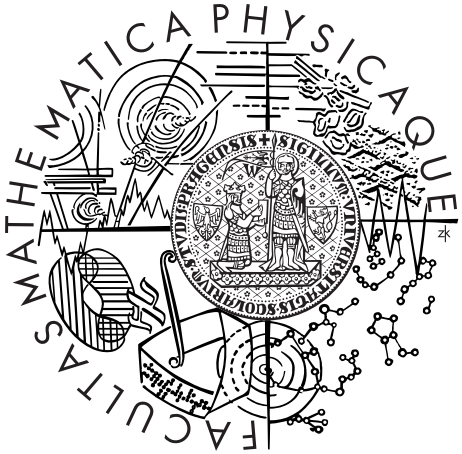
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCA



bc. Tomáš Mikuš

Aktualizace XML dat

Katedra softwarového inženýrství

Vedúci diplomovej práce:	prof. RNDr. Jaroslav Pokorný, CSc.
Študijný program:	Informatika
Študijný obor:	Softwarové systémy

Praha 2011

Na tomto mieste by som chcel poďakovať vedúcemu diplomovej práce prof. RNDr. Jaroslavovi Pokornému, CSc., ktorý mi s diplomovou prácou pomohol. Ďakujem.

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne a výhradne s použitím citovaných prameňov, literatúry a ďalších odborných zdrojov.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona v platnom znení, najmä skutočnosť, že Univerzita Karlova v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa §60 odst. 1 autorského zákona.

V Prahe dňa 9.12.2011 Podpis autora

Názov práce: Aktualizácia XML dát

Autor: bc. Tomáš Mikuš

Katedra: Katedra softwarového inženýrství
Vedúci diplomovej práce: prof. RNDr. Jaroslav Pokorný, CSc.

Abstrakt: Aktualizácia XML dát je veľmi široká oblasť, ktorá musí riešiť niekoľko neľahkých problémov. Od návrhu jazyka s dostatočnou vyjadrovacou silou až po úložisko XML dát schopné zmeny aplikovať. Spôsobov ako sa s nimi vysporiadať je niekoľko. Z tohoto pohľadu je práca veľmi úzko špecializovaná len na jazyk XQuery. Teda jeho rozšírenie pre aktualizácie, pre ktoré bolo doporučenie, konzorciom W3C, zverejnené len nedávno. Ďalšou špecializáciou tejto práce je zameranie sa len na XML dáta uložené v objektovo-relačnej databáze, s tým, že úložisko bude vynucovať validitu dokumentov voči schéme popísanej v jazyku XML Schema. Táto požiadavka v kombinácií s možnosťou dáta v úložisku aktualizovať, kladie na úložisko protichodné požiadavky. V práci je navrhnutý jazyk, ktorý vychádza z jazyka XQuery, navrhnuté a implementované vyhodnocovanie aktualizačných požiadaviek tohoto jazyka nad úložiskom a popis a implementácia samotného úložiska v objektovo-relačnej databáze.

Kľúčové slová: aktualizácia XML, XQuery, objektovo-relačná databáza, XML Schema

Title: Updating XML data

Author: bc. Tomáš Mikuš

Department: Department of Software Engineering
Supervisor: prof. RNDr. Jaroslav Pokorný, CSc.

Abstract: Updating XML data is very wide area, which must solve a number of difficult problems. From designing language with sufficient expressive power to the XML data repository able to apply the changes. Ways to deal with them are few. From this perspective, is this work very closely dedicated only to the language XQuery. Thus, its extension for updates, for which the candidate recommendation by the W3C were published only recently. Another specialization of this work is to focus only on the XML data stored in the object-relational database with that repository will enforce the validity of documents to the scheme described in XML Schema. This requirement, combined with the possibility of updating of data in the repository is on the contradictory requirements. In this thesis is designed language based on XQuery language, designed and implemented evaluating of the update queries of the language on the store and a description and implementation of the store in object-relational database.

Keywords: updating XML, XQuery, object-relational database, XML Schema

Obsah

Úvod

XML[1] je značkovací jazyk, ktorý bol vyvinutý konzorciom W3C[2]. Jeho štandard je vo forme doporučenia možné nájsť v [13]. Pri jeho návrhu boli zohľadnené skúsenosti s predchádzajúcimi značkovacími jazykmi a predstavený bol v roku 1998. XML bolo primárne využívané pre svoju schopnosť popísať akékoľvek údaje na výmenu informácií. S narastajúcim objemom vymieňaných údajov v XML bolo potrebné riešiť ich efektívne ukladanie a vyhľadávanie v nich.

Začali sa ale objavovať aj požiadavky na aktualizáciu XML dát. Bolo potrebné upraviť XML v trvalom úložisku ako napríklad natívna XML databáza, XML súbor uložený na disku alebo XML uložené v SQL databáze. Bolo potrebné upraviť stav XML správy a pridať k nej informácie vytvorené počas jej spracovania. Prípadne vykonať zmeny nad konfiguračnými súbormi, užívateľskými profilmi alebo inými záznamami reprezentovanými v XML. Pri snahe o aktualizáciu XML dát, je potrebné vyriešiť niekoľko neľahkých problémov. Od návrhu jazyka s dostatočnou vyjadrovacou silou až po úložisko XML dát schopné zmeny aplikovať.

Okrem XML sa v práci stretneme aj s jazykom XML Schema. Tiež je definovaný štandardom od konzorcia W3C, ktorý je možné nájsť v dokumente [16]. Ide o jazyk, ktorý je určený na popis štruktúry XML dokumentu a popis prípustného obsahu XML dokumentu. Veľmi pekne je tento jazyk popísaný v [9].

Ďalším jazykom od konzorcia W3C, ktorého základy je nutné poznať pre ďalšie čítanie tejto práce je XQuery. XQuery je jazyk pre dotazovanie dát z XML dokumentov. Jeho rozsiahly štandard je dostupný v dokumente [14]. S jazykom XQuery je možné sa bližšie zoznámiť napríklad v [9].

V práci je popísané rozšírenie jazyka XQuery zamerané na aktualizácie XML dát. Ide o XQuery Update Facility 1.0, ktorého štandard v [15] sa stal doporučením konzorcia W3C len nedávno. Krátky a praktický popis rozšírenia je obsiahnutý v dokumente [10].

Cieľom tejto práce je oboznámiť sa so štandardom XQuery Update Facility 1.0. Ďalšími cieľmi tejto práce sú návrh a implementácia algoritmov pre aktualizáciu XML dokumentov uložených v objektovo-relačnej databáze. Jazyk aktualizačných požiadaviek má vychádzať z jazyka XQuery Update Faciliy 1.0. Jazyk by mal mať nie len podobnú syntax ale aj sémantiku. Úložisko má podporovať uloženie dokumentov so schémou a má vynucovať validitu dokumentov po každej aktualizačnej operácií. Riešenie bude obsahovať:

- Návrh mapovania pre schému XML dokumentu popísanú v jazyku XML Schema na objektovo-relačnú schému.
- Algoritmus vyhodnotenia aktualizačnej požiadavky jazyka, ktorý vychádza z jazyka XQuery Update Facility 1.0. Tento algoritmus sa skladá z dvoch častí:
  - algoritmu pre mapovanie požiadavky na výrazy `SELECT`, ktorými sa z úložiska získajú vstupné uzly pre základné aktualizačné výrazy

[1] Extensible Markup Language - doslovný preklad je rozšíriteľný značkovací jazyk.

[2] The World Wide Web Consortium - medzinárodná komunita pracujúca na vývoji webových štandartov. Viac informácií je možné nájsť na http://www.w3.org/.

    - algoritmu pre aplikovanie zmien reprezentovaných základnými aktualizačnými výrazmi a vstupnými uzlami do úložiska

Práca je rozdelená do piatich kapitol:

- Prvá kapitola sa venuje XQuery Update Facility 1.0. Sú v nej popísané nové typy výrazov a postup vyhodnotenia, ktoré do XQuery toto rozšírenie prináša.
- V druhej kapitole sú zhrnuté ciele, o ktoré sa v riešení snažíme a porovnané z už existujúcim riešením.
- V tretej kapitole je popísané mapovanie jazyka XML Schema do objektovo-relačnej databázy, spolu s algoritmami na vytvorenie štrutkúry a na uloženie XML dokumentu do takto pripravej štruktúry.
- V štvrtej kapitole je navrhnutá gramatika jazyka pre aktualizačné požiadavky. Je tam popísané mapovanie časti aktualizačnej požiadavky na požiadavku `select` v SQL a nakoniec popis aplikácie zmien reprezentovaných aktualizačnou požiadavkou do podkladovej štruktúry.
- Piata kapitola obsahuje stručný popis implementácie, vrátane uživateľského prostredia.

V práci bolo navrhnuté mapovanie jazyka XML Schema do objektovo-relačnej databázy, tak aby vytvorená štruktúra v databáze umožňovala aktuali zácie XML dokumentov v nej uložených. Toto mapovanie tiež mapuje podmnožinu integritných obmedzení jazyka XML Schema na prostriedky objektovo-relačnej databázy, ktorými databáza vynucuje validitu XML dokumentu po každej aktualizačnej požiadavke. V práci bol navrhnutý jazyk aktualizačnej požiadavky, ktorý vychádza z jazyka pre XQuery Update Facility 1.0. Bolo navrhnuté mapovanie časti tejto požiadavky na požiadavku `select` v jazyku SQL. Takto vytvorenou požiadavkou `select` sú z podkladovej štruktúry získané vstupné hodnoty pre aktualizačné operácie. V práci bol navrhnutý postup aplikovania zmien reprezentovaných aktualizačnou požiadavkou do podkladovej štruktúry. Súčasťou práce je aj implementácia navrhnutých postupov. Implementácia umožňuje:

- Vytvorenie podkladovej štrutúry pre schému XML dokumentu v databáze Oracle 10g Release 2 (ďalej Oracle 10.2).
- Uloženie XML dokumentu do vytvorenej štruktúry.
- Vykonanie aktualizačných požiadaviek nad dokumentami uloženými v štruktúre.
- Zobrazenie dokumentu uloženého v podkladovej štruktúre.

1. Aktualizačné prostriedky pre XQuery 1.0

V pracovnej verzii požiadaviek na XML Query, zverejnenej 31. januára 2000 konzorciom W3C, sa objavila len jedna veta zaoberajúca sa aktualizáciami. Hovorí, že verzia 1.0 jazyka XML Query nesmie dopredu vylučovať možnosť pridať do ďalších verzií jazyka aktualizačné prostriedky. Neskôr sa tento jazyk premenoval na XQuery no pracovná verzia požiadaviek na aktualizačné prostriedky bola zverejnená až o päť rokov neskôr.

Cieľom bolo do budúcej verzie jazyka XQuery pridať možnosť pridávať nové hodnoty a meniť existujúce hodnoty v XML dokumentoch.

Po spísaní požiadaviek na aktualizačné prostriedky, ktoré by mali byť do jazyka XQuery doplnené, bola 27. januára 2006 zverejnená prvá pracovná verzia popisujúca aktualizačné prostriedky v XQuery. 14. marca 2008 bol tento dokument dopracovaný do podoby kandidáta na doporučenie (candidate recommendation) a 17. marca 2011 bolo zverejnené doporučenie konzorcia W3C pre aktualizačné prostriedky XQuery.

V tejto kapitole popíšeme aktualizačné prostriedky pre XQuery 1.0 tak ako sú popísané v doporučení konzorcia W3C. Zameriame sa najmä na syntax a sémantiku jazyka, s poukázaním, na niektoré zaujímavé vlastnosti jazyka.

1.1 Rozšírenie XQuery 1.0

Základným stavebným blokom XQuery 1.0 je *výraz*. XQuery obsahuje niekoľko druhov výrazov, ktoré môžu byť rôznymi spôsobmi kombinované. *XDM inštancia* je postupnosť, ktorá obsahuje žiadny alebo viacero uzlov a/alebo atomických hodnôt tak ako sú definované v dátovom modele XQuery. Vstupom pre výraz je žiadna alebo viacero XDM inštancií a výstupom je XDM inštancia. Výraz nijakým spôsobom neupravuje uzly vo vstupnej inštancii. Výstupná XDM inštancia obsahuje novovytvorené uzly s novými identitami uzlov. *Identita uzla* je vlastnosťou každého uzla v XDM inštancii a uzol je ňou jednoznačne identifikovaný.

Aktualizačné prostriedky pre XQuery 1.0 pridávajú novú kategóriu aktualizačných výrazov, ktoré umožňujú pridávať nové uzly a meniť stav existujúcich uzlov. Umožňujú na XDM inštancii vykonať nasledujúce operácie:

- vloženie uzla
- zmazanie uzla
- zmenu niektorej z vlasntností uzla pri zachovaní jeho identity
- vytvorenie upravenej kópie uzla s novou identitou uzla

1.1.1 Rozšírenie modelu spracovania

V XQuery 1.0 je výsledkom každého výrazu XDM inštancia. Aktualizačné prostriedky pre XQuery 1.0 rozširujú model spracovania XQuery tak, že výsledkom

každého výrazu je XDM inštancia a zoznam čakajúcich aktualizácií. *Zoznam čakajúcich aktualizácií* (`pending update list`) je neusporiadaná kolekcia aktualizačných primitív, ktoré reprezentujú zmeny stavov uzlov, ktoré zatial neboli aplikované. Aspoň jedna časť výsledku (XDM inštancia alebo zoznam čakajúcich aktualizácií) je vždy prázdna.

Aktualizačné operácie (`update operations`) slúžia na popísanie sémantiky aktualizačných prostriedkov pre XQuery 1.0 a delia sa na dve skupiny:

- *Aktualizačné primitívum* (`update primitive`) je položkou zoznamu čakajúcich aktualizácií a reprezentuje zmenu stavu uzla, ktorá zatial nebola aplikovaná na XDM inštanciu.
- *Aktualizačná rutina* (`update routine`) reprezentuje postupnosť akcií a slúži na popísanie sémantiky aktualizácií v XQuery. Nevyskytuje sa na zozname čakajúcich aktualizácií.

Cieľový uzol (`target node`) je prvý argument aktualizačných primitív a určuje uzol, ktorý má byť aktualizačným primitívom zmenený.

Ak výsledkom výrazu v XQuery aktualizačnej požiadavke je zoznam aktualizačných primitív, je zavolané `upd:applyUpdates`, ktoré aplikuje zmeny reprezentované aktualizačnými primitívami na XDM inštanciu. Propagácia zmien do trvalého podkladového úložiska XML dokumentov nie je súčasťou špecifikácie aktualizačných prostriedkov pre XQuery 1.0, pri implemntácií je možné zvoliť rôzne riešenia, čím je ponechaný veľký priestor pre optimalizácie.

Snapshot vymedzuje rozsah, v rámci ktorého sú výrazy vyhodnotené na nemennej XDM inštancii a všetky zmeny sa ukladajú do zoznamu čakajúcich aktualizácií. Jedna aktualizačná požiadavka XQuery 1.0 je vyhodnotená v rámci jedného snapshotu. Ten je ukončený zavolaním rutiny `upd:applyUpdates`.

Aktualizačné primitíva

- `upd:insertBefore`
- `upd:insertAfter`
- `upd:insertInto`
- `upd:insertIntoAsFirst`
- `upd:insertIntoAsLast`
- `upd:insertAttributes`
- `upd:delete`
- `upd:replaceNode`
- `upd:replaceValue`
- `upd:replaceElementContent`
- `upd:rename`
- `upd:put`

Aktualizačné rutiny

upd:mergeUpdates

```
upd:mergeUpdates(
  $pul1 as pending-update-list,
  $pul2 as pending-update-list)
```

Spojí dva vstupné zoznamy čakajúcich aktualizácií do jedného výstupného, ktorý obsahuje všetky aktualizačné primitíva zo vstupných zoznamov.

upd:applyUpdates

```
upd:applyUpdates(
  $pul as pending-update-list,
  $revalidation-mode as xs:string,
  $inherit-namespaces as xs:boolean)
```

Ukončí snapshot aplikovaním zmien reprezentovaných aktualizačnými primitívami v `$pul` na XDM inštanciu. Pre každý uzol, ktorý bol označený na revalidáciu, označme ho `$top`, zavolá `upd:revalidate($top, $revalidation-mode)`.

Aktualizácie reprezentované aktualizačnými primitívami v `$pul` sú aplikované v nasledujúcom poradí:

- `upd:insertInto`, `upd:insertAttributes`, `upd:replaceValue` a `upd:rename`
- `upd:insertBefore`, `upd:insertAfter`, `upd:insertIntoAsFirst` a `upd:insertIntoAsLast`
- `upd:replaceNode`
- `upd:replaceElementContent`
- `upd:delete`

Pri aplikovaní zmien reprezentovaných aktualizačnými primitívami v tomto poradí, je navodený dojem, že všetky zmeny, ktoré má aktualizačná požiadavka vykonať sú na XDM inštaciu aplikovné súčasne. Je to možné aj vďaka tomu, že uzly a hodnoty, ktoré sú vstupnými paramterami aktualizačných primitív (okrem cieľových uzlov) sú kópie uzlov z XDM inštancie. Tieto kópie boli vytvorené pri vyhodnotení výrazu.

upd:revalidate

```
upd:revalidate(
  $top as node(),
  $revalidation-mode as xs:string)
```

Validuje podstrom s koreňom v uzle `$top`, ktorý je uzlom typu dokument alebo element, voči schéme s cielom určiť typ pre zmenené uzly. Povolené hodnoty pre parameter `$revalidation-mode` sú hodnoty `strict`, `lax` a `skip`.

upd:removeType

```
upd:removeType(
  $N as node())
```

Táto rutina je volaná na uzly, ktorých meno alebo obsah boli zmenené. Vstupnému uzlu `$N`, ktorý je typu element alebo attribút, odstráni informáciu o jeho type. Rutina sa rekurzívne zavolá na rodiča uzla `$N` a predka uzla `$N`, ktorý už nemá rodiča, označí na revalidáciu.

upd:setToUntyped

```
upd:setToUntyped(
  $N as node())
```

Táto rutina je volaná na uzol, ktorý bol vložený do neotypovaného kontextu. Vstupnému uzlu `$N`, ktorý je typu element alebo atribút nastaví `xs:untyped`. Rutina je rekurzívne volaná na všetky deti a atribúty uzla `$N`.

upd:propagateNamespace

```
upd:propagateNamespace(
  $element as element(),
  $prefix as xs:NCName,
  $uri as xs:anyURI)
```

Pre každé dieťa vstupného elementu, označme ho `$child`, ktoré nemá naviazaný menný priestor pre `$prefix`, pridá uzlu `$child` väzbu menného priestoru (`$prefix`, `$uri`) a zavolá `upd:propagateNamespace($child, $prefix, $uri)`

1.1.2 Nové druhy výrazov

Aktualizačné prostriedky pre XQuery 1.0 pridávajú päť nových druhov výrazov, výrazy `insert`, `delete`, `replace`, `rename` a `transform`.

Základný aktualizačný výraz (`basic updating expression`) je definovaný ako výraz `insert`, `delete`, `replace`, `rename` alebo volanie aktualizačnej funkcie.

Aktualizačný výraz (`updating expression`) je základný aktualizačný výraz alebo akýkoľvek výraz (iný ako výraz `transform`), ktorý priamo obsahuje aktualizačný výraz.

Jednoduchý výraz (`simple expression`) je akýkoľvek XQuery výraz, ktorý nie je aktualizačným výrazom.

Časť upravenej gramatiky XQuery 1.0 po pridaní piatich nových výrazov:

```
ExprSingle ::= FLWORExpr
    | QuantifiedExpr
    | TypeswitchExpr
    | IfExpr
    | OrExpr
    | InsertExpr
    | DeleteExpr
    | RenameExpr
    | ReplaceExpr
    | TransformExpr
```

Insert

Výraz `insert` je základný aktualizačný výraz, ktorý kópie žiadnych alebo viacerých uzlov určených zdrojovým výrazom (`SourceExpr`) vloží na miesto zohľadňujúce cieľový uzol určený cieľovým výrazom (`TargetExpr`). Kľúčové slová `node` a `nodes` sú ľubovoľne zamenitelné, bez ohľadu na to koľko uzlov bude vložených. Syntax výrazu `insert`:

```
InsertExpr ::= "insert" ("node"|"nodes") SourceExpr
        InsertExprTargetChoice TargetExpr
InsertExprTargetChoice ::= (("as" ("first"|"last"))? "into")
    | "after"
    | "before"

TargetExpr ::= ExprSingle
SourceExpr ::= ExprSingle
```

Miesto vloženia je určené nasledovne:

- ak je zadané `before` (alebo `after`), vkladané uzly sa stanú predchádzajúcimi (alebo nasledujúcimi) súrodencami cieľového uzla
- ak je zadané `as first into` (alebo `as last into`), vkladané uzly sa stanú prvými (alebo poslednými) deťmi cieľového uzla
- ak je zadané `into` bez `as first` a bez `as last`, vkladané uzly sa stanú deťmi cieľového uzla

Pre výraz `insert` so zadaným `before`, `after`, `as first into` alebo `as last into` musí byť splnené:

- Ak je jedným výrazom `insert` vložených viacero uzlov, stanú sa susednými uzlami a ich poradie zo zdrojového výrazu zostáva zachované.
- Ak je viacerými výrazmi `insert` vložených viacero skupín uzlov, počas platnosti jedného snapshotu, musí platiť:
    - Uzly v rámci skupiny sa stanú susednými a ich poradie zostáva zachované, poradie skupín je však závislé od implementácie.
    - Medzi uzly vložené výrazom `insert before` (alebo `insert after`) a cieľový uzol možu byť vložené uzly len výrazom `insert before` (alebo `insert after`) so zhodným cieľovým uzlom.
    - Pred (alebo za) uzly vložené výrazom `insert as first into` (alebo `insert as last into`) možu byť vložené uzly len výrazom `insert as first into` (alebo `insert as last into`) so zhodným cieľovým uzlom.

Pre výraz `insert` so zadaným `into` (bez `as first` alebo `as last`) musí byť splnené:

- Ak je jedným výrazom `insert` vložených viacero uzlov, ich poradie zo zdrojového výrazu zostáva zachované, ale nemusia byť nutne vložené ako susedné uzly.

- Zvolenými pozíciami vkladaných uzlov nesmú byť porušené podmienky vyžadované ďalšími výrazmi `insert` počas platnosti jedného snapshotu.
- Pozície vkladaných uzlov medzi deťmi cieľového uzla sú teda závislé na implementácii.

Príklad aktualizačnej požiadavky, ktorá vloží tretiu položku objednávky 01 ako poslednú položku objednávky 02:

```
insert nodes
    doc('objednavka_01.xml')/objednavka/polozka[3]
  as last into
      doc('objednavka_02.xml')/objednavka
```

Výraz skladajúci sa zo štyroch výrazov `insert insert` ktoré budú vykonané počas platnosti jedného snaptshotu:

```
insert node (<g/>,<h/>) as last into A/D[@id = '1'],
insert nodes (<e/>,<f/>) into A/D[@id = '1'],
insert node <c/> after A/D/b,
insert nodes <d/> after A/D/b
```

Ak by bol aplikovaný na XML dokument v výpisu 1.1, výsledok by bol závislý na implementácií.

```
<A>
  <D id='1'>
    <A/>
    <B/>
  </D>
</A>
```

Výpis 1.1: Vstupný XML dokument

```
<D id='1'>
  <A/> <B/>
  <c/> <d/>
  <e/> <f/>
  <g/> <h/>
</D>
```

Výpis 1.2: Prípustný výsledok aktualizácie 01

```
<D id='1'>
  <e/>
  <A/>
  <f/>
  <B/>
  <d/>
  <c/>
  <g/> <h/>
</D>
```

Výpis 1.3: Prípustný výsledok aktualizácie 02

`SourceExpr` je jednoduchý výraz, ktorý je vyhodnotený ako uzavretý výraz v konštruktore uzla typu element. Výsledkom vyhodnotenia je postupnosť uzlov. Táto postupnosť nesmie obsahovať uzol typu atribút nasledujúci po uzle, ktorý nie je typu atribút. Vkladaná postupnosť sa teda skladá z postupnosti uzlov typu atribút, označme ju `$alist`, nasledovanej postupnosťou uzlov, ktoré nie sú typu atribút, označme ju `$clist`. Každá z postupností môže byť prázdna.

