Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5220-47920

Bc. Miroslav Šimulčík

TEMPORÁLNE DATABÁZY

Diplomová práca

Študijný program: Softvérové inžinierstvo

Študijný odbor: 9.2.5 Softvérové inžinierstvo

Miesto vypracovania: Ústav informatiky a softvérového inžinierstva, FIIT STU v Bratislave

Vedúci práce: Ing. Ján Máté

máj 2012

ANOTÁCIA

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: SOFTVÉROVÉ INŽINIERSTVO

Autor: Bc. Miroslav Šimulčík
Diplomová práca: Temporálne databázy
Vedúci diplomovej práce: Ing. Ján Máté
máj, 2012

Vo veľkom množstve odvetví, ako napríklad zdravotníctvo, poisťovníctvo a bankovníctvo, je jednou z hlavných požiadaviek na softvérové systémy uchovávanie historických dát. Dôvody na vedenie histórie dát sú rôzne. Jedným z nich môže byť napríklad snaha zvýšiť konkurencie schopnosť prostredníctvom rozhodnutí vychádzajúcich z analýzy vývinu dát za určité časové obdobie. Ďalším z dôvodov je zvýšenie bezpečnosti prostredníctvom evidencie všetkých zmien vykonaných nad dátami.

Najvhodnejším miestom pre zabezpečenie vedenia histórie dát je samotné úložisko dát, ktorým je vo väčšine prípadov databáza. Špeciálne databázy, ktoré zabezpečujú podporu histórie dát sa nazývajú temporálne databázy.

Náplňou úvodných častí tejto práce je analýza súčasného stavu, existujúcich štandardov a dostupných riešení v oblasti temporálnych databáz. Hlavná časť obsahuje analýzu možných riešení správy historických dát vo voľne dostupných databázových systémoch a následný návrh, implementáciu a overenie vhodnejšieho riešenia. Riešenie je koncipované tak, aby zmena pôvodnej databázy na temporálnu databázu nevyžadovala zmeny v aplikáciách, ktoré s ňou pracujú.

ANNOTATION

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: SOFTWARE ENGINEERING

Author: Bc. Miroslav Šimulčík
Master's Thesis: Temporal Databases
Supervisor: Ing. Ján Máté
2012, May

In a large number of sectors such as healthcare, insurance and banking, one of the main requirements for software systems is storing historical data. Reasons for keeping historical data are different. One of them may be an effort to increase competitiveness through decisions based on analysis of data changes over a period of time. Another reason is improving security through records of all changes made to the data.

The most appropriate place for managing historical data is the data storage itself, which is in most cases the database. Special databases, which provide support for historical data are termed temporal databases.

Contents of the introductory sections of this paper is an analysis of the current situation, existing standards and solutions available in the field of temporal databases. The main part contains an analysis of possible solutions for managing historical data in an open source database systems and the subsequent design, implementation and verification of the more suitable solution. Solution is designed so that the change of the original database to temporal database do not require changes to applications that work with it.

Obsah

1 Úvod

Vo veľkom množstve odvetví je jednou z hlavných požiadaviek na softvérové systémy uchovávanie historických dát. Dôvody na vedenie histórie dát sú rôzne. Jedným z nich môže byť napríklad snaha zvýšiť konkurencieschopnosť prostredníctvom rozhodnutí vychádzajúcich z analýzy vývinu dát za určité časové obdobie. Ďalším z dôvodov je zvýšenie bezpečnosti prostredníctvom evidencie všetkých zmien vykonaných nad dátami.

Najvhodnejším miestom pre zabezpečenie vedenia histórie dát ja samotné úložisko dát, ktorým je vo väčšine prípadov databáza. Špeciálne databázy, ktoré zabezpečujú podporu histórie dát sa nazývajú temporálne databázy.

1.1 Oblasti uplatnenia temporálnych databáz

Nie je ťažké nájsť oblasti uplatnenia temporálnych databáz, keďže ľubovoľné dáta môžu byť reprezentované ako temporálne. Otázne však je, či je nutné udržiavať všetky stavy dát. Inak povedané či vedenie histórie dát prinesie organizácii viac výhod ako nevýhod (napríklad väčší objem dát, väčšie množstvo operácií spojených udržiavaním histórie) [1]. Príklady vhodného použitia temporálnych databáz:

- Finančné aplikácie – história trhu akcií
- Rezervačné systémy – kedy boli rezervované letenky
- Medicínske systémy – história záznamov pacientov
- Poisťovníctvo – história produktov klienta
- Bankovníctvo – história stavu účtov

1.2 Prínos temporálnych databáz

Netemporálne databázové systémy zachytávajú len jeden stav sveta, väčšinou súčasný. Podporujú modifikujúce operácie spôsobujúce prechod databázy z jedného stavu do druhého, pričom nahrádzajú staré hodnoty novými. V takýchto databázach nie je problém pridať stĺpce určujúce začiatok a koniec platnosti záznamu. Problémom však je vyberať v čase sa meniace dáta a definovať obmedzenia zabezpečujúce integritu takýchto dát bez použitia na to určeného dátového modelu a dotazovacieho jazyka [11].

V dôsledku nedostupnosti existujúcich riešení sú vývojári softvéru nútení zabezpečovať správu historických dát vo vlastnej réžii mimo databázového systému (v aplikačnej logike) alebo použitím prostriedkov, ktoré databázový systém ponúka (napríklad spúšťače). Nájsť však dostačujúce riešenie použitím takýchto prostriedkov je problematické, časovo náročné a predstavuje tak veľkú záťaž pre tvorcov systémov [2].

Temporálne databázy zabezpečujú transparentnú správu historických dát a tak sa vývojári môžu sústrediť na implementáciu biznis logiky. Rozdiel temporálnych dát oproti netemporálnym je v časovom intervale pridanom k dátam, ktorý vyjadruje čas, kedy platili alebo boli uložené v databáze [3].

1.3 Ciele projektu

Cieľom tejto práce je analýza súčasného stavu, existujúcich štandardov a dostupných riešení v oblasti temporálnych databáz. Ďalej analýza možností riešenia správy historických dát vo voľne dostupných databázových systémoch a následný návrh, implementácia a overenie vhodnejšieho riešenia. Riešenie by malo byť koncipované tak, aby zmena pôvodnej databázy na temporálnu databázu nevyžadovala zmeny v aplikáciách, ktoré s ňou pracujú.

1.4 Prehľad dokumentu

Obsah dokumentu je členený nasledovný:

- Kapitola 2 – V tejto kapitole najprv analyzujeme rôzne typy časových údajov a s nimi súvisiacich druhov temporálnych databáz. Ďalej analyzujeme štandardy zaoberajúce sa vedením histórie dát vzhľadom na čas transakcie, existujúce riešenia na správu verzií záznamov a možnosti riešenia vo voľne dostupných databázových systémoch.
- Kapitola 3 – obsahuje špecifikáciu požiadaviek na riešenie správy verzií záznamov.
- Kapitola 4 – obsahuje návrh riešenia, ktoré slúži na správu verzií transparentne pre aplikačnú logiku, čiže je čisto záležitosťou databázy. Inak povedané, pri nasadení do existujúcej aplikácie nevyžadujú žiadne zmeny v aplikačnej logike. Riešenie realizuje verziovanie v zdrojových kódoch databázového systému PostgreSQL.
- Kapitola 5 – obsahuje opis implementácie navrhnutého riešenia.
- Kapitola 6 – obsahuje opis postupu overenia, či vytvorené riešenie spĺňa špecifikované požiadavky.

2 Analýza

Táto kapitola sa zaoberá analýzou súčasného stavu v oblasti temporálnych databáz. Vychádzajúc z existujúcich ANSI/ISO štandardov v tejto oblasti tu definujeme základné pojmy potrebné pri riešení projektu. Následne zhodnotíme existujúce riešenia správy verzií záznamov a možnosti jej realizácie vo voľne dostupných databázových systémoch. V kapitolách 2.1 a 2.2 vychádzame z dizertačnej práce Andreasa Steinera - A Generalisation Approach to Temporal Data Models and their Implementations [11].

2.1 Typy časových údajov

V temporálnych databázach je z dôvodu nutnosti zachytenia rôznych aspektov vzťahu medzi časom a dátami nutná existencia viacerých časových línií. Nasleduje popis rôznych vnímaní času, ktoré sú na týchto líniách zaznačené.

2.1.1 Používateľsky definovaný čas

Ide o čas, ktorý databáza nijako neinterpretuje (napríklad dátum narodenia) a berie ho ako ľubovoľný iný atribút. Tento atribút má význam len pre používateľa. Dátové modely podporujú používateľsky definovaný čas tak, že poskytujú časové dátové typy (DATE, TIMESTAMP a iné) pre hodnoty atribútov.

Rozdiel medzi interpretovanými a neinterpretovanými časovými atribútmi je v tom, že temporálne dátové modely používajú štruktúru netemporálnych dátových modelov na ukladanie temporálnych dát tak, že rozširujú schémy tabuliek o špeciálne časové atribúty. Následne pri vykonávaní dotazov s temporálnou sémantikou databázový systém pristupuje k týmto atribútom tak, že ich interpretuje. K používateľsky definovaný atribútom takto nepristupuje.

2.1.2 Čas platnosti

Vymedzuje časový úsek, počas ktorého boli, sú alebo budú dáta platné vzhľadom na reálny svet. Napríklad pri zmene trvalého bydliska sa pôvodné údaje stávajú neplatné a začínajú platiť nové údaje, pričom v databáze ostáva údaj o tom, kde osoba bývala pred tým.

Pomocou času platnosti môžeme zaznamenávať zmeny aj do budúcnosti, napríklad pri rezervovaní lístkov na divadelné predstavenie.

Tento čas je interpretovaný databázovým systémom s podporou času platnosti počas vyhodnocovania dotazov alebo kontrolovaní obmedzení (integrity, jedinečnosti). Rozsah platnosti dát musí používateľ zadať pri vkladaní dát alebo vykonávaní zmien (pri nezadaní sa spravidla berie časový úsek od teraz do nekonečna). Čas platnosti (začiatok a koniec) dát môže používateľ modifikovať, a tak skracovať alebo predlžovať platnosť dát.

2.1.3 Čas transakcie

Čas transakcie je čas kedy boli dáta vložené, opravené alebo zmazané (platiť mohli už skôr, alebo dokonca platiť ešte ani nezačali). Napríklad ak potrebujeme opraviť preklep v niektorom z údajov o trvalom bydlisku, zaznamená sa čas zmeny. V tomto prípade sú dáta stále platné (čas platnosti sa nemení) ale tým, že je zmena v údaji zaznamenaná môžeme sa kedykoľvek pozrieť na stav pred zmenou.

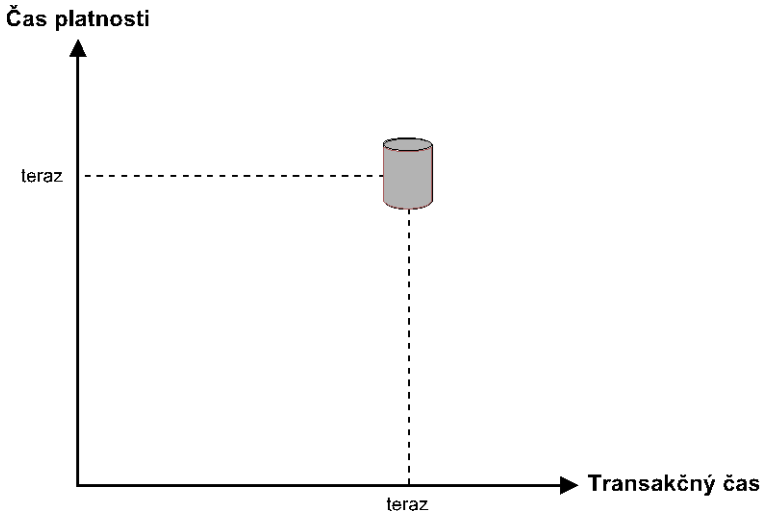
Podobne ako čas platnosti aj čas transakcie je interpretovaný databázovým systémom pri vyhodnocovaní dotazov alebo kontrole obmedzení. Čas transakcie na rozdiel od času platnosti používateľ nezadáva – môže ho len čítať. Čas transakcie zaznamenáva systém pri vykonávaní príkazov INSERT, UPDATE, DELETE a TRUNCATE. Ak by sme používateľom dovolili manipulovať s časom transakcie, mohli by to zneužiť tak, že by zmazali záznam o zmene ktorú vykonali.

2.2 Delenie temporálnych databáz

V predchádzajúcej časti sme predstavili rôzne spôsoby vnímania času. Na základe toho, pozdĺž ktorej osi zaznamenávajú zmeny, môžeme rozlíšiť niekoľko typov temporálnych databáz. V nasledujúcich podkapitolách sa na ne pozrieme bližšie.

2.2.1 Snapshot databázy

Snapshot databázy ukladajú len aktuálne dáta. Po zmene je predchádzajúci stav stratený. Nedá sa rozlíšiť kedy sú dáta platné a kedy boli v databáze zaznamenané. Pri tomto type databáz sa predpokladá, že keď sú dáta v databáze tak sú aj platné. Do tejto skupiny patrí väčšina súčasných databázových systémov. Na obrázku (**Obr. 1**) predstavuje valec stav databázy. Z obrázku je vidno, že v každom čase je k dispozícii len jediný stav – aktuálny.



Obr. 1. Snapshot databáza v kontexte času platnosti a času transakcie

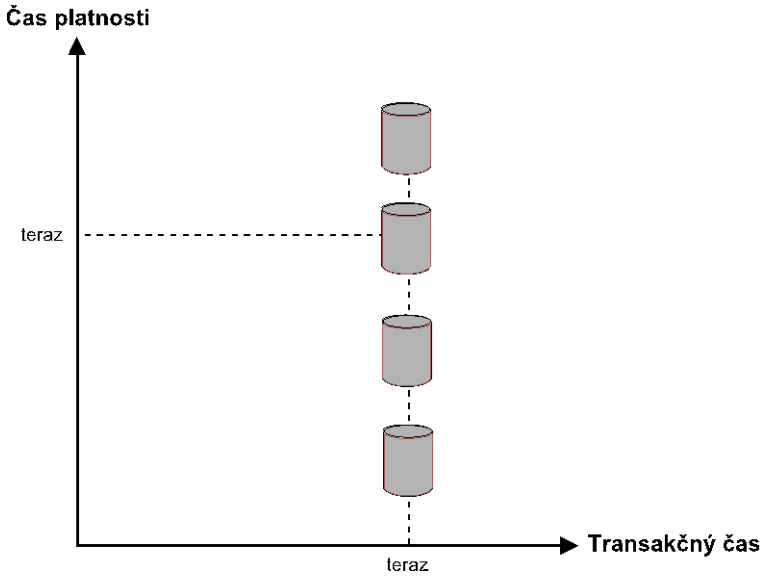
2.2.2 Historické databázy

Ukladajú dáta vzhľadom na čas platnosti. Zaznamenávajú zmeny v reálnom svete pozdĺž osi s časom platnosti. Každá zmena (napr. zmena bydliska) vedie k novému stavu databázy, čiže novému záznamu. Ak vidieť na **Obr. 2**, v jednom časovom okamihu môže byť v databáze uložený súčasný stav a aj minulé a budúce stavy. Každá zmena v reálnom svete vedie k vytvoreniu nového valca (stavu databázy).

Ako už bolo spomenuté, používateľ musí zadať čas platnosti dát pri vkladní alebo zmene. Historické databázy si vyžadujú zložitejší dotazovací jazyk berúci do úvahy čas, ktorý umožňuje hlavne:

- vytváranie tabuliek a obmedzení s podporou času transakcie
- výber dát zo špecifických stavov databázy
- určiť pri modifikujúcich príkazoch, ktoré stavy databázy sa majú upraviť

Treba si uvedomiť, že zmena času platnosti sa v historických databázach zaznamenať nedá. Je to možné len v bitemporálnych databázach, v ktorých sa pri takejto zmene vytvorí nová verzia riadku, pričom stará sa zachová.

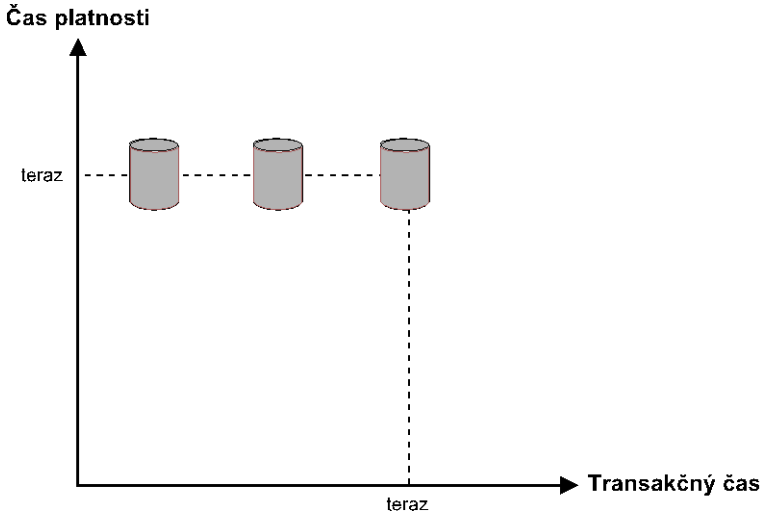


Obr. 2. Historická databáza v kontexte času platnosti a času transakcie

2.2.3 Rollback databázy

Rollback databázy ukladajú dáta vzhľadom na čas transakcie. Zaznamenávajú zmeny databázy samotnej, pozdĺž osi s časom transakcie (**Obr. 3**). Každá zmena v ľubovoľnom stĺpci vedie k vytvoreniu nového záznamu, pričom pôvodné sa ponechávajú, čo umožňuje návrat systému do stavu z minulosti. Žiadne dáta sa fyzicky nezmazávajú.

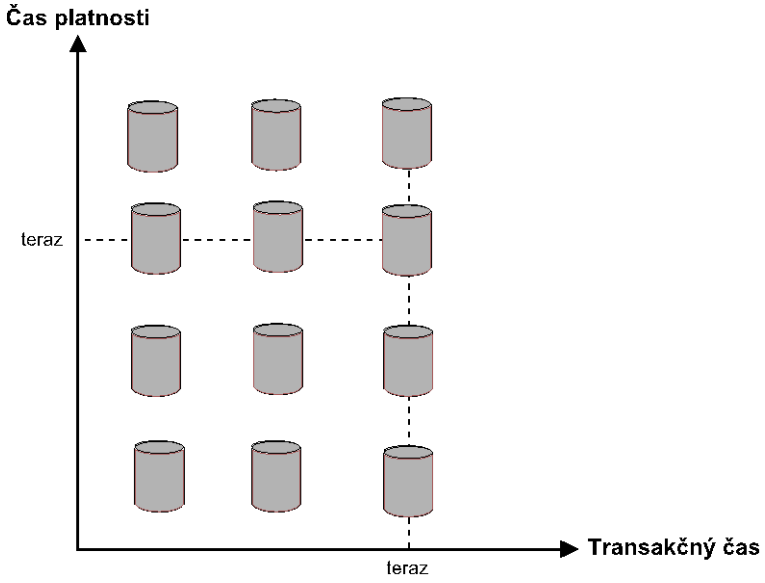
V rollback databázach nie je možné meniť dáta v minulých stavoch databázy. Je možné len vytvárať nové stavy. Tieto databázy nezaznamenávajú budúce stavy databázy, keďže systém sám udržiava verzie dát a nevie nič o budúcich udalostiach.



Obr. 3. Rollback databáza v kontexte času platnosti a času transakcie

2.2.4 Bitemporálne databázy

Bitemporálne databázy sú kombináciou historických a rollback databáz. Zaznamenávajú stavy databázy vzhľadom na čas platnosti aj čas transakcie (**Obr. 4**). Tieto databázy majú teda vlastnosti historických aj rollback databáz, preto je v nich možné zaznamenávať úpravy času platnosti faktov.



Obr. 4. Bitemporálna databáza v kontexte času platnosti a času transakcie

2.3 Temporálne dátové typy

V temporálnych databázach je nutné v nejakej podobe uchovávať údaje o čase platnosti záznamov. Na tento účel je možné použiť viac druhov časových dátových typov. Teraz nerozlišujeme dátové typy na základe časovej granularity (napríklad DATE, TIME, TIMESTAMP...), ale na základe spôsobu akým vyjadrujú čas. Rozlišujeme tieto 3 základné temporálne dátové typy [10]:

- Inštančný čas (instant) – vyjadruje časový okamih. Niečo sa stalo v nejakej časovej inštancii (napríklad „teraz“, čiže 2011-04-21 10:42:15, kedy píšem túto prácu)
- Interval (interval) – vyjadruje časovú dĺžku (napríklad 3 mesiace)
- Obdobie (period) – vyjadruje ohraničené časové trvanie (napríklad od 2011-01-24 do 2012-04-16)

2.4 Temporálny dotazovací jazyk

Z akého dôvodu potrebujeme temporálny dotazovací jazyk, keď existujú aplikácie na správu historických dát a sú založené na netemporálnom jazyku? Neznamená to, že štandardné SQL je postačujúce aj pre temporálne databázy? Skutočnosť je taká, že temporálne dotazy je možné napísať aj v netemporálnom jazyku akým je napríklad SQL, ale tieto dotazy sú veľmi zložité.

Aby sme zabezpečili jednoduchú a efektívnu prácu s temporálnymi dátami potrebujeme temporálny jazyk obsahujúci príkazy na dotazovanie, modifikáciu dát, definovanie obmedzení a iné operácie. Existujú 3 druhy temporálnych príkazov [10], ktoré vysvetlíme na nasledujúcom príklade:

Majme tabuľku výrobcov áut (**Tab. 1**) a tabuľku modelov áut (**Tab. 2**), medzi ktorými kvôli zjednodušeniu neexistuje obmedzenie zabezpečujúce referenčnú integritu. Stĺpce „platny_od“ a „platny_do“ určujú obdobie platnosti záznamov (pre prehľadnosť uvádzame len roky).

Tab. 1. Tabuľka výrobcov áut

vyrobca			
id	**nazov**	**platny_od**	**platny_do**
1	Škoda	1998	2010
2	Audi	1995	∞

Tab. 2. Tabuľka modelov áut

Model				
id	**nazov**	**vyrobca_id**	**platny_od**	**platny_do**
1	Fabia	1	2000	∞
2	Superb	1	2011	∞
3	A8	2	2012	∞

Majme nasledovný takýto príkaz pracujúci nad týmito tabuľkami:

```
SELECT *
   FROM vyrobca
      JOIN model
         ON vyrobca.id = model.vyrobca_id
```

V nasledujúcich častiach sa pozrieme na druhy temporálnych príkazov a ukážeme aký je výsledok príkazu SELECT pre každý z nich.

2.4.1 Nesekvenčné temporálne príkazy

Tento druh príkazov úplne ignoruje čas platnosti záznamov v tabuľkách [10]. Atribúty „platny_od" a „platny_do" databázový systém berie ako všetky ostatné biznis atribúty a nijak špeciálne ich neinterpretuje. Výsledok príkazu SELECT z príkladu vyhodnoteného nesekvenčne je znázornený v tabuľke (**Tab. 3**). Zobrazené sú všetky riadky karteziánskeho súčinu riadkov z tabuľky výrobcov a tabuľky modelov, pre ktoré platí že „vyrobca.id = model.vyrobca_id". Ide teda o bežne používaný INNER JOIN.

Tab. 3. Výsledok nesekvenčne vyhodnoteného príkazu SELECT z príkladu

Vyrobca				model				
id	**nazov**	**platny_od**	**platny_do**	**id**	**nazov**	**vyrobca_id**	**platny_od**	**platny_do**
1	Škoda	1998	2010	1	Fabia	1	2000	∞
1	Škoda	1998	2010	2	Superb	1	2011	∞
2	Audi	1995	∞	3	A8	2	2012	∞

2.4.2 Sekvenčné temporálne príkazy

Sekvenčné temporálne príkazy sú vyhodnocované vzhľadom na každý časový moment. Čas platnosti v tomto prípade systém interpretuje pri vykonávaní príkazov [10]. Najlepšie to vysvetľuje výsledok sekvenčne vyhodnoteného príkazu SELECT z príkladu (**Tab. 4**).

Pri redukovaní riadkov karteziánskeho súčinu tabuliek, sa navyše berie do úvahy aj čas platnosti. Dôsledkom toho je, že druhý riadok nesekvenčného výsledku (**Tab. 3**) sa v tomto prípade vôbec nevyskytuje. Je to tak preto, lebo podmienka „vyrobca.id = model.vyrobca_id" v žiadnom časovom okamihu pre tento riadok neplatí (záznam pre model Superb platí od roku 2011, ale platnosť výrobcu Škoda skončila už v roku 2010).

Časové stĺpce v tabuľke (**Tab. 4**) určujú čas platnosti riadkov výsledku. Prvý riadok je platný len od roku 2000 do roku 2010, aj napriek tomu že platnosť záznamu pre model Fabia je až nekonečna. Dôvodom je to, že po roku 2010 neplatí podmienka „vyrobca.id = model.vyrobca_id" lebo výrobca Škoda už neexistuje. Pre interval od 1998 do 2000 nie je podmienka splnená lebo vtedy ešte neexistoval model Fabia (výrobca Škoda existoval).

Tab. 4. Výsledok sekvenčne vyhodnoteného príkazu SELECT

vyrobca		model				
id	**nazov**	**id**	**nazov**	**vyrobca_id**	**platny_od**	**platny_do**
1	Škoda	1	Fabia	1	2000	2010
2	Audi	3	A8	2	2012	∞

2.4.3 Aktuálne temporálne príkazy

Aktuálne príkazy pracujú len nad aktuálne platnými záznamami. Pri vykonávaní tieto príkazy interpretujú čas platnosti záznamov. Tieto príkazy vracajú rovnaké výsledky, ako keby išlo o netemporálnu databázu, zachytávajúcu len aktuálny stav. To znamená, že neplatné dáta vôbec neuvažujú a nevracia sa ani čas platnosti (pokiaľ sa explicitne nevymenuje) [10]. Výsledok aktuálne vyhodnoteného príkazu SELECT z príkladu je zachytený v nasledujúcej tabuľke (**Tab. 5**), pričom predpokladáme, že „teraz“ je rok 2013.

Tab. 5. Výsledok aktuálne vyhodnoteného príkazu SELECT z príkladu

vyrobca		model		
id	**nazov**	**id**	**nazov**	**vyrobca_id**
2	Audi	3	A8	2

2.5 TSQL2

TSQL2 je temporálne rozšírenie štandardu jazyka SQL-92. Komisia TSQL2 bola založená v roku 1993 a v septembri roku 1994 publikovala finálnu špecifikáciu jazyka TSQL2. Táto špecifikácia spolu s komentármi a iným podporným materiálom bola v roku 1995 publikovaná ako kniha The TSQL2 Temporal Query Language [12].

TSQL2 prináša nový dátový model a nové dátové typy určené pre správu temporálnych dát [8]. Definuje novú sémantiku a syntax príkazov na prácu s temporálnymi dátami. Tento štandard, nebudeme rozoberať do hĺbky, ale vychádzajú z neho takmer všetky novšie štandardy, a tak je dôležité ho aspoň spomenúť.

2.6 SQL/Temporal

SQL/Temporal je názov jednej z častí štandardu SQL3, ktorá bola akceptovaná v roku 1995. V tejto časti bola zahrnutá modifikácia dátového typu PERIOD zo štandardu TSQL2 [4].

Potom začali diskusie o pridaní podpory času platnosti a transakčného času do SQL/Temporal. Dva návrhy boli akceptované ANSI komisiou a posunuté ISO komisii v roku 1997. Kvôli nesúhlasu ISO komisie bol však projekt temporálnej podpory zrušený na konci roku 2001 [13].

Väčšina existujúcich funkčných riešení vychádza zo štandardu SQL/Temporal, a preto sa v tejto časti pozrieme na najdôležitejšie rozšírenia sémantiky a syntaxe štandardného jazyka SQL, ktoré prináša.

2.6.1 Dátový typ PERIOD

Dátový typ `PERIOD` predstavuje, ohraničené trvanie času. V tejto časti sa pozrieme na jeho hlavné súčasti.

2.6.1.1 Konštruktor

SQL/Temporal obsahuje konštruktor `PERIOD()`. Ten umožňuje vytvoriť typ `PERIOD` z inštančných časových typov a číselných typov s pevnou desatinnou čiarkou nasledovne [10]:

- `PERIOD(DATE)`
- `PERIOD(TIME)`
- `PERIOD(TIME WITH TIME ZONE)`
- `PERIOD(TIMESTAMP)`
- `PERIOD(TIMESTAMP WITH TIME ZONE)`
- `PERIOD(INT)`
- `PERIOD(INTEGER)`
- `PERIOD(SMALLINT)`
- `PERIOD(NUMERIC)`
- `PERIOD(DECIMAL)`

Takýmto spôsobom možno ľahko určovať granularitu obdobia.

2.6.1.2 Konštanty

S konštantami typu `PERIOD` sú spojené viaceré komplikácie. Prvou z nich je, že existujú 4 varianty uzavretosti a otvorenosti hraníc obdobia [10]:

- uzavretý – uzavretý

- uzavretý – otvorený
- otvorený – uzavretý
- otvorený – otvorený

Uzavreté hranice sa označujú hranatými zátvorkami (‘[‘, ‘]‘) otvorené hranice bežnými zátvorkami (‘(‘, ‘)‘). Druhou komplikáciou je, že oddeľovač hraníc môže byť buď čiarka (‘,‘), alebo pomlčka (‘-‘). V prípade pomlčky musia byť okolo nej medzery. Treťou komplikáciou je, že časová zóna môže byť uvedená len pri koncovej hranici. Príklady zápisu konštánt typu `PERIOD(DATE)`, vyjadrujúcich to isté obdobie platnosti [10]:

- `PERIOD [1997-01-01 - 1997-12-31]`
- `PERIOD [1997-01-01 - 1998-01-01)`
- `PERIOD (1996-12-31 - 1997-12-31]`
- `PERIOD (1996-12-31 - 1998-01-01)`
- `PERIOD [1997-01-01,1997-12-31]`

Príklady zápisu konštánt typu `PERIOD(TIMESTAMP WITH TIMEZONE)`, vyjadrujúcich to isté obdobie platnosti [10]:

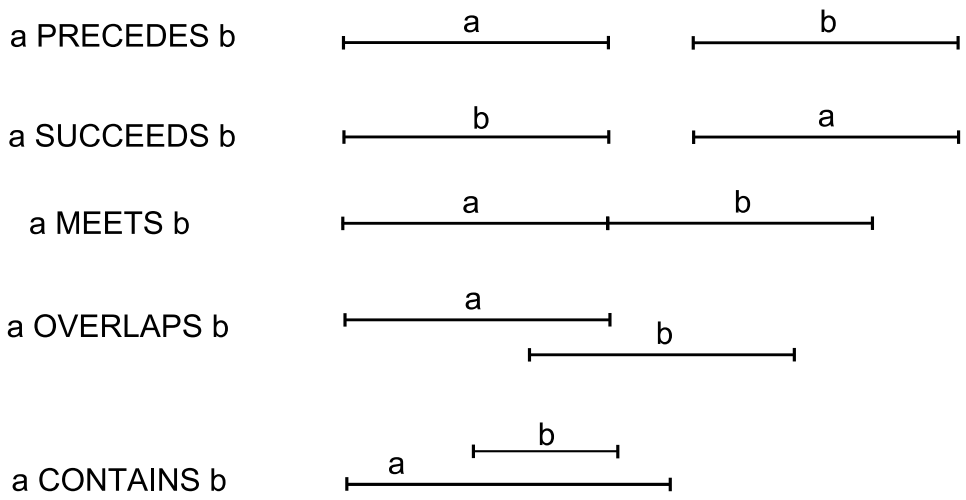
- `PERIOD [1997-01-01 00:00:00 - 1997-12-31 23:59:59-07:00]`
- `PERIOD [1997-01-01 00:00:00 - 1998-01-01 00:00:00-07:00)`
- `PERIOD (1996-12-31 23:59:59 - 1997-12-31 23:59:59-07:00]`
- `PERIOD (1996-12-31 23:59:59 - 1998-01-01 00:00:00-07:00)`

2.6.1.3 Predikáty

Pre dátový typ `PERIOD` existujú okrem bežných (‘=‘, ‘<>‘, ‘<‘, ‘>‘, `IS NULL`) aj nasledovné predikáty (znázornené na **Obr. 5**, matematicky vyjadrené v **Tab. 6**) [10]:

- `a PRECEDES b` – je vyhodnotený ako pravdivý, keď obdobie „a“ skončilo predtým ako začalo obdobie „b“, ale nie hneď.
- `a SUCCEEDS b` – je vyhodnotený ako pravdivý, keď obdobie „a“ začalo platiť až po skončení „b“, ale nie hneď.
- `a MEETS b` – je vyhodnotený ako pravdivý, keď obdobia nasledujú bezprostredne za sebou.

- `a OVERLAPS b` – je vyhodnotený ako pravdivý, keď sa obdobia „a“ a „b“ prekrývajú
- `a CONTAINS b` – je vyhodnotený ako pravdivý, keď „a“ platí výlučne počas platnosti „b“ a súčasne „a“ nerovná sa „b“.



Obr. 5. Grafické znázornenie predikátov typu PERIOD

Pri matematickom vyjadrení predikátov dátového typu PERIOD v nasledujúcej tabuľke (**Tab. 6**) používame pričítanie a odčítanie 1 časovej jednotky. Typ časovej jednotky závisí od inštančného typu, z ktorého je typ PERIOD vytvorený. Napríklad pri `PERIOD(TIMESTAMP)` je časová jednotka 1 milisekunda a pri `PERIOD(DATE)` 1 deň.

Tab. 6. Matimatické vyjadrenie predikátov typu PERIOD

Predikát	Podmienka
`a PRECEDES b`	`END(a) < BEGIN(b) - 1`
`a SUCCEEDS b`	`END(b) + 1 < BEGIN(a)`
`a MEETS b`	`END(a) = BEGIN(b) - 1` `  OR END(b) + 1 = BEGIN(a)`

`a OVERLAPS b`	`(BEGIN(a) < BEGIN(b) AND BEGIN(b) < END(a))`
	`  OR (BEGIN(b) < BEGIN(a) AND BEGIN(a) < END(b))`
	`  OR (BEGIN(b) < BEGIN(a) AND END(a) < END(b))`
	`  OR (BEGIN(a) < BEGIN(b) AND END(b) < END(a))`
	`  OR (BEGIN(b) <= BEGIN(a) AND END(b) <= END(a))`
	`  OR (BEGIN(a) <= BEGIN(b) AND END(a) <= END(b))`

`a CONTAINS b`	`BEGIN (a) < BEGIN(b) AND END(b) < END (a)`

2.6.1.4 *Operátory*

Pre dátový typ PERIOD existujú rôzne operátory, ktorých prehľad uvádzame v nasledujúcej tabuľke (**Tab. 7**).

Tab. 7. Prehľad operácií dátového typu PERIOD

Operácia	Výsledok
`BEGIN(p)`	Začiatok obdobia p
`END(p)`	Koniec obdobia p
`LAST(p)`	`END(p) - 1`
`PRIOR(t)`	t - 1 (Aplikuje sa na inštančné časové typy)
`NEXT(t)`	t + 1 (Aplikuje sa na inštančné časové typy)
`INTERVAL(p)`	interval platnosti obdobia p (napríklad 3 mesiace)
`PERIOD[a,b)`	Vytvorí obdobie so začiatkom "a" a koncom "b" (môžu sa používať všetky kombinácie uzavretosti hraníc)
`p P_UNION q`	Zjednotenie období p a q (ak neplatí, že `p OVELAPS q`, tak chyba)
`p P_EXCEPT q`	Rozdiel období p a q (ak platí, že `p CONTAINS q`, tak chyba)
`p P_INTERSECT q`	Prienik období p a q (ak neplatí, že `p OVELAPS q`, tak chyba)
`CAST(p AS type)`	Obdobie so zmenenou granularitou

2.6.2 Rozšírenia príkazov na vytváranie a modifikovanie tabuliek

Pri vytváraní tabuliek je možné v SQL/Temporal zapnúť temporálnu podporu, buď pri vytváraní tabuliek príkazom CREATE TABLE alebo aj dodatočne príkazom ALTER TABLE. CREATE TABLE je rozšírený tak, že na jeho konci je možné zadať AS TRANSACTIONTIME a/alebo AS VALIDTIME, na zapnutie temporálnej podpory. Pri čase platnosti je navyše možné určiť aj konštruktor dátového typu period, ktorý sa má použiť [6].

```
CREATE TABLE osoba (
   id INTEGER,
   meno VARCHAR,
   priezvisko VARCHAR
) AS TRANSACTION TIME AS VALIDTIME PERIOD(DATE);
```

Rozšírenie príkazu ALTER TABLE umožňuje pridať temporálnu podporu už existujúcim netemporálnym tabuľkám prostredníctvom ADD TRANSACTIONTIME alebo ADD VALIDTIME.

```
ALTER TABLE osoba ADD TRANSACTIONTIME;
ALTER TABLE osoba ADD VALIDTIME PERIOD(DATE);
```

2.6.3 Rozšírenia príkazu SELECT

SQL/Temporal rozširuje syntax aj sémantiku príkazu SELECT. Spôsob zápisu aktuálnych, sekvenčných a nesekvenčných dotazov je nasledovný:

2.6.3.1 Aktuálne dotazy

Pri vykonávaní aktuálnych dotazov, zostáva syntax príkazu rovnaká ako v štandardnom SQL [6]:

```
SELECT *
   FROM vyrobca
      JOIN model
         ON vyrobca.id = model.vyrobca_id
```

Výsledok tohto príkladu je znázornený v **Tab. 5** v kapitole 2.4.3. Vďaka tejto vlastnosti, je možné zapojiť temporálnu podporu aj do existujúcich netemporálnych systémov, lebo nie je potrebná žiadna zmena rozhrania pre prácu s databázou, takže aplikačná logika sa nemusí vôbec meniť. Systém automaticky poskytne len aktuálne verzie záznamov.

