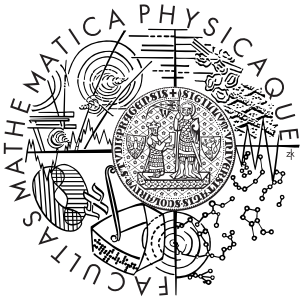
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Petr Uher

Ukládání XML dat popsaných modelem XSEM v databázi

Katedra softwarového inženýrství

Vedoucí diplomové práce: Mgr. Martin Nečaský
Studijní program: Informatika, Softwarové systémy

2007

Na tomto místě bych rád poděkoval vedoucímu své diplomové práce Mgr. Martinu Nečaskému, jehož rady a připomínky výrazně přispěly k úspěšnému dokončení této práce. Dále bych chtěl poděkovat rodičům za stálou podporu během celého studia.

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

V Praze dne 6. prosince 2007 Petr Uher

Obsah

Název práce: Ukládání XML dat popsaných modelem XSEM v databázi
Autor: Petr Uher
Katedra: Katedra softwarového inženýrství
Vedoucí diplomové práce: Mgr. Martin Nečaský
E-mail vedoucího: martin.necasky@mff.cuni.cz

Abstrakt: Jazyk XML je v současné době standardem pro výměnu dat mezi informačními systémy. Proto vzrůstají nároky na jeho popis jak na logické, tak i konceptuální úrovni. Jedním z nových konceptuálních modelů pro XML data je model XSEM. Tato práce z XSEM modelu vychází a snaží se navrhnout pravidla pro tvorbu logického schématu na základě konceptuálního schématu navrženého v XSEM. Logické schéma je vytvářeno nad objektově-relační databází a snaží se maximálně využít podpory nativního uložení XML dat, které moderní databázové systémy nabízí. Pravidla pro převod vycházejí z pravidel pro převod klasického ER konceptuálního schématu do logického databázového schématu. Dále je popsán algoritmus, který na základě navrženého logického schématu vygeneruje dotazy konstruující XML dokumenty validní vůči schématům pohledů namodelovaných v XSEM.
Klíčová slova: XML, logické schéma, XSEM

Title: Storing XML data described by XSEM in a database
Author: Petr Uher
Department: Department of Software Engineering
Supervisor: Mgr. Martin Nečaský
Supervisor's e-mail address: martin.necasky@mff.cuni.cz

Abstract: Recently, XML is a standard language used for exchanging data among information systems. Therefore, it is necessary to exactly describe it both on a conceptual and also logical level. One of the conceptual models is XSEM model. This work is based on XSEM and it tries to specify rules for creation of a logical schema from a conceptual schema designed in XSEM. The logical schema is designed over object-relational database and it is trying to use a native storage of the XML documents. The rules for transformation are based on the rules for transformation of classical ER conceptual schema to a logical database schema. In this thesis, there is also described an algorithm for generating queries that reconstruct XML documents valid against schemas modeled in XSEM.
Keywords: XML, logical schema, XSEM

Kapitola 1

Úvod

V poslední době se formát XML stává standardem pro výměnu dat mezi různými informačními systémy. Je to především díky jeho jednoduchosti, univerzálnosti a platformové nezávislosti. S jeho rostoucí oblibou vzrůstají také nároky na jeho přesný popis jak na logické, tak i na konceptuální úrovni. Na logické úrovni existuje řada XML schéma jazyků, např. XML Schema, Relax NG nebo Schematron. Pro návrh dat na konceptuální úrovni existuje také řada modelů, žádný se však zatím výrazně neprosadil. Tyto modely lze rozdělit do dvou skupin.

V první skupině jsou modely, které vycházejí z klasického ER modelu. Dokáží se pomocí speciálních konstrukcí vyrovnat s některými specifiky XML jako nepravidelná struktura, smíšený obsah nebo implicitní uspořádání. Problémem pro ně je modelování hierarchické struktury.

Druhou skupinu tvoří modely, které jsou založené na hierarchické struktuře. Konceptuální schémata jsou představována stromy, případně lesy. Tyto modely jsou vhodnější spíše pro návrh podoby XML dat než pro návrh sémantiky samotných dat.

Proto Martin Nečaský navrhl nový konceptuální model XSEM [4] pro modelování XML dat. Ten spojuje výhody obou přístupů k modelování XML dat. Využívá rozšířený ER model pro návrh sémantiky dat a k tomu připojuje možnost namodelování hierarchických struktur, ve kterých by data měla být prezentována. Část XSEM modelu určená pro návrh sémantiky dat se nazývá XSEM-ER model, část pro návrh hierarchické struktury dat XSEM-H.

Stejně jako u klasického ER modelu přichází po konceptuálním návrhu převod návrhu do logického modelu. U XSEM bude mít logický model stejně jako konceptuální dvě části. První bude logické databázové schéma, které bude popisovat způsob uložení dat v databázi. Návrh tohoto schématu bude hlavním cílem této práce. Druhou částí logického modelu bude XML model popisující jednotlivé hierarchické struktury. Tuto část zpracovává Martin Nečaský a bude ji v blízké době publikovat.

1.1 Příklad využití XSEM modelu

Pro lepší názornost uvedu příklad možného nasazení XSEM modelu v praxi.

Privátní nemocnice chce vybudovat vlastní informační systém. Veškerá data by byla uložena na centrálním datovém serveru. Komunikace mezi serverem a jednotlivými uživatelskými stanicemi by probíhala online ve formátu XML. Ukládala

by se veškerá data o návštěvách pacientů, předepsaných lécích, vyšetřeních, provedených zákrocích apod. Úkolem je navrhnout vhodné uložení dat a XML pohledy, které zpřístupní data z centrálního serveru jednotlivým uživatelům systému.

Pokud se návrhář rozhodne využít XSEM model, budou ho čekat následující kroky:

1. Podle požadavků navrhne konceptuální schéma pro data, které bude systém spravovat. K tomu využije XSEM-ER model.

2. Navrhne množinu pohledů, které budou zpřístupňovat uložená data ve formátu XML uživatelům systému. K tomu poslouží XSEM-H model, ve kterém se navrhnou jednotlivé XSEM-H diagramy[1].

3. Pro konceptuální schéma pro data navrhne logické databázové schéma. Pravidla pro návrh takového schématu se bude snažit specifikovat tato diplomová práce.

4. Pro XSEM-H diagramy navrhne logická XML schémata, která budou specifikovat formát XML dokumentů, jež budou zpřístupněné uživatelům.

5. Napíše dotazy, které vygenerují XML dokumenty validní vůči schématům navrhnutým v předchozím bodě. Popis tvorby dotazů v jazyce XQuery z navrhnutého logického schématu databáze bude také součástí této práce.

[1] V textu bude pro XSEM-H diagramy někdy také používán pojem pohled, případně XSEM-H pohled.

Kapitola 2

Cíle práce a její struktura

Úkolem práce je najít nejvhodnější logické databázové schéma popisující uložení dat popsaných schématem vytvořeným v modelu XSEM v databázi. Budou zkoumány možnosti využití objektově-relačního modelu společně s nativním uložení XML dat. Zároveň pro navržené logické schéma bude popsán algoritmus, který bude schopen zkonstruovat XML dokumenty validní vůči navrženým pohledům. Úkolem nebude vytvořit schéma co nejoptimálnější vůči navrženým pohledům ale schéma, které bude snadno rozšiřitelné bez výraznějších zásahů do něj. Tedy aby efektivita generování XML dokumentů odpovídajících pohledům byla stejná, jak pro pohledy známe při vzniku schématu, tak i pro pohledy vytvořené později. Tato vlastnost by měla zvýšit využitelnost schématu v praxi, protože většina systému prochází v průběhu svého užívání určitým vývojem.

Jelikož zatím neexistuje žádný formální zápis XSEM-ER a XSEM-H digramů do serializované podoby, bude navrženo jednoduché XML schéma v jazyce XML Schema [6], které tento zápis umožní.

Pro účely práce nebude využit celý model XSEM, ale pouze jeho podmnožina. Tato podmnožina by měla dostatečně pokrývat sémantiku tohoto modelu. Vynechány budou pomocné modelovací konstrukce u XSEM-H diagramů, které prozatím nebyly publikovány. Dále do implementace nebudou zařazené složené a vícehodnotové atributy, bude ale prodiskutováno jejich možné zpracování. Taktéž nebude pracováno s ISA-hierarchií.

Práce nebude popisovat uložení dat do navrženého logického schématu. Bude předpokládáno, že veškerá data jsou již v systému uložena a splňují všechna integritní omezení specifikovaná v logickém schématu databáze. Zároveň navržené schéma nebude umožňovat přesnou rekonstrukci XML dokumentů, ve kterých do systému budou data přicházet, tzv. round tripping [1].

Zkonstruované dotazy budou vypisovat kompletní obsahy tabulek. Nebudou nijak omezeny pomocí selekce. Selekci je do nich nutné přidat ručně. Možnosti přidání selekce budou popsány.

2.1 Struktura práce

Práce je členěna do následujících kapitol.

První kapitola představuje úvod do problematiky konceptuálního modelování XML dat. Jsou popsány současné možnosti na tomto poli a na příkladu je ukázána možnost využití modelu XSEM.

Ve druhé kapitole je přesně vymezen cíl této práce a popsána její struktura.

Třetí kapitola obsahuje analýzu celého problému. Je diskutována (ne)vhodnost použití již existujících řešení a navržen základní koncept řešení. Na základě toho jsou vybrány vhodné technologie a systémy pro implementaci.

Ve čtvrté kapitole je ve stručnosti představen XSEM model a na příkladech vysvětleny jeho základní konstrukce. Dále je navrženo schéma pro serializaci XSEM diagramů.

V páté kapitole jsou podrobně popsána pravidla pro návrh logického schématu pro čistě relační databázi a popsány jeho výhody a nevýhody.

Šestá kapitola obsahuje popis tvorby logického schématu, které využívá nativní uložení XML dat. Detailně je rozebrána konstrukce dotazů pro namodelované XSEM pohledy.

V sedmé kapitole je stručně popsána prototypová implementace a způsob jejího ovládání.

V osmé (poslední kapitole) je shrnuto naplnění vytyčených cílů, zhodnoceny výhody a možné slabiny řešení a navrženy oblasti dalšího možného zkoumání.

Kapitola 3

Přehled možností uložení XML dat a základní návrh řešení

Logické databázové schéma, které je nutné navrhnout, musí splňovat několik podmínek: 1. musí být dostatečně efektivní v ukládání dat, tj. minimální, nejlépe žádná redundance dat, 2. musí být schopné uložit i specifika formátu XML jako smíšený obsah, 3. musí dobře dodržovat referenční integritu, která u XML není založena jen na klíčích (implicitní uspořádání), 4. dotazy pro pohledy by měly být efektivní. Zejména 4. bod je důležitý a bude nejvíce viditelný navenek. Většina běžných uživatelů totiž bude posuzovat kvalitu systému podle doby odezvy na dotaz.

V současnosti existuje řada přístupů k uložení XML dat do databáze, přehled v [3]. Většina z nich se snaží pro dokumenty, které přicházejí do systému, vytvořit nějaké logické databázové schéma. To je ale odlišný přístup, než je zvolen v této práci. Zde se logické schéma databáze tvoří ne s ohledem na formát vstupních dokumentů, ale s ohledem na XSEM-ER model. Pro tvorbu logického schématu se tedy používá konceptuální schéma dat a ne logické schéma příchozích XML dat. Přesto zde budou některé metody uložení XML dat do databáze představeny.

3.1 Přehled existujících možností uložení XML dat

Všechny zde představené metody v nějaké podobě využívají toho, že data do systému přicházejí v XML podobě.

3.1.1 LOB

Nejjednodušší možností, jak uložit XML data do databáze, je uložení celého dokumentu v tabulce ve sloupci datového typu `CLOB` nebo `BLOB`. Tento způsob sice umožňuje zpětnou rekonstrukci příchozích dat (tzv. round tripping), ale jinak není příliš vhodný. Nevýhodou je velká redundance dat, malá provázanost uložených dokumentů mezi sebou a velmi omezené možnosti dotazování na data. Tedy i obtížné konstrukce XML dokumentů jiné struktury, než v jaké jsou uložené.

3.1.2 Nativní XML databáze

Nativní XML databáze je speciální druh databáze vytvořené pro ukládání a manipulaci s XML daty. Většinou poskytuje výkonné nástroje pro dotazování a manipulaci s daty jako XQuery, DOM a SAX. Velkou nevýhodou je velikost uložených dat, která může být i několikrát větší než velikost příchozích dat. Problémem také je konstrukce XML dokumentů jiné struktury než v jaké jsou data uložena.

3.1.3 (Objektově-)relační databáze s podporou XML

(Objektově-)relační databáze s podporou XML je databáze, která umožňuje ukládání XML dat v nativní podobě ve sloupci relační tabulky. Obvykle se XML data ukládají do datového typu `XML`. Jsou poskytnuty nástroje pro práci s XML daty, jejich možnosti ale nejsou tak velké jako u speciálních nativních XML databázích. Většinou jsou omezeny na dotazovací jazyk XQuery. Výhodou je možnost uložení jak nestrukturovaných dat, tak i XML v jedné tabulce. Oproti uložení v LOBu také poskytuje větší možnosti pro práci s XML daty.

3.1.4 Mapování XML dat do objektově-relačních databází

Předcházející tři techniky ukládaly XML dokumenty ve formátu XML. Ať už v podobě v jaké do systému přišly nebo nějak upravené. Existují však metody, které data z XML dokumentů ukládají do objektově-relačních tabulek. Tyto metody také mohou být využity v nativních databázích, které pro ukládání dat využívají objektově-relační technologie (fyzický model) a jejich nativita spočívá v zapouzdření a zpřístupnění těchto dat v nativní podobě (logický model), ne v podobě tabulek a sloupců. Základní rozdělení mapovacích metod je následující:

Generické metody — mapování je nezávislé na schématu XML dokumentů. Snaží se vytvořit obecné (objektově-) relační logické databázové schéma, ve kterém by mohla být uložena data XML dokumentu bez ohledu na jejich strukturu.

Schématem řízené — mapování je založené na využití informací ze schématu XML dokumentů. Schéma obvykle vytvořené v DTD nebo XML Schema. Logické schéma by mělo co nejvíce odpovídat schématu dokumentu, mělo by být tvořené „rozumným“ počtem tabulek. Velmi vhodné je pro toto mapování použít objektové prvky databázového systému.

