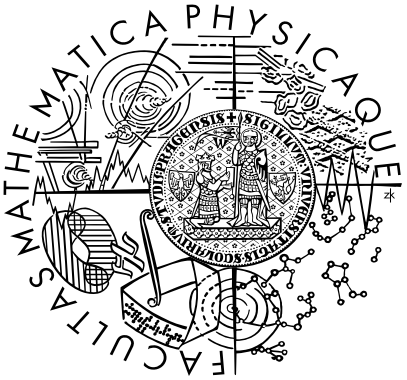
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Irena Mlýnková

XML Schema a jeho implementace v prostředí relační databáze

Katedra softwarového inženýrství
Vedoucí diplomové práce: *Prof. RNDr. Jaroslav Pokorný, CSc.*
Studijní program: *Informatika*

Na tomto místě bych ráda poděkovala vedoucímu své diplomové práce Prof. RNDr. Jaroslavu Pokornému, CSc. za odborné rady a náměty, které zásadním způsobem přispěly k dokončení této práce.

Prohlašuji, že jsem svou diplomovou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 15. srpna 2003

Irena Mlýnková

Obsah

Seznam obrázků

Seznam tabulek

Název práce: *XML Schema a jeho implementace v prostředí relační databáze*

Autor: *Irena Mlýnková*

Katedra: *Katedra softwarového inženýrství*

Vedoucí diplomové práce: *Prof. RNDr. Jaroslav Pokorný, CSc.*

e-mail vedoucího: *pokorny@ksi.mff.cuni.cz*

Abstrakt: *Jazyk XML se v poslední době velmi rychle stává klíčovým standardem pro reprezentaci a předávání informací. S touto tendencí ovšem vystává i nezbytný požadavek na efektivní ukládání a správu XML dokumentů. Jednou z cest jeho řešení může být propojení XML a relačních databází.*

Tato práce se zabývá možností ukládání XML dokumentů do objektově relačních databází. Nejprve je navržena metoda vytvoření objektově relačního schématu (definovaného normou SQL:1999) na základě schématu dat popsaném v jazyce XML Schema. Do takto vytvořeného schématu jsou následně ukládána data z XML dokumentů. Na závěr je navržena metoda transformace dotazů v jazyce XQL na SQL dotazy nad tímto schématem a transformace jejich výsledků zpět na XML dokumenty.

Klíčová slova: *XML, XML Schema, SQL:1999, objektově relační databáze, mapování*

Title: *XML Schema and its' Implementation in Relational Databases*

Author: *Irena Mlýnková*

Department: *Department of Software Engineering*

Supervisor: *Prof. RNDr. Jaroslav Pokorný, CSc.*

Supervisor's e-mail address: *pokorny@ksi.mff.cuni.cz*

Abstract: *Recently XML is quickly becoming a crucial format for representing and exchanging information. However, this tendency brings an essential demand for effective storage and management of XML documents. One solution can be found in connecting XML and relational databases.*

This thesis deals with the possibility of storing XML documents in object relational databases. First, a technique of creating an object relational schema (defined by the SQL:1999 standard) according to a schema described in XML Schema language is designed. Next, the data from XML documents are stored in such schema. Finally, a technique of transforming XQL queries into SQL queries among such schema and a technique of transforming their results back into XML documents is designed.

Keywords: *XML, XML Schema, SQL:1999, object relational database, mapping*

1 Úvod

Informace, jejich výměna a především efektivní zpracování měly od počátku vzniku lidstva strategický význam. V dnešní době jsou informace uchovávány, předávány a zpracovávány zejména v elektronické podobě a pro jejich výměnu je klíčovým prostředkem především celosvětová síť Internet. Aby spolu dva subjekty mohly komunikovat je nutné, aby oba znaly strukturu předávaných dat a pokud možno pružně reagovaly na její změny, přičemž především splnění druhého požadavku bývá obvykle problematické.

V reakci na tento problém se právě jazyk XML velmi rychle stává klíčovým standardem pro reprezentaci dat. Důvodem této tendence je zejména fakt, že XML není pouze jazyk pro popis samotných dat, ale hlavně jazyk pro popis jejich struktury. Právě díky této vlastnosti se zpracování XML dat zásadním způsobem zjednodušuje.

Ze známých jazyků se XML nejvíce podobá jazyku HTML (oba jsou podmnožinami komplexnějšího jazyka SGML). Možnosti jazyka HTML jsou již ale vyčerpány. Je využíván zejména pro určení způsobu vizualizace částí HTML dokumentů, nikoli jejich logiky a omezená množina použitelných značek brání dalšímu rozšíření možností jeho využití. Oba tyto nedostatky jsou v jazyku XML odstraněny.

S použitím XML ovšem vyvstává nezbytný požadavek na efektivní ukládání a správu XML dokumentů a dotazování nad XML daty. Jednou z cest, která se nabízí, může být ukládání XML dat do relačních databází. Tato myšlenka vyplývá zejména z popisu struktury XML dokumentů, která se v mnohém vlastnostem relačních databází podobá. Téměř všichni výrobci databázových systémů již nějakým způsobem ukládání XML dokumentů podporují, ovšem rozdíly mezi nimi jsou zásadní a vývoj v této oblasti tedy ještě zdaleka nedosáhl dostatečné úrovně.

1.1 Cíl práce

Cílem této diplomové práce je využít pro ukládání XML dokumentů do relační databáze jazyk XML Schema jakožto jazyk pro popis XML dat.

Základem práce je navržení vhodného algoritmu transformace schématu XML dat vyjádřeného v jazyce XML Schema do relačního databázového schématu. Druhým dílčím cílem je do takto vytvořené relační databáze ukládat data z XML dokumentů odpovídajících původnímu XML schématu. Posledním cílem je pak implementovat mapování podmnožiny vhodného dotazovacího jazyka nad XML dokumenty na posloupnost `SELECT` příkazů nad vytvořenou databází.

Tato práce navazuje na diplomové práce [18] a [24]. V první z nich využívá autor pro ukládání XML dokumentů do relační databáze jazyk DTD[1]. Druhá

[1]DTD (Document Type Description) – jazyk pro popis přípustné struktury XML dokumentu (viz. [12])

práce je již založena přímo na jazyku XML Schema a zabývá se jeho mapováním na relační databázové schéma přechodem přes HDM[2], s cílem zajištění rozšiřitelnosti algoritmu o další transformace. Naproti tomu se tato diplomová práce zaměřuje především na objektově orientované rysy jazyka XML Schema, jako jsou uživatelsky definované typy, dědičnost, substituovatelnost apod. Cílovým databázovým schématem je tedy schéma objektově relační, konkrétně schéma definované normou SQL:1999. Hlavní snahou navrženého mapování je, aby si objektově orientované rysy jazyka XML Schema a jim přiřazené objektově relační rysy normy SQL:1999 co nejvíce odpovídaly.

Při návrhu algoritmů je dále využita další z XML technologií – rozhraní DOM[3]. Myšlenka využití tohoto rozhraní je založena na faktu, že je každé schéma popsané v jazyce XML Schema opět XML dokumentem a může být tudíž také zpracováváno prostřednictvím tohoto rozhraní.

Jako vhodný dotazovací jazyk nad XML dokumenty je zvolen jazyk XQL[4], který není příliš složitý a současně má postačující vyjadřovací sílu. Z jeho specifikace je vybrána a implementována vhodná podmnožina, na níž je možné demonstrovat rysy navrženého řešení.

Při implementaci algoritmů je využito vhodných již implementovaných knihoven funkcí pro práci s XML dokumenty (např. knihovny pro vytváření DOM stromu daného XML dokumentu nebo XML parseru pro kontrolu validity XML dokumentu vůči danému XML schématu). Cílem práce tudíž není implementace veškeré správy XML dokumentů, nýbrž ukázková implementace navržených algoritmů.

1.2 Rozsah práce

Tato práce je logicky a tematicky rozdělena do následujících kapitol:

V první kapitole je popsán úvod do dané problematiky a seznámení s cílem práce.

Ve druhé kapitole jsou objasněny základní pojmy a popsány jednotlivé technologie, kterými se tato práce zabývá. Jak již bylo řečeno, jedná se především o jazyky XML a XML Schema, část normy jazyka SQL:1999, část specifikace dotazovacího jazyka XQL a programového rozhraní DOM.

Ve třetí kapitole jsou popsány a rozebrány existující přístupy k dané problematice a zhodnoceny jejich výhody a nevýhody. Na závěr kapitoly je pak nastíněno navržené řešení.

Podrobný popis navržených algoritmů je popsán ve čtvrté kapitole. Jsou to algoritmy mapování XML schématu popsaného v jazyce XML Schema na schéma objektově relační, ukládání XML dokumentů validních vůči danému XML schématu do odpovídajícího objektově relačního schématu a mapování jazyka XQL na posloupnost `SELECT` příkazů nad vytvořeným schématem.

[2]HDM (Hypergraph Data Model) – datový model vycházející z datové struktury hypergrafu (viz. [24])

[3]DOM (Document Object Model) – rozhraní pro práci s XML dokumenty (viz. [13])

[4]XQL (XML Query Language) – dotazovací jazyk nad XML dokumenty (viz. [27])

V páté kapitole je uveden popis architektury implementovaného systému a jeho ovládání.

Šestá (závěrečná) kapitola obsahuje zhodnocení navrženého řešení a nástin dalšího možného vývoje a vylepšení.

2 Použité technologie

V této kapitole jsou popsány jednotlivé technologie, kterými se tato práce zabývá – jazyky XML, XML Schema, SQL:1999 a XQL a programové rozhraní DOM.

Vzhledem k tomu, že se tato práce zabývá především jazykem XML Schema, je nejpodrobnější popis věnován právě jemu. V případě ostatních jazyků (a především v případě jazyka SQL:1999) jsou popsány pouze ty rysy, které jsou později použity v algoritmech mapování. V příslušných podkapitolách jsou dále uvedeny zdroje podrobných informací k danému tématu.

2.1 XML

Jazyk XML (Extensible Markup Language) byl standardizován konsorciem W3C[5] a jeho základem se stala nejpoužívanější podmnožina jazyka SGML, která byla dále omezena pevně určenými parametry a přísnou syntaxí. Je určen především pro ukládání, předávání a publikování semi-strukturovaných dat. Jedná se ve své podstatě o metajazyk, což znamená, že je určen pro popis struktury dalších jazyků.

2.1.1 Elementy

Základními prvky jazyka XML jsou *elementy*, které jsou do sebe vzájemně vnořovány. Elementy jsou určeny pomocí *značek*, přičemž většině elementů odpovídají dvě značky – *počáteční* a *koncová*. Výjimkou je *prázdný element*, který je určen pouze jednou značkou. Značky jsou uzavřeny mezi znaky ‘<’ a ‘>’, koncová značka má navíc na začátku znak ‘/’, značka pro prázdný element má znak ‘/’ na konci.

```
<kniha>
   <název>O mé rodině a jiné zvířeně</název>
   <autor>Gerald Durrell</autor>
   <obrázek/>
</kniha>
```

Každý neprázdný element musí být uzavřen mezi obě značky a toto uzavření musí být správně uzávorkované (tj. značky se nesmějí křížit). Dále musí být celý XML dokument uzavřen v právě jednom *kořenovém elementu*. Splňuje-li XML dokument tato *základní syntaktická pravidla*, říkáme, že je *správně strukturovaný* (well-formed).

Názvy, přípustný obsah a vzájemné vztahy elementů je možné definovat XML schématem popsaném v jazycích jako je DTD, XML Schema apod.

[5] W3C (World Wide Web Consortium) – `http://www.w3.org`

2.1.2 Atributy

Dalšími prvky XML dokumentu jsou *atributy*. Jsou součástí počáteční značky elementu a obsahují dodatečné informace vztahující se k danému elementu.

Každý atribut má dvě části – název a hodnotu uzavřenou do uvozovek, které oddělujeme znakem `'='`.

```
<kniha typ="povídky" cena="127 Kč">
   <název>O mé rodině a jiné zvířeně</název>
   <autor>Gerald Durrell</autor>
   <obrázek src="o_me_rodine.gif"/>
</kniha>
```

Názvy a datové typy atributů je také možné definovat prostřednictvím XML schématu.

2.1.3 XML deklarace

Na začátku XML dokumentu by měla být umístěna tzv. *XML deklarace*. Jedná se o element obsahující informaci o použité verzi XML (`version`), znakové sadě (`encoding`) a informaci o tom, zda je pro správnou interpretaci obsahu nutné použít externě definované deklarace značek (`standalone`).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

2.1.4 Další prvky XML dokumentů

Dalšími prvky, které se mohou v XML dokumentu vyskytovat jsou komentáře, sekce `CDATA` a instrukce pro zpracování.

Komentáře

Komentář je libovolný text uzavřený mezi znaky `<!--` a `-->`, který nepodléhá zpracování dokumentu.

Sekce `CDATA`

Sekce `CDATA` je vhodná pro případ, kdy potřebujeme do dokumentu vložit větší kus textu obsahujícího znaky se speciálním významem (`'<'`, `'>'`, `'&'` apod.). Uvnitř sekce `CDATA` (tj. mezi znaky `<![CDATA[` a `]]>`), je tento význam ignorován.

Instrukce pro zpracování

Instrukce pro zpracování jsou příkazy ve tvaru `<?`*«identifikátor» «data»*`?>` určené pro nadřazený program, které nepodléhají zpracování dokumentu.

2.1.5 Další informace

Základní informace týkající se jazyka XML je možné nalézt například v [19], podrobný popis pak v [10] nebo [12].

2.2 XML Schema

Jazyk XML Schema byl (stejně jako jazyk XML) standardizován konsorciem W3C a slouží k definování přípustné struktury daného XML dokumentu. Umožňuje určit elementy a atributy, které se smí v dokumentu vyskytovat, vztahy element-podelement, pořadí a počet podelementů v rámci nadelementu, přípustný obsah elementu, datové typy pro elementy i atributy a jejich implicitní hodnoty.

XML dokument, který odpovídá danému XML schématu (pokud nebude řečeno jinak, je v této kapitole pod pojmem XML schéma myšleno schéma XML dat popsané právě v jazyce XML Schema), označujeme jako *validní dokument*.

Je třeba zmínit fakt, že jazyk XML Schema umožňuje definovat tutéž věc několika způsoby. Na jedné straně tak dává uživateli do rukou silnější nástroj, na straně druhé je však specifikace o to složitější.

2.2.1 Jmenné prostory

Jmenný prostor je prostředí, ve kterém jsou všechna jména elementů a v kontextu jednoho elementu i jména atributů unikátní. Jmenné prostory jsou jednoznačně identifikovány svým URI[6] a umožňují definovat XML schéma pomocí různých *sad značek* (tj. množin elementů a atributů).

Jmenné prostory pro element a jeho podelementy určíme prostřednictvím atributu `xmlns:`*«prefix»*`=`*«URI sady značek»*. Při použití elementu nebo atributu z daného jmenného prostoru pak přidáme před jeho název námi definovaný prefix – *«prefix»*`:`*«název elementu»* nebo *«prefix»*`:`*«název atributu»*.

```
<ceník:nabídka xmlns:ceník="www.ceniky.cz/e-cenik"
               xmlns:bib="www.knihy.cz/bibliografie">
  <ceník:položka ceník:dph="22%">
    <ceník:název>
      <bib:kniha>
        <bib:název>O mé rodině a jiné zvířeně</bib:název>
        <bib:autor>Gerald Durrell</bib:autor>
      </bib:kniha>
    </ceník:název>
  </ceník:položka>
</ceník:nabídka>
```

[6] URI (Uniform Resource Identifier) – obecný identifikátor informačního zdroje

Pro zjednodušení je možné považovat jednu sadu značek za implicitní. V tom případě pro ni nespecifikujeme a nepoužíváme prefix. Další zjednodušení plyne z možnosti předefinování jmenného prostoru v podelementech.

Podrobnější informace je možné nalézt v [11].

2.2.2 Připojení XML schématu k dokumentu

Určení XML schématu příslušejícího danému XML dokumentu je možné provést dvěma způsoby. Pokud má odpovídající XML schéma definován cílový jmenný prostor (podrobněji viz. kapitola 2.2.3), provádí se připojení prostřednictvím atributu `schemaLocation`, který obsahuje název tohoto jmenného prostoru a URL dokumentu, v němž je schéma uloženo.

```
<?xml version="1.0">
<KořenovýElement
   xmlns="http://www.pr.cz/MojeSchema"
   xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
   xs:schemaLocation="http://www.pr.cz/MojeSchema
                      mojeSchema01.xsd">
  ...
</KořenovýElement>
```

Pokud odpovídající XML schéma jmenný prostor definován nemá, provádí se připojení prostřednictvím atributu `noNamespaceSchemaLocation`, který obsahuje pouze URL dokumentu, v němž je schéma uloženo. Všechny prvky XML dokumentu odpovídající tomuto schématu jsou potom uváděny bez prefixu.

```
<?xml version="1.0">
<KořenovýElement
   xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
   xs:noNamespaceSchemaLocation="mojeSchema02.xsd">
  ...
</KořenovýElement>
```

2.2.3 Prvky jazyka XML Schema

Jak již bylo řečeno, je syntaxe jazyka XML Schema založena na jazyku XML, čímž se dostáváme do situace, kdy elementy popisujeme opět pomocí elementů. Prvky jazyka XML Schema jsou tedy elementy jazyka XML.

Kořenový element

Každé XML schéma musí mít právě jeden kořenový element nazvaný `schema`. Instance tohoto elementu se v XML dokumentu nevyskytují. Na první úrovni se v XML dokumentu vyskytují až jeho potomci.

```
<?xml version="1.0">
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.priklad.cz"
           xs:targetNamespace="http://www.pr.cz/MojeSchema">
  ...
</xs:schema>
```

Atribut `xmlns:xs` indikuje, že všechny elementy a atributy s prefixem `xs` patří do jmenného prostoru jazyka XML Schema (tj. že se jedná o definici XML schématu). Atribut `xmlns` specifikuje implicitní jmenný prostor (tj. jmenný prostor pro prvky, jejichž název je uveden bez prefixu).

Dalšími atributy elementu `schema` je možné ovlivňovat celé XML schéma:

- `id` – identifikátor schématu
- `version` – verze schématu
- `xml:lang` – použitý jazyk (např. `en`, `en-GB`)
- `finalDefault` – implicitní hodnota atributu `final` v celém schématu
- `blockDefault` – implicitní hodnota atributu `block` v celém schématu
- `targetNamespace` – URI vytvářeného (cílového) jmenného prostoru
- `elementFormDefault` – implicitní hodnota atributu `form` všech elementů ve schématu
- `attributeFormDefault` – implicitní hodnota atributu `form` všech atributů ve schématu

Element `schema` může (mimo jiné) obsahovat podelementy `include`, `import` a `redefine`, které umožňují využití již definovaných (tzv. *externích*) XML schémat. Element `include` umožňuje zahrnout do cílového jmenného prostoru jmenný prostor externího schématu, element `import` umožňuje využívat při vytváření XML schématu prvky z externího schématu a element `redefine` umožňuje tyto prvky předefinovat.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include  schemaLocation="JineSchema1.xsd"/>
  <xs:import   namespace="http://www.pr.cz/JinyNamespace"/>
  <xs:redefine schemaLocation="JineSchema2.xsd">
    <xs:simpleType name="NovýŘetězec">
      <xs:restriction base="PůvodníŘetězec">
        <xs:maxLength value="5"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:redefine>
</xs:schema>
```

Vlastnosti elementu `include` ovlivňují atributy:

- `id` – identifikátor elementu `include`
- `schemaLocation` – URI schématu, jehož jmenný prostor zahrnujeme

Vlastnosti elementu `import` ovlivňují atributy:

- `id` – identifikátor elementu `import`
- `namespace` – URI jmenného prostoru, jehož prvky využíváme
- `schemaLocation` – URI schématu, jehož prvky využíváme

Vlastnosti elementu `redefine` ovlivňují atributy:

- `id` – identifikátor elementu `redefine`
- `schemaLocation` – URI schématu, jehož prvek předefinováváme

Elementy

Deklarace elementu je asociace názvu s definicí jednoduchého nebo složeného datového typu a případně implicitní hodnotou.

```
<xs:element name="Objednávka" type="TypObjednávky"/>

<xs:element name="Dárek">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Narozeniny" type="xs:date"/>
      <xs:element ref="Objednávka"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Vlastnosti elementu ovlivňují atributy elementu `element`:

- `id` – identifikátor elementu
- `name` – název elementu
- `type` – název datového typu elementu
- `ref` – odkaz na globálně deklarovaný element
- `nillable` – příznak, zda smí být instance elementu uvedena bez obsahu
- `default` – implicitní hodnota elementu uvedeného bez obsahu (pro elementy s jednoduchými typy)

- `fixed` – konstantní (jediná možná) hodnota elementu (pro elementy s jednoduchými typy)
- `minOccurs` – minimální nutný počet výskytů elementu
- `maxOccurs` – maximální možný počet výskytů elementu
- `form` – příznak, zda musí být název elementu uváděn s prefixem cílového jmenného prostoru (`qualified`) nebo ne (`unqualified`)
- `abstract` – příznak, zda je element *abstraktní*
- `substitutionGroup` – název *substituční skupiny* elementu
- `final` – v substituční skupině elementu se nesmí vyskytovat elementy, jejichž datové typy byly odvozeny rozšířením (`extension`), restrikcí (`restriction`) nebo libovolným způsobem (`#all`)
- `block` – za daný element není možné substituovat žádné elementy (`substitution`), elementy, jejichž datové typy byly odvozeny rozšířením (`extension`), restrikcí (`restriction`) nebo libovolným způsobem (`#all`)

Datový typ elementu můžeme určit hodnotou atributu `type` nebo jako podelement elementu `element` (viz. příklady). Pro zjednodušení definice XML schématu poskytují elementy následující mechanismy:

Globální a lokální elementy Deklarace elementu může být *globální* nebo *lokální* (pomocná deklarace v rámci definice složeného typu). Globální element je potomkem elementu `schema` a je viditelný v rámci celého XML schématu. Na takový element se můžeme dále odkazovat (atributem `ref`), čímž umožníme, aby se vyskytoval v kontextu aktuálního elementu a byl tak opakovaně využíván.

Substituce Elementy obsahují mechanismus substituce jednoho elementu za druhý. Elementy přiřadíme do *substituční skupiny* určené názvem *vedoucího elementu*, čímž umožníme jejich substituci za vedoucí element.

```
<xs:element name="Komentář"   type="xs:string"
            abstract="true"/>
<xs:element name="Komentář_1" type="xs:string"
            substitutionGroup="Komentář"/>
<xs:element name="Komentář_2" type="xs:string"
            substitutionGroup="Komentář"/>
```

Pokud vedoucí element označíme jako *abstraktní* (`abstract`), musí být na jeho místo vždy substituován jiný (neabstraktní) element. Vlastnosti substituční skupiny můžeme dále ovlivňovat pomocí atributů `final` a `block`. Atributem `final` ovlivňujeme strukturu substituční skupiny a atributem `block` substituce pro konkrétní instanci elementu.

Elementy v substituční skupině musí mít (pokud není parametrem `final` řečeno jinak) stejný typ jako vedoucí element nebo typ od něj odvozený.

Atributy

Deklarace atributu je asociace názvu s definicí jednoduchého datového typu, omezením výskytu a případně implicitní hodnotou.

```
<xs:attribute name="Věk" type="xs:positiveInteger"/>
```

Vlastnosti atributu ovlivňují atributy elementu `attribute`:

- `id` – identifikátor atributu
- `name` – název atributu
- `type` – název datového typu atributu
- `ref` – odkaz na globálně deklarovaný atribut
- `default` – implicitní hodnota atributu, není-li v dokumentu uveden
- `fixed` – konstantní (jediná možná) hodnota atributu
- `use` – nepovinný (`optional`), zakázaný (`prohibited`) nebo povinný (`required`) výskyt atributu
- `form` – příznak zda musí být název atributu uváděn s prefixem cílového jmenného prostoru (`qualified`) nebo ne (`unqualified`)

Podobně jako u elementu můžeme i datový typ atributu určit hodnotou atributu `type` nebo jako podelement elementu `attribute`. Stejný princip platí také pro globální a lokální deklarace atributů.

Jednoduché datové typy

Definice jednoduchého typu je množina omezení nad řetězcem a informací o hodnotách, které reprezentuje. Jednoduchý typ nesmí obsahovat elementy ani atributy.

Definice jednoduchého typu může být *vestavěná* nebo *uživatelsky definovaná*. Množinu vestavěných datových typů je možné dále dělit na *základní* (viz. tabulka 1) a *odvozené* typy (viz. tabulka 2 a 3).

Informace v tabulkách dále upřesňují následující poznámky:

- Hodnoty typů `dateTime`, `time`, `date`, `gYearMonth`, `gYear`, `gMonthDay`, `gDay` a `gMonth` lze specifikovat v UTC[7] (např. `15:30:25Z`) nebo s odchylkou od něj (např. `09:30:25+06:00`).
- Hodnoty typů `duration`, `dateTime`, `date`, `gMonth` a `gYear` mohou být i záporné.
- Vestavěné datové typy pojmenované výhradně velkými písmeny (a typy od nich odvozené) mohou být přiřazeny pouze atributům.

Uživatelsky definované typy jsou vždy odvozené. Existují tři způsoby jak ze základního nebo jiného odvozeného typu vytvořit nový odvozený typ:

1. Odvození restrikcí Pro každý jednoduchý datový typ je definována množina *parametrů* (všechny typy parametrů viz. tabulka 4). Restrikci provádíme nastavením zvolených parametrů na požadované hodnoty. Nový datový typ tedy představuje podmnožinu hodnot původního typu.

```
<xs:simpleType name="NeprázdnýŘetězec">
  <xs:restriction base="xs:string">
    <xs:minLength value="1"/>
  </xs:restriction>
</xs:simpleType>
```

Vlastnosti jednoduchého typu ovlivňují atributy elementu `simpleType`:

- `id` – identifikátor jednoduchého typu
- `name` – název jednoduchého typu
- `final` – z datového typu není možné odvozovat další typy restrikcí (`restriction`), seznamem (`list`), sjednocením (`union`) nebo libovolným způsobem (`#all`)

Vlastnosti odvození restrikcí ovlivňují atributy elementu `restriction`:

- `id` – identifikátor restrikce
- `base` – název datového typu, nad nímž provádíme restrikci

Datový typ, z něhož provádíme odvození restrikcí, můžeme určit hodnotou atributu `base` nebo jako podelement elementu `restriction`. (Přípustná omezení restrikcí pro jednotlivé vestavěné datové typy jsou uvedena v příloze A.1.)