2.6.3.2 Sekvenčné dotazy

Na vykonanie sekvenčného dotazu je potrebné na začiatok pridať kľúčové slovo TRANSACTIONTIME pre podporu transakčného času alebo VALIDTIME pre podporu času platnosti alebo obe pre kombinovanú podporu [6]. Príkaz SELECT z príkladu sa zmení nasledovne:

```
VALIDTIME SELECT *
   FROM vyrobca
      JOIN model
         ON vyrobca.id = model.vyrobca_id;
```

Tento príkaz vráti výsledok s temporálnou podporou. To neznamená, že do výsledku zahrnie stĺpec s obdobím platnosti, ale to, že tento výsledok by sa dal použiť v nejakom inom sekvenčnom príkaze SELECT. Čo sa týka období platnosti, štandard hovorí, že sú pri sekvenčných a aktuálnych dotazoch nedostupné. Dá sa k nim pristupovať len v nesekvenčných dotazoch, tak ako k bežným používateľom definovaným stĺpcom.

Ak by sme chceli vybrať len verzie z istého časového obdobia, je možné za kľúčové slová určujúce typ sekvenčného dotazu uviesť aj obdobie, z ktorého dáta požadujeme. Napríklad:

```
VALIDTIME '[1997 - 2003)' SELECT * FROM vyrobca JOIN model ON vyrobca.id =
     model.vyrobca_id
```

Kľúčové slová VALIDTIME a TRANSACTIONTIME platia pre všetky vnorené príkazy SELECT ak pre tieto nie je uvedené inak. Je nutné, aby všetky tabuľky alebo korelácie, ktoré sa vyskytujú v sekvenčnom dotaze, mali temporálnu podporu.

2.6.3.3 Nesekvenčné dotazy

Pre vykonanie nesekvenčného príkazu SELECT z príkladu je syntax nasledovná:

```
NONSEQUENCED VALIDTIME SELECT * FROM vyrobca JOIN model ON vyrobca.id =
     model.vyrobca_id
```

Rozdiel v syntaxi oproti sekvenčným dotazom je len v kľúčovom slove NONSEQUENCED, ktorého použitie je rovnaké pri podpore transakčného času aj času platnosti. Tento príkaz vráti výsledok bez temporálnej podpory. Je možné napísať aj nesekvenčný SELECT, ktorého výsledok bude mať temporálnu podporu, tak že uvedieme obdobie platnosti za slovom TRANSACTIONTIME, ktoré bude určovať platnosť riadkov výsledku [6].

Pri nesekvenčných dotazoch je možné pristupovať k obdobiam platnosti prostredníctvom operátorov VALIDTIME(t) a TRANSACTIONTIME(t), kde t je názov tabuľky alebo korelácie. Na nasledujúcom príklade si ukážeme ako vyzerá príkaz, ktorý vráti výsledok s temporálnou podporou a medzi stĺpcami výsledku bude aj obdobie platnosti záznamov.

```
NONSEQUENCED VALIDTIME valid SELECT vyrobca.*, VALIDTIME(vyrobca) AS valid FROM
     vyrobca
```

2.6.4 Rozšírenia modifikujúcich príkazov

SQL/Temporal rozširuje syntax aj sémantiku modifikujúcich príkazov INSERT, UPDATE a DELETE. V tejto časti si ukážeme ako sa líši zápis aktuálnych, sekvenčných a nesekvenčných modifikujúcich príkazov oproti štandardnému SQL. Výsledky operácií si ukážeme na bitemporálnej tabuľke osôb (**Tab. 8**), ktorá na začiatku obsahuje nasledovné záznamy:

Tab. 8. Tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

Záznamy v **Tab. 8** vznikli nasledovnou sekvenciou operácií:

1. V roku 1989 sa do tabuľky vložil používateľ Paľko Mrkvička s ID 1 s obdobím platnosti [1989-∞). Touto operáciou vznikol prvý riadok.
2. V roku 2000 sa používateľovi s ID 1 upravilo meno na Jožko. Tejto zmene sa priradilo obdobie platnosti [1999-∞). Touto operáciou vznikli posledné 2 riadky. Prvý z nich zaznamenáva ukončenie obdobia platnosti pôvodného záznamu a druhý vznik nového obdobia platnosti.

2.6.4.1 Aktuálne modifikujúce príkazy

Aktuálne modifikácie sa vykonávajú SQL príkazmi INSERT, UPDATE a DELETE, tak ako ich poznáme zo štandardu SQL99. Tak je zabezpečená spätná kompatibilita temporálnych tabuliek v exitujúcich aplikáciách. Aktuálne operácie sú dovolené na tabuľkách s podporou transakčného času aj času platnosti [5].

- INSERT - Systém automaticky pridelí vloženému riadku obdobie platnosti začínajúce v čase operácie a platné do nekonečna. Nasleduje príklad aktuálneho príkazu INSERT do pôvodnej tabuľky osôb (**Tab. 9**), ktorej obsah po príkaze je zaznamenaný v tabuľke **Tab. 10**.

Tab. 9. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
INSERT INTO osoba VALUES (2, 'Fero', ,'Krátky');
```

Tab. 10. Tabuľka osôb po vykonaní aktuálnej operácie INSERT

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)
2	Fero	Krátky	[2011-∞)	[2011-∞)

– UPDATE – Systém vykoná zmenu dát len v období začínajúcom v čase operácie a platné do nekonečna. Pri tejto operácii môže vzniknúť niekoľko nových riadkov v závislosti od toho aký čas tabuľka podporuje. Na nasledujúcom príklade aktuálneho príkazu UPDATE nad pôvodnou tabuľkou osôb (**Tab. 11**) je vidieť, že vznikli 2 nové riadky (**Tab. 12**). Prvý zaznamenáva ukončenie platnosti pôvodného záznamu a druhý vznik nového obdobia platnosti.

Tab. 11. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
UPDATE osoba SET meno = 'Viktor' WHERE id = 1;
```

Tab. 12. Tabuľka osôb po vykonaní aktuálnej operácie UPDATE

osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-2011)
1	Jožko	Mrkvička	[1999-2011)	[2011-∞)
1	Viktor	Mrkvička	[2011-∞)	[2011-∞)

– DELETE – Systém vymaže dáta len v období začínajúcom v čase operácie a platné do nekonečna. Pri tejto operácii môže vzniknúť niekoľko nových riadkov v závislosti od

toho aký čas tabuľka podporuje. Na nasledujúcom príklade aktuálneho príkazu DELETE z pôvodnej tabuľky osôb (**Tab. 13**) je vidieť, že vznikol 1 nový riadok (**Tab. 14**), ktorý zaznamenáva ukončenie platnosti pôvodného záznamu.

Tab. 13. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
DELETE FROM osoba WHERE id = 1;
```

Tab. 14. Tabuľka osôb po vykonaní aktuálnej operácie DELETE

osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-2011)
1	Jožko	Mrkvička	[1999-2011)	[2011-∞)

2.6.4.2 Sekvenčné modifikujúce príkazy

Pri sekvenčných príkazoch je možné zadať obdobie platnosti zmeny a tak vykonávať modifikácie v minulosti, v prítomnosti a v budúcnosti. Takéto modifikácie nie je možné vykonávať nad transakčným časom, aby nebolo možné robiť dodatočné zmeny v transakčnom čase historických verzií, ktorých správa je čisto záležitosťou databázového systému [5]. Syntax sekvenčných príkazov rozširuje štandardné SQL o kľúčové slovo VALIDTIME a obdobie platnosti zmien.

- INSERT – Umožňuje vkladať nové záznamy s ľubovoľným obdobím platnosti. Nasleduje príklad sekvenčného príkazu INSERT do pôvodnej tabuľky osôb (**Tab. 15**), ktorej obsah po príkaze je zaznamenaný v tabuľke **Tab. 16**.

Tab. 15. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
VALIDTIME '[2008 - 2020)' INSERT INTO osoba VALUES (2, 'Fero', ,'Krátky');
```

Tab. 16. Tabuľka osôb po vykonaní sekvenčnej operácie INSERT

osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)
2	Fero	Krátky	[2008-2020)	[2011-∞)

- UPDATE – Umožňuje upravovať dáta v zadanom období. Pri tejto operácii môže vzniknúť niekoľko nových riadkov v závislosti od toho aký čas tabuľka podporuje. Na nasledujúcom príklade sekvenčného príkazu UPDATE nad pôvodnou tabuľkou osôb (**Tab. 17**) je vidieť, že vznikli 4 nové riadky (**Tab. 18**), lebo obdobie platnosti príkazu UPDATE prekrývalo obdobia platností dvoch pôvodných záznamov. Museli sa zaznamenať 2 skrátenia období pôvodných záznamov a vytvoriť 2 nové obdobia platnosti.

Tab. 17. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
VALIDTIME '[1995 - 2005)' UPDATE osoba SET meno = 'Viktor' WHERE id = 1;
```

Tab. 18. Tabuľka osôb po vykonaní sekvenčnej operácie UPDATE

osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-2011)
1	Jožko	Mrkvička	[1999-∞)	[2000-2011)
1	Paľko	Mrkvička	[1989-1995)	[2011-∞)
1	Viktor	Mrkvička	[1995-1999)	[2011-∞)
1	Jožko	Mrkvička	[2005-∞)	[2011-∞)
1	Viktor	Mrkvička	[1999-2005)	[2011-∞)

– DELETE – Umožňuje mazať dáta v zadanom období. Pri tejto operácii môže vzniknúť niekoľko nových riadkov v závislosti od toho aký čas tabuľka podporuje. Na nasledujúcom príklade sekvenčného príkazu DELETE nad pôvodnou tabuľkou osôb (**Tab. 19**) je vidieť, že vznikli 2 nové riadky (**Tab. 20**), lebo obdobie platnosti príkazu UPDATE prekrývalo obdobia platností dvoch pôvodných záznamov. Museli sa zaznamenať 2 skrátenia období pôvodných záznamov.

Tab. 19. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
VALIDTIME '[1995 - 2005)' DELETE FROM osoba WHERE id = 1;
```

Tab. 20. Tabuľka osôb po vykonaní sekvenčnej operácie DELETE

osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-2011)
1	Jožko	Mrkvička	[1999-∞)	[2000-2011)
1	Paľko	Mrkvička	[1989-1995)	[2011-∞)
1	Jožko	Mrkvička	[2005-∞)	[2011-∞)

2.6.4.3 Nesekvenčné modifikujúce príkazy

Nesekvenčné modifikujúce príkazy berú stĺpce určujúce obdobie platnosti dát, ako všetky ostatné používateľom definované stĺpce. K týmto stĺpcom možno priamo pristupovať pod názvom VALIDTIME a modifikovať ich (pri sekvenčných a aktuálnych príkazoch to nie je možné). Pri transakčnom čase, by to znamenalo možnosť meniť čas platnosti historických verzií, čo je v temporálnych databázach neželané. Preto sú nesekvenčné modifikujúce príkazy podporované len nad časom platnosti [5]. Nesekvenčné modifikujúce príkazy sa určujú kľúčovými slovami NONSEQUENCED VALIDTIME a nepovinným obdobím platnosti.

- INSERT – Umožňuje priamo v časti príkazu VALUES() určiť hodnotu stĺpca VALIDTIME. V nasledujúcom príkaze vložíme do pôvodnej tabuľky osôb (**Tab. 21**) nový záznam prostredníctvom nesekvenčného príkazu INSERT. Výsledný stav tabuľky osôb je znázornený v tabuľke **Tab. 22**.

Tab. 21. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
NONSEQUENCED VALIDTIME INSERT INTO osoba
   VALUES (2, 'Fero', 'Krátky', '[2011 - ∞)');
```

Tab. 22. Tabuľka osôb po vykonaní nesekvenčnej operácie INSERT

osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)
2	Fero	Krátky	[2011-∞)	[2011-∞)

– UPDATE – Umožňuje priamo meniť hodnotu stĺpca VALIDTIME. Nasleduje príklad nesekvenčného príkazu UPDATE nad pôvodnou tabuľkou osôb (**Tab. 23**), ktorej obsah po príkaze je zaznamenaný v tabuľke **Tab. 24**.

Tab. 23. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
NONSEQUENCED VALIDTIME UPDATE osoba
   SET meno = 'Viktor', VALIDTIME = '[1970 - 2003)'
   WHERE id = 1 and VALIDTIME = '[1999 - ∞)';
```

Tab. 24. Tabuľka osôb po vykonaní nesekvenčnej operácie UPDATE

osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-2011)
1	Viktor	Mrkvička	[1970-2003)	[2011-∞)

– DELETE – Umožňuje zmazať ľubovoľný riadok v tabuľke bez ohľadu na čas platnosti. Nasleduje príklad nesekvenčného príkazu DELETE z pôvodnej tabuľky osôb (**Tab. 25**), ktorej obsah po príkaze je zaznamenaný v tabuľke **Tab. 26**.

Tab. 25. Pôvodná tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-∞)

```
NONSEQUENCED VALIDTIME DELETE FROM osoba
   WHERE id = 1 and VALIDTIME = '[1999 - ∞)';
```

Tab. 26. Tabuľka osôb po vykonaní nesekvenčnej operácie DELETE

osoba				
id	**meno**	**priezvisko**	**validtime**	**transactiontime**
1	Paľko	Mrkvička	[1989-∞)	[1989-2000)
1	Paľko	Mrkvička	[1989-1999)	[2000-∞)
1	Jožko	Mrkvička	[1999-∞)	[2000-2011)

2.6.5 Rozšírenia obmedzení

SQL/Temporal rozširuje jednotlivé obmedzenia (CHECK, UNIQUE, FOREIGN KEY, ...) tak, že je možné rovnako ako ostatným príkazom, určiť či majú byť aktuálne, sekvenčné alebo nesekvenčné [5]:

- Aktuálne obmedzenia sú kontrolované len nad aktuálnymi verziami. Nevyžadujú si žiadnu zmenu definície oproti štandardnému SQL.
- Sekvenčné obmedzenia sú kontrolované v každom časovom momente. Na ich definovanie je potrebné pripojiť kľúčové slovo TRANSACTIONTIME a/alebo VALIDTIME pred názov obmedzenia.
- Nesekvenčné obmedzenia sú kontrolované bez ohľadu na čas. Na ich definovanie je potrebné pripojiť kľúčové slovo NONSEQUENCED TRANSACTIONTIME a/alebo NONSEQUENCED VALIDTIME pred názov obmedzenia.

2.6.6 Množinové operácie nad dotazmi

Množinové operácie v temporálnych tabuľkách si vyžadujú špeciálne rozšírenia len v prípade sekvenčných dotazov ktoré vracajú výsledky s temporálnou podporou [9]. Množinové operácie nad aktuálnymi a nesekvenčnými dotazmi sa správajú rovnako ako pri štandardnom SQL.

Základným predpokladom pri sekvenčných množinových operáciách je, aby výsledky oboch dotazov mali temporálnu podporu. Takéto operácie sú potom vyhodnocované v každom časovom okamihu a produkujú výsledok s temporálnou podporou. Majme výsledky dvoch príkazov SELECT s temporálnou podporou zobrazené v nasledujúcich tabuľkách (**Tab. 27** a **Tab. 28**).

Tab. 27. Výsledok prvého príkazu SELECT

Osoba			
Id	**meno**	**priezvisko**	**Validtime**
1	Paľko	Malý	[1989-1999)
4	Janko	Hraško	[2006 - 2020)
2	Jožko	Mrkvička	[1999-∞)

Tab. 28. Výsledok druhého príkazu SELECT

Osoba			
Id	**meno**	**priezvisko**	**Validtime**
1	Paľko	Malý	[1960-1980)
2	Jožko	Mrkvička	[1985-2005)
3	Peter	Krasko	[1960 - 1970)

Výsledok sekvenčnej množinovej operácie INTERSECT nad dátami z týchto tabuliek (**Tab. 27** a **Tab. 28**) je znázornený v tabuľke **Tab. 29**. Ak by sme nebrali do úvahy stĺpec validtime (napríklad pri nesekvenčných operáciách), tak vo výsledku by bol zahrnutý aj riadok s id = 1. Keďže záznamy s id = 1 neplatia súčasne ani v jednom časovom okamihu do výsledku sekvenčnej operácie zaradené nie sú.

Tab. 29. Výsledok sekvenčnej operácie INTERSECT nad temporálnymi výsledkami

Osoba			
id	**meno**	**priezvisko**	**Validtime**
2	Jožko	Mrkvička	[1999-2005)

2.6.7 Agregačné funkcie

Rovnako ako množinové operácie, aj agregačné funkcie ostávajú pri nesekvenčných a aktuálnych príkazoch nezmenené. Pri sekvenčných dotazoch sú vyhodnocované v každom časovom okamihu [9].

Nasledujúci príkaz znázorňuje výsledok sekvenčného príkazu SELECT s agregačnou funkciou COUNT() vykonaného nad tabuľkou osôb s podporou času platnosti, ktorá obsahuje dáta zobrazené v tabuľke **Tab. 30**. Z výsledku tohto dotazu (**Tab. 31**), ktorý obsahuje sumu miezd všetkých osôb, je vidieť, že suma je vyrátavaná v každom časovom

momente. V období [1960 - 1970) platil len záznam s id = 1, takže suma v tomto období je len 1000. V období [1970 - 1975) platili záznamy s id = 1 a id = 2, preto je suma v tomto období 1900.

Tab. 30. Tabuľka osôb

Osoba				
id	**meno**	**priezvisko**	**mzda**	**Validtime**
1	Paľko	Malý	1000	[1960-1980)
2	Janko	Hraško	900	[1970 - ∞)
3	Jožko	Mrkvička	1500	[1975-1990)

```
VALIDTIME SELECT COUNT(*) FROM osoba;
```

Tab. 31. Výsledok sekvenčného príkazu SELECT s agregačnou funkciou COUNT()

suma	**validtime**
1000	[1960-1970)
1900	[1970-1975)
3400	[1975-1980)
2400	[1980-1990)
900	[1990-∞)

2.6.8 DISTINCT

Kľúčové slovo DISTINCT sa používa na odstránenie duplicít z výsledku dotazu. Pri sekvenčných dotazoch je jeho funkcionalita iná ako pri štandardnom SQL. Duplicity vyhodnocuje v každom časovom momente, takže ak 2 riadky obsahujú rovnaké údaje, ale platia v rôznych obdobiach vo výsledku sa zobrazia oba [9].

2.6.9 JOIN

Kľúčové slovo JOIN sa používa pri dotazovaní z dvoch tabuliek založenom na vzťahu medzi niektorými stĺpcami v týchto tabuľkách. V temporálnych databázach prebieha spájanie tabuliek pri nesekvenčných a aktuálnych dotazoch rovnako ako v štandardnom SQL. Pri sekvenčných dotazoch sa platnosť vzťahu medzi tabuľkami vyhodnocuje v každom časovom okamihu [7]. Príklad použitia JOIN v sekvenčnom dotaze je uvedený v kapitole 2.4.2.

2.7 Existujúce riešenia

V tejto analyzujeme existujúce riešenia zabezpečujúce správu verzií dát. Zhodnotíme spôsob akým zabezpečujú temporálnu podporu a do akej miery sa pri tom držia spomínaných štandardov.

2.7.1 IBM DB2 10 for z/OS

IBM DB2 10 for z/OS je proprietárny databázový systém pre mainframe počítače. V rámci implementácie správy verzií dát zavádza nové možnosti pri vytváraní tabuliek, novú syntax a sémantiku dotazov a iné prvky vychádzajúce zo SQL/Temporal špecifikácie. IBM spolupracuje s komisiami ANSI a ISO na zahrnutí týchto rozšírení do ďalšej edície SQL štandardu [15].

Postup pri vytváraní bitemporálnej tabuľky [15]:

1. Vytvorenie základnej tabuľky pre aktuálne verzie dát. Pre podporu transakčného času musí tabuľka obsahovať 3 stĺpce dátového typu TIMESTAMP – 2 pre začiatočné/koncové body systémového času (sys_start, sys_end) a jeden pre začiatok transakcie (trans_start). Databáza automaticky generuje hodnoty týchto stĺpcov pri vložení modifikácii či zmazaní záznamu. Tieto stĺpce sú nemodifikovateľné, aby sa nedalo manipulovať s časmi platnosti verzií. Pre podporu času platnosti musí tabuľka obsahovať stĺpce určujúce obdobie platnosti (bus_start, bus_end)

```
CREATE TABLE policy (
   id INT NOT NULL,
   vin VARCHAR(10),
   annual_mileage INT,
   rental_car CHAR(1),
   coverage_amt INT,
   bus_start DATE NOT NULL,
   bus_end DATE NOT NULL,
   sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL,
   sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END NOT NULL,
   trans_start TIMESTAMP(12) GENERATED ALWAYS
   AS TRANSACTION START ID IMPLICITLY HIDDEN,
   PERIOD SYSTEM_TIME (sys_start, sys_end),
   PERIOD BUSINESS_TIME(bus_start, bus_end),
   PRIMARY KEY(id, BUSINESS_TIME WITHOUT OVERLAPS)
);
```

2. Vytvorenie historickej tabuľky s rovnakou štruktúrou ako má základná tabuľka. Do tejto tabuľky sa budú uchovávať neaktuálne verzie dát.

```
CREATE TABLE policy_history LIKE policy;
```

3. Upravenie základnej tabuľky tak, aby podporovala verziovanie použitím historickej tabuľky

```
ALTER TABLE policy ADD VERSIONING USE HISTORY TABLE policy_history;
```

Nad takto vytvorenou tabuľkou sa k aktuálnym dátam pristupuje pomocou štandardného príkazu SELECT z jazyka SQL. K starším verziám záznamov možno pristúpiť prostredníctvom rozšírení umožňujúcich zadanie času (transakčného času a/alebo času platnosti), z ktorého dáta požadujeme. Tento čas sa zadáva k jednotlivým tabuľkám vo FROM klauzule príkazu SELECT [21]. Príklad výberu historických verzií:

```
SELECT coverage_amt
    FROM policy FOR BUSINESS_TIME AS OF TIMESTAMP('2010-12-01')
    WHERE id = 1111;
```

Toto riešenie podporuje len nesekvenčnú operáciu INSERT, ktorej syntax sa nijak nelíši od štandardnej operácie. Hodnoty začiatku a konca platnosti záznamu sa priradia rovnako ako hodnoty ostatných stĺpcov.

```
INSERT INTO policy(id, vin, annual_mileage, rental_car,
   coverage_amt, bus_start, bus_end)
   VALUES(1111,'A1111',10000,'Y',500000,'2012-01-01','9999-12-31');
```

Pre vykonanie sekvenčného príkazu UPDATE umožňuje systém zadať obdobie platnosti prostredníctvom rozšírenia syntaxe SQL. Nesekvenčný UPDATE má syntax rovnakú ako v štandardnom SQL a umožňuje meniť hodnoty hraníc platnosti záznamu [21]. Príklad sekvenčného príkazu UPDATE:

```
UPDATE policy
   FOR PORTION OF BUSINESS_TIME FROM '2012-06-01' TO '9999-12-31'
   SET coverage_amt = 250000, rental_car='N'
   WHERE id = 1111;
```

Použitie príkazu DELETE je rovnaké ako použitie príkazu UPDATE.

Výhody riešenia:

- Riešenie je relatívne jednoduché na implementáciu v databázovom systéme
- Používateľ má kontrolu nad stĺpcami a tabuľkami, ktoré systém používa na správu historických dát.
- Jednoduché a jasné rozšírenie syntaxe príkazov pre prácu s historickými dátami
- Podporuje spätnú kompatibilitu so SQL

Nevýhody riešenia:

- Rozdielna syntax oproti SQL/Temporal
- Zložitejšie vytváranie historických tabuliek v porovnaní so štandardom SQL/Temporal, kde stačí pridať TRANSACTIONTIME a/alebo VALIDTIME za príkaz CREATE TABLE.
- Nepodporuje sekvenčné dotazy
- Nepodporuje dátový typ PERIOD
- Nepodporuje množinové operácie nad výsledkami sekvenčných dotazov
- Nepodporuje sekvenčné agregačné funkcie
- Nepodporuje sekvenčný INNER JOIN
- Nepodporuje sekvenčný OUTER JOIN
- Nepodporuje sekvenčný DISTINCT
- Nepodporuje nesekvenčné obmedzenia

Riešenie od IBM je zaujímavé z toho pohľadu, že necháva na používateľa vytváranie stĺpcov a historických tabuliek, ktoré bude databázový systém transparentne pre používateľa využívať pri správe historických dát. Ťažko povedať či by bolo vhodnejšie vykonávať tento proces automaticky, tak ako hovorí štandard SQL/Temporal. Pre používateľa by to znamenalo menej práce pri vytváraní temporálnych tabuliek, ale viac obmedzení (napríklad predpísané názvy stĺpcov a historických tabuliek).

2.7.2 Oracle 11g Workspace Manager

Workspace Manager je súčasťou proprietárnej databázy Oracle, ktorá umožňuje spravovať viacero verzií dát v tej istej databáze. Používa pracovné miesta (workspaces) ako virtuálne prostredie na izolovanie skupiny zmien nad produkčnými dátami, uchovávanie histórie zmien dát a vytváranie viacerých scenárov operácií nad dátami [14].

Workspace Manager umožňuje zapnúť správu verzií nad jednou alebo viacerými tabuľkami v databáze. Infraštruktúra pre správu verzií nie je viditeľná pre používateľov a SQL príkazy ako napríklad SELECT, INSERT, UPDATE a DELETE pracujú rovnako aj pri tabuľkách s verziovaním, hoci nie je možné upravovať primárny kľúč týchto tabuliek [14].

Správu verzií zabezpečuje PL/SQL balík. Keď sa nad tabuľkou spustí správa verzií, táto tabuľka je najprv premenovaná. Následne sa vytvorí pohľad s pôvodným menom

tabuľky, čím je zabezpečená transparentnosť pre aplikácie pracujúce s touto tabuľkou. Nad pohľadom sa vytvoria spúšťače zabezpečujúce vykonávanie DML operácií volaných nad pohľadom. Do pôvodnej tabuľky sa pridajú stĺpce pre metadáta, ktoré umožňujú existenciu viacerých verzií riadkov s rovnakým primárnym kľúčom [20].

Pracovné miesto je virtuálne prostredie, ktoré môžu viacerí používatelia zdieľať pri vykonávaní zmien nad dátami. Logicky zoskupuje nové verzie riadkov z jednej alebo viacerých verziovaných tabuliek a izolujú tieto verzie až kým nie sú explicitne zapracované do produkčných dát alebo zahodené [14].

Pracovné miesta môžu byť organizované do hierarchií. Napríklad jedno pracovné miesto môže byť rodičom jedného alebo viacerých pracovných miest (potomkov). Najvyššie v hierarchii je pracovné miesto s názvom LIVE. Je to základné pracovné miesto pre aktivity používateľov a produkčnú verziu dát. Používatelia v pracovnom mieste stále vidia konzistentný stav databázy. To znamená, že vidia zmeny vykonané v ich súčasnom pracovnom mieste plus zvyšok dát, tak ako existoval v čase, keď bolo pracovné miesto vytvorené alebo obnovené zmenami z rodičovského pracovného miesta [20].

Workspace Manager automaticky deteguje konflikty, čiže rozdiely v dátach, ktoré vznikli ako výsledok rôznych zmien nad tým istým riadkom v pracovnom mieste a v jeho rodičovskom pracovnom mieste [20]. Záchytné body (savepoints) sú body v pracovnom mieste, ku ktorým sa možno vrátiť po vykonaní zmien vo verziovaných tabuľkách. K záchytným bodom sa môžu používatelia vrátiť, aby si pozreli v akom stave bola databáza v danom bode [20]. Najdôležitejšou z pohľadu podpory transakčného času je možnosť uchovávať históriu všetkých zmien vykonaných nad záznamom (nie len poslednú) spolu s časom platnosti jednotlivých zmien.

Popri podpore transakčného času umožňuje Workspace Manager nad tabuľkou zapnúť aj podporu času platnosti. Pri tom sa do tabuľky pridá stĺpec vyjadrujúci obdobie platnosti záznamu. Následne je možné určiť obdobie platnosti (v minulosti, prítomnosti aj v budúcnosti) pre pripojenie k databáze. Workspace Manager zabezpečí, aby zadávané príkazy pracovali len v rámci tohto obdobia [14].

Príklad použitia nástroja Workspace Manager [20]:

1. Vytvorenie tabuľky pomocou štandardného príkazu CREATE TABLE.

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

2. Zapnutie verziovania nad tabuľkou s podporou času platnosti pomocou volania PL/SQL procedúry Workspace Managera. Výsledkom je bitemporálna tabuľka.

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

3. Naplnenie tabuľky:

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

INSERT INTO employees VALUES(
  'Coleman',
  50000,
  WMSYS.WM_PERIOD(TO_DATE('01-01-2003', 'MM-DD-YYYY'),
                  TO_DATE('12-31-9999', 'MM-DD-YYYY'))
);
```

4. Nastavenie obdobia platnosti v pripojení:

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

5. Modifikovanie záznamu sekvenčným príkazom UPDATE:

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

Všetky príkazy SELECT nad tabuľkou s podporou času platnosti berú do úvahy obdobie platnosti nastavené pre pripojenie. Pokiaľ nie je v podmienke príkazu určené inak, dotazy zobrazujú riadky tabuľky, ktorých obdobie platnosti sa prekrýva s obdobím platnosti pripojenia, a ktoré spĺňajú ostatné podmienky v dotaze.

Sekvenčný UPDATE sa vykoná vtedy ak sa v príkaze explicitne nemodifikuje obdobie platnosti. Pri tom sa berie do úvahy obdobie platnosti pripojenia. Na vykonanie nesekvenčnej zmeny stačí explicitne nastaviť obdobie platnosti požadovaného záznamu. Pre príkaz DELETE platia rovnaké podmienky ako pre príkaz UPDATE.

Výhody riešenia:

- Logické zoskupovanie verzií dát do pracovných miest
- Možnosť výberu medzi uchovávaním celej histórie verzií alebo iba záchytnej verzie, ku ktorej sa je možné vrátiť
- Podporuje spätnú kompatibilitu so SQL
- Podporuje dátový typ PERIOD
- Podporuje aktuálne a nesekvenčné dotazy

Nevýhody riešenia:

- Nutnosť volania uložených PL/SQL procedúr pri práci s temporálnou databázou
- Zložitejšie a menej prehľadné riešenie z hľadiska používania
- Riešenie neimplementuje žiadne rozšírenia syntaxe a sémantiky SQL
- Nepodporuje množinové operácie nad výsledkami sekvenčných dotazov
- Nepodporuje sekvenčné dotazy
- Nepodporuje sekvenčné agregačné funkcie
- Nepodporuje sekvenčný INNER JOIN
- Nepodporuje sekvenčný OUTER JOIN
- Nepodporuje sekvenčný DISTINCT
- Nepodporuje nesekvenčné obmedzenia

Riešenie od Oracle, využíva prostriedky databázového systému na implementáciu verziovania dát v jazyku PL/SQL. Hlavnou nevýhodou tohto riešenia je, že jeho použitie je zložitejšie, lebo je nutné poznať Workspace Manager API. Zaujímavá je možnosť vytvárania hierarchie pracovných miest. Riešenie dodržiava princípy podpory transakčného času opísanej SQL/Temporal, ale nijak nerozširuje syntax jazyka SQL pre prácu s historickými dátami.

2.7.3 Teradata Database 13.10

Teradata je proprietárny databázový systém s temporálnou podporou. Toto riešenie takmer presne dodržiava štandard SQL/Temporal. Líši sa len v spôsobe vytvárania temporálnych tabuliek.