Uživatelsky definované — mapování je definované uživatelem. Je navrženo logické databázové schéma a uživatel sám navrhne mapování XML dokumentů do tohoto schématu.

3.2 Možnosti využití existujících metod uložení XML dat

Jelikož je velká pravděpodobnost, že data do systému budou přicházet v XML podobě, nabízí se možnost ukládat data ve formátu, v jakém přišla. Z již existujících řešení je pro toto nejvhodnější nativní XML databáze představená v 3.1.2. Toto

řešení však není pro uložení dat popsaných XSEM modelem příliš vhodné. Sice výborně ukládá specifika XML dat jako je smíšený obsah a implicitní uspořádání, ale velkým problémem je špatná provázanost dokumentů a velká redundance dat. Největší nevýhodou je obtížné konstruování XML dokumentů s jinou strukturou, než je struktura XML dokumentů uložených v databázi. Tedy konstrukce XML dokumentů validních vůči schématům XML pohledů by byla velice náročná.

Další možností využití nativní databáze je ukládání dat přímo do XML dokumentů, které odpovídají namodelovaným pohledům. Obrovskou výhodou je naprosto triviální dotazování a tedy i rychlost odezvy systému na dotaz. To je ale velmi drasticky kompenzováno nároky na systém při ukládání nebo aktualizaci dat. Např. jedna změna dat by mohla zapříčinit nutnost přegenerování poloviny XML dokumentů v databázi. Opět je zde obrovská redundance dat. Velmi špatná je také rozšiřitelnost systému o nové pohledy. Pro přidání nového pohledu je totiž nutné nejen vygenerovat ze stávajících dat nové XML dokumenty odpovídající novému pohledu, ale je také nutné upravit všechny procedury zajišťující vkládání a aktualizaci dat.

Uložení do datového typu LOB je již dávno překonané. Většina kvalitní databázových systémů namísto LOBu nabízí uložení XML dat do sloupce v nativní podobě. Podporu nativního uložení XML dat v (objektově-) relačních databázích lze využít pro uložení jak kompletních XML dokumentů přicházejících do systému, tak i XML dokumentů odpovídajících XSEM pohledům. Výhody a nevýhody jsou naprosto stejné jako u nativních databází. Tento způsob však není pro XSEM model vhodný. Mnohem výhodnější se zdá využití velmi kvalitní implementace, kterou poskytuje (objektově-) relační databáze společně s některou metodou mapovaní XML dat do ní. Případně je také možné využít pro mapování podporu nativního uložení XML.

Jak již bylo řečeno, mapovací metody využívají podobu ukládaných XML dokumentů. Tu však u konceptuálního schématu vzniklého pomocí XSEM modelu neznáme. Nelze tedy použít přímo některou z metod. Lze se však nechat inspirovat především některou ze schématem řízených metod. Logické schéma ukládaných dokumentů v takové případě nahradíme konceptuálním schématem dat. Generické metody jsou naprosto nevyužitelné. Jsou příliš obecné a velmi málo (nebo vůbec) využívají znalost struktury dat. Tuto znalost nám v omezené míře XSEM model poskytuje. Uživatelsky definované metody jsou pro použití s XSEM modelem vhodné, ale ne pro návrh logického schématu databáze. Své využití najdou v následném mapování XML dat do navrženého schématu. Tím se ale tato práce nezabývá.

Základní myšlenkou převzatou z metod z kategorie schématem řízených je mapování elementů (v XSEM modelu jsou to entity a relace) do relačních tabulek a uložení semistrukturovaných částí (v XSEM modelu jsou to klastry) v nativní podobě, tj. v datovém typu XML.

3.3 Základní návrh řešení

V předcházející kapitole byly představeny existující způsoby uložení XML dat. Jak bylo uvedeno, pro návrh logického databázového schématu z XSEM modelu jsou velmi těžko použitelné. Je to dáno především jejich povahou, kdy vůbec nenavazují na žádný konceptuální model XML dat. Proto bude lepší vzít klasický

ER model, ze kterého XSEM-ER model vychází. K němu existuje dobře prověřený algoritmus převodu ER konceptuálního schématu do logického schématu a patřičně tento algoritmus upravit, aby pokryl všechny rozšíření přidané XSEM-ER modelem.

Základní myšlenka byla zkusit namodelovat všechny přidané konstrukty pomocí čistě relačních tabulek. Případně s využitím objektových prvků databáze. Způsob takového uložení je podrobně rozebrán v kapitole 5. Problémem této metody bylo velké množství tabulek nutných pro uložení semistrukturovaných dat. Je to dáno jistou nepřirozeností, s jakou jsou ukládána data do relačních tabulek.

Druhé možné řešení opět vychází z algoritmu pro převod klasického konceptuálního ER schématu do logického databázového schématu. Pouze se pro uložení smíšeného obsahu použije nativního uložení XML. Podrobnosti jsou rozebrány později v kapitole 6.

Pro tvorbu XML dokumentů validních vůči namodelovaným pohledům byl zvolen s ohledem na použité technologie pro uložení dat jazyk XML Schema [6].

3.4 Výběr vhodného databázového systému

Vzhledem k navrhnutým řešením jsou požadavky na databázový systém jasné: objektově-relační systém s nativní podporou XML dat. Musí podporovat XQuery jazyk. Navíc je požadována jistá stabilita a robustnost systému. Již od počátku byl výběr zúžen na dva možné systémy: Oracle Database 10g Express Edition[1] a IBM DB2 Express-C v9[2]. Oba dva systémy nabízejí možnost číst v XQuery dotazu data z relačních tabulek. V DB2 je to pomocí funkce `db2-fn:sqlquery`. Ta umožňuje napsat jakýkoliv SQL/XML dotaz, jedinou podmínkou je, že vrácená data musí být ve formátu XML. Oracle poskytuje funkci `ora:view`, její možnosti jsou ale velmi omezené. Nejde jí jako argument předat SQL/XML dotaz, ale pouze název tabulky, kterou bude funkce načítat a převádět do XML.

S ohledem na větší variabilitu a možnosti použití funkce `db2-fn:sqlquery` byl nakonec vybrán databázový systém DB2 od společnosti IBM. Tato volba se později ukázala velmi vhodnou, kdy novější verze 9.5[3] přinesla zlepšenou podporu XML a hlavně ještě lepší provázanost jazyků XQuery a SQL.

3.5 Konfigurace testovacího počítače

Veškeré testy, jejichž výsledky jsou v práci uvedeny, byly provedeny na běžném notebooku.
Procesor: Intel Pentium M 740, 1733MHz, Cache L1 64KB, Cache L2 2048KB
Paměť: DDR-SDRAM PC-2700 (166 MHz) - [DDR-333] 1280MB
Pevný disk: Fujitsu MHT2060AH PL 60GB, 5400rpm, 8MB buffer
Souborový systém: NTFS
Operační systém: Microsoft Windows XP Professional 5.01.2600 Service Pack 2

[1] `http://www.oracle.com/global/cz/database/express_edition.html`
[2] `http://www-306.ibm.com/software/data/db2/express/`
[3] Verze 9.5 byla vydána 31.10.2007

3.6 Porovnání možností uložení dat do jednoduchých SQL datových typů a datového typu XML

Obě navržená řešení počítají s uložením dat do klasických relačních tabulek. Naopak nutnost konstrukce dotazů pro XSEM-H diagramy vyžaduje použití jazyka XQuery, který data potřebuje ve formátu XML. Zde se střetávají dvě reprezentace dat. Fyzicky by data byla uložena v relačních tabulkách a pro potřeby XQuery by se data musela převádět do XML formátu. Vyvstává tedy otázka, zda by nebylo výhodnější uložit data přímo v XML. Ne jako celé dokumenty, ale rozdělené na malé dobře formované dokumenty, kdy by každý takový dokument odpovídal právě jedné instanci entity/relace. Vlastně by tak odpovídal jednomu řádku relační tabulky. Kořenový element by měl název entity/relace, jeho potomci by byly elementy s názvy atributů (názvy sloupců tabulky).

Bylo otestováno několik možností uložení dat a jejich následný převod do XML, který je pro tvorbu XQuery dotazů nezbytný. Navíc byl přidán požadavek snadné indexovatelnosti dat, pro zvýšení rychlosti dotazů. Ukládá se entita *Osoba*, která má atributy *jméno*, *datum narození* a *pohlaví*. Navíc je přidán umělý identifikátor *id*.

Jednotlivé způsoby uložení dat:

1. Data jsou uložena v relační tabulce v XML podobě v datovém typu `XML`. Navíc je přidán sloupec s id entity. Toto id je uložené i uvnitř XML. Na data se dotazuje pomocí XQuery funkce `db2-fn:sqlquery`.

   Příklad ukazující strukturu ukládaného XML:

   ```
   <osoba>
     <id_osoba>2345</id_osoba>
     <jmeno>Pepa Novák</jmeno>
     <datum_narozeni>29.9.2000</datum_narozeni>
     <pohlavi>M</pohlavi>
   </osoba>
   ```

2. Data jsou uložena v relační tabulce v XML podobě v datovém typu `XML`. Oproti předchozí možnosti chybí sloupec s id entity. Je vytvořen index na id entity, které je uložené v XML. Dotazuje se pomocí XQuery funkce `db2-fn:xmlcolumn`.

3. Data uložena klasicky relačně ve sloupcích s jednoduchými datovými typy. Dotazuje se pomocí XQuery funkce `db2-fn:sqlquery`, v níž se každý řádek převádí pomocí funkce `XMLELEMENT`.

4. Data uložena klasicky relačně ve sloupcích s jednoduchými datovými typy. Dotazuje se pomocí XQuery funkce `db2-fn:sqlquery`, v níž se každý řádek převádí pomocí `XMLROW` funkce. Tato funkce byla do DB2 přidána od verze 9.5.

Následují schémata tabulek jednotlivých řešení. Tabulka `TEST_OS3` bude použita i pro převod pomocí funkce `XMLROW`.

```
CREATE TABLE "TEST_OS1" (
      "ID_OSOBA" INTEGER NOT NULL PRIMARY KEY,
      "TEXT" XML );

CREATE TABLE "TEST_OS2" (
      "TEXT" XML );

CREATE INDEX "OS_INDEX" ON "TEST_OS2"
    ("TEXT" ASC)
    GENERATE KEY USING XMLPATTERN '/osoba/id_osoba/text()'
      AS SQL DOUBLE IGNORE INVALID VALUES ALLOW REVERSE SCANS;

CREATE TABLE "TEST_OS3" (
      "ID_OSOBA" INTEGER NOT NULL PRIMARY KEY,
      "JMENO" VARCHAR(50) ,
      "DATUM_NAROZENI" DATE ,
      "POHLAVI" CHAR(1) );
```

Test byl prováděn pomocí XQuery dotazů, které vytvořily pro každý řádek tabulky samostatný element a do něj umístily jako elementy atributy entity. Vždy byly vypsány pouze osoby s id < 10000.

```
xquery
let $n := 10000
for $osoba in db2-fn:sqlquery('
select text from test_os1 where id_osoba < parameter(1)',
$n)
return (
<osoba>
{
  $osoba/jmeno,
  $osoba/datum_narozeni,
  $osoba/pohlavi
}
</osoba>
)

xquery
let $n := 10000
for $osoba in db2-fn:xmlcolumn('TEST_OS2.TEXT')/osoba[id_osoba/text() <
     $n]
return (
<osoba>
{
  $osoba/jmeno,
  $osoba/datum_narozeni,
  $osoba/pohlavi
}
</osoba>
)

xquery
let $n := 10000
```

```
for $osoba in db2-fn:sqlquery('
select xmlelement(name "osoba",
    xmlelement(name "id_osoba", o.id_osoba),
    xmlelement(name "jmeno", o.jmeno),
    xmlelement(name "datum_narozeni", o.datum_narozeni),
    xmlelement(name "pohlavi", o.pohlavi)
     )
from test_os3 o where o.id_osoba < parameter(1)',
$n)
return (
<osoba>
{
  $osoba/jmeno,
  $osoba/datum_narozeni,
  $osoba/pohlavi
}
</osoba>
)

xquery
let $n := 10000
for $osoba in db2-fn:sqlquery('
select xmlrow(id_osoba, jmeno, datum_narozeni, pohlavi OPTION ROW "
    osoba")
from test_os3 o where o.id_osoba < parameter(1)',
$n)
return (
<osoba>
{
  $osoba/jmeno,
  $osoba/datum_narozeni,
  $osoba/pohlavi
}
</osoba>
)
```

Doby trvání jednotlivých dotazů jsou v tabulce 3.1. Každý dotaz byl proveden 3krát.

Tabulka 3.1: Rychlost dotazů

	1. dotaz	**2. dotaz**	**3. dotaz**	**4. dotaz**
1	7860 ms	8625 ms	7375 ms	8500 ms
2	6750 ms	7953 ms	7344 ms	8610 ms
3	6750 ms	7859 ms	7390 ms	8640 ms

Z výsledků vyplývá, že nejrychlejší je uložení v XML doplněné sloupcem s id. Nevýhodou tohoto uložení je obtížné dotazování při restrikci pomocí jiného atributu než id.

I uložení pouze do sloupce typu `XML` se zdá jako vhodná varianta, ale je zde nutné data dobře indexovat. Nativní úložiště XML dat jsou poměrně novou technologií, proto možnosti indexů na XML jsou omezené. Např. z numerických for-

mátů lze vytvořit pouze index datového typu `DOUBLE`, což vzhledem ke způsobu ukládání čísel s plovoucí desetinou čárkou není nejefektivnější, pokud je vytvářen index nad celočíselnými hodnotami. Skutečný výkon indexu nad XML daty by prověřilo jedině jeho dlouhodobé nasazení na vysoce exponovaných datech.

Zajímavý výsledek poskytlo uložení bez použití datového typu `XML`. Je patrné, že převod řádku do XML podoby není časově tolik náročný. Obrovskou výhodou tohoto uložení je jednoduchá selekce v dotazu za pomoci jakéhokoliv atributu. Pro také hovoří časem dobře prověřené a otestované indexy nad daty všech možných datových typů.

