jednu speciální hodnotu a tou je: dokud nebylo změněno.

Uživatelský čas a čas platnosti mohou být dočasně nespecifikovány. Víme že fakt se stal, ale nevíme kdy.

Temporální hodnoty mohou být určeny relativně k danému okamžiku. Příkladem je "now - 1 day", což je interpretováno (pokud vykonáme příkaz ve dne 2. 1. 2010) je výsledná hodnota 1. 1. 2010. [14]

5.5 Agregační funkce

Běžné SQL-92 agregační funkce jsou rozšířeny pro přijímání temporální dat. Jsou také rozšířeny, tak aby vracely časově proměnná data a povolovaly *temporal grouping*. Některé hodnoty mohou být, během výpočtu agregačních funkcí, váženy vzhledem k jejich trvání. Je přidána nová agregační funkce `RISING` vracející nejdelší rostoucí posloupnost [16].

5.6 Tabulky s časem platnosti

Snímkové tabulky, které jsou podporovány standardem SQL-92, jsou dostupné i v TSQL2. Jsou navíc ještě specifikovány stavové tabulky. V těchto tabulkách je každé n-tici přiřazen temporální prvek.

Například máme tabulku Zaměstnanci a v ní atributy Jméno, Plat a Manažer. Řekněme, že tabulka obsahuje n-tici (Karel, 22000, Marie). Temporální prvek této n-tice bude obsahovat souvislé (nespojité) intervaly, ve kterých měl Karel plat 22000 a jeho manažerem

byla Marie. Všechny ostatní informace o tom, že měl Karel jiný plat nebo jiného manažera budou zaznamenány v jiných n-ticích. Časové razítko je implicitně spojeno s danou n-ticí, ale není dalším sloupcem v tabulce.

TSQL2 ještě specifikuje událostní tabulky. V těchto tabulkách je každé n-tici přidána množina okamžiků.

Jako příklad si uveďme tabulku Pozice s atributy Jméno a Funkce. Tabulka může obsahovat n-tici (Marie, manažer). K této n-tici je přiřazeno časové razítko, které vyjadřuje okamžik, kdy se Marie stala manažerem. Informace o jejich dalších pozicích budou uloženy v jiných n-ticích. Čas je tu implicitně spojen s každou n-ticí. [14]

5.7 Tabulky s transakčním časem a bitemporální tabulky

Transakční čas n-tice je specifikace toho, kdy byla n-tice vložena do databáze. Pokud například vložíme do databáze n-tici (Karel, 22000, Marie) 1. 1. 2014 a odstraníme ji 3. 7. 2014 bude k dané n-tici přiřazen interval 1.1.2014 - 3.7.2014.

Bitemporální tabulka kombinuje tabulku s transakčním časem s tabulkou s časem platnosti. Můžeme mít dva typy bitemporálních tabulek a to bitemporální stavovou tabulku a bitemporální událostní tabulku. [14]

5.8 Vacuuming

Máme-li obsáhlou databázi s transakčními nebo bitemporálními tabulkami tak si musíme uvědomit, že v těchto tabulkách nikdy nedochází k fyzickému mazání záznamů. Záznamy v těchto typech tabulek jsou pouze označeny za smazané, ale v databázi nadále zůstávají. Tabulky svoji velikost nikdy nesníží a mohou ji pouze zvýšit. Proto zavedeme možnost čištění dat (vaccuming). Jedná se o specifikaci, kdy data, která jsou starší než předem definovaný časový okamžik z databáze odstraníme [17].

5.9 Specifikace schématu

`CREATE TABLE` a `ALTER` jsou rozšířeny o povolení specifikování temporálních aspektů. Jedná se o specifikaci tabulky s transakčním časem nebo časem platnosti. Při specifikaci nebo změně může být také zadáno měřítko. [14]

5.10 Restruktualizace

Klauzule `FROM` povoluje v TSQL2 povoluje tabulkám restruktualizaci tak, že temporální elementy, které jsou asociovány s n-ticemi a které mají stejné hodnoty, mohou být sloučeny (někdy označováno jako slévání). Budeme-li mít tabulku Zaměstnanci s n-ticemi (Karel, 22000, Marie) a (Karel, 15000, Marie) můžeme provést restruktualizaci nad atributy Jméno a Manažer a tím získáme n-tici (Karel, Marie) s temporálním prvkem, který je složený z dvou temporálních prvků. [14]

5.11 Temporální selekce

Čas platnosti i transakční čas může hrát roli v predikátu klauzule `WHERE`. Pro čas platnosti je možné použít `VALID()` (aplikované na jméno tabulky) pro transakční čas zase `TRANSACTION()`. Operátory jsou rozšířeny tak, aby mohli pracovat s temporálními argumenty. [16, 14]

Aktualizace je rozšířena podobně jako příkaz `SELECT` pro specifikaci temporálních rozsahů při aktualizaci.

5.12 Kompatibilita s SQL-92

Některé aspekty TSQL2 jsou čistými rozšířeními SQL-92. Uživatelsky definovaný čas v TSQL2 je nahrazením času z SQL-92, aby byly podporovány různé kalendáře a textové reprezentace času. Práce s temporálními prvky je na základě SQL-92. Všechny výrazy a příkazy SQL-92 je možné provádět v TSQL2. [16]

Kapitola 6

Návrh implementace

Abychom splnili požadavek na přenositelnost temporálního rozšíření PostgreSQL do dalších verzí jsme omezeni na používání prostředků systému PostgreSQL. Mezi tyto prostředky patří triggery, funkce, procedurální jazyk PL/pgSQL a podobně. Nemožnost zasáhnout z důvodu podpory dalších verzí, do syntaktické a sémantické analýzy zpracování dotazů není možné implementovat jazyk TSQL2. Temporální funkčnost tedy bude zajištěna sadou funkcí, triggerů a jiných možností PostgreSQL. Velmi důležitým implementačním bodem musí být zajištění referenční a entitní integrity, tímto zajistíme, aby databáze neobsahovala nekonzistentní data vložená chybnou prací s temporální tabulkou.

6.1 Datové typy

Nové datové typy, které přidává TSQL2 oproti stávající implementaci PostgreSQL (ve verzi 9.3) jsou `PERIOD` a `SURROGATE`. Existuje implementace datového typu `PERIOD` od autora Jeffa Davisa, která je open-source [12]. Implementace je velmi vhodná pro tento projekt a v projektu bude využita. Tato implementace používá k vytvoření indexu datovou strukturu b-strom a index GiST. Datový typ `PERIOD` dle této implementace počítá s částečně otevřeným časovým intervalem (zprava otevřený). Je možné zadat jakýkoliv interval ať už zprava otevřený, zleva otevřený nebo uzavřený a ten je na pozadí převeden na zprava otevřený časový interval. S tímto intervalem se dále lépe pracuje.

Datový typ `SURROGATE`, také není nutno implementovat. PostgreSQL sice, žádný takovýto datový typ nenabízí, ale je zde možnost práce s OID (Object Identifier). OID je jednoznačný identifikátor, tento pojem se používá v objektovém programování, kde jednoznačně identifikuje objekt. PostgreSQL je objektově relační databázový systém a umožňuje upravit tabulku tak, aby se k ní přidal systémový sloupec s hodnotou OID jednoznačně identifikující řádek tabulky. Pro dodatečné přidání referencí na jiné tabulky budou implementovány speciální funkce.

6.2 Temporální tabulky

Vytvoření temporální tabulky se bude skládat ze dvou kroků. Uživatel databáze nejdříve vytvoří snímkovou tabulku. Na vytvořenou tabulku aplikuje funkci s jménem tabulky jako parametrem pro vytvoření temporální tabulky. Podle aplikované funkce se tabulka upraví na bitemporální tabulku, tabulku s transakčním časem nebo tabulku s časem platnosti. Kroky k vytvoření těchto tabulek budou stejné s rozdílem ukládání hodnot o typu temporální

tabulky. Pro temporální data se vytvoří nová tabulka s příslušnými sloupci. Nová tabulka zajistí skrytí temporálních dat pro uživatele. Původní (upravovaná) tabulka se rozšíří o OID, který bude plnit funkci datového typu `SURROGATE` a bude přes něj vytvořena vazba do tabulky s temporálními údaji. Reference původní tabulky (které bude možno vytvořit a upravovat pouze před vytvořením temporální tabulky) na referencované tabulky, budou převedeny na reference přes tabulku s temporálními daty. Informace o temporální tabulce se zaznamená do seznamu (tabulky) všech temporálních tabulek v dané databázi.

6.3 Referenční integrita

Jak bylo popsáno v sekci 2.7, nastává mnoho situací, které je nutné řešit. Návrh musí zabezpečit kontrolu referenční integrity ihned při provádění dotazu, který by mohl integritu porušit. Mezi dotazy, které by mohly porušit referenční integritu je `INSERT`, `UPDATE`, `DELETE`. Z tohoto důvodu budeme integritu kontrolovat v triggerech a pomocných funkcí a na základě jednotlivých případů (včetně řešení), popsaných v sekci 2.7, budeme referenční integritu řešit. Řešení problému s odkazovanou transakční tabulkou v podsekci 2.7.4 jsou dvě. My budeme problém řešit zakázáním tabulce bez transakčního času ukazovat do tabulky s referenčním časem. Z tabulek s transakčním časem se záznamy fyzicky nemažou, ale pouze se ukončuje jejich platnost, což je důvodem zvolení tohoto řešení.

6.4 Práce s daty

Vkládání dat do tabulky bude povoleno běžným způsobem, bez možnosti definovat transakční čas nebo čas platnosti. V případě transakčního času bude triggery zajištěna jeho definice (vkládání znamená transakční čas od daného okamžiku navždy). Bude-li se jednat o tabulku s časem platnosti tak čas platnosti bude definovaný od záporného nekonečna do kladného nekonečna. Uživatelem definovaný čas platnosti při vkládání bude možný s pomocí funkce (s dvěma parametry), kde první parametr bude samotný SQL dotaz na vložení dat do tabulky a druhým parametrem bude čas platnosti vkládaného záznamu.