```
CREATE MULTISET TABLE Policy(
   Policy_ID INTEGER,
   Customer_ID INTEGER,
   Policy_Type CHAR(2) NOT NULL,
   Policy_Details CHAR(40),
   Validity PERIOD(DATE) NOT NULL AS VALIDTIME
)
PRIMARY INDEX(Policy_ID);
```

V tomto prípade sa podpora transakčného času a času platnosti špecifikuje priamo pri stĺpcoch [16] a nie pre celú tabuľku ako hovorí štandard. Používateľ tak má možnosť určiť názvy stĺpcov.

Ostatné rozšírenia sémantiky a syntaxe štandardného SQL sú rovnaké ako pri SQL/Temporal a boli popísané v rámci analýzy tohto štandardu v kapitole 2.6.

Výhody riešenia:

- Syntax a sémantika rozšírení je súlade so štandardom SQL/Temporal
- Podporuje spätnú kompatibilitu so SQL
- Podporuje dátový typ PERIOD
- Podporuje všetky druhy dotazovacích príkazov
- Podporuje všetky druhy modifikujúcich príkazov
- Podporuje všetky druhy obmedzení
- Podporuje sekvenčný INNER JOIN

Nevýhody:

- Nepodporuje množinové operácie nad výsledkami sekvenčných dotazov
- Nepodporuje sekvenčné agregačné funkcie
- Nepodporuje sekvenčný OUTER JOIN
- Nepodporuje sekvenčný DISTINCT

Celkovo možno povedať, že toto riešenie je najprepracovanejšie medzi databázovými systémami s temporálnou podporou. Ponúka všetky druhy operácií potrebných pre základnú prácu s historickými dátami. Menej dôležité veci ako sekvenčné agregačné funkcie, sekvenčný DISTINCT a pod. síce nepodporuje, ale na rozdiel od ostatných riešení má nepodporovanú funkcionalitu dobre zdokumentovanú.

2.7.4 Porovnanie riešení

V nasledujúcej tabuľke (**Tab. 32**) je porovnanie riešení od IBM, Oracle a Teradata na základe súčastí temporálnych databáz, ktoré podporujú.

Tab. 32. Porovnanie riešení od IBM, Oracle a Teradata

	IBM	**Oracle**	**Teradata**
Rozšírenie syntaxe SQL	X		X
Spätná kompatibilita s netemporálnymi aplikáciami	X	X	X
Podpora transakčného času	X	X	X
Podpora času platnosti	X	X	X
SELECT			
Aktuálny SELECT	X	X	X
Sekvenčný SELECT	X*	X*	X
Nesekvenčný SELECT	X	X	X
INSERT			
Aktuálny INSERT	X	X	X
Sekvenčný INSERT	X	X	X
Nesekvenčný INSERT	X	X	X
UPDATE			
Aktuálny UPDATE	X	X	X
Sekvenčný UPDATE	X	X	X
Nesekvenčný UPDATE	X	X	X
DELETE			
Aktuálny DELETE	X	X	X
Sekvenčný DELETE	X	X	X
Nesekvenčný DELETE	X	X	X
Obmedzenia			
Aktuálne obmedzenia	X		X
Sekvenčné obmedzenia	X	X	X
Nesekvenčné obmedzenia			X
Ostatné			
Dátový typ PERIOD		X	X
Sekvenčné množinové operácie			
Sekvenčné agregačné funkcie			
Sekvenčný DISTINCT			
Sekvenčný JOIN			X

2.7.5 Ostatné riešenia

V tejto časti uvedieme pre úplnosť aj riešenia, ktoré zabezpečujú správu verzií dát mimo databázového systému:

- Asserted Versioning Framework [22] – je to middleware zabezpečujúci podporu bitemporálnych dát. Je použiteľný nad rôznymi databázovými systémami. Podporuje sekvenčné obmedzenia a modifikujúce operácie. Nepodporuje sekvenčné dotazy.
- IBM DataPropagator [23] – umožňuje použiť replikáciu databázového logu IBM DB2 na zachytenie stavov databázy pred a po každej modifikácii riadku a tak vytvoriť databázu s podporou transakčného času.
- LogExplorer [24] – je to analytický nástroj pre Microsoft SQLServer, ktorý umožňuje prehliadať logy a tak zistiť ako sa menili jednotlivé riadky.
- TimeDB [25] – je Java API, ktoré beží ako frontend nad databázou Oracle. Temporálne príkazy sú prekonvertované do sekvencií SQL-92 príkazov, ktoré sú vykonané v databáze. Poskytuje spätnú kompatibilitu a bitemporálnu podporu.
- Atempo Time Navigator [26] – nástroj na replikáciu dát zo systémov DB2, Oracle, SQL Server a Sybase, ktorý extrahuje informácie z databázy na vytváranie úložiska verzií. Následne je možné prezerať historické verzie dát a obnovovať ich.

2.7.6 Zhodnotenie existujúcich riešení

Analýza ukázala, že v súčasnosti je dostupných viacero riešení na správu historických dát. Z pohľadu temporálnych dát sú pre nás zaujímavé hlavne riešenia, ktoré temporálnu podporu zabezpečujú na databázovej úrovni a nie v middleware alebo v externej aplikácii. Zhodnotili sme 3 riešenia od IBM, Oracle a Teradata, ktoré realizujú vedenie histórie dát priamo v databáze. Ide však o proprietárne databázové systémy, čiže problémom ostáva absencia voľne dostupných temporálnych databáz.

2.8 Možnosti riešenia vo voľne dostupných databázových systémoch

Ako sme naznačili v kapitole 2.7.6, v súčasnosti neexistuje voľne dostupný databázový systém s temporálnou podporou. Z toho dôvodu treba zvážiť možnosti realizácie správy verzií v databázových systémoch s otvoreným kódom.

Tento problém by bolo najvhodnejšie riešiť priamo na databázovej vrstve bez nutnosti použitia nadstavieb nad databázovými systémami. Takéto riešenie by umožňovalo aplikáciám pracovať s databázou rovnakým spôsobom, ako keby správu verzií

nepodporovala. Tým by sa ušetrilo veľa času, ktorý sa musí stráviť návrhom a vývojom správy verzií mimo databázovej úrovne pre každý systém špeciálne. V tejto časti zanalyzujeme možnosti implementácie správy historických dát vo voľne dostupných databázových systémoch.

2.8.1 Využitím prostriedkov databázy

Pri tejto možnosti, by sa na správu historických dát využili prostriedky akými sú napríklad spúšťače (triggre), pohľady a uložené procedúry. Riešenie by spočívalo v možnosti zapnúť temporálnu podporu nad vybranými netemporálnymi tabuľkami. Výhodou je, že v tomto prípade nie je nutný zásah do zdrojových kódov databázy a rozširovanie syntaxe príkazov, čiže pri dodržaní štandardov, by bolo riešenie možné použiť aj na iných databázových systémoch. Nevýhodou je nutnosť používania uložených procedúr, potrebných pri zapínaní/vypínaní temporálnej podpory tabuliek, pri úpravách dátového modelu (napríklad pridávanie/odoberanie tabuliek a stĺpcov, premenovávaní objektov...) a pri iných podobných činnostiach.

2.8.2 Úpravou zdrojových kódov

Pri tejto možnosti, by sa na úrovni zdrojových kódov databázy zmenilo vykonávanie príkazov na vytváranie tabuliek a čítanie a modifikáciu údajov z tabuliek tak, aby bola správa historických dát podporovaná priamo vnútri databázového systému. Nevýhodou je nutnosť zásahu do zdrojového kódu a viazanosť len na jeden databázový systém. Výhodou je bezproblémová podpora všetkých operácií nad dátovým modelom.

2.9 Stanovenie cieľov projektu

V predchádzajúcich častiach analýzy sme opísali základy temporálnych databáz a predstavili sme existujúce riešenia v tejto oblasti. Zistili sme, že v súčasnosti neexistuje žiadna voľne dostupná databáza ponúkajúca temporálnu podporu. Na základe toho sme stanovili hlavný cieľ projektu, ktorým je vytvorenie temporálnej podpory pre voľne dostupný databázový systém vychádzajúc zo štandardu SQL/Temporal s dôrazom na minimalizáciu obmedzení na existujúce aplikácie.

Keďže vytvorenie kompletného riešenia, ktoré by poskytovalo podporu transakčného času aj času platnosti ja nad časový rozsah tohto projektu, rozhodli sme sa pre riešenie len

s podporou transakčného času (rollback databáza – viď kapitolu 2.2.3). Rollback databázy majú široké možnosti uplatnenia napríklad:

- v systémoch pracujúcich s kritickými dátami, ktorých strata v dôsledku vykonania nesprávnych modifikujúcich príkazov by znamenala veľké škody.
- v systémoch, ktoré potrebujú udržiavať históriu všetkých zmien nad dátami (napr. kvôli bezpečnosti)
- v systémoch vyžadujúcich možnosť obnovy predchádzajúcich stavov databázy

V kapitole 2.8 sme predstavili dve možnosti riešenia vo voľne dostupných databázových systémoch, z ktorých každá má svoje výhody aj nevýhody. Kvôli presnejšiemu stanoveniu obmedzení oboch riešení je ich potrebné aspoň nahrubo načrtnúť v konkrétnom databázovom systéme.

Pri voľbe voľne dostupného databázového systému, pre ktorý bude riešenie určené, prichádzali do úvahy MySQL a PostgreSQL, ktoré sú v súčasnosti najpopulárnejšie. Je ťažké tieto dve databázy porovnávať, pretože každá má svoje výhody a tak voľba často závisí od potrieb projektu alebo preferencií používateľa. My sme sa rozhodli pre PostgreSQL, z dôvodu jeho lepšej vybavenosti a možnostiam zachovania dátovej integrity [17] [18]. Napríklad:

- cudzie kľúče – v MySQL sú podporované len v innoDB engine.
- CHECK obmedzenia
- Spúšťače – v MySQL je možné vytvoriť z každého typu len jeden spúšťača a navyše sa nespúšťajú pri kaskádových zmenách a mazaniach riadkov.
- Čiastočné indexy

Vo zvyšnej časti analýzy načrtneme možnosti riešenia podpory transakčného času v databáze PostgreSQL prostredníctvom využitia prostriedkov databázy a prostredníctvom úpravy zdrojových kódov. Následne, na základe zistených obmedzení zhodnotíme, ktorému z riešení sa bude vhodnejšie venovať vo zvyšnej časti projektu.

2.10 Podpora transakčného času pomocou prostriedkov databázy

V tejto časti sa venujeme analýze prostriedkov potrebných na zabezpečenie správy verzií v databáze PostgreSQL, načrtneme riešenie bez modifikácie zdrojových kódov a zhodnotíme

ho z pohľadu podpory temporálnych a netemporálnych operácií. V nasledujúcich kapitolách 2.10.1, 2.10.2 a 2.10.3 vychádzame z dokumentácie k PostgreSQL [17].

2.10.1 Spúšťače

Spúšťače umožňujú špecifikovať funkcie, ktoré má databáza zavolať pri vykonaní určenej operácie. Spúšťače môžu byť definované, tak aby sa spustili pred alebo po operácii INSERT, UPDATE alebo DELETE, buď pre každý modifikovaný riadok alebo len raz pre SQL príkaz. UPDATE spúšťače môžu byť navyše nastavené tak, aby sa spustili len ak sú v časti SET príkazu UPDATE spomenuté vybrané stĺpce. Spúšťače môžu byť vykonané aj pri príkaze TRUNCATE .

Pred vytvorením spúšťača musí byť vytvorená funkcia bez argumentov, ktorá vracia typ trigger. Funkcie pre spúšťače dostávajú vstupné parametre cez špeciálne predávanú štruktúru TriggerData.

Ak je k jednej udalosti nad tou istou reláciou definovaných viacero spúšťačov, tieto budú vykonané v abecednom poradí podľa názvu. V prípade spúšťačov vykonávaných pred operáciou je modifikovaný riadok, ktorý vracia jeden spúšťač vstupom nasledujúceho. Ak niektorý z nich vráti NULL, nasledujúce sa nevykonajú.

Ak sa v spúšťači vykonávajú nejaké operácie, tieto môžu spúšťať ďalšie spúšťače. To sa nazýva kaskádovanie spúšťačov, pričom počet vnorení nie je databázovým systémom nijak limitované. Takýmto spôsobom je možné vykonávať spúšťač aj rekurzívne a je na programátorovi, aby sa vyhol nekonečným rekurziám.

Pri verziovaní môžu spúšťače poslúžiť napríklad pri vytváraní historických verzií modifikovaných záznamov alebo nastavovanie metadát (napríklad čas modifikácie, verzia a operácia) pred vložením alebo úpravou.

2.10.2 Pohľady a pravidlový systém

Pravidlový systém PostgreSQL (presnejšie povedané systém prepisovania príkazov) je úplne odlišný od uložených procedúr a spúšťačov, ktoré sú bežné aj vo väčšine databázových systémov. Prostredníctvom definovaných pravidiel modifikuje príkazy a potom ich posúva plánovaču na vytvorenie plánu a následné vykonanie.

Pravidlový systém sa nachádza medzi parserom a plánovačom. Jeho vstupmi sú strom dotazu z parseru a používateľmi definované pravidlá, ktoré sú vlastne tiež stromy

a výstupom je 0 alebo viac stromov. To znamená, že jeho vstupy a výstupy sú štruktúry, ktoré produkuje parser, a tak jediné s čím pravidlový systém pracuje sú reprezentácie SQL príkazov.

Pravidlá umožňujú určiť jeden alebo viac príkazov, ktoré sa majú vykonať namiesto alebo súčasne so zadaným príkazom. Zjednodušene sa dá povedať, že pravidlový systém buď zadaný príkaz nahradí inými príkazmi, alebo k nemu pridá ďalšie príkazy. Použitím pravidlového systému sú v PostgreSQL implementované aj pohľady. V skutočnosti nie je žiadny rozdiel medzi príkazom

```
CREATE VIEW myview AS SELECT * FROM mytab;
```

v porovnaní s dvoma príkazmi

```
CREATE TABLE myview (same column list as mytab);
CREATE RULE "_RETURN" AS ON SELECT TO myview DO INSTEAD
    SELECT * FROM mytab;
```

pretože tieto vykoná vnútorne príkaz CREATE VIEW. Dôsledkom toho je, že informácie o pohľadoch v systémových katalógoch sú úplne rovnaké ako pri tabuľkách, takže pre parser nie je žiadny rozdiel medzi tabuľkou a pohľadom (obe sú pre neho relácie).

Na to, aby sa nad pohľadmi dali vykonávať príkazy INSERT, UPDATE, DELETE, je nutné pre každý z nich vytvoriť pravidlo, ktoré zabezpečí ich vykonanie nad potrebnými tabuľkami a nie nad pohľadom, ktorý je vlastne len SELECT z nejakej tabuľky.

2.10.3 Dedenie

Dedenie síce nepotrebujeme na riešenie správy verzií, ale je nutné na túto operáciu brať ohľad. Fungovanie dedenia v PostgreSQL je najlepšie vysvetliť na príklade. Majme tri nasledovne definované tabuľky:

```
CREATE TABLE osoba (
  id integer,
  meno text,
  priezvisko text
);

CREATE TABLE zamestnanec (
  id_oddelenia integer
) INHERITS (osoba);

CREATE TABLE veduci (
  pocet_podriadenych integer
) INHERITS (zamestnanec);
```

V tomto príklade dedí zamestnanec stĺpce osoby a vedúci stĺpce zamestnanca. Zamestnanec má oproti osobe navyše id oddelenia, pod ktoré patrí a vedúci má ešte navyše počet podriadených zamestnancov.

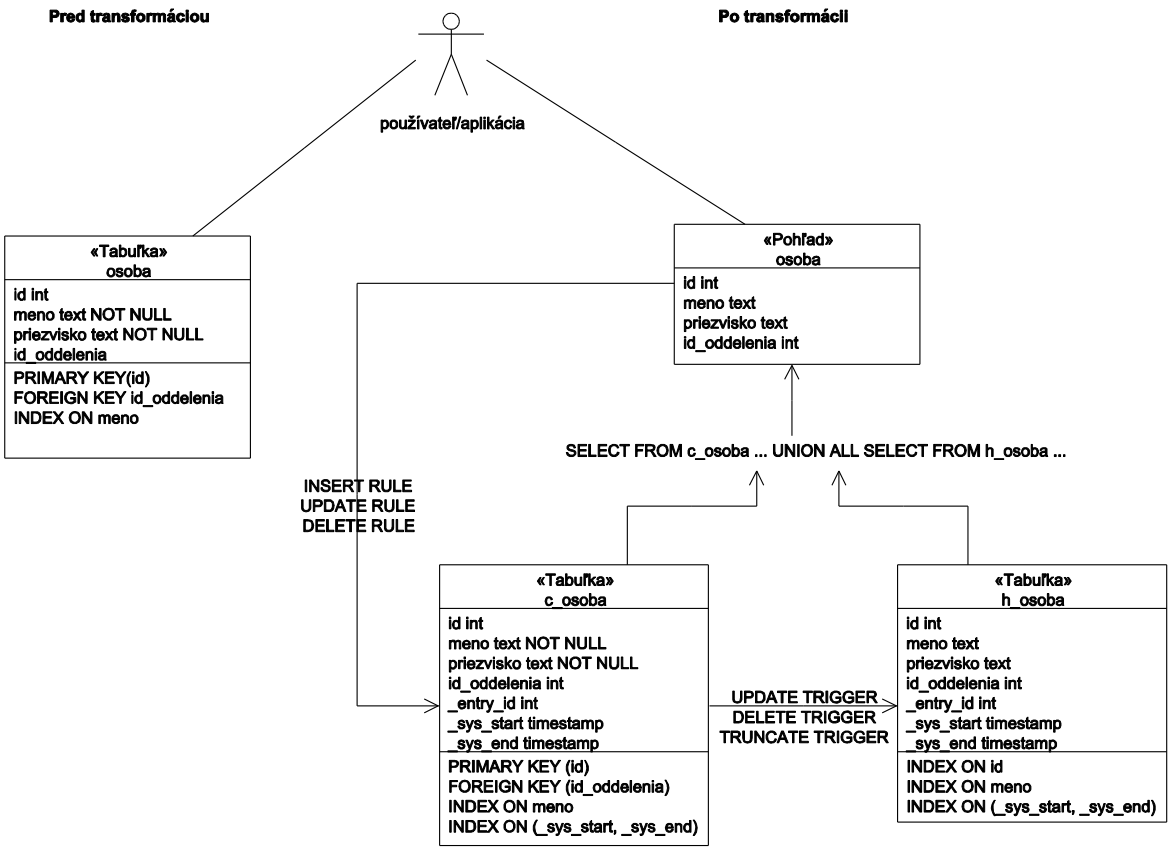
Tabuľka môže dediť od viacerých rodičovských tabuliek. V tom prípade bude obsahovať zjednotenie stĺpcov jej rodičov. Stĺpce detskej tabuľky sú k týmto stĺpcom pridané. Ak sa v dvoch rodičovských tabuľkách alebo v rodičovskej a detskej tabuľke vyskytnú stĺpce s rovnakým názvom, budú zlúčené do jedného. Na to, aby mohli byť stĺpce zlúčené, musia byť rovnakého typu, inak systém vyhlási chybu. Zlúčené stĺpce majú kópiu CHECK obmedzení všetkých rodičov a sú NOT NULL, ak aspoň jeden pôvodný bol NOT NULL. Ostatné typy obmedzení (UNIQUE, primárne kľúče a cudzie kľúče) sa nededia.

Príkazy SELECT môžu vracať výsledky len zo zvolenej tabuľky (použitím kľúčového slova ONLY) alebo zo zvolenej tabuľky a všetkých tabuliek, ktoré od nej dedia. Čo sa týka DML operácií, príkaz INSERT neberie dedenie do úvahy a vkladá údaje len do zvolenej tabuľky. Naopak, príkaz UPDATE pracuje aj nad riadkami v tabuľkách, ktoré dedia od modifikovanej tabuľky. Toto správanie je možné potlačiť použitím kľúčového slova ONLY.

2.10.4 Náčrt riešenia

Nami navrhované riešenie podpory transakčného času prostredníctvom prostriedkov databázy spočíva vo vytvorení uložených procedúr v jazyku PL/SQL, ktoré budú zabezpečovať transformáciu vybranej tabuľky na tabuľku s podporou transakčného času a taktiež spätnú transformáciu na tabuľku bez temporálnej podpory.

Pri transformácii sa na základe údajov o tabuľke získaných zo systémových katalógov, vytvoria v databáze nové databázové objekty (pohľady, spúšťače, pravidlá, tabuľky), ktoré budú spolu zabezpečovať správu verzií (**Obr. 6**). V tejto časti sa venujeme analýze týchto objektov.



Obr. 6. Podpora transakčného času použitím prostriedkov databázy

2.10.4.1 Aktuálna tabuľka

Pôvodná tabuľka je v rámci transformácie premenovaná (pridaná predpona ‘c_‘ ako current) a sú do nej pridané metadátové stĺpce. Pôvodné stĺpce, obmedzenia, spúšťače a pravidlá definované nad touto tabuľkou sa nemenia. Táto tabuľka slúži na uchovávanie aktuálnych verzií záznamov. Metadátové stĺpce sú definované ako:

```
_entry_id serial,
_sys_start timestamp DEFAULT clock_timestamp(),
_sys_end timestamp DEFAULT '294276-12-31 23:59:59.999999 AD'
```

A ich význam je nasledovný:

- _entry_id – jednoznačný identifikátor záznamu. Združuje verzie jedného záznamu. Je dôležitý, napríklad ak používateľ nedefinuje žiadny primárny kľúč alebo keď zmení hodnotu primárneho kľúča. V takýchto prípadoch by sme nevedeli, ktoré verzie patria k sebe.

- _sys_start – začiatok platnosti verzie. Nastavuje sa na aktuálny čas (funkcia clock_timestamp()).
- _sys_end – koniec platnosti verzie. Nastavuje sa na maximálny čas ('294276-12-31 23:59:59.999999 AD').

Všetky metadátové stĺpce majú nastavenú východiskovú hodnotu, ktorá sa nastaví novo vloženým riadkom. Preto tieto stĺpce nie je potrebné uvádzať v pravidle pre presmerovanie príkazu INSERT z pohľadu na aktuálnu tabuľku.

Na to, aby bola história v konzistentnom stave, je potrebné zamedziť vykonávaniu všetkých operácií INSERT a UPDATE nad metadátovými stĺpcami, vhodným nastavením používateľských oprávnení aplikačného používateľa. Týmto spôsobom síce nie je možné zamedziť vykonávaniu týchto operácií všetkým používateľom (super používateľ bude mať vždy tú možnosť), ale je to nevyhnutné preto, aby sa zamedzilo neúmyselnému poškodeniu dát používateľom, pod ktorým beží aplikácia. Super používateľ je používateľ, pod ktorým sa vytvára inštancia databázového servera (príkaz initdb) a zvyčajne, ak používateľ nezvolí inak, je nazvaný postgres. Je vlastníkom všetkých systémových katalógov a môže vykonávať všetky príkazy nad všetkými objektmi bez ohľadu na oprávnenia. Preto je zbytočné brániť super používateľovi vo vykonávaní akýchkoľvek operácií, má totiž veľa možností ako obmedzenia obísť (napríklad môže modifikovať systémové katalógy alebo spúšťať ľubovoľný kód).

2.10.4.2 Historická tabuľka

Po vytvorení aktuálnej tabuľky sa vytvorí tabuľka na odkladanie historických verzií záznamov, ktorej názov sa skladá z názvu pôvodnej tabuľky a predpony ‘h_‘ (historical). Táto tabuľka obsahuje rovnaké stĺpce ako aktuálna tabuľka.

Historická tabuľka neobsahuje obmedzenia, pravidlá a spúšťače z pôvodnej tabuľky ani východiskové hodnoty metadátových stĺpcov. Na vytvorenie takejto kópie tabuľky sa používa príkaz LIKE. Napríklad tabuľka h_osoba z obrázku sa vytvorí pomocou príkazu

```
CREATE TABLE h_osoba (LIKE c_osoba);
```

Podobne ako pre zamedzenie modifikovania metadátových stĺpcov je potrebné, vhodným nastavením používateľských oprávnení, zakázať modifikujúce operácie nad historickou tabuľkou.

Na druhej strane potrebujeme odkladať staré verzie záznamov prostredníctvom spúšťačov. Preto je potrebné aby funkcie na odkladanie starých záznamov boli vytvorené rovnakým používateľom ako historické tabuľky a boli definované s parametrom SECURITY DEFINER, ktorý zabezpečí, že sa budú vykonávať pod používateľom, ktorý ich vytvoril (je ich vlastníkom).

2.10.4.3 Pohľad

Namiesto pôvodnej tabuľky, pristupuje po transformácii aplikácia alebo používateľ k pohľadu s rovnakým menom ako mala tabuľka pred transformáciou. Tento pohľad je definovaný ako zjednotenie výsledkov dvoch príkazov SELECT. Je vhodné použiť operáciu UNION ALL, pretože tá iba spojí dokopy výsledky oboch príkazov SELECT. Ak by sme použili len UNION, zjednotenie by sa vykonávalo použitím hešovacej tabuľky kvôli odstráneniu duplicít vo výsledku, čo by bolo pomalšie. Napríklad definícia pohľadu znázorneného na obrázku je:

```
CREATE VIEW osoba AS
   SELECT id, meno, priezvisko, id_oddelenia
      FROM c_osoba
      WHERE temporal.timestamp BETWEEN _sys_start AND _sys_end
   UNION ALL
   SELECT id, meno, priezvisko, id_oddelenia
      FROM h_osoba
      WHERE temporal.timestamp BETWEEN _sys_start AND _sys_end
;
```

Pohľad obsahuje rovnaké stĺpce ako pôvodná tabuľka, takže pre aplikáciu alebo používateľa sa javí ako pôvodná tabuľka. Zmenou premennej temporal.timestamp, ktorá musí byť definovaná v konfiguračnom súbore postgresql.conf, je možné vykonávať rezy v čase. Pohľad obsahuje len tie dáta, ktoré sú platné v čase nastavenom v premennej temporal.timestamp bez ohľadu na to, či ide o aktuálne alebo historické verzie. Ak chceme zobrazovať vždy len aktuálne verzie, je potrebné nastaviť premennú na čas '294276-12-31 23:59:59.999999 AD', čo je maximálna hodnota pre dátový typ timestamp.

2.10.4.4 Pravidlá

Aby bolo možné vykonávať nad pohľadom modifikujúce operácie rovnako ako nad pôvodnou tabuľkou, je nutné vytvoriť pravidlá, ktoré slúžia na „presmerovanie" operácií z pohľadu na tabuľku s aktuálnymi verziami. Príklady pravidiel pre tabuľku osôb z obrázku:

```
CREATE OR REPLACE RULE "_INSERT" AS ON INSERT TO osoba DO INSTEAD
   INSERT INTO c_osoba(id, meno, priezvisko, id_oddelenia)
      VALUES (new.id,
              new.meno,
              new.priezvisko,
              new.id_oddelenia
             )
;

CREATE OR REPLACE RULE "_UPDATE" AS ON UPDATE TO osoba DO INSTEAD
   UPDATE c_osoba
      SET id = new.id,
          meno = new.meno,
          priezvisko = new.priezvisko,
          id_oddelenia = new.id_oddelenia,
          _sys_start = case when xmin::text::bigint = txid_current()
                        then _sys_start
                        else clock_timestamp()
                       end
      WHERE id = old.id
;

CREATE OR REPLACE RULE "_DELETE" AS ON DELETE TO osoba DO INSTEAD
   DELETE FROM c_osoba
      WHERE id = old.id
;
```

V pravidle pre UPDATE je zahrnuté aj nastavovanie hodnôt metadátových stĺpcov. Vytvára sa nová verzia záznamu, takže je potrebné nastaviť nový začiatok platnosti (_sys_start). Pre pochopenie tejto operácie je potrebné vysvetliť správanie transakcií pri príkaze UPDATE.

Ak sa v rámci transakcie vykoná príkaz UPDATE nad záznamom, transakcia uzamkne tento záznam až do jej konca. Dôsledkom je, že ostatné transakcie tento záznam nemôžu modifikovať (čakajú na koniec transakcie, ktorá ho má uzamknutý) a zároveň nevidia zmeny, ktoré nad ním transakcia vykonáva. Keď transakcia skončí, buď nie je viditeľná žiadna zmena (skončila s vrátením zmien - ROLLBACK), alebo je viditeľná len posledná zmena vykonaná nad záznamom (skončila s potvrdením - COMMIT). Okolie teda vidí len stavy záznamu pred uzamknutím (prvou modifikáciou) a následne až po skončení transakcie. To, čo so záznamom vykonáva transakcia od jeho uzamknutia po koniec transakcie je pre ostatné transakcie neviditeľné. To je dôvod, prečo je nevyhnutné, aby sme riadok do historickej tabuľky odložili, len ak ide o prvý UPDATE v transakcii. V opačnom prípade by bolo možné vidieť verzie záznamu vytvorené vo vnútri transakcie, čo nie je bežné správanie a navyše by tieto verzie mohli byť nekonzistentné s ostatnými dátami v databáze, napríklad preto, že obmedzenia (UNIQUE, cudzie kľúče a iné) môžu byť nastavené tak, že sa kontrolujú až na konci transakcie a nie po každom príkaze.

Preto sa hodnota stĺpca _sys_start volí na základe toho, či je záznam v aktuálnej transakcii modifikovaný prvý krát. To sa dá jednoducho zistiť porovnaním hodnoty

systémového stĺpca xmin (obsahuje ID poslednej transakcie, ktorá riadok modifikovala) s ID aktuálnej transakcie (získané prostredníctvom funkcie txid_current()). Ak sa hodnoty rovnajú, znamená to, že záznam bol už v rámci aktuálnej transakcie modifikovaný a začiatok platnosti sa mu nezmení. Inak sa nastaví na aktuálny čas, čím sa vlastne zaznamená čas uzamknutia záznamu transakciou.

Zvýšenú pozornosť si vyžadujú aj NOT NULL DEFAULT stĺpce pri INSERT pravidlách a WHERE podmienky pri UPDATE a DELETE pravidlách. Na to, aby fungovalo vkladanie DEFAULT hodnôt pri nevymenovaní stĺpca v príkaze INSERT, je ich potrebné priradiť aj stĺpcom pohľadu. Na to slúži príkaz ALTER VIEW.

```
ALTER VIEW osoba ALTER id SET DEFAULT nextval('osoba_id_seq'::regclass)
```

Zadaná default hodnota sa vloží do príkazu INSERT pri nevymenovaní stĺpca ešte pred aplikovaní pravidla.

WHERE podmienka v pravidlách UPDATE a DELETE musí jasne identifikovať riadok, ktorý sa má modifikovať. Ak pôvodná tabuľka obsahuje primárny kľúč alebo UNIQUE NOT NULL stĺpec, tento možno použiť v podmienke na jasnú identifikáciu riadku. V inom prípade je riadok identifikovaný kombináciou všetkých stĺpcov a v podmienke musia byť uvedené všetky.

2.10.4.5 Spúšťače

Samotné vytváranie verzií zabezpečujú spúšťače definované na tabuľke s aktuálnymi verziami. Tieto sa spustia pre každý riadok (FOR EACH ROW) po vykonaní (AFTER) operácii UPDATE alebo DELETE a raz za príkaz (FOR EACH STATEMENT) pred vykonaním (BEFORE) operácie TRUNCATE. Príkaz TRUNCATE síce nie je možné zavolať nad pohľadom, ale je vhodné mať pre neho spúšťač, ak by bol zavolaný priamo nad aktuálnou tabuľkou. Operácia INSERT nevyžaduje manipuláciu s historickou tabuľkou, a tak pre ňu spúšťač nie je potrebný. V spúšťačoch sa pre jednotlivé príkazy vykonajú nasledovné kroky:

- UPDATE – pôvodná verzia záznamu sa vloží do historickej tabuľky, ale s hodnotou stĺpca _sys_end nastavenou na čas o jednu mikrosekundu nižší ako _sys_start novej verzie záznamu. Ak sa v transakcii vykoná viac príkazov UPDATE nad tým istým záznamom, je potrebné do historickej tabuľky odložiť len verziu existujúcu pred

začiatkom transakcie. Preto ak sa ID poslednej modifikujúcej transakcie nového (new.xmin) a starého (old.xmin) riadku rovnajú odloženie do historickej tabuľky sa nevykoná. Pri vykonaní príkazu UPDATE nad pohľadom sa teda v skutočnosti vykonajú príkazy UPDATE a INSERT.

```
CREATE OR REPLACE FUNCTION update_history() RETURNS trigger AS
$$
BEGIN
    IF new.xmin = old.xmin THEN
        RETURN null;
    END IF;

    INSERT INTO h_osoba
        VALUES(
            old.id,
            old.meno,
            old.priezvisko,
            old.id_oddelenia
            old._entry_id,
            old._sys_start,
            new._sys_start - interval '1 microsecond'
        )
    ;
    RETURN null;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_history
    AFTER UPDATE ON c_osoba
    FOR EACH ROW EXECUTE PROCEDURE update_history()
;
```

– DELETE – pôvodná verzia záznamu sa vloží do historickej tabuľky, ale s hodnotou stĺpca _sys_end nastavenou na aktuálny čas. Pri vykonaní príkazu DELETE nad pohľadom sa teda v skutočnosti sa vykonajú príkazy DELETE a INSERT.

```
CREATE OR REPLACE FUNCTION delete_history() RETURNS trigger AS
$$
BEGIN
    INSERT INTO h_osoba
        VALUES(
            old.id,
            old.meno,
            old.priezvisko,
            old.id_oddelenia,
            old._entry_id,
            old._sys_start,
            clock_timestamp()
        )
    ;
    RETURN null;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER delete_history
    AFTER DELETE ON c_osoba
    FOR EACH ROW EXECUTE PROCEDURE delete_history()
;
```

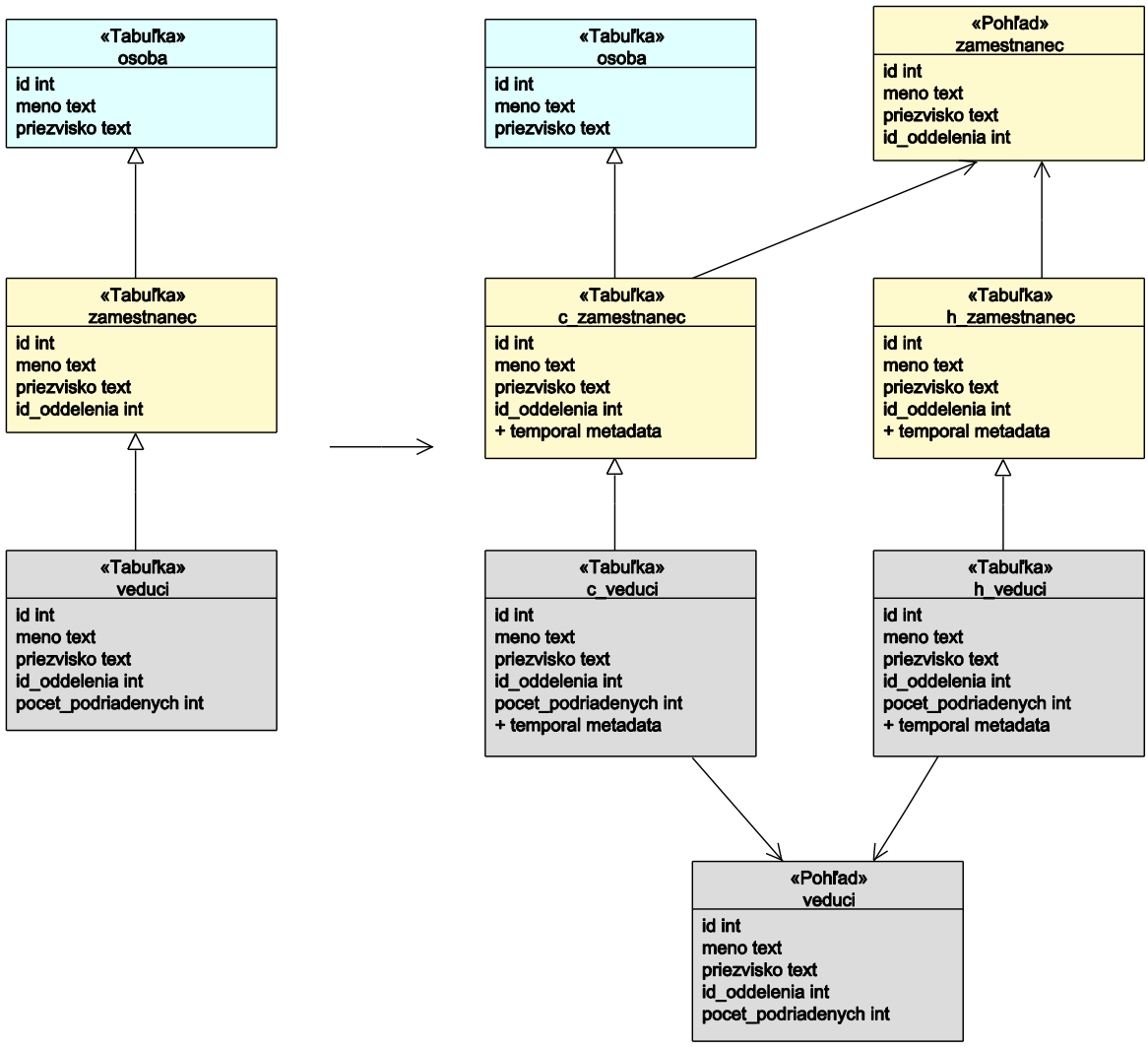
– TRUNCATE – obsah celej aktuálnej tabuľky sa vloží do historickej, ale s hodnotou stĺpca _sys_end nastavenou na aktuálny čas. Pri vykonaní príkazu TRUNCATE nad pohľadom sa teda v skutočnosti vykonajú príkazy INSERT, SELECT a TRUNCATE.