Nepřesvědčivý je výsledek při použití funkce `XMLROW`, která je časově dost neefektivní. Je to překvapivé, protože nepřidává žádnou funkcionalitu navíc. Je to pouze zkratka vyjádření téhož pomocí funkce `XMLELEMENT`.

V úvahu by mohla ještě připadat možnost využití uložených procedur, které by vracely data v požadovaném formátu. Obrovskou jejich nevýhodou je jejich neflexibilnost, kdy dotaz lze jen těžko omezit podle nějakého atributu. Proto se s jejich využitím nepočítá.

Z celkového pohledu se jako nejvhodnější zdá varianta číslo 3. Dotazy nejsou o tolik pomalejší než u ostatních uložení a navíc mají obrovskou výhodu v rychlosti při selekci pomocí jiného atributu než id.

Kapitola 4

Konceptuální XSEM model

Celá práce vychází z konceptuálního XSEM modelu, který bude v této kapitole stručně představen. Všechny jeho konstrukce budou ukázány na modelovém příkladu. Ten bude dále užíván i v následujících kapitolách.

Vzorový příklad modeluje malou databázi filmů. Každý film má alespoň jednoho režiséra. Ve filmu může hrát 0–n herců. Hercům hrajícím ve filmu je přidělena role, která má svůj název. Na každý film může být napsáno několik recenzí. Každá recenze má právě jednoho autora. Text recenze je smíšený s odkazy na jiné filmy a na osoby. Film dále může být oceněn na některé soutěži. Každá soutěž má několik kategorií. Ocenění v dané soutěži a kategorii získá vždy právě jedna osoba a právě jeden film. Např. pro kategorii *nejlepší film* je vždy nutné uvést i osobu reprezentující film, třeba režiséra. Stejně tak i pro kategorii *nejlepší režisér* je nutné uvést i film, za jehož režii režisér ocenění dostal. Kategorie jako *celoživotní přínos*, kde nelze určit právě jeden film, nejsou povoleny.

4.1 XSEM-ER model

XSEM-ER model vychází z klasického E-R modelu a je určen pro modelování sémantiky dat. To, jak budou modelovaná data organizována v hierarchických XML dokumentech, není na úrovni XSEM-ER modelu důležité. Příklad schématu pro databázi filmů navrženého v XSEM-ER modelu je na obrázku 4.1. Jsou v něm obsaženy všechny objekty, kterými XSEM-ER model disponuje.

Silný entitní typ (strong entity type) — formálně $E = (attr(E), id(E))$, kde E je název entitního typu, $attr(E)$ je seznam jeho atributů a $id(E)$ je podmnožina $attr(E)$, která se nazývá klíč dané entity. V diagramu je znázorněn obdélníkem s názvem entitního typu a seznamem jeho atributů. Silným entitním typem je např. *Osoba*.

Slabý entitní typ (weak entity type) – formálně $E = (attr(E), id(E), det(E))$, kde E je název entitního typu, $attr(E)$ je seznam jeho atributů, $id(E)$ je podmnožina $attr(E)$, která se nazývá relativní klíč dané entity, $det(E)$ je neprázdná množina dvojic (E_i, l_i), kde E_i je entitní typ[1] nebo komponentní klastr nazývaný *determinant* a l_i je označení role E_i vůči E. Toto označení může být prázdné. V diagramu je znázorněn obdélníkem s dovnitř vloženým

[1]Pokud není speficikováno zda se jedné o slabý nebo silný entitní typ, mohou to být oba.

šestiúhelníkem, názvem a seznamem atributů. Ke svým determinantům je připojen orientovanou hranou směřující do determinantu. Pokud je uvedeno označení role, potom je připsáno k příslušné hraně. Příkladem je *Role*.

Vztahový typ[2] (relationship type) — formálně $R = (attr(R), part(R))$, kde R je název vztahového typu, $attr(R)$ je seznam jeho atributů a $part(R)$ je množina nejméně dvou dvojic (E_i, l_i), kde E_i je entitní typ nebo komponentní klastr nazývaný *účastník* vztahu R a l_i je označení role E_i vůči E. Toto označení může být prázdné. V diagramu je znázorněn šestiúhelníkem s názvem vztahového typu a seznamem atributů. Účastníci vztahu jsou připojeni orientovanou hranou směřující k účastníkům. Pokud je uvedeno označení role, potom je připsáno k příslušné hraně. Příkladem je *Ocenění*.

Typ datového uzlu (data node type) — formálně $D = (type(D), par(D))$, kde D je název typu datového uzlu, $type(D)$ je datový typ omezující možné hodnoty datového uzlu a $par(D)$ je entitní typ nazývaný *rodič* datového uzlu. V diagramu je zobrazen jako elipsa s názvem typu datového uzlu. Typ datového uzlu se využívá k modelování nestrukturovaných dat. Příkladem je *TextRec*.

Komponentní klastr (component cluster) — formálně nechť $P_1, \ldots, P_N$ je seznam entitních typů. Komponentní klastr C s komponentami $P_1, \ldots, P_N$ je výraz $P_1 + \ldots + P_N$. V diagramu znázorněn kolečkem s vepsaným znaménkem +. Komponenty jsou připojeny orientovanou hranou směřující do komponenty. Je využíván k modelování nepravidelné struktury dat. Příkladem je klastr označený _8.

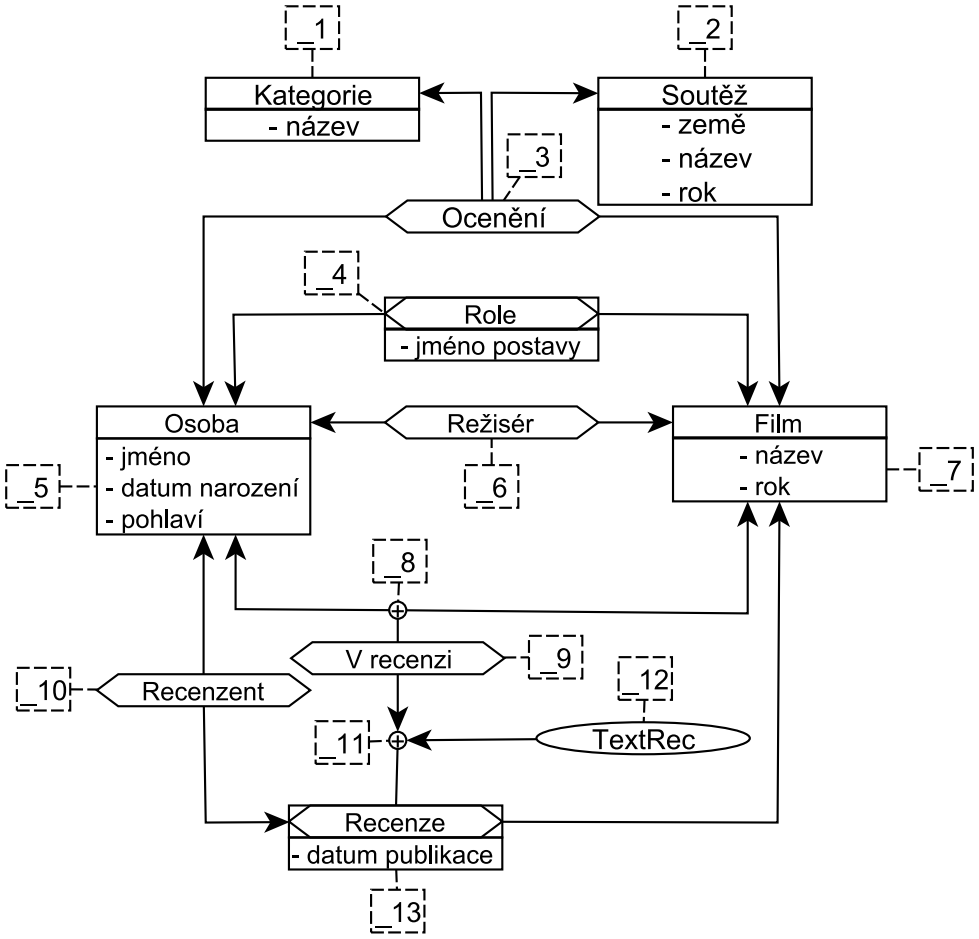
Spojovací klastr (connection cluster) — formálně nechť E je entitní typ, $P_1, \ldots, P_N$ je seznam vztahových typů, slabých entitních typů a typů datového uzlu, kteří mají E jako účastníka relace, determinant nebo rodiče. Spojovací klastr je výraz $(E, P_1 + \ldots + P_N)$. E se nazývá *rodič* klastru, P_i jsou *komponenty* klastru. V diagramu je znázorněn stejně jako spojovací klastr. K rodiči je připojen neorientovanou hranou. Komponenty jsou připojeny orientovanou hranou směřující do klastru. Využívá se k modelování smíšeného obsahu. Příkladem je klastr označený _11.

Oproti klasickému XSEM-ER schématu definovaném v [4] je zde přidán ještě jeden objekt, který však nemá žádný sémantický význam. Pouze usnadňuje pochopení diagramu a následných operací s ním. Je to obdélník s čárkovaným ohraničením, který je připojený čárkovanou hranou k jednotlivým objektů. Obsahuje identifikátor, pod kterým je objekt uložen v XML reprezentaci diagramu.

4.2 Hierarchické projekce

Hierarchické projekce poskytují nástroj pro převod „plochých“ dat namodelovaných XSEM-ER schématem do hierarchické struktury vytvářené XSEM-H modelem.

[2]Někdy se používá také název relace nebo relační typ



Obrázek 4.1: XSEM-ER schéma

Hierarchickou projekci lze nadefinovat na vztahovém typu, slabém entitním typu a typu datového uzlu. Zde bude uvedena pouze definice pro vztahový typ. Pro slabý entitní typ a typ datového uzlu by definice byla obdobná.

Hierarchická projekce (hierarchical projection) – hierarchická projekce ρ vztahového typu R je výraz $R^{T_1,\ldots,T_k}[P \to Q]$, kde $T_1, \ldots, T_k$ jsou účastníky vztahu a P, Q jsou účastníky vztahu, nebo právě jeden z nich je samotný vztahový typ R. P se nazývá *rodič*, Q *potomek* a $T_1, \ldots, T_k$ *kontext* hierarchické projekce. Kontext může být prázdný. Projekce ρ uspořádává instance P a Q do hierarchické struktury, kdy instance účastníka Q jsou potomky instance účastníka P. Instance kontextu $T_1, \ldots, T_k$ jsou v hierarchické struktuře předky instance P.

Názornější bude předvést význam hierarchické projekce na příkladu. Pokud je třeba uspořádat do hierarchické struktury entity Osoba a Film na základě vztahu Režisér, lze použít následující dva způsoby: ke každému filmu je vypsán seznam jeho režisérů, tomu odpovídá projekce (p01), nebo je ke každé osobě vypsán seznam filmů, které režírovala, k tomu slouží projekce (p02).

$$\text{Režisér}[\text{Film} \to \text{Osoba}] \quad \text{(p01)}$$
$$\text{Režisér}[\text{Osoba} \to \text{Film}] \quad \text{(p02)}$$

Pokud je třeba vypsat v hierarchické struktuře seznam filmů, ke každému filmu seznam rolí a ke každé roli osobu, která ji hraje, použijí se následující dvě projekce.

$$\begin{array}{ll} \text{Role}[\text{Film} \rightarrow \text{Role}] & \text{(p03)} \\ \text{Role}^{\text{Film}}[\text{Role} \rightarrow \text{Osoba}] & \text{(p04)} \end{array}$$

4.3 XSEM-H model

Pomocí XSEM-ER schématu je popsána sémantika dat jako celku. XSEM-H model poté umožňuje popsat, jak budou data reprezentována v hierarchických XML dokumentech. Pro každý typ XML dokumentu je vytvořeno jedno XSEM-H schéma, které je pohledem na část XSEM-ER schématu. XSEM-H schéma tedy pouze modeluje, jak budou data popsaná odpovídající částí XSEM-ER schématu organizována v daném typu XML dokumentů. Např. pro databázi filmů jsou navrženy 3 hierarchické pohledy. Jejich diagramy jsou na obrázcích 4.2, 4.3 a 4.4.

XSEM-H pohled pro XSEM-ER schéma $\mathcal{ER}$ je orientovaný graf. Je složen z následujících objektů:

Uzel (node) — formálně trojice $N = (content(N), lab(N), repr(N))$, kde N je název uzlu, $content(N)$ je množina hran vycházejících z uzlu, $lab(N)$ je označení uzlu, může být prázdné, a $repr(N)$ je entitní typ, vztahový typ nebo typ datového uzlu z $\mathcal{ER}$ reprezentovaný uzlem N. Množina hran je implicitně považována za uspořádanou. Graficky je znázorněn obdélníkem s názvem uzlu, případně je ještě uvedeno označení uzlu. Příkladem je uzel *Film* označený `h12` z pohledu na obrázku 4.2

Orientovaná hrana (oriented edge) — formálně trojice $E = (parent(E), child(E), repr(E))$, kde E je název hrany, $parent(E)$ a $child(E)$ jsou uzly, které se nazývají *rodič* a *potomek* a $repr(E)$ je hierarchická projekce reprezetovaná touto hranou. Graficky je znázorněna orientovanou hranou.

Klastr uzlů (cluster of nodes) — formálně výraz $N_1 + \ldots + N_k$, kde $N_1, \ldots, N_k$ jsou uzly. Klastr uzlů reprezentuje v XSEM-H diagramu komponentní klastr. V diagramu je znázorněn kolečkem s vepsaným znaménkem `+`. Komponenty jsou ke klastru připojeny pomocí neorientované hrany. Příkladem je klastr označený `h28` na obrázku 4.4.

Klastr hran (cluster of edges) — formálně výraz $E_1 + \ldots + E_k$, kde $E_1, \ldots, E_k$ jsou hrany se stejným rodičem. Klastr hran reprezentuje v XSEM-H diagramu spojovací klastr. V diagramu je znázorněn stejně jako klastr uzlů. K rodiči je připojen neorientovanou hranou, komponenty jsou připojené orientovanými hranami. Příkladem je klastr označený `h27` na obrázku 4.4.

Opět je přidán objekt s identifikátorem použitým v XML reprezentaci stejně jako u XSEM-ER. Navíc ještě přidán k hranám odkaz na hierarchické projekce, které jsou uvedené později v 4.3.1.