Aktualizace záznamů v tabulce s transakčním časem bude shodná jako jakýkoliv jiný `UPDATE` dotaz. V tomto případě vše zajistí databázový trigger. Dojde k replikaci starého záznamu a ukončení jeho transakčního času a aktualizaci nového s příslušným transakčním časem. U tabulky s časem platnosti bude řešení zase o trochu komplikovanější, protože budou muset být definovány funkce pro vykonání `UPDATE` dotazu a nastavení času platnosti. Funkce bude pro databázové uživatele z pohledu používání, podobná jako v případě vkládání.

Smazání dat z tabulky s časem platnosti probíhá běžným způsobem navíc se smazáním dat z pomocných temporálních tabulek. Data z tabulek s transakčním časem (nebo bitemporální tabulky) nejsou fyzicky mazána jen je pomocí triggeru ukončen transakční čas.

6.5 Výběr dat z temporálních tabulek

Při výběru dat z tabulek se bude provádět velmi důležitá vlastnost temporálních databází, kterou je slévání (restruktualizace). Vzhledem k tomu, že jsme omezeni pouze na využívání možností databázového systému PostgreSQL, jedná se o složitý úkol. V klauzuli `WHERE` dotazu `SELECT` se může vyskytovat porovnání času platnosti s hodnotou sloupce. Abychom

mohli zajistit tuto možnost, dáme v případě SQL dotazu `SELECT` možnost přistoupit k temporálním sloupcům (jejíž jména budou předem definovány, aby nedošlo k duplicitě sloupců). Uživatel databáze tyto sloupce zpřístupní voláním funkce, která ve svém názvu ponese název tabulky. Tato funkce bude zastupovat tabulku. Návratová hodnota funkce bude tabulka rozšířená o daný časový údaj. Restruktualizace se bude provádět v závislosti na parametru, který bude do funkce vstupovat. Příklad si ukážeme na jednoduchém dotazu

```
SELECT * FROM sequencedTabulka(true); ,
```

kde `true` bude znamenat, že výsledná tabulka bude restruktualizována.

I když nebudeme moci uživateli explicitně zakázat, aby si sloupec sám připojil k tabulce, tak mu bude poskytnuta tato funkce, aby nevznikala potřeba k přímému přístupu ke sloupci. Podobně jako v získávání záznamů z jednotlivých tabulek budeme získávat záznamy ze spojených tabulek tzn. přes předem připravenou funkci s názvem obsahující jména obou (nebo i několika) tabulek.

Kapitola 7

Implementace

V této kapitole budou probrány jednotlivé příklady práce s temporálním rozšířením pro PostgreSQL. Příklady budou demonstrovat funkčnost tohoto rozšíření a ukazovat možnosti použití. V následujícím textu budeme slovem transakce rozumět logické trvání záznamu v databázi. Začátkem transakce budeme rozumět časový okamžik, kdy byl záznam do databáze vložen a ukončením transakce budeme rozumět, kdy došlo ke smazání záznamu. S touto terminologií se budeme setkávat u tabulek s transakčním časem a u tabulek s časem platnosti. V této kapitole budeme pracovat s dvěmi jednoduchými tabulkami, kde první nese název `zamestnanec` a obsahuje jediný sloupec `jmeno` a druhá tabulka `pozice` se sloupcem `popis`.

7.1 Konvence

Abychom mohli udělat temporální rozšíření pro PostgreSQL, potřebujeme několik pomocných funkcí, vlastních datových typů, tabulek pro metadata, triggerů a podobně. Vše však potřebujeme uchovávat ve stejné databázi spolu i s uživatelem vytvořenými tabulkami, funkcemi a jinými databázovými prvky. Z tohoto důvodu potřebujeme zavést konvence pro pojmenovávání těchto databázových prvků a to tak, aby uživatele omezovaly co nejméně nebo v ideálním případě neomezovaly vůbec. Minimálního omezování v pojmenovávání jak funkcí, tak i jiných databázových prvků dosáhneme předponou před každým vlastním názvem, který souvisí s temporálním rozšířením. Předpona byla zvolena jako řetězec `"t__"`, kde písmeno t vyznačuje, že se jedná o temporální rozšíření a je následováno dvěmi podržítky. Dvě podtržítka jsou zvoleny, protože není obvyklé, aby uživatel pojmenovával například tabulku tak, že bude obsahovat právě dvojité podrtžítko. Obecně platí, že funkce, tabulky a podobně nejsou určeny k použití uživateli. Funkce bez zmíněné předpony jsou určeny k použití uživatelem a budou představeny v dalších kapitolách.

Snímkové tabulky, ze kterých jsou vytvořeny temporální tabulky, zůstávají v nezměněné podobě a vytváří se k nim tabulky nesoucí temporální data. Tyto tabulky obsahují sloupce `row_oid`, který jednoznačně identifikuje záznam, ke kterému jsou přiřazena temporální data a sloupce `valid_time` (tabulky s časem platnosti) nebo `transaction_time` (tabulky s transakčním časem) popřípadě oba dva sloupce (bitemporální tabulky). Tabulka má svůj název složen z předpony, názvu tabulky, ke které temporální data náleží a příponou `"__temp_data"`. Výsledný název pro tabulku například s názvem `zamestnanec` pak je `t__zamestnanec__temp_data`.

Tabulky s cizími klíči pro svázání dvou temporálních tabulek, budou pojmenovány jmény vazbených tabulek oddělených podržíkem, kde na prvním místě je tabulka z které se od-

kazujeme (referenční) a na druhém místě je tabulka odkazovaná (referencovaná) a přípona této tabulky je "__ct". Pokud tedy máme referenci z tabulky zamestnanec do tabulky plat pak vazební tabulka má název t__zamestnanec_plat__ct.

Komplikovanější je to s pojmenováváním triggerů a pravidel (rule). K triggerům jsou přiřazeny ještě funkce, které jsou vykonány v případě spuštění triggeru. Pojmenování těchto funkcí je zvoleno tak, aby bylo přehledné pro zpětné dohledání a případné úpravy v kódu temporálního rozšíření. Protože trigger nebo rule jsou vytvářeny jako spouštěče k nějaké uživatelské akci jako je výběr dat, vložení, aktualizace nebo mazání, tak zavedeme zkratky pro klíčová slova používaná při vytváření triggerů nebo pravidel. Zkratky zobrazuje tabulka 7.1. Zřetězením těchto zkratek pak vytvoříme příponu pro daný spouštěč. Například mámeli trigger "AFTER DELETE"pro tabulku zamestnanec bude výsledný název triggeru t__zamestnanec__tad a funkce s předpisem pro tento trigger t__zamestnanec__ftad. Budemeli mít například rule "DELETE DO INSTEAD"pak bude název pro t__zamestnanec__riod.

Klíčové slovo	Zkratka
AFTER	a
BEFORE	b
DO INSTEAD	io
INSERT	i
UPDATE	u
DELETE	d
TRIGGER	t
RULE	r
FUNCTION	f

Tabulka 7.1: tabulka zkratek klíčových slov SQL jazyka

Poslední částí je pojmenovávání vlastních typů. Typy budeme potřebovat pro spojování temporálních tabulek. Mohou být tři varianty spojení tabulek a to "t__temporal_sequenced_", "t__temporal_nonsequenced_"a "t__temporal_snapshot_"za kterými jsou přikonkatenovány názvy tabulek oddělené znakem "_". Pokud bychom měli datový typ nad tabulkami zamestnanec se sloupcem jmeno a tabulkou pozice se sloupcem popis byl by nad nimi vytvořen typ například t__temporal_sequenced_zamestnanec_pozice a jeho položky (sloupce) by pak měli názvy zamestnanec_jmeno a pozice_popis, kde si můžeme všimnout, že název tabulky slouží jako předpona před názvem sloupce. Datový typ může obsahovat ještě položku valid_time.

7.2 Tabulky s metadaty

Abychom mohli uchovávat informace o temporálních tabulkách, musíme zavést několik tabulek, které budou sloužit k uchování metadat. Mezi tyto tabulky patří t__reference_integrity, t__join a t__infotable. Poslední zmíněná t__infotable uchovává názvy temporálních tabulek a k nim booleovské hodnoty, zda se jedná o tabulku s transakčním časem, časem platnosti nebo bitemporální tabulku. V tabulce 7.2 je vidět jak mohou data uvnitř této tabulky vypadat (vyjma sloupce poznámka).

table_name	valid_time	transaction_time	poznámka
tabulka1	false	true	tabulka s časem platnosti
tabulka2	true	false	tabulka s transakčním časem
tabulka3	true	true	bitemporální tabulka

Tabulka 7.2: informace o temporálních tabulkách

Pro ukládání referencí slouží tabulka `t__reference_integrity`, kde jsou záznamy o tom, která tabulka je referenční a která je referencovaná jako je vidět v tabulce 7.3. Vazební tabulka obsahuje dva sloupce, kde první určuje záznam v referenční tabulce a druhý sloupec záznam v tabulce referencované. Za pomoci zavedených konvencí a údajů z tabulky 7.3 můžeme získat název vazební tabulky.

id	reference_table_name	referencing_table_name
1	referencni	referencovana1
2	referencni	referencovana2

Tabulka 7.3: informace o referenční integritě

V temporálním rozšíření lze vytvářet funkce pro pozdější získávání spojených tabulek. Názvy těchto funkcí jsou uloženy v tabulce `t__join` a lze je vypsat pomocí funkce `getTemporalJoinFunctions()`. Tabulka obsahuje, mimo názvů funkcí pro "sequenced"nebo "nonsequenced"dotazy, i pole s názvy tabulek, kterých se spojení týká. V tabulce 7.4 je vidět tabulka s názvy funkcí (tabulka `zamestnanec` je referenční a tabulky `plat` a `pozice` jsou referencované), kde ke každému spojení existuje vždy "sequenced"a "nonsequenced"varianta a v druhém sloupci je seznam tabulek, nad kterými je vytvořené spojení. Při smazání spojení nad konkrétními tabulkami se maže jak "sequenced"varianta, tak "nonsequenced"varianta. Automaticky s vytvořením referencí se vytvoří i funkce pro spojení dvou tabulek.

function_name	table_name
sequencedJoinZamestnanecPlatPozice	{zamestnanec,plat,pozice}
nonsequencedJoinZamestnanecPlatPozice	{zamestnanec,plat,pozice}
sequencedJoinZamestnanecPlat	{zamestnanec,plat}
nonsequencedJoinZamestnanecPlat	{zamestnanec,plat}
sequencedJoinZamestnanecPozice	{zamestnanec,pozice}
nonsequencedJoinZamestnanecPozice	{zamestnanec,pozice}