[7] UTC (Coordinated Universal Time) – světový koordinovaný čas

Datový typ:	**Význam:**
`anyType`	Libovolný typ
`string`	Řetězec znaků
`boolean`	Logické hodnoty `true` a `false`, popř. `1` a `0`
`decimal`	Kladné nebo záporné reálné číslo (např. `-1.23`, `12678967.543233`, `210`)
`float`	32-bitové kladné nebo záporné reálné číslo vyjádřené pomocí mantisy a exponentu (např. `-1E4`, `1267.43233E12`, `12`). Může mít i speciální hodnoty `0`, `-0`, `INF`, `-INF` nebo `NaN`.
`double`	64-bitové číslo se stejnými vlastnostmi jako `float`
`duration`	Časový úsek ve tvaru `PnYnMnDTnHnMnS`, kde `P` a `T` jsou oddělovače, `nY` znamená `n` let a pod. (např. `-P1347M`, `P0Y1347M`)
`dateTime`	Datum a čas ve tvaru `YYYY-MM-DDThh:mm:ss.ss`, kde `T` je oddělovač, `YYYY` znamená rok apod.
`time`	Čas ve tvaru `hh:mm:ss.ss`
`date`	Datum ve tvaru `YYYY-MM-DD`
`gYearMonth`	Měsíc v roce ve tvaru `YYYY-MM`
`gYear`	Rok ve tvaru `YYYY`
`gMonthDay`	Den v měsíci ve tvaru `MM-DD`
`gMonth`	Měsíc ve tvaru `MM`
`gDay`	Den ve tvaru `DD`
`hexBinary`	Hexadecimální číslo
`base64Binary`	Binární data s kódováním *Base64*
`anyURI`	Absolutní nebo relativní URI
`QName`	*XML Qualified Name*, tj. řetězec ve tvaru *«prefix»*:*«místní část»*, kde *«prefix»* je označení jmenného prostoru a *«místní část»* je prvek patřící do daného jmenného prostoru
`NOTATION`	Datový typ pro název elementu `notation`

Tabulka 1: Základní vestavěné datové typy

Datový typ:	**Význam:**
`normalizedString`	Řetězec (`string`), který neobsahuje znaky `CR`, `LF` a tabulátor
`token`	Řetězec (`normalizedString`), který nemá mezery na začátku ani na na konci a neobsahuje posloupnost mezer delší než jedna
`language`	Identifikátor jazyka (např. `en`, `en-GB`)
`NMTOKEN`	Jednoslovná hodnota
`NMTOKENS`	Seznam jednoslovných hodnot
`Name`	*XML Name*, tj. řetězec, který smí obsahovat písmena, číslice, pomlčky, podtržítka, dvojtečky a tečky
`NCName`	*XML Name*, které nesmí obsahovat dvojtečky
`ID`	Hodnota jednoznačná v rámci celého schématu
`IDREF`	Odkaz na hodnotu datového typu ID
`IDREFS`	Seznam odkazů na hodnoty datového typu ID
`ENTITY`	Pojmenování entity (prvku) schématu
`ENTITIES`	Seznam pojmenování entit schématu

Tabulka 2: Vestavěné datové typy odvozené od typu `string`

Datový typ:	**Význam:**
`integer`	Kladné nebo záporné celé číslo
`positiveInteger`	Kladné celé číslo
`negativeInteger`	Záporné celé číslo
`nonPositiveInteger`	Nekladné celé číslo
`nonNegativeInteger`	Nezáporné celé číslo
`long`	Celé číslo z intervalu $<-2^{63},\ 2^{63}-1>$
`int`	Celé číslo z intervalu $<-2^{31},\ 2^{31}-1>$
`short`	Celé číslo z intervalu $<-2^{15},\ 2^{15}-1>$
`byte`	Celé číslo z intervalu $<-2^{7},\ 2^{7}-1>$
`unsignedLong`	Nezáporné číslo menší než 2^{64}
`unsignedInt`	Nezáporné číslo menší než 2^{32}
`unsignedShort`	Nezáporné číslo menší než 2^{16}
`unsignedByte`	Nezáporné číslo menší než 2^{8}

Tabulka 3: Vestavěné datové typy odvozené od typu `decimal`

Parametr:	**Význam:**
`length`	Počet jednotek daného typu (např. znaků v řetězci)
`minLength`	Minimální počet jednotek daného typu
`maxLength`	Maximální počet jednotek daného typu
`pattern`	Regulární výraz, kterému musí hodnoty daného typu vyhovovat
`enumeration`	Explicitně vyjmenovaná množina povolených hodnot daného typu
`whiteSpace`	Zpracování bílých znaků v řetězci – `preserve` (žádné změny), `replace` (znaky `CR`, `LF` a tabulátor jsou nahrazeny mezerou), `collapse` (totéž co `replace` a navíc jsou odstraněny mezery na začátku a na konci řetězce a posloupnosti mezer nahrazeny jednou mezerou)
`maxInclusive`	Hodnoty datového typu musí být menší nebo rovny zadané hodnotě
`minInclusive`	Hodnoty datového typu musí být větší nebo rovny zadané hodnotě
`maxExclusive`	Hodnoty datového typu musí být menší než zadaná hodnota
`minExclusive`	Hodnoty datového typu musí být větší než zadaná hodnota
`totalDigits`	Maximální počet cifer
`fractionDigits`	Maximální počet cifer za desetinnou čárkou

Tabulka 4: Parametry pro restrikci

2. Odvození seznamem Tímto typem odvození vznikne datový typ představující seznam určených datových typů. Jeho hodnota je v XML dokumentu zadávána jako seznam hodnot oddělených mezerami.

Existují tři vestavěné datové typy odvozené seznamem – `NMTOKENS`, `IDREFS` a `ENTITIES`.

```
<xs:simpleType name="SeznamReálnýchČísel">
  <xs:list itemType="xs:float"/>
</xs:simpleType>
```

Vlastnosti odvození seznamem ovlivňují atributy elementu `list`:

- `id` – identifikátor odvození seznamem
- `itemType` – název datového typu, jehož seznam vytváříme

Datový typ, z něhož provádíme odvození seznamem, můžeme určit hodnotou atributu `itemType` nebo jako podelement elementu `list`. Odvození seznamem není možné provádět z jiného typu odvozeného seznamem.

3. Odvození sjednocením Tímto typem odvození vznikne datový typ představující sjednocení hodnot všech určených datových typů.

```
<xs:simpleType name="SjednoceníTypů">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:positiveInteger">
        <xs:minInclusive value="8"/>
        <xs:maxInclusive value="72"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="small"/>
        <xs:enumeration value="large"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

Vlastnosti odvození sjednocením ovlivňují atributy elementu `union`:

- `id` – identifikátor jednoduchého typu
- `memberTypes` – seznam názvů sjednocovaných typů

Datové typy, z nichž provádíme odvození sjednocením, můžeme určit hodnotami atributu `memberTypes` nebo jako podelementy elementu `union`.

Složené datové typy

Definice složeného typu je množina deklarací atributů a jednoduchých nebo složených datových typů.

```
<xs:complexType name="Adresa">
  <xs:sequence>
    <xs:element name="Jméno" type="xs:string"/>
    <xs:element name="Ulice" type="xs:string"/>
    <xs:element name="Čdomu" type="xs:unsignedShort"/>
  </xs:sequence>
  <xs:attribute name="Město" type="xs:string"/>
</xs:complexType>
```

Vlastnosti složeného typu ovlivňují atributy elementu `complexType`:

- `id` – identifikátor složeného typu
- `name` – název složeného datového typu
- `mixed` – příznak *smíšeného obsahu* (tj. element s daným typem může kromě určených podelementů obsahovat libovolný další text)
- `abstract` – příznak, zda je složený typ *abstraktní* (tj. zda smí být přiřazen některému elementu)
- `block` – za element s tímto datovým typem není možné substituovat elementy odvozené restrikcí (`restriction`), rozšířením (`extension`) nebo libovolným způsobem (`#all`)
- `final` – z datového typu není možné odvozovat další typy rozšířením (`extension`), restrikcí (`restriction`) nebo libovolným způsobem (`#all`)

Existuje šest typů složených datových typů, které jsou určeny podelementy elementu `complexType`:

1. S jednoduchým obsahem (`simpleContent`) Datový typ s jednoduchým obsahem obsahuje restrikci (`restriction`) jednoduchého typu nebo jeho rozšíření (`extension`) o atributy.

```
<xs:complexType name="Cena">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="Měna" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Vlastnosti datového typu s jednoduchým obsahem ovlivňují atributy elementu `simpleContent`:

- `id` – identifikátor datového typu s jednoduchým obsahem

Vlastnosti rozšíření ovlivňují atributy elementu `extension`:

- `id` – identifikátor rozšíření
- `base` – název rozšiřovaného datového typu

2. Se složeným obsahem (`complexContent`) Datový typ se složeným obsahem obsahuje rozšíření nebo restrikci některého z následujících čtyř typů složeného typu. Restrikce znamená vytvoření nového složeného typu, který je podmnožinou typu původního (např. omezením hodnot, omezením počtu výskytů apod.). Rozšířením vznikne složený typ, který obsahuje původní i nový typ (v tomto pořadí).

```
<xs:complexType name="Adresa">
  <xs:sequence>
    <xs:element name="ulice" type="xs:string"/>
    <xs:element name="město" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="USAdresa">
  <xs:complexContent>
    <xs:extension base="Adresa">
      <xs:sequence>
        <xs:element name="stát" type="xs:string"/>
        <xs:element name="zip"  type="xs:positiveInteger"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Vlastnosti datového typu se složeným obsahem ovlivňují atributy elementu `complexContent`:

- `id` – identifikátor datového typu se složeným obsahem
- `mixed` – příznak smíšeného obsahu

Vlastnosti restrikce složeného datového typu ovlivňují atributy elementu `restriction`:

- `id` – identifikátor restrikce složeného datového typu

- `base` – název omezovaného datového typu

Vlastnosti rozšíření složeného datového typu ovlivňují atributy elementu `extension`:

- `id` – identifikátor rozšíření složeného datového typu
- `base` – název rozšiřovaného datového typu

3. Posloupnost elementů (`sequence`) Posloupnost elementů je datový typ obsahující množinu elementů s pevně daným pořadím.

```
<xs:complexType name="Zvířata">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="Slon"/>
    <xs:element ref="Medvěd"/>
    <xs:element ref="Žížala"/>
  </xs:sequence>
</xs:complexType>
```

Vlastnosti posloupnosti elementů ovlivňují atributy elementu `sequence`:

- `id` – identifikátor posloupnosti
- `minOccurs` – minimální nutný počet výskytů posloupnosti
- `maxOccurs` – maximální možný počet výskytů posloupnosti

4. Výběr z elementů (`choice`) Výběr z elementů je datový typ obsahující jeden element z dané množiny.

```
<xs:complexType name="Stav">
  <xs:choice minOccurs="1" maxOccurs="1">
    <xs:element ref="Zvolený"/>
    <xs:element ref="Nezvolený"/>
  </xs:choice>
</xs:complexType>
```

Vlastnosti výběru z elementů ovlivňují atributy elementu `choice`:

- `id` – identifikátor výběru
- `minOccurs` – minimální nutný počet výskytů výběru
- `maxOccurs` – maximální možný počet výskytů výběru

5. Množina elementů (`all`) Množina elementů je datový typ obsahující množinu elementů s libovolným pořadím.

```
<xs:complexType  name="MojeSkupinaVěcí">
  <xs:all>
    <xs:element name="Věc1" type="xs:string"/>
    <xs:element name="Věc2" type="xs:string"/>
  </xs:all>
</xs:complexType>
```

Vlastnosti množiny elementů ovlivňují atributy elementu `all`:

- `id` – identifikátor množiny elementů
- `minOccurs` – minimální nutný počet výskytů množiny elementů
- `maxOccurs` – maximální možný počet výskytů množiny elementů

6. Modelová skupina (`group`) Modelová skupina je datový typ, který jako podelement obsahuje element `choice`, `sequence` nebo `all`, tj. skupinu elementů s určitou vlastností.

```
<xs:group name="MojeSkupinaVěcí">
  <xs:all>
    <xs:element name="Věc1" type="xs:string"/>
    <xs:element name="Věc2" type="xs:string"/>
  </xs:all>
</xs:group>

<xs:complexType name="MůjKomplexníTyp">
  <xs:group     ref="MojeSkupinaVěcí"/>
  <xs:attribute name="MůjAtribut" type="xs:decimal"/>
</xs:complexType>
```

Vlastnosti modelové skupiny ovlivňují atributy elementu `group`:

- `id` – identifikátor modelové skupiny
- `name` – název modelové skupiny
- `ref` – odkaz na globálně deklarovanou modelovou skupinu
- `minOccurs` – minimální nutný počet výskytů modelové skupiny
- `maxOccurs` – maximální možný počet výskytů modelové skupiny

Výhodou (a důvodem existence) modelové skupiny je, že je vždy deklarována globálně a tudíž je možné stejnou skupinu elementů opakovaně využívat v různých složených typech.

Skupina atributů

Definice skupiny atributů je asociace názvu s množinou deklarací atributů, umožňující využití stejné skupiny atributů v různých složených typech.

```
<xs:attributeGroup name="MojeSkupinaAtributů">
  <xs:attribute name="Atribut1" type="xs:integer"/>
  <xs:attribute name="Atribut2" type="xs:string"/>
</xs:attributeGroup>

<xs:complexType name="MůjSloženýTyp">
  <xs:attributeGroup ref="MojeSkupinaAtributů"/>
</xs:complexType>
```

Vlastnosti skupiny atributů ovlivňují atributy elementu `attributeGroup`:

- `id` – identifikátor skupiny atributů
- `name` – název skupiny atributů
- `ref` – odkaz na globálně deklarovanou skupinu atributů

Omezení identity

Definice omezení identity je asociace názvu s omezením na unikátnost nebo klíč. Každé omezení identity obsahuje výraz jazyka XPath[8], který vybírá množinu prvků (oblast), v níž pro určené prvky (elementy, atributy nebo jejich kombinace) dané omezení platí.

Existují tři typy omezení identity:

1. Unikátnost (`unique`) Omezení na unikátnost specifikuje, že musí být hodnota daného elementu, atributu nebo jejich kombinace v rámci dané oblasti unikátní.

```
<xs:unique name="unikátníISBN">
  <xs:selector xpath="kniha"/>
  <xs:field    xpath="@ISBN"/>
</xs:unique>
```

Vlastnosti omezení na unikátnost ovlivňují atributy elementu `unique`:

- `id` – identifikátor omezení na unikátnost
- `name` – název omezení na unikátnost

[8]XPath (XML Path Language) – dotazovací jazyk nad XML dokumenty (viz. [14], popř. kapitola 2.3)

Element `unique` obsahuje právě jeden podelement `selector` a alespoň jeden podelement `field`. Element `selector` určuje množinu prvků (elementů nebo atributů), v rámci níž musí být prvky určené elementem `field` unikátní.

Vlastnosti elementu `selector` ovlivňují následující atributy:

- `id` – identifikátor elementu `selector`
- `xpath` – výraz jazyka XPath

Vlastnosti elementu `field` ovlivňují následující atributy:

- `id` – identifikátor elementu `field`
- `xpath` – výraz jazyka XPath

2. Klíč (`key`) Omezení na klíč specifikuje, že hodnota daného elementu, atributu nebo jejich kombinace musí být v rámci dané oblasti klíčem. *Klíč* je unikátní, nenulová a vždy definovaná hodnota.

```
<xs:key name="klíčemJeCeléJméno">
  <xs:selector xpath="osoba/údaje"/>
  <xs:field    xpath="jméno"/>
  <xs:field    xpath="příjmení"/>
</xs:key>
```

Vlastnosti klíče ovlivňují atributy elementu `key`:

- `id` – identifikátor klíče
- `name` – název klíče

Element `key` obsahuje podelementy `selector` a `field` s obdobným významem jako v případě elementu `unique`.

3. Cizí klíč (`keyref`) Omezení na cizí klíč specifikuje, že hodnota daného elementu, atributu nebo jejich kombinace v rámci dané oblasti odpovídá hodnotě specifikované daným elementem `key` nebo `unique`.

```
<xs:keyref name="odkazNaOsobu" refer="klíčemJeCeléJméno">
  <xs:selector xpath="zaměstnanec/osobní"/>
  <xs:field    xpath="jméno"/>
  <xs:field    xpath="příjmení"/>
</xs:keyref>
```

Vlastnosti cizího klíče ovlivňují atributy elementu `keyref`:

- `id` – identifikátor cizího klíče
- `name` – název cizího klíče
- `refer` – název elementu `key` nebo `unique`

Element `keyref` obsahuje podelementy `selector` a `field` s obdobným významem jako v případě elementu `unique`.

Zástupci

Zástupci jsou prvky umožňující vložit na dané místo element (`any`) nebo atribut (`anyAttribute`) z určeného jmenného prostoru nezávisle na aktuálním kontextu.

```
<xs:complexType name="LibovolnýHTMLtext">
  <xs:sequence>
    <xs:any namespace="http://www.w3.org/1999/xhtml"
            minOccurs="1" maxOccurs="unbounded"
            processContents="lax"/>
  </xs:sequence>
  <xs:anyAttribute namespace="http://www.w3.org/1999/xhtml"/>
</xs:complexType>
```

Vlastnosti elementu `any` ovlivňují následující atributy:

- `id` – identifikátor zástupce pro element
- `minOccurs` – minimální nutný počet výskytů zástupce pro element
- `maxOccurs` – maximální nutný počet výskytů zástupce pro element
- `namespace` – určení jmenného prostoru přípustných elementů – URI konkrétního jmenného prostoru, libovolný jmenný prostor (`##any`), cílový jmenný prostor schématu (`##targetNamespace`), prostor jiný než cílový jmenný prostor (`##other`) nebo povolení elementů bez prefixu jmenného prostoru (`##local`)
- `processContents` – způsob validace zástupců pro elementy – přísná validace zástupců vůči příslušným jmenným prostorům (`strict`), validace zástupců probíhá pouze vůči známým schématům jmenných prostorů (`lax`) nebo žádná validace (`skip`)

Vlastnosti elementu `anyAttribute` ovlivňují následující atributy:

- `id` – identifikátor zástupce pro atribut
- `namespace` – určení jmenného prostoru přípustných atributů
- `processContents` – způsob validace zástupců pro atributy

Notace

Deklarace notace je asociace názvu s identifikátorem notace určené pro popis jiných než XML dat v XML dokumentu.

```
<xs:notation name="jpeg"
             public="image/jpeg" system="viewer.exe"/>
```

Vlastnosti notace ovlivňují atributy elementu `notation`:

- `id` – identifikátor notace
- `name` – název notace
- `public` – URI veřejného identifikátoru
- `system` – URI systémového identifikátoru

Anotace

Anotace je informace určená pro člověka nebo aplikaci zpracovávající XML dokument. Interpretace této informace není definována.

```
<xs:annotation>
  <xs:documentation>Informace pro čtenáře</xs:documentation>
  <xs:appinfo>Informace pro aplikaci</xs:appinfo>
</xs:annotation>
```

Vlastnosti anotace ovlivňují atributy elementu `annotation`:

- `id` – identifikátor anotace

Element `annotation` může obsahovat podelementy `documentation` (informace pro čtenáře) nebo `appinfo` (informace pro aplikaci).

Element `documentation` má následující atributy:

- `source` – URI zdroje informací pro čtenáře
- `xml:lang` – použitý jazyk

Element `appinfo` má následující atributy:

- `source` – URI zdroje informací pro aplikaci
- `xml:lang` – použitý jazyk

2.2.4 Další informace

Základní informace o jazyku XML Schema je možné nalézt v [1], podrobný popis pak v [16], [30] nebo [3].

2.3 XQL

Jazyk XQL (XML Query Language) je jedním z několika dotazovacích jazyků přizpůsobených přímo pro dotazování nad XML dokumenty. Shodou okolností existují v současné době dva jazyky s tímto názvem, ale s odlišnou syntaxí. V této práci je využíván jazyk specifikovaný v [27].

Na tomto místě je vhodné poznamenat, že se jazyk XQL velmi „podobá“ již zmíněném jazyku XPath. Jak je uvedeno např. v [26], mají oba jazyky společnou poměrně velkou podmnožinu a liší se pouze v oblastech, pro které nebyl daný jazyk primárně určen.

V této kapitole jsou stručně popsány základní rysy jazyka XQL a především pak ty, které budou později použity v navržených algoritmech.

2.3.1 Základní charakteristika

Jazyk XQL je určen pro adresování částí XML dokumentu. Umožňuje buď přímé odkazování na přesně určené elementy nebo na množinu různých elementů v daném místě XML dokumentu.

Základní syntaxe jazyka připomíná URL adresáře nebo souboru. Místo navigace hierarchií adresářů a podadresářů v souborovém systému navigujeme hierarchií elementů a podelementů v XML dokumentu. Např. výraz

```
/knihovna/kniha/autor
```

vyjadřuje dotaz na podelementy `autor` elementů `kniha`, které jsou podelementy kořenového elementu `knihovna`.

Důležitým pojmem souvisejícím s jazykem XQL je *kontext*. Kontext je množina uzlů, vůči nimž specifikujeme dotaz, tj. množina na níž závisí výsledek dotazu. Např. ptáme-li se na elementy `X` vůči kořenovému elementu, bude odpovědí na dotaz jiná množina než v případě, kdy se ptáme na elementy `X` pouze vůči určité větvi stromu elementů. Implicitně dotaz využívá aktuální kontext (můžeme jej explicitně specifikovat pomocí znaku `'.'`). Oproti tomu dotazy začínající znakem `'/'` využívají kontext kořenového elementu.

2.3.2 Operátory

Jak již bylo řečeno, zadáním přesného názvu elementu specifikujeme množinu elementů jednoho (zadaného) typu. Množinu různých elementů je možné vyjádřit prostřednictvím tzv. *operátorů*.

Operátor `*`

Operátor `*` zastupuje všechny přímé podelementy aktuálního kontextu bez ohledu na jejich název. Např. výraz

```
/knihovna/*/autor
```

vyjadřuje dotaz na všechny elementy `autor`, které jsou podelementem libovolného podelementu kořenového elementu `knihovna`.

Operátor //

Operátor `//` označuje všechny podelementy aktuálního kontextu umístěné v libovolné hloubce. Např. výraz

```
//autor
```

vyjadřuje dotaz na všechny elementy `autor`, které jsou uvedeny kdekoli v XML dokumentu.

2.3.3 Atributy

Jazyk XQL dále umožňuje specifikovat dotazy přímo na atributy elementů. Dotaz na atribut specifikujeme stejně jako dotaz na element s tím rozdílem, že před název atributu přidáme znak `'@'`. Např. výraz

```
/knihovna/kniha/@cena
```

vyjadřuje dotaz na hodnotu atributu `cena` podelementů `kniha` kořenového elementu `knihovna`.

2.3.4 Filtry

Množinu výsledků dotazu je možné dále omezit prostřednictvím tzv. *filtrů*. Filtr je podmínka, kterou je možné specifikovat uvnitř znaků `'['` a `']'` kdekoli v XQL dotazu. Podmínka má hodnotu `TRUE` nebo `FALSE`, přičemž v případě specifikace dotazu na množinu hodnota `TRUE` znamená, že je specifikovaná množina neprázdná. Např. výraz

```
/knihovna/kniha[autor]
```

vyjadřuje dotaz na všechny podelementy `kniha` kořenového elementu `knihovna`, které mají alespoň jeden podelement `autor`.

Logické operátory

V podmínkách filtrů je možné využívat logické operátory `$and$` (logické „a zároveň“), `$or$` (logické „nebo“) a `$not$` (negace). Např. výraz

```
/knihovna/kniha[autor $and$ $not$ rok]
```

vyjadřuje dotaz na všechny podelementy `kniha` kořenového elementu `knihovna`, které mají alespoň jeden podelement `autor` a současně nemají žádný podelement `rok`.

Rovnost

V podmínkách filtrů je dále možné využívat operátory pro porovnávání (=, !=, >, <, >= a <=). Např. výraz

```
/knihovna/kniha[autor = "Jim Melton"]
```

vyjadřuje dotaz na všechny podelementy `kniha` kořenového elementu `knihovna`, jejichž podelement `autor` má hodnotu `"Jim Melton"`.

Indexy

Na jednotlivé prvky množiny výsledků se můžeme odkazovat pomocí indexů. Index prvního prvku je `0`, index posledního prvku vrací funkce `end()`. Např. výraz

```
/knihovna/kniha[end()]
```

vyjadřuje dotaz na poslední podelement `kniha` kořenového elementu `knihovna`.

2.3.5 Další informace

Další informace o jazyku XQL je možné nalézt v [27].

2.4 DOM

Programové rozhraní DOM (Document Object Model) bylo také standardizováno konsorciem W3C. Toto rozhraní umožňuje pracovat s XML dokumentem jako se stromem, jehož uzly tvoří části XML dokumentu.

Vzhledem k tomu, že je standard navržen tak, aby byl co nejobecnější, jsou v této kapitole popsány pouze ty části, které budou později využívány v navržených algoritmech.

2.4.1 Základní charakteristika

Jak již bylo řečeno, DOM je definice programového rozhraní což znamená, že nezávisí na konkrétním programovacím jazyku. Standard tedy pouze definuje třídy, jejich vlastnosti a metody, které mají být při práci s dokumentem používány.

DOM může být využíváno ke zpracovávání existujícího XML dokumentu nebo k jeho vytváření a modifikaci (tj. uzly stromu můžeme nejen procházet, ale také vytvářet a rušit). Ovšem jelikož je celý DOM strom uložen v paměti, není vhodným rozhraním pro příliš velké XML dokumenty.

2.4.2 Uzly DOM stromu

Existuje několik různých typů uzlů DOM stromu, jejichž vlastnosti odpovídají různým částem XML dokumentu, které reprezentují. Některé typy uzlů mohou mít další poduzly, jiné mohou tvořit pouze listy DOM stromu.

Uzel `Node`

Všechny typy uzlů jsou odvozeny od základního uzlu `Node`, který obsahuje společné vlastnosti a metody. Mezi jeho nejdůležitější vlastnosti patří:

- `nodeType` – identifikátor typu uzlu (např. 1 = element, 2 = atribut…)
- `nodeName` – název uzlu (např. název elementu nebo atributu)
- `nodeValue` – hodnota uzlu (např. hodnota atributu)
- `parentNode` – odkaz na naduzel aktuálního uzlu
- `childNodes` – seznam poduzlů aktuálního uzlu (např. seznam podelementů aktuálního elementu)
- `attributes` – seznam atributů (uzlů typu atribut) aktuálního uzlu

Mezi nejdůležitější metody uzlu `Node` patří metody pro vkládání, odebírání, výměnu a zjišťování existence poduzlů a metoda vytvářející duplikát aktuálního uzlu.

Další typy uzlů

Jelikož jsou všechny typy uzlů odvozeny od uzlu `Node`, mají všechny výše popsané metody a vlastnosti. Každý typ dále obsahuje funkce nebo vlastnosti specifické pro danou část XML dokumentu.

Mezi základní nelistové uzly patří:

- `Document` – kořenový element XML dokumentu; obsahuje funkce pro vytváření ostatních typů uzlů DOM stromu
- `Element` – element; obsahuje funkce pro práci se svými atributy (vytváření, rušení, změnu hodnoty a pod.)
- `Attr` – atribut
- `DocumentFragment` – libovolná část dokumentu s jediným kořenovým elementem

Mezi základní listové uzly patří:

- `Text` – textový obsah uzlů `Element` a `Attr`

- `ProcessingInstruction` – instrukce pro zpracování
- `Comment` – text komentáře
- `CDATASection` – sekce CDATA

2.4.3 Další informace

Další informace o programovém rozhraní DOM je možné nalézt v [13].

2.5 SQL:1999

Norma jazyka SQL:1999 byla standardizována v roce 1999 jako další z řady norem jazyka SQL (Structured Query Language). Obsahuje velkou část normy SQL-92, řadu nových prvků a zejména úplně nový rys – objektovou orientaci, čímž se z jazyka SQL stává jazyk objektově relační.

Jelikož je norma SQL:1999 značně obsáhlá, jsou v této kapitole popsány pouze ty rysy, které budou později použity v algoritmech mapování.

2.5.1 Vestavěné datové typy

Norma SQL:1999 definuje množinu vestavěných datových typů, které je možné dále dělit na jednoduché a složené. Jednoduché datové typy mohou mít vždy právě jednu hodnotu z dané množiny přípustných hodnot. Většina jednoduchých datových typů jazyka SQL:1999 byla převzata z jazyka SQL-92, některé z nich (např. `BLOB`, `CLOB`, `BOOLEAN`) jsou nové. Zcela novými datovými typy jsou také složené datové typy, které mohou obsahovat více hodnot stejného (`ARRAY`) nebo různého (`ROW`) datového typu.

Přesné numerické typy

Mezi přesné numerické typy patří celočíselné typy `INTEGER` (zkráceně `INT`) a `SMALLINT` a reálná čísla `NUMERIC` a `DECIMAL`.

Přesnost typů `INT` a `SMALLINT` závisí na implementaci, ale musí platit, že pro `SMALLINT` není větší než pro `INT`. Přesnost typu `NUMERIC` se zadává jako `NUMERIC(p,q)`, kde `p` je počet cifer a `q` umístění desetinné čárky zprava. Typ `DECIMAL` se zadává podobně.

Aproximativní numerické typy

Mezi aproximativní numerické typy patří typy `REAL`, `DOUBLE PRECISION` (zkráceně `DOUBLE`) a `FLOAT`.

Přesnost typu `REAL` a `DOUBLE` závisí na implementaci, přičemž pro `DOUBLE` musí být větší než pro `REAL`. Přesnost typu `FLOAT` se zadává jako `FLOAT(p)`, kde `p` je obvykle počet bitů.

Znakové řetězce

Pro znakové řetězce existují typy `CHARACTER` (zkráceně `CHAR`), `CHARACTER VARYING` (zkráceně `CHAR VARYING` nebo `VARCHAR`) a nově i `CHARACTER LARGE OBJECT` (zkráceně `CHAR LARGE OBJECT` nebo `CLOB`).

`CHAR(n)` odpovídá řetězci znaků délky `n`, přičemž kratší řetězce jsou zprava doplněny mezerami. `VARCHAR(n)` odpovídá řetězci znaků maximální délky `n` (kratší řetězce se nedoplňují). `CLOB(N)` odpovídá řetězci znaků délky `N`, přičemž `N` může být „velmi velké“, popř. pro zjednodušení zadáno v kilo (`K`), mega (`M`) nebo giga (`G`) bytech.

Bitové a binární řetězce

Pro bitové a binární řetězce existují typy `BIT`, `BIT VARYING` a `BINARY LARGE OBJECT` (zkráceně `BLOB`).

`BIT(n)` odpovídá řetězci bitů délky `n`. `BIT VARYING(n)` odpovídá řetězci bitů maximální délky `n`. `BLOB(N)` odpovídá řetězci bytů délky `N`, přičemž `N` může být „velmi velké“, popř. pro zjednodušení zadáno v kilo (`K`), mega (`M`) nebo giga (`G`) bytech.

Časové datové typy

Mezi časové datové typy patří datové typy `DATE`, `TIME` a `TIMESTAMP`.

Typ `DATE` slouží pro uložení datumu ve tvaru `YYYY-MM-DD`. Typ `TIME` slouží pro uložení času ve tvaru `HH:MM:SS`, přičemž zadáním `TIME(p)` je možné určit přesnost počtu vteřin na `p` desetinných míst. Typ `TIMESTAMP` slouží pro uložení datumu i času ve tvaru `YYYY-MM-DD HH:MM:SS`, přičemž zadáním `TIMESTAMP(p)` je možné určit přesnost počtu vteřin na `p` desetinných míst.

Pro typy `TIME` a `TIMESTAMP` existuje varianta s příponou `WITH TIMEZONE` umožňující zadat příslušné hodnoty s odchylkou od UTC ve tvaru `HH:MM`.

Intervaly

Pro určení časového intervalu existují dva datové typy – intervaly typu rok-měsíc a intervaly typu den-čas.

Intervaly typu rok-měsíc mohou obsahovat roky a/nebo měsíce (`INTERVAL YEAR`, `INTERVAL MONTH` nebo `INTERVAL YEAR TO MONTH`). Zadáním `INTERVAL YEAR(p)`, `INTERVAL MONTH(p)` apod. je možné určit přesnost počtu let nebo měsíců na `p` desetinných míst. Intervaly typu den-čas mohou obsahovat dny, hodiny, minuty a/nebo vteřiny (`INTERVAL DAY TO HOUR`, `INTERVAL SECOND`, `INTERVAL DAY TO SECOND` apod.). Zadáním `INTERVAL DAY(p) TO HOUR(q)`, `INTERVAL SECOND(p)` apod. je možné určit přesnost počtu dnů, hodin, minut nebo vteřin na `p` respektive `q` desetinných míst.

Typ Boolean

Datový typ `BOOLEAN` slouží pro uložení logických hodnot `TRUE` nebo `FALSE` (a kvůli kompatibilitě i hodnoty `UNKNOWN`).

Kolekce

Prozatím byl do normy SQL:1999 začleněn pouze jeden datový typ kolekce – datový typ pole. Jiné datové typy (množina, multimnožina, seznam) budou zřejmě začleněny do některé z příštích norem.

Datový typ pole slouží pro uložení několika hodnot stejného datového typu (jakéhokoli s výjimkou pole) do jedné buňky tabulky. Např. zadáním `INT ARRAY[10]` specifikujeme pole až deseti hodnot typu `INT`. Nad poli je možné provádět následující typy operací:

- Přistupovat k jednotlivým položkám prostřednictvím indexu
- Zjistit velikost pole prostřednictvím funkce `CARDINALITY`
- Přistupovat k celému poli jeho odhnízděním do tabulky prostřednictvím operátoru `UNNEST`

Typ řádek

Datový typ řádek slouží pro uložení několika hodnot různého datového typu do jedné buňky tabulky – např. zadáním `ROW (jméno VARCHAR(10), příjmení VARCHAR(25), věk INT)` specifikujeme typ obsahující dvě hodnoty typu `VARCHAR` a jednu hodnotu typu `INT`. Na jednotlivé položky lze pak přistupovat prostřednictvím tečkové notace.

Datový typ řádek není globálně viditelný ani persistentní, tj. definici je třeba zopakovat tolikrát, kolikrát je daný datový typ použit.

2.5.2 Uživatelsky definované datové typy

Dalším, zcela novým prvkem jazyka SQL:1999 jsou uživatelsky definované typy (UDT). Existují dvě varianty UDT – odlišující (distinct) typy a strukturované typy, k nimž navíc přísluší speciální typy, tzv. *reference*. Prvky obou skupin jsou pojmenované a persistentní, tj. jejich definici je možné využívat opakovaně.

Odlišující typy

Odlišující datové typy zajišťují silnou typovou kontrolu jednoduchých datových typů. Např. definováním `CREATE TYPE Koruna AS DECIMAL(5,2)` a `CREATE TYPE Obsah AS DECIMAL(5,2)` vzniknou různé datové typy `Koruna` a `Obsah`, které není možné (bez explicitního přetypování) vzájemně porovnávat, dosazovat apod., přestože byly odvozeny od totožného typu.

Strukturované typy

Strukturované datové typy připomínají třídy objektově orientovaných programovacích jazyků. Jsou tvořeny množinou atributů (tj. proměnných vestavěného nebo UDT typu), popř. jejich implicitních hodnot a metod, které s těmito atributy pracují.