Pohled na obrázku 4.2 popisuje strukturu XML dokumentu, ve kterém je uložen seznam osob, ke každé osobě je připojen seznam filmů, které osoba režírovala a pro každý film je uveden seznam rolí s obsazením herců. Fragment tohoto XML dokumentu by mohl vypadat následovně.

```
<reziser>
  <jmeno>Jan Dvořák</jmeno>
  <datum_narozeni>1943-02-12</datum_narozeni>
  <pohlavi>M</pohlavi>
  <film>
    <nazev>Film 464</nazev>
    <rok>1968</rok>
    <role>
      <jmeno_postavy>Postava 9261</jmeno_postavy>
      <herec>
        <jmeno>David Dvořák</jmeno>
        <datum_narozeni>1943-02-14</datum_narozeni>
        <pohlavi>M</pohlavi>
      </herec>
    </role>
    <role>
      <jmeno_postavy>Postava 9262</jmeno_postavy>
      <herec>
        <jmeno>Věra Pospíšilová</jmeno>
        <datum_narozeni>1951-06-30</datum_narozeni>
        <pohlavi>Z</pohlavi>
      </herec>
    </role>
  </film>
</reziser>
```

Pohled na obrázku 4.3 popisuje strukturu XML dokumentu, který vypisuje seznam soutěží a jejich kategorií. Ke každé kategorii je uveden oceněný film a osoba. Příklad takového XML dokumentu:

```
<soutez>
  <zeme>Česko</zeme>
  <nazev>Soutez 3</nazev>
  <rok>1970</rok>
  <kategorie>
    <nazev>nejlepší režie</nazev>
    <oceneni>
      <osoba>
        <jmeno>Jarmila Němcová</jmeno>
        <datum_narozeni>1970-03-21</datum_narozeni>
        <pohlavi>Z</pohlavi>
      </osoba>
      <film>
        <nazev>Film 462</nazev>
        <rok>1974</rok>
      </film>
    </oceneni>
  </kategorie>
  <kategorie>
    <nazev>nejlepší kamera</nazev>
    <oceneni>
      <osoba>
        <jmeno>Miroslav Svoboda</jmeno>
```

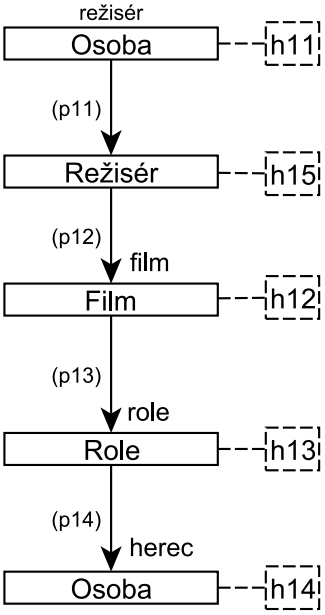
```
      <datum_narozeni>1981-12-20</datum_narozeni>
      <pohlavi>M</pohlavi>
    </osoba>
    <film>
      <nazev>Film 673</nazev>
      <rok>1999</rok>
    </film>
  </oceneni>
 </kategorie>
</soutez>
```

Pohled na obrázku 4.4 popisuje strukturu XML dokumentu, který obsahuje seznam filmů a recenzí k nim napsaných. U recenze je uveden autor a samotný text recenze smíšený s odkazy na jiné filmy a osoby.

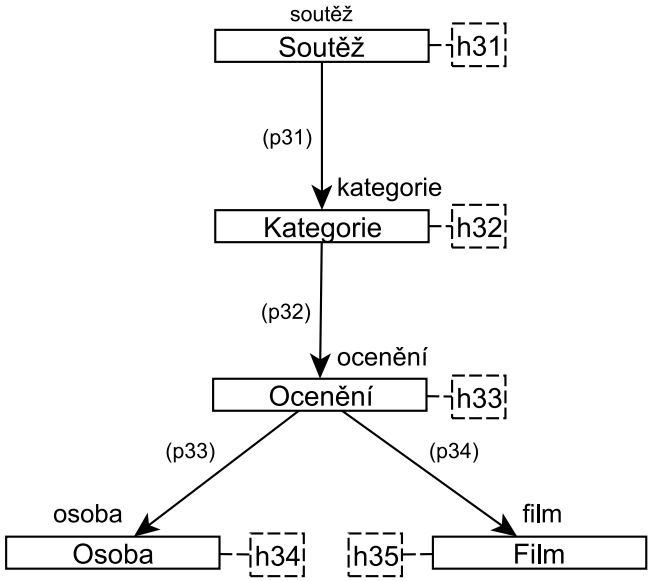
```
<film>
 <nazev>Film 5</nazev>
 <rok>1980</rok>
 <recenze>
  <datum_publikace>1980-12-30</datum_publikace>
  <autor>
   <jmeno>Hana Dvořáková</jmeno>
   <datum_narozeni>1943-09-03</datum_narozeni>
   <pohlavi>Z</pohlavi>
  </autor>
  Nějaký text. Text. Text.
  <film>
   <nazev>Film 737</nazev>
   <rok>1987</rok>
  </film>
  Další část textu recenze.
  <osoba>
  <jmeno>Martin Jelínek</jmeno>
   <datum_narozeni>1974-07-08</datum_narozeni>
   <pohlavi>M</pohlavi>
  </osoba>
  Konec textu recenze.
 </recenze>
 <recenze>
  <datum_publikace>1980-04-13</datum_publikace>
  <autor>
   <jmeno>Jitka Nováková</jmeno>
   <datum_narozeni>1960-03-26</datum_narozeni>
   <pohlavi>Z</pohlavi>
  </autor>
  Text recenze.
 </recenze>
</film>
```

4.3.1 Hierarchické projekce pro XSEM-H pohledy

Hrany v XSEM-H pohledech reprezentují tyto hierarchické projekce:



Obrázek 4.2: Diagram 1



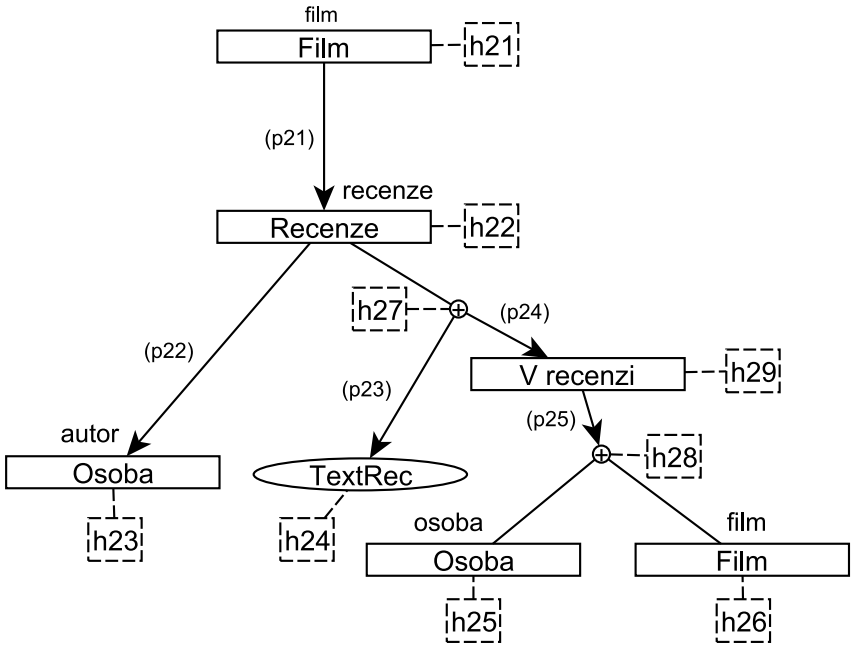
Obrázek 4.3: Diagram 3

Režisér[Osoba → Režisér]	(p11)
Režisér$^{\text{Osoba}}$[Režisér → Film]	(p12)
Role[Film → Role]	(p13)
Role$^{\text{Film}}$[Role → Osoba]	(p14)
Recenze[Film → Recenze]	(p21)
Recenzent[Recenze → Osoba]	(p22)
TextRec[Recenze → TextRec]	(p23)
V recenzi[Recenze → V recenzi]	(p24)
V recenzi$^{\text{Recenze}}$[V recenzi → Osoba+Film]	(p25)
Ocenění[Soutěž → Kategorie]	(p31)
Ocenění$^{\text{Soutěž}}$[Kategorie → Ocenění]	(p32)
Ocenění$^{\text{Soutěž, Kategorie}}$[Ocenění → Osoba]	(p33)
Ocenění$^{\text{Soutěž, Kategorie}}$[Ocenění → Film]	(p34)

4.4 Integritní omezení

K definování integritních omezení je využito hierarchických projekcí. Existují dva typy omezení: kardinalita a uspořádání.

Kardinalita (cardinality constraint) — nechť ρ je označení pro hierarchickou projekci $R^{T_1,\ldots,T_k}[P \to Q]$. Potom kardinalita objektu P v projekci ρ má tvar $card(\rho, P) = (m, n)$ a kardinalita Q v projekci ρ tvar $card(\rho, Q) = (m, n)$. Např. kardinalita označená (i01) vyjadřuje omezení na počet autorů (recenzentů) recenze. Každá recenze musí mít právě jednoho autora.



Obrázek 4.4: Diagram 2

Kardinalit je možné v XSEM modelu specifikovat velké množství. Pro vytváření logického schématu databáze budou brány v úvahu pouze omezení na hierarchických projekcích vztahových typů a to v následujícím tvaru. Pro vztahový typ $R = ((), (E_1, E_2, \ldots, E_n))$ je nutné specifikovat pro všechna $i = 1, \ldots, n$ kardinalitu E_i v hierarchické projekci $R[E_i \rightarrow R]$. Na tato integritní omezení je nahlíženo jako na kardinality vztahu v klasickém ER modelu. Je povinné tyto kardinality uvést v serializované podobě XSEM schématu.

Uspořádání (ordering) — hierarchická projekce ρ s rodičem P a potomkem Q uspořádává instance P a Q do hierarchické struktury. Projekce ρ přiřadí každé instanci P seznam instancí Q jako potomky. Tento seznam může být uspořádaný, pak je specifikováno integritní omezení $ordered(\rho)$, nebo neuspořádaný, $unordered(\rho)$.

Implicitně se považuje hierarchická projekce za uspořádanou. Není tedy nutné toto integritní omezení uvádět. Není pro něj ani navrženo vyjádření v serializovaném XSEM schématu.

Zde jsou uvedeny povinné integritní omezení, tj. integritní omezení pro relace.

card(Recenzent[Recenze → Recenzent], Recenze)=(1, 1)	(i01)
card(Recenzent[Osoba → Recenzent], Osoba)=(0, *)	(i02)
card(Režisér[Film → Režisér], Film)=(1, *)	(i03)
card(Režisér[Osoba → Režisér], Osoba)=(0, *)	(i04)
card(V recenzi[Recenze → V recenzi], Recenze)=(0, *)	(i05)
card(V recenzi[Osoba+Film → V recenzi], Osoba+Film)=(0, *)	(i06)
card(Ocenění[Kategorie → Ocenění], Kategorie)=(1, *)	(i07)
card(Ocenění[Soutěž → Ocenění], Soutěž)=(0, *)	(i08)
card(Ocenění[Osoba → Ocenění], Osoba)=(0, *)	(i09)
card(Ocenění[Film → Ocenění], Film)=(0, *)	(i10)

4.5 Serializace XSEM diagramů

Dosud nebyl k dispozici žádný algoritmus pro přepis diagramů do serializované podoby. Jednou z možností bylo využít některou stávající metodu pro přepis klasických ER nebo UML diagramů a rozšířit je o prvky, které přidává XSEM model. Jedním z kandidátů byl standart OMG XMI [5]. Ten je však velmi komplexní a pro účely práce až zbytečně složitý. Proto bylo navrženo jednoduché XML schéma pro serializaci XSEM diagramů, které svojí přehledností plně vyhovuje požadavků této práce. Navíc je napsáno v jazyce XML Schema, tedy je snadno rozšiřitelné o další možné konstrukce. Schéma postihuje pouze sémantiku obsaženou v diagramech, vůbec neřeší nakreslení diagramů jako XMI. Zdrojový kód schématu je možné nalézt v příloze.

Kapitola 5

Převod XSEM-ER schématu na čistě relační schéma

Základem celé práce je navržení pravidel pro převod XSEM-ER schématu do logického schématu databáze. První ze dvou předložených řešení je převod do čistě relačního databázového schématu. Převod bude vycházet z převodu klasického ER schématu do logického schématu databáze. Pro konstrukce, které jsou v XSEM-ER navíc, budou navržena schémata tabulek. Nejprve ale budou rozebrány možnosti uložení složených a vícehodnotových atributů. Dále bude nutné nějak vyřešit identifikaci objektů v databázi.

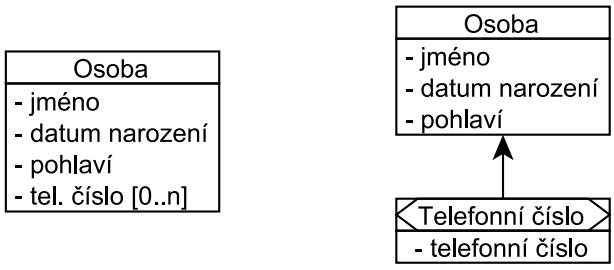
5.1 Složené a vícehodnotové atributy

Model XSEM dovoluje používat složené a vícehodnotové atributy. Jak již bylo řečeno dříve, implementace tuto vlastnost nebude zahrnovat. Budou zde ale rozebrány možnosti uložení takovýchto atributů do databáze.