Tabulka 7.4: názvy funkcí spojených tabulek

7.3 Datové typy

V databázovém systému PostgreSQL jsou k dispozici typy podporující časy, kterých využívá i datový typ `PERIOD`. Z těchto typů budeme využívat především datový typ `TIMESTAMP`, ale PostgreSQL disponuje i typy `DATE`, `TIME` a `INTERVAL`. [9]

7.3.1 PERIOD

Jak už bylo zmíněno v návrhu řešení pro datový typ period byla zvolena existující implementace od Jeffa Davise. Tento typ je definován jako období mezi dvěmi časovými okamžiky, které jsou specifikovány PostgreSQL datovým typem `TIMESTAMP`. Datový typ `TIMESTAMP` má rozsah od roku 4713 před naším letopočtem až do roku 294276 našeho letopočtu. Pro transakční čas je rozsah plně postačující a pro čas platnosti není rozsah výrazně omezující. Rozlišení tohoto datového typu je 1 mikrosekunda, čehož využívá datový typ `PERIOD`, protože všechny intervaly včetně uzavřených a otevřených jsou převedeny na intervaly z prava otevřené. Například periodu zadánu uzavřeným intervalem jako

```
period('[2015-01-01 13:00:00, 2015-03-07 12:00:00]'),
```

je interně převedena na z prava otevřenou periodu periodu

```
period('[2015-01-01 13:00:00+01, 2015-03-07 12:00:00.000001+01)').
```

Zadávámeli periodu bez závorek určující otevřenost nebo uzavřenost intervalu jako

```
period('2015-01-01 13:00:00', '2015-03-07 12:00:00'),
```

pak je tato perioda shodná s periodou

```
period('[2015-01-01 13:00:00+01, 2015-03-07 12:00:00+01)').
```

Na zmíněném příkladu si můžeme povšimnout, že zatímco v prvním případě časový okamžik '2015-03-07 12:00:00' patří do intervalu, ve druhém případě již nepatří.

7.3.2 SURROGATE

Datový typ `SURROGATE` jednoznačně identifikuje záznam v temporální tabulce, nelze jej změnit, ale pouze vytvořit a jsou umožněny operace s jeho porovnáváním. K tomuto účelu se nám hodí objektový identifikátor databázového systému PostgreSQL. Každá tabulka vytvořená s povolenými OID (numeric Object IDentifier) [7] má jednoznačně identifikované záznamy. OID je uloženo ve sloupci `oid`, který ale není standartně zobrazován dotazem typu

```
SELECT * FROM zamestnanec;,
```

ale musíme si jej explicitně vynutit dotazem typu

```
SELECT oid, * FROM zamestnanec;.
```

Všechny vytvořené temporální tabulky budou rozšířeny o tento objektový identifikátor (OID) a bude zvolen jako primární klíč. Pokud budeme vytvářet povolené reference, které zahrnují, ať už v pozici referenční nebo referencované, snímkovou tabulku, pak bude tato snímková tabulka rozšířena o objektový identifikátor. Objektové identifikátory jdou vkládány do vazebních tabulek, proto je musí mít i snímkové tabulky, které jsou v referenčním vztahu s temporální tabulkou. OID bude zpřístupněn i v pomocných tabulkách obsahující čas platnosti pro záznamy v tabulce bitemporální nebo tabulcách s časem platnosti. Budeme-li tedy mít tabulku s časem platnosti a názvem `zamestnanec`, bude mít sloupec `oid` stejně jako pomocná tabulka `t__zamestnanec__temp_data`. U pomocných tabulek bude OID využíváno k jednoznačné identifikaci při aktualizaci času platnosti, detailněji bude účel popsán v kapitole 7.7.

7.4 Výběr dat

7.4.1 Výběr dat z tabulek s časem platnosti

Výběr dat z tabulek s časem platnosi je možné dvěmi základními způsoby. Prvním způsobem je získávat data pomocí takzvané "sequenced"varianty, kdy máme k dispozici sloupec `valid_time`, který je samozřejmě typu `PERIOD` a je možné s ním jakkoliv manipulovat. Druhým způsobem je využití varianty "nonsequenced", kdy sloupec s časem platnosti není vybrán, ale je možné s ním manipulovat (například filtrovat vybrané záznamy). Funkce na výběr těchto dat jsou pojmenovány jako `sequenced` nebo `nonsequenced` a je k nim přikonkatenován název tabulky. Pro tabulku `zamestnanci` pak můžeme použít dotaz

```
SELECT * FROM sequencedZamestnanec();,
```

kterým získáme tabulku 7.5. Pokud bychom chtěli, aby časy platnosti byly restruktuali-

jmeno	valid time
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)
jan novak	[1995-01-01 00:00:00+01, 2005-01-01 00:00:00+01)
jan novak	[2010-01-01 00:00:00+01, 2015-01-01 00:00:00+01)

Tabulka 7.5: nerestruktualizovaná tabulka zamestnanec

zovány, pak nastavíme jediný parametr funkce na hodnotu `true` (ve výchozím stavu je parametr nastaven na hodnotu `false` a nemusí se uvádět). Dotaz pak může vypadat jako

```
SELECT * FROM sequencedZamestnanec(true);
```

a k němu příslušná tabulka 7.6, kterou získáme z původní tabulky 7.5.

jmeno	valid time
jan novak	[1990-01-01 00:00:00+01, 2005-01-01 00:00:00+01)
jan novak	[2010-01-01 00:00:00+01, 2015-01-01 00:00:00+01)

Tabulka 7.6: restruktualizovaná tabulka zamestnanec

Nevýhodou restruktualizace (slévání) časů platnosti v tomto temporálním rozšíření je, že slévání probíhá u záznamů, které mají shodné hodnoty pro všechny sloupce dané tabulky (vyjma sloupce s časem platnosti). Není tedy možné nechat si restruktualizovat časy platnosti jen nad některými sloupci tabulky.

Varianta "nonsequenced"nemá možnost slévání, ale jako parametr těchto funkcí vstupuje řetězec, který bude vložen do kaluzule `WHERE` před vykonáním dotazu. V tomto řetězci můžeme využívat sloupec `valid_time` a provádět nad ním porovnávání. Dotaz

```
SELECT * FROM nonsequencedZamestnanec(
    'contains(valid_time, ''2003-01-01''::timestamp)'
);
```

nám vrátí tabulku 7.7.

jmeno
jan novak

Tabulka 7.7: "nonsequenced"tabulka zamestnanec

7.4.2 Výběr dat z tabulek s transakčním časem

Transakční tabulky jsou tabulky, u kterých máme ke každému záznamu i časový údaj, kdy byl záznam do tabulky vložen a kdy byl smazán. Máme tedy k dispozici historii dat jak byly v tabulce uloženy. Obvykle nás zajímá jak databáze vypadala v určitý časový okamžik v historii a k tomu je v tomto temporálním rozšíření k dispozici "snapshot"funkce. Ke každé tabulce s transakčním časem je takováto funkce vytvořena a její název nese i jméno tabulky. Název funkce je vytvořen konkatenací slova `snapshot` s názvem tabulky. Pro tabulku `zamestnanec` je předpis funkce `snapshotZamestnanec(TIMESTAMP)`, kde jediný parametr funkce je časové razítko pro které nás zajímá stav databáze. Pro tabulku 7.8 dostaneme po vykonání dotazu

```
SELECT * FROM snapshotZamestnanec('2015-04-23 00:58:20'::timestamp);
```

záznamy zobrazené v tabulce 7.9. Zde můžeme vidět, že v daném okamžiku byl v databázi pouze záznam 'jan novak'.

jmeno	transaction
john doe	[2015-04-23 00:58:47+02, infinity)
jan novak	[2015-04-23 00:58:12+02, 2015-04-23 00:58:47+02)

Tabulka 7.8: transakční tabulka zamestnanec

jmeno
jan novak

Tabulka 7.9: transakční tabulka zamestnanec v čase '2015-04-23 00:58:20'

7.4.3 Výběr dat z bitemporálních tabulek

Pro výběr dat z bitemporálních tabulek jsou připraveny stejné funkce jako pro výběr dat z tabulek s časem platnosti. Pro tabulku `zamestnanec` tedy budememe mít "sequenced"a "nonsequenced"variantu s tím rozdílem, že funkce mají o jeden parametr navíc, který udává, jako v případě tabulek s transakčním časem, časový okamžik, ve kterém se na data dotazujeme. Můžeme tedy získat obraz databáze k určitému datu. Dotaz

```
SELECT * FROM sequencedZamestnanec(true, '2015-04-23 19:09:30'::timestamp);
```

vybere z tabulky 7.10 data, která byla uložená v databázi v čase '2015-04-23 19:09:30' což jsou data v tabulce 7.11

jmeno	valid time	transaction
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-23 19:08:47+02, infinity)
jan novak	[1995-01-01 00:00:00+01, 2005-01-01 00:00:00+01)	[2015-04-23 19:09:01+02, infinity)
jan novak	[2010-01-01 00:00:00+01, 2015-01-01 00:00:00+01)	[2015-04-23 19:09:09+02, infinity)

Tabulka 7.10: bitemporální tabulka zamestnanec

jmeno	valid time
jan novak	[1990-01-01 00:00:00+01, 2005-01-01 00:00:00+01)
jan novak	[2010-01-01 00:00:00+01, 2015-01-01 00:00:00+01)

Tabulka 7.11: bitemporální tabulka zamestnanec v čase '2015-04-23 19:09:30'

Podobně funguje i druhá varianta, kde dotaz může vypadat jako

```
SELECT * FROM nonsequencedZamestnanec(
    'contains(valid_time, ''1995-08-08''::timestamp)',
    '2015-04-23 19:09:30'::timestamp,
);,
```

a k němu výsledná tabulka 7.12.

jmeno
jan novak
jan novak

Tabulka 7.12: nonsequenced dotaz v čase '2015-04-23 19:09:30'

7.5 Spojování tabulek

7.5.1 Vytvoření nového spojení

V databázovém systému PostgreSQL neexistuje mechanismus, kterým bychom mohli jako návratovou hodnotu z funkce, navrátit tabulku s předem nespecifikovanými datovými typy sloupců ani jejich počtem. S temporálním rozšířením pro PostgreSQL zavádíme možnost registrace funkce pro spojení tabulek. Jedná se o funkci `addTemporalJoin(TEXT[])`, která jako parametr přijímá pole názvů tabulek v textové podobě, nad kterými budou vytvořeny funkce, vracející spojenou tabulku. Máme-li tabulky `zamestnanec`, `plat` a `pozice` a chceme vytvořit funkce pro temporální spojení všech tří tabulek, provedeme dotaz

```
SELECT addTemporalJoin(array('zamestnanec', 'plat', 'pozice'));,
```

který nám vytvoří dvě funkce a to funkci pro "sequenced"spojení nazvanou `sequencedJoinZamestnanecPlatPozice()` a pro "nonsequenced"spojení nazvanou `sequencedJoinZamestnanecPlatPozice()`. Tyto dvě varianty funkcí jsou, ale vytvořeny pouze pro spojení

mezi tabulkami bitemporálními nebo tabulkami z nichž alespoň jedna je tabulka s časem platnosti. Spojené tabulky s transakčním časem nemají vytvořeny funkce pro "sequenced"a "nonsequenced"varianty, ale funkci pro "snapshot"spojení, která by pro zmíněné dvě tabulky vypadala jako `snapshotJoinZamestnanecPlatPozice()`.