```
CREATE OR REPLACE FUNCTION truncate_history() RETURNS trigger AS
$$
BEGIN
    INSERT INTO h_osoba
        (SELECT
                id,
                meno,
                priezvisko,
                id_oddelenia,
                _entry_id,
                _sys_start,
                clock_timestamp()
            FROM osoba
        )
    ;
    RETURN null;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER truncate_history
    AFTER DELETE ON c_osoba
    FOR EACH STATEMENT EXECUTE PROCEDURE truncate_history()
;
```

2.10.4.6 Dedenie

Ak je použité dedenie, situácia sa trochu komplikuje. Je potrebné brať do úvahy všetky vlastnosti dedenia PostgreSQL. Majme príklad z kapitoly 2.10.3. Potrebujeme napríklad

zapnúť podporu transakčného času na tabuľke zamestnanec. Na **Obr. 7** je znázornená hierarchia tabuliek pred a po tejto operácii.



Obr. 7 Podpora transakčného času v hierarchii dedenia

Aby sa zachovali všetky vlastnosti dedenia aj po zapnutí podpory transakčného času na jednej z tabuliek v hierarchii dedenia, je nutné, aby sa podpora transakčného času zapla aj na všetkých potomkoch (ak ešte nie je zapnutá). V našom prípade má tabuľka zamestnanec jediné dieťa a tým je tabuľka vedúci.

Okrem vzťahov dedenia medzi tabuľkami s aktuálnymi záznamami je potrebné vytvoriť vzťahy dedenia aj medzi historickými tabuľkami. Bez toho by totiž nefungovali správne príkazy SELECT.

Pohľad zamestnanec je definovaný ako:

```
CREATE VIEW zamestnanec AS
   SELECT id, meno, priezvisko, id_oddelenia
      FROM c_zamestnanec
      WHERE temporal.timestamp BETWEEN _sys_start AND _sys_end
   UNION ALL
   SELECT id, meno, priezvisko, id_oddelenia
      FROM h_zamestnanec
      WHERE temporal.timestamp BETWEEN _sys_start AND _sys_end
;
```

Prvý SELECT berie kvôli dedeniu riadky z tabuliek c_zamestnanec a c_veduci. Druhý SELECT z tabuliek h_zamestnanec a h_veduci. Týmto je zabezpečené správne chovanie systému aj pri pohľadoch do minulosti.

Modifikujúce operácie, vďaka použitiu spúšťačov, nepredstavujú pre toto riešenie žiadny problém. Ak je modifikujúca operácia zavolaná nad rodičovskou tabuľkou, ale kvôli dedeniu sa v skutočnosti vykoná nad detskou tabuľkou, pôvodná verzia riadku sa správne uloží do historickej tabuľky prislúchajúcej k detskej tabuľke.

2.10.5 Temporálna podpora a obmedzenia riešenia

Z hľadiska temporálnej podpory riešenie využitím prostriedkov databázy poskytuje:

- Spätnú kompatibilitu s netemporálnymi aplikáciami
- Možnosť návratu k dátam, ktoré platili v určitom časovom momente v minulosti.

Riešenie má nasledovné obmedzenia:

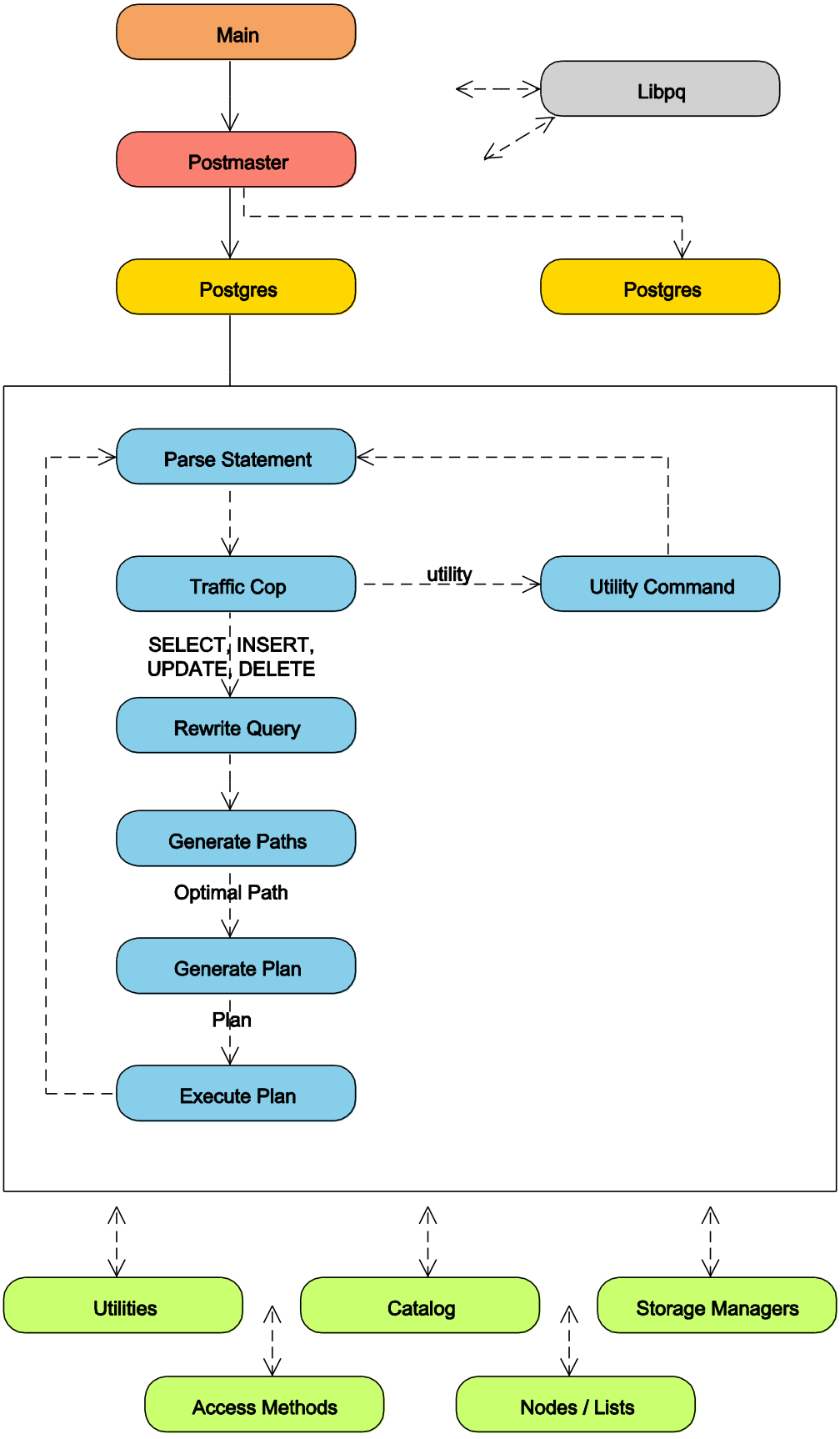
- Spúšťače – problémové sú UPDATE spúšťače, ktoré sú definované tak, že sa spustia len ak sú vybrané stĺpce spomenuté v SET časti príkazu UPDATE. Keďže pravidlá prepisujú UPDATE príkazy tak, že sa v časti SET vždy vymenujú všetky stĺpce, bez ohľadu na to, ktoré zadal používateľ, takýto spúšťač sa vykoná vždy. Okrem toho, pridanie metadátových stĺpcov môže spôsobiť nefunkčnosť používateľských spúšťačov, ktoré pracujú s celými štruktúrami OLD a NEW (napríklad ich niekam odkladajú) a spoliehajú sa na to, že obsahujú len pôvodné stĺpce.
- Príkaz ALTER/DROP TABLE – tento príkaz sa nedá priamo použiť, keďže pôvodná tabuľka je nahradená pohľadom. Pred a po volaní príkazu ALTER/DROP TABLE je nutné zavolať špeciálne uložené procedúry, ktoré zabezpečia, že zmeny v štruktúre tabuľky sa odrazia aj na všetkých objektoch potrebných pre verziovanie.
- Príkaz TRUNCATE – pre tento príkaz PostgreSQL nedovoľuje vytvoriť pravidlá, a tak ho nie je možné zavolať nad pohľadom, ktorý nahrádza pôvodnú tabuľku.

2.11 Podpora transakčného času na úrovni zdrojových kódov

Databázový systém PostgreSQL má voľne dostupné zdrojové kódy, a tak je možnosť implementovať podporu transakčného času priamo vo vnútri systému. V tejto časti sa venujeme analýze jednotlivých modulov, ktoré sa podieľajú na spracovávaní príkazov, načrtneme riešenie a zhodnotíme ho z pohľadu podpory temporálnych a netemporálnych operácií.

2.11.1 PostgreSQL backend

Na to, aby sme mohli pridať temporálnu podporu do backendu PostgreSQL, je nutné poznať celý proces spracovania príkazov a jednotlivé moduly, ktoré toto spracovanie zabezpečujú. Na nasledujúcom obrázku (**Obr. 8**) je diagram znázorňujúci tok spracovania príkazov medzi modulmi [19].



Obr. 8. Proces spracovania príkazu v backende PostgreSQL

Stručný opis procesu spracovania príkazu [19]:

1. Príkazy prichádzajú do backendu prostredníctvom paketov cez TCP/IP alebo Unix sockety.
2. Sú načítané do reťazca a predané parseru, kde lexikálny analyzátor rozdelí príkaz na tokeny (slová). Parser použije gramatiku a tokeny na identifikovanie typu príkazu a načítanie vhodnej štruktúry pre uloženie príkazu.
3. Príkaz je potom identifikovaný buď ako komplexný (SELECT, INSERT, UPDATE, DELETE) alebo jednoduchý (napríklad CREATE, ALTER, ANALYZE a iné). Jednoduché podporné príkazy sú spracované priamo funkciami určenými na ich spracovanie. Komplexné príkazy si vyžadujú zložitejšie spracovanie.
4. Ďalším krokom je modifikovanie príkazu prostredníctvom pohľadov a pravidiel, ktoré sa ho týkajú. Toto sa vykonáva v prepisovacom systéme.
5. Optimalizátor vygeneruje pre príkaz optimálny plán obsahujúci operácie, ktoré sa majú vykonať.
6. Plán je predaný modulu na vykonávanie príkazov a výsledok je vrátený naspäť klientovi.

Moduly, ktoré sa podieľajú na procese spracovania príkazov [19]:

- *Main* – Kontroluje meno procesu a rôzne príznaky a predáva kontrolu modulu Postmaster alebo Postgres.
- *Postmaster* – Má na starosti štart a zastavenie modulu postgres. Vytvára zdieľanú pamäť a v cykle čaká na žiadosti o pripojenie. Keď príde žiadosť, spustí sa postgres a predá sa mu pripojenie.
- *Libpq* – Zabezpečuje komunikáciu s klientskými procesmi
- *Traffic cop* – Hlavný modul backendu, ktorý volá ostatné moduly ako parser, optimizer, executor a utility command.
- *Parser* – Konvertuje SQL príkazy prichádzajúce z libpq na špeciálne štruktúry. SQL príkaz je lexikálne rozanalyzovaný na kľúčové slová a konštanty a predaný parseru, ktorý naplní jednotlivé elementy príkazu do štruktúr. Tieto štruktúry sú následne rozdelené na kusy, skontrolované a predané modulu utility command, alebo

skonvertované na zoznam uzlov, ktorý ďalej spracovávjú moduly optimizer a executor.

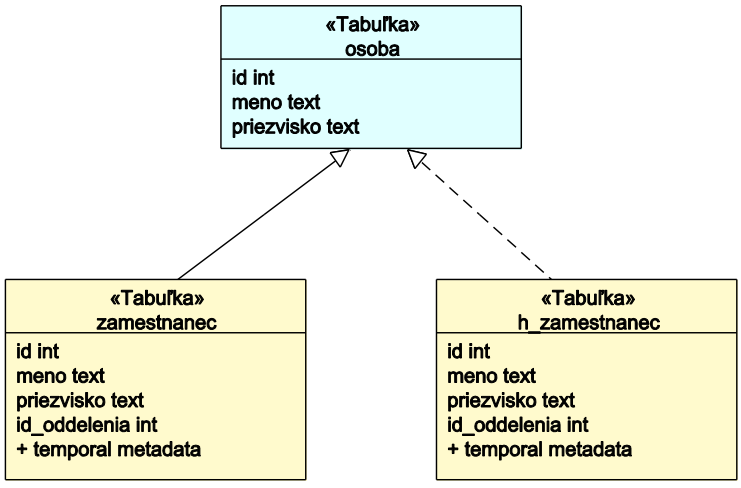
- *Rewrite* – Zabezpečuje prepis príkazov na základe pohľadov a pravidiel.
- *Optimizer* – Najprv vygeneruje všetky možné spôsoby vykonania príkazu. Pri hľadaní spôsobov vykonania a určovaní ich ceny berie do úvahy poradie operácií JOIN, podmienky vo WHERE časti a štatistiky pre tabuľku. Následne vyberie spôsob s najnižšou cenou a vytvorí pre neho plán vykonania.
- *Executor* – Vykonáva SELECT, INSERT, UPDATE a DELETE príkazy. Operácie potrebné pre vykonanie týchto príkazov zahŕňajú napríklad hľadanie v heape, hľadanie v indexoch, usporadúvanie, spájanie tabuliek, zoskupovanie, agregácie a zabezpečovanie jedinečnosti.
- *Commands* – Spracovávajú SQL príkazy, ktoré nevyžadujú komplexné zaobchádzanie. Patria sem VACUUM, COPY, ALTER TABLE, CREATE TABLE, CREATE TYPE a veľa ďalších. Tento modul prijíma štruktúry priamo od parsra a vykonáva ich prostredníctvom volania nízko úrovňových funkcií z modulu catalog
- *Catalog* – Obsahuje funkcie na manipuláciu so systémovými katalógmi. Nachádzajú sa tu napríklad funkcie na vytváranie a manipuláciu s tabuľkami, indexami, procedúrami, operátormi a typmi. Volajú sa zvyčajne vo funkciách na vyššej úrovni.
- *Storage* – Slúži na správu viacerých úložných systémov, ku ktorým zabezpečuje jednotný prístup.
- *Access* – Poskytuje rôzne spôsoby prístupu k dátam. Obsahuje funkcie na prácu s heapom, indexami a transakciami.
- *Nodes* – Poskytuje prostriedky na manipuláciu s uzlami a zoznamami. PostgreSQL uchováva informácie o SQL príkazoch v štruktúrach nazývaných uzly (nodes). Uzly sú generické kontajnery, ktoré obsahujú informáciu o type a pre typ špecifickú sekciu dát. Uzly sú väčšinou umiestňované do zoznamov. Zoznam je kontajner obsahujúci ľubovoľný element a ukazovateľ na ďalší zoznam. Štruktúry zoznam sú navzájom zreťazené a vytvárajú zreťazený zoznam. Takýmto spôsobom zreťazený zoznam môže obsahovať neobmedzený počet uzlov a každý uzol môže obsahovať ľubovoľný typ dát. Tieto štruktúry sú často používané v moduloch Parser, Optimizer a Executor.

– *Utils* – Obsahuje rôzne podporné funkcie.

2.11.2 Náčrt riešenia

Riešenie modifikáciou zdrojových kódov spočíva v rozšírení syntaxe SQL podľa štandardu SQL/Temporal (kapitola 2.6) a v zmene vykonávania príkazov vo vnútri databázového systému. Podstata je rovnaká ako v predchádzajúcom riešení. Aktuálne verzie sú uložené v jednej tabuľke a historické verzie v druhej. V tomto riešení však nie je potrebný pohľad a pravidlá, prostredníctvom ktorých sa pristupuje k aktuálnym a historickým dátam pri dotazoch a modifikujúcich operáciách. Spúšťače sa používajú aj pri tomto riešení na odkladanie pôvodných verzií riadkov pri modifikujúcich operáciách UPDATE a DELETE.

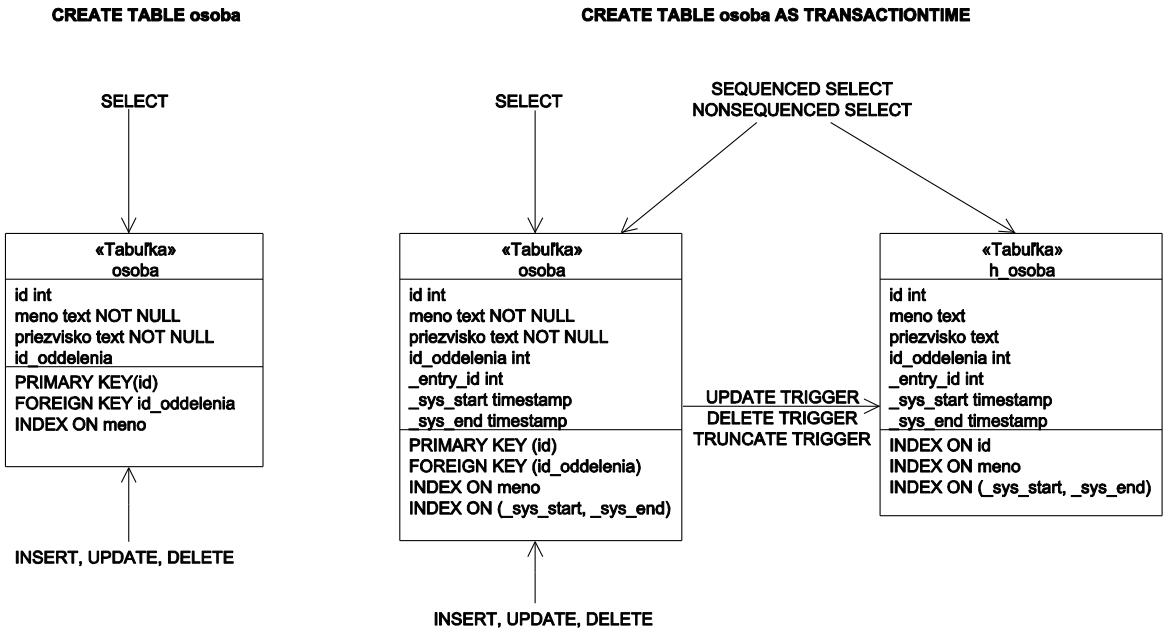
Podpora transakčného času v hierarchii dedenia je rovnaká až na to, že ak temporálna tabuľka dedí od netemporálnej, vzťah dedenia nie je len medzi rodičovskou tabuľkou a aktuálnou tabuľkou, ale aj medzi rodičovskou tabuľkou a historickou tabuľkou (**Obr. 9**). Vzťah dedenia medzi historickou a rodičovskou tabuľkou je upravený tak, že príkazy UPDATE a SELECT sa neprenášajú na historickú tabuľku, čiže dedenie platí len pre príkazy ALTER TABLE.



Obr. 9 Vzťahy dedenia medzi temporálnou a netemporálnou tabuľkou

Zamedzenie prístupu do historickej tabuľky je riešené, rovnako ako v predchádzajúcom riešení, prostredníctvom používateľských oprávnení.

Na nasledujúcom obrázku (**Obr. 10**) je znázornená štruktúra netemporálnej (vľavo) a temporálnej tabuľky (vpravo). Z obrázku je tiež vidno, s ktorými tabuľkami jednotlivé operácie pracujú.



Obr. 10. Podpora transakčného času na úrovni zdrojových kódov

Pri vytváraní temporálnej tabuľky sa súčasne vytvoria potrebné metadátové stĺpce a historická tabuľka. Tento proces je rovnaký aj pri pridávaní temporálnej podpory do existujúcej netemporálnej tabuľky.

Aktuálny príkaz SELECT sa nemení a pristupuje len k dátam v aktuálnej tabuľke. Sekvenčný a nesekvenčný SELECT vyberá podľa potreby dáta aj z historickej tabuľky. Vykonávanie príkazov INSERT, UPDATE je v prípade temporálnych tabuliek zmenené tak, že príkazy skončia s chybou ak sa pokúsia nastavovať hodnoty metadátových stĺpcov. Tieto stĺpce sú čisto v správe systému, a preto sa im v prípade príkazu INSERT hodnoty nastavia na východiskové a v prípade príkazu UPDATE sa im potrebné hodnoty nastavia v rámci spracovania príkazu (v predchádzajúcom riešení sa tieto hodnoty nastavovali v pravidle pre UPDATE).

2.11.3 Temporálna podpora a obmedzenia riešenia

Z hľadiska temporálnej podpory riešenie modifikáciou zdrojových kódov poskytuje:

- Syntax a sémantiku kompatibilnú so štandardom SQL/Temporal
- Spätnú kompatibilitu s netemporálnymi aplikáciami
- Aktuálne, sekvenčné a nesekvenčné dotazy
- Temporálne rozšírenie príkazov CREATE TABLE a ALTER TABLE

Z obmedzení, ktoré boli spomenuté pri predchádzajúcom riešení, sme pri tomto riešení identifikované len jediné:

- pridanie metadátových stĺpcov môže spôsobiť nefunkčnosť používateľských spúšťačov, ktoré pracujú s celými štruktúrami OLD a NEW (napríklad ich niekam odkladajú) a spoliehajú sa na to, že obsahujú len pôvodné stĺpce.

2.12 Zhodnotenie riešení

Načrtli sme dva spôsoby realizácie správy verzií záznamov v databázovom systéme PostgreSQL a pre každý z nich identifikovali obmedzenia. Ich vzájomné porovnanie je zachytené v **Tab. 33**.

S prihliadnutím na množstvo výhod riešenia modifikáciou zdrojových kódov sme sa rozhodli v ďalších častiach projektu venovať návrhu, implementácii a overeniu práve tohto riešenia. Pre úplnosť pripájame aj tabuľku (**Tab. 34**) porovnania funkcionality nášho riešenia modifikáciou zdrojových kódov s existujúcimi riešeniami, ktoré sme opísali v kapitole 2.7.

Hviezdička pri sekvenčnom príkaze SELECT znamená, že podporuje SELECT len ku konkrétnemu časovému momentu a nie k celému intervalu, ako to definuje štandard. Vzhľadom na to, že v tomto projekte riešime len podporu transakčného času, sme v tabuľke znázornili sivou farbou písma tie funkcie, ktoré sú definované len pre podporu času platnosti.

Tab. 33. Porovnanie riešení

Použitím prostriedkov databázy	**Modifikáciou zdrojových kódov**
Výhody	**Nevýhody**
Nie je nutný zásah do zdrojových kódov databázy	Nutný zásah do zdrojových kódov databázy
Riešenie je s malými zmenami prenositeľné aj na iné databázové systémy	Riešenie je viazané na PostgreSQL
Nevýhody	**Výhody**
Nepodporuje štandardy	Podpora syntaxe štandardu SQL/Temporal
Nutné volať uložené procedúry pri DDL operáciách	Pohodlnejšia práca pre používateľov, hlavne pri DDL operáciách (napríklad pridanie/zrušenie podpory transakčného času, pridanie/odobratie stĺpca z tabuľky s podporou transakčného času), vďaka temporálnym rozšíreniam
Pri príkaze SELECT sa vždy robí UNION z aktuálnej aj historickej tabuľky	Vykonávanie aktuálneho príkazu SELECT nad tabuľkou s podporou transakčného času je rovnaké ako nad tabuľkou bez podpory transakčného času, bez akéhokoľvek spomalenia
Všetky operácie sú vykonávané nad pohľadom	Všetky príkazy sú vykonávané priamo nad tabuľkou s aktuálnymi záznamami, vďaka čomu má riešenie vyšší výkon a minimálne obmedzenia

Tab. 34. Porovnanie s existujúcimi riešeniami

	IBM	**Oracle**	**Teradata**	**PostgreSQL temporal**
Rozšírenie syntaxe SQL	X		X	X
Spätná kompatibilita s netemporálnymi aplikáciami	X	X	X	X
Podpora transakčného času	X	X	X	X
Podpora času platnosti	X	X	X	
SELECT				
Aktuálny SELECT	X	X	X	X
Sekvenčný SELECT	X*	X*	X	X*
Nesekvenčný SELECT	X	X	X	X
INSERT				
Aktuálny INSERT	X	X	X	X
Sekvenčný INSERT	X	X	X	
Nesekvenčný INSERT	X	X	X	
UPDATE				
Aktuálny UPDATE	X	X	X	X
Sekvenčný UPDATE	X	X	X	
Nesekvenčný UPDATE	X	X	X	
DELETE				
Aktuálny DELETE	X	X	X	X
Sekvenčný DELETE	X	X	X	
Nesekvenčný DELETE	X	X	X	
Obmedzenia				
Aktuálne obmedzenia	X		X	X
Sekvenčné obmedzenia	X	X	X	
Nesekvenčné obmedzenia			X	
Ostatné				
Dátový typ PERIOD		X	X	
Sekvenčné množinové operácie				
Sekvenčné agregačné funkcie				
Sekvenčný DISTINCT				
Sekvenčný JOIN			X	

3 Špecifikácia požiadaviek

3.1 Funkcionálne požiadavky

Pre temporálne rozšírenie PostgreSQL sme definovali nasledovné funkcionálne požiadavky (v názvoch požiadaviek sme použili skratku TT pre TRANSACTIONTIME, takže TT tabuľka je tabuľka s podporou transakčného času):

1. Podpora DDL operácií
    a. Vytvorenie tabuľky s podporou transakčného času
    b. Vymazanie tabuľky s podporou transakčného času
    c. Pridanie podpory transakčného času do existujúcej tabuľky
    d. Zrušenie podpory transakčného času z tabuľky
    e. Pridanie stĺpca do tabuľky s podporou transakčného času
    f. Odobratie stĺpca z tabuľky s podporou transakčného času
    g. Zmena stĺpca v tabuľke s podporou transakčného času
    h. Vytvorenie tabuľky dediacej od tabuľky s podporou transakčného času
    i. Pridanie dedenia od tabuľky s podporou transakčného času do štandardnej tabuľky
    j. Pridanie dedenia od tabuľky s podporou transakčného času do tabuľky s podporou transakčného času
    k. Zrušenie dedenia na TT tabuľke
    l. Pridanie indexu nad stĺpec z tabuľky s podporou transakčného času
    m. Zmena ukladacích parametrov indexu TT tabuľky
    n. Zrušenie indexu nad stĺpcom z tabuľky s podporou transakčného času
    o. Zmena schémy tabuľky s podporou transakčného času
    p. Nastavenie úložných parametrov TT tabuľky
    q. Nastavenie indexu na zhlukovanie TT tabuľky
    r. Zrušenie zhlukovania TT tabuľky
    s. Zmena vlastníka TT tabuľky
    t. Zapnutie identifikátora riadkov TT tabuľky

u. Vypnutie identifikátora riadkov TT tabuľky

2. Podpora DML operácií
    a. Vloženie záznamu do tabuľky s podporou transakčného času
    b. Upravenie záznamu v tabuľke s podporou transakčného času
    c. Vymazania záznamu z tabuľky s podporou transakčného času
    d. Vyprázdnenie tabuľky s podporou transakčného času
3. Podpora dotazov
    a. Dotazy nad aktuálnymi verziami záznamov
    b. Dotazy nad aktuálnymi aj historickými verziami záznamov
    c. Dotazy nad záznamami platnými vo zvolenom čase
4. Podpora údržbových operácií
    a. Zhluknutie TT tabuľky podľa indexu
5. Podpora nastavení používateľských oprávnení
    a. Pridelenie SELECT oprávnení nad TT tabuľkou
    b. Odobratie SELECT oprávnení nad TT tabuľkou
    c. Pridelenie INSERT oprávnení nad TT tabuľkou
    d. Odobratie INSERT oprávnení nad TT tabuľkou
    e. Pridelenie UPDATE oprávnení nad TT tabuľkou
    f. Odobratie UPDATE oprávnení nad TT tabuľkou

3.2 Požiadavky na rozhranie

Keďže temporálna podpora má byť internou súčasťou PostgreSQL, pri voľbe rozhrania sú možnosti obmedzené na rozhranie poskytované týmto databázovým systémom, čiže jazyk SQL. Na to, aby bolo možné využívať podporu transakčného času, musíme množinu SQL príkazov rozšíriť o špeciálne temporálne príkazy. Požiadavkou pri takomto rozširovaní jazyka je dodržanie existujúceho štandardu SQL/Temporal, ktorý sme predstavili v kapitole 2.6.

3.3 Bezpečnostné požiadavky

Pri správe verzií záznamov je dôležité zabezpečiť, aby bežný používateľ nemohol robiť zásahy do histórie verzií záznamov. Bez takéhoto zabezpečenia by napríklad bolo možné

zmazať záznam o vykonanej operácii a potom by sa už nedalo zistiť, kedy a v akej transakcii k operácii došlo. Podobné zásahy by mali za následok skreslenie histórie vykonávaných operácií nad jednotlivými záznamami a tým by sa narušila jedna z hlavných výhod podpory transakčného času. Rovnako musíme zamedziť vykonávaniu akýchkoľvek zmien nad vnútorne vytvorenými objektmi slúžiacimi na zabezpečenie podpory transakčného času (stĺpce, indexy, sekvencie a podobne).

3.4 Výkonnostné požiadavky

Pre temporálne rozšírenie PostgreSQL sme definovali nasledovné výkonnostné požiadavky:

1. Nasadenie zmien databázového servera nemá mať pri vypnutej temporálnej podpore za následok žiadny pokles výkonu.
2. Aktuálne dotazy nad tabuľkou s podporou transakčného času musia mať rovnaký výkon ako dotazy nad tou istou tabuľkou bez podpory transakčného času

3.5 Požiadavky na spätnú kompatibilitu

Temporálne rozšírenie PostgreSQL nemá spôsobiť zmenu správania pôvodnej funkcionality systému, aby sa zachoval správny beh existujúcich aplikácií aj po rozšírení databázy o temporálnu podporu. Okrem toho zapnutie podpory transakčného času nad tabuľkami, nemá pre aplikáciu znamenať žiadne obmedzenia okrem nasledovného definovaného obmedzenia riešenia:

- Pridanie metadátových stĺpcov podpory transakčného času, môže spôsobiť zmeny v správaní (prípadne nefunkčnosť) tých častí aplikácie, ktoré pracujú s celými tabuľkovými riadkovými typmi, pretože zmeny v štruktúre tabuľky sa automaticky odrážajú aj na tabuľkových riadkových typoch. Príkladom častí aplikácie, ktorým toto obmedzenie môže spôsobiť problémy sú uložené procedúry vracajúce tabuľkový riadkový typ, alebo spúšťače odkladajúce nové resp. staré riadky do tabuľky, ktorá nie je prispôsobená novej štruktúre. Takéto situácie sa však v aplikáciách vyskytujú len zriedka, a tak ide o veľmi malé obmedzenie.

3.6 Implementačné prostredie

Ako implementačný jazyk sme zvolili jazyk C. Keďže celý systém PostgreSQL je implementovaný v tomto jazyku, je najprirodzenejšie vytvárať novú funkcionalitu v rovnakom jazyku, ak je v ňom možné splniť všetky definované požiadavky.

4 Návrh

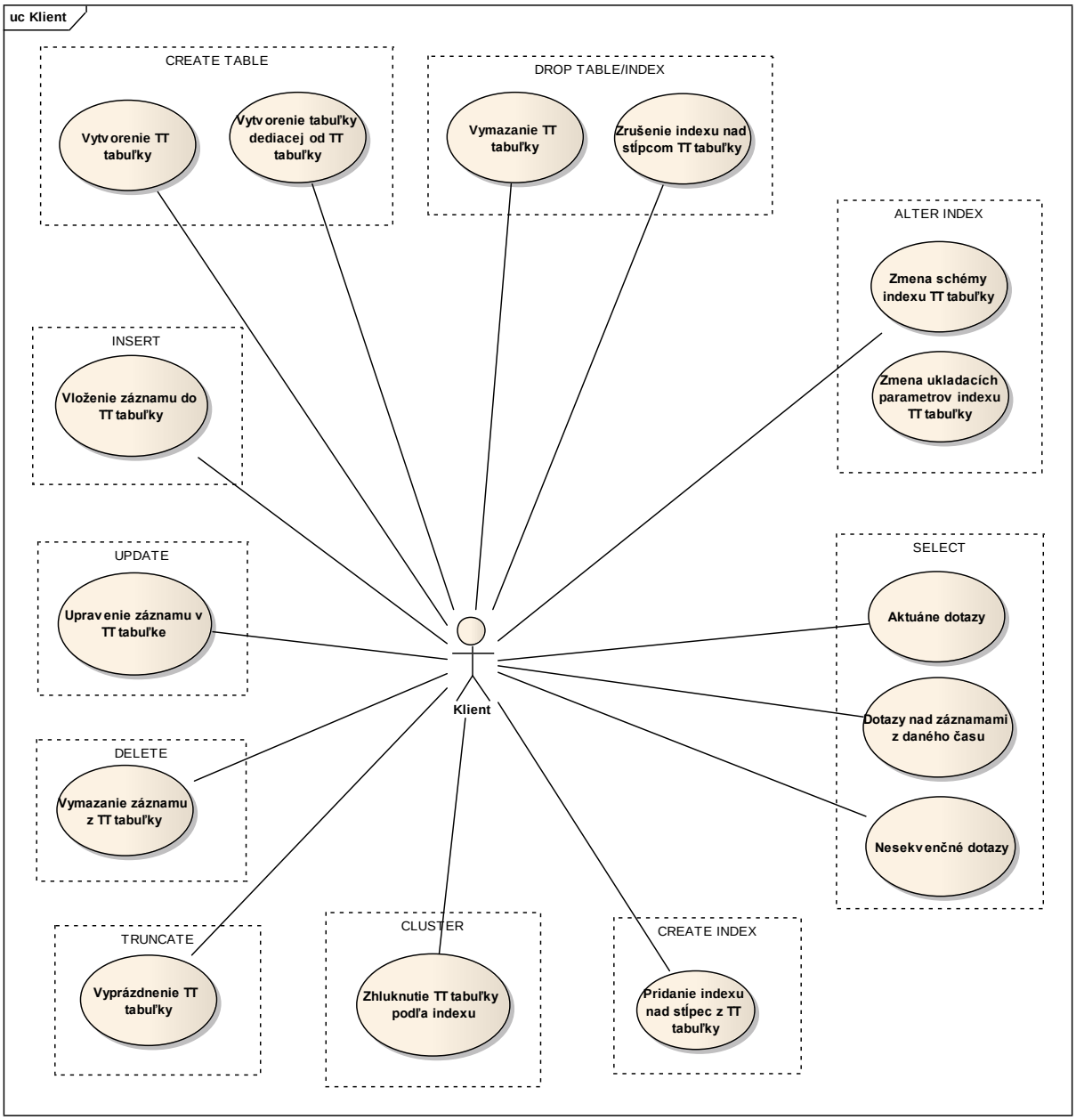
Táto kapitola sa zaoberá návrhom riešenia podpory transakčného času v databázovom systéme PostgreSQL, ktorý slúži ako základ pre implementáciu.

4.1 Návrh služieb podpory transakčného času

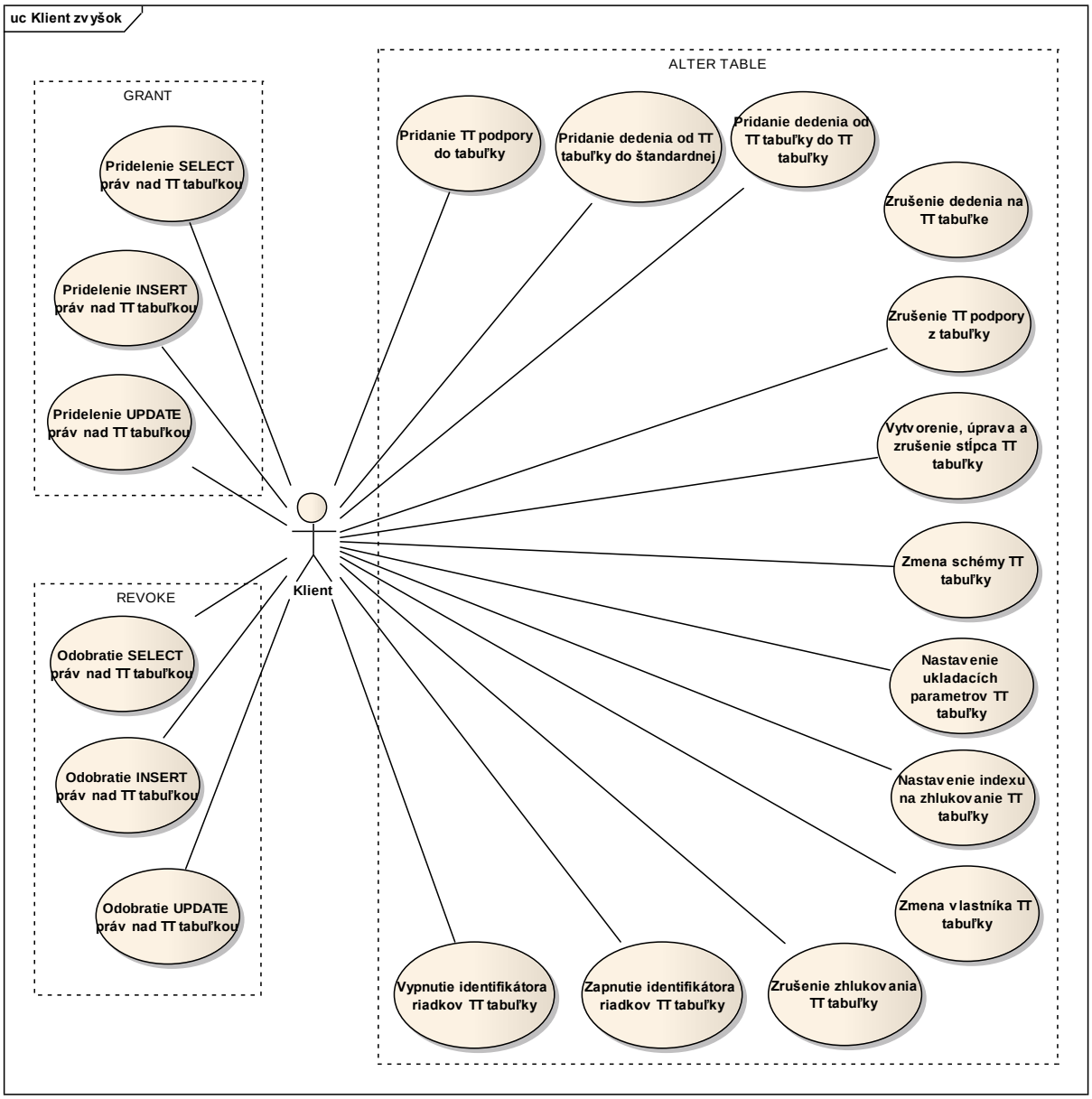
V rámci podpory transakčného času sme identifikovali, len jednu používateľskú rolu. Ide o klienta databázového servera. Klientom môžu byť rôzne aplikácie postavené nad PostgreSQL alebo používatelia pracujúci s databázou cez rôzne administračné nástroje.

Na **Obr. 11** a **Obr. 12** sú zobrazené diagramy prípadov použitia so službami, ktoré poskytuje riešenie klientovi. Služby vychádzajú z funkcionálnych požiadaviek identifikovaných v kapitole 3.1 a sú rozdelené podľa príkazov prostredníctvom ktorých sa k nim pristupuje. Pri opise služieb sme použili skratku TT pre TRANSACTIONTIME, takže TT tabuľka je tabuľka s podporou transakčného času. Ak hovoríme o príkazoch ako o štandardných, znamená to, že tieto príkazy si nevyžiadali zmenu syntaxe a ostávajú rovnaké ako je uvedené v dokumentácii PostgreSQL [17]. Zmenené je len ich správanie, ak sú zavolané nad TT tabuľkou.

V nasledujúcich podkapitolách sa venujeme podrobnému opisu jednotlivých služieb. Opisy sú zoskupené rovnako ako v diagrame - podľa príkazov. Služby sú bližšie špecifikované slovne, alebo pomocou diagramov aktivít. Vynechali sme opis služieb, ktorých podstatu tvorí len vykonanie akcie aj nad historickou tabuľkou a nevyžadujú si bližšie vysvetlenie.



Obr. 11. Diagram prípadov použitia pre rolu klient



Obr. 12. Zvyšná časť diagramu prípadov použitia pre rolu klient

4.1.1 CREATE TABLE

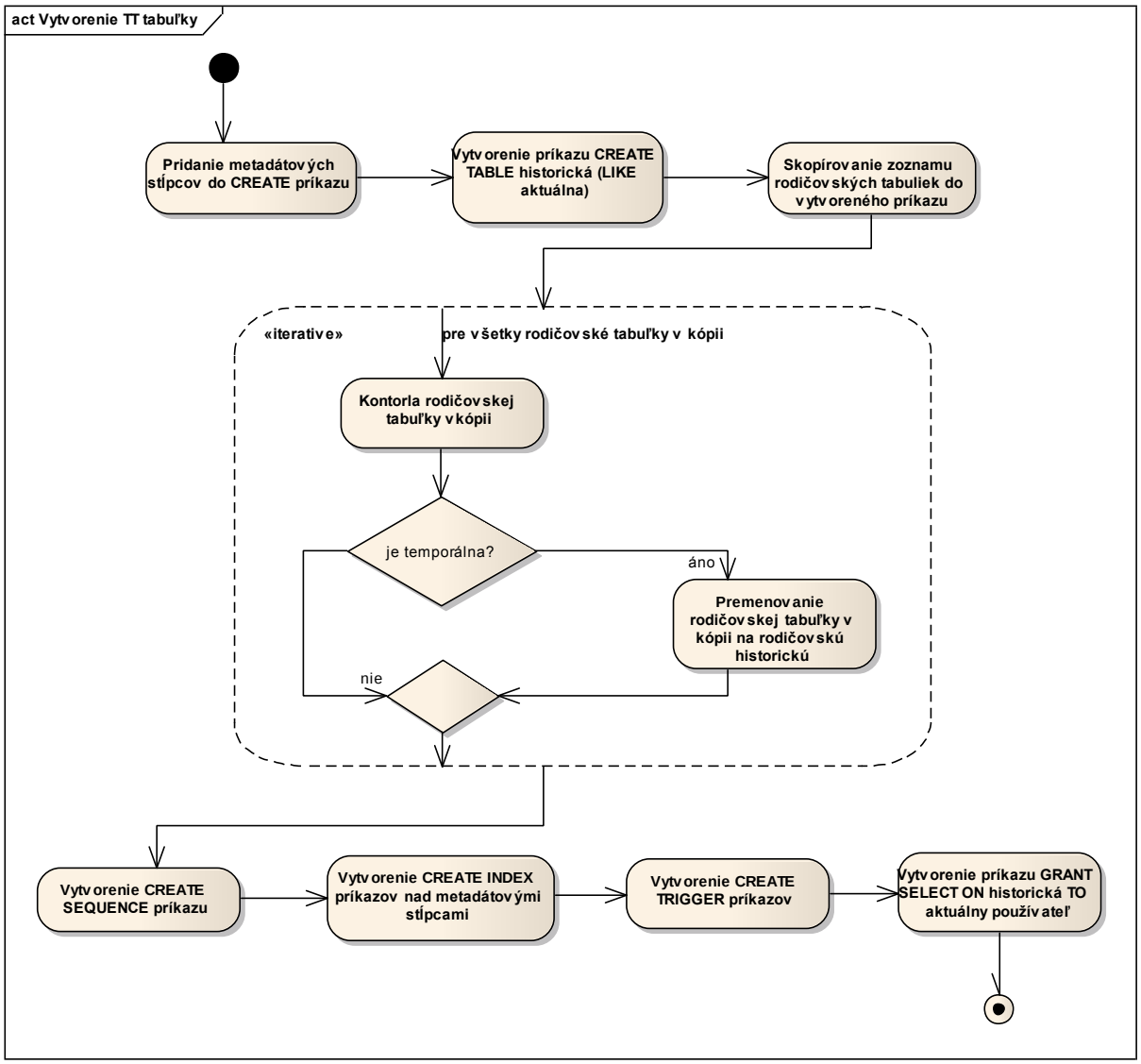
Príkaz CREATE TABLE si vyžaduje rozšírenie, aby bolo možné volať niektoré zo služieb popísaných v nasledujúcich podkapitolách. Po rozšírení ja jeho syntax nasledovná [17]:

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name ( [
  { column_name data_type [ DEFAULT default_expr ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE parent_table [ like_option ... ] }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
[ AS TRANSACTIONTIME ]
```

4.1.1.1 Vytvorenie TT tabuľky

Postup vytvorenia TT tabuľky je zapísaný v diagrame aktivít na obrázku **Obr. 13**. Pri vytváraní TT tabuľky sa do príkazu CREATE TABLE pridajú metadátové stĺpce. Vytvorí sa CREATE TABLE LIKE príkaz na vytvorenie historickej tabuľky s rovnakou štruktúrou a indexmi. Meno historickej tabuľky musí byť jedinečné, aby nedošlo ku konfliktu názvov s existujúcimi tabuľkami. Zoznam tabuliek, od ktorých aktuálna tabuľka dedí sa skopíruje aj do príkazu na vytvorenie historickej tabuľky. Ak je tabuľka zo zoznamu temporálna nahradí sa k nej prislúchajúcou historickou tabuľkou (pozri kapitolu 2.11.2). Nakoniec sa ešte vytvoria ostatné príkazy na vytvorenie objektov potrebných pre zabezpečenie správy verzií (sekvencie, indexy, spúšťače).