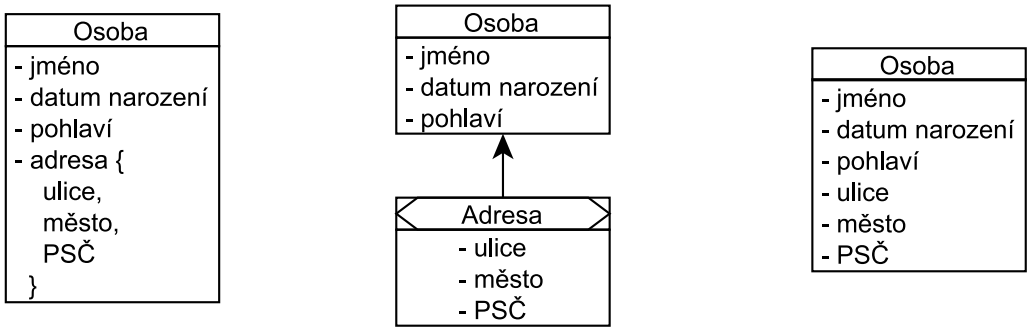
Jelikož se bude ve všech později navržených řešeních vycházet z ER modelu, nabízí se využít některou z možností, které již byly navrženy pro ER model. Nejpoužívanější možností je nahrazení vícehodnotového atributu slabým entitním typem, jak je vidět na obrázku 5.1 vpravo. Principiálně stejné řešení je vytvoření relace a silného entitního typu. Složený atribut lze jednoduše dekomponovat na jednoduché atributy, případně využít opět slabého entitního typu, nebo relace a silného entitního typu. Tato řešení jsou zobrazena na obrázku 5.2 vpravo a uprostřed. Protože jsou tyto techniky řešení převzaté z ER modelu, lze je využít i v čistě relační databázi. Jejich nevýhodou jsou další přidané tabulky a tím i spojování tabulek při dotazování.

Současné databázové systémy poskytují vyspělejší techniky, které je možno k uložení složených a vícehodnotových atributů použít. Následující dvě řešení, narozdíl od řešení převzatého z ER modelu, neřeší vícehodnotové a složené atributu na úrovni konceptuálního modelu, ale až na úrovni převodu navrženého konceptuálního schématu do logického schématu databáze. První možností je využití objektového rozšíření relační databáze. Vícehodnotový atribut by byl reprezentován polem s udanou maximální délkou. V některých systémech nejde pole použít přímo, ale je nutné ho zapouzdřit do uživatelsky definovaného typu. Složený atribut by byl reprezentován uživatelsky definovaným typem.

```
CREATE TYPE TELEFONNI_CISLA AS DECIMAL(9,0) ARRAY[];
```



Obrázek 5.1: Vícehodnotový atribut



Obrázek 5.2: Složený atribut

```
CREATE TABLE Osoba (
  id_osoba INTEGER NOT NULL PRIMARY KEY,
  jmeno VARCHAR(50),
  datum_narozeni DATE,
  pohlavi CHAR,
  tel_cisla TELEFONNI_CISLA
)

CREATE TYPE ADRESA AS (
  ulice VARCHAR(100),
  mesto VARCHAR(100),
  psc DECIMAL(5,0)
)

CREATE TABLE Osoba (
  id_osoba INTEGER NOT NULL PRIMARY KEY,
  jmeno VARCHAR(50),
  datum_narozeni DATE,
  pohlavi CHAR,
  adresa ADRESA
)
```

Poslední možnost, která zde bude uvedena, je uložení atributů ve formátu XML. Takové řešení by pokrylo jak vícehodnotovost atributu, tak i možnost slo-

ženého atributu. Jak již bylo diskutováno v části 3.6, problém nastane v selekci podle nějaké hodnoty složeného/vícehodnotového atributu, protože indexování `XML` datového typu ještě není na takové úrovni jako indexování dat klasických datových typů. V následujícím příkladu jsou schémata pro vícehodnotový atribut *telefonní číslo* a složený atribut *adresa*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  <xs:element name="telefonni_cisla">
    <xs:comlexType>
      <xs:sequence minOccur="0" maxOccur="unbounded">
        <xs:element name="telefoni_cislo">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="\d{9}"/>
            </xs:restriction>
          </xs:simpleType>
      </xs:sequence>
    </xs:comlexType>
  </xs:element>
</xs:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  <xs:element name="adresa">
    <xs:comlexType>
      <xs:sequence>
        <xs:element name="ulice" type="xs:string"/>
        <xs:element name="mesto" type="xs:string"/>
        <xs:element name="psc">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="\d{5}"/>
            </xs:restriction>
          </xs:simpleType>
         </xs:element>
      <xs:sequence>
    </xs:comlexType>
  </xs:element>
</xs:schema>
```

Jak využití objektového rozšíření tak i datového typu `XML` se zdají vhodné pro uložení složených a vícehodnotových atributů a záleží jen na návrháři, kterou možnost si vybere.

5.2 Identifikace objektů

V ER modelu má každá entita explicitně určený identifikátor/klíč[1]. Ten může být buď jednoduchý, tvořený jediným atributem, nebo složený z více atributů. Tento klíč vždy jednoznačně identifikuje instanci daného entitního typu. XML však nabízí i možnost implicitní identifikace objektů na základě výskytu objektu v XML dokumentu. Tuto možnost implicitní identifikace umožňuje i XSEM-ER model. Otázka klíčů zatím není v čase vzniku této práce v XSEM modelu zcela vyřešena. Proto zde navržené řešení také zcela nepokrývá daný problém.

Navíc používání přirozených atributů (ne uměle vygenerovaných) jako klíčových se v dnešní době nedoporučuje. Je to spíš věc metodiky výuky a v praxi se téměř vždy klíč uměle generuje. Tento trend bude dodržován i v této práci.

5.2.1 Klíč entitních typů

Klíč pro silný entitní typ určený množinou $id(E)$ může být také prázdný nebo složený z nepovinných atributů. Pro uložení dat by tedy bylo nutné rozlišovat entity s explicitně určeným klíčem a entity s klíčem, který využívá implicitní uspořádání. Navíc by bylo nutné ukládat uspořádání dat z XML dokumentu. Pro zjednodušení proto bude každému entitnímu typu přiřazen uměle generovaný identifikátor. Tím se vyřeší problém s implicitním uspořádáním, které bude udržováno pomocí vzrůstajících hodnot umělého identifikátoru. Název identifikátoru bude tvořen předponou `id_` a názvem entitního typu. Pro entitu *Osoba* by identifikátor měl název `id_osoba`. Zde je nutné připomenout, že jazyk SQL nerozlišuje velká a malá písmena. Naproti tomu jazyk XQuery rozlišuje a je třeba si na to dávat pozor.

Stejně se bude postupovat i u slabého entitního typu. Každé instanci přidáme umělý klíč.

5.2.2 Klíč vztahového typu

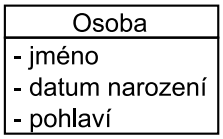
V ER modelu byla každá instance vztahového typu jednoznačně identifikována instancemi entit účastnících se vztahu. V XML toto neplatí a dané instance zúčastněných entit mohou tvořit více vztahů, které jsou rozlišeny pořadím dat v XML dokumentu. Proto bude každému vztahovému typu přiřazen uměle generovaný klíč. Ten bude také udržovat pořadí dat z dokumentu.

5.3 Tabulka pro obecný entitní typ

Pro každý entitní typ se vytvoří samostatná tabulka. Sloupce budou tvořeny atributy entitního typu, navíc je přidán sloupec umělého klíče. Atributy z $id(E)$ nemají v tabulce žádný zvláštní význam.

```
CREATE TABLE Osoba (
  id_osoba INTEGER NOT NULL PRIMARY KEY,
  jmeno VARCHAR(50),
  datum_narozeni DATE,
  pohlavi CHAR
)
```

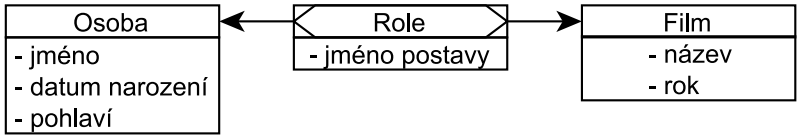
[1] klíč = identifikátor



Obrázek 5.3: Silný entitní typ

5.4 Tabulka pro slabý entitní typ

Pro každý slabý entitní typ se vytvoří samostatná tabulka. Sloupce tvoří atributy slabého entitního typu, je přidán sloupec umělého klíče a sloupce cizích klíčů, které odkazují do tabulek determinantů slabého entitního typu. Pokud je determinant slabého entitního typu připojen přes komponentní klastr, tak cizí klíč odkazuje do tabulky komponentního klastru.



Obrázek 5.4: Slabý entitní typ

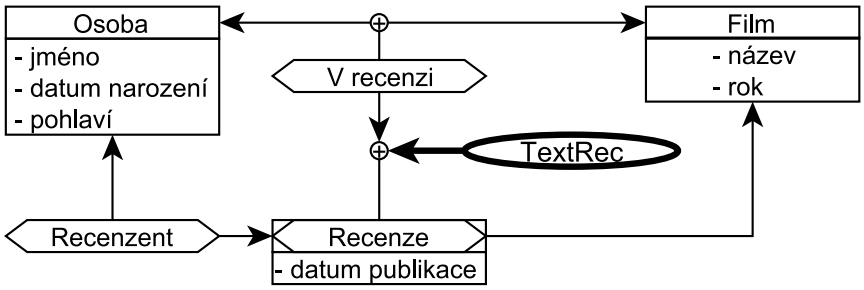
```
CREATE TABLE Role (
  id_role INTEGER NOT NULL PRIMARY KEY,
  jmeno_postavy VARCHAR(50),
  id_osoba INTEGER REFERENCES Osoba(id_osoba),
  id_film INTEGER REFERENCES Film(id_film)
)
```

5.5 Tabulka pro typ datového uzlu

Tabulka pro typ datového uzlu bude mít tři sloupce: umělý klíč, cizí klíč odkazující do tabulky rodiče typu datového uzlu a sloupec pro obsah datového uzlu. Pro uložení obsahu datového uzlu přichází v úvahu několik datových typů. Nejuniverzálnější je `CLOB`, poskytuje dostatečný prostor pro uložení obsahu, může však být při některých manipulacích pomalý. Pokud je jistota, že obsah datového uzlu nebude příliš velký, je možno využít i datový typ `VARCHAR`, případně jeho modifikaci. V DB2 je k dispozici typ `LONG VARCHAR` a ten bude pro uložení obsahu datového uzlu využit.

Pokud nebude typ datového uzlu připojen k rodiči přes spojovací klastr ale přímo, bude výhodnější připojit obsah datového uzlu k tabulce rodiče. V takovém případě by se tabulka pro typ datového uzlu vůbec nevytvářela.

```
CREATE TABLE TextRec (
  id_textrec INTEGER NOT NULL PRIMARY KEY,
  textrec LONG VARCHAR,
```

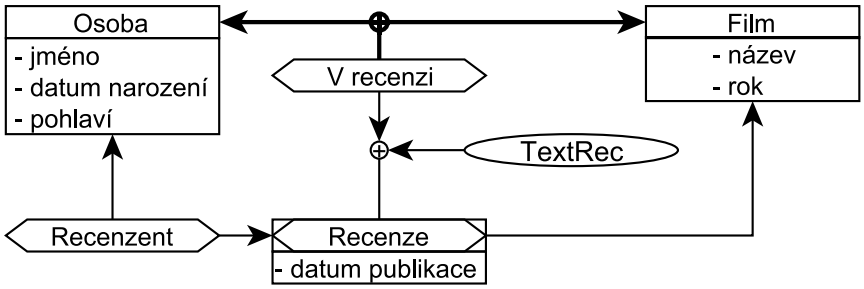


Obrázek 5.5: Typ datového uzlu

```
  id_recenze INTEGER REFERENCES Recenze(id_recenze)
)
```

5.6 Komponentní klastr

Pro každý komponentní klastr se vytvoří samostatná tabulka. Sloupce tabulky tvoří umělý klíč a cizí klíče odkazující do tabulek komponent klastru. V jednom řádku tabulky může být pouze jeden odkaz do tabulek komponent různý od hodnoty `NULL`.



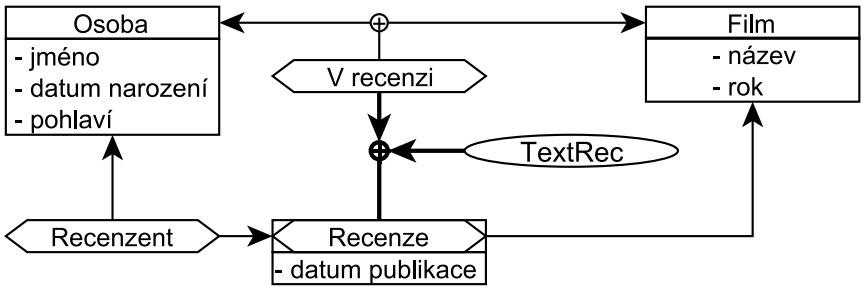
Obrázek 5.6: Komponentní klastr

```
CREATE TABLE ComponentC(
  id_componentc INTEGER NOT NULL PRIMARY KEY,
  id_osoba INTEGER REFERENCES Osoba(id_osoba),
  id_film INTEGER REFERENCES Film(id_film),
  CONSTRAINT one_not_null CHECK ( ( id_osoba IS NOT NULL AND id_film IS
      NULL ) OR ( id_osoba IS NULL AND id_film IS NOT NULL ) )
)
```

5.7 Spojovací klastr

Pro každý spojovací klastr je vytvořena samostatná tabulka. Její sloupce jsou tvořeny umělým klíčem, cizím klíčem odkazujícím do tabulky rodiče klastru

a cizími klíči odkazujícími do tabulek komponent klastru. Navíc přidán sloupec `impl_uspor`, který určuje pořadí komponent pro jednu instanci rodiče klastru, tj. pro jednu hodnotu cizího klíče do tabulky rodiče. Hodnotu různou od hodnoty `NULL` může mít v jednom řádku tabulky pouze jeden z cizích klíčů odkazujících do tabulek komponent.



Obrázek 5.7: Spojovací klastr

```
CREATE TABLE ConnectionC (
  id_connectionc INTEGER NOT NULL PRIMARY KEY,
  id_recenze INTEGER NOT NULL REFERENCES Recenze(id_recenze),
  id_text_rec INTEGER REFERENCES TextRec(id_text_rec),
  id_v_recenzi INTEGER REFERENCES V_recenzi(id_v_recenzi),
  impl_uspor INTEGER NOT NULL,
  CONSTRAINT one_not_null CHECK ( (id_text_rec IS NULL AND id_v_recenzi
       IS NOT NULL) OR (id_text_rec IS NOT NULL AND id_v_recenzi IS NULL
      ) )
)
```

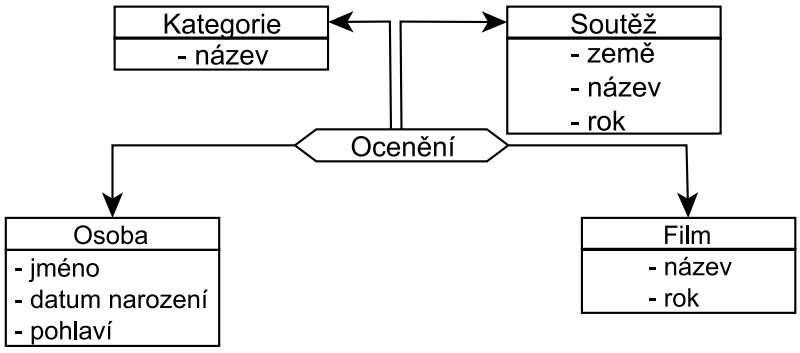
5.8 Tabulka pro vztahový typ

Obecně se pro každý vztahový typ vytvoří samostatná tabulka. Někdy je ale výhodnější sloučit tabulku vztahu s jednou či více tabulkami zúčastněných entitních typů. Vhodnost sloučení závisí na kardinalitě hierarchických projekcí vztahu.

Kardinality, které musí být povinně uvedeny a rozhoduje se podle nich o možném sloučení tabulek vztahu a účastníka vztahu, byly popsány v kapitole představující XSEM model v části 4.4.

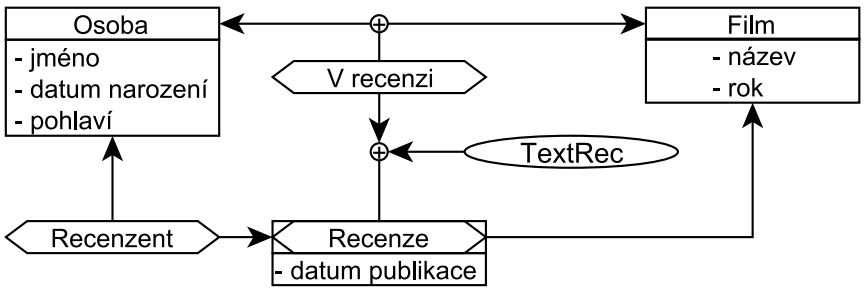
Sloupce samostatné tabulky vztahového typu tvoří umělý klíč, atributy vztahového typu a cizí klíče odkazující do tabulek účastníků vztahu.