Abychom se nemuseli obtěžovat s vytvářením funkcí pro spojení dvou tabulek, tak temporální rozšíření je za nás vytvoří v okamžiku vytvoření reference. Název tohoto spojení je pak složen konkatenací názvu referenční tabulky s názvem referencované tabulky. Se zavoláním funkce se nevytvoří pouze funkce, ale hlavně návratové typy z těchto funkcí. Funkce jsou v databázi uloženy, až do doby jejich zrušení funkcí `dropTemporalJoin(TEXT[]);`, kde je zase jako vstup pole názvů tabulek, stejně jako při vytvoření.

Návratové typy mají jednotlivé položky (sloupce) pojmenovány původním pojmenováním sloupců ve spojovaných tabulkách, ale aby nemohlo dojít k zdvojenému pojmenování sloupce, například když bychom spojovaly tabulky `zamestnanec` a `zakaznik` je možné, že se v obou tabulkách bude vyskytovat sloupec `jmeno`, je před název sloupce přikonkatenován název tabulky a znak "_". Pokud bychom tedy měli tabulky `zamestnanec` a `zakaznik` pouze se sloupcem `jmeno`, měl by výsledný datový typ pojmenován sloupce řetězcemi `zamestnanec_jmeno`, `zakaznik_jmeno`. Tímto mechanismem je vyloučeno duplicitní pojmenování sloupců a navíc je ve výpisu přehledně rozlišeno o jaké sloupce jaké tabulky ze spojovaných se jedná.

7.5.2 Získávání dat ze spojených tabulek

Z předchozí kapitoly již víme, že můžeme vytvořit "sequenced"a "nonsequenced", pořípadě "snapshot"variantu pro spojení více tabulek. Nyní si ukážeme příklady použití těchto funkcí s popisem navrácených výsledků a také si popíšeme parametry těchto funkcí. Pro jednoduchost budeme nadále uvažovat naše dvě tabulky `zamestnanec` a `pozice`.

Začneme s příkladem, kde obě tabulky budou tabulkami s časem platnosti. Funkce pro spojení budou s předpisem `sequencedJoinZamestnanecPozice(BOOLEAN)` a `nonsequencedJoinZamestnanecPozice(TEXT)`. První zmíněná funkce má vstupní parametr booleovskou hodnotu podle toho, zda požadujeme slévání časů platnosti či nikoliv. Chování je pak stejné jako u výběru dat z tabulek s časem platnosti. Druhá zmíněná funkce má jako parametr textovou hodnotu, kde je očekávána část kaluzule `WHERE`, ve které můžeme použít sloupec `valid_time` pro porovnávání stejně tak, jak tomu bylo u výběru z tabulek s časem platnosti. Pro konkrétní příklad si představme, že máme stav tabulek stejný jako je naznačen v tabulce 7.13. Potom po vykonání dotazu

```
SELECT * FROM sequencedJoinZamestnanecPozice(true);
```

dostaneme tabulku 7.14. V této tabulce jsou časy platnosti restruktualizovány. V tabulce je pouze jeden čas platnosti, který je vytvořen průnikem časů platnosti spojených záznamů v tabulce `zamestnanec` a tabulce `pozice`.Obecně se jedná o průnik všech časů platnosti

všech spojených tabulek (kontrola referenční integrity zajišťuje neprázdný průnik). Druhým dotazem

```
SELECT * FROM nonsequencedJoinZamestnanecPozice();
```

dostáváme tabulku 7.15, která nemá sloupec s časy platnosti. Posledním příkladem je dotaz se specifikovaným parametrem

```
SELECT * FROM nonsequencedJoinZamestnanecPozice(
    'contains(valid_time, ''2003-01-01''::timestamp)'
);,
```

který vybere pouze sloupce odpovídající podmínce jak je tomu v tabulce 7.16.

jmeno	valid time	popis	valid time
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	delnik	[1990-01-01 00:00:00+01, 2015-01-01 00:00:00+01)
jan novak	[1995-01-01 00:00:00+01, 2005-01-01 00:00:00+01)	delnik	[1990-01-01 00:00:00+01, 2015-01-01 00:00:00+01)
jan novak	[2010-01-01 00:00:00+01, 2015-01-01 00:00:00+01)	delnik	[1990-01-01 00:00:00+01, 2015-01-01 00:00:00+01)

Tabulka 7.13: spojená tabulka s časy platnosti

jmeno	popis	valid time
jan novak	delnik	[1990-01-01 00:00:00+01, 2005-01-01 00:00:00+01)
jan novak	delnik	[2010-01-01 00:00:00+01, 2015-01-01 00:00:00+01)

Tabulka 7.14: spojená tabulka s časy platnosti "sequenced"varianta

jmeno	popis
jan novak	delnik
jan novak	delnik
jan novak	delnik

Tabulka 7.15: spojená tabulka s časy platnosti "nonsequenced"varianta

jmeno	popis
jan novak	delnik

Tabulka 7.16: spojená tabulka s časy platnosti "nonsequenced"varianta s podmínkou

Speciálním případem je tabulka s časem platnosti, která je v referenci se snímkovou tabulkou. Temporální rozšíření nezakazuje tuto referenci a umí ji řešit i na úrovni spojení tabulek. Pro spojení jsou vytvořeny stejné funkce jako v případě spojení tabulek s časy

platnosti. Každá snímková tabulka je při spojení jakoby rozšířena o vlastní čas platnosti, který je pro každý záznam `period('-infinity'::timestamp, 'infinity'::timestamp)`.

Tabulky s transakčním časem jsou spojovány pomocí "snapshot"varianty, kdy předpis funkce je pro naše dvě tabulky `snapshotZamestnanecPozice(TIMESTAMP)`. Jediný parametr není povinný (automaticky je doplněno `now()::timestamp`) a udává časový okamžik pro který chceme znát stav databáze. Máme-li jednoduché spojení tabulek 7.17 pak po dotazu

```
SELECT * FROM snapshotJoinZamestnanecPozice(
    '2015-04-23 20:14:50'::timestamp
);
```

dostaneme tabulku 7.18. Dostaneme tedy stav databáze jak vypadala v časovém okamžiku '2015-04-23 20:14:50'.

jmeno	transaction	popis	transaction
jan novak	[2015-04-23 20:14:09+02, infinity)	delnik	[2015-04-23 20:14:48+02, infinity)
jan novak	[2015-04-23 20:14:09+02, infinity)	vedouci	[2015-04-23 20:15:26+02, infinity)

Tabulka 7.17: spojená tabulka s transakčními časy

jmeno	popis
jan novak	delnik

Tabulka 7.18: spojená tabulka s transakčními časy "snapshot"varianta

Získávání dat z bitemporálních tabulek je kombinací získávání dat s časem platnosti se získáváním dat do tabulek s transakčním časem. Protože biteporální tabulky mají čas platnosti, pak existuje "sequenced"a "nonsequenced"varianta spojení těchto tabulek. Předpis funkce pro naše dvě tabulky je `sequencedJoinZamestnanecPozice(BOOLEAN, TIMESTAMP)`, kde první parametr udává zda mají být záznamy restruktualizovány a druhý parametr je čas, ke kterému chceme znát obsah databáze. Druhým předpisem funkce je `nonsequenced-JoinZamestnanecPozice(TEXT, TIMESTAMP)`, kde druhý parametr má stejný význam jako u předchozí funkce a první parametr je textová forma klauzule `WHERE`, kde můžeme pracovat se sloupcem `valid_time`. Nyní si použití ukážeme na dvou jednoduchých příkladech a jako zdroj nám bude sloužit tabulka 7.19. První dotaz

```
SELECT * FROM nonsequencedJoinZamestnanecPozice(
    '', '2015-04-23 17:39:22'::timestamp
);
```

nám z bitemporálních tabulek vrátí data zobrazená v tabulce 7.20. V této tabulce jsou pouze dva záznamy, které se logicky nacházely v databázi v čase '2015-04-23 17:39:22'.

jmeno	valid time	transaction	popis	valid time	transaction
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-23 17:25:14+02, 2015-04-23 17:35:15+02)	delnik	[1990-01-01 00:00:00+01, 2015-01-01 00:00:00+01)	[2015-04-23 17:26:02+02, 2015-04-23 17:32:12+02)
jan novak	[1995-01-01 00:00:00+01, 2005-01-01 00:00:00+01)	[2015-04-23 17:25:33+02, 2015-04-23 17:35:15+02)	delnik	[1990-01-01 00:00:00+01, 2015-01-01 00:00:00+01)	[2015-04-23 17:26:02+02, 2015-04-23 17:32:12+02)
jan novak	[2010-01-01 00:00:00+01, 2015-01-01 00:00:00+01)	[2015-04-23 17:25:49+02, 2015-04-23 17:35:15+02)	delnik	[1990-01-01 00:00:00+01, 2015-01-01 00:00:00+01)	[2015-04-23 17:26:02+02, 2015-04-23 17:32:12+02)

Tabulka 7.19: bitemporální tabulka pro demonstraci výběru dat

jmeno	popis
jan novak	delnik
jan novak	delnik

Tabulka 7.20: bitemporální tabulka "nonsequenced"varianta

Druhým dotazem

```
SELECT * FROM sequencedJoinZamestnanecPozice(
    true, '2015-04-23 17:39:22'::timestamp
);
```

získáme jako odpověď tabulku 7.21. V této tabulce je pouze jeden záznam vyhovující časovému okamžiku '2015-04-23 17:39:22', protože časy platnosti jsou restruktualizovány do jednoho intervalu jinak bychom dostali záznamy dva jako v předchozím příkladě.

jmeno	popis	valid time
jan novak	delnik	[1990-01-01 00:00:00+01, 2005-01-01 00:00:00+01)

Tabulka 7.21: bitemporální tabulka "sequenced"varianta

7.6 Vkládání dat

Záznamy v temporálních tabulkách nemají sloupec s časem platnosti nebo transakčním časem, ale tyto hodnoty jsou uloženy v pomocné tabulce, která se na tyto záznamy odkazuje. Z tohoto důvodu není možné vkládat čas platnosti přímo, ale je nutné použít předpřipravenou funkci `insertValidTime(TEXT, PERIOD)`, kde první parametr funkce je dotaz na vložení záznamů do temporální tabulky a druhý parametr je čas platnosti. Dotaz na vložení se zadává jako řetězec, který nesmí končit středníkem a nesmí obsahovat klauzuli `RETURNING`.