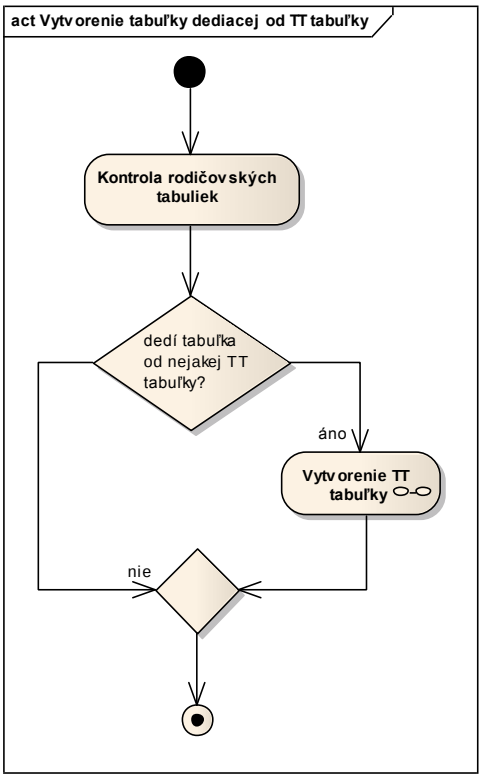
Kvôli zamedzeniu vykonávania modifikujúcich operácií nad historickou tabuľkou, tak ako sme napísali v kapitole 2.11.2, je potrebné nastaviť vlastníka historickej tabuľky na superpoužívateľa a aktuálnemu používateľovi prideliť oprávnenia na vykonávanie príkazu SELECT. Na vytvorenie TT tabuľky slúži príkaz CREATE TABLE s parametrom AS TRANSACTIONTIME.



Obr. 13 Diagram aktivít pre Vytvorenie TT tabuľky

4.1.1.2 Vytvorenie tabuľky dediacej od TT tabuľky

Tabuľka, ktorá dedí aspoň od jednej TT tabuľky musí byť tiež TT tabuľka (pozri kapitolu 2.11.2), preto je táto služba veľmi podobná predchádzajúcej. Líši sa len v tom, že na začiatku je potrebné v príkaze CREATE TABLE skontrolovať, že či vytváraná tabuľka dedí od nejakej TT tabuľky. Ak áno, postup je rovnaký ako v predchádzajúcej službe a je znázornený na **Obr. 14**. Na vytvorenie tabuľky dediacej od TT tabuľky slúži štandardný príkaz CREATE TABLE s parametrom INHERITS.



Obr. 14 Diagram aktivít pre Vytvorenie tabuľky dediacej od TT tabuľky

4.1.2 CREATE INDEX

4.1.2.1 Pridanie indexu nad stĺpce z TT tabuľky

Pri pridávaní indexu nad stĺpec z TT tabuľky je potrebné vytvoriť príkaz na vytvorenie rovnakého (až na prípadný UNIQUE parameter, ktorý v historickej tabuľke byť nemôže kvôli duplicitám vo viacerých verziách záznamu) indexu nad tými istými stĺpcami aj na historickej tabuľke. Rovnako ako pri vytváraní historickej tabuľky, aj pri indexe je potrebné vytvoriť jedinečné meno, aby nedošlo ku konfliktu názvov s existujúcimi indexmi. Na pridanie indexu nad stĺpce z TT tabuľky slúži štandardný príkaz CREATE INDEX.

4.1.3 ALTER TABLE

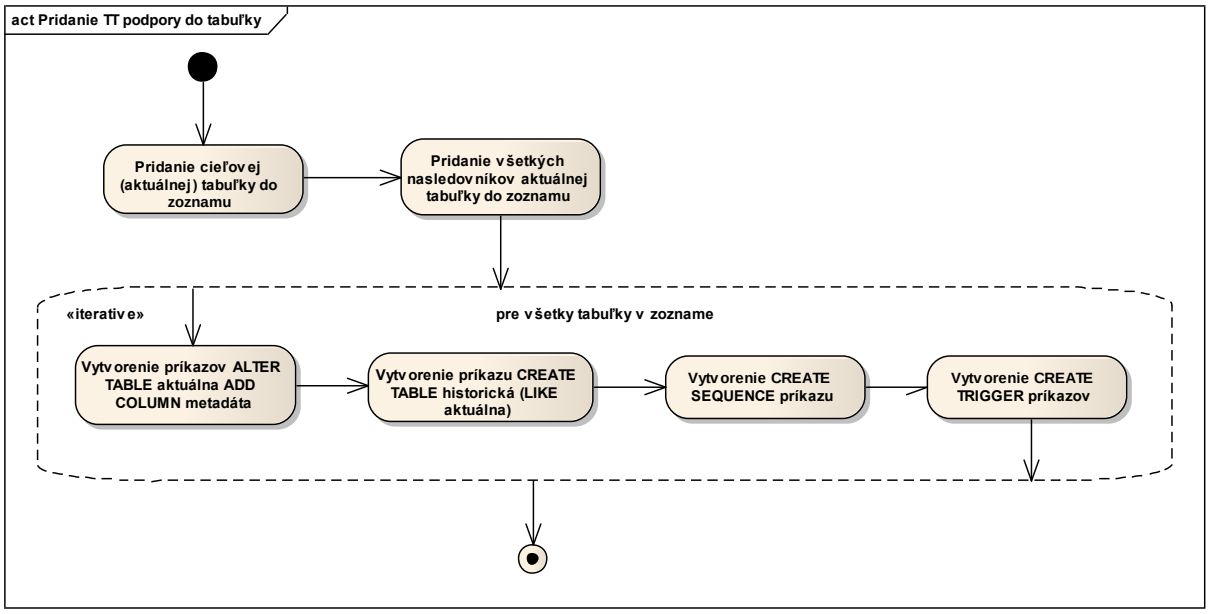
Príkaz ALTER TABLE si vyžaduje rozšírenie, aby bolo možné volať niektoré zo služieb popísaných v nasledujúcich podkapitolách. Po rozšírení je jeho syntax nasledovná [17]:

```
ALTER TABLE [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column TO new_column
ALTER TABLE name
    RENAME TO new_name
ALTER TABLE name
    SET SCHEMA new_schema

where action is one of:

    ...
    ADD TRANSACTIONTIME
    DROP TRANSACTIONTIME
```

Pre všetky varianty ALTER TABLE príkazu je potrebné zamedziť ich vykonávaniu nad historickou tabuľkou a metadátovými stĺpcami v aktuálnej tabuľke. Rovnako platí, že na historickú tabuľku sa nemôžu prenášať žiadne obmedzenia (DEFAULT, NOT NULL, CHECK, UNIQUE, PRIMARY KEY a EXCLUDE).

4.1.3.1 Pridanie TT podpory do tabuľky

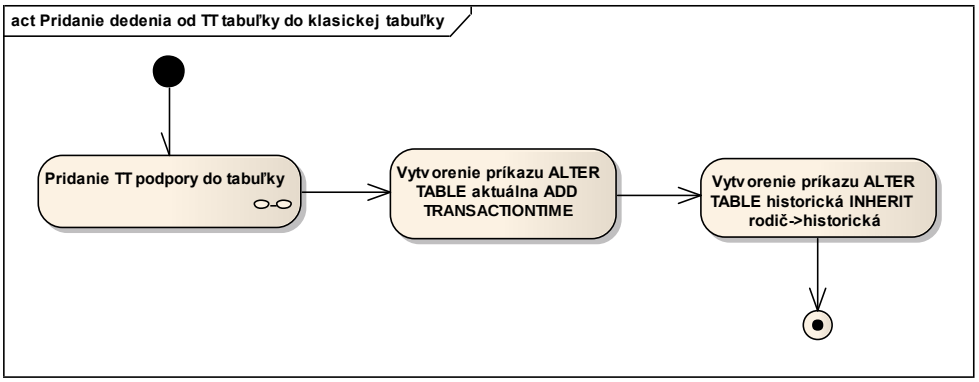
Postup pridania TT podpory do existujúcej tabuľky je znázornený na **Obr. 15**. Vzhľadom na to, že tabuľka, do ktorej sa pridáva TT podpora, môže byť súčasťou hierarchie dedenia, je potrebné na začiatku načítať všetky od nej dediace tabuľky (len tie bez TT podpory), do ktorých musí byť TT podpora pridaná tiež (pozri kapitolu 2.11.2). Následne sa pre všetky tabuľky vytvoria príkazy na vytvorenie objektov potrebných pre správu verzií (metadátové stĺpce, historická tabuľka, sekvencie, indexy a spúšťače). V rámci príkazu CREATE TABLE LIKE sa pre historickú tabuľku vytvoria aj všetky potrebné indexy (bez UNIQUE obmedzení). Na pridanie TT podpory do tabuľky slúži príkaz ALTER TABLE s akciou ADD TRANSACTIONTIME.



Obr. 15 Diagram aktivít pre Pridanie TT podpory do tabuľky

4.1.3.2 Pridanie dedenia od TT tabuľky do štandardnej

Základom pridania dedenia od TT tabuľky do štandardnej je postup pridania TT podpory do tabuľky. Okrem toho je ešte potrebné vytvoriť príkaz na označenie tabuľky ako TT tabuľky a príkaz na pridanie dedenia medzi detskou historickou tabuľkou a rodičovskou historickou tabuľkou. Tento postup je znázornený na **Obr. 16**. Na pridanie dedenia od TT tabuľky slúži obyčajný príkaz ALTER TABLE s akciou INHERIT.



Obr. 16 Diagram aktivít pre Pridanie dedenia od TT tabuľky do štandardnej

4.1.3.3 Pridanie dedenia od TT tabuľky do TT tabuľky

Pri pridaní dedenia medzi dve TT tabuľky je potrebné zabezpečiť len pridanie dedenia aj medzi historické tabuľky vytvorením príkazu ALTER TABLE historická INHERIT

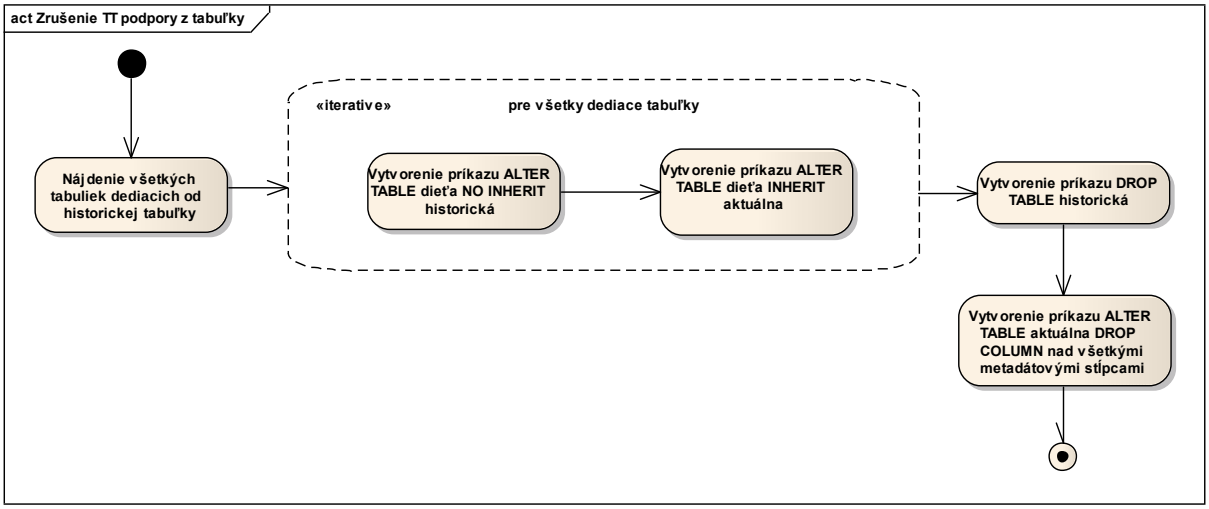
rodič.historická. Príkazom na vykonanie tejto služby je štandardný príkaz ALTER TABLE s akciou INHERIT.

4.1.3.4 Zrušenie dedenia na TT tabuľke

Pri zrušení dedenia na TT tabuľke je potrebné zabezpečiť len zrušenie dedenia aj na historickej tabuľke vytvorením príkazu ALTER TABLE historická NO INHERIT rodičovská. Príkazom na vykonanie tejto služby je štandardný príkaz ALTER TABLE s akciou NO INHERIT.

4.1.3.5 Zrušenie TT podpory z tabuľky

Postup zrušenia TT podpory z tabuľky je znázornený na **Obr. 17**. V prvom rade je potrebné zrušiť všetky objekty zabezpečujúce správu verzií. Indexy, sekvencie a spúšťače budú zrušené automaticky v rámci príkazu DROP TABLE, metadátové stĺpce treba zrušiť explicitne. Následne je potrebné všetkým tabuľkám, ktoré dedia od cieľovej tabuľky (všetky sú temporálne) presmerovať dedenie z historickej na aktuálnu. Na pridanie TT podpory do tabuľky slúži príkaz ALTER TABLE s akciou DROP TRANSACTIONTIME.



Obr. 17 Zrušenie TT podpory z tabuľky

4.1.3.6 Vytvorenie, úprava a zrušenie stĺpca TT tabuľky

Táto služba kombinuje pridanie stĺpca, zrušenie stĺpca a premenovanie stĺpca v TT tabuľke. Všetky tieto operácie je potrebné vykonať aj nad historickou tabuľkou. Pre splnenie bezpečnostných požiadaviek je potrebné zamedziť zrušeniu alebo modifikácii metadátových

stĺpcov v TT tabuľkách. Na vykonanie jednotlivých operácií slúži štandardný príkaz ALTER TABLE s akciami ADD COLUMN, DROP COLUMN a RENAME.

4.1.3.7 Nastavenie úložných parametrov TT tabuľky

Úložné parametre v PostgreSQL slúžia na napríklad na [17]:

- nastavenie miery zaplnenia stránky tabuľky príkazmi INSERT, pri ktorom sa začne ukladať na novú stránku
- zapnutie/vypnutie automatického preusporiadavania a obnovovania štatistík tabuľky (VACUUM, ANALYZE)
- minimálne množstvo operácií UPDATE a DELETE kým sa spustí preusporiadavanie a obnova štatistík

Pri zmene úložných parametrov TT tabuľky je potrebné vykonať rovnaké zmeny aj na historickej tabuľke. Príkazom na vykonanie tejto služby je štandardný príkaz ALTER TABLE s akciou SET a RESET.

4.1.3.8 Nastavenie indexu na zhlukovanie TT tabuľky

Pri zhlukovaní tabuľky sa záznamy na disku fyzicky preusporiadajú na základe informácií z indexu. Keď sa pristupuje k jednotlivým riadkom tabuľky náhodne, nezáleží na poradí v akom sú uložené. Avšak, ak pristupujeme k nejakým dátam častejšie a máme vytvorený index, ktorý ich zoskupuje, je výhodné vykonať zhlukovanie tabuľky. Využitie nachádza v prípade ak požadujeme rozsah indexovaných hodnôt z tabuľky, alebo jednu indexovanú hodnotu, ktorá sa nachádza vo viacerých riadkoch. Zhlukovanie zabezpečí, že ak sa nájde stránka pre prvý výskyt, je pravdepodobné, že aj ostatné výskyty sa budú nachádzať na tej istej stránke. Ušetrí sa na prístupoch na disk a dotazy sa zrýchlia [17].

Zhlukovanie sa vykonáva pomocou príkazu CLUSTER, ktorému možno určiť meno tabuľky a index, podľa ktorého sa má zhlukovať. Zadaný index sa zapamätá a pri ďalších volaniach ho nie je nutné uvádzať. Príkaz ALTER TABLE s akciou CLUSTER ON slúži na určenie indexu, podľa ktorého sa majú vykonávať zhlukovania nad tabuľkou. V prípade TT tabuľky je nutné príkaz vykonať aj pre príslušný index historickej tabuľky.

4.1.3.9 Zapnutie identifikátora riadkov TT tabuľky

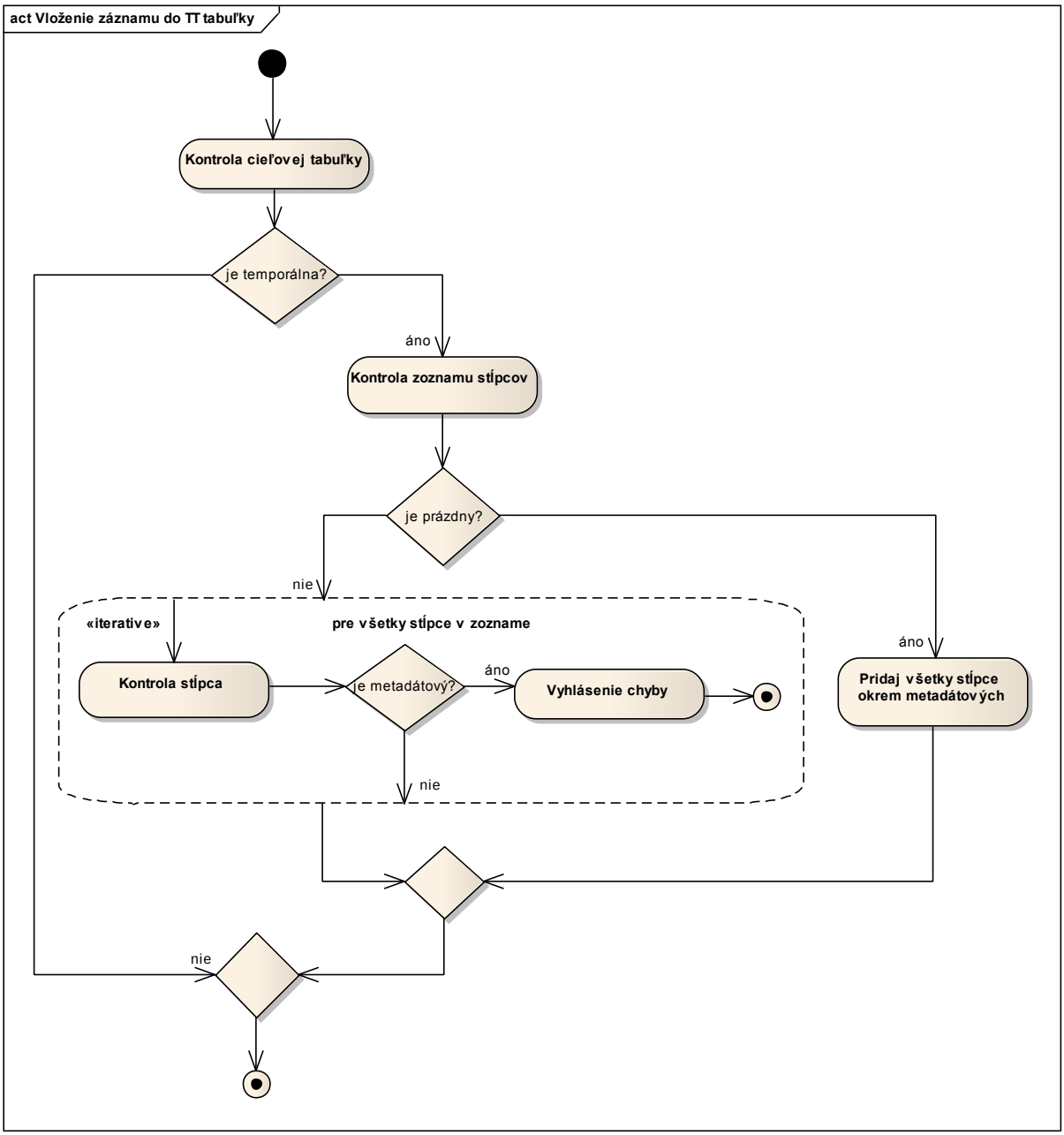
Identifikátory riadkov sa v PostgreSQL používajú hlavne ako primárne kľúče systémových tabuliek, ale dajú sa pridať aj do používateľských tabuliek. Neodporúčajú sa však používať ako primárne kľúče, majú slúžiť ako referencie do systémových katalógov.

Na pridanie identifikátora je určený príkaz ALTER TABLE s akciou SET WITH OIDS. Vzhľadom na to, že historická tabuľka musí mať rovnakú štruktúru ako aktuálna, je potrebné vykonať pridanie identifikátora riadkov aj na historickej tabuľke.

4.1.4 INSERT

4.1.4.1 Vloženie záznamu do TT tabuľky

Pri vkladaní záznamov do TT tabuľky musíme zabezpečiť, aby nebolo možné vložiť hodnoty do metadátových stĺpcov a navyše ak je zoznam stĺpcov prázdny treba očakávať len hodnoty pre používateľom definované stĺpce, aby bola zabezpečená kompatibilita s exitujúcimi aplikáciami. Postup je znázornený na **Obr. 18**. Na vloženie záznamu do TT tabuľky sa používa štandardný príkaz INSERT



Obr. 18 Diagram aktivít pre Vloženie záznamu do TT tabuľky

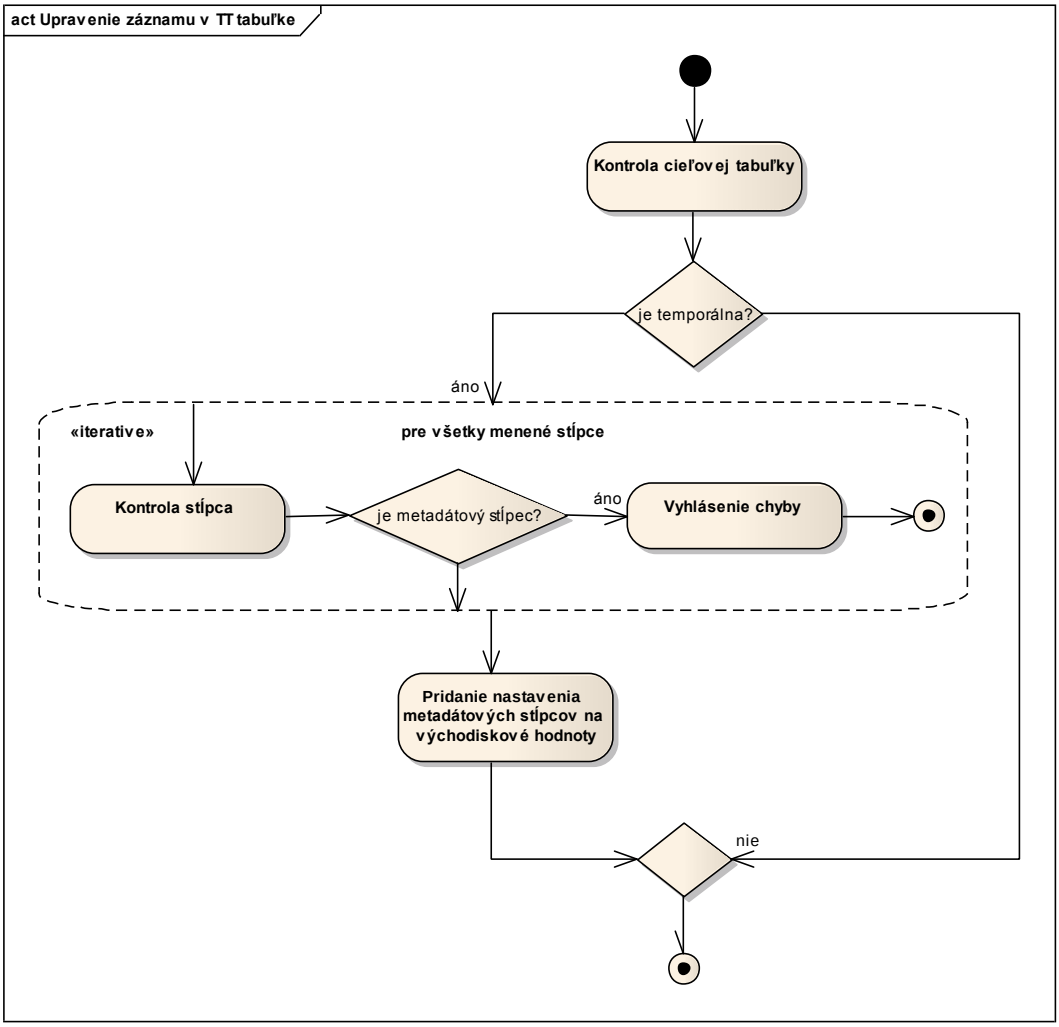
4.1.5 UPDATE

4.1.5.1 Upravenie záznamu v TT tabuľke

Pri úprave záznamov v TT tabuľky sa vykoná algoritmus znázornený na **Obr. 19**. Algoritmus slúži na zamedzenie modifikovania hodnôt v metadátových stĺpcoch TT tabuliek. Ak zoznam stĺpcov v príkaze UPDATE obsahuje metadátový stĺpec, systém vyhlási

chybu, inak sa do zoznamu stĺpcov doplní nastavenie metadátových stĺpcov na potrebné hodnoty (pozri kapitolu 2.10.4.4).

Okrem toho je nutné zamedziť, aby sa príkaz UPDATE (ako aj SELECT) zavolaný nad štandardnou tabuľkou vykonával aj nad historickými tabuľkami, ktoré od štandardnej tabuľky dedia (pozri kapitolu 2.11.2). Najpodstatnejšou časťou pri upravovaní záznamov v TT tabuľke je odkladanie starej verzie do historickej tabuľky, ktoré zabezpečuje spúšťač. Spúšťačom je venovaná kapitola 4.1.10. Na upravenie záznamu v TT tabuľke sa používa štandardný príkaz UPDATE.



Obr. 19 Diagram aktivít pre Upravenie záznamu v TT tabuľke

4.1.6 DELETE

4.1.6.1 Vymazanie záznamu z TT tabuľky

Pri vymazaní záznamu z TT tabuľky je potrebné len odloženie záznamu do historickej tabuľky, ktoré zabezpečuje spúšťač. Spúšťačom je venovaná kapitola 4.1.10. Na vymazanie záznamu z TT tabuľky sa používa štandardný príkaz DELETE.

4.1.7 TRUNCATE

4.1.7.1 Vyprázdnenie TT tabuľky

Pri vyprázdnení TT tabuľky je potrebné len odloženie všetkých záznamov do historickej tabuľky, ktoré zabezpečuje spúšťač. Spúšťačom je venovaná kapitola 4.1.10. Na vyprázdnenie TT tabuľky sa používa štandardný príkaz TRUNCATE.

4.1.8 SELECT

Pre prácu s časovými údajmi podporuje riešenie 3 typy dotazov:

- Aktuálne
- Nesekvenčné
- Dotazy nad záznamami z daného času

To, aký typ dotazu sa vykoná je možné ovplyvniť na nasledovných úrovniach (zoradené podľa stúpajúcej priority):

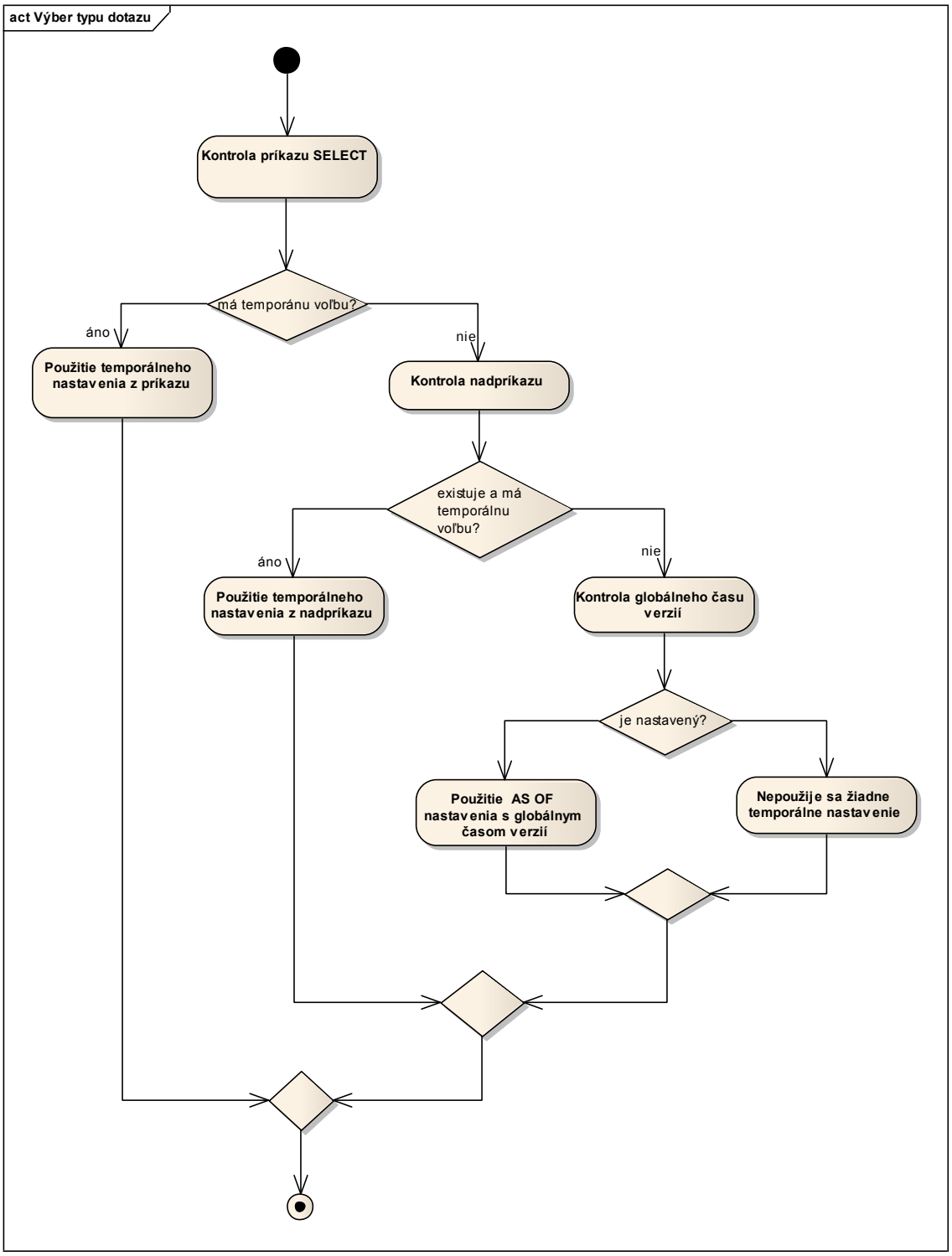
- Nastavením špeciálnej premennej history_timestamp na úrovni pripojenia
- Parametrom na úrovni príkazu SELECT

Ak nie je ani na jednej úrovni zadaná žiadna voľba, vykoná sa štandardný príkaz SELECT. Ak je zadaná voľba na úrovni pripojenia, všetky príkazy SELECT, ktoré nemajú zadanú žiadnu voľbu, preberajú voľbu zadanú na úrovni pripojenia. Ak je zadaná voľba na úrovni príkazu SELECT, všetky tabuľky vo FROM časti ju preberajú.