`TargetExpr` je jednoduchý výraz. V prípade, že je uvedená niektorá z foriem `into`, musí byť jeho výsledkom uzol typu element alebo dokument. V prípade, že je uvedené `before` alebo `after`, musí byť jeho výsledkom uzol typu element, text, komentár alebo riadiacia inštrukcia. Označme tento uzol `$target` a jeho rodiča `$parent`. Výsledkom výrazu `insert` je prázdna XDM inštancia a zoznam čakajúcich aktualizácii, ktorý pozostáva z nasledovných aktualizačných primitív:

- ak je uvedená niektorá z formiem `into` a
  - ak je `$alist` neprázdny, do zoznamu čakajúcich aktualizácií pridaj `upd:insertAttributes($target, $alist)`
  - ak je `$clist` neprázdny, do zoznamu čakajúcich aktualizácií pridaj príslušné z aktualizačných primitív:
    - `upd:insertIntoAsFirst($target, $clist)`
    - `upd:insertIntoAsLast($target, $clist)`
    - `upd:insertInto($target, $clist)`
- ak je uvedené `before` alebo `after` a
  - ak je `$alist` neprázdny, do zoznamu čakajúcich aktualizácií pridaj `upd:insertAttributes($parent, $alist)`
  - ak je `$clist` neprázdny, do zoznamu čakajúcich aktualizácií pridaj príslušné z aktualizačných primitív:
    - `upd:insertBefore($target, $clist)`
    - `upd:insertAfter($target, $clist)`

Delete

Výraz `delete` je základný aktualizačný výraz, ktorý žiadne alebo viaceré uzly určené cieľovým výrazom (`TargetExpr`) zmaže z XDM inštancie. Kľúčové slová `node` a `nodes` sú ľubovoľne zamenitelné, bez ohľadu na to koľko uzlov bude zmazaných.
Syntax výrazu `delete`:

```
DeleteExpr ::= "delete" ("node" | "nodes") TargetExpr

TargetExpr ::= ExprSingle
```

Príklad aktualizačnej požiadavky, ktorá zmaže druhú položku objednávky 01:

```
delete node
    doc('objednavka_01.xml')/objednavka/polozky/polozka[2]
```

`TargetExpr` je jednoduchý výraz. Jeho výsledkom je postupnosť uzlov, ktorá obsahuje žiadne alebo viacero uzlov. Každý uzol z tejto postupnosti, ktoý má rodiča označme `$tnode`. Výsledkom výrazu `delete` je prázdna XDM inštancia a zoznam čakajúcich aktualizácií, ktorý obsahuje aktualizačné primitívum `upd:deleteNode($tnode)` pre každý uzol `$tnode`.

Replace

Výraz `replace` je základný aktualizačný výraz, ktorý má dve formy **Nahradenie uzla** a **Nahradenie hodnoty uzla** líšiace sa uvedením kľúčových slov `value of`. Syntax výrazu `replace`:

```
ReplaceExpr ::= "replace" ("value" "of")? "node" TargetExpr
    "with" ExprSingle

TargetExpr ::= ExprSingle
```

Nahradenie uzla Ak nie sú uvedené kľúčové slová `value of`, výraz `replace` nahradí uzol určený cieľovým výrazom (`TargetExpr`), postupnosťou obsahujúcou žiadne alebo viaceré uzly. Nahradzujúce uzly sú do XDM inštancie umiestnené na miesto nahradzovaného uzla a z toho dôvodu môže byť uzol typu atribút nahradný len postupnosťou obsahujúcou žiadne alebo viaceré uzly typu atribút a uzol typu element, text, komentár alebo riadiaca inštrukcia môže byť nahradený iba postupnosťou obsahujúcou žiadne alebo viaceré uzly typu element, text, komentár alebo riadiaca inštrukcia.
Príklad aktualizačnej požiadavky, ktorá nahradí tretiu položku objednávky 01 poslednou položkou objednávky 02:

```
replace node
    doc('objednavka_01.xml')/objednavka/polozka[3]
  with
      doc('objednavka_02.xml')/objednavka/polozka[last()]
```

Výraz nasledujúci kľúčové slovo `with` je jednoduchý výraz, ktorý je vyhodnotený ako uzavretý výraz v konštruktore uzla typu element. Výsledkom vyhodnotenia je postupnosť uzlov, označme je `$rlist`.

`TargetExpr` je jednoduchý výraz, ktorého výsledkom je jeden uzol, označme ho `$target`. Ak je `$target` typu element, text, komentár alebo riadiacia inštrukcia, `$rlist` musí byť prázdny alebo musí byť zložený len z uzlov typu element, text, komentár alebo riadiacia inštrukcia. Ak je `$target` typu atribút, `$rlist` musí byť prázdny alebo musí byť zložený z uzlov typu atribút. Výsledkom výrazu `replace` je prázdna XDM inštancia a zoznam čakajúcich aktualizácií pozostávajúci z aktualizačného primitíva `upd:replaceNode($target, $rlist)`.

Nahradenie hodnoty uzla V prípade, že kľúčové slová `value of` uvedené sú, výraz `replace` nahradí hodnotu uzla určeného cieľovým výrazom (`TargetExpr`) a zachová jeho identitu uzla.
Príklad aktualizačnej požiadavky, ktorá Nahradí cenu tretej položky objednávky 01 jej dvojnásobkom:

```
replace node
    doc('objednavka_01.xml')/objednavka/polozka[3]/cena
  with
      doc('objednavka_01.xml')/objednavka/polozka[3]/cena*2
```

Výraz nasledujúci kľúčové slovo `with` je jednoduchý výraz, ktorý je vyhodnotený ako výraz obsahu v konštruktore uzla typu text. Výsledkom vyhodnotenia je prázdna postupnosť uzlov alebo uzol typu text, označme ho `$text`.

`TargetExpr` je jednoduchý výraz, ktorého výsledkom je jeden uzol, označme ho `$target`. Ak je `$target` uzol typu element, výsledkom výrazu `replace` je prázdna XDM inštancia a zoznam čakajúcich aktualizácií pozostávajúci z aktualizačného primitíva `upd:replaceElementContent($target, $text)`. Ak je `$target` uzol typu atribút, text, komentár alebo riadiacia inštrukcia, označme `$string` hodnotu uzla `$text` a výsledkom výrazu `replace` je prázdna XDM inštancia a zoznam čakajúcich aktualizácií pozostávajúci z aktualizačného primitíva `upd:replaceValue($target, $string)`.

Rename

Výraz `rename` je základný aktualizačný výraz a nahradí `meno`, vlastnosť uzla určeného cieľovým výrazom `TargetExpr`, novým menom typu `QName`.
Syntax výrazu `rename`:

```
RenameExpr ::= "rename" "node" TargetExpr "as" NewNameExpr


TargetExpr ::= ExprSingle
NewNameExpr ::= ExprSingle
```

Príklad aktualizačnej požiadavky, ktorá zmení mená všetkých elementov `komentar` v objednávke 01 na `poznamka`:

```
rename node
    doc('objednavka_01.xml')/objednavka/polozka/komentar
  as
      "poznamka"
```

`TargetExpr` je jednoduchý výraz, ktorého výsledkom je uzol typu element, attribút alebo riadiaca inštrukcia, označme ho `$target`. `NewNameExpr` je jednoduchý výraz, ktorý je vyhodnotený nasledovne:

- ak je `$target` uzol typu element, označme `$QName` výsledok vyhodnotenia `NewNameExpr` ako výrazu `name` z počítaného konštruktoru pre uzol typu element.
- ak je `$target` uzol typu atribút, označme `$QName` výsledok vyhodnotenia `NewNameExpr` ako výrazu `name` z počítaného konštruktoru pre uzol typu atribút.
- ak je `$target` uzol typu riadiaca inštrukcia, označme `$NCName` výsledok vyhodnotenia `NewNameExpr` ako výrazu `name` z počítaného konštruktoru pre uzol typu riadiaca inštrukcia a `fn:QName((), $NCName)` označme ako `$QName`.

Výsledok vyhodnotenia výrazu `rename` je prázdna XDM inštancia a zoznam čakajúcich aktualizácií pozostávajúci z aktualizačného primitíva `upd:rename($target, $QName)`.

Transform

Výraz `transform` je jednoduchý výraz a slúži na vytvorenie upravenej kópie uzlov z XDM inštancie. Každý uzol vytvorený výrazom `transform` má novú identitu uzla. Výstupom výrazu `transform` je XDM inštancia, ktorá môže obsahovať uzly vytvorené výrazom `transform` ale aj pôvodné uzly.
Syntax výrazu `transform`:

```
TransformExpr ::=
    "copy" "$" VarName ":=" ExprSingle
        ("," "$" VarName ":=" ExprSingle)*
    "modify" ExprSingle
    "return" ExprSingle
```

Príklad aktualizačnej požiadavky, ktorá vráti dokument pre objednávku 01 s dvojnásobnou cenou všetkých položiek:

```
copy $objednavka := doc('objednavka_01.xml')
  modify
    replace $objednavka/objednavka/polozka/cena
      with $objednavka/objednavka/polozka/cena*2
  return
    $objednavka
```

Klauzula `copy` obsahuje jednu alebo viacero inicializácií premenných, pozostávajúcich z mena premennej a výrazu, ktorého výsledkom je jeden uzol. Označme ho `$node`. Je vytvorená kópia uzla `$node` a všetkých uzlov, ktorých je predkom. Skopírované uzly majú novú identitu, ich vlastnosti `parent`, `children` a `attributes` sú zmenené, aby zachovali vzťahy medzi skopírovanými uzlami. Vlastnosť `parent` kópie uzla `$node` je nastavená ako prázdna. Premenná je inicializovaná kópiou uzla `$node`.

Klauzula `modify` je aktualizačný výraz alebo prázdny výraz, ktorého výsledkom je zoznam čakajúcich aktualizácií, označme ho `$pul`. Všetky aktulizačné primitíva, ktoré `$pul` obsahuje, môžu mať ako cieľový uzol len niektorý zo skopírovaných uzlov.

Aktualizačné primitíva z `$pul` sú pomocou aktualizačnej rutiny `upd:upplyUpdates` aplikované na skopírované uzly. Klauzula `return` je jednoduchý výraz, ktorý môže obsahovať skopírované uzly, aj s aplikovanými zmenami, ale aj uzly s pôvodnou identitou. Výsledok klauzuly `return` je výsledkom celého výrazu `transform`.

Kombinovanie aktualizačných operácií

Aktualizačná rutina `upd:upplyUpdates` má definované neprípustné kombinácie aktualizačných primitív v jednom zozname čakajúcich aktualizačných operácií. V dôsledku toho sú neprípustné nižšie uvedené kombinácie aktualizačných výrazov počas platnosti jedného snapshotu.

- viac ako jeden výraz `rename` s rovnakých cieľovým uzlom
- viac ako jeden výraz `replace` (bez kľúčových slov `value of`) s rovnakých cieľovým uzlom
- viac ako jeden výraz `replace value of` s rovnakých cieľovým uzlom
- viacnásobné volanie funkcie `fn:put` s rovnakým parametrom URI

2. Prehľad riešení

Téme aktualizácie XML dát uložených v relačnej databáze sa venuje niekoľko prác. Bližšie sa ale pozrieme na prácu popísanú v [3]. V riešení navrhnutom v spomínanej práci sú XML dokumenty ukladané do štruktúry v relačnej databáze. Táto štruktúra je výsledkom schémou riadeného mapovania pre schému popísanú v jazyku DTD. Jazyk aktualizačnej požiadavky má podobnú syntax ako jazyk pre XQuery Update Facility 1.0. Zásadne sa však tento jazyk od XQuery líši sémantikou a spôsobom vyhodnocovania aktualizačnej požiadavky. Tento rozdiel je spôsobený rozdielnym poradím aplikácie zmien do podkladového úložiska.

2.1 Poradie aplikácie zmien

V aktualizačnej požiadavke na aktualizáciu XML dokumentu sa môže vyskytovať niekoľko základných aktualizačných výrazov. Pre jednoznačný výsledok aktualizačnej požiadavky je potrebné zaviesť pravidlá na určenie poradia, v akom majú byť jednotlivé základné aktualizačné výrazy aplikované na XML dokument. V [3] je toto poradie totožné s poradím, v akom sa základné aktualizačné výrazy vyskytujú v aktualizačnej požiadavke.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="A" type="ComplexType1"/>

 <xs:complexType name='ComplexType1'>
  <xs:sequence>
   <xs:element name="B" type="ComplexType2" minOccurs="0"
       maxOccurs="2" />
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name='ComplexType2'>
  <xs:sequence>
   <xs:element name="C" type="xs:string" minOccurs="0"
       maxOccurs="2" />
  </xs:sequence>
 </xs:complexType>

</xs:schema>
```

Výpis 2.1: XML schéma 005.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<A xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="005.xsd">
 <B>
  <C>prvy</C>
 </B>
```

```
</A>
```

Výpis 2.2: XML dokument 005.xml

```
for $p in doc("005.xml")/A/B
 delete $p/C,
 insert <C>druhy</C> into $p
```

Výpis 2.3: aktualizačná požiadavka

```
for $p in doc("005.xml")/A/B
 insert <C>druhy</C> into $p,
 delete $p/C
```

Výpis 2.4: aktualizačná požiadavka

```
<?xml version="1.0" encoding="UTF-8"?>
<A xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="005.xsd">
 <B>
  <C>druhy</C>
 </B>
</A>
```

Výpis 2.5: XML dokument 005.xml po aktualizácii vo výpise 2.3

Ako prvý bol aplikovaný základný aktualizačný výraz `delete`, ktorým bol zmazaný uzol C a následne bol výrazom `insert` vložený nový uzol C s iným obsahom.

```
<?xml version="1.0" encoding="UTF-8"?>
<A xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="005.xsd">
 <B>
 </B>
</A>
```

Výpis 2.6: XML dokument 005.xml po aktualizácii vo výpise 2.4

Ako prvý bol aplikovaný základný aktualizačný výraz `insert`, ktorým bol vložený druhý uzol C a následne boli oba uzly C zmazané výrazom `delete`. Takýto pomerne jednoduchý prístup je možné popísať:

1. podľa poradia v aktualizačnej požiadavke pre každý základný aktualizačný výraz:
   (a) vyhodnoť vstupné parametre základného aktualizačného výrazu
   (b) aplikuj zmeny reprezentované základným aktualizačným výrazom

To umožňuje základné aktualizačné výrazy prepísať na SQL požiadavky a následne ich vykonať podľa poradia v aktualizačnej požiadavke. Takýto prístup dovolí, že len zmenou poradia základných aktualizačných výrazov v aktualizačnej požiadavke, sa z vykonateľnej aktualizačnej požiadavky stane aktualizačná požiadavka končiaca chybou.

Toto sa podarilo odstrániť v aktualizačných prostriedkoch pre XQuery 1.0 kde výsledný XML dokument je úplne nezávislý od poradia základných aktualizačných výrazov v aktualizačnej požiadavke. Jednotlivé zmeny sú od seba izolované a navzájom sa neovplyvňujú. Určenie poradia v akom majú byť zmeny aplikované je ale značne komplikovanejšie a prepísanie základného aktualizačného výrazu na SQL požiadavky a ich následné aplikovanie už nie je možné.

Prístup aktualizácií XQuery je možné zjednodušene popísať:

1. v ľubovolnom poradí pre každý základný aktualizačný výraz:
   (a) vyhodnoť vstupné parametre základného aktualizačného výrazu a ich hodnotu uchovaj
2. v poradí určenom špecifikáciou pre každú zmenu:
   (a) aplikuj zmenu do podkladového úložiska

2.2 Zvolené riešenie

Ciele tejto práce, ktoré sú popísané v úvode, sa pretavili do cieľov, ktoré sa snažíme dosiahnuť v riešení:

- implementovať jazyk aktualizačných požiadaviek s podobnou syntaxou a sémantikou ako má jazyk XQuery Update Facility 1.0
- čo najväčšiu časť vyhodnotenia aktualizačnej požiadavky vykonať v relačnej databáze
- namapovať na prostriedky relačnej databázy čo najviac z integritných obmedzení jazyka XML Schema, tak aby nepovolili vykonanie aktualizačných operácií, ktoré ich porušujú
- využiť pre uloženie hodnôt dátových typov XML Schema podobné dátové typy relačnej databázy
- nie príliš veľký počet vytvorených tabuliek v objektovo-relačnej schéme
- možnosť uložiť viacero validných dokumentov s rovnakou XML schémou
- možnosť uložiť viacero validných dokumentov pre rôzne XML schémy

Algoritmy sú navrhnuté pre relačnú databázu Oracle 10.2, v ktorej sú aj implementované. Pôvodne bola databáza Oracle 10.2 vybraná pre malý počet odchýliek od normy SQL a algoritmy mali byť popísané aj v tejto norme. Rozdiely boli ale natoľko zásadné, že by bolo potrebné vytvoriť dva odlišné algoritmy, jeden pre normu SQL a druhý pre Oracle 10.2 pre implementáciu. Z uvedeného dôvodu boli navrhnuté algoritmi len pre relačnú databázu Oracle 10.2 a algoritmi pre normu SQL navrhnuté neboli.

3. Mapovanie XML Schema do objektovo-relačnej databázy

Jednou z možností ako implementovať aktualizačné operácie nad XML dokumentom je využiť ako úložisko relačnú databázu prípadne objektovo-relačnú databázu. Hlavnou z výhod, ktorú relačná databáza poskytuje sú transakcie. Vďaka nim je možné aplikovať zložité aktualizačné požiadavky a v prípadne, že výsledný dokument nie je validný voči schéme, jednoducho všetky vykonané čiastkové zmeny vrátiť späť. Ďalšou výhodou je prepracovaný systém integritných obmedzení. Pomocou neho sa dá zaručiť, že XML dokument, nad ktorým sa vykonávajú aktualizačné operácie, bude aj po ich skončení v konzistentnom stave a validný voči pôvodnej XML schéme. Jazyk XML Schema však poskytuje integritné obmedzenia, ktoré nie je možné priamo mapovať na integritné obmedzenia relačnej databázy. Je teda potrebné zvoliť, akú časť jazyka XML Schema bude riešenie využívajúce relačnú databázu podporovať.

Navrhnuté riešenie ukladania XML dokumentov do objektovo-relačnej databázy vychádza z riešenia popísaného v [1]. Každé z mapovaní na prostriedky relačnej databázy a následne aj algoritmy, z pôvodného riešenia, museli byť prehodnotené s ohľadom na vykonateľnosť aktualizačných operácií a ďalších cieľov tejto práce.

3.1 Popis mapovania

V tejto kapitole popíšeme, na ktorý z prostriedkov relačnej databázy Oracle 10.2 sú jednotlivé prvky jazyka XML Schema v riešení mapované. Popis obsahuje zdôvodnenie výberu daného postupu, prípadne aj popis inej alternatívy, ktorá by sa na prvý pohľad mohla zdať ako lepšia.

3.1.1 Krátky popis mapovania

- Jednoduchý typ je mapovaný na SQL dátový typ a množinu integritných obmedzení.
- Viac-hodnotový jednoduchý typ je mapovaný na dátový typ VARCHAR2.
- Atribúty sú mapované na UDT[1], ktorý obsahuje jediný atribút s príslušným jednoduchým typom.
- Modelové skupiny a komplexné typy sú mapované na UDT. Ich poduzly sú mapované na atribúty UDT. Atribúty komplexných typov sú mapované rovnako na atribúty UDT. Typom atribútu UDT môže byť UDT, referencia na UDT, pole UDT prípadne pole referencií na UDT v závislosti od vzťahu uzlu k poduzlom.

[1]User Defined Type - uživateľom definovaný typ v objektovo-relačnej databáze Oracle. Obdoba tried v objektových jazykoch so štandartným objektovým modelom.

- Elementy s komplexným typom sú mapované na UDT tohoto komplexného typu. Je pre ne vytvárana objektová tabuľka.
- Elementy s jednoduchým dátovým typom sú mapované na UDT, ktorý obsahuje jediný atribút uchovávajúci hodnotu elementu. Typ atribútu je určený jednoduchým typom elementu. Tiež je pre ne vytváraná objektová tabuľka.
- Vzťah uzol a poduzol s viac-násobným výskytom je mapovaný na pole (VARRAY).

3.1.2 Jednoduché dátové typy

Jednoduché dátové typy z jazyka XML Schema, sú mapované na jednoduché SQL dátové typy relačnej databázy Oracle 10.2. Predpokladáme, že v riešení budú aktualizačné požiadavky prepisované do jazyka SQL a teda nestačí, aby relačná databáza dokázala len uchovať hodnotu pre daný dátový typ. Je potrebné, aby relačná databáza podporovala vyhodnotenie aspoň základných operátorov nad týmito dátovými typmi. V prípade, že v relačnej databáze neexistuje dátový typ s veľmi podobným priestorom hodnôt a s veľmi podobným vyhodnocovaním operátorov ako v jazyku XQuery, sú na výber dve možnosti:

- Mapovať takéto typy na UDT uchovávajúci textovú hodnotu. UDT by obsahovalo metódy, ktoré kontrolujú či daná hodnota patrí do lexikálneho priestoru a metódy na vyhodnocovanie operátorov podľa špecifikácie XQuery.
- Mapovať takýto dátový typ na `varchar2`.

V riešení je použitá druhá možnosť, nakoľko je pre potreby tejto práce dostatočná.

Dátový typ `boolean`

Nakoľko relačná databáza Oracle 10.2 nepodporuje žiadny podobný typ, je typ `boolean` mapovaný na typ `varchar2(5)`. Pre obmedzenie prípustných hodnôt musí hodnota spĺňať podmienku:

```
regexp_like(column, '^(true|false|1|0)$')
```

Kde `column` označuje stĺpec.

Dátový typ `decimal`

Dátový typ `decimal` je mapovaný na typ `NUMBER(L, K)`. `L` a `K` sú určené reštrikciou.

Dátový typ `string`

Dátový typ `string` je mapovaný na typ `varchar2(L)`. `L` je určené reštrikciou.

Dátový typ `hexBinary`

`HexBinary` je mapované na typ `varchar2(L)`, kde `L` je určené reštrikciou. Pre obmedzenie prípustných hodnôt musí hodnota spĺňať podmienku:

```
regexp_like(column, '^[0-8a-fA-F]*$').
```

Kde `column` označuje stĺpec.

Dátový typ `base64Binary`

`base64Binary` je mapované na typ `varchar2(L)`, kde `L` je určené reštrikciou. Pre čiastočné obmedzenie prípustných hodnôt musí hodnota spĺňať podmienku:

```
regexp_like(column, '^[A-Za-z0-9+/= ]*$')
```

Kde `column` označuje stĺpec.

Dátové typy `anyURI`, `NOTATION`, `QName`

Tieto dátové typy sú mapované na typ `varchar2(L)`, kde `L` je určené reštrikciou. V relačnej databáze nie sú prostriedky, na ktoré by mohli byť obmedzenia pre prípustné hodnoty mapované.

Dátový typ `date`

Dátový typ `date` by bolo možné mapovať na SQL dátový typ `DATE`. Ten však pripúšťa hodnoty pre rok len v intervale od -4712 do 9999 a pri vyhodnocovaní výrazov by bolo potrebné riešiť konverziu z reťazca na typovú hodnotu, čo pre formát definovaný v špecifikácií XML Schema relačná databáza nerobí automaticky. Preto je dátový typ `date` mapovaný na `varchar2`.