Klauzule `RETURNING` v PostgreSQL umožňuje vrátit hodnoty, které jsou založené na vkládaných záznamech [8]. Vnitřní implementace funkce přiřetězí k `INSERT` dotazu právě klauzuli `RETURNING` pro získání hodnot `oid` pro identifikaci vloženého řádku a `tableoid` pro identifikaci tabulky. Na základě hodnot `oid` a `tableoid` víme, do které tabulky vkládat čas platnosti a který záznam k němu přiřadit. Jako příklad budeme vkládat nové osoby do tabulky `zamestnanec` jako

```
SELECT * FROM insertValidTime(
    'INSERT INTO zamestnanec VALUES
        (''John'', ''Doe''),
        (''Maria'', ''Smith'')',
    period(now(), 'infinity'::timestamp)
);
```

Uvnitř funkce dojde k provedení upraveného dotazu

```
INSERT INTO zamestnanec
    VALUES (''John'', ''Doe''), (''Maria'', ''Smith'')
    RETURNING oid, tableoid
```

a následné vložení času platnosti do tabulky `t__zamestnanec__temp_data`.

Konkrétní příklady na vkládání dat například do bitemporální tabulky můžou dotazy vypadat jako

```
INSERT INTO zamestnanec VALUES ('jan novak');
SELECT insertValidTime(
    'INSERT INTO zamestnanec VALUES (''john doe'')',
    period('1990-01-01', '2000-01-01')
);
```

a tomuto dotazu pak odpovídá tabulka 7.22. V tabulce jsou automaticky vloženy i časy vložení o které se stará temporální rozšíření. Pokud bychom použili stejné dotazy jako v

popis	valid time	transaction
jan novak	[-infinity, infinity)	[2015-04-23 00:52:56+02, infinity)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-23 00:53:23+02, infinity)

Tabulka 7.22: vložená data v bitemporální taublce

případě vkládání záznamů do tabulky bitemporální s tím rozdílem, že bychom vkládali záznamy do tabulky s časem platnosti pak bychom dostali tabulku 7.23. Zde si můžeme všimnout stejně jako v tabulce bitemporální, že čas platnosti, pokud není definován, se vloží jako interval neomezený interval.

Mimo vkládání běžných záznamů do tabulek, musíme mít i možnost vkládat záznamy, konkrétně cizí klíče, do vazebních tabulek. Abychom byli odstíněni od toho, že mají dvě tabulky, které jsou v referenci svou vlastní vazební tabulku, je v temporálním rozšíření připravena funkce `insertReferenceIntegrity(TEXT, TEXT, OID[], OID[])`. Funkce má čtyři parametry, kde první dva jsou textové parametry, ve kterých je název referenční (první

popis	valid time
jan novak	[-infinity, infinity)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)

Tabulka 7.23: vložená data v tabulce s časem platnosti

parametr) a název referencované (druhý parametr) tabulky v textové podobě. Další dvojice parametrů jsou pole OID, tedy primárních klíčů, které mají být spárovány. Třetí parametr jsou OID referenční tabulky a poslední parametr jsou OID referencované tabulky. V případě několika klíčů v každém poli jsou záznamy spárovány, tak, že každé OID z jednoho pole je spárováno s každým OID z pole druhého. Funkce je univerzální a je možné ji použít pro jakékoli (povolené) typy tabulek, které mají mezi sebou vazbu. Názorně pak dotaz může vypadat jako

```
SELECT insertReferenceIntegrity(
    'zamestnanec',
    'pozice',
    array(SELECT oid FROM zamestnanec),
    array(SELECT oid FROM pozice)
);
```

kde je referenční tabulkou tabulka `zamestnanec` a referencovanou tabulkou `pozice`. Funkce `array()` vytvoří pole ze všech záznamů, které jsou vybrány dotazem vstupujícím jako parametr.

7.7 Aktualizace dat

Temporální rozšíření musí podporovat aktualizaci záznamů. Tabulky s transakčním časem jsou na aktualizaci z pohledu uživatele databázového rozšíření nejsnažší. Z pohledu aktualizace nemusí uživatel ani vědět, že aktualizuje tabulku s transakčním časem, protože rozšíření vše obstará za něj. Máme-li jednoduchou tabulku 7.24 s transakčním časem (pro názornost je zpřístupněn sloupec s transakčním časem), pak stačí provést běžný dotaz například

```
UPDATE zamestnanec SET jmeno = 'john doe' WHERE jmeno = 'jan novak';
```

a temporální rozšíření se postará o ukončení transakce původního záznamu a vložení nového jako v tabulce 7.25.

popis	transaction
jan novak	[2015-04-23 00:47:14+02, infinity)

Tabulka 7.24: jednoduchá tabulka s transakčním časem

popis	transaction
john doe	[2015-04-23 00:58:47+02, infinity)"
jan novak	[2015-04-23 00:58:12+02, 2015-04-23 00:58:47+02)

Tabulka 7.25: jednoduchá tabulka s transakčním časem po aktualizaci

Tabulky s časem platnosti jsou aktualizovány běžnými dotazy stejně jako tabulky bitemporální s rozdílem aktualizace času platnosti. Pro aktualizaci času platnosti máme připravenou speciální funkci `updateValidTime(TEXT, OID[], TEXT, PERIOD)` s příkladem použití

```
SELECT updateValidTime(
    'zamestnanec',
    array(SELECT oid FROM zamestnanec WHERE jmeno = 'jan novak'),
    'contains(valid_time, ''2000-01-01''::timestamp)',
    period('2001-01-01'::timestamp, now()::timestamp)
);,
```

kde jako první parametr je řetězec se jménem tabulky, druhým parametem je pole OID záznamů, u kterých se bude čas platnosti měnit. Třetím parametrem je řetězec, do kterého je možné napsat vlastní podmínky (které budou vloženy do klauzule `WHERE`) s tím rozšířením, že je možné zde použít sloupec `valid_time`. Dotaz tedy bude aktualizovat všechny časy platnosti patřící záznamu se jménem 'jan novak' a obsahující časový údaj '2000-01-01'. Poslední parametr funkce je nový čas platnosti.

V následujících odstavcích se budeme zabývat aktualizací času platnosti u bitemporálních tabulek. Aktualizace tabulek s časem platnosti probíhá stejným způsobem s tím rozdílem, že záznamy, popřípadě podintervaly, které jsou v bitemporálních tabulkách smazány pouze logicky, tak v tabulkách s časem platnosti jsou smazány fyzicky.

popis	valid time	transaction
delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-18 20:31:36+02, infinity)

Tabulka 7.26: bitemporální tabulka pozice

Při aktualizaci času platnosti u bitemporálních tabulek máme celkem 5 možností toho jak se mohou aktualizovaný a nový interval vzájemě překrývat. Pro následující ukázky budeme vycházet vždy z tabulky 7.26, která ukazuje obsah tabulky `pozice` i s transakčním časem záznamů, který běžně není možné zobrazit. V prvním případě budeme stávající čas platnosti dělníka aktualizovat na interval

```
period('[1991-01-01 00:00:00+01, 1994-01-01 00:00:00+01)').
```

Zobrazíme-li si graficky tyto dva intervaly (příklad 5) vidíme, že nový interval (na druhém řádku) je součástí původního intervalu, proto se nemusí původní interval mazat, ale je pouze upravena jeho velikost a jeho setrvání v databázi nadále pokračuje. V databázi, ale logicky nezůstávají (je ukončen transakční čas) dva podintervaly původního intervalu a to konkrétně interval `[a, c)` a interval `[d, b)`. Temporální rozšíření a konkrétně funkce pro aktualizaci času platnosti tyto dva podintervaly do databáze znovu vloží, ale čas počátku transakce

je převzat od počátečního času transakce původního intervalu. Samozřejmě čas ukončení transakce je nastaven na aktuální čas. V tabulce 7.27 jsou vidět jednotlivé záznamy po aktualizaci tohoto typu.

```
a   [--------------------)   b
c         [----------)         d
```

Příklad 5: nový interval je obsažen v původním intervalu

popis	valid time	transaction
delnik	[1991-01-01 00:00:00+01, 1994-01-01 00:00:00+01)	[2015-04-18 20:31:36+02, infinity)
delnik	[1990-01-01 00:00:00+01, 1991-01-01 00:00:00+01)	[2015-04-18 20:31:36+02, 2015-04-18 20:31:38+02)
delnik	[1994-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-18 20:31:36+02, 2015-04-18 20:31:38+02)

Tabulka 7.27: bitemporální tabulka pozice po aktualizaci času platnosti jako je naznačeno v příkladě 5

Druhým případem aktualizace je aktualizace na delší časový interval, ve kterém je původní interval obsažen. Náš původní záznam budeme aktualizovat na interval

```
period('[1989-01-01 00:00:00+01, 1996-01-01 00:00:00+01)').
```