```
CREATE TYPE Obdélník AS (
  strana_a REAL DEFAULT 2.5,
  strana_b REAL DEFAULT 1.5)
  METHOD Obsah() RETURNS REAL;

CREATE INSTANCE METHOD Obsah() RETURNS REAL FOR Obdélník
  RETURN SELF.strana_a * SELF.strana_b;
```

Jak je vidět z příkladu, k jednotlivým atributům UDT lze (podobně jako u typu řádek) přistupovat pomocí tečkové notace.

Dalším objektově orientovaným rysem strukturovaných typů je *dědičnost*. Norma připouští pouze jednonásobnou dědičnost, při níž podtyp dědí od nadtypu všechny atributy a metody. Zděděné metody je navíc možné *překrývat*, tj. v podtypu znovu nadefinovat se stejným názvem a parametry, ale jiným obsahem.

Pro dědičnost je dále možné specifikovat následující parametry:

- `[NOT] INSTANTIABLE` – je/není možné vytvářet instance daného typu
- `[NOT] FINAL` – od daného typu je/není možné vytvářet odvozené typy[9]

Datové typy s parametrem `NOT INSTANTIABLE` jsou obvykle označovány jako *abstraktní datové typy*.

```
CREATE TYPE Osoba AS (
   id INT,
   jméno VARCHAR(10),
   příjmení VARCHAR(35) ) INSTANTIABLE NOT FINAL;

CREATE TYPE Student UNDER Osoba AS (
  ročník INT,
  obor CHAR ) INSTANTIABLE FINAL;
```

Dalším pojmem souvisejícím s dědičností a překrýváním metod je *substituovatelnost*, tj. vlastnost podtypů vyskytovat se na místě svého nadtypu. Se substituovatelností je dále spjat tzv. *typový predikát* umožňující určit typ, který je v dané proměnné aktuálně uložen. Např. definováním `osoba_1 IS OF (Osoba)` můžeme určit, zda je v proměnné `osoba_1` uložen typ `Osoba` nebo některý z jeho podtypů, definováním `osoba_2 IS OF (ONLY Osoba)` zda je v proměnné `osoba_2` uložen právě typ `Osoba`.

[9]Norma SQL:1999 připouští parametr `NOT FINAL` pouze pro strukturované typy a parametr `FINAL` pouze pro odlišující typy. Obě možnosti (které jsou již obvykle podporovány) budou zřejmě začleněny do některé z příštích norem.

Reference

Reference je datový typ, který vždy přísluší k určitému strukturovanému typu. Princip referencí připomíná ukazatele na objekty v objektově orientovaných jazycích. Např. definováním `zaměstnanec REF (Osoba)` specifikujeme proměnnou `zaměstnanec` typu reference na typ `Osoba`. K jednotlivým atributům referencovaného UDT přistupujeme prostřednictvím operátoru `->`.

Konkrétní instance strukturovaného typu na který se reference odkazuje by měla být jednoznačně identifikovatelná. Při definici strukturovaného typu může být tato identifikace určena některým z následujících způsobů:

- `REF IS SYSTEM GENERATED` – systémově generovaná
- `REF USING` *«datový typ»* – uživatelsky definovaná daným typem
- `REF FROM` *«seznam atributů»* – odvozená z daných atributů

Při vytvoření proměnné typu reference je dále možné specifikovat, zda má být kontrolována *referenční integrita* tohoto vztahu, tj. jaká má být reakce na smazání instance typu, na kterou se daná reference odkazuje:

- `REFERENCES ARE NOT CHECKED` – referenční integrita není kontrolována
- `REFERENCES ARE CHECKED ON DELETE` *«referenční akce»* – při smazání instance na kterou se reference odkazuje je provedena příslušná akce:
  - `SET DEFAULT` – nastavení reference na implicitní hodnotu
  - `SET NULL` – nastavení reference na hodnotu `NULL`
  - `CASCADE` – záznam tabulky, do kterého reference náleží, je smazán
  - `NO ACTION` – vyvolání varovného hlášení (smazání proběhne)
  - `RESTRICT` – vyvolání varovného hlášení (smazání neproběhne)

Dále je třeba zmínit fakt, že norma SQL:1999 prozatím nepřipouští, aby bylo možné do místa, kam se reference odkazuje ukládat typy odvozené od typu reference. Tato (obvykle podporovaná) vlastnost bude zřejmě začleněna do některé z příštích norem.

2.5.3 Tabulky

Všechna data jsou v databázi uložena ve formě tabulek. Každá tabulka má jednoznačné jméno a je tvořena množinou atributů (sloupců), z nichž každý má datový typ a v rámci tabulky jednoznačné jméno. Z hlediska základní práce s tabulkami obsahuje jazyk SQL:1999 dvě podmnožiny – jazyk pro definici dat a jazyk pro manipulaci s daty.

Jazyk pro definici dat

Jazyk pro definici dat obsahuje příkazy pro vytváření a rušení tabulek a případně modifikaci jejich struktury. Existují dva způsoby vytváření tabulek – „klasicky“ přímým výčtem atributů a jejich typů a tzv. *typové tabulky.*

První způsob byl převzat z předchozích norem jazyka SQL a provádí se příkazem `CREATE TABLE`, který obsahuje seznam atributů a jejich typů.

```
CREATE TABLE filmy (
  název     VARCHAR(40),
  natočen   INT,
  hvězda    ROW ( jméno      VARCHAR(15),
                  příjmení   VARCHAR(35) ),
  ostatní   VARCHAR(50) ARRAY[10],
  délka     INTERVAL HOUR TO MINUTE );
```

Druhý způsob souvisí se strukturovanými datovými typy a provádí se příkazem `CREATE TABLE`, který obsahuje název strukturovaného typu. Atributy vytvořené tabulky odpovídají atributům příslušného strukturovaného typu.

```
CREATE TABLE lidé OF Osoba;
```

Jak již bylo řečeno, typové tabulky souvisí s principem referencí – každá reference je odkazem na jeden záznam typové tabulky. Při definici typové tabulky musí být definován tzv. *samoodkazující sloupec* a opět specifikován typ jednoznačné identifikace jejich záznamů (odpovídající typu identifikace specifikovaném při definici daného strukturovaného typu). Definice se provádí ve tvaru `REF IS` *«název samoodkazujícího sloupce»* *«typ identifikace»*, kde *«typ identifikace»* má jednu z následujících hodnot:

- `SYSTEM GENERATED` – systémově generovaná (jednoznačnou hodnotu ve sloupci určuje systém)
- `USER GENERATED` – uživatelsky definovaná (jednoznačnou hodnotu ve sloupci určuje uživatel)
- `DERIVED` – odvozená (jednoznačnou hodnotu ve sloupci určuje systém na základě hodnot v zadaných sloupcích)

Jazyk pro manipulaci s daty

Jazyk pro manipulaci s daty obsahuje příkazy pro vkládání, odstraňování a modifikaci řádků tabulek a dovoluje formulovat dotazy.

Vkládání do „klasických“ tabulek se provádí příkazem `INSERT INTO` *«název tabulky»* *«seznam sloupců»* `VALUES` *«seznam hodnot»*, přičemž *«seznam sloupců»* není třeba uvádět v případě, kdy uvádíme hodnoty všech sloupců a v pořadí, v němž byly uvedeny v příkazu `CREATE TABLE`.

```
INSERT INTO filmy
  VALUES ('Pupendo', 2003, ROW('Bolek','Polívka'),
          ARRAY['J. Dušek','E. Holubová'], '01:50');
```

Vkládání do typových tabulek (i sloupců se strukturovaným datovým typem) se také provádí příkazem `INSERT INTO` za použití (implicitního) konstruktoru. Substituovatelnost navíc umožňuje vkládání podtypů na pozici nadtypu.

```
INSERT INTO lidé VALUES Osoba(1,'Karel','Skočdopole');
INSERT INTO lidé VALUES Student(2,'Irena','Mlýnková',5,'I');
```

Speciálním případem je vkládání hodnot do atributů s datovým typem reference. Vkládání se provádí pomocí `SELECT` dotazu nad odpovídající typovou tabulkou, kterým je vybrán řádek, na nějž má reference odkazovat.

```
CREATE TABLE s_referencí (
  zaměstnanec REF (Osoba),
  oddělení INT );
INSERT INTO s_referencí VALUES (
  (SELECT REF(li) FROM lidé li WHERE li.id = 1), 150);
```

Dotazování nad tabulkami se provádí příkazem `SELECT` *«seznam sloupců»* `FROM` *«seznam tabulek»* `WHERE` *«podmínky»*. Např.

```
SELECT název FROM filmy WHERE (natočen > 1930);
SELECT * FROM lidé li WHERE li IS OF (Student);
SELECT zaměstnanec->příjmení FROM s_referencí;
```

V povinné části `SELECT` se vyskytuje seznam požadovaných sloupců (znak `'*'` zastupuje všechny sloupce). V povinné části `FROM` se uvádí seznam tabulek, nad nimiž se dotazujeme. V nepovinné části `WHERE` jsou uvedeny podmínky, za kterých mají být řádky tabulek začleněny do výsledku.

2.5.4 Integritní omezení

Integritní omezení jsou pravidla, která musí platit pro hodnoty v daném sloupci tabulky. Existují následující typy integritních omezení:

- `DEFAULT` *«implicitní hodnota»* – určení implicitní hodnoty
- `[NOT] NULL` – hodnoty musí být (ne)nulové
- `UNIQUE` – hodnoty musí být v rámci tabulky unikátní
- `PRIMARY KEY` – hodnoty musí být v rámci tabulky unikátní a nenulové

- `FOREIGN KEY` – odkaz(y) na sloupce v jiné tabulce (`REFERENCES` *«název tabulky» «seznam sloupců»*), které jednoznačně identifikují její záznamy; stejně jako u referencí je i v tomto případě možné specifikovat referenční akce při smazání (`ON DELETE`) nebo změně (`ON UPDATE`) odkazovaného záznamu

- `CHECK` – hodnoty musí splňovat zadanou podmínku; může být dáno:

  - Rozmezí přípustných hodnot – je možné použít operátory porovnání (`<`, `>`, `=`, `<=` a `>=`) s konstantou nebo dotazem, logické operátory (`AND`, `OR`, `NOT`) a predikát `BETWEEN` *«1. hodnota»* `AND` *«2. hodnota»*
  - Seznam přípustných hodnot – predikát `IN [`*«seznam hodnot»*`]`
  - Jednoduchá maska přípustných hodnot – predikát `LIKE` *«vzor»*, kde *«vzor»* může obsahovat zástupné znaky `'_'` (libovolný znak) a `'%'` (libovolné znaky)
  - Komplexní maska přípustných hodnot – predikát `SIMILAR TO` *«regulární výraz»*

- `SCOPE` *«název tabulky»* – omezení specifické pro reference omezující jejich hodnoty na danou typovou tabulku

Pro „klasické“ tabulky existují dva typy integritních omezení:

- Nepojmenované – definujeme za definicí sloupce, k němuž se vztahuje – např. `CREATE TABLE tabulka1 (id INT PRIMARY KEY, ...);`

- Pojmenované – definujeme kdekoli uvnitř definice tabulky – např. `CONSTRAINT nenulovéPříjmení CHECK (příjmení NOT NULL)`

Pro strukturované typy je možné definovat pouze implicitní hodnotu atributu. Ostatní omezení je možné definovat až při jeho použití v tabulce.

Pro typové tabulky se integritní omezení definuje podobně jako pojmenované omezení „klasických“ tabulek ve tvaru *«název sloupce»* `WITH OPTIONS CONSTRAINT` *«název omezení» «definice omezení»*.

2.5.5 Další informace

Základní informace o jazyku SQL:1999 je možné nalézt v [25], podrobný popis pak v [23] a popis objektově relačních rysů v [22].

3 Rozbor problematiky

V této kapitole je uveden přehled a zhodnocení existujících technik pro ukládání XML dokumentů do databází. Zdroje podrobných informací k jednotlivým přístupům jsou uvedeny přímo v textu.

3.1 Typy XML dokumentů

XML dokumenty je možné dělit na dva základní typy – *datově orientované* (data-centric) a *dokumentově orientované* (document-centric). Obecně sice platí, že toto dělení není pro každý dokument jednoznačně definovatelné, přesto může být dobrou pomůckou pro určení vhodného přístupu.

3.1.1 Datově orientované dokumenty

Datově orientované XML dokumenty mají pravidelnou strukturu, poměrně hodně členitá data a obvykle neobsahují elementy se smíšeným obsahem. Pro tento typ dokumentů typicky není podstatné pořadí sousedních elementů a jsou určeny především pro zpracování aplikacemi.

```
<Let>
  <Společnost>ČSA</Společnost>
  <Odkud>Praha</Odkud>
  <Kam>New York</Kam>
  <Čas><Odlet>06:15</Odlet><Přílet>10:26</Přílet></Čas>
  <Čas><Odlet>08:15</Odlet><Přílet>12:26</Přílet></Čas>
  <Čas><Odlet>12:15</Odlet><Přílet>16:26</Přílet></Čas>
  <Čas><Odlet>15:15</Odlet><Přílet>19:26</Přílet></Čas>
</Let>
```

3.1.2 Dokumentově orientované dokumenty

Dokumentově orientované XML dokumenty mají nepravidelnou strukturu, menší členitost dat a často obsahují mnoho elementů se smíšeným obsahem. Jsou určeny pro čtení uživatelem a tudíž je pro ně obvykle podstatné pořadí sousedních elementů.

```
<Produkt>
  <Název>Turecký klíč</Název>
  <Souhrn>Podobný francouzskému klíči, ne tak velký.</Souhrn>
  <Popis>
    <Para>Turecký klíč existuje <i>ve verzi pro leváky i
    praváky</i>. Je vyroben z <b>kvalitní nerezavějící
    oceli</b>. Jeho pogumovaná rukojeť se rychle přizpůsobí
    ruce v každé situaci.</Para>
  </Popis>
</Produkt>
```

3.2 Propojení XML a databází

Technologie XML má velké množství databázových prvků – ukládání dat (XML dokumentů), schéma (DTD, XML Schema…), dotazovací jazyky (XQL, XPath…) a programová rozhraní (DOM, SAX[10]…). Na rozdíl od databází má však několik nevýhod – ukládání není efektivní, neobsahuje databázové mechanismy jako jsou indexy, bezpečnost, transakce, víceuživatelský přístup, triggery, dotazy nad více dokumenty apod. Tyto nevýhody je však možné propojením technologie XML s databázemi odstranit.

Jak je uvedeno v [7], existuje několik technik pro realizaci tohoto propojení, které dělíme podle typu XML dokumentů. Vzhledem k tomu, že je určení typu XML dokumentu mnohdy nejednoznačné, je toto dělení spíše doporučením než pravidlem.

3.2.1 Techniky pro dokumentově orientované dokumenty

Jak již bylo řečeno, je pro dokumentově orientované XML dokumenty podstatné pořadí sousedních elementů, což je třeba při ukládání zohlednit.

LOB

Nejjednodušší technikou propojení technologie XML a databází je uložení XML dokumentu do jednoho sloupce tabulky s datovým typem `BLOB` nebo `CLOB`.

Toto uložení umožňuje manipulovat s XML dokumentem jako s jedním objektem se všemi možnostmi, které pro takový objekt databáze nabízejí. Navíc je zaručeno, že data, která do databáze uložíme, získáme v nezměněné podobě zpět.

Zásadní nevýhodou je ztráta možnosti dotazovat se nad daty, která jsou v XML dokumentu uložena. Některé databázové systémy sice obsahují mechanismy pro indexování LOB sloupců, vyhledávání v úplném textu nebo fuzzy vyhledávání, ovšem ani tyto nástroje základní nevýhodu příliš neřeší.

Nativní XML databáze

Nativní XML databáze je přímo přizpůsobena pro práci s XML dokumenty. Poskytuje databázové mechanismy jako jsou transakce, bezpečnost, multiuživatelský přístup, dotazovací jazyky apod. Rozdílem oproti jiným databázím je fakt, že je její vnitřní model založen na technologii XML.

Existují dva typy nativních XML databází:

- Textově založené – Ukládají XML dokumenty jako text, LOB nebo v jiném (textovém) formátu. Pro dotazování nad XML dokumenty vytvářejí indexy, které zohledňují XML formát dat.

[10]SAX (Simple API for XML) – rozhraní pro práci s XML dokumenty založené na událostech vyvolávaných postupným načítáním elementů XML dokumentu (viz. [21])

- Modelové založené – Vytvářejí vlastní objektový model XML dokumentu, který ukládají do (relační nebo objektově relační) databáze.

Nativní databáze umožňují pracovat s XML dokumenty „přirozenou“ cestou – adresování částí XML dokumentů, dotazování nad XML daty apod. Do databáze se navíc ukládají všechny informace z XML dokumentů, což zaručuje jejich opětovné získání.

Nevýhodou nativních XML databází je, že vzhledem k uspořádání a indexování dat poskytují pouze jeden pohled na XML data. To může vést k časově náročným dotazům v případě, kdy zvolené indexy nezohledňují požadavky dané aplikace.

Nativní databáze bývají často využívány i pro ukládání datově orientovaných XML dokumentů.

PDOM

Technologie PDOM (Persistent Document Object Model) je speciálním typem nativní XML databáze, který je založen na myšlence persistentních DOM stromů XML dokumentů. Jako většina nativních databází i databáze založená technologií PDOM umožňuje vrátit DOM strom XML dokumentu. Persistence v tomto případě znamená, že se jakékoli změny v DOM stromu okamžitě projeví také v databázi.

Technologie PDOM tedy zajišťuje persistentní ukládání dat aplikací pracujících s DOM stromy a současně poskytuje obdobu virtuální paměti. Druhá vlastnost je výhodná především pro aplikace, které pracují s velkými dokumenty, jelikož velikost DOM stromu může značně přesáhnout velikost příslušného XML dokumentu.

Systém pro správu obsahu

Systém pro správu obsahu (content management system) je také speciálním typem nativní XML databáze. Je určen pro správu ručně psaných dokumentů. Zajišťuje řízení vícenásobného přístupu a správu verzí dokumentů, umožňuje vyhledávání, obsahuje XML editory apod.

Výhodou tohoto systému (oproti systému pro správu dokumentů) je, že umožňuje rozdělit dokumenty na jednotlivé části (příklady, kapitoly…) a metadata (jména autorů, čísla verzí…). S těmi je pak možné pracovat odděleně, namísto časově náročné práce s celým dokumentem.

3.2.2 Techniky pro datově orientované dokumenty

Techniky propojení datově orientovaných XML dokumentů a databází mají společnou myšlenku: XML data jsou uložena a zpracovávána v relační nebo objektově relační databázi a prostřednictvím určité metody probíhá jejich převod mezi relacemi a XML dokumenty. Tyto metody jsou popsány v kapitole 3.3).

Jak již bylo řečeno, u datově orientovaných XML dokumentů obvykle nezáleží na pořadí sousedních elementů, čehož je ve většině technik využíváno. Po uložení a následné rekonstrukci tedy obvykle neobdržíme tentýž dokument – není dodrženo pořadí sousedních elementů a mohou být vypuštěny pro aplikaci nepodstatné informace (komentáře, anotace apod.).

Middleware

Middleware je software, který je využíván datově orientovanými aplikacemi pro převod XML dat mezi XML dokumenty a (relační nebo objektově relační) databází.

Databáze s XML rozšířením

Databáze s XML rozšířením je speciální typ databáze obsahující funkce a nástroje pro převod XML dat mezi XML dokumenty a jejími vnitřními strukturami. Většina databází s XML rozšířením typicky ukládá celé XML dokumenty do jednoho sloupce tabulky a pro práci s nimi využívá speciální funkce pro práci s texty.

XML Data Binding

Technologie *XML Data Binding* je založena na myšlence mapování XML dat na třídy nějakého objektově orientovaného jazyka, které odpovídají jejich struktuře. Tato technika umožňuje pracovat s XML dokumenty způsobem, který je pro aplikace přirozenější než práce s DOM stromem.

XML server

XML server je speciální typ serveru, který je přizpůsoben pro práci s XML daty. Může být určen pro vytváření distribuovaných (e-commerce nebo business-to-business) aplikací, publikování XML dokumentů v síti Internet apod. XML servery obvykle pracují s datově i dokumentově orientovanými XML dokumenty.

Nástroje pro dotazování nad XML dokumenty

Nástroje pro dotazování nad XML dokumenty jsou samostatné aplikace, které slouží speciálně pro dotazování nad (datově i dokumentově orientovanými) XML dokumenty.

3.2.3 Další informace

Základní informace o technikách propojení XML a databází je možné nalézt v [29], podrobný popis pak v [7]. V [8] je navíc uveden (v rámci možností) aktuální přehled existujících produktů.

3.3 Metody převodu dat mezi XML a databázemi

V této kapitole je uveden přehled existujících metod převodu dat mezi XML dokumenty a databázemi (tzv. *mapování*). Ve většině případů se tyto metody týkají datově orientovaných XML dokumentů.

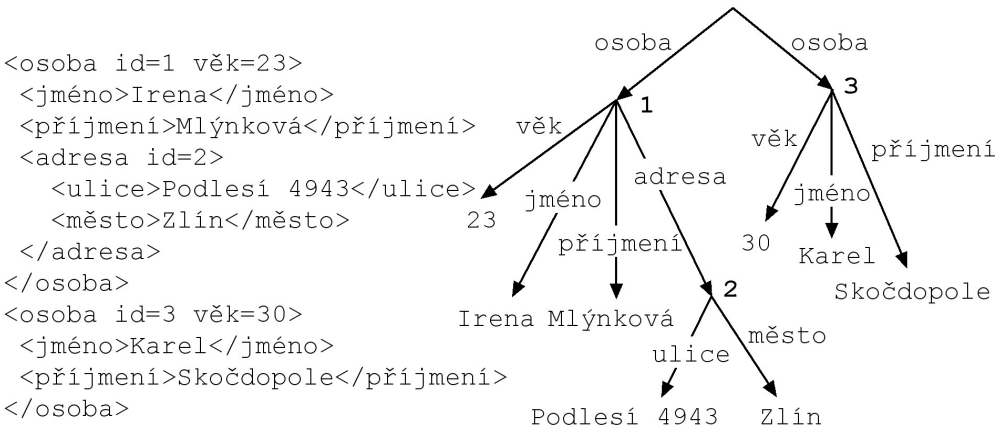
Jak je uvedeno v [2], existuje v současné době několik metod mapování, které dělíme na tři skupiny:

- Generické – metody, které pro mapování nevyužívají schéma XML dokumentu
- Schématem řízené – metody, které jsou založeny na existujícím schématu XML dokumentu
- Uživatelsky definované – metody založené na uživatelsky definovaném mapování

3.3.1 Generické mapování

Metody generického mapování nevyužívají existující schéma XML dokumentu. Tyto metody pohlíží na XML dokument jako na určitý typ orientovaného stromu a definují obecné cílové schéma, do něhož je možné uložit libovolný XML dokument.

Strom XML dokumentu má pevně definovanou strukturu. Vnitřním uzlům stromu jsou přiřazena jednoznačná ID, listům jsou přiřazeny hodnoty atributů nebo elementů. Hrany vedoucí mezi vnitřními uzly stromu jsou označeny názvy elementů, hrany vedoucí do listů názvy atributů nebo elementů, které neobsahují další podelementy. Hrany vycházející z daného uzlu reprezentují podelementy nebo atributy elementu určeného hranou do uzlu vcházející. Příklad XML dokumentu a jeho generického stromu je uveden na obrázku 1.



Obrázek 1: Příklad stromu pro generické mapování

Existuje několik základních metod generického mapování, které se dělí podle způsobu uložení výše uvedeného stromu XML dokumentu – hranové, atributové[11], univerzální a normalizované univerzální. Dále existují speciální typy generického mapování , jako je např. tabulkové mapování. Všechny zmíněné techniky jsou nastíněny v následujících podkapitolách.

Hranové mapování

Tato technika ukládá všechny hrany stromu do jedné tabulky s pevně definovanou strukturou:

```
Hrana(zdroj,cíl,název,typ,pořadí)
```

Tabulka obsahuje ID uzlů, mezi nimiž hrana vede (`zdroj` a `cíl`), název hrany (`název`), příznak zda se jedná o vnitřní hranu nebo o hranu vedoucí do listu (`typ`) a pořadí hrany v rámci sousedních hran (`pořadí`).

Atributové mapování

V této technice je pro každý název hrany (tzv. *atribut*) vytvořena samostatná tabulka, do níž jsou ukládány všechny hrany s daným názvem. Až na chybějící sloupec s názvem hrany mají tyto tabulky stejnou strukturu jako tabulka pro hranové mapování.

Univerzální mapování

Tato technika ukládá hrany stromu do tzv. *univerzální tabulky*, která odpovídá vnějšímu spojení všech tabulek vytvořených při atributovém mapování. Výsledná tabulka má velké množství sloupců, mnoho prázdných hodnot a je značně redundantní.

Normalizované univerzální mapování

Tato technika se snaží řešit nedostatky univerzálního mapování. Pro každý název hrany je vytvořena tzv. *tabulka přetečení*, jejíž struktura odpovídá tabulkám z atributového mapování. V univerzální tabulce se pak pro každý název hrany nachází právě jeden řádek. Ostatní řádky jsou uloženy do tabulek přetečení.

Existují i různé varianty těchto metod. Ve všech popsaných přístupech mohou být hodnoty v listech stromu ukládány do samostatné tabulky nebo do dalšího sloupce tabulek určených pro hrany. Další (tzv. *hybridní*) metody mohou vzniknout kombinací popsaných přístupů – např. vytvořením jedné hranové tabulky přetečení, ukládáním často se vyskytujících hran atributově a ostatních hranově apod. Podrobná analýza těchto metod je uvedena v [17].

[11] Názvy hran jsou v algoritmu označovány jako „atributy“. Toto označení nesouvisí s pojmem atributu v XML.

Tabulkové mapování

Speciální metodou generického mapování popsanou např. v [7] je tabulkové mapování. Základem této metody je předpoklad, že struktura XML dokumentu odpovídá seznamu tabulek – např.