Dátový typ `dateTime`

Dátový typ `dateTime` by bolo možné mapovať na dátový typ `TIMESTAMP(L) WITH TIME ZONE` pre vhodne zvolené `L`. Ten však pripúšťa hodnoty pre rok len v intervale od -4712 do 9999 a pri vyhodnocovaní výrazov by bolo potrebné riešiť konverziu z reťazca na typovú hodnotu, čo pre formát definovaný v špecifikácií XML Schema relačná databáza nerobí automaticky. Preto je dátový typ `dateTime` mapovaný na `varchar2`.

Dátový typ `time`

Pre tento dátový typ neexistuje v Oracle 10.2. dátový typ s podobnými vlastnosťami. Preto je mapovaný na `varchar2`.

Dátový typ `duration`, `gDay`, `gMonth`, `gMonthDay`, `gYear`, `gYearMonth`

Pre tieto dátové typy neexistujú v Oracle 10.2. dátové typy s podobnými vlastnosťami. Preto sú mapované na `varchar2`.

Dátový typ `float`

Typ `float` je mapovaný na typ `BINARY_FLOAT`, ktorý spĺňa prevažnú časť normy IEEE 754-1985[2], ktorá je vyžadovaná špecifikáciou XML Schema. Podporuje operátory + - * / a operátory porovnania.

Dátový typ `double`

Typ `double` je mapovaný na typ `BINARY_DOUBLE`, ktorý spĺňa prevažnú časť normy IEEE 754-1985, ktorá je vyžadovaná špecifikáciou XML Schema. Podporuje operátory + - * / a operátory porovnania.

3.1.3 Odvodené jednoduché dátové typy

Mapovanie vstavaného aj užívateľsky definovaného odvodeného jednoduchého dátového typu je závislé od spôsobu odvodenia.

Odvodenie zoznamom

Jednou z možností ako mapovať jednoduchý dátový typ odvodený zoznamom je mapovať ho na pole (`VARRAY`) jednoduchých SQL dátových typov. Typom prvkov poľa by bol typ, na ktorý je mapovaný jednoduchý typ prvkov zoznamu. Reštrikcie pre prvky by boli kontrolované triggerom a nie integritnými obmedzeniami pre tabuľku. Komplikované by bolo vyhodnocovanie výrazov a aj aktualizačné operácie. Preto je v riešení jednoduchý dátový typ odvodený zoznamom mapovaný na `varchar2`. Jednoduché dátové typy odvodené zoznamom budeme ďalej označovať ako viac-hodnotové dátové typy.

Odvodenie zjednotením

Takýto dátový typ by mohol byť mapovaný na dátové typy, ktoré zjednocuje. Pri ukladaní by hodnota bola uložená na miesto pre príslušný dátový typ zo zjednotenia. Pri takomto mapovaní by bolo problematické zistiť, akého typu zo zjednotenia aktuálna hodnota vlastne je. Z uvedeného dôvodu je jednoduchý dátový typ odvodený zjednotením mapovaný na `varchar2`.

Odvodenie reštrikciou

Pri tomto druhu odvodenia zostáva SQL dátový typ zachovaný. Pridajú sa len integritné obmedzenia, na ktoré je reštrikcia mapovaná. Integritné obmedzenie sa do relačnej databázy premietne buď ako integritné obmedzenie `CHECK` alebo ako podmienka v triggeri.

Reštrikcia `whiteSpace`

Určuje akým spôsobom majú byť spracované biele znaky pri vytvorení normalizovanej hodnoty z textovej hodnoty. Prostriedok, na ktorý by bola táto reštrikcia mapovaná v relačnej databáze neexistuje.

[2]Norma popisújuca formát číselných dátových typov s pohyblivou desatinnou čiarkou. Jej komplentné znenie je možné nájsť v dokumente [12].

Reštrikcia `totalDigits` a `fractionDigits`

Táto reštrikcia je povolená pre typ `decimal` a typy od neho odvodené. Reštrikcia je mapovaná parametre SQL dátového typu `NUMBER`. Maximálnou hodnotou podporovanou relačnou databázou Oracle 10.2 `totalDigits` je 38. Všetky väčšie hodnoty sú nahradené touto hodnotou.

`fractionDigits = N`	`number(L, N)`
`totalDigits = N`	`number(N, L)`

Tabuľka 3.1: Mapovanie reštrikcie `totalDigits`

Reštrikcia `maxExclusive`, `minExclusive`, `maxInclusive` a `minInclusive`

Táto reštrikcia je povolená pre typy `decimal`, `float`, `double` a typy od nich odvodené. Tiež je povolená pre typy pre dátum a čas, v riešení je však pre tieto typy ignorovaná.

`maxExclusive = M`	`(column < M)` kde `column` je meno stĺpca
`minExclusive = M`	`(column > M)`
`maxInclusive = M`	`(column <= M)`
`minInclusive = M`	`(column >= M)`

Tabuľka 3.2: Mapovanie reštrikcie `maxExclusive`

Reštrikcia `enumeration`

Táto reštrikcia je povolená pre všetky vstavané typy a typy od nich odvodené okrem typu `boolean`. `enumeration` = "$E_1, E_2, \ldots E_n$" je mapované na podmienku `( column in (`$E_1, E_2, \ldots E_N$`) )` kde `column` je meno stĺpca.

Reštrikcia `length`, `maxLength`, `minLength`

Tieto reštrikcie sú povolené pre typ `string` a typy od neho odvodené a obmedzuje počet znakov v reťazci. Sú tiež povolené pre typy odvodené zoznamom, kde obmedzuje počet prvkov v zozname. V druhom prípade sú tieto reštrikcie v riešení ignorované.

`length = L`	`varchar2(L)` a podmienka `(length(column) = L)` kde `column` je meno stĺpca
`maxLength = L`	`varchar2(L)`
`minLength = L`	podmienka `(length(column) >= L)`

Tabuľka 3.3: Mapovanie reštrikcie `length`

Maximálna dĺžka typu `varchar2`, ktorú pre stĺpec podporuje relačná databáza Oracle 10.2 je 4000. Všetky väčšie hodnoty sú nahradené touto hodnotou.

Reštrikcia `pattern`

Táto reštrikcia by mohla byť mapovaná na podmienku v kombinácií s funkciou `regexp_like`. Reštrikcia `pattern` však využíva komplikovanejšie regulárne výrazy ako funkcia `regexp_like` a teda toto riešenie je nepoužiteľné. V implementácií je reštrikcia `pattern` z XML schémy ignorovaná. Pri snahe mapovať reštrikciu `pattern` je potrebné zvážiť, že integritné obmedzenie `pattern` sa aplikuje na normalizovanú hodnotu, nie na typovú hodnotu a ani na kanonickú reprezentáciu danej typovej hodnoty. To v kombinácii s úložiskom XML dokumentov, ktoré uchováva len typovú hodnotu, môže spôsobiť problém. Spracovanie textovej hodnoty na typovú hodnotu prebieha nasledovne:

1. Z textovej hodnoty vytvor normalizovanú hodnotu spracovaním bielych znakov podľa hodnoty reštrikcie `whiteSpace`. Normalizovaná hodnota patrí do lexikálny priestor.

2. Z normalizovanej hodnoty získaj typovú hodnotu. Typová hodnota je prvkom priestoru hodnôt.

Jedna hodnota z priestoru hodnôt je typovou hodnotou pre jednu alebo viac normalizovaných hodnôt. Jedna hodnota z lexikálneho priestoru je normalizovanou hodnotou pre jednu alebo viac textových hodnôt. Pre každú typovú hodnotu existuje práve jedna kanonická reprezentácia, ktorá je tiež normalizovanou hodnotou.

lexikálny priestor	{`true, false, 1, 0`}
priestor hodnôt	{`true, false`}
kanonická reprezentácia	{`true, false`}

Tabuľka 3.4: Hodnoty dátového typu `boolean`

V dokumentoch získaných z úložiska uchovávajúceho len typovú hodnotu by boli použité kanonické reprezentácie uložených typových hodnôt a nie pôvodné textové (normalizované) hodnoty. To spôsobí, že dokument získaný z úložiska nemusí byť validný voči schéme aj v prípade ak pôvodný dokument uložený do úložiska validný voči schéme bol.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="A" type="ComplexType1"/>

 <xs:simpleType name="SimpleType1">
  <xs:restriction base="xs:boolean">
   <xs:pattern value="[01]"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name='ComplexType1'>
  <xs:sequence>
   <xs:element name="B" type="SimpleType1"/>
   <xs:element name="C" type="SimpleType1"/>
   <xs:element name="D" type="xs:boolean"/>
```

```
   <xs:element name="E" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Výpis 3.1: XML schéma 006.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<A xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="006.xsd">
 <B>0</B>
 <C>1</C>
 <D>true</D>
 <E>1</E>
</A>
```

Výpis 3.2: XML dokument 006.xml

V prípade uloženia dokumentu 006.xml do vyššie popísaného úložiska a následným získaním dokumentu z úložiska dôjde k jeho zneplatneniu voči schéme. Element A/B bude obsahovať hodnotu "`false`" a elementy A/C a A/E hodnotu "`true`". Tento problém sa dá odstrániť, ak si úložisko XML dokumentov uchováva (aj) normalizovanú hodnotu. Druhý problém nastáva pri aktualizácii.

```
let $p in doc("006.xml")/A/C
   replace value of node $p with $p and 0
```

Výpis 3.3: XQuery aktualizačná požiadavka 006-1.xq

Požiadavka 006-1.xq sa snaží nahradiť hodnotu "`1`" elementu A/C, ktorá spĺňa integritné obmedzenie `pattern`, hodnotou výrazu "`1 and 0`". Výslednou hodnotou výrazu je typová hodnota "`false`". Pri pokuse uložiť túto hodnotu do elementu A/C je integritné obmedzenie `pattern` aplikované na kanonickú reprezentáciu tejto typovej hodnoty teda "false". Lenže reštrikcia `pattern` očakáva hodnoty "`0`" alebo "`1`" a teda celá požiadavka skončí chybou. Pre tento problém elegantné riešenie neexistuje. Jediná možnosť je pri vytváraní schémy pre XML dokument dobre zvážiť použité reštrikcie `pattern`.

3.1.4 Komplexné dátové typy

Komplexné typy reprezentujú obsah elementov, ktorý sa skladá z atribútov a elementov. V riešení sú mapované na UDT v relačnej databáze Oracle 10g Release 2. Poduzly tohoto typu sú mapované na atribúty UDT.

Jednoduchý obsah

Komplexný typ s jednoduchým obsahom je mapovaný na UDT. Jednoduchý obsah je mapovaný na atribút UDT. Jeho typom je príslušný jednoduchý SQL dátový typ, ktorý zodpovedá mapovaniu jednoduchých typov popísanému vyššie. Integritné obmedzenia pre jednoduchý obsah sú kontrolované integritným obmedzením `CHECK` alebo triggerom.

Atribúty komplexného typu sú mapované na atribúty UDT. Mapovanie atribútov komplexného typu je popísané ďalej.

Komplexný obsah

Komplexný typ s komplexným obsahom je mapovaný na UDT. Komplexný obsah je reprezentovaný modelovou skupinou, ktorá je mapovaná na UDT. V komplexnom type je teda komplexný obsah mapovaný na atribút UDT, ktorého typ je buď UDT, referencia na UDT, pole UDT alebo pole referencií na UDT pre príslušnú modelovú skupinu.

Atribúty komplexného typu sú mapované na atribúty UDT.

3.1.5 Modelové skupiny

Modelové skupiny slúžia na definovanie obsahu komplexných typov. V riešení je každá definícia modelovej skupiny mapovaná na UDT. Modelové skupiny sú hniezdené do UDT naduzla. Výnimku tvoria iba globálne definované modelové skupiny a modelové skupiny, ktorých naduzlom je modelová skupina typu výber z uzlov. V týchto dvoch prípadoch je pre UDT pre modelovú skupinu vytvorená objektová tabuľka a modelová skupina je do naduzla mapovaná na referenciu, prípadne pole referencií. UDT pre vzájomne vnorené modelové skupiny môžu tvoriť pomerne zložitú štruktúru medzi UDT elementu a UDT jeho rodičovského elementu.

Modelová skupina typu postupnosť uzlov

Modelová skupina typu postupnosť uzlov je mapovaná na UDT. Každý z jej poduzlov je mapovaný na atribút tohoto UDT. Typ atribútu je závislý na type poduzla a môže ním byť UDT, referencia na UDT, pole UDT alebo pole referencií na UDT.

Modelová skupina typu množina uzlov

Modelová skupina typu množina uzlov je mapovaná na UDT. Každý z jej poduzlov je mapovaný na dva atribúty tohoto UDT. Typ prvého atribútu je závislý na type poduzla a môže ním byť len UDT alebo referencia na UDT nakoľko v modelovej skupine typu množina uzlov nie je povolený viacnásobný výskyt poduzlov. Druhý atribút slúži na uloženie poradia uzla v XML dokumente.

Modelová skupina typu výber z uzlov

Modelová skupina typu výber z uzlov je mapovaná na UDT s jedným atribútom typu referencia na UDT, ktorý je spoločným predkom všetkých UDT pre konkrétnu XML schému. Všetky prvky modelovej skupiny sú mapované na UDT, vrátane elementov jednoduchého typu. Každý z prvkov má vlastnú objektovú tabuľku.

Alternatívne riešenie mapuje modelovú skupinu typu výber z uzlov na rovnaké UDT. Ku každej modelovej skupine typu výber z uzlov je však vytvorená samostatná objektová tabuľka pre typ, ktorý je spoločným predkom všetkých UDT pre konkrétnu XML schému. Do nej sú ukladané inštancie UDT pre prvky modelovej skupiny. Integritným obmedzením `SCOPE` je zaručené, že referencia bude odkazovať vždy do tejto tabuľky a integritnými obmedzeniami je povolené do tejto tabuľky ukladať len inštancie UDT, na ktorý sú mapované prvky modelovej

skupiny. Pre modelovú skupinu typu výber z uzlov sú všetky prvky, a teda aj elementy, ktoré sú jednoduchého typu, mapované na UDT. Obmedzením databázy Oracle je, že integritné obmedzenie `SCOPE` nemôže byť použité pre prvky poľa. Preto by v tomto riešení bolo potrebné UDT pre modelovú skupinu typu výber z uzlov, uložiť do objektovej tabuľky alebo do niektorého zo stĺpcov tabuľky.

Výhodou alternatívneho riešenia je, že umožňuje uložiť práve XML dokumenty, ktoré sú validné voči konkrétnej XML schéme. Pri prvom riešení môže referencia odkazovať na ľubovolný UDT, čo umožňuje vložiť aj dokumenty, ktoré nie sú validné voči schéme. Nevýhodou druhého riešenia je nutnosť pretypovania pri prístupe k inštanciám v tabuľke pre prvky modelovej skupiny. Po pretypovaní sa pre prístup k atribútom uložených objektov využíva objektový pohľad nad tabuľkou. Cez objektový pohľad nie je možné aktualizovať jednotlivé atribúty objektov uložených v tejto tabuľke. Navyše objektová tabuľka v relačnej databáze Oracle 10.2 obsahuje neviditeľné stĺpce pre atribúty všetkých podtypov typu, pre ktorý bola definovaná. Práve vďaka tomu je možné do nej uložiť všetky podtypy. V prípade komplikovanej XML schémy, tak môže mať objektová tabuľka pre typ, ktorý je predkom všetkých UDT pre danú schému, veľké množstvo stĺpcov. Do implementácie bolo zvolené prvé riešenie, nakoľko je vhodnejšie pre aktualizácie, ktoré sú predmetom tejto práce.

Mapovanie atribútov `maxOccurs` a `minOccurs` pre modelové skupiny

To či je modelová skupina do naduzla mapovaná na pole alebo na atribút je závislé práve od atribútu `maxOccurs`. Mapovanie týchto atribútov pre modelové skupiny je rovnaké ako pre elementy a je popísané ďalej.

Dedičnosť komplexných dátových typov

XML Schéma ponúka dva druhy dedičnosti komplexných typov, dedenie rozšírením a dedenie reštrikciou. Dedenie rozšírením je štandardné dedenie tak ako ho poznáme z iných objektových modelov, kde v potomkovi môžu pribudnúť nové atribúty a elementy. Dedenie reštrikciou naopak môže elementy a atribúty z potomka odoberať. Pre dedičnosť reštrikciou neexistuje mapovanie do objektovo relačnej databázy a preto je dedičnosť v implementácii ignorovaná.

Alternatívne riešenie, ktoré takmer podporuje dedičnosť komplexných typov ale hlavne možnosť nahradiť typ elementu, typom od tohto typu odvodeným elementu, si popíšeme v nasledujúcich odstavcoch.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="eA" type="A"/>

 <xs:complexType name="A">
  <xs:simpleContent>
   <xs:restriction base="xs:string"/>
  </xs:simpleContent>
 </xs:complexType>

 <xs:complexType name="B">
  <xs:simpleContent>
   <xs:restriction base="A">
```

```
    <xs:length value="20"/>
   </xs:restriction>
  </xs:simpleContent>
 </xs:complexType>

 <xs:complexType name="C">
  <xs:simpleContent>
   <xs:extension base="A">
    <xs:atribute name="d" type="xs:positiveInteger"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:schema>
```

Výpis 3.4: XML schéma 005.xsd

Všetky tri typy uvedené XML schéma 005.xsd by boli mapované na UDT so spoločným predkom `o`, ktorý je predkom všetkých UDT pre schému 005.xsd.

```
object ( element_id decimal,
  udt_id decimal,
  record_id decimal
) NOT FINAL;

create type a under o (
   c_0         varchar2(4000)
);

create type b under o (
   c_0         varchar2(20)
);

create type c under o (
   c_0         varchar2(4000),
   c_1         number
);
```

Výpis 3.5: alternatívne riešenie - UDT pre schému 005.xsd

Ak by bol element `eA` mapovaný na atribút UDT nadradeného uzla, tento atribút by bol typu `o`. Do tabuľky by bolo potrebné pridať nasledovné integritné obmedzenia (označenie cesta predstavuje cestu k atribútu UDT):

```
check (
   cesta is of type (only a, only b, only c)
),
check (
   cesta is not of type (only b) or
   (
      treat (cesta as b).c_0 is not null and
      length(treat (cesta as b).c_0) = 20
   )
```

```
),
check (
   cesta is not of type (only c) or
   treat (cesta as c).c_1 > 0
)
```

Výpis 3.6: alternatívne riešenie - integrtiné obmedzenia pre schému 005.xsd

V prípade elementu s viacnásobným výskytom, by bol element `eA` mapovaný na pole s prvkami typu `o`. Uvedené integritné obmedzenia by boli kontrolované triggerom.

Ak by bol element `eA` globálne definovaný element, musela by vzniknúť v objektovo relačnej schéme objektová tabuľka pre typ `o`. V tejto tabuľke by bolo potrebné definovať vyššie uvedené integritné obmedzenia a hodnotu označenia cesta nahradiť reťazcom `object_value`. Označenie `object_value` umožňuje odkázať sa na objekt v objektovej tabuľke v triggeroch a integritných obmedzeniach. Potom by bol element `eA` v UDT nadradeného uzla mapovaný na atribút typu referencia na `o`. Doplnením integritného obmedzenia `SCOPE` pre tento atribút je možné dosiahnuť, že objektovo relačná schéma bude na mieste elementu `eA` vynucovať jeden z typov `a`, `b` alebo `c`.

V prípade globálne definovaného elementu s viacnásobným výskytom, by bol element `eA` mapovaný na pole referencií na typ `o`. Na prvky poľa, ale nie je možné použiť integritné obmedzenie `SCOPE` a teda by referencia mohla odkazovať na ľubovoľný objekt, ktorý je potomkom typu `o`. Preto toto riešenie nie je použité.

Ak by aj tento problém bol odstránený, pre podporu dedičnosti a substituovatelnosti predka je potrebné spracovanie atribútu `xsi:type`[3] v elementoch XML dokumentov. Týmto atribútom je určené meno typu elementu. Prípustnými hodnotami tohoto atribútu sú mená potomkov komplexného typu pre daný element.

```
<?xml version="1.0" encoding="UTF-8"?>
<eA xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="005.xsd"
   xsi:type="C" d="11">jahoda</eA>
```

Výpis 3.7: XML dokument 005.xml

Je ním určené, ktorý z potomkov typu je typom elementu. Objektovo-relačná databáza by musela obsahovať informáciu na aké UDT sa má typ s uvedeným menom mapovať. Vyhodnocovanie atribútu `xsi:type` by bolo potrebné aj pri aktualizačných operáciach, napríklad pri vkladaní nových uzlov.

3.1.6 Atribúty

Všetky atribúty elementov sú spracovávané ako komplexné typy s jednoduchým obsahom a sú mapované na UDT, ktoré je uložené ako atribút UDT nadradeného uzla. Ako lepšie riešenie sa javí mapovať atribúty na jednoduchý typ, čím by boli tri stavy atribútu v XML dokumente zachytené troma stavmi v relačnej databáze. Popísané v tabuľke 3.5.

[3] http://www.w3.org/2001/XMLSchema-instance:type je celé označenie atribútu aj s URI pre namespace.

Atribút v XML dokumente	Mapovanie v relačnej databáze
sa nevyskytol	je vložená hodnota null
sa vyskytol s hodnotou reťazec dĺžky nula	je vložený reťazec dĺžky nula
sa vyskytol s hodnotou reťazec nenulovej dĺžky	je vložená hodnota reťazca

Tabuľka 3.5: Mapovanie stavu a hodnoty atribútu - alternatíva

Mapovanie podľa tabuľky 3.5 nie je možné použiť pri implementácii v relačnej databáze Oracle. Oracle z historických dôvodov pre reprezentovanie hodnoty null využíva reťazec s dĺžkou nula. Z uvedeného dôvodu, v rozpore s normou SQL, nie sú hodnota null a reťazec dĺžky nula v relačnej databáze Oracle dodnes odlíšiteľné. Preto sú atribúty elementov mapované na UDT s jediným atribútom, čím je umožnené reprezentovať stavy v tabuľke 3.6.

Atribút v XML dokumente	Mapovanie v relačnej databáze
sa nevyskytol	je vložená hodnota null na mieste UDT
sa vyskytol s hodnotou reťazec dĺžky nula	je vložené UDT s hodnotou atribútu nastavenou na hodnotu null
sa vyskytol s hodnotou reťazec nenulovej dĺžky	je vložené UDT s hodnotou atribútu nastavenou na túto hodnotu

Tabuľka 3.6: Mapovanie stavu a hodnoty atribútu

Pre každé dva atribúty, ktorých jednoduché dátové typy by boli mapované na rovnaký SQL dátový typ a zhodnú množinu integritných obmedzení je zbytočné definovať dva UDT, ktoré sa budú líšiť len názvom. Preto je v záujme zníženia počtu vytváraných UDT pre takéto dva atribúty zadefinovaný len jeden spoločný UDT. Dôsledkom je, že aj dva atribúty s odlišnými jednoduchými dátovými typmi v XML schéme sú mapované na rovnaký UDT, v prípade, že tieto dva jednoduché typy by boli mapované na rovnaký SQL dátový typ s rovnakou množinou integritných obmedzení.

Mapovanie atribútov elementu `attribute`

Atribút `default` elementu `attribute` z XML Schema je mapovaný na podmienku v triggeri zjednodušene zapísanú:

`IF` *COLUMN* `is null THEN` *COLUMN* `:=` *UDT*`(`*DEFAULT*`)`

- *COLUMN* je meno stĺpca, v ktorom je UDT uložený
- *UDT* je konštruktor UDT, na ktorý je atribút mapovaný
- *DEFAULT* je hodnota atribútu `default`

Tým je dosiahnuté správanie popísané v tabuľke 3.7.