Z příkladu 6 je zřejmé, že funkce, která aktualizaci provádí, nechává původní interval nezměněn a pouze vloží nové dva intervaly `[c, a)` a `[b, d)`. Výsledné záznamy si můžeme prohlédnout a ověřit v tabulce 7.28.

```
a         [----------)         b
c   [--------------------)   d
```

Příklad 6: původní interval je obsažen v novém intervalu

popis	valid time	transaction
delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-18 20:33:54+02, infinity)
delnik	[1989-01-01 00:00:00+01, 1990-01-01 00:00:00+01)	[2015-04-18 20:33:56+02, infinity)
delnik	[1995-01-01 00:00:00+01, 1996-01-01 00:00:00+01)	[2015-04-18 20:33:56+02, infinity)

Tabulka 7.28: bitemporální tabulka pozice po aktualizaci času platnosti jako je naznačeno v příkladě 6

Zatím jsme se zabývali intervaly, které byly plně obsažené v jiném intervalu a teď přejdeme na intervaly, které mají částečné průniky. Nejdříve budeme aktualizovat původní interval intervalem jehož počáteční část náleží do původního, ale koncová část již ne jak je vidět v příkladě 7. Pro náš příklad si zvolíme interval

```
period('[1993-01-01 00:00:00+01, 1996-01-01 00:00:00+01)').
```

Z příkladu vidíme, že část intervalu, konkrétně `[c, b)`, nadále logicky setrvává v databázi, část intervalu `[a, c)` je z databáze logicky smazána a část intervalu `[b, d)` je do databáze vložena. Výsledek představuje tabulka 7.29.

```
a  [--------------)        b
c           [------------)  d
```

Příklad 7: počátek nového intervalu je obsažen v původním

popis	valid time	transaction
delnik	[1993-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-18 20:39:47+02, infinity)
delnik	[1990-01-01 00:00:00+01, 1993-01-01 00:00:00+01)	[2015-04-18 20:39:47+02, 2015-04-18 20:40:41+02)
delnik	[1995-01-01 00:00:00+01, 1996-01-01 00:00:00+01)	[2015-04-18 20:40:41+02, infinity)

Tabulka 7.29: bitemporální tabulka pozice po aktualizaci času platnosti jako je naznačeno v příkladě 7

Dalším případem, který je podobný předchozímu, je situace, kdy první část nového intervalu nenáleží původnímu a druhá část náleží původnímu intervalu jako je ukázáno v příkladu 8. Konkrétní příklad si ukážeme na aktualizaci původního intervalu na interval

```
period('[1985-01-01 00:00:00+01, 1993-01-01 00:00:00+01)').
```

K provedení aktualizace je potřeba vložit interval `[c, a)`, původní interval zkrátit z `[a, b)` na interval `[a, d)` a na konec logicky smazat interval `[d, b)`. Tabulka 7.30 zobrazuje výsledek po těchto operacích.

```
a           [------------)  b
c  [--------------)         d
```

Příklad 8: počátek původního intervalu je obsažen v novém

popis	valid time	transaction
delnik	[1990-01-01 00:00:00+01, 1993-01-01 00:00:00+01)	[2015-04-18 20:42:09+02, infinity)
delnik	[1993-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-18 20:42:09+02, 2015-04-18 20:42:25+02)
delnik	[1985-01-01 00:00:00+01, 1990-01-01 00:00:00+01)	[2015-04-18 20:42:25+02, infinity)

Tabulka 7.30: bitemporální tabulka pozice po aktualizaci času platnosti jako je naznačeno v příkladě 8

Na závěr si uvedeme nejjednodušší případ, který je graficky znázorněn v příkladě 9 a tím je případ, kdy mají intervaly prázdný průnik. Ukážeme si situaci při aktualizaci původního intervalu intervalem

```
period('[2000-01-01 00:00:00+01, 2005-01-01 00:00:00+01)').
```

Vidíme, že nemusíme žádný interval rozdělovat na žádné dílčí intervaly, ale pouze logicky smažeme původní interval a vložíme nový, tak jak to ilustruje tabulka 7.31.

```
a   [-------)                b
c                  [-------)  d
```

Příklad 9: prázdný průnik intervalů

popis	valid time	transaction
delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-18 20:43:23+02, 2015-04-18 20:43:25+02)
delnik	[2000-01-01 00:00:00+01, 2005-01-01 00:00:00+01)	[2015-04-18 20:43:25+02, infinity)

Tabulka 7.31: bitemporální tabulka pozice po aktualizaci času platnosti jako je naznačeno v příkladě 9

7.7.1 Mazání záznamů

Temporální rozšíření obsahuje funkci s předpisem `deleteValidTime(TEXT, OID[], PERIOD)`, která smaže daný časový interval z času platnosti. Funkci je možné použít pouze pro tabulky s časem platnosti nebo pro tabulky bitemporální. Prvním parametrem funkce je jméno tabulky v textové podobě, druhým parametrem je pole OID, pro které má být interval smazán a posledním parametrem je mazaný interval. Na příkladu si ukážeme jak probíhá mazání v tabulce 7.32 s časem platnosti. Dotaz

```
SELECT deleteValidTime(
    'zamestnanec',
    array(SELECT oid FROM zamestnanec WHERE jmeno = 'jan novak'),
    period('1991-01-01'::timestamp, '1999-01-01'::timestamp)
);
```

smaže z tabulky `zamestnanec` danou periodu a dostaneme pak tabulku 7.33. Z výsledné tabulky je zřejmé, že došlo k rozdělení pvodního intervalu na 3 podintervaly z nichž byl požadovaný smazán. Funkce podporuje všechny typy překrytí mazaného a zdrojového intervalu. Z bitemporálních tabulek není záznam smazán fyzicky, ale pouze logicky jak je očekáváno.

jmeno	valid time
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)

Tabulka 7.32: tabulka pro demonstraci smazání časového intervalu

popis	valid time
jan novak	[1990-01-01 00:00:00+01, 1991-01-01 00:00:00+01)
jan novak	[1999-01-01 00:00:00+01, 2000-01-01 00:00:00+01)

Tabulka 7.33: tabulka po smazání časového intervalu

Mazání záznamů z temporálních tabulek probíhá běžným způsobem. U tabulek s transakčním časem je ukončena transakce daného záznamu stejně jako u tabulek bitemporálních. Z tabulek s časem platnosti je odstraněn i přidružený časový údaj, který je uložen v samostatné tabulce.

V případě že chceme smazat temporální tabulku použijeme funkci `dropTemporalTable(TEXT)`, kde jako parametr uvedeme název mazané temporální tabulky v textové podobě. Například můžeme smazat tabulku `zamestnanec` dotazem

```
SELECT dropTemporalTable('zamestnanec');,
```

kde funkce zařídí smazání všech potřebných funkcí, typů a podobně.

7.8 Zajištění referenční integrity

7.8.1 Povolené reference

V databázi se běžně setkáváme se snímkovými tabulkami, což jsou běžné tabulky, které známe. Temporální rozšíření přidává navíc tabulky s transakčním časem, tabulky s časem platnosti a bitemporální tabulky. Mezi těmito tabulkami je možné vytvářet reference. Jak jsme si uvedli v kapitole 6.3, v implementovaném rozšíření nejsou povoleny jakékoliv kombinace spojení dvou tabulek. Mezi povolené spojení patří samozřejmě spojení snímkových tabulek, které se v temporálním rozšíření nijak nemění. Tabulky s transakčním časem mohou odkazovat pouze na tabulky s transakčním časem, z čehož plyne, že je zakázáno odkazovat z tabulky s transakčním časem do tabulky s časem platnosti nebo do snímkové tabulky a zákázáno je i odkazování do bitemporální tabulky. Stejně tak není možné, aby snímková tabulka, bitemporální tabulka nebo tabulka s časem platnosti odkazovala do tabulky s transakčním časem. Větší volnost máme u tabulek s časem platnosti, které mohou odkazovat na jiné tabulky s časem platnosti nebo na snímkové tabulky, stejně tak na ně může být s těchto tabulek odkazováno. Bitemporální tabulky nemohou odkazovat na tabulky s časem platnosti ani nemůžeme odkazovat z tabulky s časem platnosti do bitemporální tabulky. Z bitemporální tabulky můžeme odkazovat pouze do bitemporální tabulky.

Pokud se pokusíme vytvořit referenci mezi zakázanými typy tabulek jako například

```
SELECT createReferenceTable('transaction_time_table', 'valid_time_table');
```

dostaneme jako odpověď chybu:

```
ERROR: Forbidden operation! Reference and referencing table have unsuported
formats.,
```

která nám říká, že není možné spárovat tabulky s těmito typy.

7.8.2 Vytvoření referencí

Nyní se podíváme na možnosti, jakými je možné spárovat (vytvořit reference) dvě tabulky. Způsob vytváření referencí je složen z několika kroků. Nejprve si vytvoříme tabulky `zamestnanec` a `pozice` dotazy

```
CREATE TABLE zamestnanec ( jmeno TEXT );
CREATE TABLE pozice ( popis TEXT );
```

Příklad 10: vytvoření tabulek

Tyto tabulky budeme dále využívat v následujících kapitolách. Prozatím si tabulky převedeme na tabulky s časem platnosti dvojicí dotazů

```
SELECT createValidTimeTable('zamestnanec');
SELECT createValidTimeTable('pozice');
```

Příklad 11: převod na tabulky s časem platnosti

Jak jsme si již uvedli v předchozí kapitole, vytváření referencí mezi tabulkami s časem platnosti je povoleno, tak můžeme vytvořit referenci z tabulky `zamestnanec` do tabulky `pozice` dotazem

```
SELECT createReferenceTable('zamestnanec', 'pozice');
```

Příklad 12: spojení tabulek

Protože víme, že pro zajištění referenční integrity je vytvořena párovací tabulka, aby bylo možné zachytit vztah m:n mezi těmito tabulkami, tak v podstatě nezáleží na pořadí názvů tabulek ve funkci `createReferenceTable()`. Je však doporučeno dodržovat pořadí názvů tabulek ve funkci, nejdříve zadáváme referenční tabulku a poté referencovanou. Pořadí je doporučeno především z důvodu přehlednosti, protože s vytvořením reference se vytvoří i funkce pro výběr spojených dat (jak bylo popsáno v kapitole 6.3), kde doména spojených dat vypadá pro naše dvě tabulky jako `(zamestnanec_jmeno, pozice_popis, valid_time)`.