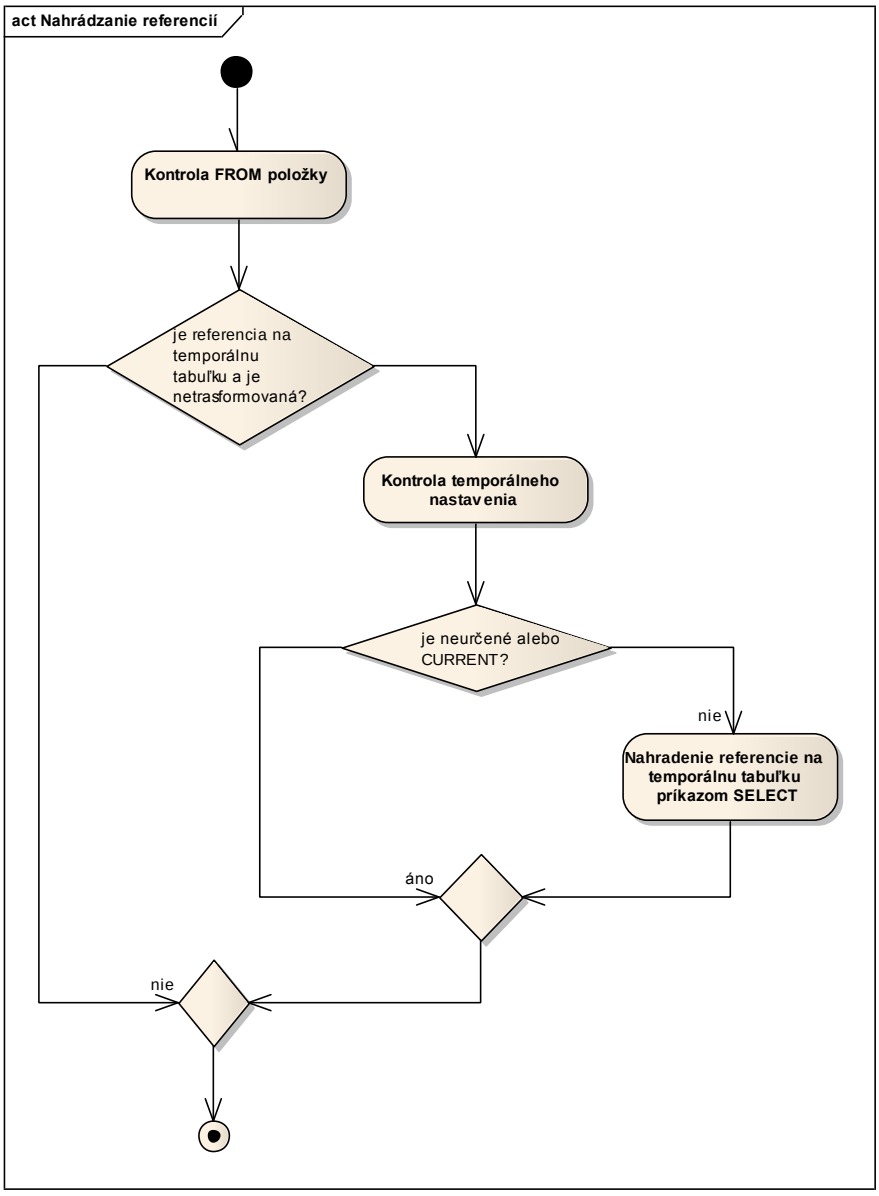
Ak výsledný typ dotazu nad TT tabuľkou vo FROM časti je iný ako aktuálny, nahradí sa referencia na tabuľku podpríkazom SELECT, ktorý pristupuje aj k záznamom v historickej tabuľke. Výber typu dotazu pre jednotlivé tabuľky a nahrádzanie referencií sú zobrazené na **Obr. 20** a **Obr. 21**. Aby sme zabezpečili kompatibilitu príkazu SELECT s existujúcimi

aplikáciami, musíme pri zadaní '*' do zoznamu výsledkov, vrátiť len používateľom definované stĺpce. Syntax príkazu SELECT rozšíreného pre potreby TT podpory je nasledovná [17]:

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
[ CURRENT TRANSACTIONTIME | NONSEQUENCED TRANSACTIONTIME | TRANSACTIONTIME AS OF
     expression ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ [ AS ] output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST
     } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [...] ]

...
```



Obr. 20 Diagram aktivít pre výber typu dotazu



Obr. 21 Diagram aktivít pre nahrádzanie referencií

4.1.8.1 Aktuálne dotazy

Aktuálne dotazy operujú len nad tabuľkou s aktuálnymi záznamami, a tak pri tomto type dotazu nie je potrebný žiadny prepis príkazu SELECT. Na to, aby sa vykonal aktuálny dotaz, musí byť:

- nezadaná žiadna možnosť na všetkých troch úrovniach
- alebo zadaný parameter CURRENT TRANSACTIONTIME na úrovni príkazu SELECT

4.1.8.2 Nesekvenčné dotazy

Nesekvenčné dotazy operujú nad zjednotením záznamov z aktuálnej a historickej tabuľky. Pri tomto type dotazov je nutný prepis zvolených TT tabuliek vo FROM časti na podpríkazy, ktoré vyberajú dáta z oboch tabuliek (pozri kapitolu 2.11.2). Na to aby sa vykonal nesekvenčný dotaz musí byť zadaný parameter NONSEQUENCED TRANSACTIONTIME na úrovni príkazu SELECT

4.1.8.3 Dotazy nad záznamami z daného času

Dotazy nad záznamami z daného času pristupujú do aktuálnej aj historickej tabuľky. Preto je nutný prepis zvolených TT tabuliek vo FROM časti na podpríkazy, ktoré vyberajú dáta z oboch tabuliek (pozri kapitolu 2.11.2). Tento druh dotazov síce nie je súčasťou štandardu SQL/Temporal, ale keďže umožňuje vykonávanie rezov v čase, je pre prácu s temporálnymi tabuľkami veľmi užitočný. Na to, aby sa vykonal dotaz nad záznamami platnými v danom čase musí byť:

- nastavený čas v premennej history_timestamp na úrovni pripojenia
- alebo zadaný parameter TRANSACTIONTIME AS OF na úrovni príkazu SELECT

4.1.9 GRANT/REVOKE

Jedno z bezpečnostných opatrení, ktoré sme do riešenia zaviedli je, že vlastník historických tabuliek sa pri vytváraní nastaví automaticky na superpoužívateľa a používateľovi, ktorý vytvára TT tabuľku (aktuálny používateľ), sa pridelia oprávnenia len na čítanie z historickej tabuľky.

Okrem toho treba pri príkazoch GRANT a REVOKE zobrať do úvahy aj existenciu sekvencie pre generovanie ID záznamov (metadátový stĺpec _entry_id), ktorú vlastní používateľ vytvárajúci TT tabuľku (aktuálny používateľ), a tak ju môže používať len on a super používateľ. Pri nasledovných typoch oprávnení je potrebné vykonať úpravy kvôli podpore transakčného času:

- INSERT – pri pridelení/odobratí oprávnenia INSERT nad TT tabuľkou, je potrebné prideliť/odobrať aj oprávnenie na používanie (USAGE) sekvencie pre stĺpec _entry_id, aby bolo možné vytvárať ID záznamov.

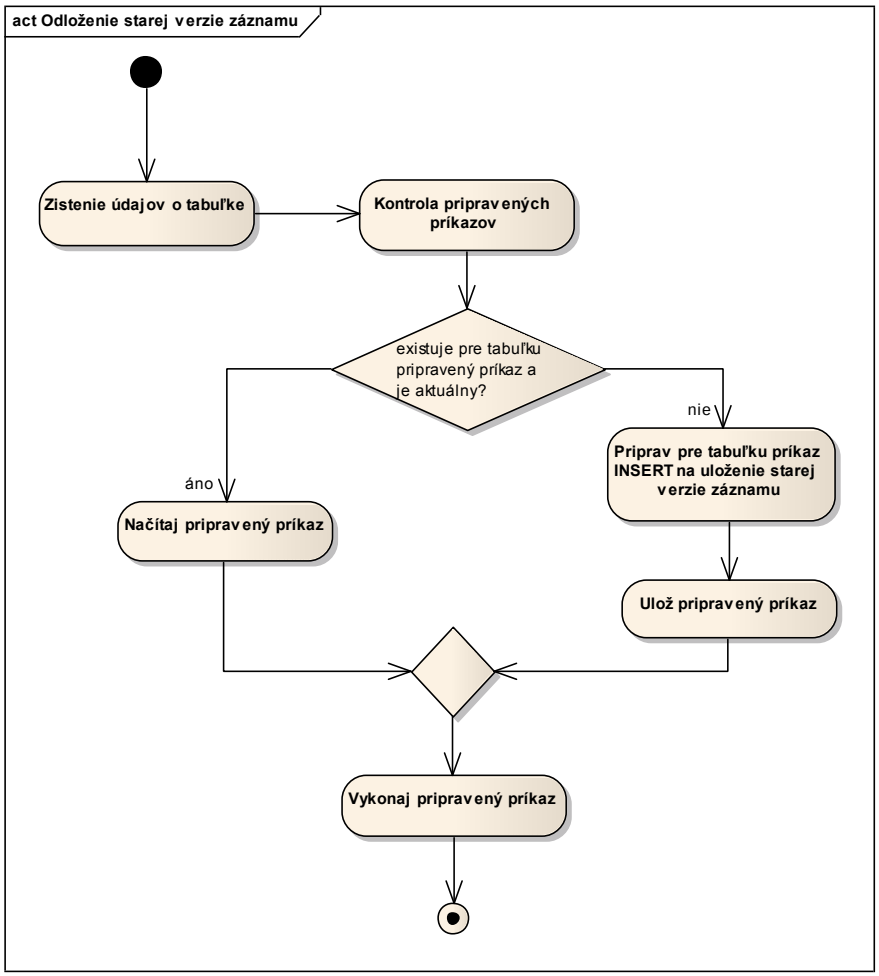
- UPDATE – pri pridelení/odobratí oprávnenia UPDATE nad stĺpcom TT tabuľky, je potrebné prideliť/odobrať oprávnenie UPDATE aj na metadátové stĺpce _sys_start a _sys_end. Inak by totiž nebolo možné nastavovať začiatok a koniec platnosti modifikovaného záznamu.
- SELECT – pri pridelení/odobratí oprávnenia SELECT nad TT tabuľkou, je potrebné prideliť/odobrať oprávnenie SELECT aj nad historickou tabuľkou, aby bolo možné pristupovať k historickým verziám záznamov.

4.1.10 Spúšťače na odkladanie starých verzií záznamov

4.1.10.1 Odloženie starej verzie záznamu

Funkcie, používané v spúšťačoch sa vytvárajú pri vytvorení databázy. Ich vlastníkom je superpoužívateľ a sú definované s parametrom SECURITY DEFINER, ktorý zabezpečuje, že sú vykonávané pod používateľom, ktorý ich vytvoril. Keďže superpoužívateľ je aj vlastníkom všetkých historických tabuliek, tieto funkcie môžu nad tabuľkami vykonávať operácie potrebné pri odkladaní starých verzií záznamov. Algoritmus funkcií je zachytený na **Obr. 22**.

Kvôli výkonnosti spúšťačov na odkladanie starých verzií záznamov je vhodné použiť pripravené príkazy. Výhodou je, že takéto príkazy sa raz pripravia, čím vytvoria sa ich plány vykonávania, ktoré sa uložia a možno ich používať počas celého trvania pripojenia. Následne stačí takto pripravené príkazy spustiť s potrebnými parametrami a uložené plány sa priamo vykonajú. Preklad a plánovanie sa tak robí len jeden krát. Pri každom volaní spúšťača je potrebné skontrolovať, či existuje pripravený príkaz a či je aktuálny (neaktuálnym sa môže stať napríklad, keď sa zruší stĺpec tabuľky). Ak existuje a je aktuálny priamo sa vykoná, inak sa najprv musí vytvoriť a uložiť.



Obr. 22 Diagram aktivít pre odloženie starej verzie záznamu

5 Implementácia

Táto kapitola sa zaoberá opisom implementačných rozhodnutí a postupov. Podrobné opisy implementácie vybraných služieb opísaných v kapitole 4, sa nachádzajú v technickej dokumentácii (Príloha A).

5.1 Modifikované moduly PostgreSQL

Jedným z najväčších problémov pri implementácii bolo zorientovanie sa v zdrojových kódoch PostgreSQL. Užitočnými zdrojmi informácií boli:

- Opis PostgreSQL backendu, ktorý sme uviedli v kapitole 2.11.1
- Sekcia Developer FAQ na PostgreSQL wiki [28]
- PostgreSQL mail archive [29]
- Webový prehliadač zdrojových súborov doxygen [30]
- Komentáre v zdrojových kódoch
- Prezentácia o modifikovaní PostgreSQL [31]

Na začiatku sme museli pracne analyzovať zdrojové kódy a hľadať miesta, na ktorých navrhnuté služby implementovať a niektoré riešenia konzultovať priamo s komunitou vývojárov PostgreSQL. Nakoniec sa väčšina zmien, ktoré si vyžiadalo pridanie podpory transakčného času, týkala modulov parser (zaoberá sa prekladom a spracovávaním SQL príkazov preložených do vnútornej štruktúry) alebo command (zabezpečuje spracovanie a vykonávanie príkazov na vytváranie a úpravu databázových objektov). Tým, že sa nám podarilo takmer všetky potrebné zmeny sústrediť do týchto modulov, sme sa vyhli modifikácii zložitých častí akými sú planner a executor.

5.2 Úpravy systémových katalógov

Riešenie je založené na vytváraní interných objektov, ktoré zabezpečujú správu verzií záznamov a informácie o týchto objektoch bolo nutné niekde uložiť. Na to slúžia systémové katalógy, ktoré sme pre vlastné potreby rozšírili o stĺpce potrebné pre zachytenie typov objektov a vzťahov medzi nimi. Napríklad do katalógu pg_class, ktorý ukladá informácie o tabuľkách, sekvenciách a indexoch sme pridali stĺpce:

- reltemporaltype – určuje či má tabuľka podporu transakčného času
- relisinternal – určuje či je tabuľka, sekvencia alebo index interná
- relhistoryrel – ak je tabuľka temporálna, obsahuje ID k nej prislúchajúcej historickej tabuľky

Štruktúra katalógov a dáta pre iniciálne naplnenie sú definované v hlavičkových súboroch jazyka C, z ktorých sa generujú (pomocou jazyka Perl) špeciálne skripty na ich vytváranie a napĺňanie vo fáze inicializácie databázovej inštancie (príkaz initdb).

5.3 Rozšírenie syntaxe SQL príkazov

Rozšírenie syntaxe jazyka SQL si vyžiadalo úpravu gramatiky zapísanej v BNF forme, z ktorej sa pomocou nástroja Bison [27] generuje prekladač. Rozšírenie príkazov v gramatike sa odrazilo aj na vnútorných štruktúrach. Buď bolo potrebné pridať úplne nové štruktúry, alebo existujúce rozšíriť o nové položky. Nasledujúci úsek kódu zachytáva prechodové pravidlá neterminálneho symbolu opt_transactiontime, ktorý reprezentuje temporálnu voľbu v príkazoch SELECT. Pri preklade sú informácie ukladané do nami vytvorenej vnútornej štruktúry TransTimeClause.

```
opt_transactiontime:
        NONSEQUENCED TRANSACTIONTIME
            {
                $$ = makeNode(TransTimeClause);
                $$->ttQueryType = TRANSACTIONTIME_NONSEQUENCED;
                $$->ttAsOfTimestamp = NULL;
            }
        | CURRENT_P TRANSACTIONTIME
            {
                $$ = makeNode(TransTimeClause);
                $$->ttQueryType = TRANSACTIONTIME_CURRENT;
                $$->ttAsOfTimestamp = NULL;
            }
        | TRANSACTIONTIME AS OF a_expr
            {
                $$ = makeNode(TransTimeClause);
                $$->ttQueryType = TRANSACTIONTIME_AS_OF;
                $$->ttAsOfTimestamp = $4;
            }
        | /*EMPTY*/
            { $$ = NULL; }
    ;
```

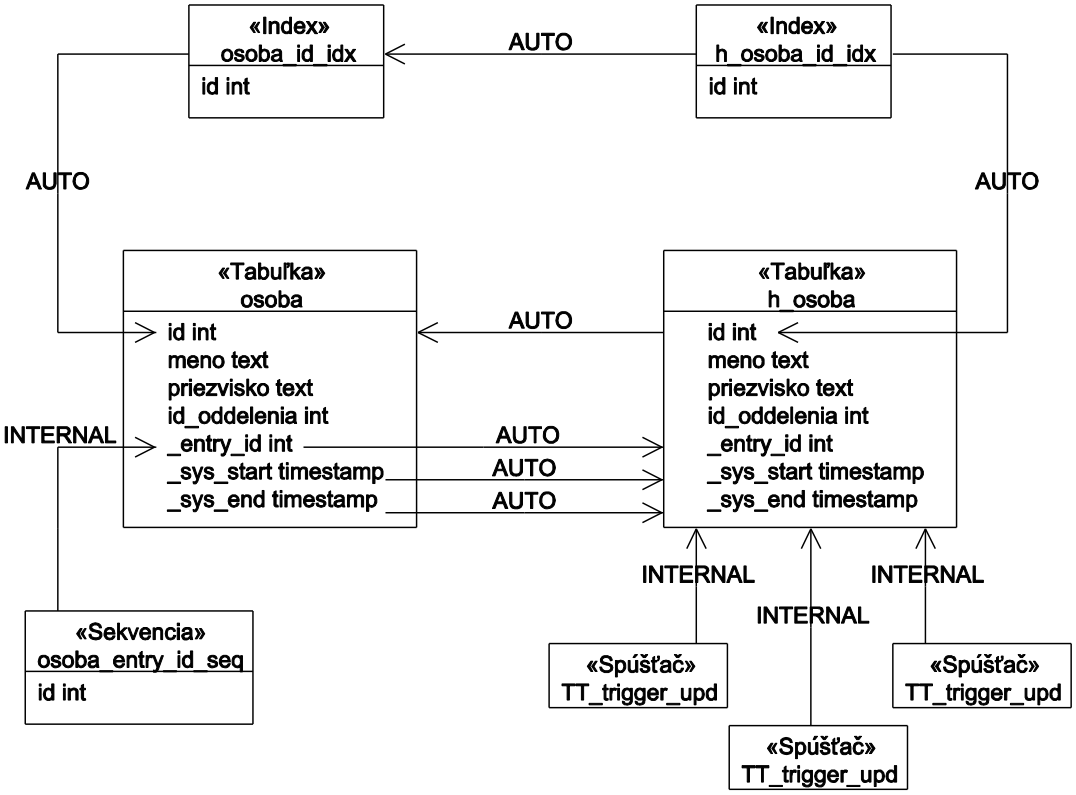
5.4 Závislosti interných objektov

V PostgreSQL sú závislosti medzi objektmi zachytené v systémovom katalógu pg_depend. Používa sa najmä pri zmazávaní objektov, kedy je potrebné zmazať aj závislé objekty alebo práve naopak zamedziť zmazaniu, lebo iné objekty daný objekt potrebujú. Definované sú nasledovné druhy závislostí [17]:

- DEPENDENCY_NORMAL – normálna závislosť medzi osobitne vytvorenými objektmi. Objekt môže byť zmazaný bez ohľadu na objekt, od ktorého je závislý. Objekt, od ktorého sú závislé iné objekty, môže byť zmazaný len pri uvedení módu CASCADE v príkaze DROP, kedy sa zmažú aj všetky závislé objekty.
- DEPENDENCY_AUTO – objekt môže byť zmazaný osobitne od objektu, od ktorého je závislý a bude automaticky zmazaný (bez ohľadu na uvedený mód CASCADE alebo RESTRICT) pri zmazaní objektu, od ktorého je závislý.
- DEPENDENCY_INTERNAL – vytvorenie objektu bolo súčasťou vytvorenia objektu, od ktorého je závislý a slúži pre jeho interné potreby. Zmazanie takéhoto objektu je automaticky zakázané a používateľ je vyzvaný, aby zmazal objekt, od ktorého je závislý. Zmazanie objektu, od ktorého interne závisia iné objekty spôsobí bez ohľadu na mód zmazanie závislých objektov.
- DEPENDENCY_PIN – znamená, že systém samotný je závislý na objekte a tak tento objekt nemôže byť nikdy zmazaný. Objekty s takouto závislosťou sú vytvárané len pri inicializácii databázovej inštancie (príkaz initdb).

Pri vytváraní objektov pre podporu transakčného času využívame závislosti DEPENDENCY_AUTO a DEPENDENCY_INTERNAL. Na **Obr. 23** sú znázornené závislosti temporálnej tabuľky osoba. Cieľom takto definovaných závislostí je, aby sa pri zmazaní historickej tabuľky zmazali aj všetky ostatné objekty vytvorené pri zapnutí podpory transakčného času. Preto sú tieto objekty buď priamo, alebo nepriamo naviazané na historickú tabuľku. Napríklad sekvencia pre stĺpec _entry_id je nepriamo (prostredníctvom tohto stĺpca) závislá na historickej tabuľke, takže zmazanie historickej tabuľky spôsobí zmazanie metadátových stĺpcov a následne aj sekvencie. Vďaka tomuto riešeniu stačí pri zrušení podpory transakčného času z tabuľky (ALTER TABLE DROP TRANSACTIONTIME)

len zmazať historickú tabuľku. Bolo však nuté na rôzne miesta modulu parser pridať zaznamenávanie závislostí.



Obr. 23. Závislosti medzi objektmi zabezpečujúcimi temporálnu podporu

5.5 Hierarchie dedení

Pridávanie podpory transakčného času do tabuľky, ktorá je koreňom hierarchie dedení, si vyžaduje automatické pridanie podpory transakčného času aj do všetkých jej nasledovníkov. Aby celá operácia prebehla úspešne, je potrebné dodržať správne poradie vytvárania interných objektov (metadátové stĺpce, historické tabuľky, indexy, sekvencie a spúšťače). Správne poradie je definované nasledovnými pravidlami:

1. Metadátové stĺpce sa musia pridať najprv potomkom, až potom rodičovi.
   V opačnom poradí by ich potomkovia zdedili a následný pokus o pridanie by skončil s chybou. Riešenie predpokladá, že metadátové stĺpce musí mať každá tabuľka vlastné (nezdedené).

2. Metadátové stĺpce v aktuálnej tabuľke a k nim prislúchajúce indexy sa musia vytvoriť pred vytvorením historickej tabuľky. Dôvodom je použitie príkazu CREATE TABLE LIKE na vytváranie historickej tabuľky, ktoré umožňuje skopírovanie štruktúry a indexov tabuľky, podľa ktorej sa vytvára.
3. Historická tabuľka sa musí vytvoriť najprv pre rodiča, až potom pre potomkov. Toto pravidlo je nevyhnutné preto, aby bolo možné vytvoriť vzťahy dedenia aj medzi historickými tabuľkami. Rodičovská historická tabuľka musí byť pri vytváraní detskej historickej tabuľky už vytvorená, keď od nej má detská dediť. Inak by postup skončil s chybou.

Dodržanie pravidiel sme pri implementácii dosiahli použitím zoznamu tabuliek, do ktorých je potrebné pridať podporu transakčného času. Tento zoznam sme naplnili prehľadávaním hierarchie dedenia do šírky. V prvom kroku je potrebné prejsť zoznam od konca (od listov ku koreňu) a každej netemporálnej tabuľke pridať metadátové stĺpce, indexy a sekvencie, čím zabezpečíme splnenie pravidiel 1 a 2. V druhom kroku potom stačí prejsť zoznam od začiatku (od koreňa k listom) a vytvoriť zvyšné objekty (historické tabuľky a spúšťače), čím zabezpečíme splnenie pravidla 3.

5.6 Spúšťače

Pri implementácii spúšťačov na odkladanie starých verzií záznamov do historickej tabuľky bolo naším cieľom zabezpečiť čo najvyšší výkon, preto sme sa rozhodli pre ich implementáciu v jazyku C.

V kapitole návrhu sme spomenuli použitie pripravených príkazov, ktoré výrazne prispejú k rýchlosti spúšťačov, ale aby bolo možné využiť ich výhody, museli sme zvoliť vhodný spôsob ich ukladania. Pripravené príkazy ukladáme do hešovacej tabuľky, ktorej kľúče sú tvorené z položiek ID tabuľky a typ spúšťača (UPDATE, DELETE alebo TRUNCATE). Takéto kľúče jednoznačne identifikujú jednotlivé príkazy vytvárané v spúšťačoch a umožňujú k nim rýchly prístup počas celého pripojenia.

6 Overenie riešenia

V tejto kapitole sa venujeme opisu testov, ktorými sme overili, či vytvorené riešenie spĺňa požiadavky špecifikované v kapitole 3. Nasledujúce kapitoly obsahujú podrobný opis jednotlivých typov testov.

6.1 Regresné testovanie

PostgreSQL obsahuje nástroj na regresné testovanie, aby bolo možné overiť, či sa modifikáciou zdrojových kódov neporušila existujúca funkcionalita. Nástroj pozostáva z testovacích skriptov, ktoré obsahujú SQL príkazy. Ku každému skriptu existuje súbor s očakávanými výsledkami jeho behu. Pri spustení regresných testov sa skripty vykonajú a ich výsledky sa porovnajú s očakávanými. Ak sú rovnaké test prešiel, inak to znamená, že došlo k narušeniu existujúcej funkcionality.

Po nasadení temporálneho rozšírenia všetky testy prešli bez chyby, čím sme overili splnenie prvej požiadavky na spätnú kompatibilitu. Okrem toho sme upravili testy tak, že všetky tabuľky sa vytvorili s podporou transakčného času. Takto upravené testy zlyhali len v prípadoch, ktoré sme definovali ako obmedzenia riešenia, čo potvrdilo splnenie druhej požiadavky na spätnú kompatibilitu.

6.2 Akceptačné testovanie

Cieľom akceptačného testovania bolo overiť splnenie funkcionálnych požiadaviek. Keďže rozhraním systému je jazyk SQL, akceptačné testovanie sme zautomatizovali pomocou SQL skriptu, ktorý sa dá jednoducho spustiť a z jeho výstupu možno overiť, či správanie príkazov korešponduje s návrhom.

Pri vytváraní testovacieho skriptu sme vychádzali zo zoznamu funkcionálnych požiadaviek, na ktorom je založená aj štruktúra skriptu. Ku každej požiadavke v zozname prislúcha množina príkazov jazyka SQL, zvolených tak aby sme pokryli všetky scenáre použitia testovanej služby. Tento skript sme zaradili aj do množiny skriptov pre regresné testovanie PostgreSQL, aby bolo možné po vykonaní zmien overiť, či nedošlo k narušeniu správania temporálneho rozšírenia.

Výsledkom akceptačného testovania bolo splnenie všetkých funkcionálnych požiadaviek. Výstup testu je dostupný na priloženom elektronickom médiu a slúži aj ako očakávaný výstup pri regresných testoch. Testovací skript sa nachádza v prílohe B.

6.3 Výkonnostné testovanie

Na výkonnostné testovanie sme využili nástroj pgbench, ktorý je súčasťou PostgreSQL a slúži na testovanie výkonu databázového servera. Dookola spúšťa sekvenciu SQL príkazov v niekoľkých paralelných pripojeniach a vypočítava počet transakcií za sekundu (ďalej len TPS), ktoré dokáže server spracovať. Základný testovací scenár obahuje nasledovné príkazy:

```
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid,
     :aid, :delta, CURRENT_TIMESTAMP);
END;
```

Pomocou prepínačov je možné vykonať, len niektoré z týchto príkazov (napríklad len príkaz SELECT) alebo je možné použiť vlastný testovací skript. Pre naše potreby je základný scenár postačujúci, pretože umožňuje testovať výkon modifikujúcich príkazov a aj príkazov SELECT. Pred spustením skriptu je potrebné databázu zinicializovať, aby sa vytvorili tabuľky a naplnili dátami. Prepínačom je možné určovať množstvo dát, ktorým sa tabuľky naplnia.

Kvôli možnosti porovnania výkonu sme testy vykonávali na inštalácii PostgreSQL 9.0.4 bez temporálneho rozšírenia a inštalácii s temporálnym rozšírením. Pred testovaním sme nastavili v konfiguračnom súbore postgresql.conf parametre:

```
shared_buffers = 256MB
wal_buffers = 16MB
checkpoint_segments = 16
```

Hardvérová konfigurácia pri testoch bola nasledovná:

- Procesor: Intel i7 – 2670QM, 8 jadier s frekvenciou 2.20 GHz.
- RAM: 8GB DDR3
- OS: Windows 7 64bit

Testovaciu databázu sme vytvorili použitím príkazov:

```
createdb -U postgres pgbench
pgbench -U postgres -i -s 10 pgbench
```

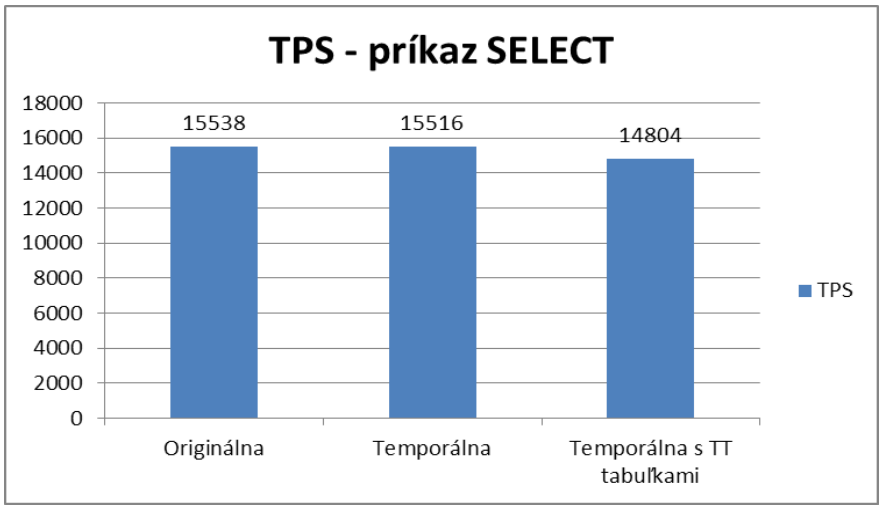
Tabuľka pgbench_accounts obsahuje, pri hodnote prepínača –s 10, 1000000 záznamov. Tieto nastavenia boli východiskové pre všetky vykonané testy, ktoré opíšeme v nasledujúcich podkapitolách.

6.3.1 Porovnanie výkonu príkazu SELECT

Tento test spočíval vo vykonávaní príkazov SELECT nad testovacou databázou. Porovnávali sme pri tom inštaláciu bez temporálneho rozšírenia , inštaláciu s temporálnym rozšírením a inštaláciu s temporálnym rozšírením so zapnutou podporou transakčného času na všetkých tabuľkách. Príkaz na vykonanie testu má tvar:

```
pgbench -U postgres -T 30 -j 2 -S -c 4 pgbench
```

Test trvá 30 sekúnd (-T 30), počas ktorých 4 klienti (-c 4) bežiaci v 2 vláknach (-j 2), dookola vykonávajú len príkaz SELECT (-S) zo základného pgbench scenáru zobrazeného na začiatku kapitoly 6.3. Pre každú inštaláciu sme test nechali zbehnúť 6 krát a výsledky jednotlivých behov nakoniec spriemerovali, aby bolo meranie čo najpresnejšie. Výsledky testov sú zachytené v grafe na **Obr. 24**. Z grafu je vidieť, že temporálne rozšírenie nijak neznížilo výkonnosť príkazu SELECT a že pridanie podpory transakčného času do tabuliek spôsobilo len 5,7% zhoršenie, čo je pravdepodobne spôsobené zvýšením objemu dát v tabuľke v dôsledku pridania metadátových stĺpcov.



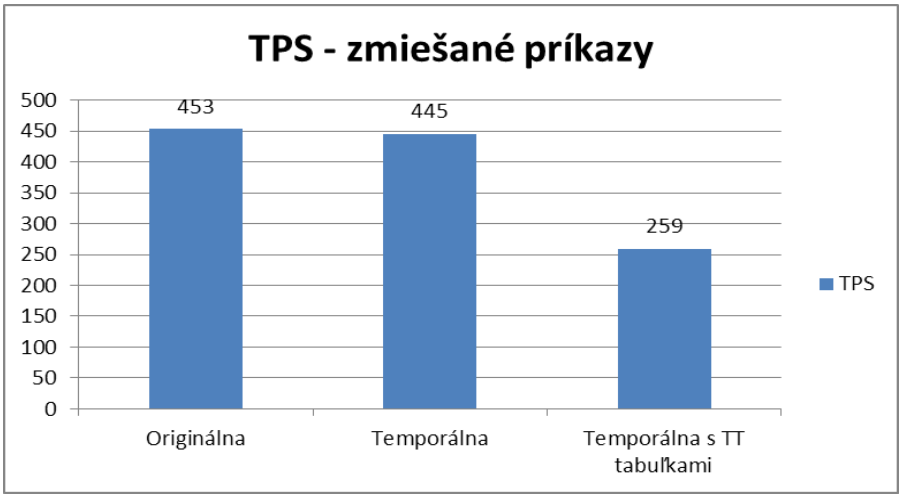
Obr. 24. Graf porovnania výkonnosti inštalácií pri príkaze SELECT

6.3.2 Porovnanie výkonu zmiešaných príkazov

Tento test spočíval vo vykonávaní celého skriptu dodávaného s pgbench. Porovnávali sme, rovnako ako v predchádzajúcom prípade, inštaláciu bez temporálneho rozšírenia , inštaláciu s temporálnym rozšírením a inštaláciu s temporálnym rozšírením so zapnutou podporou transakčného času na všetkých tabuľkách. Príkaz na vykonanie testu má tvar:

```
pgbench -U postgres -T 30 -j 2 -c 4 pgbench
```

Test trvá 30 sekúnd (-T 30), počas ktorých 4 klienti (-c 4) bežiaci v 2 vláknach (-j 2), dookola vykonávajú základný pgbench scenár zobrazený na začiatku kapitoly 6.3. Pre každú inštaláciu sme test nechali zbehnúť 6 krát a výsledky jednotlivých behov nakoniec spriemerovali, aby bolo meranie čo najpresnejšie. Výsledky testov sú zachytené v grafe na **Obr. 25**. Z grafu je vidieť, že temporálne rozšírenie nijak neznížilo výkon príkazov a že pridanie podpory transakčného času do tabuliek spôsobilo 41,7% zhoršenie, čo je spôsobené vedením histórie zmien záznamov.



Obr. 25. Graf porovnania výkonnosti inštalácií pri zmiešaných príkazoch

6.3.3 Porovnanie výkonu temporálneho a netemporálneho príkazu SELECT

Tento test spočíval vo vykonávaní temporálneho a netemporálneho príkazu SELECT na inštalácii s temporálnym rozšírením a zapnutou podporou transakčného času. Porovnávali sme ich výkon nad tabuľkami, na ktorých bol predtým vykonaný rôzny počet modifikujúcich operácií (to znamená rôzne objemy dát v historických tabuľkách). Počty

modifikujúcich operácií sme menili spúšťaním základného pgbench skriptu na rôzne dlhý čas:

```
pgbench -U postgres -T 30 -j 2 -c 4 pgbench
pgbench -U postgres -T 45 -j 2 -c 4 pgbench
pgbench -U postgres -T 60 -j 2 -c 4 pgbench
```

Príkaz na vykonanie testu s netemporálnym príkazom SELECT má tvar:

```
pgbench -U postgres -T 30 -j 2 -S -c 4 pgbench
```

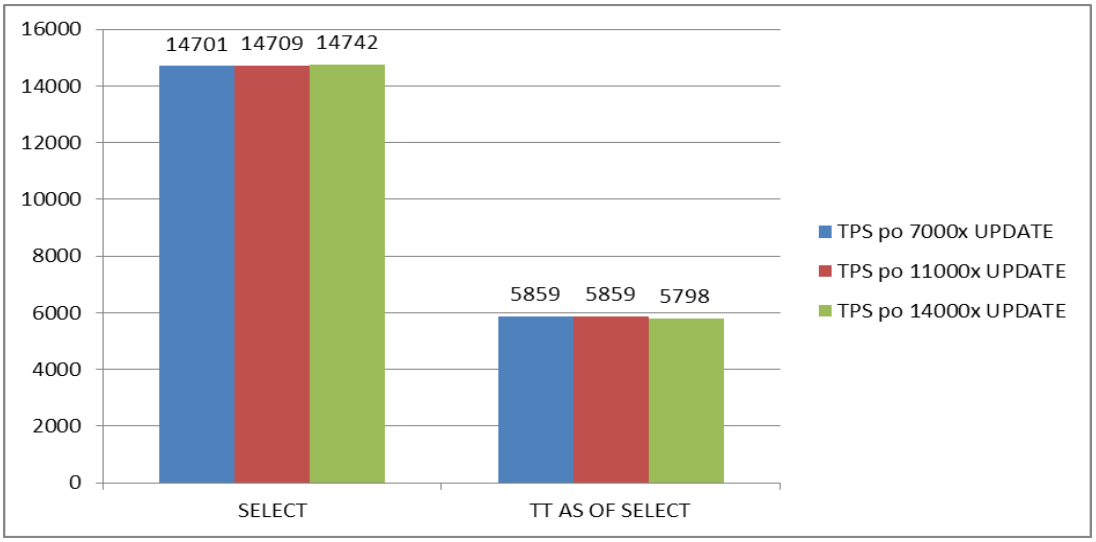
Test trvá 30 sekúnd (-T 30), počas ktorých 4 klienti (-c 4) bežiaci v 2 vláknach (-j 2), dookola vykonávajú len príkaz SELECT (-S) zo základného pgbench scenáru zobrazeného na začiatku kapitoly 6.3. Príkaz na vykonanie testu s netemporálnym príkazom SELECT má tvar:

```
pgbench -U postgres -T 30 -j 2 -x -c 4 pgbench
```

Tento príkaz sa od predchádzajúceho líši len v prepínači –x, ktorý nahradil prepínač –S. Prepínač –x sme doplnili do nástroja pgbench pre potreby testovania temporálnych príkazov SELECT a obsahuje volanie príkazu:

```
TRANSACTIONTIME AS OF '2012-12-12 12:12:12.123456' SELECT abalance FROM
     pgbench_accounts WHERE aid = :aid;
```

Pre všetky počty modifikujúcich operácií sme oba testy nechali zbehnúť 6 krát a výsledky jednotlivých behov nakoniec spriemerovali, aby bolo meranie čo najpresnejšie. Výsledky testov sú zachytené v grafe na **Obr. 26**. Z grafu je vidieť, že temporálny SELECT dosahuje o 60% horší výkon, čo je spôsobené tým, že musí pristupovať aj k dátam v historickej tabuľke. Okrem toho je vidieť, že zmeny veľkosti historickej tabuľky o rádovo 1000 záznamov nemajú vplyv na výkon temporálnych príkazov SELECT.



Obr. 26. Porovnanie temporálneho a netemporálneho príkazu SELECT

6.4 Použitie riešenia v existujúcej aplikácii

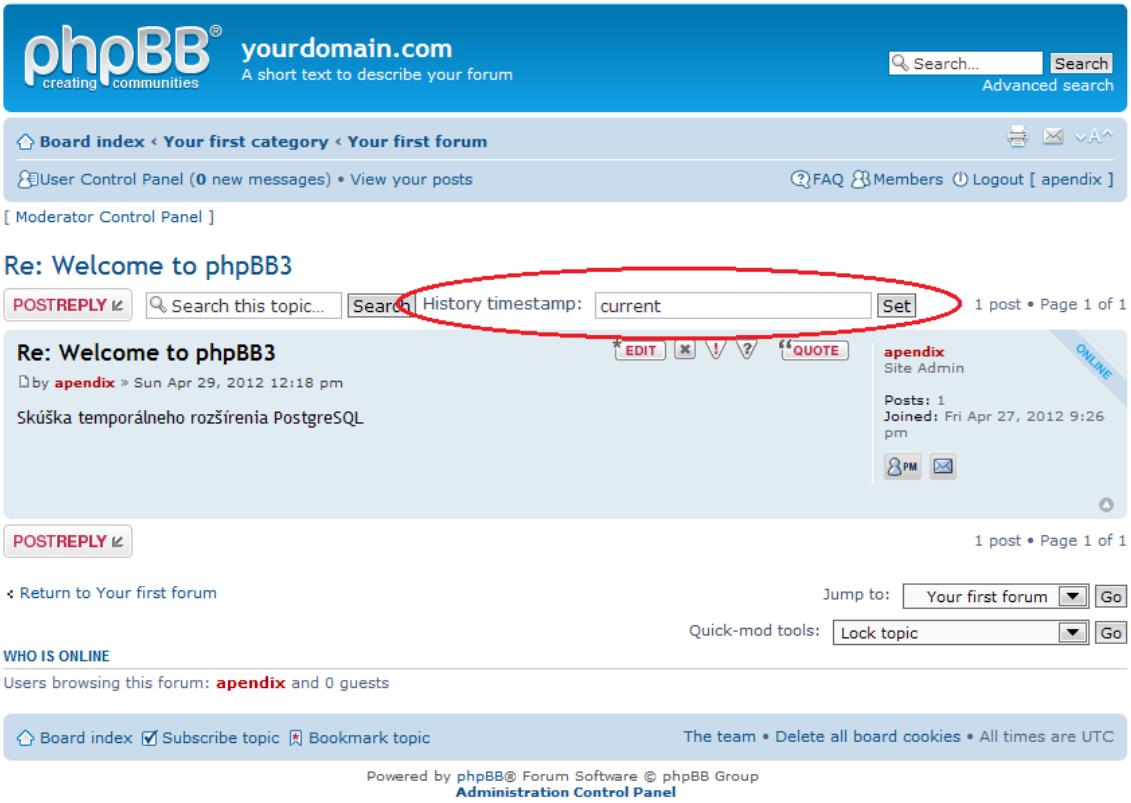
Jednou z hlavných požiadaviek na riešenie bola možnosť použitia v existujúcej aplikácii bez nutnosti zmeny aplikácie. Splnenie tejto požiadavky sme overili na webovom fóre s otvorenými zdrojovými kódmi – phpBB [32].

Po inštalácii sme pridali podporu transakčného času na tabuľku obsahujúcu príspevky v jednotlivých témach, aby sa bolo možné vracať k upraveným alebo zmazaným príspevkom. Na to stačilo vykonať príkaz:

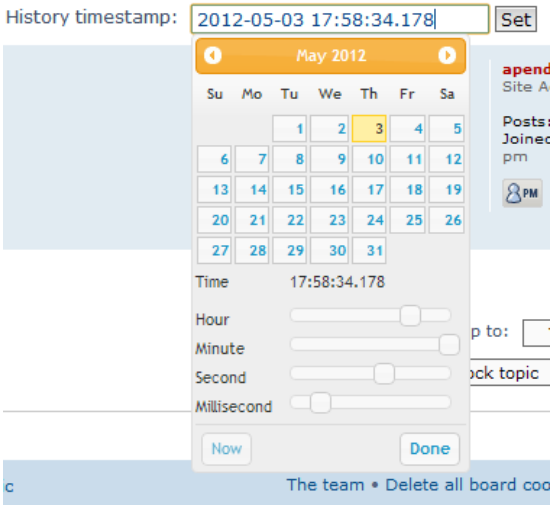
```
ALTER TABLE phpbb_posts ADD TRANSACTIONTIME;
```

V aplikácii sme nevykonali žiadne zmeny, správanie sa pridaním podpory transakčného času nijak nezmenilo a verziovanie príspevkov fungovalo správne, čím sme dokázali splnenie požiadavky na použitie riešenia v existujúcej aplikácii.

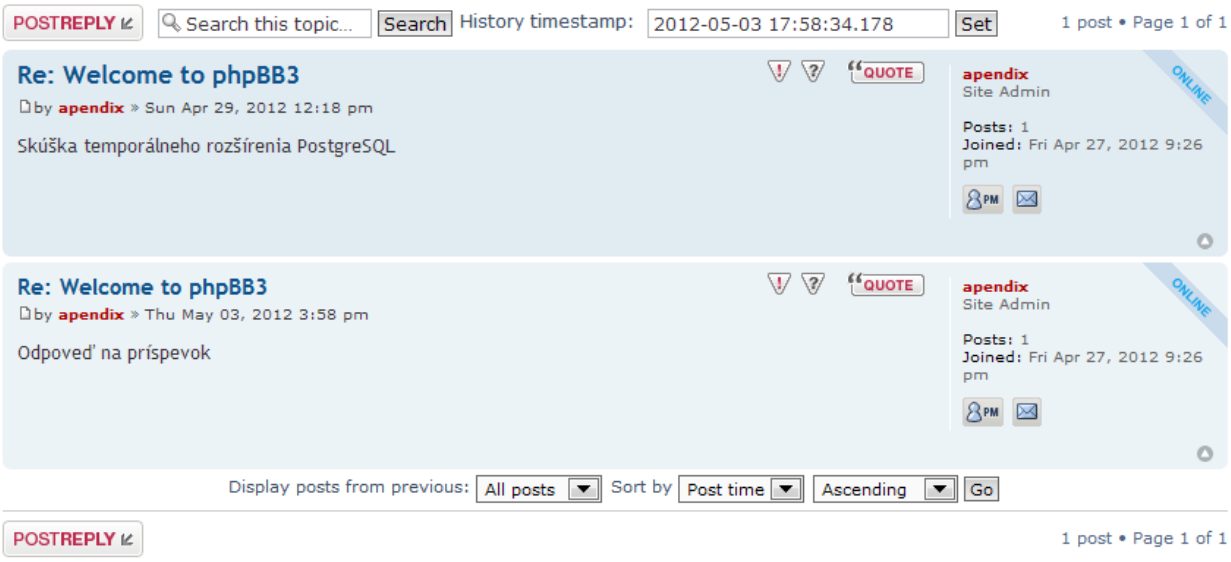
Kvôli praktickej ukážke využitia riešenia sme do aplikácie pridali vstupné pole na zadávanie času (**Obr. 27**) a nastavovanie databázovej premennej history_timestamp (na čas zo vstupu) pred načítaním príspevkov. Vďaka týmto dvom zmenám majú používatelia možnosť vidieť stav témy v ľubovoľnom čase. Po zvolení času (**Obr. 28**) a potvrdení tlačidlom „Set“ sa používateľovi zobrazia príspevky existujúce v zadanom časovom momente (**Obr. 29**).



Obr. 27. phpBB – upravené rozhranie



Obr. 28. phpBB – výber dátumu a času



Obr. 29. phpBB – zobrazenie príspevkov z minulosti

7 Zhodnotenie

V tejto práci sme priniesli rozsiahly prehľad oblasti temporálnych databáz. V rámci uvedenia do problematiky sme, vychádzajúc z dostupných štandardov, vysvetlili základné pojmy v tejto oblasti. Následne sme zanalyzovali existujúce riešenia, vzájomne ich porovnali a zhodnotili. Zistili sme, že neexistuje žiadna voľne dostupná databáza, ktorá by poskytovala temporálnu podporu.

V ďalších kapitolách sme opísali dve možnosti riešenia správy histórie dát v databázovom systéme PostgreSQL - použitím prostriedkov databázy (bez zásahu do zdrojových kódov) a modifikovaním zdrojových kódov databázy. Kvôli množstvu výhod druhého riešenia sme sa v práci ďalej venovali už len tomuto riešeniu.

S cieľom vytvorenia riešenia, ktoré by dokázalo konkurovať existujúcim proprietárnym systémom, sme špecifikovali požiadavky na temporálne rozšírenie PostgreSQL. Na ich základe sme navrhli, naimplementovali a otestovali produkt, ktorý umožňuje vedenie histórie v čase sa meniacich dát priamo v databáze pri minimálnych obmedzeniach na existujúce aplikácie, ktoré s ňou pracujú. Bez zásahu do aplikácie, je možné pridať podporu transakčného času na vybrané tabuľky a následne sa vracať k stavom dát z minulosti, ktoré by boli inak stratené.

Práca na projekte pokračuje ďalej aj po odovzdaní diplomovej práce. V súčasnosti rozbiehame proces zaradenia riešenia do budúcich verzií PostgreSQL, ktorý ak skončí úspešne, tak sa nám podarí vyplniť dieru v oblasti voľne dostupných temporálnych databáz. Riešenie je možné ďalej rozvíjať napríklad pridaním podpory času platnosti, temporálnych obmedzení, množinových operácií a agregačných funkcií.

8 Literatúra

1. Novikov, B. A., Gorshkova, E. A.: Temporal Databases: From Theory to Applications. In: *Programming and Computer Software*. Springer, 2008, Vol. 34, No. 1, pp. 1-6.
2. Chau, V.T.N., Chittayasothorn, S.: A Temporal Compatible Object Relational Database System. In: *SoutheastCon*, 2007. Proceedings. IEEE. pp. 93-98.
3. Mahmood, N., Burney, A., Ahsan, K.: A Logical Temporal Relational Data Model. In: *IJCSI International Journal of Computer Science Issues*. 2010, Vol. 7, Issue 1, No. 1, pp. 1-9.
4. Snodgrass, R. T., Bohlen, M. H., Jensen, C. S., Steiner, A.: Transitioning Temporal Support in TSQL2 to SQL3. In: *Temporal Databases: Research and Practice*. Springer, 1998, pp. 150-194.
5. Snodgrass, R. T., Bohlen, M. H., Jensen, C. S., Steiner, A.: Adding Valid Time to SQL/Temporal, ANSI X3H2-96-501r2, 1996.
6. Snodgrass, R. T., Bohlen, M. H., Jensen, C. S., Steiner, A.: Adding Transaction Time to SQL/Temporal, ANSI X3H2-96-502r2, 1996.
7. Snodgrass, R. T., Ahn, I., Ariav, G., Batory, D., Clifford, J., Dyreson, C. E., Elmasri, R., Grandi, F., Jensen, C. S., Kaefer, W., Kline, N., Kulkarni, K., Leung, T. Y. C., Lorentzos, N., Roddick, J. F., Segev, A., Soo, M. D., Sripada, S. M.: TSQL2 Language Specification. Sigmod record, Vol. 23, No. 1, 1994.
8. Snodgrass, R. T., Ahn, I., Ariav, G., Batory, D., Clifford, J., Dyreson, C. E., Elmasri, R., Grandi, F., Jensen, C. S., Kaefer, W., Kline, N., Kulkarni, K., Leung, T. Y. C., Lorentzos, N., Roddick, J. F., Segev, A., Soo, M. D., Sripada, S. M.: A TSQL2 Tutorial. Sigmod record, Vol. 23, No. 3, 1994.
9. Snodgrass, R. T., Ahn, I., Ariav, G., Batory, D., Clifford, J., Dyreson, C. E., Elmasri, R., Grandi, F., Jensen, C. S., Kaefer, W., Kline, N., Kulkarni, K., Leung, T. Y. C., Lorentzos, N., Roddick, J. F., Segev, A., Soo, M. D., Sripada, S. M.: The TSQL2 Temporal Query Language, Kluwer Academic Publishers, 1995.
10. Snodgrass, R. T.: Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann Publishers, San Francisco, 2000.

11. Steiner, A.: A Generalisation Approach to Temporal Data Models and their Implementations, 1998.
12. Snodgrass, R. 2011. TSQL2 Temporal Query Language. http://www.cs.arizona.edu/people/rts/tsql2.html
13. Snodgrass, R. 2011. TSQL2 and SQL3 Interactions. http://www.cs.arizona.edu/people/rts/sql3.html
14. Beauregard, B., Murray, C., Speckhard, B.: Oracle Database 11g Workspace Manager Overview. An Oracle White Paper, 2009.
15. Saracco, C. M., Nicola, M., Gandhi, L.: A Matter of Time: Temporal Data Management in DB2 for z/OS. IBM Corporation, New York, 2010.
16. Teradata Database: Temporal Table Support. Teradata Corporation, 2010.
17. PostgreSQL. 2011. Documentatation. http://www.postgresql.org/docs/
18. MySQL. 2011. Documentation. http://dev.mysql.com/doc/refman/5.5/en/
19. PostgreSQL. 2011. How PostgreSQL Processes a Query. http://www.postgresql.org/developer/ext.backend.html
20. Oracle. 2010. Introduction to Workspace Manager. http://download.oracle.com/docs/cd/E11882_01/appdev.112/e11826/long_intro.htm#i1012409
21. IBM DB2 10 for z/OS. 2010. Documentation. http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db2z10.doc/db2z_10_prodhome.htm
22. Asserted Versioning Framework. 2010. http://www.assertedversioning.com/solution.asp
23. IBM - DB2 DataPropagator. 2010. http://www-01.ibm.com/software/data/integration/replication
24. LogExplorer. 2010. http://www.ssw.com.au/LogExplorer
25. TimeDB. 2005. http://www.timeconsult.com/Software/Software.html
26. Atempo Time Navigator. 2011. http://www.atempo.com/products/timeNavigator/default.asp
27. Bison. 2012. http://www.gnu.org/software/bison/
28. PostgreSQL wiki - Developer FAQ. 2012. http://wiki.postgresql.org/wiki/Developer_FAQ

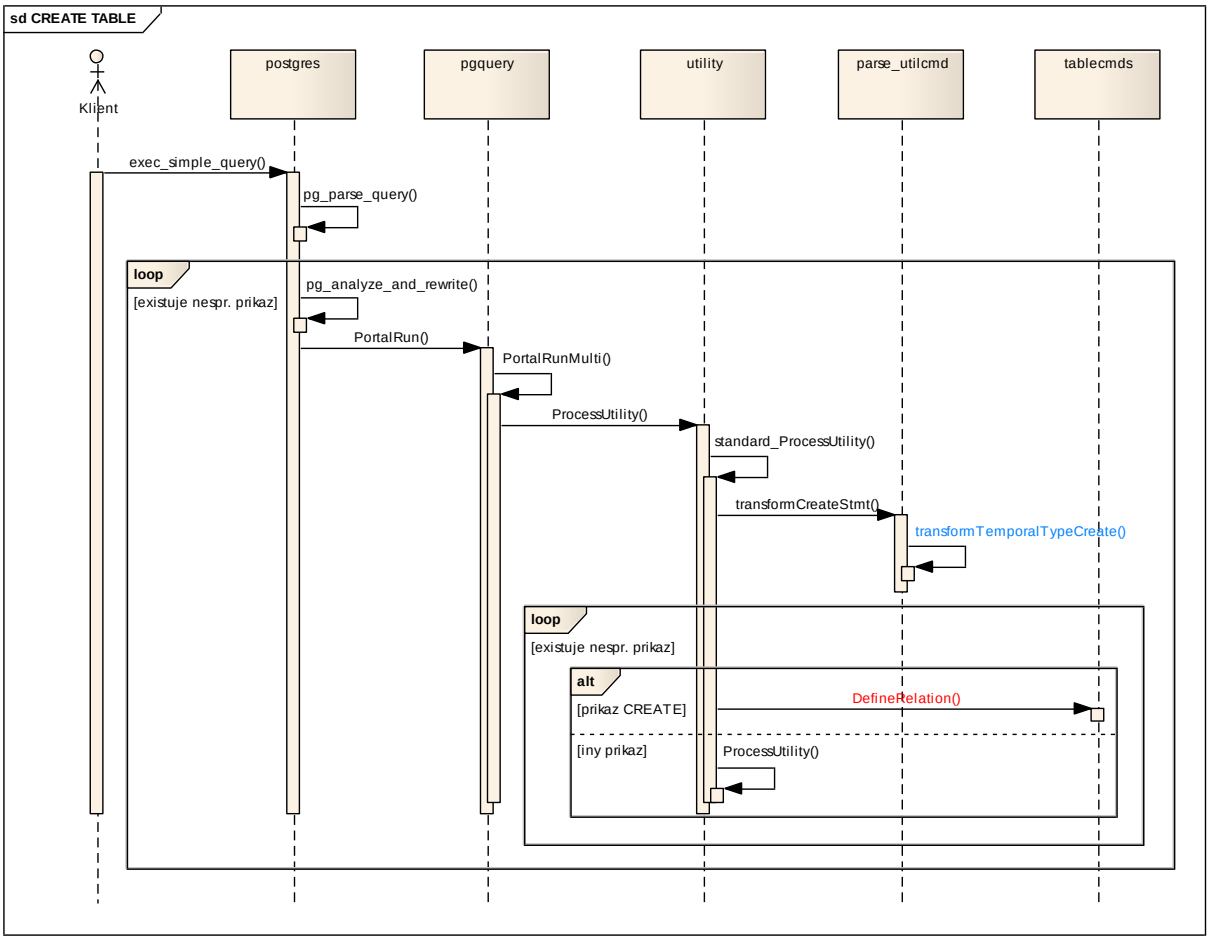
29. PostgreSQL Mailing Lists Archives. 2012. http://archives.postgresql.org/
30. PostgreSQL Source Code Browser. 2012. http://doxygen.postgresql.org/
31. Sherry G.: Introduction to Hacking PostgreSQL. PgCon, Ottawa, 2007. http://www.neilconway.org/talks/hacking/ottawa/ottawa_slides.pdf
32. phpBB. 2012. http://www.phpbb.com/
33. MinGW. 2012. http://www.mingw.org/

Príloha A: Technická dokumentácia k implementácii

V tejto prílohe sa nachádzajú podrobné opisy implementácie vybraných služieb opísaných v kapitole 4, rozdelených podľa príkazov, ktorými sa k nim pristupuje. Obsahuje ukážky kódu a sekvenčné diagramy znázorňujúce funkcie, ktoré boli pridané (zvýraznené modrou farbou) alebo modifikované (zvýraznené červenou farbou), v kontexte spracovania príkazu v PostgreSQL.

A.1 CREATE TABLE

Spracovanie príkazu CREATE TABLE je znázornené na **Obr. 30**. Na začiatku sa v module postgres vykoná preklad príkazu/príkazov prostredníctvom funkcie pg_parse_query. Následne sa preložený príkaz analyzuje vo funkcii pg_analyze_and_rewrite. Pre nás zaujímavou je v tomto prípade funkcia transformCreateStmt, ktorá na vstupe dostane samostatný príkaz CREATE TABLE a jej výstupom je množina príkazov, ktoré sa musia vykonať popri vytváraní tabuľky (napríklad príkaz na vytvorenie indexu pri zadefinovaní UNIQUE stĺpca alebo príkaz na vytvorenie sekvencie pri zadefinovaní stĺpca typu SERIAL). Práve táto funkcia je vhodným miestom na vytvorenie objektov potrebných pre podporu transakčného času (historická tabuľka, metadátové stĺpce, spúšťače a iné). Vytvorené príkazy sa rekurzívne spracovávajú volaním ProcessUtility a spracovanie CREATE TABLE príkazu pokračuje vo funkcii DefineRelation.



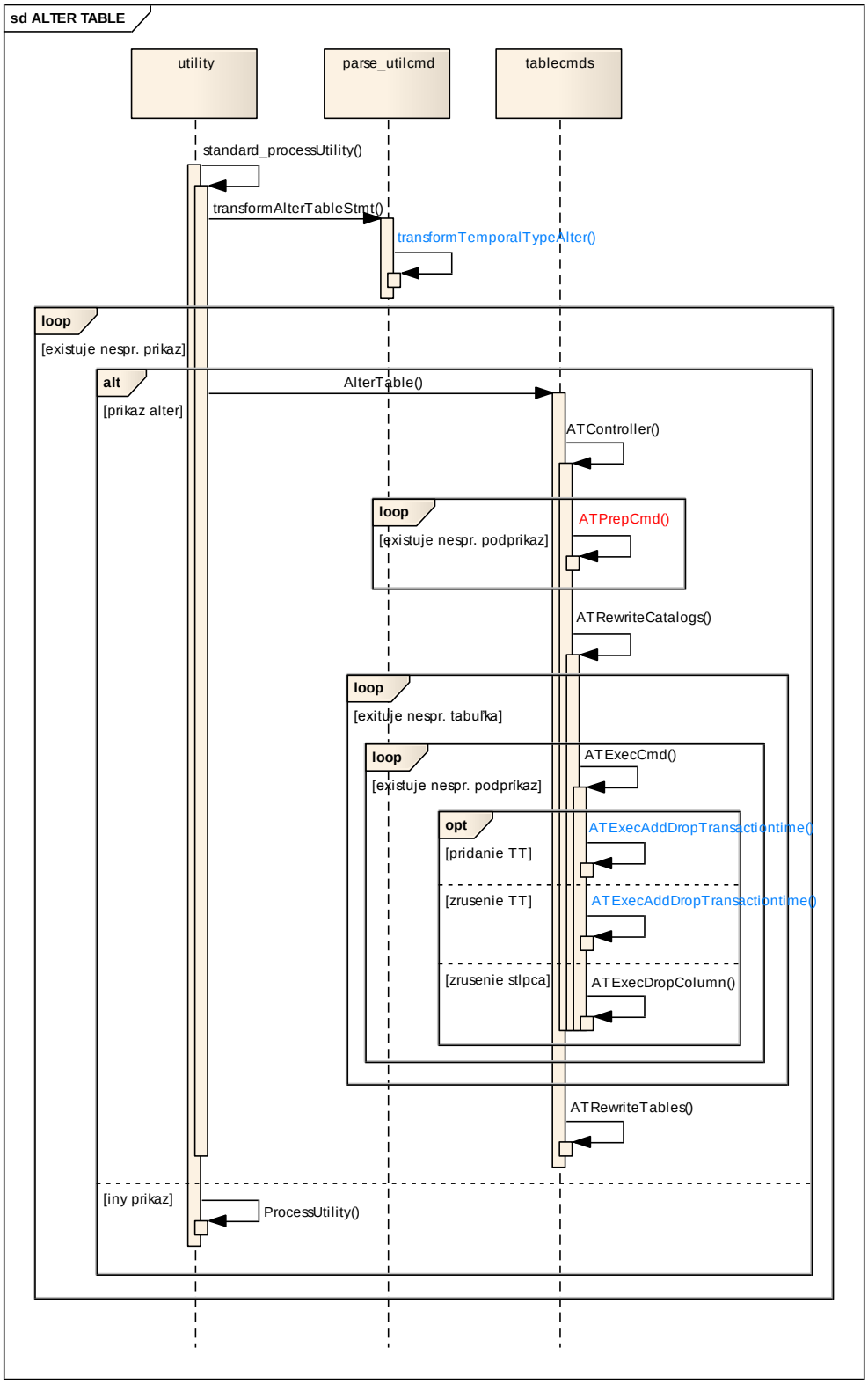
Obr. 30. Sekvenčný diagram príkazu CREATE TABLE

Hlavná časť vytvárania TT tabuliek je naimplementovaná vo funkcii transformTemporalTypeCreate, ktorá zabezpečuje vytváranie všetkých potrebných objektov a presmerovávanie dedenia podľa návrhu z kapitoly 4.1.1.1. Vo funkcii DefineRelation bolo potrebné pridať zaznamenávanie vzťahov medzi vytvorenými objektmi do systémových katalógov, ktoré je zachytené na nasledujúcej ukážke kódu.

A.2 ALTER TABLE

Začiatok spracovania príkazu ALTER TABLE je rovnaký ako pri príkaze CREATE TABLE, preto sme ho v diagrame na **Obr. 31** vynechali. Hlavnou je funkcia transformAlterTableStmt, ktorej princíp je rovnaký, ako pri funkcii transformCreateStmt. V diagrame je ďalej bližšie rozpísané spracovanie samotného príkazu ALTER TABLE. Vo Funkcii ATPrepCmd sa kontrolujú používateľské oprávnenia a ak sa má príkaz vykonať aj na dediacich tabuľkách, tak sa dediace tabuľky načítajú. Následne sa vo funkcii ATRewriteCatalogs pre každú

tabuľku (cieľovú a dediace) a každý podpríkaz (ALTER TABLE môže mať viac podpríkazov napríklad ALTER TABLE tab1 ADD COLUMN col1, ADD COLUMN col2;) vykoná funkcia ATExecCmd, v ktorej sa spravia príslušné zmeny v systémových katalógoch.



Obr. 31. Sekvenčný diagram pre príkaz ALTER TABLE

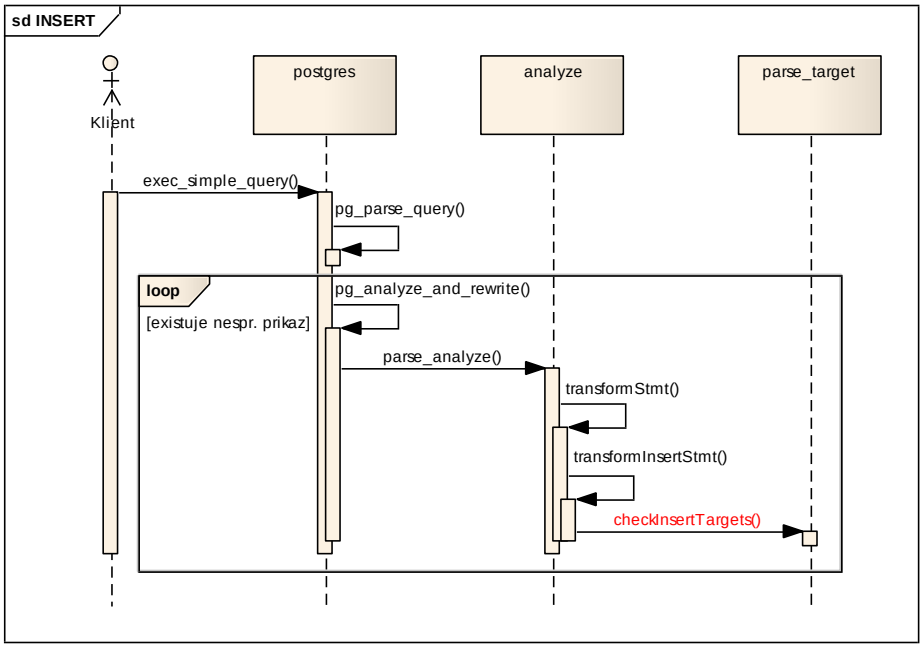
Hlavná časť pridávania TT podpory do tabuliek je naimplementovaná vo funkcii transformTemporalTypeAlter, ktorá zabezpečuje vytváranie všetkých potrebných objektov a presmerovávanie dedenia podľa návrhu z kapitoly 4.1.3.1. Okrem toho si pridanie TT podpory vyžaduje aj jej zaznamenanie do systémových katalógov, čo je náplňou funkcie ATExecAddDropTransactiontime zobrazenej na nasledujúcom úseku kódu.

```
static void
ATExecAddDropTransactiontime(Relation rel, char temporalType)
{
    Oid         myrelid = RelationGetRelid(rel);
    Relation    targetrelation;
    Relation    relrelation;    /* for RELATION relation */
    HeapTuple   reltup;
    Form_pg_class relform;

    /*
     * Grab an exclusive lock on the target table
     */
    targetrelation = relation_open(myrelid, AccessExclusiveLock);

    /*
     * Find relation's pg_class tuple
     */
    relrelation = heap_open(RelationRelationId, RowExclusiveLock);

    reltup = SearchSysCacheCopy1(RELOID, ObjectIdGetDatum(myrelid));
    if (!HeapTupleIsValid(reltup))        /* shouldn't happen */
        elog(ERROR, "cache lookup failed for relation %u", myrelid);
    relform = (Form_pg_class) GETSTRUCT(reltup);

    /*
     * Update pg_class tuple with new temporal type. (Scribbling
     * on reltup is OK because it's a copy...)
     */
    relform->reltemporaltype = temporalType;

    simple_heap_update(relrelation, &reltup->t_self, reltup);

    /* keep the system catalog indexes current */
    CatalogUpdateIndexes(relrelation, reltup);

    heap_freetuple(reltup);
    heap_close(relrelation, RowExclusiveLock);

    /*
     * Close rel, but keep exclusive lock!
     */
    relation_close(targetrelation, NoLock);
}
```

A.3 INSERT

Spracovanie príkazu INSERT je znázornené na **Obr. 32**. Po preklade zadaných príkazov vo funkcii pg_parse_query sa každý z nich zanalyzuje (funkcia parse_analyze). Pre pridanie TT podpory je v časti analýzy najdôležitejšia funkcia transformInsertStmt, ktorá zabezpečuje

rôzne kontroly a vnútorné spracovanie príkazu. V nej sa volá funkcia checkInsertTargets, v ktorej sme podľa návrhu z kapitoly 4.1.4.1 implementovali zamedzenie vkladania hodnôt do metadátových stĺpcov a vytváranie zoznamu stĺpcov pri jeho nezadaní.



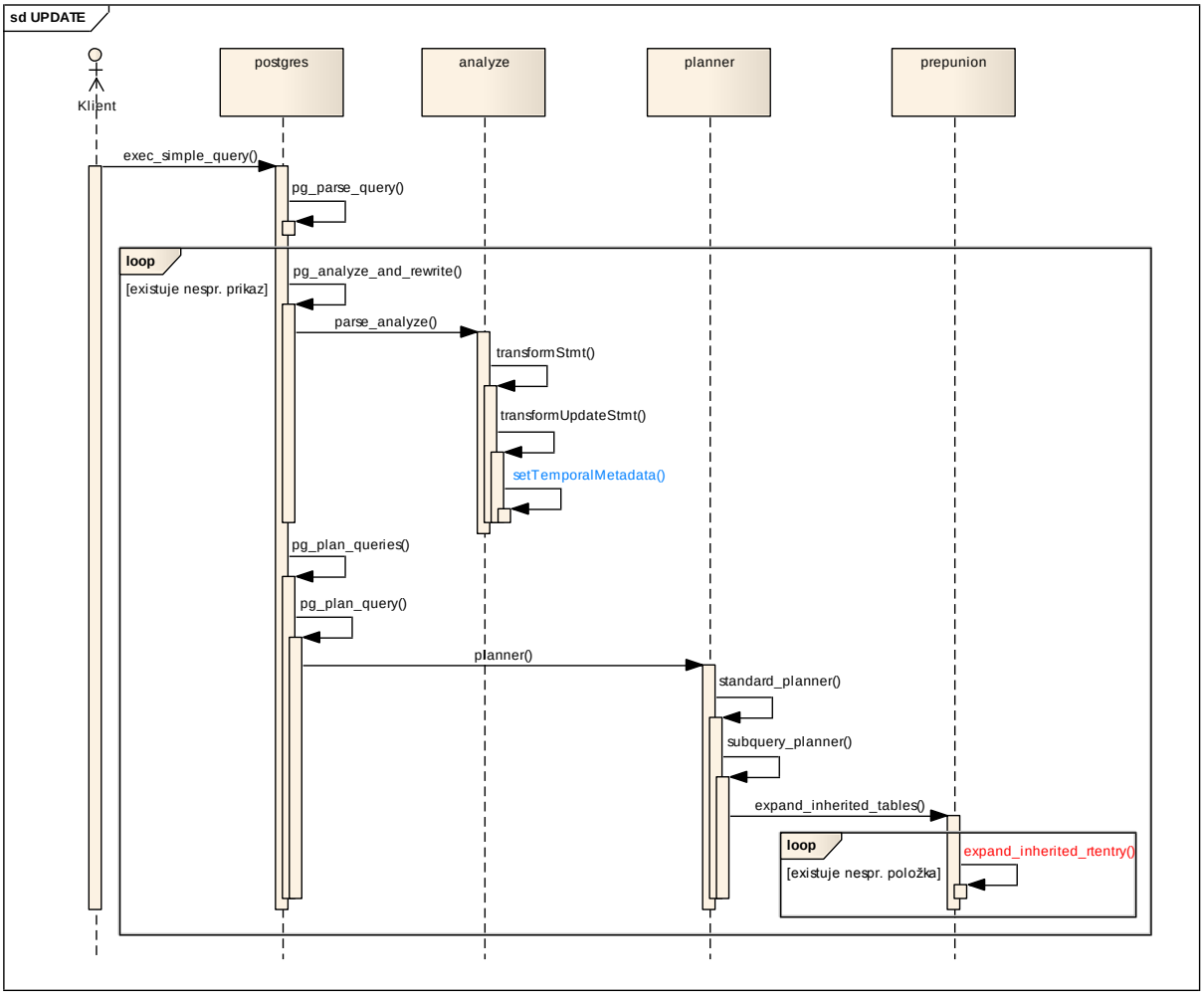
Obr. 32. Sekvenčný diagram pre príkaz INSERT

Nasledujúca ukážka kódu zachytáva vytváranie zoznamu stĺpcov príkazu INSERT. V cykle sa prejdú všetky stĺpce tabuľky a vynechajú sa tie, ktoré majú v systémovom katalógu pg_attribute nastavaný príznak attissysmaint (sú metadátové).

```
for (i = 0; i < numcol; i++)
{
    ResTarget  *col;

    /*
     * if attribute is dropped or is system maintained,
     * don't include it to list
     */
    if (attr[i]->attisdropped || attr[i]->attissysmaint)
        continue;

    col = makeNode(ResTarget);
    col->name = pstrdup(NameStr(attr[i]->attname));
    col->indirection = NIL;
    col->val = NULL;
    col->location = -1;
    cols = lappend(cols, col);
    *attrnos = lappend_int(*attrnos, i + 1);
}
```

A.4 UPDATE

Spracovanie príkazu UPDATE je znázornené na **Obr. 33**. Vo fáze analýzy (funkcia parse_analyze) je z pohľadu pridania TT podpory dôležitá funkcia transformUpdateStmt, ktorá zabezpečuje rôzne kontroly a vnútorné spracovanie príkazu. Po analýze príkazu nasleduje fáza plánovania, ktorá začína vo funkcii pg_plan_query. V tejto fáze nás zaujíma funkcia expand_inherited_tables, ktorá slúži na zahrnutie dediacich tabuliek do príkazu UPDATE (aj SELECT).



Obr. 33. Sekvenčný diagram pre príkaz UPDATE

Vo funkcii setTemporalMetadata sa vykonáva kontrola modifikácie metadátových stĺpcov a ich nastavovanie na východiskové hodnoty, čo zobrazuje aj nasledujúca ukážka kódu. Nájde sa stĺpec _sys_start a nastaví sa mu hodnota, ktorá je tvorená konštrukciou CASE WHEN, pretože rozhodnúť o správnom čase začiatku platnosti sa dá až vo fáze vykonávania, keď sa pristupuje ku konkrétnym riadkom (viac informácií v kapitole 2.10.4.4).