Atribút v XML dokumente	Mapovanie v relačnej databáze
sa nevyskytol	je vložené UDT s hodnotou atribútu nastavenou na hodnotu atribútu `default`
sa vyskytol s hodnotou reťazec dĺžky nula	je vložené UDT s hodnotou atribútu nastavenou na hodnotu null
sa vyskytol s hodnotou reťazec nenulovej dĺžky	je vložené UDT s touto hodnotou

Tabuľka 3.7: Mapovanie stavu a hodnoty atribútu - `default`

Atribút `fixed` elementu `attribute` z XML Schema je mapovaný rovnako, s tým rozdielom, že je pridaná ďalšia podmienka na kontrolu hodnoty jediného atribútu UDT:

COLUMN `=` *FIXED*

Ak hodnota atribútu `fixed` je reťazec dĺžky nula tak:

COLUMN `is null`

- *COLUMN* je meno jediného atribútu UDT
- *FIXED* je hodnota atribútu `fixed`

Atribút `use` elementu `attribute` z XML Schema je podľa hodnoty mapovaný nasledovne:

- `prohibited` - celý element `attribute` je ignorovaný
- `optional` - je ignorovaný
- `required` - je mapovaná na podmienku: (*COLUMN* `is not null`)
    - *COLUMN* je meno stĺpca, v ktorom je UDT uložený

3.1.7 Elementy

Element s jednoduchým obsahom v XML dokumente môže nadobudnúť štyri stavy:

- element sa v XML dokumente nevyskytol
- element sa vyskytol ako prázdny, zadaný jednou značkou
- element sa vyskytol bez hodnoty, zadaný dvoma značkami
- element sa vyskytol s hodnotou medzi dvoma značkami

Druhý a tretí stav sú z pohľadu XML Schema totožné a ďalej bude tento stav označovaný ako prázdny element.

Pre korektné uloženie XML dokumentu do relačnej databázy je potrebné, aby štruktúra v relačnej databáze bola schopná tieto tri stavy reprezentovať. Je to potrebné najmä pre účinnú kontrolu validity uloženého dokumentu. Ako zrejmé

Element v XML dokumente	Mapovanie v relačnej databáze
sa nevyskytol	je vložená hodnota null
sa vyskytol prázdny	je vložený reťazec dĺžky nula
sa vyskytol s hodnotou	je vložená hodnota

Tabuľka 3.8: Mapovanie elementu s jednoduchým typom - alternatíva

sa javí mapovať element s jednoduchým obsahom na jednoduchý dátový typ podľa tabuľky 3.8.

Z rovnakého dôvodu ako pri atribútoch sú elementy s jednoduchým obsahom mapované na UDT s jediným atribútom príslušného jednoduchého typu. Tým je implementácia schopná reprezentovať všetky potrebné stavy

Element v XML dokumente	Mapovanie v relačnej databáze
sa nevyskytol	je vložená hodnota null na mieste UDT
sa vyskytol prázdny	je vložené UDT s hodnotou atribútu nastavenou na hodnotu null
sa vyskytol s hodnotou	je vložené UDT s hodnotou atribútu nastavenou na túto hodnotu

Tabuľka 3.9: Mapovanie elementu s jednoduchým typom

Elementy sú mapované na UDT svojho typu. Pre každú definíciu komplexného typu, je vytvorený UDT s objektovou tabuľkou a element s týmto typom je naň mapovaný. Pre každé dva elementy s jednoduchými dátovými typmi, ktoré by boli mapované na rovnaký SQL dátový typ a zhodnú množinu integritných obmedzení je zbytočné definovať dva UDT, ktoré sa budú líšiť len názvom. Preto je v záujme zníženia počtu vytváraných UDT pre takéto dva elementy zadefinovaný len jeden spoločný UDT. Dôsledkom je, že aj dva elementy s odlišnými jednoduchými dátovými typmi v XML schéme sú mapované na rovnaký UDT, v prípade, že tieto dva jednoduché typy by boli mapované na rovnaký SQL dátový typ a rovnakú množinu integritných obmedzení. Pre každý takto zadefinovaný UDT je vytvorená objektová tabuľka.

V prípade, že hodnota `maxOccurs` je väčšia ako 1, je vzťah uzol a poduzol mapovaný na pole referencií na UDT. V relačnej databáze Oracle 10.2 je v objektovej tabuľke možné aktualizovať objekt v nej uložený a atribúty tohoto objektu. Aktualizovať len niektorý z hniezdených objektov nie je možné. Aby boli aktualizačné operácie pre atribúty elementov čo najjednoduchšie vykonateľné, je potrebné, aby tieto atribúty boli aktualizovateľné bez nutnosti aktualizovať celý objekt v objektovej tabuľke. Na to je potrebné, aby atribúty elementov boli mapované na atribúty UDT v objektovej tabuľke a teda, aby všetky komplexné typy obsahujúce atribúty elementov boli mapované na UDT, pre ktoré je vytvorená objektová tabuľka.

Pre aktualizačné operácie je potrebné mať možnosť pristúpiť k uzlu reprezentujúcemu komplexný typ s komplexným obsahom, aj v prípade, že niektorý z jeho predkov bol presunutý na iné miesto v štruktúre. V štandardnom objektovom prostredí by to bolo možné zabezpečiť ukazovateľom na tento uzol. V prostredí

objektovo-relačnej databázy Oracle 10.2 je možné na to využiť referenciu. Referencia má však obmedzenie a je možné ju vytvoriť len na objekty uložené v objektovej tabuľke. Komplexné typy s komplexným obsahom sú teda mapované na UDT, pre ktoré je vytvorená objektová tabuľka.

Uloženie UDT aj pre elementy s jednoduchým dátovým typom do objektovej tabuľky je vhodné pre aktualizáciu ich hodnoty. Aplikovanie zmien reprezentovaných aktualizačným výrazom `replace value of` je teda možné vykonať jednoduchou SQL požiadavkou `update`.

Pre ostatné aktualizačné výrazy na elementoch je potrebné vykonať preusporiadanie súrodencov, pri ktorom môže dôjsť k presunutiu elementu na miesto, niektorého z jeho súrodencov. Pre túto operáciu je nevyhnutné, aby elementy s rovnakým menom a rovnakým typom, ktoré sú súrodencami, boli mapované na rovnaké UDT. Toto sa v implementácii podarilo dosiahnuť.

Atribúty elementu `element`

Atribút `nillable` by na prvý pohľad mohol byť mapovaný na integritné obmedzenie `[NOT] NULL`, teda povoliť na mieste elementu s atribútom `nillable` hodnotu `NULL`. Pri tomto riešení však bude dochádzať ku konfliktom s inými integritnými obmedzeniami (napr. `minOccurs`) a mapovanie na iný prostriedok relačnej databázy neexistuje. Preto je v implementácii atribút nillable ignorovaný. Validné XML dokumenty obsahujúce elementy s atribútom `nil` teda implementácia označí za nevalidné.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="shipTo" nillable="true" minOccurs="1">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:length="5"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:element>
</xs:schema>
```

Výpis 3.8: XML schéma 004.xsd

Element `shipTo` v XML dokumente	**Stav XML dokumentu**
element `shipTo` sa nevyskytol	nie je validný
`<shipTo xsi:nil="true"/>`	je validný
`<shipTo></shipTo>`	nie je validný

Tabuľka 3.10: Možné stavy dokumentu pre 004.xsd

V tomto odstavci si v krátkosti popíšeme čo by muselo riešenie, ktoré by korektne pracovalo s atribútom `nillable` v XML schéme a s atribútom `nil` v XML dokumente obsahovať. V popise XML schémy uloženej v relačnej databáze by pre každý element musel byť samostatný príznak `nillable`. Ten by určoval či je pre daný element hodnota `nil` prípustná, alebo nie. Pri každej aktualizačnej operácií na elemente s atribútom `nil` by bolo potrebné tento príznak kontrolovať. Ďalej

by bolo potrebné v popise XML dokumentu uloženého v relačnej databáze mať príznak výskytu atribútu `nil`. Potom by bolo možné integritnými obmedzeniami povoliť uloženie len dokumentov, ktoré sú validné voči schéme.

Atribút `default` z definície elementu v XML schéme je do objektovo-relačnej schémy premietnutý na podmienku v triggeri zjednodušene zapísanú:
`IF element_id =` *ELEMENT_ID* `and` *COLUMN* `is null`
`THEN` *COLUMN* `:=` *DEFAULT*

- *ELEMENT_ID* je jednoznačný identifikátor elementu v danej schéme
- *COLUMN* je jediného atribútu UDT
- *DEFAULT* je hodnota atribútu `default`

Tým je dosiahnuté správanie popísané v tabuľke 3.11.

Element v XML dokumente	**Mapovanie v relačnej databáze**
sa nevyskytol	na miesto UDT je vložená hodnota null
sa vyskytol prázdny	je vložené UDT s hodnotou atribútu nastavenou na hodnotu atribútu `default`
sa vyskytol s hodnotou	je vložené UDT s touto hodnotou

Tabuľka 3.11: Mapovanie stavu a hodnoty elementu - `default`

Pri mapovaní atribútu `default` je potrebné sa zamyslieť nad mapovaním jednoduchých typov a komplexných typov s jednoduchým obsahom. Ak by takýto typ bol mapovaný na UDT a bola by preň vytvorená objektová tabuľka, do ktorej by boli ukladané všetky inštancie tohoto typu v XML dokumente, prejavilo by sa, že atribút `default` je obmedzením viažúcim sa priamo k elementu a nie k typu elementu.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="order" type="PurchaseOrderType">

 <xs:complexType name="PurchaseOrder">
  <xs:sequence>
   <xs:element name="shipTo" type="Country" default="SR"/>
   <xs:element name="billTo" type="Country" default="CZ"/>
  </xs:sequence>
 </xs:complexType>

 <xs:simpleType name="Country">
  <xs:restriction base="xs:string">
   <xs:length="2"/>
  </xs:restriction>
 </xs:simpleType>

</xs:schema>
```

Výpis 3.9: XML schéma 003.xsd

V XML schéme 003.xsd sa jednoduchý typ `Country` namapuje na UDT s jedným atribútom typu `varchar2(2)`. Keďže ide o globálne definovaný typ vytvorila by sa preň objektová tabuľka, do ktorej sa budú ukladať všetky inštancie tohto UDT. `PuchaseOrder` bude podobne namapovaný na UDT s objektovou tabuľkou. Elementy `shipTo` a `billTo` budú namapované na atribúty UDT pre `PurchaseOrder` a budú typu referencia na UDT pre `Country`. Integritné obmedzenie `default` nie je možné použiť v tabuľke pre `Country`, pretože nie je jasné ktorú z predvolených hodnôt "`SK`" alebo "`CZ`" zvoliť. Integritné obmedzenie `default` nie je možné použiť ani v tabuľke pre `PurchaseOrder`, nakoľko sú oba atribúty typu referencia. Nie je dokonca možné použiť ani trigger nad tabuľkou pre `PurchaseOrder`, pretože v čase vloženia záznamu do tabuľky pre `PurchaseOrder` sú oba záznamy, na ktoré sa odkazujú referencie, už vložené.

V implementácii je táto komplikácia riešená pridaním kontroly ID elementu do podmienky, na kontrolu `default` hodnoty. Pre jeden UDT teda môže existovať niekoľko podmienok na kontrolu `default` hodnoty pre rôzne elementy.

Atribút `fixed` je mapovaný rovnako, s tým rozdielom, že je pridaná podmienka, ktorá kontroluje hodnotu atribútu UDT. Aj táto podmienka obsahuje kontrolu ID elementu.

Atribút `maxOccurs` sa premietne do typu atribútu, na ktorý je uzol mapovaný:

- `maxOccurs = 1` - element je mapovaný do UDT nadradeného uzla na atribút alebo na referenciu
- `maxOccurs > 1` - element je mapovaný do UDT nadradeného uzla na pole alebo pole referencií

Atribút `minOccurs` sa do objektovo-relačnej schémy premietne v závislosti od hodnoty atribútu `maxOccurs`:

- `maxOccurs = 1`
  - `minOccurs = 0` - sa v objektovo-relačnej schéme neprejaví
  - `minOccurs = 1` - je mapovaný na podmienku (*COLUMN* `is not null`) kde *COLUMN* je meno stĺpca
- `maxOccurs > 1`
  - `minOccurs = 0` - sa v objektovo-relačnej schéme neprejaví
  - `minOccurs >= 1` - je mapovaný na podmienku (*COLUMN*`.count >= MIN_OCCURS`)
    - *COLUMN* je meno stĺpca
    - *MIN_OCCURS* je hodnota atribútu `minOccurs`

3.2 Algoritmus vytvorenia objektovo-relačnej schémy

Na vstupe sa očakáva súbor s XML schémou prípadne XML dokument s URI na XML schému, ktorá je dostupná na spracovanie. XML schéma musí spĺňať požiadavku, že jej dátový model je deterministický. Táto požiadavka sa pri spracovaní neoveruje.

3.2.1 Prípravná fáza

1. Načítaj XML schému zo súboru a vytvor DOM[4].
2. Ak schéma nie je validná, skonči s chybou.
3. Ak schéma v definícií využíva viac ako jeden priestor mien, skonči s chybou.
4. Ak už je XML schéma s týmto URI uložená v databáze, skonči s chybou.
5. Vygeneruj pre URI jednoznačný identifikátor a vlož záznam do tabuľky *xmlSchemaTable*.
6. Vytvor spoločného predka pre všetky UDT danej schémy.

3.2.2 Vytvorenie DOM grafu

1. Vytvor zoznam globálnych uzlov, do ktorého zaraď modelové skupiny, komplexné typy a elementy. Každému z nich vygeneruj jednoznačný identifikátor.
2. Do zoznamu globálnych uzlov pridaj jednoduché typy, atribúty a skupiny atribútov. Bez generovania jednoznačného identifikátora.
3. Prejdi DOM modelom do hĺbky:
   (a) Pre modelové skupiny a elementy vygeneruj jednoznačný identifikátor a ulož ho do DOM grafu. Pri lokálne definovaných komplexných typoch použi jednoznačný identifikátor nadradeného elementu.
   (b) Spracuj poduzly:
      i. Pri spracovaní elementu, ktorý má jednoduchý typ, spracuj tento typ. Výsledkom spracovania je SQL dátový typ a množina integritných obmedzení. Ak pre tento SQL dátový typ a množinu integritných obmedzení už bol vytvorený UDT
         A. namapuj element na referenciu na toto UDT
         B. inak vytvor nový UDT s príznakom pre vytvorenie objektovej tabuľky a namapuj element na referenciu na toto UDT
      ii. Pri spracovaní modelovej skupiny typu výber z uzlov nahraď poduzol s viacnásobným výskytom za modelovú skupinu typu postupnosť uzlov s jednonásobným výskytom, ktorá ako jediný poduzol obsahuje pôvodný poduzol s viacnásobným výskytom.

[4]Document Object Model - objektový model XML dokumentu.

iii. Aj koreňový element je spracovávaný ako modelová skupina typu výber z uzlov. Pre globálne uzly ale nie je povolený viacnásobný výskyt, takže nie je potrebné ich upravovať podla bodu 3.2.2.

(c) Vytvor hrany od uzla k poduzlom.

(d) Pri spracovaní XML atribútu `xs:type` vytvor spätnú hranu na globálny jednoduchý alebo komplexný typ. Na nájdenie konkrétneho typu je využitý zoznam globálnych uzlov.

(e) Pri spracovaní XML atribútu `xs:ref` vytvor spätnú hranu na globálny element, atribút, skupinu atribútov alebo modelovú skupinu. Na nájdenie konkrétneho uzlu je využitý zoznam globálnych uzlov.

3.2.3 Spracovanie DOM grafu

Pre globálne komplexné typy, globálne modelové skupiny a pre lokálne definované komplexné typy globálnych elementov vytvor neúplné UDT. Prejdi DOM grafom do hĺbky, začni spracovávať globálne komplexné typy, globálne modelové skupiny a globálne elementy. Pri spracovaní najprv rekurzívne spracuj poduzly a až následne samotný uzol.

Ďalej je uvedený postup spracovania jednotlivých typov uzlov:

Jednoduchý typ

Každý jednoduchý typ je výsledkom niekoľkých odvodení od vstavaného jednoduchého typu. Vstavaný jednoduchý typ je namapovaný priamo na niektorý SQL dátový typ a určitú množinu integritných obmedzení. Každým odvodením dochádza len k zmenám v spomenutej množine integritných obmedzení.

- Pri odvodení reštrikciou pridaj do zoznamu integritných obmedzení získaných od predka integritné obmedzenia definované pre aktuálny typ.
- Pri odvodení zoznamom vytvor prázdny zoznam integritných obmedzení a pridaj doň integritné obmedzenia definované pre aktuálny typ. Pôvodný SQL dátový typ nahraď `varchar2(4000)`.
- Pri odvodení zjednotením vytvor prázdny zoznam integritných obmedzení a pridaj doň integritné obmedzenia definované pre aktuálny typ. Pôvodný SQL dátový typ nahraď `varchar2(4000)`.

Atribút / Skupina atribútov

1. Spracuj jednoduchý typ, ktorý je atribútu pridelený.
2. Spracuj XML atribúty uzla `attribute`.
3. Ulož požadované informácie do tabuľky *xmlAtrTable*.

Skupiny atribútov spracuj postupne po jednotlivých atribútoch. Vnorenú skupinu atribútov spracuj rekurzívne.

Modelová skupina

Modelová skupina je mapovaná na samostatný UDT a jej spracovanie je rozdielne pre každý typ modelovej skupiny.

1. Spracuj poduzly modelovej skupiny.
2. Vytvor UDT pre modelovú skupinu (UDT je potomkom spoločného predka pre danú schému):
   (a) V prípade modelovej skupiny typu postupnosť uzlov je každý poduzol mapovaný na jeden atribút UDT. Typ atribútu je závislý od typu poduzla, môže ním byť UDT, referencia na UDT, pole UDT a aj pole referencií na UDT v prípadne ak je povolený viacnásobný výskyt poduzla. Poradie atribútov v UDT je zhodné s poradím v akom sú poduzly definované v DOM grafe.
   (b) V prípade modelovej skupiny typu množina uzlov je každý poduzol mapovaný na dva atribúty UDT. Prvý z nich má rovnaký význam ako v prípade modelovej skupiny typu postupnosť uzlov (typom atribútu nemôže byť pole, pretože viacnásobný výskyt poduzla nie je prípustný) a druhý slúži na uloženie poradia v akom sa jednotlivé poduzly vyskytnú v XML dokumente. Poradie atribútov v UDT je zhodné s poradím v akom sú poduzly definované v DOM grafe.
   (c) V prípade modelovej skupiny typu výber z uzlov je vytvorené UDT s jediným atribútom typu referencia na predka všetkých UDT pre danú schému.
3. Ulož príslušné informácie o modelovej skupine do tabuľky xmlElemTable.
4. Pre všetky poduzly modelovej skupiny ulož informácie do tabuľky *xmlElemElemTable*.
5. Pre všetky poduzly, ktoré sú mapované na atribút typu referencia na UDT, pole referencií na UDT alebo pre všetky poduzly modelovej skupiny typu výber z uzlov nastav príznak vytvorenia typovej tabuľky.

Komplexný typ

Komplexný typ je mapovaný na samostatný UDT a jeho spracovanie je rozdielne pre každý typ obsahu. Vo všetkých prípadoch je UDT potomkom spoločného predka pre danú schému. Potrebné informácie sú do tabuliek *xmlElemTable* a *xmlElemAtrTable* ukladané až v momente spracovania elementu, v ktorom je daný komplexný typ použitý.

1. Jednoduchý obsah
   (a) Spracuj jednoduchý typ, ktorý je mapovaný na atribút UDT. Typom atribútu môže byť len SQL dátový typ.
   (b) Spracuj atribúty komplexného typu. Každý z nich je mapovaný atribút UDT, ktoré ho typ je UDT.
   (c) Vytvor UDT pre komplexný typ.

2. Komplexný obsah

   (a) Spracuj modelovú skupinu, ktorá je mapovaná na atribút UDT. Typom atribútu môže byť UDT, referencia na UDT pole UDT ale aj pole referencií na UDT.

   (b) Spracuj atribúty komplexného typu. Každý z nich je mapovaný na atribút UDT, ktorého typom je UDT.

   (c) Vytvor UDT pre komplexný typ.

   (d) Vlož záznam do tabuľky *xmlElemElemTable*.

Element

Spracovanie elementu je závislé od dátového typu elementu:

1. Spracuj XML atribúty uzla `element`.

2. Ak má komplexný typ:

   (a) Ak má globálne definovaný komplexný typ:

      i. V čase spracovania elementu je už pre globálny komplexný typ vytvorený neúplný typ. V závislosti od poradia spracovania, už tento neúplný typ mohol byť nahradený úplným typom. Ako typ pre atribút UDT, na ktorý bude element mapovaný teda môže byť použitá referencia alebo pole referencií na UDT komplexného typu

      ii. Ulož informácie do tabuľky *xmlElemTable* a *xmlElemAtrTable*

   (b) Ak má lokálne definovaný komplexný typ

      i. Spracuj komplexný typ

      ii. Ako typ pre atribút UDT, na ktorý bude element mapovaný bude použité referencia alebo pole referencií na UDT komplexného typu

      iii. Ulož informácie do tabuľky *xmlElemTable* a *xmlElemAtrTable*

Koreňový uzol

Koreňový uzol je spracovaný rovnako ako modelová skupina typu výber z uzlov. Je preň teda vytvorené UDT a navyše objektová tabuľka. Poduzlami koreňového uzla sú globálne definované elementy. Do tabuľky *xmlSchemaTable* je do záznamu pre danú schému doplnený identifikátor koreňového uzla.

Vytvorenie tabuliek

Vytvorenie objektových tabuliek pre UDT pre uzly z DOM grafu s príznakom pre ich vytvorenie je vykonané jedným prechodom DOM grafu. Tabuľky sú doplnené o integritné obmedzenia, prípadne o trigger podľa podmienok.

Integritné obmedzenia

Každý uzol DOM grafu má definovanú množinu podmienok, ktoré sa majú premienuť do objektovo-relačnej schémy. Je nimi kontrolovaná validita uloženého XML dokumentu. Každá z podmienok má príznak, či je možné ju pridať ako integritné obmedzenie, alebo je nutné kontrolovať ju triggerom / uloženou procedúrou.

Pre všetky uzly, ktorých UDT bude uložené v objektovej tabuľke alebo bude prvkom poľa, sú nasledovným algoritmom rekurzívne zozbierané podmienky z hniezdených poduzlov. Po spracovaní uzla je do naduzla vrátená množina podmienok, ktoré majú byť v naduzle kontrolované.

1. Nastav aktuálnu množinu podmienok na prázdnu
2. Do aktuálnej množiny pridaj podmienky aktuálneho uzla
3. Pre každý poduzol, ktorý je mapovaný na UDT
   (a) Rekurzívne získaj množinu podmienok z poduzla a pridaj ju do aktuálnej množiny
4. Vráť aktuálnu množinu podmienok

Podmienky pre uzol je nevyhnutné doplniť o kontrolu nenulovosti naduzla tak, aby samotná podmienka bola kontrolovaná len v prípade ak naduzol, v ktorom je uzol hniezdený, má hodnotu rôznu od `NULL`. Inak by dochádzalo k porušeniu integritných obmedzení aj validnými XML dokumentami.

- Pre UDT, pre ktoré bola vytvorená objektová tabuľka, sú z výslednej množiny vytvorené integritné obmedzenia (CHECK), prípadne podmienky v triggeri.
- Pre UDT, ktoré je prvkom poľa je z výslednej množiny vytvorená funkcia, ktorá tieto podmienky na prvkoch poľa kontroluje. Funkcia je navrhnutá tak, že umožňuje viacnásobné hniezdenie polí.