Pokud bychom vytvářeli temporální referenci mezi tabulkami, které obsahují reference na jiné, například snímkové tabulky, tak tyto reference budou smazány.

7.8.3 Zajištění referenční integrity

V této kapitole budeme pracovat s předpisem tabulek, které jsme si vytvořili dotazem z příkladu 10. Tyto tabulky budeme měnit v závislosti na daném případu. Změna tabulek bude spočívat v jejich rozšíření ze snímkové tabulky na tabulku s transakčním časem, časem platnosti nebo na bitemporální tabulku. Nejprve si ukážeme jakým způsobem je zajištěna referenční integrita u dvou tabulek s časem platnosi, které jsou vytvořeny dotazy z příkladů 11 a 12 z předchozí kapitoly.

Mějme tabulky s časem platnosti pojmenované `zamestnanec` s daty, které zobrazuje tabulka 7.34 a tabulku `pozice` jejíž obsah zobrazuje tabulka 7.35. Záznamy v těchto tabulkách jsou mezi sebou spárovány jak je tomu vidět po provedení dotazu z příkladu 13 v tabulce 7.36.

```
SELECT * FROM sequencedZamestnanecPozice();
```

Příklad 13: spojení tabulek

jmeno	valid time
jan novak	[1990-01-01 00:00:00+01, 2005-05-06 00:00:00+02)

Tabulka 7.34: valid time tabulka zamestnanec

popis	valid time
delnik	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)
vedouci	[2000-01-01 00:00:00+01, 2005-05-06 00:00:00+02)

Tabulka 7.35: valid time tabulka pozice

jmeno	popis	valid time
jan novak	delnik	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)
jan novak	vedouci	[2000-01-01 00:00:00+01, 2005-05-06 00:00:00+02)

Tabulka 7.36: spojené tabulky zamestnanec a pozice

Abychom ukázali, že nemůžeme spárovat dva záznamy z tabulek s časem platnosti, které mají neprázdný průnik časů platnosti, tak nejdříve vložíme nový záznam do tabulky pozice, kterému nastavíme čas platnosti na takový, který má prázdný průnik s časem platnosti záznamu v tabulce `zamestnanec` a následně se pokusíme tyto dva záznamy spárovat. Dotaz je zobrazen v příkladě 14.

```
SELECT insertValidTime(
    'INSERT INTO pozice VALUES (''reditel'')',
    period('2008-01-01', '2010-01-01')
);

SELECT insertReferenceIntegrity(
    'zamestnanec',
    'pozice',
    array(SELECT oid FROM zamestnanec),
    array(SELECT oid FROM pozice WHERE popis = 'reditel')
);
```

Příklad 14: vložení a spárování záznamů

Po provedení těchto dotazů nám databázový systém skončí s chybou

```
ERROR: Forbiden operation. Inserted references are not overlaps.
```

právě proto, protože je mezi časy platnosti záznamů prázdný průnik. Abychom pokryli všechny možnosti zajištění referenční integrity u tabulek s časem platnosti ukážeme si ještě aktualizaci času platnosti u našeho jediného zaměstnance. Tentokrát, protože je zaměstnanec spárován s dvěmy záznamy, aktualizujeme čas platnosti, tak, aby pro jeden záznam měla aktualizace neprázdný průnik časů platnosti a pro druhý záznam prázdný průnik. I v tomto příkladě, jehož dotaz vidíme v příkladě 15, nám systém odpoví stejnou chybou jako v předchozím případě. Pokud bychom chtěli smazat nějaký záznam, tak dojde i ke smazání

příslušné reference (mazání se provádí běžným SQL dotazem).

```
SELECT updateValidTime(
    'zamestnanec',
    array(SELECT oid FROM zamestnanec),
    '',
    period('1990-01-01', '1995-01-01')
);
```

Příklad 15: vložení a spárování záznamů

Nyní si ukážeme zajištění referenční integrity u tabulek s transakčním časem. Budeme využívat stejných tabulek s rozdílem, že nyní nebudou rozšířeny o čas platnosti, ale o transakční čas. Běžně není dovolené zobrazit transakční čas, můžeme se na něj pouze dotazovat, ale v následujících tabulkách jej pro názornost zobrazíme do sloupce "transaction". Obsah transakční tabulky `zamestnanec` zobrazuje tabulka 7.37 a obsah transakční tabulky `pozice` je v tabulce 7.38.

jmeno	transaction
jan novak	[2015-04-18 14:22:46+02, infinity)

Tabulka 7.37: transakční tabulka zamestnanec

popis	transaction
delnik	[2015-04-18 14:22:46+02, infinity)
vedouci	[2015-04-18 14:22:46+02, infinity)

Tabulka 7.38: transakční tabulka pozice

Vazby mezi těmito tabulkami jsou ukázány v tabulce 7.39 a stejně jako u předchozích dvou tabulek, tak i v této tabulce máme pro názornost zobrazeny transakční časy, které není možné povolenými dotazy zobrazit.

jmeno	transaction	popis	transaction
jan novak	[2015-04-18 14:22:46+02, infinity)	delnik	[2015-04-18 14:22:46+02, infinity)
jan novak	[2015-04-18 14:22:46+02, infinity)	vedouci	[2015-04-18 14:22:46+02, infinity)

Tabulka 7.39: spojené transakční tabulky zamestnanec a pozice

Nyní už můžeme přejít rovnou k příkladům zajištění referenční integrity. S tabulkami s transakčním časem pracujeme, až na vyjímky, v tomto temporálním rozšíření stejně, jak jsme zvyklí pracovat se snímkovými tabulkami. Proto můžeme provést dotaz na aktualizaci jména zaměstnance dotazem z příkladu 16.

```
UPDATE zamestnanec SET jmeno = 'john doe' WHERE jmeno = 'jan novak';
```

Příklad 16: aktualizace jména zaměstnance

Během provedení tohoto dotazu se na pozadí stane několik akcí. Nejdříve se ukončí transakce původního záznamu 'jan novak' a potom se vloží nový záznam 'john doe' i s patřičnými referencemi, které jsou převzaty od původního záznamu. Toto chování si můžeme ověřit na spojené tabulce 7.40, kterou dostaneme po provedení dotazu.

jmeno	transaction	popis	transaction
jan novak	[2015-04-18 14:22:46+02, 2015-04-18 14:25:19+02)	delnik	[2015-04-18 14:22:46+02, infinity)
jan novak	[2015-04-18 14:22:46+02, 2015-04-18 14:25:19+02)	vedouci	[2015-04-18 14:22:46+02, infinity)
john doe	[2015-04-18 14:25:19+02, infinity)	delnik	[2015-04-18 14:22:46+02, infinity)
john doe	[2015-04-18 14:25:19+02, infinity)	vedouci	[2015-04-18 14:22:46+02, infinity)

Tabulka 7.40: transakční tabulky po aktualizaci zaměstnance

Ukážeme si ještě jeden dotaz pro aktualizaci, pro změnu zase tabulky s pozicí zaměstnance. Dotaz z příkladu 17 nám samozřejmě ukončí transakci původního záznamu a vloží záznam nový.

```
UPDATE pozice SET popis = 'reditel' WHERE popis = 'vedouci';
```

Příklad 17: aktualizace jména zaměstnance

Malá změna zde je v aktualizaci referencí, kdy je záznam spárován se zaměstnancem 'john doe', protože ten nemá ukončen transakční čas, ale záznam již není spárován se zaměstnancem 'jan novak'. Toto chování je očekávající, protože zaměstnanec, kterého jsme smazali nebo aktualizovali se v databázi logicky nenachází a tak není možné s tímto záznamem nadále pracovat. Tabulka 7.41 ukazuje spárované záznamy.

jmeno	transaction	popis	transaction
jan novak	[2015-04-18 14:22:46+02, 2015-04-18 14:25:19+02)	delnik	[2015-04-18 14:22:46+02, infinity)
jan novak	[2015-04-18 14:22:46+02, 2015-04-18 14:25:19+02)	vedouci	[2015-04-18 14:22:46+02, 2015-04-18 14:27:40+02)
john doe	[2015-04-18 14:25:19+02, infinity)	delnik	[2015-04-18 14:22:46+02, infinity)
john doe	[2015-04-18 14:25:19+02, infinity)	vedouci	[2015-04-18 14:22:46+02, 2015-04-18 14:27:40+02)
john doe	[2015-04-18 14:25:19+02, infinity)	reditel	[2015-04-18 14:27:40+02, infinity)

Tabulka 7.41: transakční tabulky po aktualizaci pozice

V případě smazání nějakého záznamu běžným dotazem, dojde k ukončení jeho transakce. Spárování se po smazání zánamu nemaže, protože v historii byly tyto záznamy v relaci, tudíž ji musíme z historického hlediska zachovat. Pokud tedy provedeme dotaz z příkladu 18 dostaneme výslednou tabulku 7.42.

```
DELETE FROM pozice WHERE popis = 'reditel';
```

Příklad 18: smazání pozice

jmeno	transaction	popis	transaction
jan novak	[2015-04-18 14:33:23+02, 2015-04-18 14:33:31+02)	delnik	[2015-04-18 14:33:23+02, infinity)
jan novak	[2015-04-18 14:33:23+02, 2015-04-18 14:33:31+02)	vedouci	[2015-04-18 14:33:23+02, 2015-04-18 14:33:35+02)
john doe	[2015-04-18 14:33:31+02, infinity)	delnik	[2015-04-18 14:33:23+02, infinity)
john doe	[2015-04-18 14:33:31+02, infinity)	vedouci	[2015-04-18 14:33:23+02, 2015-04-18 14:33:35+02)
john doe	[2015-04-18 14:33:31+02, infinity)	reditel	[2015-04-18 14:33:35+02, 2015-04-18 14:33:49+02)

Tabulka 7.42: transakční tabulky po smazání pozice

Nyní je na čase abychom přešli k nejsložitějšímu zajišťování referenční integrity a to právě pro bitemporální tabulky. U bitemporálních tabulek musíme zajistit prvky z transakčních tabulek stejně tak jako prvky z tabulek s časem platnosti. Pro ukázky budeme používat původní tabulky rozšířené o čas platnosti a transakční čas. Záznamy v tabulkách budou podobné jako v předchozích příkladech stejně tak jako vazby, proto si ukážeme rovnou spárovanou tabulku 7.43 s bitemporálními daty. Pro názornost jsou ukázány transakční časy i časy platnosti jednotlivých záznamů.