```
<databáze>
  <tabulka1>
    <řádek1>
      <sloupec1>...</sloupec1>
      <sloupec2>...</sloupec2>
      ...
    </řádek1>
    ...
  </tabulka1>
  ...
</databáze>
```

Mapování je tedy v tomto případě přesně definováno strukturou XML dokumentu. Tato technika je vhodná zejména pro přenos dat mezi dvěma relačními databázemi.

3.3.2 Schématem řízené mapování

Metody schématem řízeného mapování jsou založeny na existujícím schématu XML dokumentu (popsaném v jazycích jako je DTD, XML Schema apod.), které je mapováno na schéma relační. Do takto vytvořeného relačního schématu jsou následně uložena data z XML dokumentů validních vůči původnímu XML schématu.

Cílem těchto metod je vytvoření optimálního relačního schématu, které obsahuje přiměřené množství tabulek a jehož struktura co nejvíce odpovídá struktuře původního XML schématu. Všechny tyto metody se snaží o vylepšení základní myšlenky „vytvořit pro každý element jednu relaci složenou z jeho atributů a vztahy element-podelement mapovat pomocí klíčů a cizích klíčů“. Hlavní nevýhodou této myšlenky je totiž nadměrné množství relací.

Jak je uvedeno v [2], mají všechny tyto techniky několik společných principů založených na informacích uvedených v XML schématu (např. vztazích element-podelement, minimálním a maximálním přípustném počtu výskytů elementů apod.). Nejzajímavější z nich jsou tyto:

- Podelementy, které se mohou vyskytovat maximálně jednou, jsou místo do samostatných tabulek mapovány do tabulek nadelementů (tzv. *inlining*). Tímto způsobem dojde k redukci počtu tabulek a tudíž k odstranění jejich časově náročného spojování při dotazování.

- Elementy, které mají nepovinný výskyt (tj. jejich minimální počet výskytů je nulový), jsou mapovány do sloupců s prázdnou (`NULL`) hodnotou.

- Podelementy, které mají vícenásobný výskyt, jsou mapovány na samostatné tabulky. Vztah element-podelement je pak mapován pomocí klíčů, cizích klíčů a referenční integrity.
- Výskyt alternativních podelementů je mapován na samostatné tabulky (podobně jako v předchozím bodě) nebo do jedné univerzální tabulky obsahující sloupce s prázdnými hodnotami.
- Pokud je nutné zachovat pořadí sousedních elementů, je tato informace mapována na speciální sloupec obsahující potřebné informace (např. pořadová čísla).
- Elementy se smíšeným obsahem (tj. elementy, které mohou kromě určených podelementů obsahovat libovolný další text) nejsou obvykle podporovány. Tyto elementy jsou využívány spíše v dokumentově orientovaných XML dokumentech a jejich mapování by vyžadovalo vytvoření velkého množství sloupců s prázdnými hodnotami.
- Přes předchozí optimalizace vyžaduje obvykle rekonstrukce elementu spojení více tabulek.

Existuje několik metod schématem řízeného mapování, které je možné dělit podle jejich přístupu na *fixní* a *flexibilní*. Metody fixního mapování mapují schéma XML dokumentu bez ohledu na jeho obsah nebo budoucí využití. Metody flexibilního mapování zohledňují při mapování různé statistiky výskytů elementů apod.

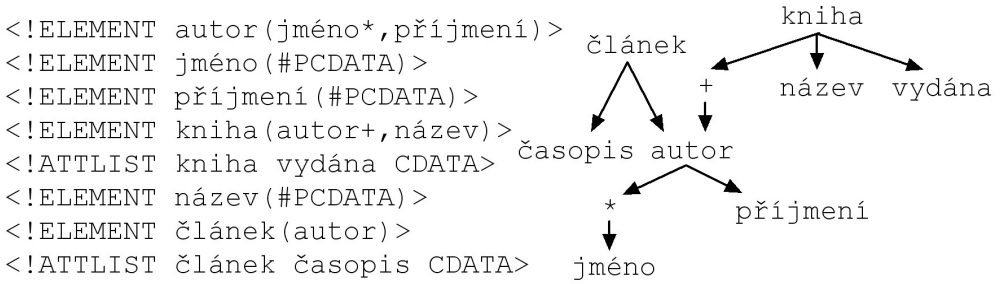
Jelikož do kategorie schématem řízeného mapování spadá mnoho algoritmů, budou na tomto místě zmíněny pouze nejznámější z nich. V kategorii fixního mapování to bude trojice algoritmů *Basic*, *Shared* a *Hybrid*, na ně navazující trojice algoritmů *CPI*, *NeT* a *CoT* a technika objektově relačního mapování. V kategorii flexibilního mapování bude popsán algoritmus navržený v systému *LegoDB*.

Basic, Shared a Hybrid

Nejznámějším představitelem fixního schématem řízeného mapování je trojice algoritmů popsaná např. v [28] – algoritmy Basic, Shared a Hybrid. Tyto algoritmy se zabývají mapováním schématu XML dokumentu popsaném v jazyce DTD na relační schéma.

Jejich základem je vytvoření tzv. *DTD grafu*, který reprezentuje strukturu DTD schématu. Uzly grafu tvoří elementy (vyskytující se v grafu právě jednou), atributy a operátory (vyskytující se tolikrát, kolikrát jsou v DTD uvedeny). Hrany grafu určují vztahy element-atribut, element-podelement nebo element-operátor a operátor-podelement. Příklad schématu popsaném v jazyce DTD a jeho DTD graf je uveden na obrázku 2. (Element `!ELEMENT`

slouží pro definici elementů a jejich podelementů nebo datových typů. Element `!ATTRIBUTE` slouží pro definici atributů a jejich datových typů. Kleeneho operátory `+` a `*` určují přípustný počet výskytů podelementu v rámci nadelementu.)



Obrázek 2: Příklad DTD grafu

Snahou těchto algoritmů je postupné vylepšování základní myšlenky „vytvořit pro každý element jednu relaci“. Všechny tři algoritmy se snaží tento přístup vylepšit metodou *inlining*, tj. různými typy vkládání (neopakujících se podelementů) do tabulek nadelementů.

Tyto algoritmy byly dále rozpracovány také v diplomové práci [18].

CPI, NeT a CoT

Dalším představitelem fixního mapování je trojice algoritmů CPI, NeT a CoT popsaných např. v [20]. Algoritmus CPI se také zabývá mapováním schématu popsaném v jazyce DTD na relační schéma, přičemž se jedná o další vylepšení již zmíněného algoritmu Hybrid. Naproti tomu jsou algoritmy NeT a CoT určeny pro mapování relačního schématu na XML dokument.

Hlavním cílem algoritmu CPI je zachovat při mapování sémantická omezení, která je možné v DTD nalézt. Algoritmus se zaměřuje na čtyři typy omezení – vztahy element-podelement, určení kardinality, referenční integritu a omezení na unikátnost.

Cílem algoritmů NeT a CoT je vylepšení základní myšlenky mapování relačního schématu na XML dokument „mapovat každou tabulku na jeden element a její sloupce na atributy tohoto elementu“. Snahou těchto algoritmů je nalezení vhodného převodu plochých relací na relace hnízděné, z nichž je již možné vytvořit hierarchický a tedy uživateli bližší XML dokument.

Objektově relační mapování

Objektově relační mapování také spadá do kategorie fixních mapování. Tato metoda probíhá ve dvou krocích:

V prvním kroku algoritmu je schéma XML dokumentu mapováno na strom, jehož uzly jsou tvořeny třídami a hrany reprezentují vztahy mezi nimi. Elementy s elementovým nebo smíšeným obsahem jsou obvykle mapovány na

třídy, jejich atributy a elementy s textovým obsahem na vlastnosti těchto tříd. Hrany stromu pak určují vztahy element-podelement.

Ve druhém kroku je takto vytvořený strom tříd uložen do relační (popř. objektově relační) databáze. Třídy jsou mapovány na tabulky a jejich vlastnosti na sloupce. Vztahy element-podelement jsou uloženy jako dvojice klíč a cizí klíč.

Metoda objektově relačního mapování DTD na relační schéma je podrobně popsána v [9] a [5]. Nástin metody objektově relačního mapování XML schématu popsaném v jazyce XML Schema na relační schéma je uveden v [6].

LegoDB mapování

Příkladem flexibilního schématem řízeného mapování je technika použitá v systému *LegoDB* popsaném např. v [4]. Tento přístup využívá pro mapování schéma XML dokumentu popsané v jazyce *XML Query Algebra*[12].

Cílem tohoto algoritmu je vytvořit optimální relační schéma XML dokumentu vzhledem k dotazům, které na něj budou kladeny. Vstupem algoritmu je zpracovávané schéma a statistiky týkající se obvyklých dotazů nad schématem. Tyto statistiky obsahují pro každý datový typ informace o jeho velikosti, největší a nejmenší použité hodnotě v dotazech, počtu použitých různých hodnot, počtu výskytů apod. Na jejich základě je pak schéma opakovaně ohodnocováno a transformováno dokud není dosaženo optima.

3.3.3 Uživatelsky definované mapování

Uživatelsky definované mapování je nejčastěji využíváno v komerčních systémech. Tento přístup vyžaduje, aby uživatel nejprve definoval relační nebo objektově relační schéma a požadované mapování pak určil pomocí určitého dotazovacího jazyka, deklarativního rozhraní nebo prostřednictvím anotací doplněných do ukládaného dokumentu.

Jak je uvedeno v [15], tento přístup využívají např. systémy *Oracle9i*, *IBM DB2* nebo *Microsoft SQL Server 2000*.

3.4 Zvolené řešení

Jak bylo nastíněno v úvodu, je cílem této práce návrh a implementace algoritmů ukládání XML dat do relační databáze založených na využití jazyka XML Schema. Jedná se tedy o návrh algoritmů (fixního) schématem řízeného mapování. Budou navrženy celkem tři algoritmy:

1. Algoritmus vytvoření objektově relačního schématu na základě schématu XML dat popsaném v jazyce XML Schema

[12] Jiný přepis jazyka XML Schema

2. Algoritmus ukládání XML dokumentů validních vůči danému XML schématu do takto vytvořeného objektově relačního schématu

3. Algoritmus mapování podmnožiny jazyka XQL na posloupnost `SELECT` příkazů nad vytvořeným schématem

Jednotlivé cíle a podmínky kladené na algoritmy je možné shrnout do několika bodů:

- Využití co největšího množství prvků, které se mohou v XML schématu vyskytnout. Důraz by měl být kladen zejména na využití XML datových typů, přenesení integritních omezení XML schématu a zajištění referenční integrity dat.

- Využití možností, které poskytuje norma SQL:1999. Cílem je zejména nalezení co nejvhodnějšího mapování objektově orientovaných rysů jazyka XML Schema na objektově relační rysy této normy.

- Množství vytvořených relací (tabulek) by nemělo být příliš velké, ani příliš malé. Struktura, skladba a jejich vzájemné vztahy by měly co nejvíce odpovídat původnímu XML schématu.

- Do vytvořeného objektově relačního schématu by mělo být možné uložit více validních XML dokumentů.

- Předpokládá se ukládání převážně datově orientovaných XML dokumentů a využití jejich obvyklých vlastností.

Zaměření práce na objektově relační vlastnosti normy SQL:1999 plyne zejména z poměrně velkého množství objektově orientovaných rysů jazyka XML Schema. Myšlenka vychází z častého využití jazyka XML Schema v technologiích XML Data Binding, kde je XML schéma popsané v jazyce XML Schema mapováno na třídy nějakého objektově orientovaného jazyka (nejčastěji jazyka Java). Samotné propojení objektově orientovaných rysů normy SQL:1999 s jazykem XML Schema je již (v kombinaci s uživatelsky definovaným mapováním) částečně využito v systému Oracle9i Release 2. Ovšem implicitně vytvářené objektově relační schéma je poměrně jednoduché a jakékoli složitější struktury mohou být vytvořeny pouze prostřednictvím uživatelsky definovaného mapování.

Z nalezených prací se mapováním schématu XML dat popsaném v jazyku XML Schema zabývají články [31] a [6] a diplomová práce [24]. Ve všech případech se jedná o fixní schématem řízené mapování na relační databáze. V článku [6] je sice popsané mapování označováno jako objektově relační, ovšem důvodem tohoto pojmenování je přechod přes pomocný strom tříd (objektů) a nikoli typ cílového schématu.

Podrobný popis návrhu algoritmů a jejich implementace je uveden v následujících dvou kapitolách.

4 Popis řešení

V této kapitole jsou podrobně popsány navržené algoritmy včetně diskuse nad možnými řešeními, zdůvodnění příslušné volby a její případná omezení.

4.1 Algoritmus vytvoření relačního schématu

Tato kapitola obsahuje popis algoritmu mapování XML schématu popsaném v jazyce XML Schema na objektově relační schéma jazyka SQL:1999. V případě, kdy existuje více možností mapování, jsou tyto možnosti popsány, zhodnoceny a je uvedeno, která z variant byla zvolena pro implementaci algoritmu. Popis dále obsahuje informace o tom, které prvky jazyka XML Schema nebudou implementovány a proč.

Pro přehlednost a srozumitelnost je nejprve popsáno mapování jednotlivých prvků XML schématu od jednoduchých ke složitějším. Nejprve je tedy popsáno mapování jednoduchých datových typů, odvozených jednoduchých datových typů, složených datových typů atd. V poslední kapitole je pak popsán samotný algoritmus vytvoření objektově relačního schématu vycházející z popsaného mapování jednotlivých prvků.

4.1.1 Mapování vestavěných jednoduchých typů

Mapování vestavěných jednoduchých typů je určeno tak, aby vlastnosti SQL datových typů co nejvíce odpovídaly vlastnostem XML datových typů.

Typ `string` a typy z něj odvozené

Datový typ `string` je mapován na SQL typ `VARCHAR(M)`, kde `M` je vhodně zvolený parametr. Pokud by byly řetězce v XML dokumentech příliš dlouhé, bylo by možné využít datový typ `CLOB`.

Mapování datových typů odvozených od typu `string` je uvedeno v tabulce 5. Datové typy nebo integritní omezení, která by přímo odpovídala těmto XML datovým typům v normě SQL:1999 nejsou. Jejich specifické vlastnosti by bylo možné kontrolovat např. pomocí triggerů při vkládání do tabulky.

Typy `ID`, `IDREF` a `IDREFS`

Jak již bylo řečeno, mají datové typy `ID`, `IDREF` a `IDREFS` popř. typy z nich odvozené specifické vlastnosti pro zajištění jednoznačnosti elementů a jejich vzájemného odkazování. Pro jejich mapování se tudíž nabízí primární a cizí klíče jazyka SQL:1999, ovšem jejich využití je nedostačující. Zatímco hodnoty typů `ID` musí být jednoznačné v rámci celého XML dokumentu a hodnoty typů `IDREF` a `IDREFS` se mohou odkazovat na libovolnou z nich, jsou primární a cizí klíče vázány vždy právě k jedné tabulce. Jednoznačnost XML datových

Datový typ:	Mapování:
`normalizedString`	`VARCHAR(M)`
`token`	`VARCHAR(M)`
`language`	`VARCHAR(M)`
`NMTOKEN`	`VARCHAR(M)`
`NMTOKENS`	`VARCHAR(M) ARRAY[N]`, kde `N` je vhodně zvolený parametr
`Name`	`VARCHAR(M)`
`NCName`	`VARCHAR(M)`
`ENTITY`	`VARCHAR(M)`
`ENTITIES`	`VARCHAR(M) ARRAY[N]`

Tabulka 5: Mapování datových typů odvozených od typu `string`

typů je tedy obecnější a její mapování na jednoznačnost poskytovanou jazykem SQL:1999 tudíž nemožné.

Mapování těchto typů na SQL datové typy je tedy obdobné jako u ostatních typů odvozených od typu `string` a je uvedeno v tabulce 6.

Datový typ:	Mapování:
`ID`	`VARCHAR(M)`
`IDREF`	`VARCHAR(M)`
`IDREFS`	`VARCHAR(M) ARRAY[N]`

Tabulka 6: Mapování datových typů `ID`, `IDREF` a `IDREFS`

Typ `boolean`

Datový typ `boolean` je mapován na SQL typ `BOOLEAN`.

Typ `decimal` a typy z něj odvozené

Datový typ `decimal` je možné mapovat na SQL typy `DECIMAL(M,N)` nebo `NUMERIC(M,N)`, kde `M` a `N` jsou vhodně zvolené parametry. Volba závisí na jejich implementaci a maximální použité hodnotě v XML dokumentech.

Všechny datové typy odvozené od typu `decimal` jsou mapovány na typ `DECIMAL(M,0)`, respektive `NUMERIC(M,0)` s integritním omezením odpovídajícím příslušnému typu (viz. tabulka 7).

Typ `float`

Datový typ `float` je možné mapovat na SQL typy `REAL` nebo `FLOAT(32)`. Volba závisí na jejich implementaci.

Datový typ:	Integritní omezení:
`integer`	žádné
`positiveInteger`	`CHECK (Col > 0)`, kde `Col` je název sloupce, na nějž je datový typ mapován
`negativeInteger`	`CHECK (Col < 0)`
`nonPositiveInteger`	`CHECK (Col <= 0)`
`nonNegativeInteger`	`CHECK (Col >= 0)`
`long`	`CHECK (Col >= -9223372036854775808 AND Col < 9223372036854775808)`
`int`	`CHECK (Col >= -2147483648 AND Col < 2147483648)`
`short`	`CHECK (Col >= -32768 AND Col < 32768)`
`byte`	`CHECK (Col >= -128 AND Col < 128)`
`unsignedLong`	`CHECK (Col >= 0 AND Col < 18446744073709551616)`
`unsignedInt`	`CHECK (Col >= 0 AND Col < 4294967296)`
`unsignedShort`	`CHECK (Col >= 0 AND Col < 65536)`
`unsignedByte`	`CHECK (Col >= 0 AND Col < 256)`

Tabulka 7: Mapování datových typů odvozených od typu `decimal`

Typ `double`

Datový typ `double` je možné mapovat na SQL typy `FLOAT(64)` nebo `DOUBLE`. Volba závisí na jejich implementaci.

Typ `duration`

Datový typ `duration` je mapován na dvojici SQL typů `INTERVAL YEAR TO MONTH` a `INTERVAL DAY TO SECOND(M)`, kde `M` je vhodně zvolený parametr. Nevýhodou tohoto řešení je, že zatímco všechna pole XML typu `duration` mají neomezenou velikost, u SQL typů `INTERVAL` mohou mít neomezenou velikost pouze první položky (rok a den). Tento problém by bylo možné vyřešit např. vytvořením složeného typu:

```
ROW ( sign BIT, year INT, month  INT, day    INT,
                hour INT, minute INT, second DECIMAL )
```

Pro ukázkovou implementaci algoritmu bude použito první popsané řešení, tj. mapování na typy `INTERVAL`.

Typ `dateTime`

Datový typ `dateTime` je mapován na SQL typ `TIMESTAMP(M) WITH TIME ZONE`, kde `M` je vhodně zvolený parametr. Nevýhodou je, že typ `TIMESTAMP` nepřipouští záporné hodnoty a hodnoty roku větší než 9999. Tento problém by bylo možné vyřešit podobně, jako v případě typu `duration`.

Typ `time`

Datový typ `time` je mapován na SQL typ `TIME(M) WITH TIME ZONE`, kde `M` je vhodně zvolený parametr.

Typ `date` a další typy pro datum

Datový typ `date` je mapován na SQL typ `DATE`. Nevýhodou je, že typ `DATE` neumožňuje specifikovat časovou zónu, záporné hodnoty a roky delší než 9999. Tento problém by bylo možné vyřešit podobně, jako v případě typu `duration`.

Další datové typy specifikující datum (`gYearMonth`, `gYear`, `gMonthDay`, `gMonth` a `gDay`) jsou mapovány na SQL typ `VARCHAR(M)`, kde `M` je vhodně zvolený parametr. Pro tyto případy neexistují specifické SQL typy a datový typ `DATE` není možné využít vzhledem k tomu, že tento typ vyžaduje vložení všech tří polí (tj. den, měsíc i rok) s odpovídající přípustnou hodnotou (např. pro měsíc je to 1 až 12).

Typ `hexBinary`

Datový typ `hexBinary` je mapován na SQL typ `VARCHAR(M)` s integritním omezením `CHECK (Col SIMILAR TO '[0-9a-fA-F]*')`, kde `M` je vhodně zvolený parametr a `Col` název sloupce, na nějž je typ mapován.

Typ `base64Binary`

Datový typ `base64Binary` je mapován na SQL typ `BLOB(M)`, kde `M` je vhodně zvolený parametr. V tomto případě neexistuje SQL typ specifický pro kódování Base64.

Typ `anyURI`

Datový typ `anyURI` je mapován na SQL typ `VARCHAR(M)`, kde `M` je vhodně zvolený parametr. V tomto případě neexistuje SQL datový typ odpovídající normě RFC 2396 specifikující definici URI. Kontrolu přípustnosti by bylo možné provádět např. pomocí triggeru při vkládání do tabulky.

Typ `QName`

Datový typ `QName` je mapován na SQL typ `VARCHAR(M)`, kde `M` je vhodně zvolený parametr. Vzhledem k tomu, že součástí typu `QName` je také datový typ `anyURI`, platí pro kontrolu přípustnosti totéž co pro typ `anyURI`.

Typ `NOTATION`

Datový typ `NOTATION` je mapován na SQL typ `VARCHAR(M)`, kde `M` je vhodně zvolený parametr.

Typ `anyType`

Pokud je dle kontextu datový typ `anyType` považován za jednoduchý datový typ, je mapován na SQL typ `VARCHAR(M)`, kde `M` je vhodně zvolený parametr.

4.1.2 Mapování uživatelsky definovaných jednoduchých typů

Mapování uživatelsky definovaných jednoduchých typů se liší v závislosti na způsobu jejich odvození (restrikcí, seznamem nebo sjednocením). Pro přehlednost jsou přípustné restrikce pro jednotlivé vestavěné typy uvedeny v příloze A.1.

Jednoduché datové typy, ze kterých je odvození prováděno, je možné dělit na jednohodnotové a vícehodnotové. Mezi vícehodnotové vestavěné typy patří `NMTOKENS`, `IDREFS` a `ENTITIES`, jednohodnotové typy jsou všechny ostatní.

Restrikce `length`, `minLength` a `maxLength`

Tyto typy restrikcí jsou přípustné pro datový typ `string` a typy od něj odvozené, typ `hexBinary`, `base64Binary`, `anyURI`, `QName`, `NOTATION` a uživatelsky definované typy odvozené seznamem. Pro jednohodnotové typy znamenají omezení délky, pro vícehodnotové znamenají omezení počtu prvků v seznamu. Jejich mapování je uvedeno v tabulce 8.

Datový typ:	**Restrikce:**	**Integritní omezení:**
jednohodnotový	`length=N`	`CHECK (LENGTH(Col) = N)`, kde `Col` je název sloupce, na nějž je mapován příslušný datový typ
	`minLength=N`	`CHECK (LENGTH(Col) >= N)`
	`maxLength=N`	`VARCHAR(N)`
vícehodnotový	`length=N`	`ARRAY[N]` a integritní omezení `CHECK (CARDINALITY(Col) = N)`
	`minLength=N`	`ARRAY[M]`, kde `M` je vhodně zvolený parametr a integritní omezení `CHECK (CARDINALITY(Col) >= N)`
	`maxLength=N`	`ARRAY[N]`

Tabulka 8: Mapování restrikcí `length`, `minLength` a `maxLength`

Restrikce `pattern`

Tento typ restrikce je přípustný pro všechny jednohodnotové datové typy. Jeho mapování je řešeno prostřednictvím integritního omezení `CHECK (Col SIMILAR TO regExp)`, kde `Col` je název sloupce, na nějž je mapován příslušný datový typ a `regExp` hodnota atributu `pattern` (tj. regulární výraz).

Vzhledem k tomu, že norma jazyka XML Schema podporuje silnější regulární výrazy než norma SQL:1999, je možné hodnotu atributu `pattern`

mapovat pouze v případě, kdy použité operátory vyhovují normě SQL:1999. Jinak by tuto kontrolu bylo možné ošetřit např. pomocí triggeru.

Restrikce `enumeration`

Tento typ restrikce je přípustný pro všechny datové typy s výjimkou typu `boolean`. Jeho mapování je pro jednohodnotové typy řešeno prostřednictvím integritního omezení `CHECK (Col IN [`$\mathtt{E}_1$`, ... `$\mathtt{E}_k$`])`, kde `Col` je název sloupce, na nějž je mapován příslušný datový typ a $\mathtt{E}_1$, ... $\mathtt{E}_k$ množina jeho povolených hodnot. Pro vícehodnotové datové typy v SQL vhodné integritní omezení neexistuje.

Restrikce `whiteSpace`

Tento typ restrikce je přípustný pro všechny datové typy. Pro vícehodnotové typy mají bílé znaky význam oddělovačů a tudíž nemá její mapování význam. Pro jednohodnotové datové typy v SQL:1999 vhodné integritní omezení neexistuje.

Restrikce `maxInclusive`, `maxExclusive`, `minInclusive` a `minExclusive`

Tyto typy restrikcí jsou přípustné pro číselné datové typy a typy pro datum a čas. Jejich mapování je uvedeno v tabulce 9.

Restrikce:	**Integritní omezení:**
`maxInclusive=N`	`CHECK (Col <= N)`, kde `Col` je název sloupce, na nějž je mapován příslušný datový typ
`maxExclusive=N`	`CHECK (Col < N)`
`minInclusive=N`	`CHECK (Col >= N)`
`minExclusive=N`	`CHECK (Col > N)`

Tabulka 9: Mapování restrikcí `max/minInclusive` a `max/minExclusive`

Restrikce `totalDigits` a `fractionDigits`

Tyto typy restrikcí jsou přípustné pro datový typ `decimal` a typy z něj odvozené. Jejich mapování je uvedeno v tabulce 10.

Restrikce:	**Integritní omezení:**
`totalDigits=N`	`DECIMAL(M,N)`, kde `M` je vhodně zvolený parametr
`fractionDigits=N`	`DECIMAL(N,M)`

Tabulka 10: Mapování restrikcí `totalDigits` a `fractionDigits`

Odvození seznamem

Odvození seznamem je možné provádět pouze z jednohodnotových typů. Odvození seznamem je mapováno na SQL datový typ `ARRAY[M]` příslušných typů, kde `M` je vhodně zvolený parametr.

Mapování restrikcí takto vytvořeného datového typu se provádí stejně, jako mapování restrikcí vestavěných vícehodnotových datových typů. Nevýhodou tohoto řešení je, že vzhledem k vlastnostem jazyka SQL:1999 již není možné mapovat restrikce jednoduchých typů, z nichž je seznam vytvořen.

Odvození sjednocením

Datový typ odvozený sjednocením umožňuje, aby mohl mít příslušný element hodnoty z různých datových typů. Vzhledem ke způsobu zpracování XML dokumentů není možné (jednoduše) zjistit, který z typů je právě načítán. Vzhledem k tomu je nutné mapovat datové typy odvozené sjednocením na SQL typ, do něhož je možné uložit libovolnou hodnotu, tj. `VARCHAR(M)`, kde `M` je vhodně zvolený parametr.

Mapování restrikcí takto vytvořených datových typů se provádí stejně jako mapování restrikcí vestavěných datových typů.

Atributy elementu `simpleType`

Z atributů elementu `simpleType` nemá pro vytváření objektově relačního schématu význam žádný.

4.1.3 Mapování složených datových typů

Složený datový typ určuje množinu atributů a obsah elementu, kterému je přiřazen. Složené datové typy (definované lokálně i globálně) jsou mapovány na uživatelsky definované typy (UDT) jazyka SQL:1999. XML atributy složených typů jsou vždy mapovány na atributy UDT (podrobněji viz. kapitola 4.1.4), mapování jednotlivých typů obsahu složeného datového typu je popsáno v následujících podkapitolách.

Jednoduchý obsah

Jednoduchý obsah složeného typu je mapován (stejně jako jeho atributy) na atribut UDT, který má příslušný jednoduchý datový typ.

Složený obsah – rozšíření

Složený obsah datového typu vytvořený rozšířením principiálně odpovídá dědičnosti, tj. aktuální UDT je potomkem UDT, na který je mapován rozšiřovaný složený typ (jeho název je určen atributem `base`). Výjimkou z tohoto

pravidla je případ, kdy je rozšiřovaným typem typ `anyType`, který není považován za předka, tj. nový UDT je (stejně jako ostatní) odvozen od společného předka.

Mapování obsahu rozšíření (posloupnost, výběr, množina nebo modelová skupina) je popsáno v následujících podkapitolách.

Složený obsah – restrikce

Pro odvození restrikcí není v SQL:1999 žádný prostředek (dědičnost datový typ rozšiřuje, nikoli omezuje). Pro mapování je tedy použit původní datový typ, tj. restrikce je ignorována. Nevýhodou tohoto řešení je, že původní datový typ může obsahovat nadbytečné položky, nicméně je do něj možné uložit i instance typu odvozeného restrikcí.

Mapování obsahu restrikce (posloupnost, výběr, množina, modelová skupina) je popsáno v následujících podkapitolách.

Složený obsah – posloupnost elementů

Posloupnost elementů je mapována na vlastní UDT a jemu odpovídající tzv. *pomocný element*. Každý prvek posloupnosti (tj. element, modelová skupina, výběr z elementů nebo posloupnost elementů) je také mapován na vlastní UDT. Vztah posloupnost-prvek posloupnosti je potom mapován v závislosti na typu prvku (globálně nebo lokálně definovaný) a maximálním počtu jeho výskytů (na hodnotě XML atributu `maxOccurs`):

- Globálně definovaný prvek je uložen do společné typové tabulky odpovídající jeho UDT a do UDT posloupnosti je dle hodnoty atributu `maxOccurs` přidán atribut vyjadřující vztah posloupnost-prvek posloupnosti:
  - Je-li `=1`, je atributem reference do této tabulky.
  - Je-li `>1`, je atributem pole referencí (jehož velikost určuje hodnota XML atributu `maxOccurs`) do této tabulky.
- Lokálně definovaný prvek je mapován dle atributu `maxOccurs`:
  - Je-li `=1`, je mapován na atribut UDT posloupnosti.
  - Je-li `>1`, je pro UDT prvku vytvořena vlastní typová tabulka a do UDT posloupnosti je přidán atribut pole referencí (jehož velikost určuje hodnota XML atributu `maxOccurs`) do této tabulky.

Všechny reference jsou vždy systémově generované a obsahují příslušné integritní omezení `SCOPE`.

Hodnota atributu `minOccurs` je mapována v závislosti na typu mapování odpovídajícího prvku na následující integritní omezení:

- Je-li prvek mapován na pole referencí, pak je integritním omezením `CHECK (CARDINALITY(Col) >= minOccurs)`, kde `Col` je název příslušného atributu.
- Jinak (`minOccurs` je `0` nebo `1`), je integritním omezením `[NOT] NULL`.

Stejně tak, tj. v závislosti na typu a maximálním počtu výskytů elementu `sequence`, je mapován vztah nadelement-pomocný (pod)element odpovídající posloupnosti.

Složený obsah – množina elementů

Množina elementů smí obsahovat pouze elementy, jejichž atribut `maxOccurs` má hodnotu maximálně 1 (stejné omezení platí i pro celou množinu elementů). Dále platí, že se elementy z množiny mohou v XML dokumentu vyskytovat v libovolném pořadí. V tomto případě je tedy nutné uložit i pořadí jednotlivých elementů. Množina elementů je tedy mapována podobným způsobem jako posloupnost elementů, tj. na vlastní UDT (a jemu odpovídající pomocný element). Rozdílem je, že do UDT množiny elementů je pro každý atribut odpovídající podelementu přidán atribut pro uložení jeho pořadí.

Složený obsah – výběr z elementů

Výběr z elementů umožňuje, aby se na určeném místě vyskytoval některý z dané množiny elementů. Tuto vlastnost je možné zajistit prostřednictvím dědičnosti: Všechny vytvářené UDT jsou vždy odvozeny od společného UDT. Výběr z elementů je potom mapován na UDT (a jemu odpovídající pomocný element) obsahující jediný atribut typu reference na tohoto předka (bez integritního omezení `SCOPE`). Pokud je prvkem výběru globálně definovaný prvek, jsou jeho instance ukládány a odpovídající reference se odkazují do společné typové tabulky. Pokud je prvkem výběru lokálně definovaný prvek, je pro něj vytvořena vlastní typová tabulka.

Takto navržené mapování je ovšem možné použít pouze v případě, že mají všechny prvky výběru z elementů jednonásobný výskyt. Tuto vlastnost je ale možné zajistit jednoduchou modifikací XML schématu: Elementy výběru, které mají vícenásobný výskyt, jsou nahrazeny posloupností s jednonásobným výskytem obsahující původní element s vícenásobným výskytem.

Vztah nadelement-pomocný (pod)element odpovídající výběru z elementů je řešen stejně jako v případě posloupnosti elementů, tj. v závislosti na typu a maximálním počtu výskytů elementu `choice`.

Složený obsah – modelová skupina

Modelová skupina slouží k tomu, aby mohla být stejná skupina elementů vkládána do různých složených typů. Její mapování je tedy stejné, jako mapování jejího obsahu (posloupnosti, výběru nebo množiny elementů), tj. na odpovídající UDT. Modelová skupina navíc ovlivňuje minimální a maximální počet výskytů svého obsahu – výsledný minimální (maximální) počet výskytů je součin minimálního (maximálního) počtu výskytů modelové skupiny a minimálního (maximálního) počtu výskytů obsahu.

Modelové skupiny (jakožto prostředku pro globální definici skupiny elementů) je využito pro snížení počtu vytvářených tabulek. Pro UDT (obsahu) modelové skupiny je vytvořena právě jedna typová tabulka a do ní jsou ukládány všechny výskyty modelové skupiny.

Atributy elementu `complexType`

Z atributů elementu `complexType` se do vytvářeného objektově relačního schématu promítnou následující:

- `abstract` – UDT je označen jako `[NOT] INSTANTIABLE`

4.1.4 Mapování atributů a skupin atributů

Atributy mohou být definovány globálně, lokálně nebo uvnitř (lokálně či globálně definované) skupiny atributů. Všechny tyto možnosti slouží pouze pro zjednodušení popisu schématu. Dále platí, že se každý atribut může vyskytovat na příslušném místě maximálně jednou. Vzhledem k těmto faktům jsou XML atributy vždy mapovány na atributy UDT, ke kterým přísluší, bez ohledu na to, jakým způsobem jsou definovány.

Atributy elementu `attribute`

Z atributů elementu `attribute` se do vytvářeného objektově relačního schématu promítnou následující:

- `default` – pouze pro jednohodnotové datové typy je mapován na integritní omezení `DEFAULT` *«implicitní hodnota»*
- `fixed` – mapován dle datového typu:
  - jednohodnotový – integritní omezení `CHECK (Col =` *«konstatní hodnota»*`)`, kde `Col` je název sloupce, na nějž je typ mapován
  - vícehodnotový – libovolný XML typ je mapován na SQL typ `VARCHAR(M)` s integritním omezením `CHECK (Col =` *«konstatní hodnota»*`)`
- `use` – mapován na integritní omezení `[NOT] NULL`

4.1.5 Mapování elementů

Mapování elementu odpovídá mapování jeho datového typu. Každý element je tedy mapován na UDT tvořený množinou atributů odpovídajících XML atributům elementu a množinou atributů odpovídajících typu obsahu elementu (každý z těchto atributů odpovídá jednomu kroku dědění). Jak již bylo uvedeno, jsou instance UDT uloženy v závislosti na typu elementu a hodnotě XML atributu `maxOccurs` buď do sloupce tabulky svého nadelementu nebo do vlastní typové tabulky.

V případě ukládání do typových tabulek je pro snížení počtu vytvářených tabulek využíváno globálně definovaných typů, respektive elementů. Pokud se jedná o globálně definovaný typ elementu/element, je pro něj vytvořena pouze jedna typová tabulka, do níž jsou ukládány všechny jeho výskyty.

Atributy elementu `element`

Z atributů elementu `element` se do vytvářeného objektově relačního schématu promítnou následující:

- `nillable` – mapován na integritní omezení `[NOT] NULL` atributů odpovídajících obsahu elementu
- `default` – mapován stejně jako v případě atributů
- `fixed` – mapován stejně jako v případě atributů
- `minOccurs` – mapován stejně jako v případě složených typů
- `maxOccurs` – mapován stejně jako v případě složených typů

Kořenový element

Kořenový element je speciálním případem elementu – jeho obsah (a tedy i mapování) odpovídá složenému datovému typu obsahujícím výběr z (globálních) elementů. Z atributů elementu `schema` nemá pro vytváření objektově relačního schématu význam žádný.

4.1.6 Mapování ostatních prvků

Mapování ostatních prvků jazyka XML Schema nebude v navrženém algoritmu řešeno. Zdůvodnění je uvedeno v následujících podkapitolách.

Omezení identity

Omezení identity je v podstatě zobecněním identity, kterou poskytují jednoduché typy `ID`, `IDREF` a `IDREFS` a typy z nich odvozené. Jak již bylo řečeno, jsou možnosti těchto tří typů příliš obecné na to, aby mohly být nějakým způsobem mapovány na vlastnosti jazyka SQL:1999. V případě omezení identity

se tedy jedná o stejný problém. Tudíž příslušné prvky nejsou na objektově relační schéma mapovány (tj. jejich výskyt v XML schématu je ignorován).

Zástupci

Existují dva typy zástupců – pro atributy a pro elementy. Oba tyto prvky umožňují, aby se na daném místě v XML dokumentu vyskytoval v podstatě libovolný element respektive atribut. Tuto vlastnost není možné v SQL:1999 vyjádřit přesně. V zásadě by bylo možné tyto případy mapovat na nějaký obecný datový typ (např. `VARCHAR`), ovšem využití takto mapovaného prvku je téměř nulové (např. na podelementy takto mapovaného elementu se nelze dotazovat apod.). Tudíž není jejich mapování do ukázkové implementace začleněno (tj. XML schémata obsahující tyto prvky není možné zpracovat).

Notace

Notace slouží pro popis jiných než XML dat, ovšem pro vytváření XML schématu nemají žádný význam (obvykle se jedná o informace pro nadřazenou aplikaci). Tudíž nejsou na objektově relační schéma mapovány (tj. jejich výskyt v XML schématu je ignorován).

Anotace

Anotace slouží pro popis (dokumentaci) XML schématu a nemají žádný vliv na jeho strukturu. Tudíž nejsou na objektově relační schéma mapovány (tj. jejich výskyt v XML schématu je ignorován).

Externí schémata

Při vytváření XML schématu je možné prostřednictvím elementů `include`, `import` a `redefine` využívat již existující XML schémata:

- Element `include` umožňuje začlenit do aktuálního XML schématu prvky jiného XML schématu. Jeho mapování tedy odpovídá mapování sjednocení prvků obou schémat.
- Element `import` umožňuje využívat při vytváření aktuálního XML schématu prvky jiného XML schématu. Jeho mapování tedy odpovídá mapování prvků importovaného schématu a následně prvků aktuálního schématu.
- Element `redefine` umožňuje začlenit a předefinovat prvky jiného XML schématu přičemž předefinováním je míněna restrikce nebo rozšíření. Jeho mapování tedy odpovídá mapování prvků importovaného schématu, do něhož předefinovaný prvek patří a následně mapování příslušné restrikce nebo rozšíření.

Vzhledem k tomu, že tyto elementy slouží pouze pro zjednodušení definice XML schématu, pro jednoduchost není jejich mapování do ukázkové implementace začleněno (tj. XML schémata obsahující tyto prvky není možné zpracovat).

Substituční skupiny

Princip substitučních skupin umožňuje určit elementy, které se mohou vzájemně nahrazovat. Přestože se na první pohled jedná o velmi jednoduchý nástroj, je současně velmi silný, neboť zcela mění strukturu definovaného XML schématu. Některé typy substitučních skupin by bylo možné mapovat podobně jako výběr z elementů, ovšem obecně je jejich zpracování značně komplikované. Pro jednoduchost tedy není jejich mapování do ukázkové implementace začleněno (tj. XML schémata obsahující substituční skupiny není možné zpracovat).

4.1.7 Pomocné tabulky

Aby bylo možné s objektově relačním schématem navrženým v předchozích kapitolách dále pracovat, je navíc vytvořena následující sada pomocných tabulek udržujících dodatečné informace o mapování.

Tabulka pro správu XML schémat

Jedním z požadavků na algoritmus je, aby bylo možné v jednom databázovém systému vytvořit na základě různých XML schémat různá objektově relační schémata. Tudíž je nutné názvy vytvářených UDT a tabulek pro jednotlivá schémata jednoznačně rozlišit. K tomu je využito URL XML schématu, které by mělo být jednoznačné. Jelikož jeho název může být poměrně dlouhý, je ke každému URL přiřazen jednoznačný identifikátor (`schemaID`) a tento vztah je (spolu s identifikátorem kořenového elementu) uložen do následující tabulky:

```
xmlSchemaTable(uri,schemaID,rootElemID)
```

Jmenná konvence

Jak již bylo řečeno, je do názvů UDT a tabulek začleněna identifikace XML schématu. Názvy prvků objektově relačního schématu jsou tedy generovány následujícím způsobem:

- Názvy UDT – při vytváření databázového schématu je každému UDT přiřazen jednoznačný identifikátor (`udtID`) a v názvu UDT je dále obsažena informace o jeho typu:
  - UDT elementu: `'E_' + schemaID + '_' + udtID`
  - UDT pomocného elementu: `'P_' + schemaID + '_' + udtID`

- Názvy atributů UDT – každý UDT má pro správnou konstrukci instance typu definováno pořadí svých atributů (`order`) a v názvu atributu je dále obsažena informace o typu jeho obsahu:
  - XML atribut elementu: `'A_' + order`
  - Obsah elementu: `'C_' + order`
  - Pořadí elementu: `'O_' + order`
- Názvy (typových) tabulek – při vytváření je využita jednoznačnost pojmenování příslušných UDT a fakt, že je pro každý UDT vytvořena maximálně jedna tabulka: `'T_' +` *«název UDT»*

Tabulka pro správu XML dokumentů

Dalším z požadavků na algoritmus je, aby bylo možné do jednoho objektově relačního schématu uložit více XML dokumentů validních vůči původnímu XML schématu. Podobně jako v předchozím případě je využito URL XML dokumentu, k němuž je přiřazen jednoznačný identifikátor, který současně odpovídá identifikátoru záznamu v typové tabulce kořenového elementu XML schématu. Tento vztah je uložen do následující tabulky:

```
xmlDocTable(schemaID,url,docID)
```

Tabulky pro správu vztahů

Aby bylo možné do vytvořeného schématu ukládat validní XML dokumenty a nad schématem pokládat dotazy, je nutné uložit pomocné informace o vztazích element-podelement a element-atribut, způsobu mapování jednotlivých prvků XML schématu a SQL datových typech, které byly pro mapování použity. Pro uložení těchto informací slouží následující tabulky:

```
xmlAttrTable(schemaID,attrID,xmlName,mapType)
```

Tato tabulka obsahuje informace o tom, jakým způsobem jsou mapovány atributy XML schématu:

- `attrID` – v rámci daného schématu jednoznačný identifikátor atributu
- `xmlName` – název atributu v XML dokumentu
- `mapType` – identifikátor SQL typu atributu (`'n'` = číselný typ, `'s'` = znakový typ, `'N'` = pole číselných typů, `'S'` = pole znakových typů)

```
xmlElemTable(schemaID,elemID,xmlName,mapType,elemType,udtName)
```

Tato tabulka obsahuje informace o tom, jakým způsobem jsou mapovány prvky XML schématu, tj. elementy a pomocné elementy:

- `elemID` – v rámci daného schématu jednoznačný identifikátor prvku
- `xmlName` – název prvku v XML dokumentu (pouze pro XML elementy)
- `mapType` – identifikátor SQL typu obsahu elementu (`'n'` = číselný typ, `'s'` = znakový typ, `'N'` = pole číselných typů, `'S'` = pole znakových typů, `'C'` = složený typ, `'o'` = složený typ bez obsahu)
- `elemType` – typ elementu (`'C'` = pomocný element typu výběr z elementů, `'S'` = pomocný element typu posloupnost elementů, `'A'` = pomocný element typu množina elementů, `'E'` = element)
- `udtName` – název UDT daného prvku

```
xmlElemAttrTable(schemaID,elemID,attrID,ord)
```

Tato tabulka obsahuje informace o M:N vztahu element-atribut (element může mít více atributů a současně může být globální atribut vložen do různých elementů):

- `ord` – pořadové číslo atributu v rámci všech atributů UDT (tj. atributů odpovídajících XML atributům i atributů pro uložení obsahu)

```
xmlElemElemTable(schemaID,elemID,subElemID,ord,
                 mapType,min,max)
```

Tato tabulka obsahuje informace o M:N vztahu (pomocný) element-podelement (element může mít více podelementů a současně může být globální element podelementem různých elementů):

- `ord` – pořadové číslo podelementu v rámci všech atributů UDT
- `mapType` – typ mapování vztahu element-podelement (`'C'` = sloupec nadelementu, `'R'` = reference, `'A'` = pole referencí)
- `min` – minimální počet výskytů podelementu
- `max` – maximální počet výskytů podelementu

```
xmlPathTable(schemaID,elemID,subElemID,path)
```

Tato tabulka obsahuje popis „cesty“ přes pomocné elementy mezi elementem a podelementem:

- `path` – pole identifikátorů pomocných elementů

4.1.8 Datový model

V jazyce XML Schema je prostřednictvím modelových skupin možné definovat tzv. nedeterministický datový model. Nedeterminismus v tomto případě znamená, že při sekvenčním zpracování jednotlivých podelementů není vždy možné jednoznačně určit, podle které z přípustných variant má zpracování pokračovat. Jednoduchým příkladem může být následující část XML schématu:

```
<xs:complextype name="příkladNedeterminismu">
  <xs:choice>
    <xs:sequence>
      <xs:element ref="A"/>
      <xs:element ref="B"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element ref="A"/>
      <xs:element ref="C"/>
    </xs:sequence>
  </xs:choice>
</xs:complextype>
```

Jak je uvedeno např. v [6], pro mapování XML schémat existují v souvislosti s datovým modelem dva přístupy – mapování všech podelementů stejného typu (vzhledem k nadelementu) stejným nebo různým způsobem. Při načítání odpovídajícího XML dokumentu je tudíž v prvním případě vztahem element-podelement vždy jednoznačně určen způsob uložení daného podelementu, zatímco ve druhém případě to obecně platit nemusí. Ve výše uvedeném příkladu by byl tedy v prvním případě element `A` uložen vždy stejným způsobem (tj. bez ohledu na to, zda následuje element `B` nebo `C`), zatímco ve druhém případě by bylo nejprve nutné zjistit, která z variant (tj. `A`, `B` nebo `A`, `C`) byla zvolena.

Metody mapování druhého zmíněného typu tedy neumožňují zpracovávat XML schémata s nedeterministickým datovým modelem. Na druhou stranu tyto metody lépe zohledňují strukturu XML schématu a pořadí elementů v rámci nadelementu. Navíc, jak je uvedeno v [12], kvůli kompatibilitě se SGML jsou nedeterministické datové modely obecně nepřípustné.

Jak je patrno z předchozího návrhu, patří metoda navržená v této práci do druhé výše popsané skupiny metod. Tudíž zpracování XML schémat s nedeterministickým datovým modelem neumožňuje.

4.1.9 Vytvoření objektově relačního schématu

Postup vytvoření objektově relačního schématu vychází z popsaného návrhu mapování jednotlivých prvků XML schématu. Algoritmus dále bere v úvahu požadavek vytvoření více různých schémat a požadavek uložit do každého schématu data z více validních XML dokumentů.

Dále je využíván fakt, že každé XML schéma popsané v jazyce XML Schema je opět XML dokumentem a tudíž je možné zpracovávat jej pomocí programových rozhraní SAX nebo DOM. Pro vytváření objektově relačního schématu byla zvolena druhá možnost neboť lze předpokládat, že XML schéma obvykle nebude příliš velké (tj. vytvoření jeho DOM stromu nebude příliš náročné na systémové prostředky) a současně je tímto způsobem možné procházet procházet XML schéma různými směry.

Základní myšlenkou algoritmu je tedy průchod (modifikovaného) DOM stromu XML schématu a zpracovávání jeho uzlů (tj. elementů definujících XML schéma) ve vhodném pořadí. Jednotlivé uzly jsou zpracovávány v závislosti na svém typu a obsahu. Přehled přípustného obsahu všech prvků XML schématu je uveden v příloze A.2.

Algoritmus je možné rozdělit do několika fází:

Fáze I. – Přípravná fáze

V přípravné fázi probíhá kontrola přípustnosti XML schématu a kontrola existence pomocných tabulek a typů:

1. Vytvoř DOM strom XML schématu (včetně kontroly jeho validity a deterministočnosti jeho datového modelu).

2. Zkontroluj přípustnost XML schématu. XML schéma nesmí obsahovat elementy `any`, `anyAttribute`, `import`, `include` a `redefine`, nesmí obsahovat substituční skupiny a nesmí připouštět smíšený obsah elementů. Ostatní prvky nepodléhající zpracování (tj. elementy `notation`, `annotation`, `unique`, `key` a `keyref`, komentáře, instrukce pro zpracování a sekce `CDATA`) jsou ignorovány.

3. Zkontroluj jmenný prostor XML schématu. XML schéma smí obsahovat pouze jmenný prostor jazyka XML Schema, jinak algoritmus končí.

4. Zkontroluj existenci pomocných tabulek. Pokud neexistují, vytvoř je.

5. Zkontroluj existenci mapování aktuálního XML schématu. Pokud již existuje, algoritmus končí.

6. Vytvoř nový jednoznačný identifikátor pro aktuální schéma.

7. Ulož odpovídající záznam do tabulky `xmlSchemaTable` (`rootElemID` je prozatím nastaveno na `NULL` a doplněno po dokončení zpracování kořenového elementu).

8. Vytvoř abstraktního předka `'xmlAncestor_' + schemaID` všech UDT aktuálního schématu, obsahující dva atributy – identifikátor záznamu (`recordID`) a identifikátor elementu (`elemID`).

Fáze II. – Vytvoření DOM grafu

Jak již bylo řečeno v úvodu, je celý algoritmus založen na průchodu modifikovaným DOM stromem XML schématu. Modifikace DOM stromu spočívá v jeho „doplnění“ na orientovaný graf a následné eliminaci jeho případných kružnic. Takto vytvořený orientovaný graf je pro účely této práce nazýván *DOM graf*.

Definice. *DOM graf* příslušející k DOM stromu $G_S = (V_S, E_S)$ je orientovaný graf $G = (V, E)$, kde
$V = V_S$ a
$E = \{(v_x, v_y) \mid \{v_x, v_y\} \in E_S \wedge (v_y \text{ je podelementem } v_x)\} \cup \{(v_x, v_y) \mid$ prvku v_x je přiřazen globálně definovaný datový typ $v_y\} \cup \{(v_x, v_y) \mid$ součástí prvku v_x je globálně definovaný prvek $v_y\}$.

DOM graf je tedy tvořen uzly původního DOM stromu a obsahuje následující typy hran:

- Zorientované hrany původního DOM stromu
- Hrany vyjadřující použití globálně definovaného prvku, tj. hrany vedoucí z listových do globálních elementů (tzv. *zpětné hrany*):
  - Hrany určující, že je danému atributu nebo elementu přiřazen globálně definovaný jednoduchý nebo složený datový typ
  - Hrany určující, že je součástí daného prvku jiný globálně definovaný prvek (element, modelová skupina, atribut nebo skupina atributů)

Pro následující příklad XML schématu je odpovídající DOM strom uveden na obrázku 3 a odpovídající DOM graf na obrázku 4.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="zamestnanci">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="osoba">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="jmeno"/>
            </xs:sequence>
            <xs:attribute name="volno" type="anoNe"
                          use="required"/>
            <xs:attribute name="poznamka" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
```

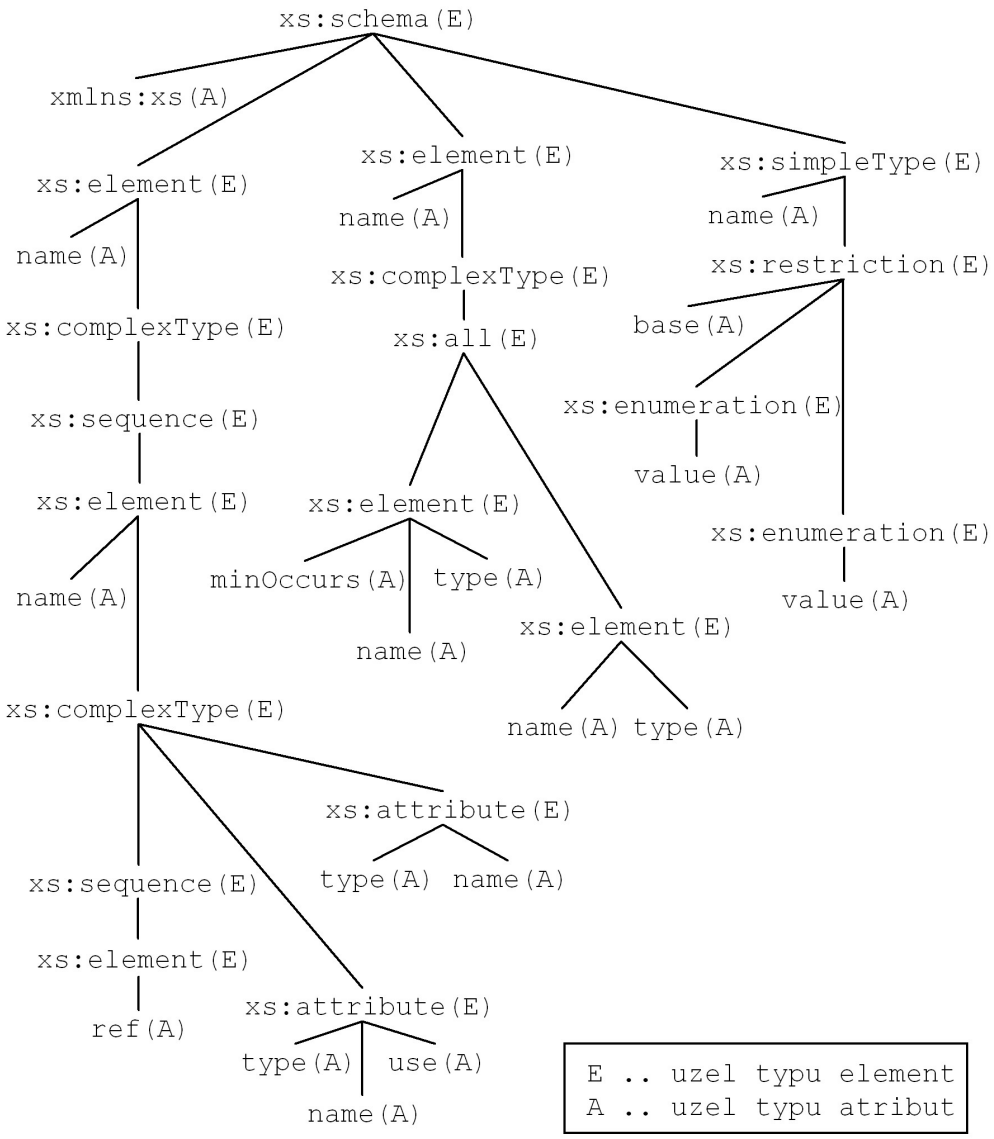
```
    </xs:complexType>
  </xs:element>
  <xs:element name="jmeno">
    <xs:complexType>
      <xs:all>
        <xs:element name="krestni" type="xs:string"
                    minOccurs="0"/>
        <xs:element name="prijmeni" type="xs:string"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="anoNe">
    <xs:restriction base="xs:string">
      <xs:enumeration value="ano"/>
      <xs:enumeration value="ne"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Cykly v DOM grafu

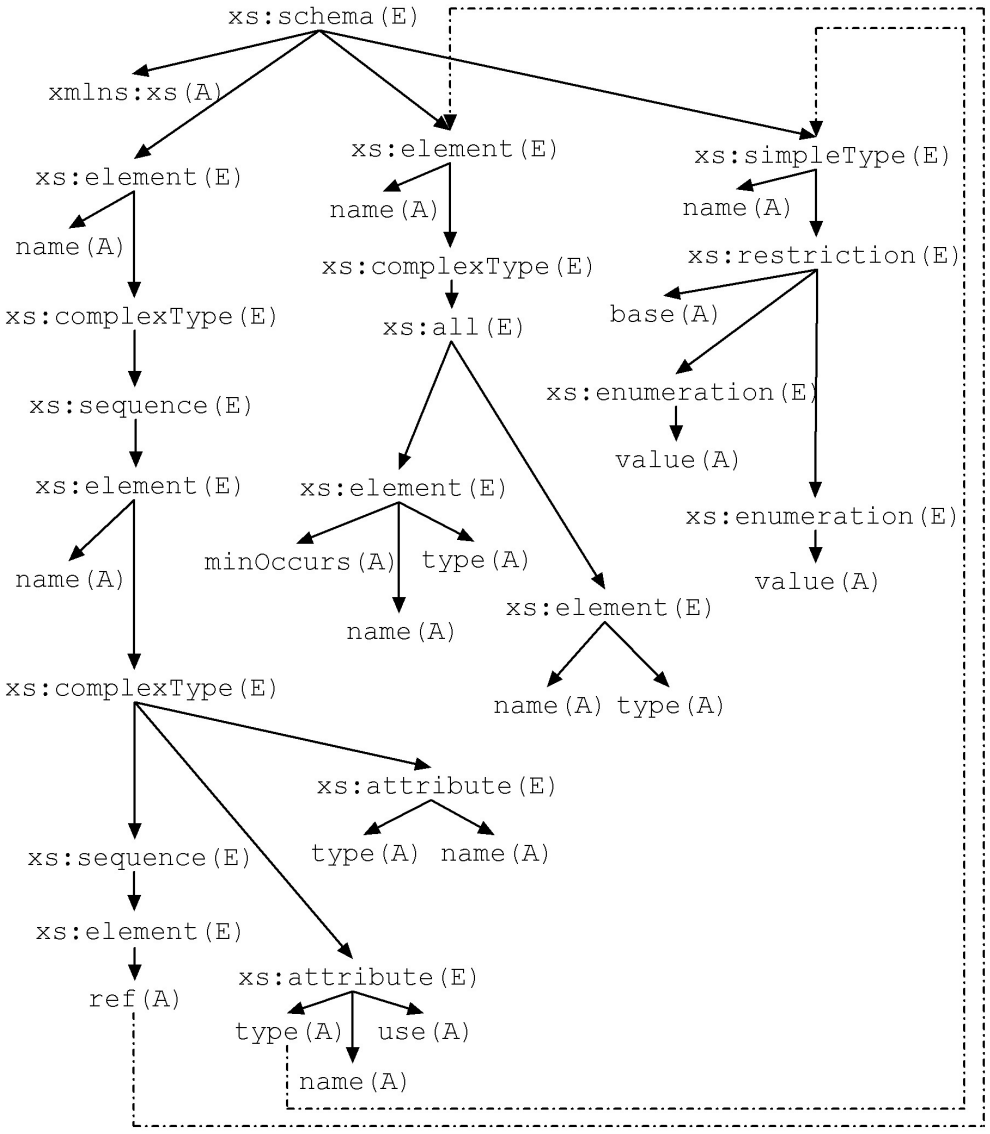
Norma jazyka XML Schema připouští v definici XML schématu několik typů cyklů, které mohou vznikat vzájemným použitím globálně definovaných elementů nebo globálně definovaných složených typů – např.