```
CREATE TABLE Oceneni (
  id_oceneni INTEGER NOT NULL PRIMARY KEY,
  id_osoba INTEGER NOT NULL REFERENCES Osoba(id_osoba),
  id_film INTEGER NOT NULL REFERENCES Film(id_film),
  id_kategorie INTEGER NOT NULL REFERENCES Kategorie(id_kategorie),
  id_soutez INTEGER NOT NULL REFERENCES Soutez(id_soutez),
)
```



Obrázek 5.8: Vztahový typ

Pokud jsou jedinými účastníky vztahového typu komponentní klastr a entita, ke které je vztah připojen přes spojovací klastr, pak se samostatná tabulka pro vztahový typ nemusí vytvářet. Atributy vztahového typu se připojí k tabulce komponentního klastru. Navíc je k tabulce komponentního klastru přidán cizí klíč odkazující do tabulky druhého účastníka vztahového typu, entity. V takovém případě cizí klíč z tabulky spojovacího klastru neodkazuje do tabulky vztahového typu, protože neexistuje, ale do tabulky komponentního klastru.

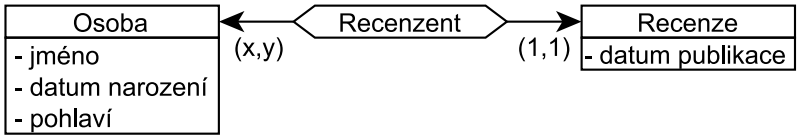


Obrázek 5.9: Vztahový typ mezi komponentním a spojovacím klastrem

```
CREATE TABLE ComponentC(
  id_componentc INTEGER NOT NULL PRIMARY KEY,
  id_osoba INTEGER REFERENCES Osoba(id_osoba),
  id_film INTEGER REFERENCES Film(id_film),
  id_recenze INTEGER REFERENCES Recenze(id_recenze),
  CONSTRAINT one_not_null CHECK ( ( id_osoba IS NOT NULL AND id_film IS
      NULL ) OR ( id_osoba IS NULL AND id_film IS NOT NULL ) )
)
CREATE TABLE ConnectionC (
  id_connectionc INTEGER NOT NULL PRIMARY KEY,
  id_text_rec INTEGER REFERENCES TextRec(id_text_rec),
  id_componentc INTEGER REFERENCES ComponentC(id_componentc),
  id_recenze INTEGER NOT NULL REFERENCES Recenze(id_recenze),
  impl_uspor INTEGER NOT NULL,
  CONSTRAINT one_not_null CHECK ( (id_text_rec IS NULL AND id_v_recenzi
```

```
      IS NOT NULL) OR (id_text_rec IS NOT NULL AND id_v_recenzi IS NULL
     ) )
)
```

Samostatná tabulka pro vztahový typ se nebude vytvářet pouze pro binární vztah, který má vůči jednomu účastníku kardinalitu $(1, 1)$. Označme ho U_1. V takovém případě se všechny atributy vztahu připojí k tabulce účastníka U_1. Navíc se k tabulce účastníka U_1 připojí cizí klíč odkazující do tabulky druhého účastníka vztahu. Pokud by měli kardinalitu $(1, 1)$ vůči vztahu oba účastníci, pak vybereme jednoho z nich a k jeho tabulce připojíme atributy vztahu a cizí klíč.



Obrázek 5.10: Binární vztah s kardinalitou $(1, 1)$

```
CREATE TABLE Osoba (
  id_osoba INTEGER NOT NULL PRIMARY KEY,
  jmeno VARCHAR(50),
  datum_narozeni DATE,
  pohlavi CHAR(1)
);
CREATE TABLE Recenze (
  id_recenze INTEGER NOT NULL PRIMARY KEY,
  datum_publikace DATE,
  id_osoba INTEGER REFERENCES Osoba(id_osoba)
);
```

5.9 Indexy v tabulkách

Pro lepší výkon při dotazování je nutné použít indexy. Databázový systém by měl automaticky generovat indexy na primárních klíčích tabulek. Pokud ne, je třeba indexy na primárních klíčích vygenerovat ručně. Dále se velmi doporučuje vytvořit indexy nad všemi cizími klíči v tabulkách. U některých dotazů tyto indexy zkrátí dobu vykonání dotazu na polovinu. Další indexy už velmi závisí na konkrétních dotazech a jejich četnosti zadávání. Nelze je tudíž vygenerovat z pouhého XSEM schématu. Je zde potřeba zásah zkušeného databázového specialisty.

5.10 Konstrukce dotazů

Pro konstrukci XML dokumentů odpovídajících jednotlivým XSEM-H diagramům je nutné použít jazyk XQuery spolu s vnořenými SQL/XML dotazy. Bližší popis [2].

Tvorba dotazů je velmi složitá, ale algoritmicky popsatelná. Problémem bude rychlost takto „strojově“ vytvořených dotazů. Např. několik do sebe vnořených

`for`-cyklů s SQL/XML dotazy do relačních tabulek představuje velký nárok na čtení dat z disku.

V systému DB2 verze 9 byla velkým problémem nemožnost použití XQuery proměnných uvnitř fce. `db2-fn:sqlquery`. Ve verzi 9.5 již je tato možnost zabudována. Bez možnosti selekce podle XQuery proměnné již na úrovni SQL dotazu ve funkci `db2-fn:sqlquery`, by se vždy řádek po řádku převáděla data do formátu XML a selekce se prováděla až na úrovni XQuery dotazu. Například pokud by se měla nalézt data s id = 10000, načítaly by se postupně všechny řádky uložené na disku před hledanými daty a porovnávalo by se jejich id s požadovaným id. Tím by zůstala většina výhod, které nám poskytuje relační databázový systém nevyužita.

1. Načtení dat z relační tabulky bez možnosti použít XQuery proměnnou uvnitř fce. `db2-fn:sqlquery`

```
xquery
let $id_film = 10
for $film in db2-fn:sqlquery('
select xmlelement(name "film",
    xmlelement(name "id_film", f.id_film),
    xmlelement(name "nazev", f.nazev),
    xmlelement(name "rok", f.rok)
     )
from Film f')[id_film < $id_film]
return (
...
)
```

2. Načtení dat z relační tabulky s možností použít XQuery proměnnou uvnitř fce. `db2-fn:sqlquery`

```
xquery
let $id_film = 10
for $film in db2-fn:sqlquery('
select xmlelement(name "film",
    xmlelement(name "id_film", f.id_film),
    xmlelement(name "nazev", f.nazev),
    xmlelement(name "rok", f.rok)
     )
from Film f where f.id_film < parameter(1)',
$id_film)
return (
...
)
```

5.10.1 Základní myšlenka konstrukce dotazu

Celý dotaz je psán v jazyce XQuery. Pro získávání dat z relačních tabulek je využívána funkce `db2-fn:sqlquery`.

Postupujeme v XSEM-H diagramu od kořenového uzlu k listům pomocí rekurze. Vezmeme hierarchickou projekci $R^{T_1,\ldots,T_k}[P \rightarrow Q]$ určující hranu XSEM-H

diagramu. V SQL/XML dotazu provedeme spojení tabulky pro R s tabulkou potomka hierarchické projekce Q přes primární klíč tabulky pro Q a cizí klíč tabulky pro R, který do tabulky pro Q odkazuje. Navíc do dotazu přidáme selekci pomocí klíče tabulky rodiče hierarchické projekce P a klíčů kontextu projekce $T_1, \ldots, T_k$. Rekurzivně pokračujeme na další hranu XSEM-H diagramu. Využíváme algoritmus prohledávání do hloubky (depth-first search).

Klastr uzlů řešíme na úrovni XQuery pomocí `if-then-else` klauzule. Klastr hran pomocí klauzule `ORDER BY` na sloupci `impl_uspor` v SQL dotazu. Právě rekonstrukce smíšeného obsahu XML dokumentu, který klastr hran reprezentuje, je velmi časově náročná operace a odhaluje tak slabé místo navrženého logického schématu.

5.11 Shrnutí

Ukládání dat popsaných pomocí XSEM modelu do čistě relační databáze je možné, ale klade jisté nároky na použitý databázový systém. Ten musí hlavně podporovat SQL/XML a XQuery dotazy.

Výhody:

- velká obecnost, lze zkonstruovat jakýkoliv dotaz, tj. pro jakýkoliv XSEM-H diagram, se stejnou „složitostí“ dotazu. Je tedy možné dotazy konstruovat i později beze změn tabulek, ve kterých jsou data uložena.
- lépe zvládnutá technologie relačních databází než nativních XML databází, lepší možnost indexování

Nevýhody:

- velké množství tabulek → velké množství spojování tabulek
- velká složitost tvorby dotazů
- těžká optimalizovatelnost dotazů
- možné velké množství `NULL` hodnot v tabulkách klastrů
- nepřirozená forma uložení smíšeného obsahu, která způsobuje složitost jeho zpětné rekonstrukce

Kapitola 6

Převod do relací s využitím nativního XML úložiště pro spojovací klastr

Řešení popsané v předchozí kapitole má jednu zásadní slabinu v uložení smíšeného obsahu. Používá k tomu dost nepřirozenou konstrukci přes tabulku s odkazy do tabulek typů datových uzlů a tabulek dalších komponent spojovacího klastru, který smíšený obsah v XSEM modeluje. Tato tabulka obsahuje velké množství `NULL` hodnot a navíc se data z ní musí při konstrukci smíšeného obsahu třídit podle atributu `impl_uspor`.

Jedním z možných řešení pro tento problém je využít nativního uložení XML dat. Tento způsob uložení je přirozenější než umělá konstrukce pomocí tabulky a lépe odpovídá struktuře ukládaných dat. Zjednoduší se tím i dotazování pomocí XQuery.

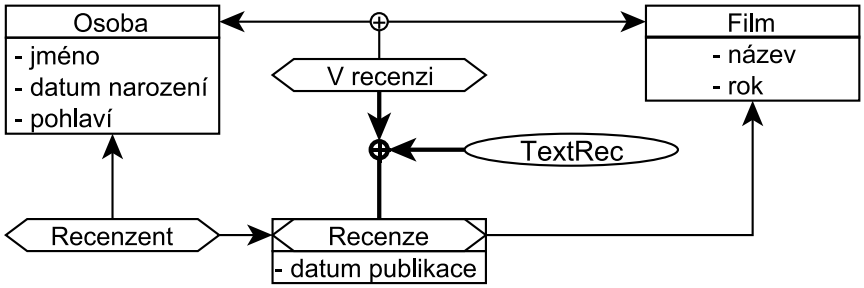
Pro entity, relace a komponentní klastry tedy budou vytvořeny stejné tabulky jako při tvorbě čistě relačního schématu popsaného v kapitole 5. Jediná změna bude v uložení spojovacího klastru a typu datového uzlu.

6.1 Uložení spojovacího klastru a typu datového uzlu

Pro spojovací klastr se nebude vytvářet samostatná tabulka, místo ní se k rodiči spojovacího klastru připojí sloupec datového typu `XML`. Tento sloupec bude mít povinně název `mixed_content` a bude mít přesně specifikovaný obsah. Tabulka se nebude vytvářet ani pro typ datového uzlu, protože jeho obsah bude uložen přímo ve sloupci `mixed_content` odpovídajícího spojovacího klastru, jehož je komponentou.

Sloupec `mixed_content` bude dobře formovaný XML dokument s kořenovým elementem `<content>`. Jeho obsah budou tvořit hodnoty typů datových uzlů přiřazených ke spojovacímu klastru, který tento sloupec reprezentuje, společně s odkazy do tabulek relací a slabých entitních typů, které jsou k tomuto klastru připojeny. Odkazy budou uvedeny v elementech pojmenovaných názvem relační tabulky odpovídající odkazovanému vztahu/slabé entitě. Pokud pro vztah neexistuje samostatná tabulka, tj. tabulka vztahu je sloučena s tabulkou komponent-

ního klastru nebo entity, odkaz směřuje do tabulky, se kterou je tabulka vztahu sloučena. Tento formát uložení nám zároveň zajišťuje implicitní uspořádání komponent spojovacího klastru.



Obrázek 6.1: Spojovací klastr

Následuje příklad obsahu sloupce `mixed_content` pro tabulku Recenze. Odkazy nevedou do tabulky relace, ale do tabulky komponentního klastru, se kterým je tabulka relace sloučena.