Týmto riešením dôjde k odtrhnutiu integritného obmedzenia v XML schéme od integritného obmedzenia v relačnej databáze, na ktoré je mapované. V prípade porušenia XML schémy niektorou z aktualizačných požiadaviek, relačná databáza síce nedovolí takúto požiadavku vykonať ale užívateľovi poskytne len nič nehovoriace chybové hlásenie. Zaujímavým rozšírením tejto práce, by bolo zachovanie vzťahu (integritné obmedzenie v XML schéme - integritné obmedzenie v relačnej databáze), čo by umožnilo implementovanie pre užívateľa zrozumiteľných chybových hlásení.

Naplnenie tabuľky *xmlPathTable*

Naplnenie tabuľky *xmlPathTable* (popísaná ďalej) je možné na jeden prechod DOM grafom. Pri spracovaní uzla je do poduzlov predávaný zoznam uzlov na aktuálne spracovávanej ceste.

1. Ak je aktuálny uzol typu element

(a) Na koniec zoznamu uzlov získaného z naduzla pridaj identifikátor aktuálneho uzla

(b) Zoznam uzlov zapíš do tabuľky *xmlPathTable*

(c) Ak je typom aktuálneho uzla komplexný typ s komplexným obsahom

i. Vytvor zoznam uzlov obsahujúci len identifikátor aktuálneho uzla

ii. Rekurzívne spracuj poduzly, ktoré nie sú typu atribút

2. Inak

(a) Na koniec zoznamu uzlov získaného z naduzla pridaj identifikátor aktuálneho uzla

(b) Rekurzívne spracuj poduzly

3.3 Popis vytváranej objektovo-relačnej schémy

Pri aktualizačných operáciách sú využívané štruktúry, do ktorých je potrebné uložiť inštancie ktoréhokoľvek UDT. Z tohoto dôvodu je vytvorené UDT, ktoré je predkom všetkých UDT vo všetkých schémach. Jeho meno je "`o`" a obsahuje tri atribúty:

- `record_id` - jednoznačný identifikátor inštancie UDT
- `element_id` - jednoznačný identifikátor uzla v rámci schémy
- `udt_id` - jednoznačný identifikátor UDT v rámci schémy

Pre každú XML schému je vytvorený predok všetkých UDT v danej schéme, ktorý je potomkom `o`. Jeho meno je `"o_" +` *SCHEMA_ID* a nepridáva žiadny ďalší atribút. Toto UDT je použité v modelovej skupine výber z uzlov, kde je potrebná referencia na niekoľko UDT z danej schémy. Názvy ďalších objektov v objektovo-relačnej schéme sú:

- pre UDT uzlov - `"o_" +` *SCHEMA_ID* `+ "_" +` *UDT_ID*
- pre tabuľku pre UDT - *UDT_NAME* + `"_t"`
- pre trigger nad tabuľkou pre UDT - *TABLE_NAME* + `"_tr"`

Pri každom mapovaní vzťahu uzol poduzly, kde je využité pole je definovaný typ VARRAY s názvom *UDT_NAME* `+ "_v_" +` *ORDER*. Kde *ORDER* je poradie atribútu v tomto UDT, pre ktorý je pole použité. Typom jeho prvkov je UDT poduzla alebo referencia na UDT poduzla.

Atribúty elementov aj poduzly, ktoré sú mapované na atribúty UDT sú označované rovnako. Majú názov `"c_" +` *ORDER* kde *ORDER* je poradie atribútu. V UDT pre modelovú skupinu typu množina je poradie uzlov ukladané do atribútu s názvom `"or_" +` *ORDER*.

3.3.1 Metódy UDT pre uzly

Funkcia `c_check`

Funkcia `c_check` je do UDT pridaná pre každý atribút typu pole. Nemá žiadne vstupné parametre. Telo funkcie je tvorené podmienkami, na ktoré boli namapované integritné obmedzenia z XML schémy. Ak sú podmienky splnené pre všetky prvky poľa, návratová hodnota je 0 inak 1.

Presný názov funkcie je `"c_" + ORDER + "_check"` kde *ORDER* je poradie atribútu.

Funkcia je volaná triggerom nad tabuľkou, prípadne je volaná z funkcie `c_check` pre pole, v ktorom je UDT hniezdené. To umožňuje kontrolu integritných obmedzení aj vo viacnásobne hniezdených UDT.

Funkcia `get_tree`

Funkcia `get_tree` je definovaná pre UDT všetkých uzlov. Nemá žiadne vstupné parametre.

Typom návratovej hodnoty je postunosť kópií elementov. Kópiou elementu je myslená kópia celého podstromu XML dokumentu s koreňom v danom elemente. Ak bola funkcia zavolaná na inštancii UDT pre uzol u, vo vrátenej postunosti sú uzly v, pre ktoré platí:

- typom v je element
- ak u je typu element
  - potom u a v predstavujú ten istý uzol
  - inak u je predkom v
- na ceste medzi v a u nie je žiadny uzol typu element

Uzly v sú elementy jedného rodičovského elementu a teda sú súrodencami. Ich poradie v postupnosti je zhodné s poradím v XML dokumente.

Funkcia `get_element_children`

Funkcia `get_element_children` je definovaná pre UDT všetkých uzlov. Nemá žiadne vstupné parametre.

Typom návratovej hodnoty je postunosť referencií na UDT v štruktúre. Ak bola funkcia zavolaná na inštancii UDT pre uzol u, vo vrátenej postunosti sú uzly v, pre ktoré platí:

- typom v je element
- u je predkom v
- na ceste medzi v a u nie je žiadny uzol typu element

Uzly v sú elementy jedného rodičovského elementu a teda sú súrodencami. Ich poradie v postupnosti je zhodné s poradím v XML dokumente.

Procedúra `del_tree`

Procedúra `del_tree` je definovaná pre UDT všetkých uzlov. Nemá žiadne vstupné parametre.

Ak bola procedúra zavolaná na inštancii UDT pre uzol u, zo štruktúry zmaže inštancie UDT pre uzly v, pre ktoré platí:

- inštancia pre uzol v je uložená v objektovej tabuľke
- u je predkom v

Pri zmazaní inštancie UDT uloženej v objektovej tabuľke zákonite dôjde aj k zmazaniu inštancií UDT v ňom hniezdených.

Procedura `del_element`

Procedúra `del_element` je definovaná pre UDT všetkých uzlov. Nemá žiadne vstupné parametre.

Ak bola procedúra zavolaná na inštancii UDT pre uzol u, zo štruktúry zmaže inštancie UDT pre uzly v, pre ktoré platí:

- inštancia pre uzol v je uložená v objektovej tabuľke
- typom v je modelová skupina
- u je predkom v
- na ceste medzi v a u nie je žiadny uzol typu element

Pri zmazaní inštancie UDT uloženej v objektovej tabuľke zákonite dôjde aj k zmazaniu inštancií UDT v ňom hniezdených.

3.3.2 Pomocné tabuľky

Obsahujú informácie potrebné pre operácie nad vytvorenou objektovo-relačnou schémou.

xmlSchemaTable

Obsahuje jeden záznam pre každú podporovanú XML schému. Podporovanou schémou je myslená XML schéma, pre ktorú bola v databáze vytvorená objektovo-relačná schéma a teda je možné XML dokumenty validné voči tejto schéme ukladať do databázy.

- `uri` - cesta k popisu XML schémy, ktorá XML schému jednoznačne určuje
- `schema_id` - jednoznačný číselný identifikátor XML schémy vygenerovaný pri vytvorení objektovo-relačnej schémy
- `root_elem_id` - `udt_id` koreňového UDT

xmlDocTable

Obsahuje jeden záznam pre každý XML dokument, uložený v databáze.

- `schema_id` - číselný identifikátor schémy XML dokumentu
- `uri` - cesta k XML dokumentu, ktorá XML dokument jednoznačne určuje
- `doc_id` - jednoznačný číselný identifikátor XML dokumentu

Hodnota `doc_id` je rovná hodnote `record_id` koreňového uzla.

xmlAttrTable

Obsahuje záznamy pre všetky atribúty v podporovaných schémach.

- `schema_id` - číselný identifikátor schémy atribútu
- `attr_id` - jednoznačný číselný identifikátor atribútu v rámci schémy
- `xml_name` - meno atribútu z XML schémy
- `map_type` - jednoduchý dátový typ atribútu určený jednou zo skratiek:
  - `'s'` - reťazec
  - `'n'` - číselná hodnota
  - `'f'` - číselná hodnota s plávajúcou desatinnou čiarkou
  - `'d'` - číselná hodnota s plávajúcou desatinnou čiarkou a väčšou presnosťou
- `udt_id` - číselný identifikátor UDT, na ktorý je atribút mapovaný

xmlElemTable

Obsahuje záznamy pre všetky elementy a modelové skupiny (ďalej len uzly) v podporovaných schémach.

- `schema_id` - číselný identifikátor schémy uzla
- `elem_id` - jednoznačný číselný identifikátor uzla v rámci schémy
- `xml_name` - meno uzla z XML schémy. Hodnota `xml_name` je `null` v prípade, že ide o modelovú skupinu.
- `map_type` - dátový typ uzla určený jednou zo skratiek:
  - `'s'` - reťazec
  - `'n'` - číselná hodnota
  - `'f'` - číselná hodnota s plávajúcou desatinnou čiarkou
  - `'d'` - číselná hodnota s plávajúcou desatinnou čiarkou a väčšou presnosťou
  - `'C'` - určuje, že uzol nemá jednoduchý typ ani komplexný typ s jednoduchým obsahom. Podrobnejšie je typ uzla určený v `elem_type`.

    - `'o'` - prázdny komplexný typ

- `elem_type` - jednou zo skratiek je určené, či ide o element s komplexným obsahom alebo o jednu z modelových skupín:
    - `'e'` - element s komplexným typom
    - `'c'` - modelová skupina typu výber z uzlov
    - `'a'` - modelová skupina typu množina uzlov
    - `'s'` - modelová skupina typu postupnosť uzlov
- `udt_id` - číselný identifikátor UDT, na ktorý je uzol mapovaný

xmlElemAttrTable

Zachytáva vzťah element atribút.

- `schema_id` - číselný idenifikátor schémy atribútu a elementu
- `udt_id` - číselný identifikátor UDT, na ktorý je element mapovaný
- `attr_id` - jednoznačný číselný identifikátor atribútu
- `ord` - poradie atribútu UDT, na ktorý je atribút elementu mapovaný

xmlElemElemTable

Zachytáva vzťah medzi uzlom a jeho poduzlom.

- `schema_id` - číselný identifikátor schémy uzlov
- `udt_id` - číselný identifikátor UDT, na ktorý je uzol mapovaný
- `sub_elem_id` - jednoznačný identifikátor poduzla
- `ord` - poradie atribútu UDT, na ktorý je poduzol mapovaný
- `map_type` - skratkou určený typ mapovania vzťahu k poduzlu:
    - `'C'` - mapovanie na atribút UDT
    - `'c'` - mapovanie na referenciu
    - `'A'` - mapovanie na pole
    - `'a'` - mapovanie na pole referencií
- `min` - určuje minimálny počet výskytov poduzla
- `max` - určuje maximálny počet výskytov poduzla

xmlPathTable

Reprezentuje cesty medzi dvoma uzlami typu element cez uzly typu modelová skupina.

- `schema_id` - číselný identifikátor schémy uzlov
- `elem_id` - jednoznačný identifikátor elementu
- `sub_elem_id` - jednoznačný identifikátor podelementu
- `path` - zoznam `udt_id` uzlov, ktoré sú na ceste z elementu do podelementu

3.4 Algoritmus uloženia XML dokumentu

V pomocných tabuľkách v databáze je uložený popis XML schémy a jej mapovania na schému v objektovo-relačnej databáze. Pri ukladaní XML dokumentu sú súčasne prechádzané tieto informácie a DOM XML dokumentu. Neustále sú udržiavané dva ukazovatele:

- ukazovateľ na aktuálne spracovávaný uzol v schéme v databáze
- ukazovateľ na aktuálne spracovávaný element v DOM XML dokumentu

Prechodom do hĺbky sú najprv spracovaní potomkovia uzla a až následne samotný uzol. Výsledkom spracovania uzla je časť požiadavky do databázy, ktorou sa údaje z XML dokumentu uložia do databázy. Pri spracovaní každého uzla je pripravený konštruktor príslušného UDT.

- Pri spracovaní uzla mapovaného do naduzla na atribút/pole, je vytvorený konštruktor zodpovedajúceho UDT. Týmto konštruktorom je inicializovaný atribút alebo jeden z prvkov v konštruktore poľa.
- Pri spracovaní uzla mapovaného do naduzla na referenciu/pole referencií, je vytvorený konštruktor zodpovedajúceho UDT. Následne je týmto konštruktorom vytvorená inštancia UDT, ktorá je vložená do typovej tabuľky pre daný UDT. Výsledkom spracovania je požiadavka typu select na vloženú inštanciu UDT, ktorou bude inicializovaná hodnota referencie.

V popise XML schémy nie sú vždy uvedené presné počty výskytov jednotlivých uzlov, najčastejšie je uvedený prípustný rozsah. Pri spracovaní XML dokumentu teda nie je dopredu známe koľko krát sa jednotlivé elementy v dokumente vyskytnú, dokonca pri niektorých nie je ani známe či sa vôbec v dokumente vyskytnú. Spracovanie dokumentu to jemne komplikuje, ale vďaka predpokladu, že XML schéma má deterministický dátový model je prvé namapovanie elementu na uzol z popisu XML schémy, spĺňajúce podmienky, definitívne a spracovanie ďalších uzlov ho už nemôže zmeniť ani zrušiť.

3.4.1 Spracovanie dokumentu

1. Načítaj XML dokument zo súboru a vytvor DOM.
2. Ak sa v databáze našiel XML dokument so zhodným URI, skonči s chybou.
3. Z dokumentu získaj URI XML schémy.
4. Ak sa v databáze nenašla XML schéma pre URI XML schémy, skonči s chybou.
5. Ak dokument nie je validný vočí schéme, skonči s chybou.
6. Získaj popis XML schémy z tabuľky `xmlElemTable` pre koreňový uzol.
7. Zavolaj **Spracovanie uzla** pre koreňový uzol a koreňový element spracovávaného dokumentu.
8. Konštruktorom z návratovej hodnoty vytvor inštanciu UDT a ulož ju do objektovej tabuľky pre UDT koreňového uzla.
9. Vlož záznam do tabuľky *xmlDocTable*. Jednoznačným identifikátorom dokumentu je `record_id` koreňového uzla.

3.4.2 Spracovanie uzla

Vstupnými hodnotami je ukazovateľ na uzol v popise XML schémy, v algoritme označovaný ako aktuálny uzol a ukazovateľ na element v DOM, označovaný ako aktuálny element.

Výstupnou hodnotou je požiadavka do databázy na vytvorenie inštancie UDT (konštruktor) alebo požiadavka na získanie referencie na už vytvorenú inštanciu UDT uloženú v objektovej tabuľke.

1. Podľa typu aktuálneho uzla vytvor konštruktor UDT:
   (a) Ak je ním element:
      i. Ak je ukazovateľ na aktuálny element prázdny, ukonči spracovanie neúspechom.
      ii. Ak meno aktuálneho elementu a meno aktuálneho uzla nie je zhodné, ukonči spracovanie neúspechom.
      iii. Podľa typu obsahu aktuálneho uzla:
         A. Ak ide o jednoduchý typ obsahu na príslušné miesto konštruktoru UDT ulož hodnotu získanú z aktuálneho elementu.
         B. Ak ide o komplexný obsah, pre ukazovateľ na aktuálny element nastavený na prvého potomka aktuálneho elementu, spracuj element rovnako ako modelovú skupinu typu postupnosť uzlov, teda podľa bodu 1.(d).
      iv. Spracuj atribúty aktuálneho elementu nasledujúcim postupom:
         A. Získaj zoznam atribútov z popisu XML schémy z DB pre komplexný typ aktuálneho uzla.

B. Pre každý atribút zo zoznamu ulož na príslušné miesto v konštruktore UDT konštruktor UDT pre atribút inicializovaný hodnotou získanou z atribútu aktuálneho elementu. Ak sa atribút v aktuálnom elemente nevyskytuje, ulož na príslušné miesto v konštruktore UDT hodnotu `NULL`.

v. Posuň ukazovateľ na aktuálny element na jeho nasledujúceho súrodenca. Ak taký neexistuje nastav ukazovateľ na prázdny.

(b) Ak je ním modelová skupina typu množina:

i. Získaj z popisu XML schémy v DB zoznam potomkov aktuálneho uzla (zoradené podľa atribútu `ord`).

ii. Označ prvý nespracovaný uzol zo zoznamu potomkov za aktuálneho potomka. Ak taký neexistuje na neobsadené miesta v konštruktore UDT doplň hodnotu `NULL` a tým skončilo spracovanie podľa bodu 1.(b).

iii. Rekurzívne spracuj aktuálny element a aktuálneho potomka.

iv. Ak spracovanie skončilo úspechom:

A. Označ aktuálneho potomka za spracovaného, ulož výsledok spracovania na príslušné miesto v konštruktore UDT spolu s poradovým číslom jeho výskytu a pokračuj bodom 1.(b).ii.

v. Ak spracovanie skončilo neúspechom:

A. Označ nasledujúci nespracovaný uzol zo zoznamu potomkov za aktuálneho potomka a pokračuj bodom 1.(b).iii. Ak taký neexistuje na neobsadené miesta v konštruktore UDT doplň hodnotu `NULL` a tým skončilo spracovanie podľa bodu 1.(b).

(c) Ak je ním modelová skupina typu výber z uzlov:

i. Získaj z popisu XML schémy v DB zoznam potomkov aktuálneho uzla (zoradené podľa atribútu `ord`).

ii. Označ prvý uzol zo zoznamu potomkov za aktuálneho potomka.

iii. Rekurzívne spracuj aktuálny element a aktuálneho potomka.

iv. Ak spracovanie skončilo úspechom:

A. Výsledok spracovania ulož na príslušné miesto v konštruktore UDT.

v. Ak spracovanie skončilo neúspechom:

A. Označ nasledujúci uzol zo zoznamu potomkov za aktuálneho potomka a pokračuj bodom 1.(c).iii. Ak taký neexistuje ukonči spracovanie neúspechom.

(d) Ak je ním modelová skupina typu postupnosť uzlov:

i. Získaj z popisu XML schémy v DB zoznam potomkov aktuálneho uzla (zoradené podľa atribútu `ord`).

ii. Označ nasledujúci uzol zo zoznamu potomkov za aktuálneho potomka. Ak taký neexistuje spracovanie podľa bodu 1.(d). skončilo.

iii. Rekurzívne spracuj aktuálny element a aktuálneho potomka.

iv. Ak spracovanie skončilo úspechom:

A. Ak je pre aktuálneho potomka v popise XML schémy v DB povolený viacnásobný výskyt, je mapovaný na atribút UDT typu pole UDT alebo pole referencií na UDT. Výsledok spracovania teda ulož do konštruktoru príslušného poľa. Ak počet prvkov v konštruktore poľa dosiahol maximálnu povolenú hodnotu, pokračuj bodom 1.(d).v.A. Inak bodom 1.(d).iii.

B. Ak nie je pre aktuálneho potomka v popise XML schémy v DB povolený viacnásobný výskyt, výsledok spracovania ulož na príslušné miesto v konštruktore UDT a pokračuj bodom 1.(d).ii.

v. Ak spracovanie skončilo neúspechom:

A. Ak je pre aktuálneho potomka v popise XML schémy v DB povolený viacnásobný výskyt ulož konštruktor poľa na príslušné miesto v konštruktore UDT a pokračuj bodom 1.(d).ii.

B. Ak nie je pre aktuálneho potomka v popise XML schémy v DB povolený viacnásobný výskyt. na príslušné miesto v konštruktore UDT ulož hodnotou `NULL`.

2. Ak všetky parametre konštruktoru UDT sú inicializované na hodnotu `NULL` alebo prázdny konštruktor poľa, ukonči spracovanie neúspechom.

3. Podľa typu mapovania aktuálneho uzla do naduzla vykonaj:

   (a) Ak ide o mapovanie do naduzla na atribút alebo pole, vráť konštruktor UDT.

   (b) Ak ide o mapovanie do naduzla na referenciu alebo pole referencií. Konštruktorom vytvorenú inštanciu UDT ulož do objektovej tabuľky a vráť požiadavku typu select na referenciu na túto inštanciu UDT.

4. Aktualizácia XML dát

V tejto kapitole je popísaný algoritmus pre aktualizáciu XML dokumentov uložených v podkladovej štruktúre popísanej v predchádzajúcej kapitole.

4.1 Gramatika

Pre potreby tejto diplomovej práce bola zvolená nie príliš veľká podmnožina jazyka XQuery 1.0 s podporou aktualizácií. Navrhnutá gramatika musela zohľadniť možnosti podkladovej štruktúry ale aj možnosti jazyka SQL, ktorý je pre vyhodnotenie požiadaviek tejto gramatiky použitý. Syntax tohoto jazyka je popísaná gramatikou v prílohe 1.

Gramatika popisuje aktualizačnú požiadavku, ktorej základom sú výrazy `FOR`, `LET`, `WHERE` a `RETURN`. Z pätice výrazov `FLWOR` v jazyku XQuery sa tak stratil výraz `ORDER BY`. Aktualizačná požiadavka môže obsahovať jeden alebo viacej základných aktualizačných výrazov.

4.1.1 Základné aktualizačné výrazy

Insert

Základný aktualizačný výraz `insert` má tri formy popísané nižšie. Slúži na vloženie nových uzlov do XML dokumentu. Má dva vstupné parametre `target` a `source`. Prvý určuje postupnosť cieľových uzlov a druhý postupnosť zdrojových uzlov. Syntax výrazu `insert`:

```
InsertExpr ::= "insert" ("node" | "nodes") $source
  InsertExprTargetChoice $target
InsertExprTargetChoice ::= (("as" ("first" | "last"))? "into")
    | "after"
    | "before"
```

`insert into`

- Postupnosť `$target` môže obsahovať maximálne jeden uzol. Ak je prázdna, `insert` sa nevykoná. Ak obsahuje viac ako jeden uzol, požiadavka skončí s chybou. Povolené typy uzlov sú:
  - Ak `$source` obsahuje atribúty, sú povolené všetky typy elementov.
  - Ak `$source` obsahuje elementy, sú povolené len elementy s komplexným typom s komplexným obsahom.
- Postupnosť `$source` môže obsahovať žiadny alebo viacero uzlov. Všetky uzly v postupnosti musia byť typu atribút alebo všetky uzly v postupnosti musia byť typu element.

`insert as first into`, `insert as last into`

- Postupnosť `$target` môže obsahovať maximálne jeden uzol. Ak je prázdna, `insert` sa nevykoná. Ak obsahuje viac ako jeden uzol, požiadavka skončí s chybou. Povolené typy uzlov sú elementy s komplexným typom s komplexným obsahom.
- Postupnosť `$source` môže obsahovať žiadny alebo viacero uzlov. Všetky uzly v postupnosti musia byť typu element.

`insert before`, `insert after`

- Postupnosť `$target` môže obsahovať maximálne jeden uzol. Ak je prázdna, `insert` sa nevykoná. Ak obsahuje viac ako jeden uzol požiadavka skončí s chybou. Povolené typy uzlov sú elementy.
- Postupnosť `$source` môže obsahovať žiadny alebo viacero uzlov. Všetky uzly v postupnosti musia byť typu element.