jmeno	valid time	transaction	popis	valid time	transaction
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)	delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)	vedouci	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)

Tabulka 7.43: spojené bitemporální tabulky

Pokud provedeme dotaz z příkladu 16 pak dostaneme tabulku 7.44, ve které můžeme vidět kombinaci chování zachování referenční integrity u tabulek s časem platnosti s tabulkami transakčními, konkrétně se ukončily transakce zaměstnanci 'jan novak' a byl vložen nový zaměstnanec 'john doe' se zachováním referencí.

jmeno	valid time	transaction	popis	valid time	transaction
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:48:21+02)	delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:48:21+02)	vedouci	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:48:21+02, infinity)	delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:48:21+02, infinity)	vedouci	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)

Tabulka 7.44: bitemporalní tabulky po aktualizaci zaměstnance

Dalším příkladem 17 je stejně jako u transakčních tabulek aktualizován záznam a spárován pouze se záznamem, který se v databázi logicky nachází. Výsledná data jsou v tabulce 7.45.

jmeno	valid time	transaction	popis	valid time	transaction
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:48:21+02)	delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:48:21+02)	vedouci	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:49:15+02)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:48:21+02, infinity)	delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:48:21+02, infinity)	vedouci	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:49:15+02)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:48:21+02, infinity)	reditel	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:49:15+02, infinity)

Tabulka 7.45: bitemporální tabulky po aktualizaci pozice

Pokud budeme data mazat, například dotazem z příkladu 18, pak se bitemporální tabulky budou chovat stejně jako v případě transakčních tabulek a ukončí transakci mazaného záznamu, což je k vidění v tabulce 7.46.

jmeno	valid time	transaction	popis	valid time	transaction
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:48:21+02)	delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)
jan novak	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:48:21+02)	vedouci	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:49:15+02)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:48:21+02, infinity)	delnik	[1990-01-01 00:00:00+01, 1995-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, infinity)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:48:21+02, infinity)	vedouci	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:43:14+02, 2015-04-20 00:49:15+02)
john doe	[1990-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:48:21+02, infinity)	reditel	[1995-01-01 00:00:00+01, 2000-01-01 00:00:00+01)	[2015-04-20 00:49:15+02, 2015-04-20 00:50:03+02)

Tabulka 7.46: bitemporální tabulky po smazání

Zatím jsme na příkladech viděli chování stejné jako u transakčních tabulek, tabulky s časem platnosti zde vystupují v kontrole průniku časů platnosti spárovaných záznamů. Pokud bychom tedy aktualizovali čas platnosti, například dotazem z příkladu 14 dostali bychom chybu s prázdným průnikem časů platnosti.

Na výše uvedených příkladech jsme si ukázali konkrétní příklady práce s temporálními tabulkami, na kterých jsme mohli vidět jakými způsoby je zajištěna referenční integrita v tomto temporálním rozšíření pro databázový systém PostgreSQL. Rozšíření zajišťuje referenční integritu pouze mezi některými typy temporálních tabulek popřípadě v kombinaci temporální tabulky se snímkovou tabulkou. Reference mezi zakázanými tabulkami, zde nemusí být nutně omezující, protože máme k dispozici zajištěnou referenční integritu mezi bitemporálními tabulkami na kterou můžeme převést jakoukoliv tabulku. Pokud bychom převedli tabulku s transakčním časem na tabulku bitemporální, nemusíme se nutně starat o čas platnosti, protože budemeli vkládat data běžnými dotazy, tak se čas platnosti vkládá automaticky a my se musíme smířit pouze s větší paměťovou náročností záznamu.

Kapitola 8

Závěr

Temporální rozšíření pro databázový systém PostgreSQL popsané v tomto dokumentu podporuje základní mechanismy, se kterými se setkáváme v jiných temporálních rozšíření. Hlavní rozdíl oproti ostatním rozšířením je ten, že ostatní rozšíření mají podporu zavedenu přímo do dotazovacího jazyka, kde je obvykle používáno TSQL2, kdežto rozšíření pro PostgreSQL využívá plně možností databázového systému a především jazyka PL/pgSQL. Rozšíření využívá databázových triggerů, pravidel (rule) a funkcí.

Ve srovnání s možnostmi jazyka TSQL2 zmíníme několik vlastností. Čas platnosti má být a v našem rozšíření je podporován pro minulý i budoucí čas. Oproti TSQL2 je typ `TIMESTAMP` (implementovaný v databázovém systému PostgreSQL) omezen minimální a maximální hodnotou. Toto omezení nemusí být zásadně omezující, protože rozsah je poměrně dostačující a navíc máme možnost pracovat s konstantami vyadřující záporné a kladné nekonečno. Jazyk stejně jako toto temporální rozšíření také dává možnost volitelné temporální podpory. Co však v tomto rozšíření chybí, tak to je podpora pro agregační funkce v temporálních tabulkách a chybí i možnost "vacuuming"neboli fyzické odstranění logicky smazaných dat.

Nemůžeme považovat výslednou implementaci za konečnou. Implementace je pouze základním temporálním rozšířením a umožňuje pracovat s temporálními tabulkami. Možných dalších rozšíření této implementace zajisté najdeme několik. Největším nedostatkem je absence lepšího slévání. Konkrétně by bylo dobré nějakým způsobem umožnit slévání nad určitou podmnožinou sloupců tabulky (popřípadě tabulek) a ne pouze slévání nad všemi sloupci. Možnost jak přistupovat k řešení tohoto problému může být podobná, jako při řešení temporálního spojování tabulek. Další možností rozšíření by mohla být implementace funkce, která bude mazat záznamy z tabulky s časem platnosti na základě podmínky s využitím sloupce s časem platnosti. Funkce by například mohla umožňovat smazat všechny záznamy, které jsou platné v určitém časovém intervalu. Další menší nevyhodou implementovaného rozšíření je spojování tabulek. Spojování v implementaci totiž na pozadí nepoužívá klíčových slov "JOIN", ale všechny spojované tabulky jsou vypsány v kaluzuli "FROM"a spojení přes primární klíče se realizuje v klauzuli "WHERE". Abychom se blíže přiblížili možnostem jazyku TSQL2 nabízí se možnost rozšířit implementaci o temporální podporu agregačních funkcí a temporální agregační funkci `RISING` a o možnost "vacuuming".

Temporální rozšíření je svými možnostmi srovnatelné s jinými temporálními rozšířeními implementovanými pro jiné databázové systémy. Implementované temporální rozšíření pro PostgreSQL je použitelné za dodržování určitých zásad a doporučení, která jsou v dokumentu popsána. Mezi tyto zásady patří hlavně používání předpřipravených funkcí a nezasahování do žádných databázových prostředků (ať už se jedná o triggery, funkce atp.), které zajišťují temporální podporu. Rozšíření bylo psáno pro PostgreSQL ve verzi 9.3 a je velmi

dobře připraveno pro přenesení do dalších verzí databázového systému PostgreSQL, protože je psáno v jazyku PL/pgSQL.

Literatura

[1] Information Technology - Database Language SQL. http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt, Červen 1992, [cit. 2015-01-03], (Second Informal Review Draft) ISO/IEC 9075:1992.

[2] TimeConsult [online]. http://www.timeconsult.com/, Květen 2005.

[3] Coordinated Universal Time. http://cs.wikipedia.org/wiki/Coordinated_Universal_Time, 2014, [cit. 2015-01-03].

[4] Relational database management system [online]. http://en.wikipedia.org/wiki/Relational_database_management_system, Prosinec 2014, [cit. 2015-01-03].

[5] PostgreSQL: The world's most advanced open source database. http://www.postgresql.org , [cit. 2015-01-03].

[6] Time Travel [online]. http://www.teradatamagazine.com/v10n04/Tech2Tech/Time-Travel/, [cit. 2015-01-03].

[7] INSERT [online]. http://www.postgresql.org/docs/9.4/static/datatype-oid.html, [cit. 2015-04-30].

[8] INSERT [online]. http://www.postgresql.org/docs/9.4/static/sql-insert.html, [cit. 2015-04-30].

[9] Date/Time Types [online]. http://www.postgresql.org/docs/9.4/static/datatype-datetime.html , [cit. 2015-21-04].

[10] Boehlen, M.: ChronoLog [online]. http://www09.sigmod.org/sigmod/databaseSoftware/chronolog.txt, [cit. 2015-01-03].

[11] Conway, N.: Introduction to Hacking PostgreSQL [online]. http://neilconway.org/talks/hacking/ottawa/ottawa_slides.pdf , 2007, [cit. 2014-12-30].

[12] Davis, J.: PostgreSQL Temporal [online]. http://sourceforge.net/p/pgsql-temporal/code/ci/master/tree/, 2011, [cit. 2015-01-05].

[13] Salvisberg, P.: Multi-temporal Features in Oracle 12c [online]. http://www.salvis.com/blog/2014/01/04/multi-temporal-database-features-in-oracle-12c/, Leden 2014, [cit. 2015-01-03].

[14] Snodgrass, R. T.: TSQL2 Language Specification [online]. http://secsem.cs.arizona.edu/~rts/pubs/SIGMODRecordMar94p65.pdf, Březen 1994.

[15] Snodgrass, R. T.: Developing Time-Oriented Database Applications in SQL [online]. http://www.cs.arizona.edu/people/rts/tdbbook.pdf, http://www.cs.arizona.edu/people/rts/cdrom.tgz, Červen, 1999, [cit. 2014-12-30].

[16] Szkandera, J.: TSQL2 interpret nad post-relačními databázemi v Oracle Database. Diplomová práce [online]. https://www.fit.vutbr.cz/study/DP/DP.php?id=12078 , 2011, [cit. 2014-04-30].

[17] Tomek, J.: TSQL2 interpret nad relační databází. Diplomová práce [online]. https://www.fit.vutbr.cz/study/DP/DP.php?id=8603 , 2009, [cit. 2014-04-30].

Příloha A

Obsah CD

Temporální rozšíření PostgreSQL