```
<xs:complexType name="t_podkapitola">
  <xs:sequence>
    <xs:element name="podkapitola" type="t_podkapitola"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="sekce">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="kapitola"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="kapitola">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sekce"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

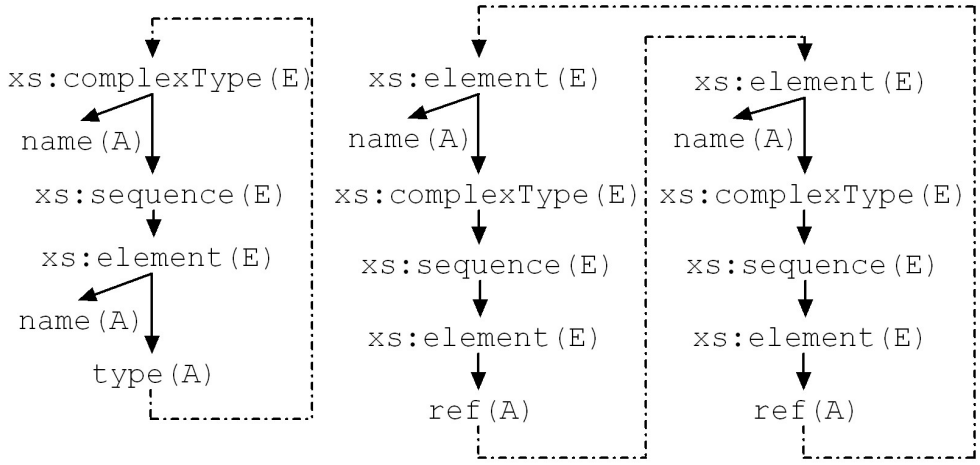


Obrázek 3: Příklad DOM stromu



Obrázek 4: Příklad DOM grafu

Z hlediska vytváření DOM grafu je možné přípustné cykly rozdělit na dvě skupiny – cykly v rámci podstromu jednoho globálního prvku a cykly procházející podstromy několika globálních prvků. Oba typy cyklů (odpovídajících předchozím dvěma příkladům) jsou schematicky znázorněny na obrázku 5.



Obrázek 5: Cykly v DOM grafu

Jak je uvedeno v [22], norma jazyka SQL:1999 prozatím nepřipouští v UDT žádnou cyklickou definici. Nicméně určité přípustné typy cyklů budou zřejmě začleněny do některé z příštích norem.

Na druhou stranu některé ze současných objektově relačních databázových systémů cykly v definici UDT podporují. Např. systém Oracle9i umožňuje (podobně jako objektově orientované programovací jazyky) definici tzv. *neúplného typu*, tj. typu, který je definován bez svého obsahu. Tento typ je poté možné použít v místech, kde by byl použit odpovídající úplný typ (s výjimkou použití jako předka při dědění) a dodefinovat až v okamžiku, kdy jsou definovány všechny typy, které jsou jeho součástí (tedy i ty typy, jejichž součástí neúplný typ je).

S využitím neúplných typů je možné zpracovat i DOM graf obsahující cykly. Jak již bylo řečeno, cykly vznikají pouze díky zpětným hranám (vedoucím do globálně definovaných uzlů). Před zpracováním DOM grafu tedy stačí pro všechny globálně definované složené typy a globálně definované elementy se složeným typem vytvořit odpovídající neúplný UDT, který bude využíván namísto dosud nedefinovaného úplného UDT. Takto předzpracované uzly jsou ve všech uzlech s výjimkou kořenového uzlu `schema` považovány za zpracované. Jejich dodefinování je provedeno až po zpracování všech jejich potomků.

Speciální zpracování potom vyžaduje situace, kdy je zpracovávána hrana vedoucí do globálně definovaného složeného typu, který má neúplný UDT:

- Pokud je složený typ rozšiřován (tj. jeho rozšíření je mapováno na dědičnost), musí být nejprve jeho UDT dodefinován. Jelikož není možné

vytvořit cyklus pouhým rozšiřováním složených typů, existuje v každém cyklu vždy alespoň jedna hrana, která nevede do rozšiřovaného typu. Tudíž ani při zpracování nemůže k zacyklení dojít.

- Pokud je složený typ přiřazen elementu, je tento element označen jako neúplný (tj. nejsou pro něj vytvářeny záznamy v pomocných tabulkách) a je vložen do seznamu neúplných elementů. Seznam těchto elementů je zpracován (tj. odpovídající záznamy jsou vytvořeny) až po zpracování celého DOM stromu (tj. po dodefinování neúplných typů).

Algoritmus vytvoření DOM grafu

Dle předchozího popisu probíhá algoritmus vytvoření DOM grafu následujícím způsobem:

1. Průchodem DOM stromu do hloubky vytvoř orientované hrany odpovídající hranám DOM stromu, vedoucích do prvků, které podléhají zpracování.
2. Pokud je při průchodu nalezen uzel odpovídající prvku výběru z elementů s vícenásobným výskytem, ještě před vytvořením odpovídající hrany DOM grafu proveď příslušnou modifikaci.
3. Při dosažení listu DOM stromu zkontroluj použití globálního elementu. Odpovídající hranu doplň v případě, že:
   (a) Daný listový uzel má atribut `base`, `type` nebo `itemType` odkazující se na globálně definovaný jednoduchý nebo složený typ.
   (b) Daný listový uzel má atribut `ref` odkazující se na globálně definovaný element, atribut, skupinu atributů nebo modelovou skupinu.
4. Pro globálně definované uzly odpovídající složeným typům:
   (a) Vytvoř jednoznačný identifikátor UDT.
   (b) Vytvoř neúplný UDT a ulož jeho název do DOM grafu.
5. Pro globálně definované uzly odpovídající elementům se složeným datovým typem:
   (a) Pokud se jedná o lokálně definovaný složený typ zpracuj jeho uzel stejně jako v předchozím případě.
   (b) Propaguj název UDT do uzlu elementu.
   (c) Vygeneruj a ulož do DOM grafu jednoznačný identifikátor elementu.

Fáze III. – Zpracování DOM grafu

Průchodem předzpracovaného DOM grafu jsou postupně vytvářeny jednotlivé prvky objektově relačního schématu a současně plněny pomocné tabulky. Algoritmus prochází graf do hloubky – nejprve jsou zpracováni všichni (doposud nezpracovaní) potomci aktuálního uzlu a na jejich základě je potom zpracován aktuální uzel.

Zpracování jednotlivých uzlů grafu dle typu XML prvku je uvedeno v následujících podkapitolách:

Zpracování jednoduchého typu

Podgraf odpovídající definici jednoduchého typu je tvořen hierarchií odvození aktuálního typu z již zpracovaného nebo vestavěného typu. Inicializace zpracování podgrafu je určena dle počátečního vestavěného typu (který je umístěn v listovém uzlu). Tento typ určuje odpovídající SQL datový typ a případně i inicializaci některých omezení restrikcí.

Jeden krok odvození je potom zpracován v závislosti na svém typu:

1. Jedná-li se o odvození restrikcí, propaguj doposud určené restrikce ze zpracovaného do aktuálního uzlu a rozšiř je o novou restrikci.
2. Jedná-li se o odvození seznamem, vynuluj doposud určené restrikce a změň příslušný SQL typ na pole původního SQL typu.
3. Jedná-li se o odvození sjednocením, vynuluj doposud určené restrikce a změň příslušný SQL typ na typ `VARCHAR`.

Zpracování atributu/skupiny atributů

Zpracování atributu je prováděno v následujících krocích:

1. Načti a zpracuj hodnoty atributů elementu `attribute`.
2. Vygeneruj jednoznačný identifikátor atributu.
3. Ulož odpovídající záznam do tabulky `xmlAttrTable`.
4. Ulož identifikátor atributu do DOM grafu (pro pozdější zaznamenání vztahů element-atribut).

Zpracování skupiny atributů odpovídá sloučení identifikátorů prvků, kterými je skupina tvořena do jednoho seznamu. Těmito prvky mohou být jednotlivé (již zpracované) atributy nebo skupiny atributů.

Zpracování modelové skupiny a jejího obsahu

Zpracování obsahu modelové skupiny se liší v závislosti na jeho typu (posloupnost/množina/výběr z elementů). Každý typ je mapován na tzv. pomocný element s vlastním UDT. Postup jejich vytvoření je následující:

1. Vygeneruj jednoznačný identifikátor UDT.
2. Dle typu obsahu vytvoř odpovídající UDT:
   (a) Pro obsah typu posloupnost elementů vytvoř UDT obsahující seznam atributů odpovídajících prvkům posloupnosti. (Jejich datové typy, tj. přímo UDT prvku, reference na UDT nebo pole referencí na UDT, jsou určeny dle typů prvků posloupnosti a hodnot jejich atributů `maxOccurs`.)
   (b) Pro obsah typu množina elementů vytvoř UDT obsahující seznam dvojic atributů odpovídajících prvku množiny a jeho pořadí. (Jejich datové typy, tj. přímo UDT prvku nebo reference na UDT, jsou určeny dle typů prvků množiny.)
   (c) Pro obsah typu výběr z elementů vytvoř UDT obsahující jediný atribut s datovým typem reference na společného předka UDT.
3. Ulož název UDT do DOM grafu.
4. Vygeneruj jednoznačný identifikátor (pomocného) elementu.
5. Ulož odpovídající záznam do tabulky `xmlElemTable`.
6. Pro všechny prvky posloupnosti/množiny/výběru z elementů, ulož do tabulky `xmlElemElemTable` odpovídající záznam. Pokud je daný prvek lokálně definovaný a je mapován na pole referencí nebo je prvkem výběru z elementů, nastav jeho uzlu příznak vytvoření typové tabulky.
7. Ulož identifikátor pomocného elementu do DOM grafu.

Zpracování modelové skupiny odpovídá propagaci informací o jejím obsahu do uzlu modelové skupiny a zpracování jejích atributů.

Zpracování složeného typu

Zpracování složeného typu (tj. vytvoření odpovídajícího UDT) probíhá v závislosti na typu jeho obsahu. Současně je vytvářen společný seznam identifikátorů atributů odpovídajících XML atributům a atributů odpovídajících (pomocným) podelementům. Identifikátory jsou do seznamu vkládány v tom pořadí, v němž byly odpovídající atributy deklarovány při vytváření UDT. Při rozšiřování složeného typu je seznam propagován od rozšiřovaného k rozšiřujícímu typu, tj. identifikátory nových atributů jsou doplněny do

původního seznamu. Tento seznam slouží pro pozdější zaznamenání vztahů element-atribut a element-podelement.

Postup zpracování složeného typu dle typů jeho obsahu je následující:

1. Má-li jednoduchý obsah:
   (a) Pokud nemá jednoznačný identifikátor UDT, vygeneruj ho.
   (b) Vytvoř UDT (odvozený od společného předka) obsahující seznam atributů odpovídajících všem XML atributům v podstromu složeného typu a jeden atribut pro uložení obsahu odpovídající příslušnému jednoduchému typu.
   (c) Vytvoř seznam identifikátorů atributů.
   (d) Ulož název UDT do DOM grafu.
2. Má-li složený obsah, závisí zpracování na typu složeného obsahu:
   (a) Jedná-li se o restrikci, je již odpovídající UDT vytvořen (restrikce typu je ignorována).
   (b) Jedná-li se o rozšíření:
      i. Pokud nemá jednoznačný identifikátor UDT, vygeneruj ho.
      ii. Pokud uzel rozšiřovaného typu nebyl zpracován, zpracuj jej.
      iii. Propaguj seznam identifikátorů atributů a elementů z uzlu rozšiřovaného typu do aktuálního uzlu.
      iv. Vytvoř UDT (odvozený od UDT rozšiřovaného typu), jehož obsah je určen obsahem rozšíření.
      v. Doplň seznam identifikátorů o identifikátory přidaných atributů a identifikátor pomocného podelementu.
      vi. Ulož název UDT do DOM grafu.
3. Je-li typu modelová skupina/posloupnost/množina/výběr z elementů:
   (a) Pokud nemá jednoznačný identifikátor UDT, vygeneruj ho.
   (b) Vytvoř UDT (odvozený od společného předka) obsahující:
      i. Seznam atributů odpovídajících všem XML atributům v podstromu složeného typu.
      ii. Jeden atribut pro uložení odkazu na pomocný element určující obsah. (Jeho datový typ, tj. přímo UDT pomocného elementu, reference na UDT nebo pole referencí na UDT, je určen dle typu pomocného elementu a hodnoty atributu `maxOccurs`.)
   (c) Vytvoř seznam identifikátorů atributů a identifikátoru pomocného elementu.
   (d) Ulož název UDT do DOM grafu.

UDT jsou navíc vytvořeny v závislosti na hodnotě atributu `abstract` elementu `complexType`.

Zpracování elementu

Zpracování elementu probíhá v následujících krocích:

1. Načti a zpracuj hodnoty atributů elementu `element`.
2. Pokud element nemá jednoznačný identifikátor, vygeneruj ho.
3. Dle datového typu elementu vytvoř odpovídající UDT:
   (a) Má-li jednoduchý typ:
      i. Vygeneruj jednoznačný identifikátor UDT.
      ii. Vytvoř UDT (odvozený od společného předka) tvořený jediným atributem pro uložení jednoduchého obsahu elementu.
      iii. Ulož identifikátor UDT do DOM grafu.
   (b) Má-li složený typ, je již odpovídající UDT vytvořen.
4. Dle datového typu elementu proveď následující:
   (a) Má-li globálně definovaný složený typ s neúplným UDT, ulož element do seznamu neúplných elementů.
   (b) Jinak:
      i. Ulož odpovídající záznam do tabulky `xmlElemTable`.
      ii. Pro všechny atributy elementu ulož odpovídající záznam do tabulky `xmlElemAttrTable`.
      iii. Pro všechny podelementy elementu ulož odpovídající záznam do tabulky `xmlElemElemTable`.
5. Ulož identifikátor elementu do DOM grafu.

Zpracování kořenového elementu

Zpracování kořenového elementu odpovídá zpracování elementu se složeným datovým typem (bez atributů) obsahujícím výběr z (globálních) elementů. Kořenový element má navíc vždy nastaven příznak vytvoření odpovídající typové tabulky.

Všem potomkům kořenového elementu typu element nebo modelová skupina (tj. globálně definovaným elementům nebo pomocným elementům) je navíc nastaven příznak vytvoření společné typové tabulky.

Fáze IV. – Závěrečná fáze

V závěrečné fázi algoritmu probíhá dodefinování neúplných typů, vytvoření typových tabulek a naplnění pomocné tabulky `xmlPathTable`.

Dodefinování neúplných elementů

Po zpracování celého DOM grafu je třeba dodefinovat neúplné elementy uložené v pomocném seznamu. Dodefinování spočívá v uložení odpovídajících záznamů do tabulek `xmlElemTable`, `xmlElemAttrTable`, `xmlElemElemTable`.

Vytváření tabulek

Pokud původní DOM graf neobsahoval cykly, je možné vytvářet odpovídající typové tabulky v okamžiku, kdy je vytvořen příslušný UDT, včetně specifikace omezení `SCOPE` pro reference (odpovídající referencovaná tabulka vždy existuje). Pokud původní DOM graf cykly obsahoval, je nutné nejprve provést vytvoření všech UDT schématu (tj. zajistit dodefinování neúplných typů), dalším průchodem vytvořit tabulky bez specifikace omezení `SCOPE` pro reference a specifikaci doplnit až při třetím průchodu, kdy již existují všechny tabulky.

Naplnění tabulky `xmlPathTable`

Naplnění tabulky `xmlPathTable` je možné provést jedním průchodem DOM grafu do hloubky. Pro každý uzel typu element je postupným procházením DOM grafu vytvářeno pole identifikátorů pomocných elementů. Pokud se průchod dostane do dalšího uzlu typu element, je odpovídající záznam uložen do tabulky a zpracování pokračuje od nově nalezeného uzlu nebo se vrací k poslednímu dosud nezpracovanému uzlu.

4.2 Algoritmus ukládání XML dokumentů

Tato kapitola obsahuje popis algoritmu uložení XML dokumentu validního vůči původnímu XML schématu do odpovídajícího objektově relačního schématu vytvořeného předchozím algoritmem.

Algoritmus je možné rozdělit do několika fází:

Fáze I. – Přípravná fáze

Přípravná fáze algoritmu probíhá v následujících krocích:

1. Vytvoř DOM strom XML dokumentu (včetně kontroly validity).
2. Z atributu `schemaLocation` nebo `noNamespaceSchemaLocation` kořenového elementu XML dokumentu zjisti URL XML schématu.
3. Pro URL XML schématu najdi v tabulce `xmlSchemaTable` odpovídající záznam. Pokud neexistuje, algoritmus končí.
4. Pro URL XML dokumentu a identifikátor XML schématu zkontroluj existenci záznamu v tabulce `xmlDocTable`. Pokud existuje (tj. XML dokument už byl uložen), algoritmus končí.

Fáze II. – Fáze ukládání XML dokumentu

Fáze ukládání XML dokumentu je založena na průchodu popisem mapování XML schématu (uloženém v pomocných tabulkách). Průchod je řízen strukturou DOM stromu ukládaného XML dokumentu. Popis mapování XML schématu odpovídá orientovanému grafu – jeho uzly tvoří elementy, pomocné elementy a atributy a hrany určují vztahy element-podelement a element-atribut. Vzhledem ke způsobu ukládání referencí musí být vždy nejprve rekurzivně zpracovány podelementy aktuálně zpracovávaného elementu, tj. graf je procházen do hloubky. Při návratu z rekurze jsou z dat v DOM stromu XML dokumentu generovány konstruktory instancí UDT odpovídajících elementů.

Jelikož prvky XML schématu mohou mít nepovinný výskyt, alternativní výskyt nebo počet výskytů daný rozsahem hodnot, je vždy nutné otestovat, která z variant byla v daném místě XML dokumentu zvolena. Vzhledem k předpokladu deterministického datového modelu je zvolená varianta vždy jednoznačně určena prvním úspěšným testem.

Uložení dat závisí na typu mapování elementu vzhledem k nadelementu:

- Pokud je element mapován na sloupec, je jeho konstruktor předán nadelementu, který jej uloží na odpovídající místo ve svém konstruktoru.
- Pokud je element mapován na referenci, jsou aktuální data uložena do odpovídající typové tabulky jako nový záznam. Nadelementu je předán `SELECT` příkaz odpovídající výběru tohoto záznamu, který jej uloží na odpovídající místo ve svém konstruktoru.
- Pokud je element mapován na pole referencí, je jeho zpracování podobné jako v předchozím případě. Pouze v konstruktoru nadelementu je `SELECT` příkaz (spolu s ostatními) uložen do konstruktoru pole referencí.

Podrobněji je možné algoritmus popsat takto: V průběhu ukládání je udržován ukazatel na právě zpracovávaný (aktuální) uzel DOM stromu ukládaného XML dokumentu, identifikátor aktuálního elementu v popisu mapování a typ jeho mapování vzhledem k nadelementu. (Typ mapování kořenového elementu odpovídá mapování na referenci, identifikátor vytvořeného záznamu je uložen do tabulky `xmlDocTable`.) Zpracovávání jednoho elementu tedy probíhá takto:

1. Načti z tabulky `xmlElemTable` odpovídající záznam.
2. Dle typu aktuálního elementu vytvoř konstruktor jeho obsahu:
   (a) Jedná-li se o element:
      i. Pokud název aktuálního uzlu neodpovídá XML názvu aktuálního elementu, ukonči jeho zpracování neúspěchem.

ii. Má-li jednoduchý typ, ulož na odpovídající místo v konstruktoru elementu konstruktor SQL datového typu obsahu.

iii. Má-li složený typ, odpovídá zpracování jeho obsahu zpracování posloupnosti elementů (viz. bod (b))

iv. Označ aktuální uzel za zpracovaný.

(b) Jedná-li se o pomocný element typu posloupnost elementů:

i. Z tabulky `xmlElemElemTable` načti do pomocného seznamu identifikátory podelementů (v pořadí dle atributu `order`).

ii. První dosud nezpracovaný identifikátor ze seznamu identifikátorů prohlaš za aktuální identifikátor.

iii. První dosud nezpracovaný uzel ze seznamu potomků aktuálního uzlu prohlaš za aktuální uzel. (Pokud už žádný není, ukonči zpracování obsahu.)

iv. Nový aktuální uzel a identifikátor rekurzivně zpracuj.

v. Pokud zpracování skončilo úspěchem:

A. Je-li podelement mapován na sloupec (nebo referenci), ulož jeho konstruktor (nebo příslušný `SELECT` příkaz) na odpovídající místo v konstruktoru elementu. Označ aktuální identifikátor za zpracovaný a pokračuj bodem ii.

B. Je-li podelement mapován na pole referencí, ulož odpovídající `SELECT` příkaz do seznamu `SELECT` příkazů aktuálního podelementu. Pokud je počet prvků v seznamu menší než maximální přípustný počet výskytů aktuálního podelementu, pokračuj bodem iii., jinak bodem vi.B.

vi. Pokud zpracování skončilo neúspěchem:

A. Jedná-li se o (první) podelement s povinným výskytem, ukonči zpracování celé posloupnosti neúspěchem.

B. Je-li seznam `SELECT` příkazů aktuálního podelementu neprázdný, vytvoř konstruktor pole referencí a ulož na odpovídající místo v konstruktoru elementu. (Je-li prázdný, ulož na odpovídající místo hodnotu `NULL`.) Označ aktuální identifikátor za zpracovaný a pokračuj bodem ii.

(c) Jedná-li se o pomocný element typu množina elementů:

i. První dosud nezpracovaný uzel ze seznamu potomků prohlaš za aktuální uzel. (Pokud už žádný není, pokračuj bodem v.)

ii. Pro název aktuálního uzlu a aktuální identifikátor vyhledej odpovídající (nezpracovaný) identifikátor podelementu. Pokud neexistuje, ukonči zpracování množiny neúspěchem, jinak prohlaš nalezený identifikátor za aktuální.

iii. Nový aktuální uzel a identifikátor rekurzivně zpracuj.

iv. Získaný konstruktor podelementu ulož (spolu s pořadím) na odpovídající místo v konstruktoru elementu. Identifikátor označ za zpracovaný a pokračuj bodem i.

v. Pro podelementy, které nebyly v dokumentu uvedeny, ulož na odpovídající místo v konstruktoru elementu hodnotu `NULL`.

(d) Jedná-li se o pomocný element typu výběr z elementů:

i. První dosud nezpracovaný uzel ze seznamu potomků prohlaš za aktuální uzel.

ii. Z tabulky `xmlElemElemTable` načti do pomocného seznamu identifikátory podelementů pomocného elementu.

iii. První dosud nezpracovaný identifikátor ze seznamu prohlaš za aktuální identifikátor.

iv. Nový aktuální uzel a identifikátor rekurzivně zpracuj.

v. Pokud zpracování skončilo úspěchem, ulož získaný konstruktor na odpovídající místo v konstruktoru elementu a ukonči zpracování obsahu.

vi. Pokud zpracování skončilo neúspěchem, označ aktuální identifikátor za zpracovaný a pokračuj bodem iii.

3. Vytvoř konstruktory atributů aktuálního elementu:

(a) V tabulce `xmlAttrTable` vyhledej pro každý atribut odpovídající záznam.

(b) Na odpovídající místo v konstruktoru elementu ulož konstruktor SQL datového typu hodnoty atributu.

(c) Pro atributy, které nebyly v XML dokumentu uvedeny ulož na odpovídající místo v konstruktoru elementu hodnotu `NULL`.

4. Jsou-li všechny položky konstruktoru aktuálního elementu prázdné, předej nadelementu hodnotu `NULL`.

5. Jinak vygeneruj jednoznačný identifikátor záznamu elementu.

6. Dle typu mapování elementu vzhledem k nadelementu proveď:

(a) Je-li mapován na referenci/pole referencí:

i. Ulož nový záznam do odpovídající typové tabulky.

ii. Pro identifikátor záznamu vytvoř odpovídající `SELECT` příkaz a ten předej nadelementu.

(b) Je-li mapován na sloupec nadelementu, předej nadelementu vytvořený konstruktor záznamu.

4.3 Algoritmus mapování jazyka XQL

Tato kapitola obsahuje popis algoritmu mapování dotazu v jazyce XQL nad zadaným XML dokumentem na posloupnost SQL dotazů nad objektově relačním schématem, v němž je zadaný dokument uložen. Výsledek SQL dotazu je následně transformován zpět na XML dokument.

Algoritmus má tedy dvě části – v první části jsou z databáze získána data vyhovující dotazu, ve druhé části je z těchto dat zkonstruován odpovídající XML dokument.

4.3.1 Algoritmus vytvoření SQL dotazu

V této kapitole je popsán algoritmus mapování XQL dotazu na posloupnost odpovídajících SQL dotazů. V algoritmu nejsou podporovány všechny typy dotazů – pro účely práce byla zvolena vhodná a ne příliš velká podmnožina, na níž je možné demonstrovat vlastnosti navrženého mapování. Přípustné jsou tedy pouze následující typy dotazů:

- Dotaz na elementy: `/knihovna/kniha/autor`
- Dotaz na hodnotu atributu: `/knihovna/kniha/@cena`
- Dotaz s filtrem:
  - Na nenulovou hodnotu atributu: `/knihovna/kniha[@cena]`
  - Na porovnání hodnoty elementu (s jednoduchým datovým typem obsahu) pomocí aktuálního kontextu:
    `/knihovna/kniha/autor[. = "Jim Melton"]`
  - Na porovnání hodnoty atributu:
    `/knihovna/kniha[@cena >= "20"]`

Vstupem algoritmu je tedy (dle výše uvedených pravidel) přípustný XQL dotaz a URL XML dokumentu, nad nímž se dotazujeme. Algoritmus je možné rozdělit do několika fází:

Fáze I. – Přípravná fáze

Přípravná fáze probíhá v následujících krocích:

1. Zkontroluj, zda je dotaz (dle výše uvedených pravidel) přípustný. Pokud ne, algoritmus končí.
2. V tabulce `xmlDocTable` vyhledej záznam odpovídající URL XML dokumentu. Pokud neexistuje, algoritmus končí.