```
for (i = 0; i < rd->rd_rel->relnatts; i++)
{
    Form_pg_attribute att = rd->rd_att->attrs[i];

    /* if it is _sys_start atribute we must set default value */
    if (att->attissysmaint &&
        namestrcmp(&(att->attname), "_sys_start") == 0)
    {
        ResTarget          *res = makeNode(ResTarget);
        CaseExpr           *caseEx = makeNode(CaseExpr);
        CaseWhen           *whenThen = makeNode(CaseWhen);

        /* create CASE WHEN caluse*/
        whenThen->expr =
            (Expr *) makeSimpleA_Expr(
                AEXPR_OP, "=",
                makeTypeCast(
                    makeTypeCast(
                        makeColumnRef("xmin", NIL, -1),
                        SystemTypeName("text"), -1
                    ),
                    SystemTypeName("int8"), -1
                ),
                makeSystemFuncCall("txid_current", NIL, -1), -1
            );
        whenThen->result =
            (Expr *) makeColumnRef("_sys_start", NIL, -1);
        whenThen->location = -1;

        caseEx->casetype = InvalidOid;
        caseEx->arg = NULL;
        caseEx->args = list_make1(whenThen);
        caseEx->defresult =
            (Expr *) makeSystemFuncCall("clock_timestamp", NIL, -1);
        caseEx->location = -1;

        res->name = att->attname.data;
        res->indirection = NIL;
        res->val = (Node *) caseEx;
        res->location = -1;

        stmt->targetList = lappend(stmt->targetList,res);
    }
}
```

Okrem toho bolo nutné vo funkcii expand_inherited_rtentry zamedziť, aby sa príkaz UPDATE (aj SELECT) zavolaný nad obyčajnou tabuľkou vykonával aj nad historickými tabuľkami, ktoré od obyčajnej tabuľky dedia (pozri kapitolu 2.11.2).

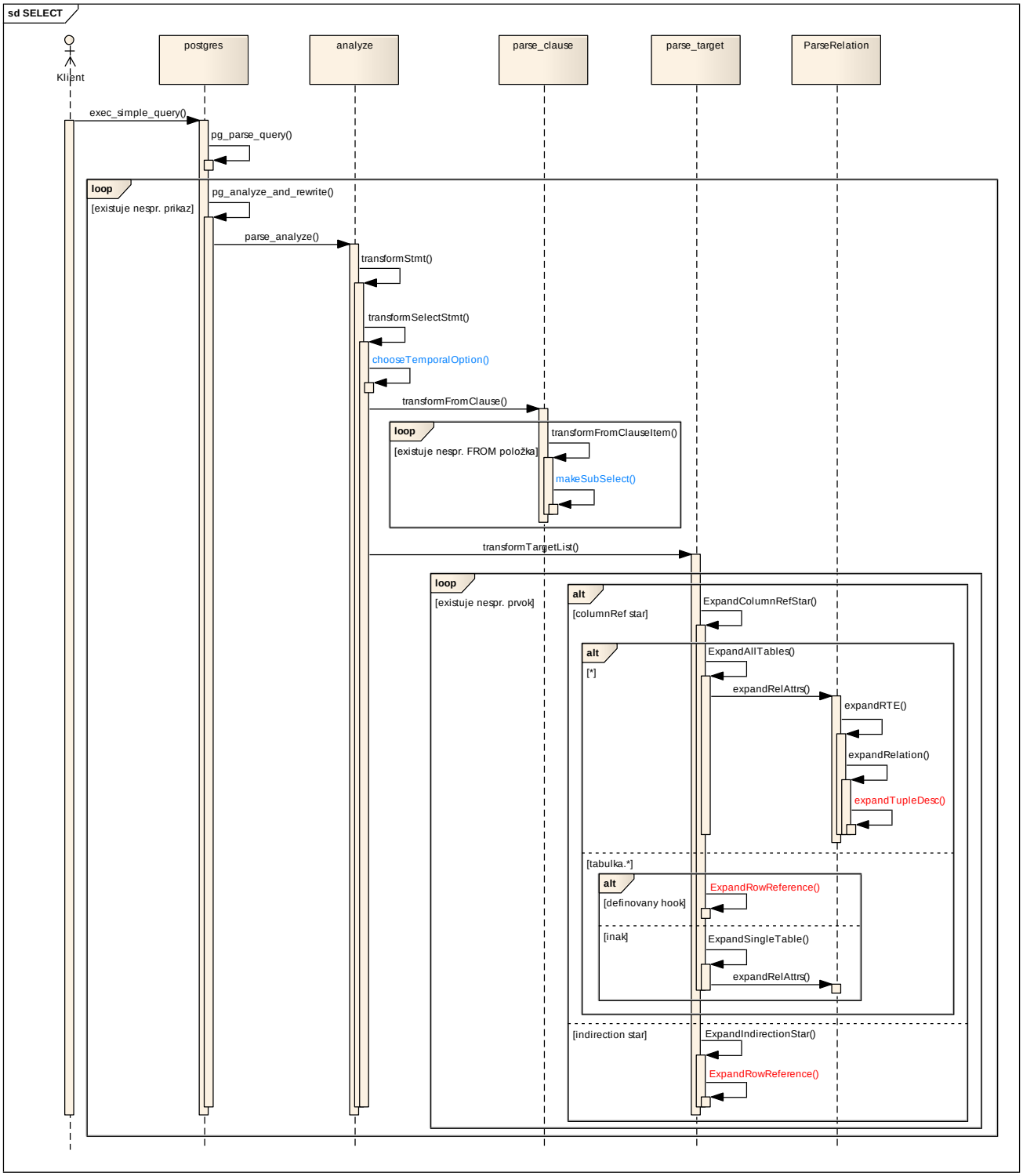
A.5 SELECT

Spracovanie príkazu SELECT je znázornené na **Obr. 34**. Všetky, pre pridanie TT podpory dôležité funkcie, sa nachádzajú vo fáze analýzy (funkcia parse_analyze). Jednou z nich je funkcia transformSelectStmt, v ktorej sa kontrolujú a ďalej spracovávajú jednotlivé časti príkazu. Ďalšou dôležitou funkciou je transformFromClause, ktorá má na starosti

spracovanie elementov z FROM časti príkazu (napríklad rekurzívne spracovanie podpríkazov).

Funkcia transformTargetList je pre nás zaujímavá, lebo sa v nej zabezpečuje prepis znaku hviezdičky na zoznam stĺpcov (vo funkciách zvýraznených červenou farbou). Pri vytváraní zoznamu stĺpcov bolo nutné preskočiť metadátové stĺpce, ku ktorým sa dá pristupovať len explicitným vymenovaním.

Výber výsledného temporálneho nastavenia príkazu SELECT prebieha vo funkcii chooseTemporalOption a ak je výsledkom nesekvenčný dotaz alebo dotaz nad záznamami z daného času, tak sa vo funkcii makeSubSelect nahradia TT tabuľky podpríkazmi SELECT.

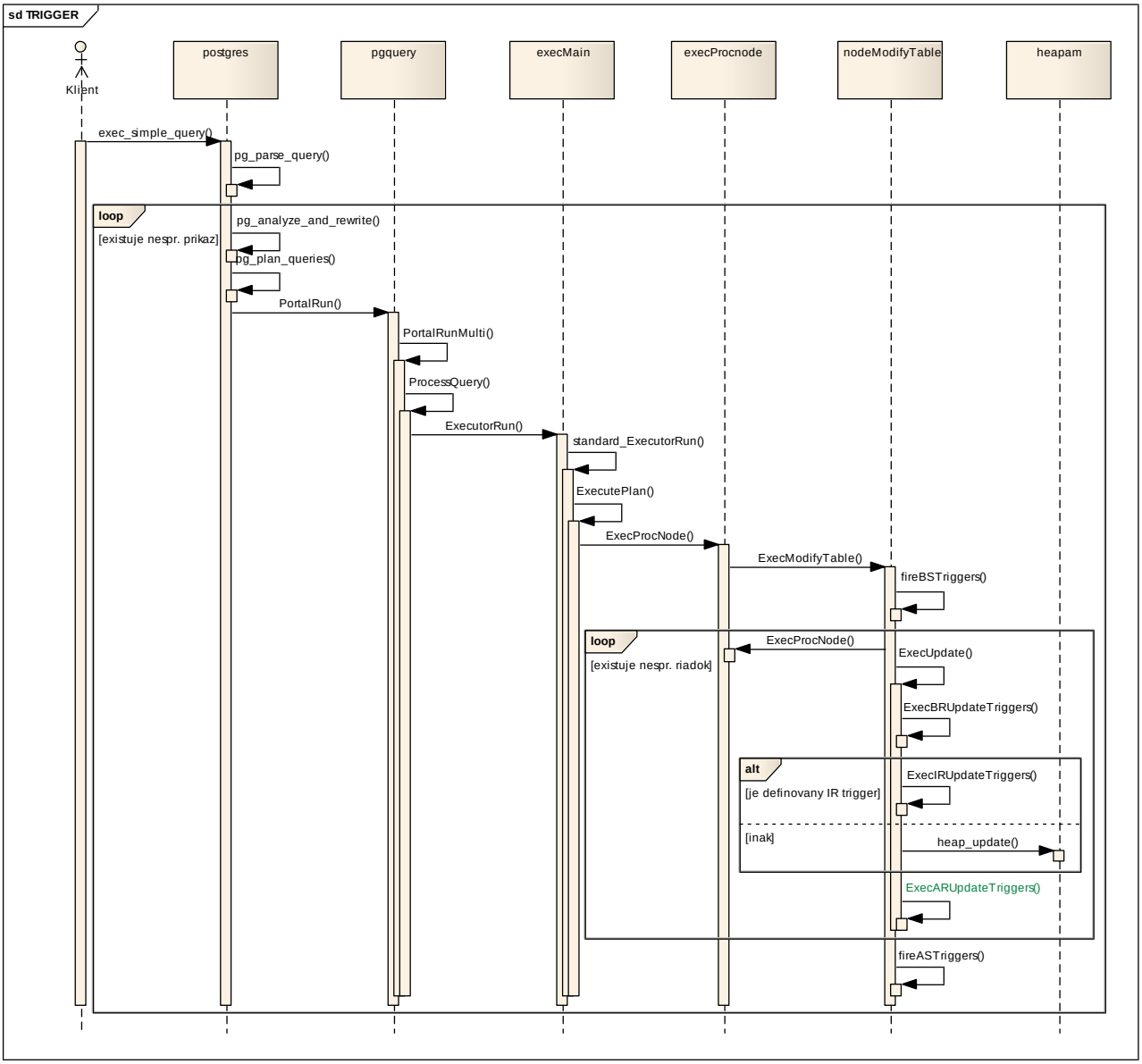


Obr. 34. Sekvenčný diagram pre príkaz SELECT

A.6 Spúšťače na odkladanie starých verzií záznamov

Na **Obr. 35** ja znázornené spracovanie príkazu UPDATE so zameraním na spúšťače. Dôležitá pre naše riešenie je funkcia ExecARUpdateTriggers, v ktorej sa volajú spúšťače nastavené na

vykonanie po príkaze. V tejto funkcii nie sú potrebné žiadne zmeny, ale je vhodné ukázať kontext, v ktorom sa spúšťače volajú.



Obr. 35. Sekvenčný diagram pre volanie spúšťačov

Príloha B: Technická dokumentácia k testovaniu

Táto príloha obsahuje skript použitý pri akceptačnom testovaní riešenia. Príkazy sú zoskupené hierarchicky podľa funkcionálnych požiadaviek špecifikovaných v kapitole 3.1.

```
CREATE SCHEMA test1;
CREATE ROLE test_user LOGIN PASSWORD 'drowssap';
CREATE ROLE tester CREATEROLE LOGIN PASSWORD 'drowssap' IN ROLE test_user;
CREATE ROLE tester2 CREATEROLE LOGIN PASSWORD 'drowssap';
GRANT CREATE, USAGE ON SCHEMA test1 to tester;
SET SESSION AUTHORIZATION tester;

-- 1. Podpora DDL operácií
   -- a. Vytvorenie tabuľky s podporou transakčného času
         CREATE TABLE osoba(
            id serial PRIMARY KEY,
            meno varchar(50) NOT NULL
         ) AS TRANSACTIONTIME;
         -- overenie, či sa vytvorili všetky potrebné objekty
         SELECT relname FROM pg_class WHERE relname like 'osoba%';
         -- overenie, či sa vytvorili všetky potrebné stĺpce
         \d osoba
         \d osoba_hist

   -- b. Vymazanie tabuľky s podporou transakčného času
         -- pokus o zmazanie historickej tabuľky - má skončiť s chybou
         DROP TABLE osoba_hist;
         DROP TABLE osoba;
         -- overenie, či sa vymazali všetky potrebné objekty
         SELECT relname FROM pg_class WHERE relname like 'osoba%';

   -- c. Pridanie podpory transakčného času do existujúcej tabuľky
         CREATE TABLE osoba(
            id serial PRIMARY KEY,
            meno varchar(50) NOT NULL
         );
         ALTER TABLE osoba ADD TRANSACTIONTIME;
         -- overenie, či sa vytvorili všetky potrebné objekty
         SELECT relname FROM pg_class WHERE relname like 'osoba%';
         -- overenie či sa vytvorili všetky potrebné stĺpce
         \d osoba
         \d osoba_hist

   -- d. Zrušenie podpory transakčného času z tabuľky
         -- pokus o modifikáciu historickej tabuľky - má skončiť s chybou
         ALTER TABLE osoba_hist DROP TRANSACTIONTIME;
         ALTER TABLE osoba DROP TRANSACTIONTIME;
         -- overenie, či sa vymazali všetky potrebné objekty
         SELECT relname FROM pg_class WHERE relname like 'osoba%';
         \d osoba
         --vrátenie do pôvodného stavu
         ALTER TABLE osoba ADD TRANSACTIONTIME;

   -- e. Pridanie stĺpca do tabuľky s podporou transakčného času
         -- pokus o modifikáciu historickej tabuľky - má skončiť s chybou
         ALTER TABLE osoba_hist ADD COLUMN vyska integer UNIQUE NOT NULL DEFAULT 170;
         ALTER TABLE osoba ADD COLUMN vyska integer UNIQUE NOT NULL DEFAULT 170;
         -- overenie, či sa vytvorili všetky potrebné stĺpce
         \d osoba
         \d osoba_hist

   -- f. Odobratie stĺpca z tabuľky s podporou transakčného času
         -- pokus o modifikáciu historickej tabuľky - má skončiť s chybou
         ALTER TABLE osoba_hist DROP COLUMN vyska;
         ALTER TABLE osoba DROP COLUMN vyska;
         -- overenie či sa odobrali všetky potrebné stĺpce
         \d osoba
         \d osoba_hist

   -- g. Zmena stĺpca v tabuľke s podporou transakčného času
         -- zmena dátoveho typu
         -- pokus o modifikáciu historickej tabuľky - má skončiť s chybou
```

```
        ALTER TABLE osoba_hist ALTER COLUMN meno SET DATA TYPE varchar(51);
        ALTER TABLE osoba ALTER COLUMN meno SET DATA TYPE varchar(51);
        -- overenie, či sa vykonali potrebné zmeny
        \d osoba
        \d osoba_hist
        -- vrátenie do pôvodného stavu
        ALTER TABLE osoba ALTER COLUMN meno SET DATA TYPE varchar(50);

        -- pridanie východiskovej hodnoty
        -- pokus o modifikáciu historickej tabuľky - má skončiť s chybou
        ALTER TABLE osoba_hist ALTER COLUMN meno SET DEFAULT 'Miroslav';
        ALTER TABLE osoba ALTER COLUMN meno SET DEFAULT 'Miroslav';
        -- overenie, či sa vykonali potrebné zmeny
        \d osoba
        \d osoba_hist

        -- zrušenie východiskovej hodnoty
        -- pokus o modifikáciu historickej tabuľky - má skončiť s chybou
        ALTER TABLE osoba_hist ALTER COLUMN meno DROP DEFAULT;
        ALTER TABLE osoba ALTER COLUMN meno DROP DEFAULT;
        -- overenie či sa vykonali potrebné zmeny
        \d osoba
        \d osoba_hist

        -- zapnutie NOT NULL obmedzenia
        -- pokus o modifikáciu historickej tabuľky - má skončiť s chybou
        ALTER TABLE osoba_hist ALTER COLUMN meno SET NOT NULL;
        ALTER TABLE osoba ALTER COLUMN meno SET NOT NULL;
        -- overenie, či sa vykonali potrebné zmeny
        \d osoba
        \d osoba_hist

        -- vypnutie NOT NULL obmedzenia
        -- pokus o modifikáciu historickej tabuľky - má skončiť s chybou
        ALTER TABLE osoba_hist ALTER COLUMN meno DROP NOT NULL;
        ALTER TABLE osoba ALTER COLUMN meno DROP NOT NULL;
        -- overenie, či sa vykonali potrebné zmeny
        \d osoba
        \d osoba_hist

        -- premenovanie
        ALTER TABLE osoba_hist RENAME COLUMN meno TO priezvisko;
        ALTER TABLE osoba RENAME COLUMN meno TO priezvisko;
        -- overenie, či sa správne premenovali stĺpce
        \d osoba
        \d osoba_hist
        -- vrátenie do pôvodného stavu
        ALTER TABLE osoba RENAME COLUMN priezvisko TO meno;

    -- h. Vytvorenie tabuľky dediacej od tabuľky s podporou transakčného času
        -- pokus o dedenie od historickej tabuľky - má skončiť s chybou
        CREATE TABLE zamestnanec(
           mzda real
        ) inherits (osoba_hist);
        CREATE TABLE zamestnanec(
           mzda real
        ) inherits (osoba);
        -- overenie či sa vytvorili všetky potrebné objekty
        SELECT relname FROM pg_class WHERE relname like 'zamestnanec%';
        -- overenie, či sa vytvorili všetky potrebné stĺpce
        \d zamestnanec
        \d zamestnanec_hist
        -- vrátenie do pôvodného stavu
        DROP TABLE zamestnanec;

    -- i. Pridanie dedenia od tabuľky s podporou transakčného času do štandardnej tabuľky
        CREATE TABLE zamestnanec(
           id integer NOT NULL,
           meno varchar(50) NOT NULL,
           mzda real
        );
        -- pokus o dedenie od historickej tabuľky - má skončiť s chybou
        ALTER TABLE zamestnanec INHERIT osoba_hist;
        ALTER TABLE zamestnanec INHERIT osoba;
        -- overenie či sa vytvorili všetky potrebné objekty
        SELECT relname FROM pg_class WHERE relname like 'zamestnanec%';
        -- overenie, či sa vytvorili všetky potrebné stĺpce
        \d zamestnanec
        \d zamestnanec_hist
        -- vrátenie do pôvodného stavu
        DROP TABLE zamestnanec;
```

```
   -- j. Pridanie dedenia od tabuľky s podporou transakčného času do tabuľky s podporou transakčného
času
         CREATE TABLE zamestnanec(
            id integer NOT NULL,
            meno varchar(50) NOT NULL,
            mzda real
         ) AS TRANSACTIONTIME;
         -- pokus o dedenie od historickej tabuľky - má skončiť s chybou
         ALTER TABLE zamestnanec INHERIT osoba_hist;
         ALTER TABLE zamestnanec INHERIT osoba;
         -- overenie, či sa vytvorili všetky potrebné objekty
         SELECT relname FROM pg_class WHERE relname like 'zamestnanec%';
         -- overenie, či sa správne vytvorili vzťahy dedenia
         \d zamestnanec
         \d zamestnanec_hist

   -- k. Zrušenie dedenia na TT tabuľke
         -- pokus o zrušenie dedenia na historickej tabuľke - má skončiť s chybou
         ALTER TABLE zamestnanec_hist NO INHERIT osoba_hist;
         ALTER TABLE zamestnanec NO INHERIT osoba;
          -- overenie, či sa správne zrušili vzťahy dedenia
         \d zamestnanec
         \d zamestnanec_hist
         -- vrátenie do pôvodného stavu
         DROP TABLE zamestnanec;

   -- l. Pridanie indexu nad stĺpec z tabuľky s podporou transakčného času
         -- pokus o pridenie indexu nad stĺpec historickej tabuľky - má skončiť s chybou
         CREATE UNIQUE INDEX osoba_meno_index ON osoba_hist(meno);
         CREATE UNIQUE INDEX osoba_meno_index ON osoba(meno);
         -- overenie, či sa správne vytvorili indexy
         \d osoba
         \d osoba_hist

   -- m. Zmena ukladacích parametrov indexu TT tabuľky
         -- pokus o zmenu indexu historickej tabuľky - má skončiť s chybou
         ALTER INDEX osoba_hist_meno_idx SET (fillfactor = 20);
         ALTER INDEX osoba_meno_index SET (fillfactor = 20);
         ALTER INDEX osoba_meno_index RESET (fillfactor);

   -- n. Zrušenie indexu nad stĺpcom z tabuľky s podporou transakčného času
         -- pokus o zrušenie indexu historickej tabuľky - má skončiť s chybou
         DROP INDEX osoba_hist_meno_idx;
         DROP INDEX osoba_meno_index;
         -- overenie, či sa správne zrušili indexy
         \d osoba
         \d osoba_hist

   -- o. Zmena schémy tabuľky s podporou transakčného času
         -- pokus o zmenu schémy historickej tabuľky - má skončiť s chybou
         ALTER TABLE osoba_hist SET SCHEMA test1;
         ALTER TABLE osoba SET SCHEMA test1;
         -- overenie, či sa správne zmenila schéma
         SELECT c.relname,
                n.nspname
            FROM pg_class c
                JOIN pg_namespace n
                    ON c.relnamespace = n.oid
            WHERE relname LIKE 'osoba%';
         -- vrátenie do pôvodného stavu
         ALTER TABLE test1.osoba SET SCHEMA public;

   -- p. Nastavenie úložných parametrov TT tabuľky
         -- pokus o nastavenie úložných parametrov historickej tabuľky - má skončiť s chybou
         ALTER TABLE osoba_hist SET (fillfactor = 20);
         ALTER TABLE osoba SET (fillfactor = 20);
         ALTER TABLE osoba RESET (fillfactor);

   -- q. Nastavenie indexu na zhlukovanie TT tabuľky
         CREATE INDEX osoba_meno_index ON osoba(meno);
         -- pokus o nastavenie zhlukovacieho indexu na historickej tabuľke - má skončiť s chybou
         ALTER TABLE osoba_hist CLUSTER ON osoba_meno_index;
         ALTER TABLE osoba CLUSTER ON osoba_meno_index;
         -- overenie, či sa zhlukovanie nastavilo správne
         SELECT c.relname, i.indisclustered
            FROM pg_class c
                JOIN pg_index i
                    ON c.oid = i.indexrelid
            WHERE c.relname like 'osoba%'
         ;

   -- r. Zrušenie zhlukovania TT tabuľky
```

```
        -- pokus o zrušenie zhlukovania na historickej tabuľke - má skončiť s chybou
        ALTER TABLE osoba_hist SET WITHOUT CLUSTER;
        ALTER TABLE osoba SET WITHOUT CLUSTER;
        -- overenie, či sa zhlukovanie nastavilo správne
        SELECT c.relname, i.indisclustered
           FROM pg_class c
               JOIN pg_index i
                   ON c.oid = i.indexrelid
           WHERE c.relname like 'osoba%'
        ;
        -- vrátenie do pôvodného stavu
        DROP INDEX osoba_meno_index;

   -- s. Zmena vlastníka TT tabuľky
        -- pokus o zmenu vlastníka historickej tabuľky - má skončiť s chybou
        ALTER TABLE osoba_hist OWNER TO test_user;
        ALTER TABLE osoba OWNER TO test_user;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist
        --vrátenie do pôvodného stavu
        ALTER TABLE osoba OWNER TO tester;

   -- t. Zapnutie identifikátora riadkov TT tabuľky
        -- pokus o zapnutie identifikátora riadkov na historickej tabuľke - má skončiť s chybou
        ALTER TABLE osoba_hist SET WITH OIDS;
        ALTER TABLE osoba SET WITH OIDS;
        -- overenie zapnutia identifikátorov
        SELECT relhasoids FROM pg_class WHERE relname = 'osoba';
        SELECT relhasoids FROM pg_class WHERE relname = 'osoba_hist';

   -- u. Vypnutie identifikátora riadkov TT tabuľky
        -- pokus o vypnutie identifikátora riadkov na historickej tabuľke - má skončiť s chybou
        ALTER TABLE osoba_hist SET WITHOUT OIDS;
        ALTER TABLE osoba SET WITHOUT OIDS;
        -- overenie vypnutia identifikátorov
        SELECT relhasoids FROM pg_class WHERE relname = 'osoba';
        SELECT relhasoids FROM pg_class WHERE relname = 'osoba_hist';

-- 2. Podpora DML operácií
   -- a. Vloženie záznamu do tabuľky s podporou transakčného času
        -- pokus o vloženie hodnoty do metadátového stĺpca - má skončiť s chybou
        INSERT INTO osoba(id, meno, _entry_id) VALUES(0, 'miro', 1);
        -- pokus o vloženie záznamu do historickej tabuľky - má skončiť s chybou
        INSERT INTO osoba_hist VALUES(0, 'miro');
        INSERT INTO osoba VALUES(0, 'miro');
        -- overenie vloženia záznamu
        SELECT * FROM osoba;
        SELECT *, _entry_id, _sys_start, _sys_end FROM osoba;
        SELECT * FROM osoba_hist;

   -- b. Upravenie záznamu v tabuľke s podporou transakčného času
        -- pokus o zmenu hodnoty v metadátovom stĺpci - má skončiť s chybou
        UPDATE osoba SET _entry_id = 0;
        -- pokus o modifikáciu záznamu v historickej tabuľke - má skončiť s chybou
        UPDATE osoba_hist SET meno = 'Miroslav';
        UPDATE osoba SET meno = 'Miroslav';
        -- overenie upravenia záznamu
        SELECT * FROM osoba;
        SELECT *, _entry_id, _sys_start, _sys_end FROM osoba;
        SELECT * FROM osoba_hist;

   -- c. Vymazania záznamu z tabuľky s podporou transakčného času
        -- pokus o vymazanie záznamu z historickej tabuľky - má skončiť s chybou
        DELETE FROM osoba_hist;
        DELETE FROM osoba;
        -- overenie vymazania záznamu
        SELECT * FROM osoba;
        SELECT *, _entry_id, _sys_start, _sys_end FROM osoba;
        SELECT * FROM osoba_hist;
   -- d. Vyprázdnenie tabuľky s podporou transakčného času
        INSERT INTO osoba(meno) VALUES('fero');
        INSERT INTO osoba(meno) VALUES('jano');
        -- pokus o vyprázdnenie historickej tabuľky - má skončiť s chybou
        TRUNCATE osoba_hist;
        TRUNCATE osoba;
        -- overenie vyprázdnenia tabuľky
        SELECT * FROM osoba;
        SELECT *, _entry_id, _sys_start, _sys_end FROM osoba;
        SELECT * FROM osoba_hist;
-- 3. Podpora dotazov
   -- a. Dotazy nad aktuálnymi verziami záznamov
```

```
        INSERT INTO osoba(meno) VALUES('zuzana');
        INSERT INTO osoba(meno) VALUES('jozo');
        SELECT * FROM osoba;
        SELECT *, _entry_id, _sys_start, _sys_end FROM osoba;
        SET history_timestamp to '1999-12-12 12:12:12.123456';
        CURRENT TRANSACTIONTIME SELECT * FROM osoba;
        CURRENT TRANSACTIONTIME SELECT *, _entry_id, _sys_start, _sys_end FROM osoba;
        RESET history_timestamp;

   -- b. Dotazy nad aktuálnymi aj historickými verziami záznamov
        NONSEQUENCED TRANSACTIONTIME SELECT * FROM osoba;
        NONSEQUENCED TRANSACTIONTIME SELECT *, _entry_id, _sys_start, _sys_end FROM osoba;

   -- c. Dotazy nad záznamami platnými vo zvolenom čase
        TRANSACTIONTIME AS OF (clock_timestamp() - interval '1000 microsecond') SELECT * FROM
osoba;
        TRANSACTIONTIME AS OF (clock_timestamp() - interval '1000 microsecond') SELECT *,
_entry_id, _sys_start, _sys_end FROM osoba;
        SET history_timestamp to '2020-12-12 12:12:12.123456';
        SELECT * FROM osoba;
        SELECT *, _entry_id, _sys_start, _sys_end FROM osoba;
        RESET history_timestamp;

-- 4. Podpora údržbových operácií
   -- a. Zhluknutie TT tabuľky podľa indexu
        CREATE INDEX osoba_meno_index ON osoba(meno);
        -- pokus o zhluknutie historickej tabuľky - má skončiť s chybou
        CLUSTER osoba_hist USING osoba_hist_meno_idx;
        CLUSTER osoba USING osoba_meno_index;
        -- overenie, či sa zhlukovanie nastavilo správne
        SELECT c.relname, i.indisclustered
           FROM pg_class c
               JOIN pg_index i
                   ON c.oid = i.indexrelid
           WHERE c.relname like 'osoba%'
        ;
        -- vrátenie do pôvodného stavu
        ALTER TABLE osoba SET WITHOUT CLUSTER;
        DROP INDEX osoba_meno_index;

-- 5. Podpora nastavení používateľských oprávnení
   -- a. Pridelenie SELECT oprávnení nad TT tabuľkou
        -- pokus o pridelenie oprávnení na historickej tabuľke - má skončiť s chybou
        GRANT SELECT(meno) ON osoba_hist TO tester2;
        GRANT SELECT(meno) ON osoba TO tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist
        -- pokus o pridelenie oprávnení na historickej tabuľke - má skončiť s chybou
        GRANT SELECT ON osoba_hist TO tester2;
        GRANT SELECT ON osoba TO tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist

   -- b. Odobratie SELECT oprávnení nad TT tabuľkou
        -- pokus o odobratie oprávnení na historickej tabuľke - má skončiť s chybou
        REVOKE SELECT(meno) ON osoba_hist FROM tester2;
        REVOKE SELECT(meno) ON osoba FROM tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist
        -- pokus o odobratie oprávnení na historickej tabuľke - má skončiť s chybou
        REVOKE SELECT ON osoba_hist FROM tester2;
        REVOKE SELECT ON osoba FROM tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist

   -- c. Pridelenie INSERT oprávnení nad TT tabuľkou
        -- pokus o pridelenie oprávnení na historickej tabuľke - má skončiť s chybou
        GRANT INSERT(meno) ON osoba_hist TO tester2;
        GRANT INSERT(meno) ON osoba TO tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba__entry_id_seq
        -- pokus o pridelenie oprávnení na historickej tabuľke - má skončiť s chybou
        GRANT INSERT ON osoba_hist TO tester2;
        GRANT INSERT ON osoba TO tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba__entry_id_seq
```

```
  -- d. Odobratie INSERT oprávnení nad TT tabuľkou
        -- pokus o odobratie oprávnení na historickej tabuľke - má skončiť s chybou
        REVOKE INSERT(meno) ON osoba_hist FROM tester2;
        REVOKE INSERT(meno) ON osoba FROM tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba__entry_id_seq
        -- pokus o odobratie oprávnení na historickej tabuľke - má skončiť s chybou
        REVOKE INSERT ON osoba_hist FROM tester2;
        REVOKE INSERT ON osoba FROM tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba__entry_id_seq

  -- e. Pridelenie UPDATE oprávnení nad TT tabuľkou
        -- pokus o pridelenie oprávnení na historickej tabuľke - má skončiť s chybou
        GRANT UPDATE(meno) ON osoba_hist TO tester2;
        GRANT UPDATE(meno) ON osoba TO tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist
        -- pokus o pridelenie oprávnení na historickej tabuľke - má skončiť s chybou
        GRANT UPDATE ON osoba_hist TO tester2;
        GRANT UPDATE ON osoba TO tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist

  -- f. Odobratie UPDATE oprávnení nad TT tabuľkou
        -- pokus o odobratie oprávnení na historickej tabuľke - má skončiť s chybou
        REVOKE UPDATE(meno) ON osoba_hist FROM tester2;
        REVOKE UPDATE(meno) ON osoba FROM tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist
        -- pokus o odobratie oprávnení na historickej tabuľke - má skončiť s chybou
        REVOKE UPDATE ON osoba_hist FROM tester2;
        REVOKE UPDATE ON osoba FROM tester2;
        -- overenie zmeny vlastníka
        \dp osoba
        \dp osoba_hist
        --vrátenie do pôvodného stavu
        DROP TABLE osoba;

RESET SESSION AUTHORIZATION;
DROP SCHEMA test1;
DROP ROLE tester;
DROP ROLE tester2;
DROP role test_user;
```

Príloha C: Technická dokumentácia k prevádzke

C.1 Inštalácia systému

V tejto časti sa venujeme inštalácii PostgreSQL zo zdrojových kódov na operačný systém Windows použitím balíka nástrojov MinGW [33]. Pre inštaláciu na iný operačný systém je potrebné postupovať podľa návodu v dokumentácii PostgreSQL [17].

Nasledujúci postup sa realizuje v MinGW shell, ktorý po stiahnutí a nainštalovaní MinGW nájdeme v Start->MinGW->MinGW shell. Zdrojové kódy PostgreSQL je potrebné nakopírovať do adresára msys/1.0 (alebo v jeho ľuvovoľných podadresároch), ktorý sa nachádza v inštalačnom adresári MinGW. V našom prípade bola pri inštalácii cesta k zdrojovým kódom vo windowse "C:\MinGW\msys\1.0\home\Miro\sources" čo v MinGW shell predstavuje cestu "/home/Miro/sources". Výsledkom postupu je databázový server nainštalovaný v "C:\MinGW\msys\1.0\local\psql" ("/local/psql") s vytvorenou databázou test.

```
cd /home/Miro/sources/postgresql-9.0.4-temporal
./configure --without-zlib
make
make install
cd /local/psql
mkdir data
cp ./lib/* ./bin
cd bin
initdb --pgdata=/local/psql/data --encoding=UTF-8 --locale=Slovak_Slovakia --
     username=postgres -pwprompt
pg_ctl -D /local/psql/data start
createdb  --username=postgres --password --encoding=UTF-8 --
     locale=Slovak_Slovakia  --owner=postgres test
psql --username=postgres --dbname=test --password
```

C.2 Používateľská príručka

V rámci používateľskej príručky opisujeme len použitie príkazov, ktoré neboli súčasťou PostgreSQL 9.0.4 a slúžia na prácu s temporálnym rozšírením. Informácie k použitiu pôvodných príkazov sú dostupné v dokumentácii PostgreSQL [17].

Vytvorenie temporálnej tabuľky

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name ( [
  { column_name data_type [ DEFAULT default_expr ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE parent_table [ like_option ... ] }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
[ AS TRANSACTIONTIME ]
```

Na vytvorenie tabuľky s podporou transakčného času je potrebné použiť príkaz CREATE TABLE s klauzulou AS TRANSACTIONTIME. Napríklad:

```
CREATE TABLE osoba(meno varchar(50)) AS TRANSACTIONTIME;
```

Pridanie/zrušenie podpory transakčného času

```
ALTER TABLE [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column TO new_column
ALTER TABLE name
    RENAME TO new_name
ALTER TABLE name
    SET SCHEMA new_schema

where action is one of:

    ...
    ADD TRANSACTIONTIME
    DROP TRANSACTIONTIME
```

Na pridanie/zrušenie podpory transakčného času je potrebné použiť príkaz ALTER TABLE s akciou ADD/DROP TRANSACTIONTIME. Napríklad:

```
ALTER TABLE osoba ADD TRANSACTIONTIME;
ALTER TABLE osoba DROP TRANSACTIONTIME;
```

Temporálne dotazy nad tabuľkami s podporou transakčného času

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
[ CURRENT TRANSACTIONTIME | NONSEQUENCED TRANSACTIONTIME | TRANSACTIONTIME AS OF
     expression ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ [ AS ] output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST
     } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [...] ]

...
```

Správanie jednotlivých temporálnych volieb je nasledovné:

- CURRENT TRANSACTIONTIME – príkaz SELECT pristupuje len k aktuálnymi verziám záznamov
- NONSEQUENCED TRANSACTIONTIME – príkaz SELECT pristupuje ku všetkým verziám záznamov
- TRANSACTIONTIME AS OF – príkaz SELECT pristupuje len k verziám platným v zadanom čase

Príklad:

```
CURRENT TRANSACTIONTIME SELECT * FROM osoba;
NONSEQUENCED TRANSACTIONTIME SELECT * FROM osoba;
TRANSACTIONTIME AS OF '2012-12-12 12:32:45.123456' SELECT * FROM osoba;
```

Konfiguračné parametre

V temporálnom rozšírení PostgreSQL sme zaviedli nasledovné konfiguračné parametre:

- history_timestamp – na úrovni pripojenia nastavuje čas, ku ktorému majú príkazy SELECT vyberať dáta ak nie je uvdená žiadna temporálna voľba. Východiskovú hodnotu má „current“ a možno ho nastaviť len v rámci pripojenia.
- automatic_transactiontime – ak je nastavený na hodnotu „on“, všetky vytvárané tabuľky budú mať automaticky podporou transakčného času (bez nutnosti uvádzať

AS TRANSACTIONTIME). Východiskovú hodnotu má „off“ a možno ho nastaviť v rámci pripojenia alebo pre celú databázu v konfiguračnom súbore postgresq.conf.

- forbid_modifications_when_in_history - ak je nastavený na hodnotu „on“, a súčasne je parameter history_timestamp nastavený na nejaký čas, systém zakáže vykonávať akékoľvek modifikujúce operácie nad tabuľkami s podporou transakčného času. Východiskovú hodnotu má „on“ a možno ho nastaviť v rámci pripojenia alebo pre celú databázu v konfiguračnom súbore postgresq.conf.