Delete

Základný aktualizačný výraz `delete` slúži na zmazanie uzlov z XML dokumentu. Má jeden vstupný parameter `$target`, ktorý určuje postupnosť cieľových uzlov. Syntax výrazu `delete`:

```
DeleteExpr ::= "delete" ("node" | "nodes") $target
```

- Postupnosť `$target` môže obsahovať žiadny alebo viacero uzlov. Ak je postupnosť prázdna `delete` sa nevykoná. Povolené typy uzlov sú atribúty aj elementy.

Replace

Základný aktualizačný výraz `replace` má dve formy. Syntax výrazu `replace`:

```
ReplaceExpr ::= "replace" ("value" "of")? "node" $target
  "with" $source
```

`replace` Forma `replace` slúži na nahradenie uzla v XML dokumente novými uzlami. Má vstupné parametre `target` a `source`. Prvý určuje postupnosť cieľových uzlov a druhý postupnosť zdrojových uzlov.

- Postupnosť `$target` môže obsahovať maximálne jeden uzol. Ak je prázdna, `replace node` sa nevykoná. Ak obsahuje viac ako jeden uzol, požiadavka skončí s chybou. Povolené typy uzlov sú atribúty aj elementy.
- Postupnosť `$source` môže obsahovať žiadny alebo viacero uzlov. Ak je prázdna dôjde len k zmazaniu cieľového uzla.
    - Ak je cieľovým uzlom atribút, všetky zdrojové uzly musia byť typu atribút.

    - Ak je cieľovým uzlom element, všetky zdrojové uzly musia byť typu element.

- V rámci jednej aktualizačnej požiadavky môže byť uzol cieľovým uzlom základného aktualizačného výrazu `replace`, forma `replace node`, iba raz, inak aktualizačná požiadavka skončí s chybou.

`replace value of` Forma `replace value of` slúži na nahradenie hodnoty uzla pri zachovaní jeho identity. Má vstupné parametre `$target` a `$source`. Prvý určuje postupnosť cieľových uzlov a druhý postupnosť zdrojových uzlov.

- Postupnosť `$target` môže obsahovať maximálne jeden uzol. Ak je prázdna, `replace value` sa nevykoná. Ak obsahuje viac ako jeden uzol, požiadavka skončí s chybou. Povolené typy uzlov sú atribút, element s jednoduchým typom a element s komplexným typom s jednoduchým obsahom.

- Postupnosť `$source` môže obsahovať žiadny alebo viacero uzlov. Typom uzlov je jednoduchá hodnota, uzol typu atribút, element s jednoduchým typom alebo element s komplexným typom s jednoduchým obsahom. Hodnoty a hodnoty uzlov z tejto postupnosti sú spojené do výslednej textovej hodnoty oddelené medzerami.

- V rámci jednej aktualizačnej požiadavky môže byť uzol cieľovým uzlom základného aktualizačného výrazu replace, forma `replace value of`, iba raz, inak aktualizačná požiadavka skončí s chybou.

Typy aktualizačných výrazov sú: `insert`, `insert as first`, `insert as last`, `insert before`, `insert after`, `delete`, `replace node`, `replace value`.

4.2 Dátový model

Dátový model navrhnutého jazyka je neporovnateľne jednoduchší ako dátový model XQuery. Je to dané tým, že časť aktualizačnej požiadavky, ktorou sú určené vstupné hodnoty pre základné aktualizačné výrazy, je mapovaná na jednu požiadavku typu `select` z SQL. Všetky operátory v aktualizačnej požiadavke sú mapované na operátory SQL a na relačnú databázu boli ponechané aj pretypovania medzi dátový typmi.

BOOLEAN

Relačná databáza Oracle 10.2 nepodporuje dátový typ `BOOLEAN`. Nie sú preň preto definované žiadne literály ani pretypovania na iné typy. Jediná možnosť ako hodnotu typu `BOOLEAN` vytvoriť je ako výsledok vyhodnotenia niektorého z operátorov `ValueComp`.

VALUE

Je súhrnné označenie pre jednoduché dátové typy podporované podkladovým úložiskom. Pri spracovaní požiadavky sa medzi jednotlivými typmi z `VALUE` nerozlišuje. Pretypovania medzi týmito typmi sú ponechané čisto na relačnú databázu. `VALUE` zahňuje tieto štyri typy:

- `VARCHAR2`
- `NUMBER`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`

Pri spracovaní požiadavky môže byť hodnota typu `VALUE` pretypovaná na postupnosť zdrojových uzlov. Je to dosiahnuté úpravou výrazu `select` tak, aby vrátil postupnosť zdrojových uzlov s jedným zdrojovým uzlom typu text s touto hodnotou.

Postupnosť uzlov

Postupnosť uzlov je výsledkom vyhodnotenia výrazu `XPath`. Môže byť pretypovaná na `VALUE`, čo je zabezpečené nasledovným postupom:

- Pri spracovaní požiadavky:
  - Ak postupnosť bude obsahovať len uzly s jednoduchým dátovým typom (toto je možné overiť na základe informácií o XML schéme uložených v pomocných tabuľkách):
    - Potom uprav výraz `select` pre výraz `XPath` tak, aby vrátil hodnoty uzlov.
    - Inak skonči s chybou.
- Pri vyhodnotení požiadavky v SQL:
  - Ak je výsledkom takéhoto výrazu select viac ako jedna hodnota relačná databáza ukončí vyhodnotenie s chybou.

Pri spracovaní požiadavky môže byť postupnosť uzlov pretypovaná na postupnosť cieľových uzlov, prípadne na postupnosť zdrojových uzlov.

Postupnosť cieľových uzlov

Postupnosť cieľových uzlov je typom parametru `$target` pre základné aktualizačné výrazy.

Cieľový uzol musí jednoznačne určovať uzol v podkladovej štruktúre a tiež musí obsahovať potrebné údaje na vykonanie zmien. V riešení sa vyskytujú dva typy cieľových uzlov:

- Cieľový uzol pre atribút. Musí umožňovať aktualizovanie obsahu uzla ale aj zmazanie uzla samotného. Na to je potrebné SQL požiadavkou typu `update` aktualizovať stĺpec objektovej tabuľky, v ktorej je inštancia UDT pre rodičovský element atribútu uložená. Na vygenerovanie takejto požiadavky sú potrebné tieto údaje:
  - `schema_id` - jednoznačný identifikátor schémy
  - `parent_udt_id` - identifikátor UDT rodičovského elementu

- `parent_record_id` - jednoznačný identifikátor inštancie UDT v úložisku
- `order` - poradie atribútu UDT, na ktorý je cieľový atribút mapovaný
- `attribut_udt_id` - identifikátor UDT atribútu
- `attribut_record_id` - jednoznačný identifikátor inštancie UDT v úložisku

- Cieľový uzol pre element. Musí umožňovať získanie postupnosti všetkých potomkov rodičovského uzla a jej prípadné vloženie. Musí tiež umožňovať aktualizovanie obsahu uzla. Na to je potrebné SQL požiadavkou typu `update` aktualizovať stĺpce objektovej tabuľky, v ktorej je inštancia UDT pre cieľový element uložená. Cieľový uzol pre element obsahuje:
  - `schema_id` - jednoznačný identifikátor schémy
  - `parent_udt` - inštancia UDT rodičovského elementu
  - `udt` - inštancia UDT cieľového uzla

Postupnosť zdrojových uzlov

Postupnosť zdrojových uzlov je typom parametru `$source` pre základné aktualizačné výrazy.

Zdrojový uzol reprezentuje podstrom XML dokumentu a je jedného z troch typov:

- `hodnota` - textový reťazec
- `atribút` - obsahuje hodnoty:
  - meno atribútu
  - samotnú hodnotu atribútu
- `element` - obsahuje hodnoty:
  - meno elementu
  - postupnosť atribútov
  - podľa typu elementu buď hodnotu elementu alebo postupnosť podelementov

Zdrojový uzol neobsahuje žiadny odkaz na uzly v podkladovom úložisku. Ak aj bol v aktualizačnej požiadavke určený výrazom `XPath` predstavuje kópiu podstromu XML dokumentu určeného týmto výrazom. Táto kópia je vytvorená pri vyhodnotení SQL požiadavky typu `select` v druhej fáze pomocou funkcie `get_tree` definovanej pre UDT uzla.

4.3 Vyhodnotenie aktualizačnej požiadavky

Vyhodnotenie aktualizačnej požiadavky je rozdelené do troch fáz. V prvých dvoch fázach nedochádza k žiadnej zmene v podkladovom úložisku XML dokumentov. Je tým zaručené, že spracovanie jednotlivých základných aktualizačných výrazov je na sebe nezávisle a na poradí ich spracovania teda nezáleží.

V prvej fáze sú na základe informácií o XML schémach uložených v relačnej databáze, vygenerované SQL požiadavky. Pre každý základný aktualizačný výraz je vygenerovaný práve jeden výraz `select`. Môže byť tvorený niekoľkými výrazmi `select`, zlúčenými operátorom `UNION ALL`. Jeho vyhodnotením v relačnej databáze sa získajú hodnoty vstupných parameterov pre základné aktualizačné výrazy. Takto vytvorený výraz `select` zabezpečuje, okrem iného, iterovanie cez všetky hodnoty premennej definovanej výrazom `ForExpr`, ale aj vyhodnotenie výrazov pre podporované dátové typy. Výsledkom prvej fázy sú dvojice typ základného aktualizačného výrazu a príslušný výraz `select`. Tieto dvojice sú predané ako vstup uloženej procedúre v relačnej databáze.

V druhej fáze uložená procedúra pre každú dvojicu najprv vykoná SQL požiadavku. Pre každý jeden riadok odstráni nulové vstupné uzly a overí ich počty. Do zoznamu čakajúcich aktualizačných primitív je pridané jedno aktualizačné primitívum pre každý riadok vstupu. Vyhodnotí konflikty aktualizačných primitív.

V tretej fáze sú aktualizačné primitíva zoradené. Pre každé jedno primitívum je overené, či vstupné uzly spĺňajú požadované podmienky. Nakoniec je zmena reprezentovaná aktualizačným primitívom vykonaná nad podkladovým úložiskom XML dokumentov v relačnej databáze.

4.4 Prvá fáza

Pre určenie elementu v kolekcii XML dokumentov slúži výraz `XPath` popísaný pravidlom `XPathExpr` gramatiky. Mapovanie tohoto výrazu, do jazyka SQL tvorí podstatnú časť celého procesu mapovania. Výraz `select` je generovaný na základe popisu XML schémy uloženého v pomocných tabuľkách. Pri spracovaní požiadavky sa pracuje s reprezentáciou výrazu `select`, ďalej označovaná len ako reprezentácia. Výsledkom spracovania jedného výrazu `XPath` môže byť nula a viac výrazov `select`. Pri spracovaní je teda udržiavaná kolekcia reprezentácií, ďalej označovaná len ako kolekcia.

Pri mapovaní aktualizačnej požiadavky sú najprv spracované výrazy `XPath` pre všetky premenné, v poradí ako sú zadefinované v požiadavke. Výsledkom spracovania je pre každú premennú kolekcia reprezentácií výrazov `select`. Následne sú spracované výrazy `XPath` vo vstupných parametroch aktualizačných výrazov, tiež do podoby kolekcií. Z týchto kolekcií je nakoniec pre každý základný aktualizačný výraz vytvorený jeden výsledný výraz `select`.

Vo výslednom výraze `select` sú premenné zadefinované výrazom `ForExpr` premietnuté na tabuľky. Spojením týchto tabuliek relačná databáza zabezpečí iterovanie cez kombinácie hodnôt premenných zadefinovaných výrazom `ForExpr`. Výrazy `XPath` a `SourceExpr` vo vstupných parametroch základných aktualizačných výrazov sú ako vnorené výrazy `select` premietnuté na stĺpce výsledného výrazu `select`. Pomocou funkcie `collect` je zabezpečené, že výsledkom takýchto

výrazov je postupnosť hodnôt. Uložené procedúry reprezentujúce základné aktualizačné výrazy takéto postupnosti očakávajú ako vstupné parametre.

Vo výrazoch select sú využité štandardné operátory (`+ - / *`) pre vyhodnocovanie výrazov. Tie nepodporujú postupnosti ako operandy. Ak je teda operandom vnorený výraz `select`, ten môže vrátiť len nula alebo jednu hodnotu inak vyhodnotenie výrazu `select` relačnou databázou skončí s chybou.

Počas spracovania sú pre každú kolekciu výrazov `select` udržiavané tieto hodnoty:

- *variantsFor* - množina nadradených kolekcií reprezentácií výrazov `select`.

Pre každú reprezentáciu výrazu `select` sú počas spracovania udržiavané tieto hodnoty:

- *column* - časť výrazu `select` popisújúca stĺpce tabuľky.
- *table* - časť výrazu `select` popisújúca tabuľky.
- *condition* - časť výrazu `select` popisújúca podmienky výrazu.
- *node* - ukazovateľ na spracovaný uzol.
- *variantsFor* - množina nadradených reprezentácií výrazov `select`.

Počas spracovania je udržiavaný globálny zoznam definovaných premenných. Zoznam pre každú premennú obsahuje jej meno a kolekciu reprezentácií výrazov `select`.

4.4.1 Vytvorenie kolekcií reprezentácií výrazov select

Spracovanie ForClause

Pri spracovaní dochádza k pridaniu ďalších premenných do globálneho zoznamu definovaných premenných.

1. Postupne spracuj všetky výrazy `for` oddelené čiarkami:
   (a) Pre spracovanie výrazu `XPath` zavolaj **Spracovanie XPathExpr** a výslednú kolekciu označ R.
   (b) Do $R.variantsFor$ pridaj R.
   (c) Pre všetky reprezentácie r z R:
      i. Do $r.variantsFor$ pridaj r.
   (d) Do zoznamu definovaných premenných pridaj dvojicu `VarName` a R.

Syntax výrazu `ForClause`:

```
ForClause ::= "for" "$" VarName "in" XPathExpr
  ("," "$" VarName "in" XPathExpr)*
```

Spracovanie LetClause

Pri spracovaní dochádza k pridaniu ďalších premenných do globálneho zoznamu definovaných premenných.

1. Postupne spracuj všetky výrazy `let` oddelené čiarkami:
   (a) Pre spracovanie výrazu `XPath` zavolaj **Spracovanie XPathExpr** a výslednú kolekciu označ R.
   (b) Do zoznamu definovaných premenných pridaj dvojicu `VarName` a R.

Syntax výrazu `LetClause`:

```
LetClause ::= "let" "$" VarName ":=" XPathExpr
  ("," "$" VarName ":=" XPathExpr)*
```

Spracovanie XPathExpr

Je vytvorená nová kolekcia reprezentácií R, ktorá je výsledkom spracovania.

1. Spracuj začiatok výrazu:
   (a) Ak výraz začína funkciou `fn:doc`
      i. Vytvor prázdnu kolekciu R
      ii. V tabuľke *xmlDocTable* nájdi dokument jednoznačne identifikovaný URI uvedeným vo funkcii `fn:doc`.
      iii. Ak sa dokument nenašiel ukonči spracovanie, vráť R.
      iv. Vlož do R reprezentáciu výrazu `select` s takto nastavenými premennými:
         A. *column* nastav na `value(`*ALIAS*`)`
         B. *table* nastav na *TABLE ALIAS*
         C. *condition* nastav na *ALIAS*`.record_id =` *DOCUMENT_ID*
         D. *node* nastav na koreňový uzol pre schému nájdeného dokumentu
            - *ALIAS* je unikátny identifikátor tabuľky vrámci celej požiadavky.
            - *TABLE* je meno tabuľky, v ktorej je uložená inštancia UDT, na ktorý je mapovaný koreňový uzol schémy.
            - *DOCUMENT_ID* unikátny identifikátor dokumentu a zároveň identifikátor inštancie UDT reprezentujúcej daný dokument.
   (b) Ak výraz začína znakom '$':
      i. V zozname zadefinovaných premenných nájdi `VarName`.
      ii. Ak sa `VarName` nepodarilo nájsť ukonči spracovanie s chybou (chybne zadaná požiadavka).
      iii. Vytvor kópiu kolekcie reprezentácií nájdenej premennej a označ ju R.
      iv. Ak bola premenná zadefinovaná pravidlom `ForClause`:

A. Každej reprezentácií v R nastav *table* a *condition* na prázdny reťazec.

2. Postupne spracuj všetky výrazy `StepExpr` vo výraze `XPathExpr`:
   (a) Všetky reprezentácie v R označ za nespracované.
   (b) Kým je v R nespracovaná reprezentácia:
      i. Ďalšiu nespracovanú reprezentáciu v R označ ako r.
      ii. Pokračuj v **Spracuj QName v pravidle StepExpr** pre r a R.
      iii. r označ za spracovaný.
   (c) Ak výraz `StepExpr` obsahuje výraz `Condition`:
      i. Spracovanie výrazu `Condition` do podoby kolekcie reprezentácia výrazov `select` a jej spojenie s R sú popísané v časti **Spájanie kolekcií reprezentácií výrazov select**.

Syntax výrazu `XPath`:

```
XPathExpr ::=   "fn:doc" "(" URILiteral ")" ("/" StepExpr)+
        ("/" "@" StepExpr)?
    | "$" VarName ("/" StepExpr)* ("/" "@" StepExpr)?
```

Spracovanie QName v StepExpr

Spracovanie sa vykonáva pre reprezentáciu r a kolekciu R, do ktorej r patrí.

1. Ak je pred `QName` znak '@':
   (a) V tabuľkách *xmlAttrTable* a *xmlElemAttrTable* nájdi atribút pre *r.node* označený `QName`.
   (b) Ak žiadny atribút nebol nájdený, odober r z R.
   (c) Inak:
      i. K *r.column* pridaj `.c_`*ORD*
         - *ORD* je poradie atribútu UDT, na ktorý je atribút mapovaný
2. Inak:
   (a) V tabuľke *xmlPathTable* nájdi postupnosť uzlov predstavujúcu cestu z *r.node* k elementu označenému `QName`.
   (b) Ak žiadna cesta nebola nájdená, odober r z V.
   (c) Inak:
      i. Pre druhú a ďalšiu nájdenú postupnosť:
         A. Vytvor kópiu r označ ju s a vlož ju do R.
         B. Zavolaj **Spracovanie postupnosti uzlov** pre s postupnosť uzlov.
      ii. Zavolaj **Spracovanie postupnosti uzlov** pre r a prvú postupnosť.
3. Vráť R.

Syntax výrazu `StepExpr`:

```
StepExpr ::= QName ( "\[" Condition "\]" )?
```

Spracovanie postupnosti uzlov

Spracovanie sa vykonáva pre reprezentáciu výrazu `select` r a postupnosť uzlov. Pri spracovaní dochádza k úprave hodnôt reprezentácie. Uzly postupnosti reprezentujú uzly XML schémy. Postupnosť reprezentuje cestu cez pomocné uzly typu modelová skupina k uzlu typu element.

1. Prvý uzol postupnosti označ ako u.
2. Spracuj u:
   (a) Ak je u modelová skupina typu výber z uzlov:
      i. K *r.condition* pridaj `and` *r.column*`.c_`*ORD*`.udt_id =` *UDT_ID*
      ii. Na začiatok *r.column* vlož `treat(`
      iii. K *r.column* pridaj `.c_`*ORD* `as ref` *UDT_NAME*`)`
         - *ORD* je poradie atribútu, na ktorý je referencia na u mapovaná
         - *UDT_ID* je jednoznačný identifikátor UDT, na ktorý je u mapovaný.
         - *UDT_NAME* je meno UDT, na ktorý je u mapovaný.
   (b) Ak je u mapovaný na pole alebo pole referencií:
      i. K *r.table* pridaj `, table(`*r.column*`.c_`*ORD*`)` *ALIAS*
      ii. *r.column* nastav na `value(`*ALIAS*`)`
         - *ORD* je poradie atribútu, na ktorý je pole mapované.
         - *ALIAS* je jednoznačný identifikátor tabuľky v rámci celej požiadavky.
   (c) Inak k *r.column* pridaj `.c_`*ORD*
      - *ORD* je poradie atribútu, na ktorý je u mapovaný.
3. Nastav *r.node* na u.
4. Ak je v postupnosti ďalší uzol, označ ho ako u a pokračuj v bode 2.

Spracovanie Value

Je vytvorená nová kolekcia reprezentácií R, ktorá je výsledkom spracovania.

1. Spracuj výraz podľa druhu:
   (a) Pre výraz `CurrentValue`:
      i. Označ X kolekciu reprezentácií pre výraz `XPath`, voči ktorému je výraz `CurrentValue` relatívny.
      ii. Vytvor kópiu X a označ ju R.
      iii. Do *R.variantsFor* pridaj X.
      iv. Pre všetky reprezentácie r z R:
         A. Do *r.variantsFor* pridaj reprezentáciu, z ktorej bola skopírovaná.

B. Nastav *r.table* a *r.condition* na prázdny reťazec.

(b) Pre výraz `RelativeXPathExpr`:

i. Označ X kolekciu výrazov pre výraz `XPath`, voči ktorému je výraz `RelativeXPathExpr` relatívny.

ii. Vytvor kópiu X a označ ju R.

iii. Do *R.variantsFor* pridaj X.

iv. Pre všetky reprezentácie r z R:

A. Do *r.variantsFor* pridaj reprezentáciu, z ktorej bola skopírovaná.

B. Nastav *r.table* a *r.condition* na prázdny reťazec.

v. V spracovaní pokračuj v bode 2. v **Spracovanie XPathExpr**.

(c) Pre výraz `XPathExpr`:

i. Výraz `XPath` spracuj podľa **Spracovanie XPathExpr**

(d) Pre výraz `Literal`:

i. Vytvor prázdnu kolekciu reprezentácií výrazov `select` R.

ii. Vlož do R reprezentáciu výrazu `select` s takto nastavenými premennými:

A. *column* nastav na hodnotu literalu.

B. Všetky ostatné premenné nechaj prázdne.

Syntax výrazu `Value`:

```
ValueExpr ::= Value
    | ParenthesizedExpr
```

Vyhodnotením týchto pravidiel sú z informácií o XML schémach uložených v databáze vytvorené reprezentácie výrazov `select` a ich kolekcie. Pri vyhodnocovaní ďalších pravidiel dochádza už len k spájaniu týchto kolekcií a k spájaniu týchto reprezentácií do výslednej požiadavky typu `select`.

4.4.2 Spájanie kolekcií reprezentácií výrazov select

Kolekcie reprezentácií výrazov `select` je potrebné vzájomne spájať. Pri spájaní nedochádza vždy ku kartézkemu súčinu reprezentácií, ktoré obsahujú. K spájaniu dochádza napríklad pri spracovaní operátorov vo výraze `OrExr` alebo pri spracovaní výrazu `Condition` vo výraze `StepExpr`.

Ku každej kolekcii reprezentácií výrazov `select` je udržiavaná množina nadradených kolekcií reprezentácií výrazov `select`, každú z nich označme nadradenou kolekciou. Môžu existovať dva druhy nadradených kolekcií:

- Ak kolekcia reprezentuje výraz `XPath`, ktorý obsahuje premennú zadefinovanú pravidlom `ForExpr`, jej nadradenou kolekciou je kolekcia reprezentácií výrazov `select` pre túto premennú.

- Ak kolekcia reprezentuje relatívny výraz `XPath`, jej nadradenou kolekciou je kolekcia reprezentácií výrazov `select` pre výraz `XPath`, ku ktorému je relatívny.

Pre každú reprezentáciu z kolekcie existuje práve jedna nadradená reprezentácia z každej jej nadradenej kolekcie. Ku každej reprezentácií z aktuálnej kolekcie je udržiavaná množina reprezentácií.

Po spracovaní výrazu `XPath` môže kolekcia reprezentácií obsahovať niekoľko reprezentácií so zhodnými množinami nadradených reprezentácií. Pre spájenie kolekcií je nevyhnutné, aby žiadne dve reprezentácie v jednej kolekcii nemali zhodné množiny nadradených reprezentácií. Pred prvým spojením s inou kolekciou je teda potrebné reprezentácie výrazov select, ktoré majú zhodné množiny nadradených reprezentácií, spojiť do jednej reprezentácie výrazu `select`.