Fáze II. – Převedení úseků dotazu

Tato fáze pracuje pouze s přípustnými XQL dotazy obsahujícími názvy elementů, názvy atributů a jim odpovídající filtry. Každý dotaz má tedy následující tvar:

`(/`*«název elementu»*`([`*«filtr»*`])?)+(/@`*«název atributu»*`)?`

Části dotazu oddělené znakem `'/'` budou v dalším textu nazývány *úseky*, přičemž zpracování celého dotazu odpovídá postupnému zpracovávání jednotlivých úseků. Zpracování jednoho úseku odpovídá příslušné modifikaci doposud vytvořených klauzulí `SELECT`, `FROM` a `WHERE` v závislosti na vlastnostech aktuálně zpracovávaného elementu (a jemu odpovídajícího filtru) nebo jeho atributu.

Na začátku jsou jednotlivé klauzule inicializovány takto:

- Klauzule `SELECT`: *«alias»*
- Klauzule `FROM`: *«tabulka kořenového elementu» «alias»*
- Klauzule `WHERE`: *«alias»*`.recordID` = *«identifikátor XML dokumentu»*

Dále je během zpracování udržován identifikátor právě zpracovávaného (aktuálního) elementu (který je inicializován na identifikátor kořenového elementu), identifikátor předchozího elementu a příznak použití reference (který je inicializován na `FALSE`).

Zpracování jednoho úseku odpovídajícího elementu probíhá následujícím způsobem:

1. Identifikátor aktuálního elementu označ za předchozí.
2. Dle názvu elementu aktuálního úseku a identifikátoru předchozího elementu vyhledej v tabulkách `xmlPathTable` a `xmlElemTable` identifikátor aktuálního elementu a „cestu“ přes pomocné elementy. (Pokud je nalezeno více záznamů, vygeneruj pro každý záznam vlastní dotaz.)
3. Pro jednotlivé dvojice identifikátorů element-podelement v „cestě“ (začínající identifikátorem předchozího elementu a prvního pomocného elementu a končící identifikátorem posledního pomocného elementu a aktuálního elementu) proveď dle typu mapování podelementu vzhledem k nadelementu následující:
   (a) Pokud byl nastaven příznak použití reference, prohlaš za aktuální „napojení“ `->` jinak `.` (tečku).
   (b) Jedná-li se o prvek výběru z elementů, přidej do klauzule `WHERE` podmínky:
   «klauzule ***SELECT****»* `IS OF (ONLY` *«typ elementu»*`)`
   «klauzule ***SELECT****»«napojení»*`elemID =` *«identifikátor aktuálního elementu»*

(c) Je-li mapován na sloupec:

i. Prodluž klauzuli `SELECT` o *«napojení»«název sloupce»*

ii. Příznak použití reference nastav na `FALSE`.

(d) Je-li mapován na referenci:

i. Prodluž klauzuli `SELECT` o *«napojení»«název sloupce»*

ii. Příznak použití reference nastav na `TRUE`.

(e) Je-li mapován na pole referencí:

i. Z aktuálních hodnot klauzulí vygeneruj pomocný dotaz.

ii. Klauzuli `SELECT` změň na: *«alias»*

iii. Klauzuli `FROM` změň na:
«pomocný dotaz» «alias dotazu»,
`UNNEST(`*«alias dotazu»«napojení»«název sloupce»*`)` *«alias»*

iv. Klauzuli `WHERE` vynuluj.

v. Příznak použití reference nastav na `TRUE`.

4. Obsahuje-li aktuální úsek filtr:

(a) Jedná-li se o filtr pro atribut:

i. Aktuální obsah všech tří klauzulí zkopíruj a doplň o krok nalezení odpovídajícího atributu.

ii. Jedná-li se o dotaz na nenulovou hodnotu, přidej do klauzule `WHERE` podmínku: *«doplněná klauzule* ***`SELECT`****»* `IS NOT NULL`

iii. Jinak do klauzule `WHERE` přidej odpovídající porovnání obsahu doplněné klauzule `SELECT` se zadanou hodnotou.

(b) Jedná-li se o filtr pro podelement:

i. Aktuální obsah všech tří klauzulí zkopíruj a doplň o krok nalezení odpovídajícího sloupce s obsahem elementu.

ii. Do klauzule `WHERE` přidej odpovídající porovnání obsahu doplněné klauzule `SELECT` se zadanou hodnotou.

Zpracování jednoho úseku odpovídajícího atributu probíhá následujícím způsobem:

1. V tabulkách `xmlElemAttrTable` a `xmlAttrTable` vyhledej odpovídající záznamy.

2. Pokud byl nastaven příznak použití reference, prohlaš za aktuální „napojení“ `->` jinak `.` (tečku).

3. Klauzuli `SELECT` prodluž o *«napojení»«název sloupce atributu»*

4.3.2 Algoritmus vytvoření XML dokumentu

V této kapitole je popsán algoritmus vytvoření XML dokumentu odpovídajícího výsledkům dotazů (sestavených z klauzulí) vygenerovaných předchozím algoritmem. Jak již bylo řečeno, mohou být dotazy dvou typů (dotazy na atribut nebo dotazy na element), jejichž zpracování se zásadním způsobem liší.

Samotné vytvoření XML dokumentu je prováděno přechodem přes jeho DOM strom. Algoritmy (pro oba typy dotazů) tedy postaví odpovídající DOM strom výsledků, který je následně uložen jako XML dokument.

Algoritmus pro dotazy na atributy

V případě dotazu na hodnotu atributu je vstupem algoritmu seznam trojic klauzulí SQL dotazů. Výsledkem dotazu na atribut je vždy jednosloupcová tabulka obsahující hodnoty tohoto atributu.

Algoritmus probíhá v následujících krocích:

1. Vytvoř prázdný elementový uzel s názvem `RESULT`.
2. Pro každou trojici klauzulí proveď:
   (a) Vygeneruj a vyvolej dotaz nad odpovídajícím schématem.
   (b) Pro každý záznam výsledné tabulky proveď:
      i. Je-li hodnota typu pole, vytvoř z jeho prvků seznam oddělený mezerami.
      ii. Vytvoř prázdný elementový uzel s obsahem daným záznamem a s názvem `VALUE_` + *«pořadové číslo»*
      iii. Přidej uzel do seznamu podelementů uzlu `RESULT`.
3. Ulož DOM strom jako XML dokument.

Algoritmus pro dotazy na elementy

V případě dotazu na element je vstupem algoritmu seznam trojic klauzulí SQL dotazů a jemu odpovídající seznam identifikátorů požadovaných elementů. Výsledkem dotazu na element je vždy jednosloupcová tabulka obsahující příslušný odkaz na daný element (tj. typový sloupec, referenci nebo pole referencí).

Algoritmus probíhá v následujících krocích:

1. Vytvoř prázdný elementový uzel s názvem `RESULT`.
2. Pro každou trojici klauzulí proveď:
   (a) Vygeneruj a vyvolej dotaz nad odpovídajícím schématem.

(b) V tabulce `xmlElemTable` vyhledej odpovídající záznam.

(c) Pro každý záznam výsledné tabulky proveď:

i. Vytvoř prázdný elementový uzel s názvem odpovídajícím názvu hledaného elementu.

ii. Rekurzivně zpracuj jednotlivé atributy elementu:

A. Zkopíruj obsah všech tří klauzulí aktuálního dotazu.

B. Do kopie klauzule `WHERE` přidej podmínku:
«kaluzule ***`SELECT`****»«napojení»*`recordID` = *«identifikátor aktuálního záznamu»*

C. Kopii klauzulí doplň o krok nalezení odpovídajícího atributu.

D. Nově vzniklé klauzule rekurzivně zpracuj a výsledné atributové uzly přidej do seznamu atributů právě zpracovávaného elementu.

iii. Nemá-li element obsah, ukonči zpracování aktuálního záznamu.

iv. Má-li element jednoduchý typ, ulož jeho hodnotu do elementového uzlu a zpracování aktuálního záznamu ukonči.

v. Má-li element složený typ, rekurzivně zpracuj jednotlivé podelementy:

A. Zkopíruj obsah všech tří klauzulí aktuálního dotazu.

B. Do kopie klauzule `WHERE` přidej podmínku:
«kaluzule ***`SELECT`****»«napojení»*`recordID` = *«identifikátor aktuálního záznamu»*

C. Kopii klauzulí doplň o krok nalezení odpovídajícího podelementu.

D. Nově vzniklé klauzule rekurzivně zpracuj a výsledné elementové uzly přidej do seznamu podelementů právě zpracovávaného elementu.

3. Ulož DOM strom jako XML dokument.

5 Implementace

V této kapitole jsou uvedeny informace týkající se ukázkové implementace navržených algoritmů. Jedná se o přehled použitých nástrojů, popis architektury a popis ovládání implementovaného systému.

5.1 Použité nástroje

Pro ukázkovou implementaci navržených algoritmů byl zvolen programovací jazyk *Java* (SDK 1.4[13]). Důvodem této volby bylo především velké množství knihoven pro práci s XML, přenositelnost výsledné aplikace a v neposlední řadě i předchozí dobré zkušenosti s tímto jazykem.

Z existujících databázových systémů byl po předchozí analýze zvolen systém *Oracle9i Release 2*[14]. Tento systém byl zvolen zejména proto, že podporuje největší množství (objektově orientovaných) rysů normy SQL:1999, které jsou využívány v navržených algoritmech. Prvky, které tento systém nepodporuje, následný způsob řešení tohoto nedostatku v ukázkové implementaci a odlišnosti v syntaxi SQL příkazů od normy SQL:1999 jsou podrobněji popsány v kapitole 5.1.1.

Pro komunikaci s databázovým systémem je využíváno standardní rozhraní pro práci s databázemi v jazyce Java – JDBC (Java Database Connectivity), konkrétně *Oracle JDBC Drivers release 9.2.0*.

Pro zpracování XML schémat i XML dokumentů (tj. pro kontrolu jejich validity a vytvoření jim odpovídajících DOM stromů) je využíván XML parser *Xerces2 Java Parser 2.5.0*[15]. Tento parser byl zvolen proto, že podporuje většinu standardů a doporučení konsorcia W3C týkajících se jazyka XML, XML Schema, programového rozhraní DOM a dalších. Malou nevýhodou je, že tento parser (a tudíž ani ukázková implementace) neumí rozpoznat nedeterministický datový model zpracovávaného XML schématu.

5.1.1 Specifika systému Oracle9i Release 2

Jak již bylo řečeno, systém Oracle 9i Release 2 byl zvolen proto, že podporuje největší množství prvků a rysů normy SQL:1999. Přesto tento systém nepodporuje všechny prvky normy, které jsou využívány v navržených algoritmech, nebo se u některých podporovaných prvků odlišuje v syntaxi. Tyto odlišnosti a nedostatky jsou (včetně řešení v ukázkové implementaci) popsány v následujících podkapitolách. Prvky, které zde nejsou uvedeny, podporuje systém dle normy.

[13] viz. `http://java.sun.com`

[14] viz. `http://www.oracle.com/products/`

[15] viz. `http://xml.apache.org/xerces2-j/`

Vestavěné typy a integritní omezení

Z jednoduchých datových typů nejsou podporovány typy `TIME` a `BOOLEAN` – v implementaci jsou nahrazeny typem `VARCHAR`, který je podporován pod názvem `VARCHAR2`. Z integritních omezení není podporován predikát `SIMILAR TO` – v implementaci je jeho využití vynecháno.

Složený datový typ pole je podporován pod názvem `VARRAY` a s mírně odlišnou syntaxí při vytváření i plnění – např.

```
CREATE TYPE pole_cen AS VARRAY(10) OF NUMERIC(12,2);
CREATE TABLE výrobky (
  název VARCHAR2(20),
  ceny pole_cen );
INSERT INTO výrobky
  VALUES ('Zmrzlina Mrož', pole_cen(8.50, 10.20, 9.30));
```

Operátor `CARDINALITY` podporován není – v implementaci je jeho využití vynecháno. Operátor `UNNEST` je podporován pod názvem `TABLE` (sloupec odhnízděné tabulky, v němž jsou prvky pole, se nazývá `COLUMN_VALUE`).

Uživatelsky definované typy a substituovatelnost

Uživatelsky definované typy jsou podporovány (s parametrem `FINAL` i `NOT FINAL`) s mírně odlišnou syntaxí při vytváření – např.

```
CREATE TYPE osoba AS OBJECT (
  jméno VARCHAR2(30),
  telefon VARCHAR2(20) ) NOT INSTANTIABLE NOT FINAL;
CREATE TYPE student UNDER osoba (id INT);
```

Predikát `DEFAULT` je možné specifikovat až při vytváření typové tabulky a pouze pro nehnízděné atributy – v implementaci je tudíž jeho využití vynecháno. Predikát `NOT NULL` není možné specifikovat pro hnízděné atributy typu reference. Typový predikát `IS OF` je při dotazování podporován v kombinaci s operátorem `TREAT` pro přetypování výsledku na odpovídající datový typ – např.

```
INSERT INTO lidé
  VALUES (student('Irena Mlýnková','603773917',13));
SELECT TREAT(VALUE(li) AS student).id FROM lidé li
  WHERE VALUE(li) IS OF (student);
```

Reference a jejich integrita

Reference jsou podporovány a připouští možnost ukládat do referencovaného místa i odvozené UDT. Rozdíl je pouze v syntaxi při přístupu na atributy referencovaného UDT – místo operátoru `->` je definován operátor `DEREF`.

Určení typu identifikace referencovaného UDT se provádí pouze při vytváření odpovídající typové tabulky, implicitním typem je systémově generovaný identifikátor. Integritní omezení `SCOPE` i určení referenční integrity je přípustné pouze pro sloupce typu reference, nikoli pro sloupce typu pole referencí.

5.2 Architektura

Aplikace představující ukázkovou implementaci navržených algoritmů byla nazvána *XMLSchemaStore*. Její architektura sestává s několika hlavních a pomocných modulů odpovídajících třídám jazyka Java.

Hlavními moduly jsou:

- `XMLSchemaStore` – hlavní část celé aplikace zajišťující komunikaci s uživatelem a volání ostatních modulů
- `XMLSchema2DB` – modul zajišťující vytvoření objektově relačního schématu na základě XML schématu (a případně i vytvoření pomocných tabulek)
- `XMLdoc2DB` – modul zajišťující uložení dat z XML dokumentu do již existujícího objektově relačního schématu
- `XQL2XML` — modul zajišťující mapování XQL dotazu na SQL dotaz(y) a následné vytvoření odpovídajícího XML dokumentu, který obsahuje výsledky dotazu

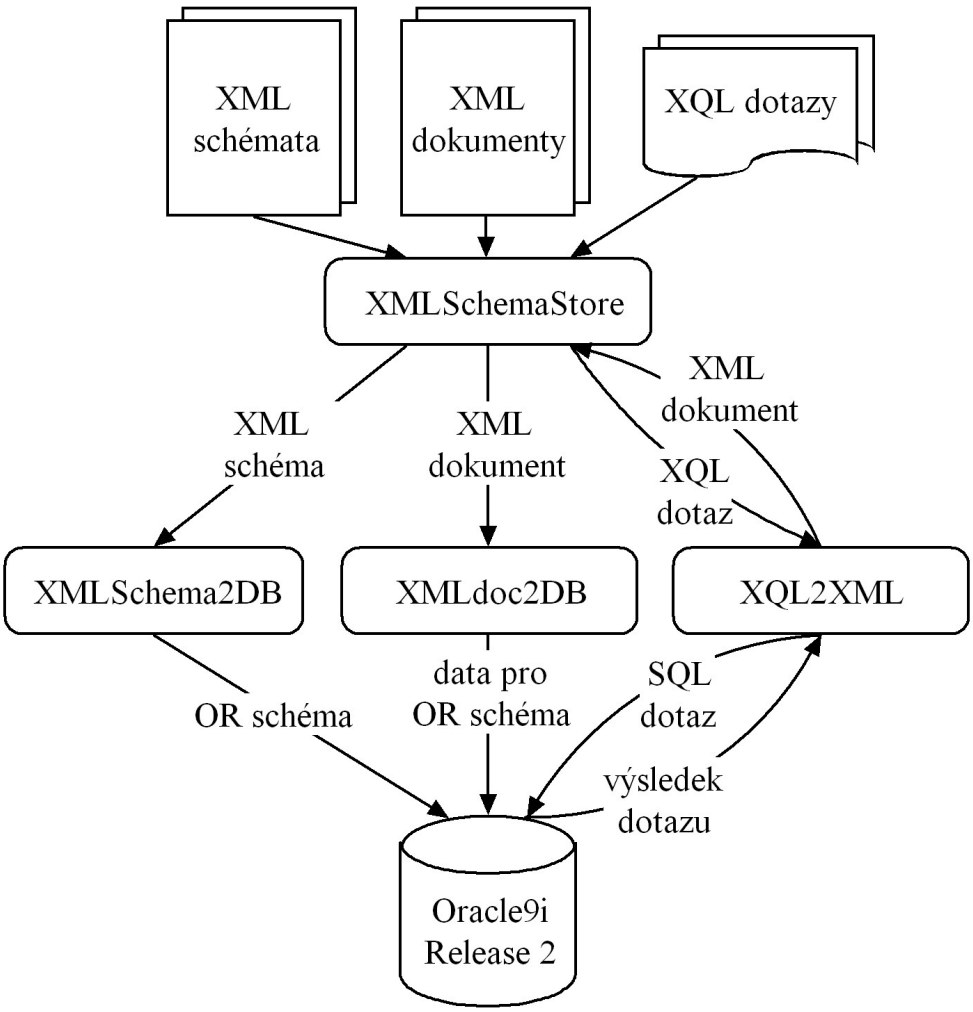
Nástin vzájemného propojení hlavních modulů je schematicky zobrazen na obrázku 6.

Mezi pomocné moduly patří modul `DBCall` obsahující funkce pro komunikaci s databázovým systémem a sada modulů definujících a obsluhujících jednotlivá okna aplikace.

5.3 Ovládání aplikace

S aplikací XMLSchemaStore je možné pracovat pouze pomocí grafického uživatelského rozhraní (GUI), jehož prostřednictvím jsou zadávány parametry jednotlivých operací.

Po spuštění aplikace se zobrazí hlavní okno programu, v jehož horní části se nachází ovládací menu a zbylou plochu vyplňuje textové pole, které slouží pro výpisy informací o průběhu prováděných operací. Submenu *Program* obsahuje položky pro připojení k databázovému systému (*Connect...*), odpojení od systému (*Disconnect*) a ukončení celého programu (*Close*). Submenu *Process* obsahuje položky pro zpracování XML schématu (*XML schema...*), XML dokumentu (*XML document...*) a XQL dotazu (*XQL query...*). Submenu *Help* obsahuje položku (*About...*) pro otevření okna s informacemi o programu.



Obrázek 6: Schéma architektury aplikace

Doporučený postup práce s aplikací je následující:

1. Z menu *Program/Connect...* otevřít okno *Connect* obsahující položky pro zadání parametrů pro připojení k databázovému systému, zadat příslušné parametry a potvrdit stiskem tlačítka `[OK]`.

2. Z menu *Process/XML schema...* otevřít dialog pro volbu souboru, zvolit požadované XML schéma a potvrdit stiskem tlačítka `[Open]`.

3. Z menu *Process/XML document...* otevřít dialog pro volbu souboru, zvolit požadovaný XML soubor a potvrdit stiskem tlačítka `[Open]`.

4. Z menu *Process/XQL query...* otevřít okno *XQL query* obsahující položku pro zadání XQL dotazu, zadat dotaz a potvrdit stiskem tlačítka `[OK]`. V následujícím okně *XML document* zvolit XML dokument ulo-

žený v databázi, nad kterým je dotaz pokládán a potvrdit stiskem tlačítka `[OK]`.

Do textového pole v hlavním okně programu je vypisován průběh jednotlivých operací. Jedná se o informativní a chybová hlášení a o výpisy výsledků jednotlivých operací:

- Po zpracování XML schématu je vypsán seznam SQL příkazů pro vytvoření odpovídajícího objektově relačního schématu.
- Po zpracování XML dokumentu je vypsán seznam SQL příkazů pro uložení XML dat do odpovídajícího objektově relačního schématu.
- Po zpracování XQL dotazu je vypsán seznam SQL příkazů, na něž byl XQL dotaz mapován a obsah výsledného XML dokumentu.

Tyto výpisy jsou navíc ukládány do pomocného adresáře `/tmp` do souborů:

- `listingXMLSchema.txt` – SQL příkazy pro zpracování posledního zadaného XML schématu
- `listingXMLdoc.txt` – SQL příkazy pro uložení posledního zadaného XML dokumentu
- `listingXQL.txt` a `XMLresult.xml` – SQL dotazy a výsledný XML dokument odpovídající poslednímu zadanému XQL dotazu

V tomto adresáři je navíc generován script `dropScript.sql` obsahující příkazy pro zrušení všech prvků objektově relačního schématu, které byly aplikací vytvořeny od posledního připojení k databázi.

Zdrojové i přeložené soubory aplikace jsou uloženy na přiloženém CD-ROM, popis jeho obsahu je uveden v příloze B. Příklad XML schématu, XML dokumentu, XQL dotazů nad dokumentem a jim odpovídající výsledky zpracování aplikací jsou uvedeny v příloze C.

6 Závěr

Průběh vzniku této diplomové práce je možné rozdělit na dvě části – teoretickou a praktickou. Teoretická část probíhala od průzkumu základních informací až po návrh algoritmů, praktickou část tvoří implementace navržených algoritmů.

V teoretické části bylo nejprve nutné detailně prostudovat normy a specifikace jazyků XML a XML Schema. Na základě tohoto průzkumu (a zjištěných objektově orientovaných rysů jazyka XML Schema) bylo jako cílové databázové schéma zvoleno objektově relační schéma definované normou SQL:1999. Jazyk XQL (respektive jeho podmnožina) byla zvolena na základě analýzy dotazovacích jazyků nad XML dokumenty pro svoji jednoduchost a současně dostatečnou vyjadřovací sílu. V další fázi teoretické části proběhla podrobná analýza existujících přístupů k zadanému problému a jejich zhodnocení. Na základě této analýzy (tj. zjištěných výhod a nevýhod již existujících řešení) a s přihlédnutím ke specifickým vlastnostem jazyka XML Schema byly následně navrženy vlastní algoritmy.

V praktické části byly nejprve nalezeny vhodné pomocné nástroje (jako např. XML parser apod.) a s jejich využitím následně implementovány algoritmy navržené v teoretické části.

Vlastní přínos této práce spočívá především ve volbě cílového (objektově relačního) schématu a jemu odpovídající návrh algoritmů. Přestože jsou výsledkem algoritmu opět relace, je postup jejich vytvoření (tj. s využitím uživatelsky definovaných typů) a zajištění jejich vzájemné referenční integrity (tj. s využitím referencí) z hlediska analyzovaných existujících řešení poměrně originální. Jak již bylo řečeno, tato myšlenka byla inspirována objektově orientovanými rysy jazyka XML Schema a jejich využitím při mapování na třídy objektově orientovaných programovacích jazyků v technologiích *XML Data Binding.* Původní je také návrh modifikace DOM stromu, tj. definice tzv. *DOM grafu* a způsob jeho zpracování (včetně zpracování jeho případných kružnic), který byl částečně inspirován tzv. *DTD grafem* specifikovaným v článku [28] zabývajícím se mapováním schématu dat popsaném v jazyce DTD.

Závěrem je možné říci, že všechny předem stanovené cíle práce byly v zásadě splněny. Přesto je v navrženém řešení prostor pro další rozšíření. Jedním z nejzajímavějších by mohlo být využití substitučních skupin, které poskytují velmi silný nástroj pro modifikaci XML schématu. Dalším možným rozšířením by mohlo být využití prvků pro omezení identity (tj. elementů `unique`, `key` a `keyref` a datových typů `ID`, `IDREF` a `IDREFS`), jejichž specifických vlastností nebylo možné využít vzhledem ke stávajícím vlastnostem normy SQL:1999. Jak již bylo uvedeno v popisu základních prvků této normy, s jejím využitím jsou spojeny i další nedostatky, související s prozatím nedořešenými detaily jejích objektově orientovaných rysů. Zajímavé modifikace navržených algoritmů by tedy mohlo přinést využití nové normy jazyka SQL, která by měla (jak je uvedeno např. v [22]) obsahovat další rozšíření těchto rysů.

Reference

[1] *XML Schema Tutorial*. W3Schools. www.w3schools.com/schema/default.asp.

[2] S. Amer-Yahia, M. Fernández. *Overview of Existing XML Storage Techniques*. AT&T Labs, 2002. www.research.att.com/~sihem/publications/SIGRECORD02.pdf.

[3] P. V. Biron, A. Malhotra. *XML Schema Part 2: Datatypes*. W3C, 2001. www.w3.org/TR/xmlschema-2/.

[4] P. Bohannon, J. Freire, P. Roy, J. Simeon. *From XML Schema to Relations: A Cost-Based Approach to XML Storage*. 2002.

[5] R. Bourret. *Mapping DTDs to Databases*. 2001. www.xml.com/pub/a/2001/05/09/dtdtodbs.html.

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

[7] R. Bourret. *XML and Databases*. 2003. www.rpbourret.com/xml/XMLAndDatabases.htm.

[8] R. Bourret. *XML Database Products*. 2003. www.rpbourret.com/xml/XMLDatabaseProds.htm.

[9] R. Bourret, C. Bornhövd, A. P. Buchmann. *A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases*. 1999.

[10] N. Bradley. *XML, kompletní průvodce*. Grada, 2000.

[11] T. Bray, D. Hollander, A. Layman. *Namespaces in XML*. W3C, 1999. www.w3.org/TR/REC-xml-names/.

[12] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C, 2000. www.w3.org/TR/REC-xml.

[13] M. Champion, S. Byrne, G. Nicol, L. Wood. *Document Object Model (Core) Level 1*. W3C, 1997. www.w3.org/TR/REC-DOM-Level-1/level-one-core.html.

[14] J. Clark, S. DeRose. *XML Path Language (XPath) Version 1.0*. W3C. www.w3.org/TR/xpath.

[15] I. Dayen. *Storing XML in Relational Databases*. 2001. www.xml.com/pub/a/2001/06/20/databases.html.

[16] D. C. Fallside. *XML Schema Part 0: Primer*. W3C, 2001. www.w3.org/TR/xmlschema-0/.

[17] D. Florescu, D. Kossmann. *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*. 1999. www-personal.umich.edu/~krunapon/research/xml/xml.pdf.

[18] K. Havlík. *XML a relační databáze*. ČVUT Praha, 2002.

[19] J. Kosek. *XML pro každého, podrobný průvodce*. Grada, 2000.

[20] D. Lee, M. Mani, W. W. Chu. *Effective Schema Conversions between XML and Relational Models*. 2002. www.cobase.cs.ucla.edu/tech-docs/dongwon/ecaiot02.pdf.

[21] D. Megginson. *SAX*. SourceForge, 1998. www.saxproject.org/.

[22] J. Melton. *Advanced SQL: 1999 - Understanding Object-Relational and Other Advanced Features*. Morgan Kaufmann Publishers, 2003.

[23] J. Melton, A. R. Simon. *SQL: 1999 - Understanding Relational Language Components*. Morgan Kaufmann Publishers, 2002.

[24] J. Papež. *XML schéma a jeho reprezentace pomocí relační databáze*. ČVUT Praha, 2003.

[25] J. Pokorný. *Dotazovací jazyky*. Karolinum, 2002.

[26] J. Robie. *XQL (XML Query Language)*. 1999. www.ibiblio.org/xql/xql-proposal.html.

[27] J. Robie, J. Lapp, M. F. Fernandez, D. Schach. *XML Query Language (XQL)*. W3C. www.w3.org/TandS/QL/QL98/pp/xql.html.

[28] J. Shanmugasundaram, K. Tufte, G. He a další. *Relational Databases for Querying XML Documents: Limitations and Opportunities*. 1999. www-personal.umich.edu/~krunapon/research/xml/RdbmsForXML.pdf.

[29] T. Sundsted. *Storing XML Data*. SUN.com, 2002.

[30] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn. *XML Schema Part 1: Structures*. W3C, 2001. www.w3.org/TR/xmlschema-1/.

[31] I. Varlamis, M. Vazirgiannis. *Bridging XML-Schema and relational databases*. 2001. www.db-net.aueb.gr/hercules/papers/doceng01.pdf.

A Tabulky

A.1 Přípustná omezení jednoduchých typů

V tabulkách 12, 13 a 14 je uveden přehled přípustných omezení restrikcí pro vestavěné jednoduché datové typy, v tabulce 15 pro jednoduché typy odvozené seznamem a sjednocením. Znak • určuje, že je daný typ restrikce přípustný pro příslušný datový typ. Legenda ke sloupcům tabulek je uvedena v tabulce 11.