```
<content>
...
Tento film je přímým pokračováním úspěšné komedie <componentc>12</
    componentc>.
Hlavní roli v něm ztvárnil známý anglický komik <componentc>37</
    componentc>.
...
</content>
```

6.2 Zajištění integrity dat sloupce `mixed_content`

Jelikož sloupec `mixed_content` obsahuje odkazy do jiných tabulek, je nutné zajistit správnost těchto odkazů. Vzhledem k uložení v datovém typu `XML` je to velmi složitý úkol. Současné systémy pro to nenabízejí příliš nástrojů. Přesto zde některé možnosti budou předvedeny, nebudou však probrány příliš do hloubky.

6.2.1 XML schéma pro sloupec `mixed_content`

Naprostou nutností pro udržení integrity dat je XML schéma pro obsah sloupce `mixed_content`. To by mělo být součástí každého navrženého logického schématu databáze. Pro zápis schématu byl vybrán jazyk XML Schema [6]. Je to totiž jediný schéma jazyk v současnosti podporovaný systémem DB2.

Příklad schématu sloupec `mixed_content` tabulky *Recenze* z obrázku 6.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="content">
  <!-- Pokud je ke spojovacímu klastru připojen typ datového uzlu musí
      být atribut mixed nastaven na true -->
```

```
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <!-- Zde jsou definovány všechny elementy pro komponenty
          spojovacího klastru, které nejsou typu datový uzel -->
      <xs:element name="componentc" type="xs:integer"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
</xs:schema>
```

6.2.2 Validace vůči XML schématu

V následujících řádcích bude představeno několik technických detailů, které by měly zjednodušit práci při zkoušení navržených řešení. Všechny následující příklady jsou otestovány v systému IBM DB2. S jistými obměnami by mohly fungovat i na jiných databázových systémech podporujících uložení dat v datovém typu `XML` (např. Oracle). Podrobný popis příkazů se nachází v dokumentaci k systému DB2 [2].

Pro validaci XML dokumentů vůči navrženým schématům je nutné nejprve schémata zaregistrovat v systému následujícím příkazem.

```
REGISTER XMLSCHEMA 'http://xsem.cuni.cz/recenze.mixed_content'
FROM 'file:///C:\schema.xsd'
AS Recenze.mixed_content COMPLETE;
```

Validace dokumentů vůči registrovanému schématu se provádí pomocí funkce `XMLVALIDATION`. Nejlépe je tuto funkci volat v triggeru vyvolávaném před vložení dat do tabulky (before insert triggeru). Případně i v triggeru vyvolávaném při změně dat v tabulce (before update trigger).

```
CREATE TRIGGER NEW_RECENZE_MIXED_CONTENT NO CASCADE BEFORE INSERT ON
   RECENZE
  REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  WHEN ( N.MIXED_CONTENT IS NOT VALIDATED )
     SET (N.MIXED_CONTENT) = XMLVALIDATE(N.MIXED_CONTENT
     ACCORDING TO XMLSCHEMA ID Recenze.mixed_content);
```

Aby byla referenční integrita dodržena dokonale, je třeba ještě ověřit platnost odkazů ze sloupce `mixed_content` do tabulek odkazovaných entit případně relací. Zde již nejde použít trigger, protože DB2 nedovoluje v triggeru manipulovat s přechodovou (transition) proměnnou datového typu `XML`. Povolené je pouze její nastavení pomocí `SET` nebo validace pomocí `XMLVALIDATE`. Bližší podrobnosti v [2]. Jediným možným řešením je vytvořit funkci, která by ověření platnosti odkazů provedla před provedením příkazu `INSERT`. Nevýhodou je, že takovouto funkci lze obejít a data vložit bez ověření. To by se dalo vyřešit pomocí práv, kdy by uživatel měl pouze právo spustit takovouto proceduru, ale neměl by právo provést příkaz INSERT.

Následuje ukázka, jak by taková procedura mohla vypadat:

```
CREATE PROCEDURE CHECK_REF_INTEGRITY (IN param1 XML, OUT param2 INT)
READS SQL DATA
```

```
LANGUAGE SQL
BEGIN
  DECLARE res XML;

  SET res = XMLQUERY('for $e in $text/content/child::element()
           return (
             if ( fn:compare(fn:local-name($e), "componentc") = 0 )
             then (
             let $c := db2-fn:sqlquery("
             select XMLELEMENT(name ""c"", count(*))
             from ComponentC
             where id_componentc = parameter(1)", $e/text()
             )
             return (
             <d>{
               if ( $c/text() = 1 )
               then (
                 <ok/>
               ) else (
                 <error/>
               )
             }</d>)) else (
             )
           )' PASSING BY REF param1 AS "text");
  IF XMLEXISTS('$c/error' PASSING BY REF res AS "c") THEN
    SET param2 = 0;
    SIGNAL SQLSTATE '75001'
          SET MESSAGE_TEXT = 'Constraint violation in XML data type';
  END IF;
END
```

Uvedená PL/SQL procedura má spíše informativní charakter a poskytuje návod jak problém řešit. Zajištění správnosti odkazů ze sloupce `mixed_content` nejde řešit na úrovni logického schématu, ale až při vkládání dat do databáze. To však je již mimo rozsah této práce.

6.3 Algoritmus konstrukce tabulek

Vzhledem k předpokládanému velkému množství tabulek a množství cizích klíčů v nich je třeba tabulky vytvářet ve správném pořadí. Proto je zde uveden algoritmus konstrukce tabulek. Tabulky se vytvářejí podle dříve popsaných pravidel.

```
S := množina všech silných entitních typů;
W := množina všech slabých entitních typů;
D := množina všech typů datového uzlu;
I := množina všech spojovacích klastrů;
O := množina všech komponentních klastrů;
R := množinu všech relací;

while ( !empty(S) ){
  S := S - s;
```

```
  vytvoř pro s relační tabulku;
  E := E + s;
}

E1 := 0;

while ( E1 != E ){
  E1 := E;

  //SLABE ENTITNI TYPY
  while ( W obsahuje w, který má všechny své determinanty v E ){
    W := W - w;
    vytvoř relační tabulku pro w;
    E := E + w;
  }

  //KOMPONENTNI KLASTRY
  while ( O obsahuje o, který má všechny komponenty v E ){
    O := O - o;
    vytvoř relační tabulku pro o;
    E := E + o;
  }

  //SPOJOVACI KLASTRY
  while ( I obsahuje i, který má svého rodiče v E ){
    I := I - i;
    připoj k tabulce rodiče sloupec mixed_content datového typu XML;
    E := E + i;
  }

  //TYPY DATOVÉHO UZLU
  while ( D obsahuje d, který má svého rodiče v E ){
    D := D - d;
    if ( d připojen k rodiči přímo ){ //ne přes spojovací klastr
      připoj k tabulce rodiče sloupec s názvem d datového typu LONG
          VARCHAR;
    }
    E := E + d;
  }

  //VZTAHOVE TYPY
  while ( R obsahuje r, který má všechny účastníky v E ){
    R := R - r;
    if ( r je binární && (jeden z účastníku r je komponentní klastr (u1)
        a druhý účastník (u2) je připojen přes spojovací klastr ) ){
      k tabulce pro u1 přidej všechny atributy r;
      k tabulce pro u1 přidej cizí klíč odkazující do tabulky u2;
    } else if ( r je binární && ( kardinalita k jednomu účastníku (u1)
        je (1, 1) ) ){
      k tabulce pro u1 přidej všechny atributy r;
```

```
      k tabulce pro u1 přidej cizí klíč odkazující do tabulky druhého
         účastníka (u2);
    } else {
      vytvoř tabulku pro r;
    }
    E := E + r;
  }
}
```

6.4 Indexy v tabulkách

Pro větší rychlost zpracování dotazů je opět nutné použít indexy na cizí klíče v tabulkách. Index na datovém typu `XML` používat nemusíme, protože ho procházíme sekvenčně.

6.5 Konstrukce dotazu

Konstrukce dotazu je velmi podobná konstrukci dotazu z dat uložených v čistě relační databázi. Jediná změna v logickém schématu nastala v uložení spojovacího klastru. Zde již je „kostra“ výsledku přímo uložena v tabulce rodiče spojovacího klastru. Pouze je třeba nahradit odkazy do tabulek účastníků klastru odpovídajícími obsahy elementů. Tento algoritmus bude popsán podrobněji v 6.5.3. Předtím ještě bude rozebráno několik problémů, které při konstrukci dotazů vyvstaly.

6.5.1 Způsob převodu dat z relačních tabulek do XML

Při konstrukci dotazů pro XSEM-H diagramy je nutné převést data z relačních tabulek do XML. Hodnoty ze sloupců budou uzavřeny v elementech s názvem sloupce. Navíc hodnoty z jednoho řádku tabulky budou uzavřeny do elementu s názvem tabulky. Využije se k tomu rozšíření SQL/XML jazyka SQL, konkrétně funkce `XMLELEMENT`. Důležité je nastavení parametru `ON NULL` této funkce. To musí být explicitně nastaveno na `NULL ON NULL`. Při tomto nastavení se pro hodnotu `NULL` nebude vytvářet element. Implicitně je tento parametr nastaven na hodnotu `EMPTY ON NULL`, toto nastavení pro hodnotu `NULL` vytvoří prázdný element.
Příklad:

```
select
  xmlelement(name "soutez",
    xmlelement(name "id_soutez", s.id_soutez OPTION NULL ON NULL),
    xmlelement(name "zeme", s.zeme OPTION NULL ON NULL),
    xmlelement(name "nazev", s.nazev OPTION NULL ON NULL),
    xmlelement(name "rok", s.rok OPTION NULL ON NULL)
  )
from Soutez s
```

6.5.2 Porovnání rychlosti spojení tabulek na úrovni SQL a XQuery

Při konstrukci dotazu je nutné spojovat tabulky pomocí cizích klíčů. Jelikož dotazy jsou psány v jazyce XQuery [7], ale zároveň se data získávají z tabulek SQL dotazy přes funkci `db2-fn:sqlquery`, můžeme spojení tabulek provádět jak na úrovni SQL tak i XQuery. Byly provedeny testy obou možností a překvapivě jsou zhruba stejně rychlé.

Pro měření rychlosti byl vybrán dotaz, který pro všechny osoby s id < 10000 vypíše ve formátu XML seznam filmů, které režírovali.

1. Spojení tabulek provádíme až na úrovni XQuery. Tj. převádíme do XML jak tabulku relace, tak odkazované entity. U odkazované entity, ale převádíme již jen data vyhovující spojení s tabulkou relace. Není to tedy úplně primitivní spojení na úrovni XQuery, kdy by se převáděly do XML všechny řádky jak tabulky relace, tak entity a spojovala by se až data převedená do XML.

```
xquery
for $osoba in db2-fn:sqlquery('
select xmlelement(name "osoba",
    xmlelement(name "id_osoba", o.id_osoba),
    xmlelement(name "jmeno", o.jmeno),
    xmlelement(name "datum_narozeni", o.datum_narozeni),
    xmlelement(name "pohlavi", o.pohlavi)
     )
from Osoba o where o.id_osoba < 10000
')
return (
<osoba>
{
  $osoba/jmeno,
  $osoba/datum_narozeni,
  $osoba/pohlavi,
  (: čtu tabulku relace :)
  for $reziser in db2-fn:sqlquery('
  select xmlelement(name "reziser",
      xmlelement(name "id_film", r.id_film),
      xmlelement(name "id_osoba", r.id_osoba),
      xmlelement(name "id_reziser", r.id_reziser)
       )
  from Reziser r
  where r.id_osoba = parameter(1)',
  $osoba/id_osoba)
  (: provádím spojení tabulky entity s tabulkou relace :)
  for $film in db2-fn:sqlquery('
  select xmlelement(name "film",
      xmlelement(name "id_film", f.id_film),
      xmlelement(name "nazev", f.nazev),
      xmlelement(name "rok", f.rok)
       )
  from Film f
```

```
  where f.id_film = parameter(1)',
  $reziser/id_film)
  return (
    $film
  )
}
</osoba>
)
```

2. Spojení provádíme na úrovni SQL. Tj. do XML převádíme jen tabulku odkazované entity.

```
xquery
for $osoba in db2-fn:sqlquery('
select xmlelement(name "osoba",
    xmlelement(name "id_osoba", o.id_osoba),
    xmlelement(name "jmeno", o.jmeno),
    xmlelement(name "datum_narozeni", o.datum_narozeni),
    xmlelement(name "pohlavi", o.pohlavi)
     )
from Osoba o where o.id_osoba < 10000
')
return (
<osoba>
{
  $osoba/jmeno,
  $osoba/datum_narozeni,
  $osoba/pohlavi,
  (: čtu pouze tabulku entity, spojení s tabulkou relace provádím
     uvnitř SQL dotazu :)
  for $film in db2-fn:sqlquery('
  select xmlelement(name "film",
      xmlelement(name "id_film", f.id_film),
      xmlelement(name "nazev", f.nazev),
      xmlelement(name "rok", f.rok)
       )
  from Film f, Reziser r
  where f.id_film = r.id_film AND r.id_osoba = parameter(1)',
  $osoba/id_osoba)
  return (
    $film
  )
}
</osoba>
)
```

Každý dotaz byl proveden 10krát.

Z výsledků v tabulce 6.1 je zřejmé, že obě možnosti jsou téměř stejně rychlé. Pro konstrukci dotazů byla vybrána druhá možnost, kdy jsou tabulky spojovány na úrovni SQL.

Tabulka 6.1: Rychlosti dotazů

	1. dotaz	**2. dotaz**
1	15640 ms	14718 ms
2	14360 ms	14250 ms
3	14531 ms	14672 ms
4	14328 ms	14485 ms
5	15250 ms	14078 ms
6	14188 ms	14375 ms
7	14187 ms	14531 ms
8	14391 ms	14375 ms
9	14297 ms	14281 ms
10	14531 ms	14235 ms
AVG	14570 ms	14400 ms

6.5.3 Algoritmus konstrukce dotazu

Stejně jako při konstrukci dotazu nad čistě relačním schématem bude konstrukce probíhat rekurzivním průchodem XSEM-H diagramu do hloubky. Postupně budou procházeny hierarchické projekce určující hrany XSEM-H diagramu a podle typu objektů, na kterých je projekce definována, bude tvořen XQuery dotaz. Obecně pro každou hierarchickou projekci $R^{T_1,\ldots,T_k}[P \rightarrow Q]$ bude postupováno následovně:

1. V SQL dotazu q se provede spojení tabulek pro R a Q. Pokud $R = Q$ tak se tabulky nespojují, protože jsou shodné, a použije se jedna z nich. Tento SQL dotaz bude převádět do XML data z tabulky pro objekt Q.