Uprav na column

Vstupom je kolekcia R, ktorá je spracovaním upravená.

Uprav na column ako postupnosť

1. Pre všetky reprezentácie r z kolekcie R:
   (a) Odstráň r z R.
   (b) Vlož do R reprezentáciu výrazu `select` s takto nastavenými premennými:
   - *variantsFor* nastav na *r.variantsFor*
   - *column* nastav na `(select collect(`*r.column*`) from` *r.table* `where` *r.condition*`)`
     - Kde `collect` je funkcia relačnej databázy, ktorá zreťazí výsledky výrazu select do postupnosti.
2. Kým v R existuje dvojica reprezentácií r1 a r2 so zhodnými nadradenými množinami:
   (a) Odstráň r1 a r2 z R.
   (b) Vlož do R reprezentáciu výrazu select s takto nastavenými premennými:
   - variantsFor nastav na r1.variantsFor
   - column nastav na concat(r1.column, r2.column)
     - Kde concat je funkcia na zreťazenie postupností.

Uprav na column ako hodnotu

1. Z reprezentácií z kolekcie R so zhodnou množinou nadradených reprezentácií vytvor n-tice.
2. Pre každú n-ticu (príklad pre trojicu reprezentácií r, s, t; pre n-tice s iným počtom prvkov obdobne):
   (a) Odstráň r, s, t z R.
   (b) Vlož do R reprezentáciu výrazu `select` s takto nastavenými premennými:

- *variantsFor* nastav na *r.variantsFor*
- *column* nastav na

```
(
  select r.column from r.table where r.condition
    UNION ALL
  select s.column from s.table where s.condition
    UNION ALL
  select t.column from t.table where t.condition
)
```

Pri spojení dvoch kolekcií výrazov select teda nakoniec dochádza len k spájaniu dvojíc výrazov select.

Spárovanie reprezentácií

Na vstupe očakáva dve kolekcie reprezentácií výrazov `select` R_1 a R_2. Výstupom je kolekcia dvojíc reprezentácií výrazov `select` D.

1. Nastav D na prázdnu kolekciu.
2. Pre všetky výrazy select r_1 z R_1:
   (a) Pre všetky výrazy select r_2 z R_2:
      i. Ak platí $|R_1.variantsFor \cap R_2.variantsFor| = |r_1.variantsFor \cap r_2.variantsFor|$ potom:
         A. Pridaj do D dvojicu r_1 a r_2, kde ich množinou nadradených reprezentácií je $r_1.variantsFor$ zjednotenie $r_2.variantsFor$
3. Nastav $D.variantsFor$ na R_1 zjednotenie R_2.

Spojenie pre operátory vo výraze OrExpr

Sú spájané dve kolekcie reprezentácií výrazov select R_1 a R_2 pre operátor *op*.

1. Zavolaj **Uprav na column ako hodnotu** pre R_1.
2. Zavolaj **Uprav na column ako hodnotu** pre R_2.
3. Zavolaj **Spárovanie výrazov select** pre R_1 a R_2 a výslednú kolekciu dvojíc označ D.
4. Nastav R na prázdnu kolekciu.
5. Nastav $R.variantsFor$ na $D.variantsFor$
6. Pre všetky dvojice d z D (prvý prvok dvojice je označovaný d_1, druhý d_2):
   (a) Vlož do R reprezentáciu výrazu select s takto nastavenými premennými:
      - *variantsFor* nastav na $d.variantsFor$
      - *column* nastav na $d_1.columnopd_2.column$
7. Výsledkom spojenia je R.

Spojenie pre Condition vo výraze StepExpr

Sú spájané dve kolekcie reprezentácií výrazov `select` V pre výraz `XPath` a C pre výraz `Condition`.

1. Zavolaj **Uprav na column ako hodnotu** pre C.
2. Do $V.variantsFor$ pridaj V.
3. Pre všetky výrazy v z V:
   (a) Do $v.variantsFor$ pridaj v.
4. Zavolaj **Spárovanie výrazov select** pre V a C. Výslednú kolekciu dvojíc označ D.
5. Vytvor prázdnu kolekciu a označ ju R.
6. Nastav $R.variantsFor$ na $D.variantsFor$
7. Z $R.variantsFor$ odober R.
8. Pre všetky dvojice d z D (prvý prvok dvojice je označovaný d_1, druhý d_2):
   (a) Vytvor reprezentáciu výrazu `select` r s takto nastavenými premennými:
      - $column$ nastav na $d_1.column$
      - $table$ nastav na $d_1.table$
      - $condition$ nastav na $d_1.conditionandd_2.column$
      - $node$ nastav na $d_1.node$
      - $variantsFor$ nastav na $d.variantsFor$
   (b) Z $r.variantsFor$ odober d_1
   (c) r pridaj do R
9. Výsledkom spojenia je R.

Spojenie premenných zadefinovaných pravidlom ForClause

1. Vytvor prázdnu kolekciu a označ ju R.
2. Vlož do R reprezentáciu výrazu `select` s prázdnymi všetkými premennými.
3. Pre všetky premenné z globálneho zoznamu zadefinované výrazom `ForClause`:
   (a) Označ F kolekciu reprezentácií pre premennú.
   (b) Zavolaj **Spárovanie výrazov select** pre R a F. Výslednú kolekciu dvojíc označ D.
   (c) Nastav R na prázdnu kolekciu.
   (d) Nastav $R.variantsFor$ na $D.variantsFor$
   (e) Pre všetky dvojice d z D (prvý prvok dvojice je označovaný d_1, druhý d_2):

i. Vlož do R reprezentáciu výrazu `select` s takto nastavenými premennými:
   - $variantsFor$ nastav na $d.variantsFor$
   - $table$ nastav na $d_1.table, d2.table$ (niektorá z premenných môže byť prázdna)
   - $condition$ nastav na $d1.conditionandd2.condition$

4. Výsledkom spojenia je R.

Spojenie pre Condition vo výraze WhereClause

Na vstupe očakáva kolekciu R a kolekciu C pre `Condition`.

1. Zavolaj **Uprav na column ako hodnotu** pre C.
2. Zavolaj **Spárovanie reprezentácií** pre R a C Výslednú kolekciu dvojíc označ D.
3. Nastav R na prázdnu kolekciu.
4. Nastav $R.variantsFor$ na $D.variantsFor$
5. Pre všetky dvojice d z D (prvý prvok dvojice je označovaný d_1, druhý d_2):
   (a) Vlož do R reprezentáciu výrazu `select` s takto nastavenými premennými:
      - $column$ nastav na $d_1.column$
      - $table$ nastav na $d_1.table$
      - $condition$ nastav na $d_1.conditionandd2.column$
      - $variantsFor$ nastav na $d.variantsFor$
6. Výsledkom spojenia je R.

Spojenie pre parametre aktualizačných výrazov

Na vstupe očakáva kolekciu K a kolekciu P_1 a P_2 pre parametre.

1. Zavolaj **Uprav na column ako postupnosť** pre P_1.
2. Zavolaj **Uprav na column ako postupnosť** pre P_2.
3. Zavolaj **Spárovanie reprezentácií** pre K a P_1 a výslednú kolekciu dvojíc označ D.
4. Nastav R na prázdnu kolekciu.
5. Nastav $R.variantsFor$ na $D.variantsFor$
6. Pre všetky dvojice d z D (prvý prvok dvojice je označovaný d_1, druhý d_2):
   (a) Vlož do R reprezentáciu výrazu `select` s takto nastavenými premennými:

- *column* nastav na $d_2.column$
- *table* nastav na $d_1.table$
- *condition* nastav na $d_1.condition$
- $variantsFor$ nastav na $d.variantsFor$

7. Zavolaj **Spárovanie reprezentácií** pre R a P_2 a výslednú kolekciu dvojíc označ D.
8. Nastav R na prázdnu kolekciu.
9. Nastav $R.variantsFor$ na $D.variantsFor$
10. Pre všetky dvojice d z D (prvý prvok dvojice je označovaný d_1, druhý d_2):
    (a) Vlož do R reprezentáciu výrazu `select` s takto nastavenými premennými:
        - *column* nastav na $d_1.column, d2.column$
        - *table* nastav na $d_1.table$
        - *condition* nastav na $d_1.condition$
        - $variantsFor$ nastav na $d.variantsFor$
11. Výsledkom spojenia je R.

Vytvorenie výsledného výrazu

Zoznam výsledných výrazov je vytvorený nasledovne:

1. Zavolaj **Spojenie premenných zadefinovaných pravidlom ForClause** a výslednú kolekciu označ R.
2. Ak sa v požiadavke vyskytol výraz `WhereClause`
    (a) Potom zavolaj **Spojenie pre Condition vo výraze WhereClause** pre R a kolekciu pre výraz `Condtion`. Výslednú kolekciu označ S.
    (b) Inak R označ S.
3. Pre všetky základné aktualizačné výrazy v požiadavke:
    (a) P_1 označ kolekciu pre cieľový parameter a P_2 označ kolekciu pre zdrojový parameter základného aktualizačného výrazu. Pre základný aktualizačný výraz `DELETE` pre P_2 použi kolekciu s jedinou reprezentáciou. Táto reprezentácia má premennú *column* nastavenú na reťazec `NULL`.
    (b) Zavolaj **Spojenie pre parametre aktualizačných výrazov** pre S, P_1 a P_2, výslednú kolekciu označ T.
    (c) Do zoznamu výsledných výrazov pridaj dvojicu typ základného aktualizačného výrazu a požiadavku typu select vytvorenú z reprezentácií v T, spojených pomocou operátoru `UNION ALL`.

4.5 Druhá fáza

Druhá fáza očakáva na vstupe zoznam dvojíc tvorených typom základného aktualizačného výrazu a SQL požiadavkou typu `select`. Výstupom každej požiadavky typu `select` je postupnosť cieľových uzlov a postupnosť zdrojových uzlov. Zoznam je spracovaný nasledovným algoritmom:

1. Pre každú dvojicu vo vstupnom zozname:
   (a) Vyhodnoť požiadavku typu `select`.
   (b) Pre každý riadok výstupu:
      i. Odstráň nulové uzly z postupnosti cieľových uzlov.
      ii. Ak je postupnosť cieľových uzlov prázdna pokračuj v bode 1.(b).
      iii. Ak je typ základného aktualizačného výrazu rôzny od `DELETE`:
         A. Ak je počet cieľových uzlov väčší ako 1, potom ukonči vyhodnotenie celej aktualizačnej požiadavky chybou - nevhodný počet cieľových uzlov.
         B. Ak je typ základného aktualizačného výrazu `replace node` a ak cieľový uzol je cieľovým uzlom iného základného aktualizačného výrazu `replace node` - skonči spracovanie požiadavky chybou.
         C. Ak je typ základného aktualizačného výrazu `replace value` a ak cieľový uzol je cieľovým uzlom iného základného aktualizačného výrazu `replace value` - skonči spracovanie požiadavky chybou.
         D. Odstráň nulové uzly z postupnosti zdrojových uzlov.
      iv. Vytvor aktualizačné primitívum a pridaj ho do zoznamu čakajúcich aktualizácií.

Aktualizačné primitívum je tvorené typom základného aktualizačného výrazu, z ktorého bolo vytvorené a dvojice postupnosť zdrojových uzlov a postupnosť cieľových uzlov.

4.6 Tretia fáza

Tretia fáza očakáva na vstupe zoznam čakajúcich aktualizácií. Zmeny reprezentované aktualizačnými primitívami na tomto zozname je možné rozdeliť na dve skupiny. Na tie, ktoré je možné aplikovať do podkladového úložiska jednou SQL požiadavkou typu update a komplikovanejšie zmeny, pri ktorých je potrebné vykonať preusporiadanie detí.

4.6.1 Preusporiadanie detí

Už pri jednoduchých aktualizačných operáciách ako `insert` a `delete` môže nastať situácia, kedy nestačí vložiť/odobrať element na dané miesto, ale je potrebné presunúť súrodencov pridávaného/odoberaného elementu. Ukážeme si to na príklade.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="A" type="ComplexType1"/>

 <xs:simpleType name="SimpleType1">
  <xs:restriction base="xs:string">
   <xs:maxLength value="20"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="SimpleType2">
  <xs:restriction base="xs:string">
   <xs:maxLength value="40"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name='ComplexType1'>
  <xs:sequence>
   <xs:sequence minOccurs="0"> <!-- prva postupnost -->
    <xs:element name="B" type="xs:string"/>
    <xs:element name="C" type="SimpleType1"/>
   </xs:sequence>
   <xs:sequence minOccurs="0"> <!-- druha postupnost -->
    <xs:element name="D" type="xs:string" minOccurs="0"/>
    <xs:element name="C" type="SimpleType1"/>
   </xs:sequence>
  </xs:sequence>
 </xs:complexType>

</xs:schema>
```

Výpis 4.1: XML schéma 007.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<A xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="007.xsd">
  <B>text for B</B>
  <C>text for C</C>
</A>
```

Výpis 4.2: XML dokumenty 007.xml

Dokument vo výpise 4.2 je validný voči schéma vo výpise 4.1. Elementy A/B aj A/C sú potomkami prvej modelovej skupiny typu postupnosť uzlov.

```
  delete node fn:doc("007.xml")/A/B
```

Výpis 4.3: XQuery požiadavka 007-1.xq

```
<?xml version="1.0" encoding="UTF-8"?>
<A xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="007.xsd">
```

```
  <C>text for C</C>
</A>
```

Výpis 4.4: XML dokument 007.xml (po vykonaní požiadavky 007-1.xq)

Požiadavkou vo výpise 4.3 sa odstráni element A/B a výsledný dokument je opäť validný voči schéme. Okrem odstránenia elementu A/B došlo ale aj k ďalšej zmene. Element A/C je potomkom druhej modelovej skupiny typu výber z uzlov a to bez toho, aby bola vykonaná akákoľvek požiadavka na zmenu elementu A/C.

V podkladovej štruktúre sú prvá a druhá modelová skupina mapované na rôzne UDT a teda inštancie UDT pre element A/C pre prvú a druhú modelovú skupinu sú uložené na iných miestach. Pri vykonaní požiadavky vo výpise 4.3 je potrebné presunúť inštanciu UDT pre element A/C na zodpovedajúce miesto. V prípade, že sa tak nestane, dôjde k porušeniu integritného obmedzenia v podkladovej štruktúre a vykonanie aktualizačnej požiadavky bude skončené s chybou.

Uvedený príklad zobrazuje jednoduchú situáciu, ale pri zložitejších aktualizačných požiadavkách na zložitejších dokumentoch môže ísť o veľký počet operácií, ktoré bude potrebné na udržanie validity dokumentu v podkladovom úložisku vykonať.

XML Schema nedovoľuje, aby element mal dvoch potomkov s rovnakým menom ale rôznym typom. Týmto obmedzením je zabránené tomu, aby pri podobnej požiadavke ako je uvedená vo výpise 4.3 došlo nielen k presunutiu elementu, ale aj k zmene jeho typu. Prípadná zmena typu elementu by zásadne komplikovala vykonanie požiadavky.

4.6.2 Postup aplikovania zmien

Navrhnutý postup pre aplikovanie zmien do podkladovej štruktúry tento problém rieši. Postup aktualizácie sa dá popísať takto:

1. Z podkladovej štruktúry načítaj postupnosť referencií na deti elementu.
2. Aplikuj zmeny reprezentované aktualizačnými primitívami na postupnosť.
3. Zmaž z podkladovej štruktúry uzly typu modelová skupina, ktoré sú na ceste medzi elementami v postupnosti a ich rodičom.
4. Vytvor a ulož do podkladovej štruktúry nové uzly typu modelová skupina, ktoré budú zohľadňovať zmeny vykonané aktualizačnými požiadavkami.

Algoritmus využíva tri metódy z UDT pre uzly podkladovej štruktúry `del_tree`, `del_element` a `get_element_children`. Všetky sú popísané v predchádzajúcej kapitole. Algoritmus ďalej využíva jemne modifikovaný algoritmus pre spracovanie uzla, tiež z predchádzajúcej kapitoly. Upravený algoritmus je popísaný ďalej.

Načítanie detí elementu

Pre vstupný element e, ktorý je komplexného typu s komplexným obsahom, je touto operáciou z podkladového úložiska načítaná postupnosť referencií na podelemety (deti). Deti sú v postupnosti zoradené podľa poradia v XML dokumente. Pre načítanie je využitá funkcia `get_element_children` definovaná pre UDT, na ktoré je element mapovaný.

Počas spracovania všetkých aktualizačných primitív sú v pomocnej štruktúre pre elementy udržiavané dvojice element a postupnosť.

1. Ak e nie je komplexného typu s komplexným obsahom skonči celú aktualizačnú požiadavku s chybou.
2. Ak sa dvojica pre e nenachádza v pomocnej štruktúre pre elementy:
   (a) Načítaj postupnosť referencií na deti e pomocou funkcie `get_element_children`.
   (b) Pridaj dvojicu e a postupnosť detí do pomocnej štruktúry.

Načítanie atribútov elementu

Pre vstupný element e je touto operáciou z podkladového úložiska načítaná postupnosť jeho atribútov.

Počas spracovania všetkých aktualizačných primitív sú v pomocnej štruktúre pre atribúty udržiavané dvojice element a postupnosť jeho atribútov.

1. Ak sa dvojica pre e nenachádza v pomocnej štruktúre pre atribúty:
   (a) Načítaj postupnosť atribútov elementu e.
   (b) Pridaj dvojicu e a postupnosť atribútov do pomocnej štruktúry pre atribúty.

Spracovanie uzla

Algoritmus na vstupe očakáva dvojicu element a postupnosť elementov. Prvkami postupnosti sú referencie na elementy do podkladovej štruktúry, alebo zdrojové uzly pre elementy. Element predstavuje rodičovský element elementov v postupnosti. Výstupom je konštruktor UDT. Počas spracovania sú udržiavané dva ukazovatele. Jeden na aktuálny element ukazuje do vstupnej postupnosti, prípadne na element v niektorom zo zdrojových uzlov v postupnosti. Druhý ukazuje na uzol v XML schéme, ktorý je momentálne spracovávaný.

Na začiatku je ukazovateľ na aktuálny element nastavený na prvý prvok vstupnej postupnosti a ukazovateľ na uzol je nastavený na vstupný element. Algoritmus začína v kroku (d).1. čím je vstupný uzol, ktorý je typu element, spravovaný ako modelová skupina typu postupnosť uzlov. Ak po spracovaní nie je ukazovateľ na element prázdny, vstupný element a jeho deti sa nepodarilo uložiť do podkladovej štruktúry a teda celá požiadavka končí s chybou.

1. Podľa typu aktuálneho uzla vytvor konštruktor UDT alebo ako návratovú hodnotu použi referenciu na uzol do podkladovej štruktúry:
   (a) Ak je ním element:
      i. Ak je ukazovateľ na aktuálny element prázdny, ukonči spracovanie neúspechom.
      ii. Ak meno aktuálneho elementu a meno aktuálneho uzla nie je zhodné, ukonči spracovanie neúspechom.
      iii. Podľa typu obsahu aktuálneho uzla:

A. Ak ide o referenciu na uzol do podkladovej štruktúry použi ju ako návratovú hodnotu.

B. Ak ide o zdrojový uzol a jednoduchý typ obsahu na príslušné miesto konštruktoru UDT ulož hodnotu získanú z aktuálneho elementu.

C. Ak ide o zdrojový uzol komplexný obsah, pre ukazovateľ na aktuálny element nastavený na prvého potomka aktuálneho elementu, spracuj element rovnako ako modelovú skupinu typu postupnosť uzlov, teda podľa bodu 1.(d).

iv. Ak bol vytvorený konštruktor spracuj atribúty aktuálneho elementu nasledujúcim postupom:

A. Získaj zoznam atribútov z popisu XML schémy z DB pre komplexný typ aktuálneho uzla.

B. Pre každý atribút zo zoznamu ulož na príslušné miesto v konštruktore UDT konštruktor UDT pre atribút inicializovaný hodnotou získanou z atribútu aktuálneho elementu. Ak sa atribút v aktuálnom elemente nevyskytuje, ulož na príslušné miesto v konštruktore UDT hodnotu `NULL`.

v. Posuň ukazovateľ na aktuálny element na jeho nasledujúceho súrodenca. Ak taký neexistuje nastav ukazovateľ na prázdny.

(b) Ak je ním modelová skupina typu množina - postup spracovania zostal nezmenený.

(c) Ak je ním modelová skupina typu výber z uzlov - postup spracovania zostal nezmenený.

(d) Ak je ním modelová skupina typu postupnosť uzlov - postup spracovania zostal nezmenený.

2. Ak bol vytvorený konštruktor a všetky parametre konštruktoru UDT sú inicializované na hodnotu `NULL` alebo prázdny konštruktor poľa, ukonči spracovanie neúspechom.

3. Ak bol vytvorený konštruktor a podľa typu mapovania aktuálneho uzla do naduzla vykonaj:

(a) Ak ide o mapovanie do naduzla na atribút alebo pole, vráť konštruktor UDT.

(b) Ak ide o mapovanie do naduzla na referenciu alebo pole referencií. Konštruktorom vytvorenú inštanciu UDT ulož do objektovej tabuľky a vráť požiadavku typu select na referenciu na túto inštanciu UDT.

Uloženie detí elementu

1. Pre všetky dvojice v pomocnej štruktúre pre elementy:

(a) Ak bol element z podkladovej štruktúry zmazaný, pokračuj v spracovaní ďalšej dvojice.

(b) Pomocou procedúry `del_element` definovanej pre UDT, zmaž pre element všetky uzly typu modelová skupina z objektových tabuliek.

(c) Pomocou algoritmu pre Spracovanie uzla, pre element a postupnosť, do podkladovej štruktúry vytvor nové uzly typu modelová skupina.

(d) V inštancii UDT pre element v podkladovej štruktúre, pomocou SQL požiadavky update, aktualizuj hodnoty všetkých atribútov UDT, na ktoré boli modelové skupiny v elemente mapované.

Uloženie atribútov elementu

1. Pre všetky dvojice v pomocnej štruktúre pre atribúty:

   (a) Ak bol element z podkladovej štruktúry zmazaný, pokračuj v spracovaní ďalšej dvojice.

2. Ak postupnosť atribútov obsahuje dva atribúty s rovnakým menom, skonči celú aktualizačnú požiadavku s chybou.

3. V inštancii UDT pre element v podkladovej štruktúre, pomocou SQL požiadavky `update`, aktualizuj hodnoty všetkých atribútov UDT, na ktoré boli atribúty elementu mapované:

   (a) Ak sa v postupnosti atribút s daným menom vyskytuje potom ako hodnotu použi atribút z postupnosti a z postupnosti ho odstráň.

   (b) Inak použi hodnotu NULL.

4. Ak postupnosť atribútov nie je prázdna, skonči celú aktualizačnú požiadavku s chybou.

Algoritmus aplikácie zmien do podkladovej štruktúry

1. Pre všetky aktualizačné primitíva typu `insert`:

   (a) Ak postupnosť zdrojových uzlov obsahuje atribút aj element, skonči aktualizačnú požiadavku chybou.

   (b) Ak postupnosť zdrojových uzlov je tvorená len elementami:

      i. Načítaj deti elementu pre cieľový uzol.

      ii. Pridaj zdrojové uzly na koniec postupnosti.

   (c) Ak postupnosť zdrojových uzlov je tvorená len atribútmi:

      i. Načítaj atribúty pre cieľový uzol.

      ii. Pridaj zdrojové uzly na koniec postupnosti.

2. Pre všetky aktualizačné primitíva typu `insert as first` a `insert as last`:

   (a) Načítaj deti elementu pre cieľový uzol.