A	`length`	**E**	`enumeration`	**I**	`minInclusive`
B	`minLength`	**F**	`whiteSpace`	**J**	`minExclusive`
C	`maxLength`	**G**	`maxInclusive`	**K**	`totalDigits`
D	`pattern`	**H**	`maxExclusive`	**L**	`fractionDigits`

Tabulka 11: Legenda k tabulkám přípustných omezení

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**
`string`	•	•	•	•	•	•						
`boolean`				•		•						
`decimal`				•	•	•	•	•	•	•	•	•
`float`				•	•	•	•	•	•	•		
`double`				•	•	•	•	•	•	•		
`duration`				•	•	•	•	•	•	•		
`dateTime`				•	•	•	•	•	•	•		
`time`				•	•	•	•	•	•	•		
`date`				•	•	•	•	•	•	•		
`gYearMonth`				•	•	•	•	•	•	•		
`gYear`				•	•	•	•	•	•	•		
`gMonthDay`				•	•	•	•	•	•	•		
`gDay`				•	•	•	•	•	•	•		
`gMonth`				•	•	•	•	•	•	•		
`hexBinary`	•	•	•	•	•	•						
`base64Binary`	•	•	•	•	•	•						
`anyURI`	•	•	•	•	•	•						
`QName`	•	•	•	•	•	•						
`NOTATION`	•	•	•	•	•	•						

Tabulka 12: Přípustná omezení základních vestavěných datových typů

	A	B	C	D	E	F	G	H	I	J	K	L
`normalizedString`	•	•	•	•	•	•						
`token`	•	•	•	•	•	•						
`language`	•	•	•	•	•	•						
`NMTOKEN`	•	•	•	•	•	•						
`NMTOKENS`	•	•	•		•	•						
`Name`	•	•	•	•	•	•						
`NCName`	•	•	•	•	•	•						
`ID`	•	•	•	•	•	•						
`IDREF`	•	•	•	•	•	•						
`IDREFS`	•	•	•		•	•						
`ENTITY`	•	•	•	•	•	•						
`ENTITIES`	•	•	•		•	•						

Tabulka 13: Přípustná omezení typů odvozených od typu `string`

	A	B	C	D	E	F	G	H	I	J	K	L
`integer`				•	•	•	•	•	•	•	•	•
`positiveInteger`				•	•	•	•	•	•	•	•	•
`negativeInteger`				•	•	•	•	•	•	•	•	•
`nonPositiveInteger`				•	•	•	•	•	•	•	•	•
`nonNegativeInteger`				•	•	•	•	•	•	•	•	•
`long`				•	•	•	•	•	•	•	•	•
`int`				•	•	•	•	•	•	•	•	•
`short`				•	•	•	•	•	•	•	•	•
`byte`				•	•	•	•	•	•	•	•	•
`unsignedLong`				•	•	•	•	•	•	•	•	•
`unsignedInt`				•	•	•	•	•	•	•	•	•
`unsignedShort`				•	•	•	•	•	•	•	•	•
`unsignedByte`				•	•	•	•	•	•	•	•	•

Tabulka 14: Přípustná omezení typů odvozených od typu `decimal`

	A	B	C	D	E	F	G	H	I	J	K	L
`list`	•	•	•		•	•						
`union`				•	•							

Tabulka 15: Přípustná omezení typů odvozených seznamem a sjednocením

A.2 Přípustný obsah elementů

V tabulkách 16 a 17 je uveden přehled přípustných obsahů elementů jazyka XML Schema, vyjádřených pomocí Kleeneho operátorů (+, * a ?) a logického nebo (|).

Element:	**Přípustný obsah:**
all	(annotation?, element*)
annotation	(appinfo | documentation)*
any	(annotation?)
anyAttribute	(annotation?)
appInfo	(any)*
attribute	(annotation?, (simpleType?))
attributeGroup	(annotation?, ((attribute | attributeGroup)*, anyAttribute?))
choice	(annotation?, (element | group | choice | sequence | any)*)
complexContent	(annotation?, (restriction | extension))
complexType	(annotation?, (simpleContent | complexContent | ((group | all | choice | sequence)?, ((attribute | attributeGroup)*, anyAttribute?))))
documentation	(any)*
element	(annotation?, ((simpleType | complexType)?, (unique | key | keyref)*))
extension (simpleContent)	(annotation?, ((attribute | attributeGroup)*, anyAttribute?))
extension (complexContent)	(annotation?, ((group | all | choice | sequence)?, ((attribute | attributeGroup)*, anyAttribute?)))
enumeration	(annotation?)
field	(annotation?)
fractionDigits	(annotation?)
group	(annotation?, (all | choice | sequence))
import	(annotation?)
include	(annotation?)
key	(annotation?, (selector, field+))
keyref	(annotation?, (selector, field+))
length	(annotation?)
list	(annotation?, (simpleType?))
maxInclusive	(annotation?)
maxExclusive	(annotation?)

Tabulka 16: Přípustný obsah elementů – část 1.

Element:	Přípustný obsah:
maxLength	(annotation?)
minInclusive	(annotation?)
minExclusive	(annotation?)
minLength	(annotation?)
notation	(annotation?)
pattern	(annotation?)
redefine	(annotation | (simpleType | complexType | group | attributeGroup))*
restriction (simpleType)	(annotation?, (simpleType?, (minExclusive | minInclusive | maxExclusive | maxInclusive | totalDigits | fractionDigits | length | minLength | maxLength | enumeration | whiteSpace | pattern)*))
restriction (simpleContent)	(annotation?, (simpleType?, (minExclusive | minInclusive | maxExclusive | maxInclusive | totalDigits | fractionDigits | length | minLength | maxLength | enumeration | whiteSpace | pattern)*)?, ((attribute | attributeGroup)*, anyAttribute?))
restriction (complexContent)	(annotation?, (group | all | choice | sequence)?, ((attribute | attributeGroup)*, anyAttribute?))
schema	((include | import | redefine | annotation)*, (((simpleType | complexType | group | attributeGroup) | element | attribute | notation), annotation*)*)
selector	(annotation?)
sequence	(annotation?, (element | group | choice | sequence | any)*)
simpleContent	(annotation?, (restriction | extension))
simpleType	(annotation?, (restriction | list | union))
totalDigits	(annotation?)
union	(annotation?, (simpleType*))
unique	(annotation?, (selector, field+))
whiteSpace	(annotation?)

Tabulka 17: Přípustný obsah elementů – část 2.

B Obsah CD-ROM

Součástí této diplomové práce je i přiložený CD-ROM obsahující text práce a především zdrojové a přeložené soubory aplikace XMLSchemaStore. CD-ROM obsahuje následující soubory a adresáře:

- `obsah.txt` – soubor s popisem obsahu CD-ROM
- `/text` – adresář obsahující text diplomové práce
- `/src` – adresář obsahující:
  - zdrojové soubory (`*.java`) aplikace
  - pomocné soubory (`*.txt`) pro překlad aplikace
  - dávkový soubor (`make.bat`) obsahující příkazy pro překlad aplikace pod systémy Win32
  - podadresáře s používanými JAR soubory (`/lib`) a s obrázky (`/images`)
- `/jar` – adresář obsahující:
  - výsledný JAR soubor `XMLSchemaStore.jar`
  - dávkový soubor (`run.bat`) obsahující příkaz pro spuštění aplikace
  - podadresář s používanými JAR soubory (`/lib`)
  - podadresář s příklady XML schémat a XML souborů (`/data`)

C Příklad

V této kapitole je uveden příklad XML schématu, XML dokumentu, XQL dotazů nad tímto dokumentem a jim odpovídající výsledky zpracování aplikací XMLSchemaStore.

XML schéma

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="zamestnanci">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="osoba" minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="osoba">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="jmeno"/>
    <xs:element ref="email"  minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="url"    minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="vztahy" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
   <xs:attribute name="id"       type="xs:ID" use="required"/>
   <xs:attribute name="poznamka" type="xs:string"/>
   <xs:attribute name="dovolena" default="ne">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="ano"/>
      <xs:enumeration value="ne"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="plat" type="xs:integer"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="jmeno">
  <xs:complexType>
   <xs:all>
    <xs:element ref="krestni" minOccurs="0"/>
    <xs:element ref="prijmeni"/>
   </xs:all>
  </xs:complexType>
 </xs:element>
```

```
 <xs:element name="krestni" type="xs:string"/>

 <xs:element name="prijmeni" type="xs:string"/>

 <xs:element name="email" type="xs:string"/>

 <xs:element name="url">
  <xs:complexType>
   <xs:attribute name="href" type="xs:string" default="http://"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="vztahy">
  <xs:complexType>
   <xs:attribute name="vedouci" type="xs:IDREF"/>
   <xs:attribute name="podrizeni" type="xs:IDREFS"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Vytvoření objektově relačního schématu

```
INSERT INTO xmlSchemaTable
   VALUES ('zamestnanci.xsd', 1, NULL);
CREATE TYPE xmlAncestor_1 AS OBJECT (
            elemID INT, recordID INT ) NOT FINAL;

CREATE TYPE E_1_1;
CREATE TYPE E_1_2;
CREATE TYPE E_1_3;
CREATE TYPE E_1_4;
CREATE TYPE E_1_5;

CREATE TYPE P_1_6_1 AS VARRAY(100) OF REF E_1_2;
CREATE TYPE P_1_6 UNDER xmlAncestor_1 (
            C_1 P_1_6_1 ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 6, null, 'C', 'S', 'P_1_6');
INSERT INTO xmlElemElemTable
   VALUES (1, 6, 2, 1, 'A', 1, 100);

CREATE TYPE E_1_1 UNDER xmlAncestor_1 (
            C_1 P_1_6 ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 1, 'zamestnanci', 'C', 'E', 'E_1_1');
INSERT INTO xmlElemElemTable
   VALUES (1, 1, 6, 1, 'C', 1, 1);
```

```
CREATE TYPE E_1_7 UNDER xmlAncestor_1 (
            C_1 VARCHAR2(200) ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 7, 'email', 's', 'E', 'E_1_7');

CREATE TYPE P_1_8_2 AS VARRAY(100) OF REF E_1_7;
CREATE TYPE P_1_8_3 AS VARRAY(100) OF REF E_1_4;
CREATE TYPE P_1_8 UNDER xmlAncestor_1 (
            C_1 REF E_1_3,
            C_2 P_1_8_2,
            C_3 P_1_8_3,
            C_4 REF E_1_5 ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 8, null, 'C', 'S', 'P_1_8');
INSERT INTO xmlElemElemTable
   VALUES (1, 8, 3, 1, 'R', 1, 1);
INSERT INTO xmlElemElemTable
   VALUES (1, 8, 7, 2, 'A', 0, 100);
INSERT INTO xmlElemElemTable
   VALUES (1, 8, 4, 3, 'A', 0, 100);
INSERT INTO xmlElemElemTable
   VALUES (1, 8, 5, 4, 'R', 0, 1);

INSERT INTO xmlAttrTable
   VALUES (1, 1, 'id', 's');

INSERT INTO xmlAttrTable
   VALUES (1, 2, 'poznamka', 's');

INSERT INTO xmlAttrTable
   VALUES (1, 3, 'dovolena', 's');

INSERT INTO xmlAttrTable
   VALUES (1, 4, 'plat', 'n');

CREATE TYPE E_1_2 UNDER xmlAncestor_1 (
            A_1 VARCHAR2(200),
            A_2 VARCHAR2(200),
            A_3 VARCHAR2(200),
            A_4 DECIMAL(38,0),
            C_5 P_1_8 ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 2, 'osoba', 'C', 'E', 'E_1_2');
INSERT INTO xmlElemAttrTable
   VALUES (1, 2, 1, 1);
```

```
INSERT INTO xmlElemAttrTable
   VALUES (1, 2, 2, 2);
INSERT INTO xmlElemAttrTable
   VALUES (1, 2, 3, 3);
INSERT INTO xmlElemAttrTable
   VALUES (1, 2, 4, 4);
INSERT INTO xmlElemElemTable
   VALUES (1, 2, 8, 5, 'C', 1, 1);

CREATE TYPE E_1_9 UNDER xmlAncestor_1 (
            C_1 VARCHAR2(200) ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 9, 'krestni', 's', 'E', 'E_1_9');

CREATE TYPE E_1_10 UNDER xmlAncestor_1 (
            C_1 VARCHAR2(200) ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 10, 'prijmeni', 's', 'E', 'E_1_10');

CREATE TYPE P_1_11 UNDER xmlAncestor_1 (
            C_1 REF E_1_9,
            O_1 INT,
            C_2 REF E_1_10,
            O_2 INT ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 11, null, 'C', 'A', 'P_1_11');
INSERT INTO xmlElemElemTable
   VALUES (1, 11, 9, 1, 'R', 0, 1);
INSERT INTO xmlElemElemTable
   VALUES (1, 11, 10, 2, 'R', 1, 1);

CREATE TYPE E_1_3 UNDER xmlAncestor_1 (
            C_1 P_1_11 ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 3, 'jmeno', 'C', 'E', 'E_1_3');
INSERT INTO xmlElemElemTable
   VALUES (1, 3, 11, 1, 'C', 1, 1);

INSERT INTO xmlAttrTable
   VALUES (1, 5, 'href', 's');

CREATE TYPE E_1_4 UNDER xmlAncestor_1 (
            A_1 VARCHAR2(200) ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 4, 'url', 'o', 'E', 'E_1_4');
INSERT INTO xmlElemAttrTable
   VALUES (1, 4, 5, 1);
```

```
INSERT INTO xmlAttrTable
   VALUES (1, 6, 'vedouci', 's');

INSERT INTO xmlAttrTable
   VALUES (1, 7, 'podrizeni', 'S');

CREATE TYPE E_1_5_2 AS VARRAY(50) OF VARCHAR2(200);
CREATE TYPE E_1_5 UNDER xmlAncestor_1 (
            A_1 VARCHAR2(200),
            A_2 E_1_5_2 ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 5, 'vztahy', 'o', 'E', 'E_1_5');
INSERT INTO xmlElemAttrTable
   VALUES (1, 5, 6, 1);
INSERT INTO xmlElemAttrTable
   VALUES (1, 5, 7, 2);

CREATE TYPE P_1_12 UNDER xmlAncestor_1 (
            C_1 REF xmlAncestor_1 ) INSTANTIABLE NOT FINAL;
INSERT INTO xmlElemTable
   VALUES (1, 12, null, 'C', 'C', 'P_1_12');
INSERT INTO xmlElemElemTable
   VALUES (1, 12, 1, 1, 'R', 1, 1);
INSERT INTO xmlElemElemTable
   VALUES (1, 12, 2, 1, 'R', 1, 100);
INSERT INTO xmlElemElemTable
   VALUES (1, 12, 3, 1, 'R', 1, 1);
INSERT INTO xmlElemElemTable
   VALUES (1, 12, 9, 1, 'R', 0, 1);
INSERT INTO xmlElemElemTable
   VALUES (1, 12, 10, 1, 'R', 1, 1);
INSERT INTO xmlElemElemTable
   VALUES (1, 12, 7, 1, 'R', 0, 100);
INSERT INTO xmlElemElemTable
   VALUES (1, 12, 4, 1, 'R', 0, 100);
INSERT INTO xmlElemElemTable
   VALUES (1, 12, 5, 1, 'R', 0, 1);
UPDATE xmlSchemaTable SET rootElemID = 12
   WHERE schemaID = 1;

CREATE TABLE T_E_1_1  OF E_1_1;
CREATE TABLE T_E_1_7  OF E_1_7;
CREATE TABLE T_E_1_2  OF E_1_2;
CREATE TABLE T_E_1_9  OF E_1_9;
CREATE TABLE T_E_1_10 OF E_1_10;
CREATE TABLE T_E_1_3  OF E_1_3;
```

```
CREATE TABLE T_E_1_4  OF E_1_4;
CREATE TABLE T_E_1_5  OF E_1_5;
CREATE TABLE T_P_1_12 OF P_1_12;

ALTER TABLE T_E_1_1 ADD (CHECK (C_1 IS NOT NULL));
ALTER TABLE T_E_1_2 ADD (CHECK (A_1 IS NOT NULL));
ALTER TABLE T_E_1_2 ADD (CHECK (A_3 IN ('ano', 'ne')));
ALTER TABLE T_E_1_2 ADD (SCOPE FOR (C_5.C_1) IS T_E_1_3);
ALTER TABLE T_E_1_2 ADD (SCOPE FOR (C_5.C_4) IS T_E_1_5);
ALTER TABLE T_E_1_2 ADD (CHECK (C_5 IS NOT NULL));
ALTER TABLE T_E_1_3 ADD (SCOPE FOR (C_1.C_1) IS T_E_1_9);
ALTER TABLE T_E_1_3 ADD (SCOPE FOR (C_1.C_2) IS T_E_1_10);
ALTER TABLE T_E_1_3 ADD (CHECK (C_1 IS NOT NULL));

INSERT INTO xmlPathTable VALUES (1, 1,  2, arrayOfIDs(6));
INSERT INTO xmlPathTable VALUES (1, 2,  3, arrayOfIDs(8));
INSERT INTO xmlPathTable VALUES (1, 3,  9, arrayOfIDs(11));
INSERT INTO xmlPathTable VALUES (1, 3, 10, arrayOfIDs(11));
INSERT INTO xmlPathTable VALUES (1, 2,  7, arrayOfIDs(8));
INSERT INTO xmlPathTable VALUES (1, 2,  4, arrayOfIDs(8));
INSERT INTO xmlPathTable VALUES (1, 2,  5, arrayOfIDs(8));
```

XML dokument

```
<zamestnanci xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:noNamespaceSchemaLocation='zamestnanci.xsd'>

 <osoba id="ved01" dovolena="ano" poznamka="Nemecky Cech">
  <jmeno>
   <krestni>Karel</krestni>
   <prijmeni>Nemec</prijmeni>
  </jmeno>
  <email>karel.nemec@cimr.com</email>
  <vztahy podrizeni="zam01 zam02 zam03 zam04 zam05"/>
 </osoba>

 <osoba id="zam01">
  <jmeno>
   <krestni>Vojtech</krestni>
   <prijmeni>Sofr</prijmeni>
  </jmeno>
  <email>vojtech.sofr@cimr.com</email>
  <vztahy vedouci="ved01"/>
 </osoba>
```

```
  <osoba id="zam02">
   <jmeno>
    <krestni>Vaclav</krestni>
    <prijmeni>Poustka</prijmeni>
   </jmeno>
   <email>vaclav.poustka@cimr.com</email>
   <vztahy vedouci="ved01"/>
  </osoba>

  <osoba id="zam03" poznamka="Ja bych sned i Kratochvila.">
   <jmeno>
    <krestni>Varel</krestni>
    <prijmeni>Fristensky</prijmeni>
   </jmeno>
   <email>varel.fristensky@cimr.com</email>
   <vztahy vedouci="ved01"/>
  </osoba>

  <osoba id="zam04" poznamka="Americky Cech">
   <jmeno>
    <krestni>George</krestni>
    <prijmeni>Beran</prijmeni>
   </jmeno>
   <email>george.beran@cimr.com</email>
   <vztahy vedouci="ved01"/>
  </osoba>

  <osoba id="zam05">
   <jmeno>
    <prijmeni>Mac Donald</prijmeni>
   </jmeno>
   <email>macdonald@cimr.com</email>
   <vztahy vedouci="ved01"/>
  </osoba>
</zamestnanci>
```

Naplnění objektově relačního schématu

```
INSERT INTO T_E_1_9
   VALUES (E_1_9(9, 1, 'Karel'));
INSERT INTO T_E_1_10
   VALUES (E_1_10(10, 2, 'Nemec'));
INSERT INTO T_E_1_3
   VALUES (E_1_3(3, 4, P_1_11(11, 3,
      (SELECT REF(T) FROM T_E_1_9 T WHERE recordID = 1), 1,
      (SELECT REF(T) FROM T_E_1_10 T WHERE recordID = 2), 2)));
INSERT INTO T_E_1_7
   VALUES (E_1_7(7, 5, 'karel.nemec@cimr.com'));
```

```
INSERT INTO T_E_1_5
   VALUES (E_1_5(5, 6, NULL,
      E_1_5_2('zam01', 'zam02', 'zam03', 'zam04', 'zam05')));
INSERT INTO T_E_1_2
   VALUES (E_1_2(2, 8, 'ved01', 'Nemecky Cech', 'ano', NULL,
      P_1_8(8, 7,
      (SELECT REF(T) FROM T_E_1_3 T WHERE recordID = 4),
      P_1_8_2((SELECT REF(T) FROM T_E_1_7 T WHERE recordID = 5)),
      NULL, (SELECT REF(T) FROM T_E_1_5 T WHERE recordID = 6))));
INSERT INTO T_E_1_9
   VALUES (E_1_9(9, 9, 'Vojtech'));
INSERT INTO T_E_1_10
   VALUES (E_1_10(10, 10, 'Sofr'));
INSERT INTO T_E_1_3
   VALUES (E_1_3(3, 12, P_1_11(11, 11,
      (SELECT REF(T) FROM T_E_1_9 T WHERE recordID = 9), 1,
      (SELECT REF(T) FROM T_E_1_10 T WHERE recordID = 10), 2)));
INSERT INTO T_E_1_7
   VALUES (E_1_7(7, 13, 'vojtech.sofr@cimr.com'));
INSERT INTO T_E_1_5
   VALUES (E_1_5(5, 14, 'ved01', NULL));
INSERT INTO T_E_1_2
   VALUES (E_1_2(2, 16, 'zam01', NULL, 'ne', NULL,
      P_1_8(8, 15,
      (SELECT REF(T) FROM T_E_1_3 T WHERE recordID = 12),
      P_1_8_2((SELECT REF(T) FROM T_E_1_7 T WHERE recordID = 13)),
      NULL, (SELECT REF(T) FROM T_E_1_5 T WHERE recordID = 14))));
INSERT INTO T_E_1_9
   VALUES (E_1_9(9, 17, 'Vaclav'));
INSERT INTO T_E_1_10
   VALUES (E_1_10(10, 18, 'Poustka'));
INSERT INTO T_E_1_3
   VALUES (E_1_3(3, 20, P_1_11(11, 19,
      (SELECT REF(T) FROM T_E_1_9 T WHERE recordID = 17), 1,
      (SELECT REF(T) FROM T_E_1_10 T WHERE recordID = 18), 2)));
INSERT INTO T_E_1_7
   VALUES (E_1_7(7, 21, 'vaclav.poustka@cimr.com'));
INSERT INTO T_E_1_5
   VALUES (E_1_5(5, 22, 'ved01', NULL));
INSERT INTO T_E_1_2
   VALUES (E_1_2(2, 24, 'zam02', NULL, 'ne', NULL,
      P_1_8(8, 23,
      (SELECT REF(T) FROM T_E_1_3 T WHERE recordID = 20),
      P_1_8_2((SELECT REF(T) FROM T_E_1_7 T WHERE recordID = 21)),
      NULL, (SELECT REF(T) FROM T_E_1_5 T WHERE recordID = 22))));
INSERT INTO T_E_1_9
   VALUES (E_1_9(9, 25, 'Varel'));
```

```
INSERT INTO T_E_1_10
   VALUES (E_1_10(10, 26, 'Fristensky'));
INSERT INTO T_E_1_3
   VALUES (E_1_3(3, 28, P_1_11(11, 27,
      (SELECT REF(T) FROM T_E_1_9 T WHERE recordID = 25), 1,
      (SELECT REF(T) FROM T_E_1_10 T WHERE recordID = 26), 2)));
INSERT INTO T_E_1_7
   VALUES (E_1_7(7, 29, 'varel.fristensky@cimr.com'));
INSERT INTO T_E_1_5
   VALUES (E_1_5(5, 30, 'ved01', NULL));
INSERT INTO T_E_1_2
   VALUES (E_1_2(2, 32, 'zam03', 'Ja bych sned i Kratochvila.',
      'ne', NULL,
      P_1_8(8, 31,
      (SELECT REF(T) FROM T_E_1_3 T WHERE recordID = 28),
      P_1_8_2((SELECT REF(T) FROM T_E_1_7 T WHERE recordID = 29)),
      NULL, (SELECT REF(T) FROM T_E_1_5 T WHERE recordID = 30))));
INSERT INTO T_E_1_9
   VALUES (E_1_9(9, 33, 'George'));
INSERT INTO T_E_1_10
   VALUES (E_1_10(10, 34, 'Beran'));
INSERT INTO T_E_1_3
   VALUES (E_1_3(3, 36, P_1_11(11, 35,
      (SELECT REF(T) FROM T_E_1_9 T WHERE recordID = 33), 1,
      (SELECT REF(T) FROM T_E_1_10 T WHERE recordID = 34), 2)));
INSERT INTO T_E_1_7
   VALUES (E_1_7(7, 37, 'george.beran@cimr.com'));
INSERT INTO T_E_1_5
   VALUES (E_1_5(5, 38, 'ved01', NULL));
INSERT INTO T_E_1_2
   VALUES (E_1_2(2, 40, 'zam04', 'Americky Cech', 'ne', NULL,
      P_1_8(8, 39,
      (SELECT REF(T) FROM T_E_1_3 T WHERE recordID = 36),
      P_1_8_2((SELECT REF(T) FROM T_E_1_7 T WHERE recordID = 37)),
      NULL, (SELECT REF(T) FROM T_E_1_5 T WHERE recordID = 38))));
INSERT INTO T_E_1_10
   VALUES (E_1_10(10, 41, 'Mac Donald'));
INSERT INTO T_E_1_3
   VALUES (E_1_3(3, 43, P_1_11(11, 42, NULL, 0,
      (SELECT REF(T) FROM T_E_1_10 T WHERE recordID = 41), 1)));
INSERT INTO T_E_1_7
   VALUES (E_1_7(7, 44, 'macdonald@cimr.com'));
INSERT INTO T_E_1_5
   VALUES (E_1_5(5, 45, 'ved01', NULL));
```

```
INSERT INTO T_E_1_2
   VALUES (E_1_2(2, 47, 'zam05', NULL, 'ne', NULL, P_1_8(8, 46,
      (SELECT REF(T) FROM T_E_1_3 T WHERE recordID = 43),
      P_1_8_2((SELECT REF(T) FROM T_E_1_7 T WHERE recordID = 44)),
      NULL, (SELECT REF(T) FROM T_E_1_5 T WHERE recordID = 45))));
INSERT INTO T_E_1_1
   VALUES (E_1_1(1, 49, P_1_6(6, 48, P_1_6_1(
      (SELECT REF(T) FROM T_E_1_2 T WHERE recordID = 8),
      (SELECT REF(T) FROM T_E_1_2 T WHERE recordID = 16),
      (SELECT REF(T) FROM T_E_1_2 T WHERE recordID = 24),
      (SELECT REF(T) FROM T_E_1_2 T WHERE recordID = 32),
      (SELECT REF(T) FROM T_E_1_2 T WHERE recordID = 40),
      (SELECT REF(T) FROM T_E_1_2 T WHERE recordID = 47)))));
INSERT INTO T_P_1_12
   VALUES (P_1_12(12, 50,
          (SELECT REF(T) FROM T_E_1_1 T WHERE recordID = 49)));
INSERT INTO xmlDocTable
   VALUES (1, 'zamestnanci.xml', 50);
```

Dotazy nad objektově relačním schématem

```
/zamestnanci/osoba/jmeno
```

```
SELECT DEREF(A3.COLUMN_VALUE).C_5.C_1
FROM (SELECT TREAT(DEREF(A1.C_1) AS E_1_1).C_1 AS temp_col
      FROM T_P_1_12 A1
      WHERE A1.recordID = 50 AND
            DEREF(A1.C_1) IS OF (E_1_1) AND
            DEREF(A1.C_1).elemID = 1) A2,
     TABLE(A2.temp_col.C_1) A3;
```

```
<RESULT>
 <jmeno>
  <krestni>Karel</krestni>
  <prijmeni>Nemec</prijmeni>
 </jmeno>
 <jmeno>
  <krestni>Vojtech</krestni>
  <prijmeni>Sofr</prijmeni>
 </jmeno>
 <jmeno>
  <krestni>Vaclav</krestni>
  <prijmeni>Poustka</prijmeni>
 </jmeno>
 <jmeno>
  <krestni>Varel</krestni>
  <prijmeni>Fristensky</prijmeni>
 </jmeno>
```

```
 <jmeno>
  <krestni>George</krestni>
  <prijmeni>Beran</prijmeni>
 </jmeno>
 <jmeno>
  <prijmeni>Mac Donald</prijmeni>
 </jmeno>
</RESULT>
```

```
/zamestnanci/osoba[@poznamka]/email
```

```
SELECT A5.COLUMN_VALUE
FROM (SELECT DEREF(A3.COLUMN_VALUE).C_5 AS temp_col
      FROM (SELECT TREAT(DEREF(A1.C_1) AS E_1_1).C_1 AS temp_col
            FROM T_P_1_12 A1
            WHERE A1.recordID = 50 AND
                  DEREF(A1.C_1) IS OF (E_1_1) AND
                  DEREF(A1.C_1).elemID = 1) A2,
            TABLE(A2.temp_col.C_1) A3
      WHERE DEREF(A3.COLUMN_VALUE).A_2 IS NOT NULL) A4,
     TABLE(A4.temp_col.C_2) A5;
```

```
<RESULT>
 <email>karel.nemec@cimr.com</email>
 <email>varel.fristensky@cimr.com</email>
 <email>george.beran@cimr.com</email>
</RESULT>
```