   (b) Pridaj zdrojové uzly na začiatok / koniec postupnosti.

3. Pre všetky aktualizačné primitíva typu `insert before` a `insert after`:

   (a) Načítaj deti elementu pre rodiča cieľového uzla.

(b) Pridaj zdrojové uzly pred / za cieľový uzol v postupnosti.

4. Pre všetky aktualizačné primitíva typu `replace value`:
   (a) Ak je typ cieľového uzla atribút:
      i. Načítaj atribúty elementu pre rodiča cieľového uzla.
      ii. Uprav hodnotu cieľového uzla v postupnosti.
5. Pre všetky aktualizačné primitíva typu `replace node`:
   (a) Ak je typ cieľového uzla element:
      i. Načítaj deti elementu pre rodičia cieľového uzla.
      ii. Za cieľový uzol do postupnosti vlož zdrojové uzly.
      iii. Odstráň cieľový uzol z postupnosti.
      iv. Označ cieľový uzol na zmazanie.
   (b) Ak je typ cieľového uzla atribút:
      i. Načítaj atribúty elementu pre rodiča cieľového uzla.
      ii. Do postupnosti vlož zdrojové uzly.
      iii. Odstráň cieľový uzol z postupnosti.
6. Pre všetky aktualizačné primitíva typu `delete`:
   (a) Ak je typ cieľového uzla element:
      i. Načítaj deti elementu pre rodičia cieľového uzla.
      ii. Ak sa v postupnosti nachádza cieľový uzol, odstráň ho.
      iii. Označ cieľový uzol na zmazanie.
   (b) Ak je typ cieľového uzla atribút:
      i. Načítaj atribúty elementu pre rodiča cieľového uzla.
      ii. Ak sa v postupnosti nachádza cieľový uzol, odstráň ho.
7. Pre všetky uzly označené na zmazanie:
   (a) Ak je stále v podkladovej štruktúre uložená inštancia pre uzol označený na zmazanie:
      i. Zavolaj na nej procedúru `del_tree`.
      ii. Odstráň inštanciu z podkladovej štruktúry.
8. Zavolaj Ulož deti elementu.
9. Zavolaj Ulož atribúty elementu.
10. Pre všetky aktualizačné primitíva typu `replace value`:
    (a) Ak je typ cieľového uzla element a ak nebol z podkladovej štruktúry zmazaný:
       i. Pomocou aktualizačnej požiadavky `update` zmeň hodnotu v podkladovej štruktúre.

5. Implementácia

V tejto kapitole je popísaná implementácia riešenia navrhnutého v predchádzajúcich kapitolách. V prvej časti sa zameriame na použité technológie. V druhej je popísané užívateľské prostredie a funkcie, ktoré implementácia ponúka.

5.1 Použité technológie

Implementácia sa skladá z dvoch častí z externej aplikácie napísanej v jazyku Java a z objektov a uložených procedúr v relačnej databáze Oracle 10.2 napísaných v jazyku PL/SQL.
Úlohy, ktoré plní externá aplikácia sú:

- Načítanie súboru s XML schémou popísanou v jazyku XML Schema a vytvorenie zodpovedajúcej štruktúry v databáze.
- Načítanie súboru s XML dokumentom a uloženie tohoto dokumentu do pripravenej štruktúry v databáze.
- Načítanie súboru s aktualizačnou požiadavkou. Preklad aktualizačnej požiadavky na požiadavky `select` a zavolanie uloženej procedúry, ktorá zmeny reprezentované aktualizačnou požiadavkou aplikuje na hodnoty v štruktúre.
- Poskytuje grafické rozhranie pre ovládanie.

Jazyk Java bol zvolený pre existenciu knižníc, ktoré urýchlili implementáciu riešenia a pre prepracované prepojenie aplikácií v Java s relačnou databázou. Použitými knižnicami v riešení sú Xerces2[1], CUP[2] a JFlex[3]. Prvá umožňuje načítanie XML dokumentu a vytvorenie jeho DOM. Rovnako sa dá použiť pre načítanie XML schémy, ktorá je tiež XML dokumentom. Druhá a tretia knižnica slúžia na spracovanie vstupnej požiadavky. Umožňujú pre pravidlá navrhnutej gramatiky vygenerovať zdrojový kód triedy pre lexikálnu analýzu a parsovanie aktualizačnej požiadavky. Všetky tri knižnice predstavujú štandard, či už pre spracovanie XML alebo generovanie parserov, medzi knižnicami v Java.
Úlohy, ktoré plnia uložené procedúry:

- Zmazanie XML dokumentu zo štruktúry.
- Zmazanie štruktúry pre XML schému a všetkých dokumentov v nej.
- Vyhodnotenie vstupných parameterov pre základné aktualizačné výrazy a aplikovanie zmien, ktoré reprezentujú do štruktúry.

[1]Apache Xerces2 Java je knižnica pre spracovanie XML. Viac informácií je možné nájsť na http://xerces.apache.org/.

[2]CUP Parser Generator for Java je knižnica pre generovanie parserov. Viac informácií je možné nájsť na http://www.cs.princeton.edu/ appel/modern/java/CUP/.

[3]The Fast Scanner Generator for Java je knižnica pre generovanie lexikálneho analyzátoru. Domovská stránka projektu je http://jflex.de/.

Relačná databáza Oracle bola zvolená pre prepracovaný objektový model, ktorý poskytuje. Verzia 10.2. tejto databázy bola v čase začatia prác na implementácii riešenia aktuálnou verziou.
Dôvody rozdelenia vyhodnotenia aktualizačnej požiadavky, na spracovanie v externej aplikácií a v uloženej procedúre sú nasledovné:

- Parsovanie aktualizačnej požiadavky v uloženej procedúre, bez podpory knižnice CUP by bolo zbytočne komplikované.
- Vyhodnotenie vstupných parametrov pre základné aktualizačné výrazy často znamená vytvoriť kópiu celého podstromu XML dokumentu. Prenášať takýto podstrom do externej aplikácie pre spracovanie, by bolo z výkonového hľadiska veľmi náročné.
- Pri aplikovaní zmien reprezentovaných aktualizačnou požiadavkou je vykonané nemalé množstvo požiadaviek do relačnej databázy, čo je efektívnejšie vykonať z uloženej procedúry ako z externej aplikácie.

5.2 Užívateľský návod

V tejto časti si popíšeme ako v pár krokoch spustiť implementáciu a overiť jej funkčnosť. Podrobnejší popis užívateľského prostredia je na priloženom CD v súbore `dist/dokumentacia.pdf`.

Na priloženom CD v adresári `dist` je implementácia pripravená na spustenie. Na počítači, na ktorom bude spustená je potrebné mať nainštalované JRE[4] 1.6 alebo novšie. Program sa bude pripájať do relačnej databázy Oracle 10.2. Konfiguračný súbor `dist/conf/xquery2sql.conf` obsahuje nastavenia pre pripojenie k bežiacej inštancii tejto databázy. Ide o vývojový server a jeho prevádzka nie je zaručená. Pre otestovanie funkčnosti programu však môže postačovať. Pomocné objekty v tejto databáze už boli vytvorené.

Súbormi `dist/xquery2sql.bat` a `dist/xquery2sql.sh` je možné spustiť program s grafickým užívateľským prostredím pre operačný systém Windows a Linux.

Grafické prostredie je možné ovládať cez menu, ktoré poskytuje nasledujúce položky:

- `reconnect` - pokúsi sa o obnovenie spojenia s databázou
- `create schema` - v novom okne s formulárom je možné zadať URI (url, absolútna alebo relatívna cesta) pre XML dokument. Po kliknutí na tlačítko `Ok`. program vytvorí v databáze štruktúry pre XML schému daného dokumentu.
- `delete schema` - v novom okne so zoznamom XML schém je možné vybrať schému na zmazanie.
- `create document` - v novom okne s formulárom je možné zadať URI pre XML dokument. Po kliknutí na `Ok`. program uloží dokument do databázy.

[4]Java Runtime Environment - Behové Prostredie Java. Inšlačný balíček je možné stiahnuť z www.java.com/getjava/.

- `delete document` - v novom okne so zoznamom XML dokumentov je možné vybrať okument na zmazanie
- `execute query` - v novom okne s formulárom je možné zadať cestu k súboru s aktualizačnou požiadavkou.
- `exit` - ukončí program

V adresári `dist/data` sú umiestnené príklady XML schém (prípona xsd), XML dokumentov (prípona xml) a aktualizačných požiadaviek (prípona xq) pre testovanie.

- Pre vytvorenie štruktúry pre schému súboru `dist/data/001.xml`, je potrebné v menu zvoliť `create schema` a zadať cestu `data/001.xml`.
- Pre uloženie dokumentu `dist/data/001.xml`, je potrebné v menu zvoliť `create document` a zadať cestu `data/001.xml`.
- Pre vykonanie aktualizačnej požiadavky `dist/data/001-1.xq`, je potrebné v menu zvoliť `execute query` a zadať cestu `data/001-1.xq`.

V hlavnom okne je zobrazený zoznam URI XML schém, pre ktoré bola v databáze pripravená štruktúra. Tlačítkom `show schema` je možné zobraziť schému, pre URI zo zoznamu. V novom okne sa zobrazia koreňové uzly, pre danú schému. Postupným rozkliknutím uzlov je možné zobraziť celý strom. Po ukázaní kurzorom na niektorý z uzlov sa zobrazia informácie o uzle. Význam jednotlivých uzlov stromu:

- list stromu predstavuje:
  - jednoduchý dátový typ elementu s jednoduchým obsahom, alebo
  - meno a jednoduchý dátový typ atribútu, oddelené znakom ':'
- označenie *TYP*: *UDT_NAME* <*XML_NAME*> je použité pre nelistový uzol stromu kde:
  - *UDT_NAME* je meno UDT, na ktorý je uzol mapovaný
  - *XML_NAME* je v prípade elementov meno elementu, inak je prázdne
  - *TYP* je jeden znak označujúci typ uzla:
    - `e` - element
    - `a` - modelová skupina typu množina uzlov (all)
    - `c` - modelová skupina typu výber z uzlov (choice)
    - `s` - modelová skupina typu postupnosť uzlov (sequence)

V hlavnom okne je ďalej zobrazený zoznam URI XML dokumentov, ktoré sú v databáze uložené. Tlačítkom `show document` je možné zobraziť dokument, pre URI zo zoznamu. V novom okne sa zobrazí koreňový element, pre daný dokument. Postupným rozkliknutím uzlov je možné zobraziť celý strom. Po ukázaní kurzorom na niektorý z uzlov sa zobrazia informácie o uzle. Význam jednotlivých uzlov stromu:

- list stromu predstavuje:
  - hodnotu elementu s jednoduchým obsahom, alebo
  - meno a hodnotu atribútu, oddelené znakom ':'
- nelistový uzol stromu predstavuje element označený menom elementu

V hlavnom okne je ďalej zobrazené pole pre zadávanie aktualizačných požiadaviek. Aktualizačnú požiadavku je možné vykonať stlačením tlačítka `execute query`. Po úspešnom vykonaní aktualizačnej požiadavky sa zobrazí dialógové okno s počtom cieľových uzlov, pre ktoré boli vykonané aktualizácie. V prípade chyby sa zobrazí dialógové okno s krátkou správou o chybe. Podrobnejší popis chyby je možné nájsť na chybovom výstupe na konzole programu.

Záver

Hneď v úvode záveru tejto práce je nutné spomenúť, že ciele tejto práce, stanovené v úvode práce, sa podarilo splniť.

Bol preskúmaný štandard XQuery Update Facility 1.0, ktorým je jazyk XQuery doplnený o aktualizácie. V práci sa podarilo navrhnúť a implementovať aktualizáciu XML dát uložených v objektovo-relačnej databáze. Bolo navrhnuté mapovanie jazyka XML Schema na prostriedky objektovo-relačnej databázy, tak aby vytvorená štruktúra umožňovala aktualizovanie XML dát, ktoré sú v nej uložené. Tiež sa podarilo dosiahnuť, že vytvorená štruktúra vynucuje validitu uloženého dokumentu po každej aktualizačnej požiadavke. V práci bol takisto navrhnutý jazyk pre aktualizačné požiadavky, ktorý vychádza z jazyka XQuery Update Facility 1.0, tak, že tento jazyk má nielen podobnú syntax ale aj sémantiku. Bolo navrhnuté a implementované vyhodnocovanie takýchto požiadaviek. Toto vyhodnocovanie sa skladá z časti, kde je požiadavka namapovaná na požiadavky `select` v SQL, ktorými sú získané hodnoty vstupných parameterov pre základné aktualizačné výrazy. Druhou časťou vyhodnocovania je aplikovanie zmien reprezentovaných základnými aktualizačnými výrazmi do podkladovej štruktúry.

Niektoré z cieľov, ktoré boli na začiatku na riešenie kladené výrazne skomplikovali jeho návrh a implementáciu, pritom pre aktualizáciu XML dát uložených objektovo-relačnej databáze neboli nevyhnutné. Jedným z nich je požiadavka na využitie čo najväčšieho počtu prostriedkov objektovo-relačnej databázy, prípadne samotné mapovanie jazyka XML Schema na prostriedky objektovo-relačnej databázy. Jazyk XML Schema je veľmi rozsiahli a niektoré jeho časti museli byť z návrhu vynechané, pretože pre ne v objektovo-relačnej databáze neexistuje prostriedok, na ktorý by mohli byť mapované. Tým je navrhnuté riešenie v praxi len veľmi ťažko použiteľné. Cieľ mapovať dátové typy jazyka XML Schema na dátové typy SQL sa ukázal, tiež ako veľmi obmedzujúci.

Nebyť týchto cieľov mohlo byť využité niektoré z obecných mapovaní XML dokumentov do objektovo-relačnej databázy. To by výrazne zjednodušilo aplikovanie zmien reprezentovaných jednotlivými základnými aktualizačnými výrazmi do podkladového úložiska. Ďalej by to umožnilo do riešenia zahrnúť väčšiu časť jazykov XML Schema aj XQuery. Relačná databáza by bola len úložiskom XML dát bez ďalších funkcií. Kontrola validity dokumentu by potom musela byť zabezpečená externou aplikáciou alebo uloženou procedúrou voči informáciám o XML schéme, ktoré by boli uložené v pomocných tabuľkách databázy. Takéto riešenie by bolo schopné podporovať prevažnú časť z jazykov XML Schema aj XQuery a možno by mohlo byť dopracované do stavu, v ktorom by bolo použitelné v praxi.

Napriek uvedenému aj riešenie popísané v tejto práci poskytuje priestor pre ďalšie rozšírenie. Mohlo by ísť napríklad o doplnenie operátorov do výrazu `XPath` v aktualizačnej požiadavke, čo by umožnilo vytvárať komplexnejšie požiadavky. Zaujímavým rozšírením práce by bolo nahradenie štandardných operátorov SQL pre jednoduché dátové typy, za operátori podporujúce postupnosti ako operandy.

Zoznam použitej literatúry

[1] I. Mlýnková. *XML Schema a jeho implementace v prostředí relační databáze*. Univerzita Karlova v Praze, 2003.

[2] P. Bohannon, J. Freire, P. Roy, J. Siméon. *From XML Schema to Relations: A Cost-Based Approach to XML Storage*. Bell Laboratories, 2002.

[3] P. Amornsinlaphachai, N. Rossiter, M. A. Ali. *Translating XML Update Language into SQL*. Journal of computing and information technology - CIT, vol. 14, no. 2, pp. 81-100, 2006.

[4] H. Sun, S. Zhang, J. Zhou, J. Wang. *Constraints-Preserving Mapping Algorithm from XML-Schema to Relational Schema*. Engineering and Deployment of Cooperative Information Systems, vol. 2480, pp. 284-288, 2002.

[5] I. Mlýnková, J.Pokorný. *XML in the World of (Object-) Relational Database Systems*. Charles University, Prague, 2003.

[6] B. Kane, H. Su, E. A. Rundensteiner. *Consistently Updating XML Documents using Incremental Constraint Check Queries*. Worcester Polytechnic Institute, Worcester, 2002.

[7] R. Bourret. *Mapping W3C Schemas to Object Schemas to Relational Schemas*. 2001. http://www.rpbourret.com/xml/SchemaMap.htm.

[8] P. Amornsinlaphachai, M. A. Ali, N. Rossiter. *Updating XML Using Object-Relational Database*. BNCOD 2005, LNCS 3567, pp. 155-160, 2005.

[9] I. Mlýnková, J. Pokorný, K. Richta, K. Toman, V. Toman. *Technologie XML*. Univerzita Karlova v Praze, Nakladatelství Karolinum, Praha, 2006.

[10] X. Franc. *XQuery Update for the impatient*. http://www.xmlmind.com/tutorials/XQueryUpdate/index.html.

[11] I. Tatarinov, Z. G. Ives, A. Y. Halevy, D. S. Weld. *Updating XML*. ACM SIGMOD International Conference on Management of Data, pp. 413-424, 2001.

[12] IEEE Standards Association. *IEEE Standard for Binary Floating-Point Arithmetic*. http://standards.ieee.org/findstds/standard/754-1985.html.

[13] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C, 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[14] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon. *XQuery 1.0: An XML Query Language (Second Edition)*. w3C, 2010. http://www.w3.org/TR/2010/REC-xquery-20101214/.

[15] J. Robie, D. Chamberlin, M. Dyck, D. Florescu, J. Melton, J. Siméon. *XQuery Update Facility 1.0*. W3C, 2011. http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/.

[16] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn. *XML Schema Part 1: Structures Second Edition*. W3C, 2004. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[17] P. V. Biron, A. Malhotra. *XML Schema Part 2: Datatypes Second Edition*. W3C, 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

Zoznam tabuliek

Zoznam výpisov

Zoznam použitých skratiek

W3C The World Wide Web Consortium - medzinárodná komunita pracujúca na vývoji webových štandartov. Viac informácií je možné nájsť na http://www.w3.org/. 2

UDT User Defined Type - uživateľom definovaný typ v objektovo-relačnej databáze Oracle. Obdoba tried v objektových jazykoch so štandartným objektovým modelom. 18

IEEE 754-1985 Norma popisújuca formát číselných dátových typov s pohyblivou desatinnou čiarkou. Jej komplentné znenie je možné nájsť v dokumente [12]. 21

xsi:type http://www.w3.org/2001/XMLSchema-instance:type je celé označenie atribútu aj s URI pre namespace. 28

DOM Document Object Model - objektový model XML dokumentu. 35

XML Extensible Markup Language - doslovný preklad je rozšíriteľný značkovací jazyk. 2

Xerces2 Apache Xerces2 Java je knižnica pre spracovanie XML. Viac informácií je možné nájsť na http://xerces.apache.org/. 72

CUP CUP Parser Generator for Java je knižnica pre generovanie parserov. Viac informácií je možné nájsť na http://www.cs.princeton.edu/ appel/modern/java/CUP/. 72

JFlex The Fast Scanner Generator for Java je knižnica pre generovanie lexikálneho analyzátoru. Domovská stránka projektu je http://jflex.de/. 72

JRE Java Runtime Environment - Behové Prostredie Java. Inšlačný balíček je možné stiahnuť z www.java.com/getjava/. 73

Prílohy

1. Gramatika aktualizačnej požiadavky

```
Query ::= ( ExprSingle | ( (ForClause | LetClause)+ WhereClause?
  "return" Expr ) )

Expr ::= ExprSingle | "(" ExprSingle ("," ExprSingle)* ")"

ForClause ::= "for" "$" VarName "in" XPathExpr
  ("," "$" VarName "in" XPathExpr)*
LetClause ::= "let" "$" VarName ":=" XPathExpr
  ("," "$" VarName ":=" XPathExpr)*

WhereClause ::= "where" Condition

ExprSingle ::= InsertExpr
    | DeleteExpr
    | ReplaceExpr

RelativeXPathExpr ::= StepExpr ("/" StepExpr)* ("/" "@" StepExpr)?
XPathExpr ::=   "fn:doc" "(" URILiteral ")" ("/" StepExpr)+
        ("/" "@" StepExpr)?
    | "$" VarName ("/" StepExpr)* ("/" "@" StepExpr)?

StepExpr ::= QName ( "\[" Condition "\]" )?

Condition ::= OrExpr

ParenthesizedExpr ::= "(" OrExpr ")"

OrExpr ::= AndExpr ( "or" AndExpr )*
AndExpr ::= ComparisonExpr ( "and" ComparisonExpr )*

ComparisonExpr ::= AdditiveExpr ( ValueComp ) AdditiveExpr )?

AdditiveExpr ::= MultiplicativeExpr ( ("+" | "-") MultiplicativeExpr )*
MultiplicativeExpr ::= UnaryExpr ( ("*" | "div" | "mod") UnaryExpr )*
UnaryExpr ::= ("-" | "+")* ValueExpr

ValueExpr ::= Value
    | ParenthesizedExpr

ValueComp ::= "eq" | "ne" | "lt" | "le" | "gt" | "ge"

Value ::= CurrentValue
    | RelativeXPathExpr
```

```
    | XPathExpr
    | Literal

CurrentValue ::= "."

InsertExpr ::= "insert" ("node" | "nodes") SourceExpr
  InsertExprTargetChoice XPathExpr

DeleteExpr ::= "delete" ("node" | "nodes") XPathExpr

ReplaceExpr ::= "replace" ("value" "of")? "node" XPathExpr
  "with" SourceExpr

InsertExprTargetChoice ::= (("as" ("first" | "last"))? "into")
    | "after"
    | "before"

SourceExpr ::= OrExpr
    | Constructor

Constructor ::= "<" QName DirAttributeList ( "/>" | (">" (
        DirectConstructor* | DirElemContent* ) "</" QName ">"))
    | "attribute" QName "{" OrExpr "}"

DirAttributeList ::= (QName "=" DirAttributeValue)*
DirAttributeValue ::= ('"' (QuotAttrValueContent)* '"')
    | ("'" (EscapeApos | AposAttrValueContent)* "'")

QuotAttrValueContent ::= QuotAttrContentChar
    | CommonContent
    | EscapeQuot

AposAttrValueContent ::= AposAttrContentChar
    | CommonContent
    | EscapeApos

DirElemContent ::= CommonContent
    | ElementContentChar

CommonContent ::= PredefinedEntityRef
    | CharRef
    | "{{"
    | "}}"
    | EnclosedExpr
EclosedExpr ::= "{" OrExpr "}"

VarName ::= QName
URILiteral ::= StringLiteral
```

```
Literal ::= NumericLiteral
    | StringLiteral
NumericLiteral ::= IntegerLiteral
    | DecimalLiteral
    | DoubleLiteral

IntegerLiteral ::= Digits
DecimalLiteral ::= ("." Digits) | (Digits "." [0-9]*)
DoubleLiteral ::= (("." Digits) | (Digits ("." [0-9]*)?))
        [eE] [+-]? Digits
StringLiteral ::= ('"' (PredefinedEntityRef | CharRef |
        EscapeQuot | [^"&])* '"')
    | ("'" (PredefinedEntityRef | CharRef |
        EscapeApos | [^'&])* "'")

PredefinedEntityRef ::= "&" ("lt" | "gt" | "amp" | "quot" | "apos") ";"

EscapeQuot ::= '""'
EscapeApos ::= "''"

ElementContentChar ::= Char - [{}<&]
QuotAttrContentChar ::= Char - ["{}<&]
AposAttrContentChar ::= Char - ['{}<&]
CharRef  ::= &# [0-9]+ ; | &#x [0-9a-fA-F]+ ;

QName ::= ([A-Z] | _ | [a-z]) | ([A-Z] | _ | [a-z] | - | \. | [0-9])*

S ::= (\\u0020 | \\u0009 | \\u000D | \\u000A)+
Char ::= \\u0009 | \u000A | \u000D | [\u0020-\uD7FF] | [\uE000-\uFFFD]
Digits ::= [0-9]+
```