2. Pokud hrana odpovídající projekci nevede z kořenového uzlu XSEM-H diagramu, pak se do dotazu q přidá selekce podle klíče tabulky pro objekt P.

3. Pokud je kontext $T_1, \ldots, T_k$ neprázdný, přidá se do dotazu q ještě selekce pomocí klíčů tabulek pro objekty $T_1, \ldots, T_k$.

4. Vypíší se atributy objektu Q načtené v dotazu q, které nejsou ani primární klíčem ani cizími klíči.

5. Pokud uzel reprezentující Q není listem XSEM-H diagramu, postupuje se rekurzivně na další hierarchickou projekci. Jinak rekurzivní návrat.

Názornější bude ukázka XQuery kódu vzniklého při zpracování hierarchické projekce $R^{T_1,\ldots,T_k}[P \rightarrow Q]$.

```
for $q in db2-fn:sqlquery('
select
xmlelement(name q,
  xmlelement(name atribut1, q.atribut1 OPTION NULL ON NULL),
  ...
  xmlelement(name atributN, q.atributN OPTION NULL ON NULL)
)
from Q q, R r
```

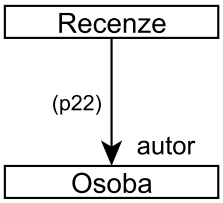
```
where q.id_q = r.id_r AND r.id_p = parameter(1) AND
  r.id_t1 = parameter(2) AND ... AND r.id_tN = parameter(N+1)',
$p/id_p, $t1/id_t1, ..., $tN/id_tN)
return (
  $q/atribut1,
  ...
  $q/atributN,
  (: rekurzivní zpracování další hierarchické projekce :)
)
```

Všechny selekce mohou být provedeny, protože podle definice hierarchické projekce musí být jak $T_1, \ldots, T_k$ tak i P a Q účastníky vztahu R (R nemusí být jen vztahový typ, ale platí to i pro slabý entitní typ a typ datového uzlu). Dále z definice XSEM-H pohledu vyplývá, že v XSEM-H diagramu musí hraně s projekcí $R^{T_1,\ldots,T_k}[P \to Q]$ předcházet hrany s projekcemi ve tvaru $R[T_1 \to T_2]$, $R^{T_1}[T_2 \to T_3]$, ..., $R^{T_1,\ldots,T_{k-1}}[T_k \to P]$ v pořadí směrem od kořene k listům. Při vytváření dotazu tedy již tyto hierarchické projekce byly zpracovány a data z relačních tabulek pro objekty P, $T_1, \ldots, T_k$ byla načtena a uložena v proměnných jazyka XQuery.

V následujících odstavcích budou podrobně rozebrány hierarchické projekce podle typů objektů, které se jich účastní. Většina projekcí bude použita z diagramu 4.4.

Hierarchická projekce bez kontextu, kdy rodič i potomek nejsou ani spojovací ani komponentní klastr

$$\text{Recenzent}[\text{Recenze} \to \text{Osoba}] \quad \text{(p22)}$$



Obrázek 6.2: Hierarchická projekce (p22)

V tomto případě se postupuje přesně podle obecného návodu uvedeného výše. Spojí se tabulka pro vztahový typ *Recenzent* s tabulkou pro entitní typ *Osoba*. Toto spojení se navíc omezí klíčem tabulky rodiče projekce, slabého entitního typu *Recenze*. Jelikož uzel hierarchického pohledu *Osoba* má specifikované označení, uzavřou se vypisované atributy entitního typu *Osoba* do elementu s názvem označení. Uzel *Osoba* je listovým uzlem XSEM-H pohledu, algoritmus prohledání do hloubky se tedy vrací po hraně zpět. Fragment XQuery dotazu by v tomto případě vypadal následovně.

```
for $osoba in db2-fn:sqlquery('
select xmlelement(name "osoba",
    xmlelement(name "id_osoba", a.id_osoba OPTION NULL ON NULL),
```

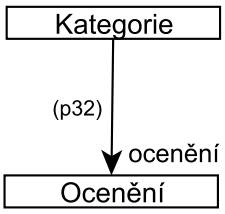
```
    xmlelement(name "jmeno", a.jmeno OPTION NULL ON NULL),
    xmlelement( name "datum_narozeni", a.datum_narozeni OPTION NULL ON
        NULL),
    xmlelement( name "pohlavi", a.pohlavi OPTION NULL ON NULL))
from osoba a, recenze b
where a.id_osoba = b.id_osoba AND b.id_recenze = parameter(1)',
$recenze/id_recenze)
return (
<autor>
{
  $osoba/jmeno,
  $osoba/datum_narozeni,
  $osoba/pohlavi
}
</autor>
)
```

Tabulka vztahového typu *Recenzent* je zde prezentována tabulkou entity *Recenze*, přesně podle pravidla o spojení tabulek vztahového typu a entitního typu s kardinalitou $(1, 1)$.

Hierarchická projekce s kontextem, kdy rodič i potomek nejsou ani spojovací ani komponentní klastr

Pohled na obrázku 4.4 takovouto projekci neobsahuje, je tedy použita projekce (p32) z pohledu 4.3.

$$\text{Ocenění}^{\text{Soutěž}}[\text{Kategorie} \rightarrow \text{Ocenění}] \quad (\text{p32})$$



Obrázek 6.3: Hierarchická projekce (p32)

Hierarchická projekce obsahuje kontext *Soutěž*, je tedy třeba SQL dotaz omezit i pomocí primárního klíče tabulky pro entitu *Soutěž*. Jinak se postupuje naprosto stejně jako v případě bez kontextu. Pokud by se kontext skládal z více objektů, přidá se k SQL dotazů více selekcí podle kontextu.

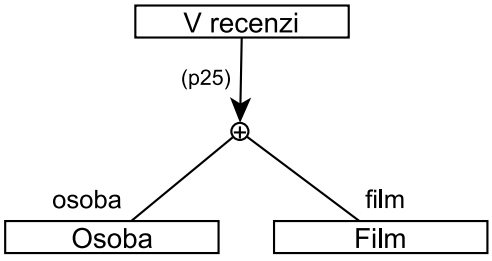
```
for $oceneni in db2-fn:sqlquery('
select
xmlelement(name "oceneni",
  xmlelement(name "id_oceneni", a.id_oceneni OPTION NULL ON NULL),
  xmlelement(name "id_kategorie", a.id_kategorie OPTION NULL ON NULL),
  xmlelement(name "id_soutez", a.id_soutez OPTION NULL ON NULL),
  xmlelement(name "id_osoba", a.id_osoba OPTION NULL ON NULL),
```

```
  xmlelement(name "id_film", a.id_film OPTION NULL ON NULL)
)
from oceneni a
where a.id_kategorie = parameter(1) AND a.id_soutez = parameter(2)',
$kategorie/id_kategorie, $soutez/id_soutez)
return (
<oceneni>
{
 (: rekurzivní zpracování další hrany :)
}
</oceneni>
)
```

Vztahový typ *Ocenění* nemá žádné atributy, proto se žádné v dotazu nevypisují.

Hierarchická projekce, kdy potomek je komponentní klastr

$$\text{V recenzi}^{\text{Recenze}}[\text{V recenzi} \rightarrow \text{Osoba+Film}] \quad \text{(p25)}$$



Obrázek 6.4: Hierarchická projekce (p25)

Komponentní klastr je uložen v tabulce s cizími klíči, které odkazují do tabulek komponent klastru. V jednom řádku je různá od hodnoty `NULL` pouze jedna z hodnot cizích klíčů. Této vlastnosti bude v XQuery dotazu využito. Prvním krokem dotazu je spojení tabulky komponentního klastru *Osoba+Film* a vztahového typu V recenzi a převod dat z tabulky klastru do XML.

Při načítání tabulky komponentního klastru musí být dodrženo přesné pořadí elementů odpovídajících sloupcům tabulky. Prvním elementem je primární klíč převáděné tabulky, následují cizí klíče do tabulek komponent klastru. Nastavení parametru funkce `XMLELEMENT` na `NULL ON NULL` zaručí, že řádek tabulky komponentního klastru převedený do XML bude mít pouze 2 elementy: element s primárním klíčem a element se jménem komponenty klastru obsahující cizí klíč do tabulky komponenty klastru. Poté je již snadné rozlišit pomocí klauzule `if-then-else` o jakou komponentu se jedná.

Po rozlišení komponenty se bude dále pokračovat zpracováním hierarchické projekce, kde komponentní klastr na pozici potomka projekce nahradí vybraná komponenta klastru. Tímto se rozpadne hierarchická projekce do tolika hierarchických projekcí, jaký je počet komponent původního klastru. Projekce (p25) se tedy rozpadne do následujících dvou projekcí:

$$\text{V recenzi}^{\text{Recenze}}[\text{V recenzi} \rightarrow \text{Osoba}] \quad \text{(p25a)}$$
$$\text{V recenzi}^{\text{Recenze}}[\text{V recenzi} \rightarrow \text{Film}] \quad \text{(p25b)}$$

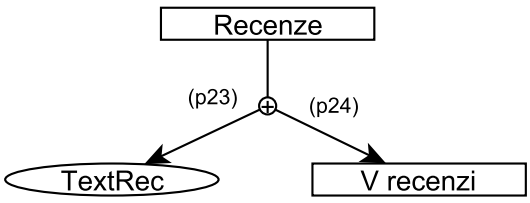
```
for $componentc in db2-fn:sqlquery('
select xmlelement(name "componentc",
    xmlelement(name "id_componentc", a.id_componentc OPTION NULL ON NULL
        ),
    xmlelement(name "id_osoba", a.id_osoba OPTION NULL ON NULL),
    xmlelement(name "id_film", a.id_film OPTION NULL ON NULL))
from componentc a
where a.id_componentc = parameter(1) AND a.id_recenze = parameter(2)',
$componentc1/id_componentc, $recenze/id_recenze)
return (
  if ( fn:compare(fn:local-name($componentc/child::element()[2]), "
     id_osoba") = 0 )
  then (
    (: rekurzivní zpracování projekce (p25a) :)
  ) else ( ),
  if ( fn:compare(fn:local-name($componentc/child::element()[2]), "
     id_film") = 0 )
  then (
    (: rekurzivní zpracování projekce (p25b) :)
  ) else ( )
)
```

Zde stojí za povšimnutí, že tabulka relace *V recenzi* je stejná jako tabulka komponentního klastru. Přesně podle pravidla o spojení tabulky relace a komponentního klastru, pokud druhým účastníkem relace je spojovací klastr.

Hierarchické projekce, kdy jsou hrany v XSEM-H digramu připojeny k rodiči pomocí klastru hran

Klastr hran v XSEM-H diagramu odpovídá spojovacímu klastru v XSEM-ER diagramu, tedy uložení dat ve sloupci datového typu `XML`.

$$\text{TextRec}[\text{Recenze} \rightarrow \text{TextRec}] \quad \text{(p23)}$$
$$\text{V recenzi}[\text{Recenze} \rightarrow \text{V recenzi}] \quad \text{(p24)}$$



Obrázek 6.5: Hierarchické projekce (p23) a (p24)

Při zpracovávání klastru hran je jediným úkolem rozparsovat uložený XML dokument ze sloupce `mixed_content` a nahradit případné odkazy do tabulek komponent spojovacího klastru, kterému klastr hran odpovídá. Opět se využije klauzule

`if-then-else`. Tentokrát se bude rozlišovat, zda uzel XML dokumentu načteného ze sloupce `mixed_content` je element, případně jaký má název. Pokud elementem nebude, tak bude rovnou vypsán, protože to bude obsah typu datového uzlu ke spojovacímu klastru připojenému.

```
let $content := db2-fn:sqlquery('
select MIXED_CONTENT from recenze where id_recenze = parameter(1)',
    $recenze/id_recenze)
for $node in $content/content/child::node()
return (
  if ( fn:empty($node/self::element()) )
  then (
    (: obsah typu datového uzlu, vlastně zpracováni projekce (p23) :)
    $node/self::text()
  ) else ( ),
  if ( fn:compare(fn:local-name($node/self::element()), "componentc") =
      0 )
  then (
    (: rekurzivní zpracování projekce (p24) :)
  ) else ( )
)
```

6.6 Pohledy vytvořené pomocí XQuery dotazů

Z XQuery dotazů se také dají vytvořit klasické SQL pohledy (VIEW). Stačí k tomu použít následující konstrukci.

```
CREATE VIEW nazev-pohledu (nazev-sloupce) AS
SELECT XMLQUERY('xquery dotaz') FROM SYSIBM.SYSDUMMY1
```

Velkým problémem je dotazování na pohled, kdy systém nedokáže dotaz na pohled nijak zapracovat do XQuery dotazu, který definuje tento pohled. Systém tedy nejdříve vykoná celý XQuery dotaz definující pohled a až poté se nad vygenerovaným XML dokumentem dotazuje.

Tato vlastnost systému velmi omezuje použití pohledů definovaných XQuery dotazy vygenerovaných na základě XSEM-H diagramů. Každý XQuery dotaz odpovídající XSEM-H diagramu totiž implicitně vypisuje všechna data z tabulek objektů vyskytujících se v diagramu. To ale při větším množství dat představuje časově velmi náročnou operaci. Pokud by se celá tato operace musela provádět při každém sebejednodušším dotazu na pohled, pak rychlost vrácení výsledku dotazu systémem by byla v praxi nepoužitelná.

Rozumnější se jeví možnost napsat sadu funkcí, které na základě vloženého parametru upraví XQuery dotaz odpovídající XSEM-H diagramu. Příkladem by mohlo být omezení XQuery dotazu pro diagram na obrázku 4.2 na osoby s id < 100, případně s id $= 100$. Tato se selekce by se přidala do části XQuery dotazu, kde se načítají data pro entitu *Osoba* z relační tabulky do XML. Rozumné je tyto selekce používat pouze pro objekty, které jsou kořeny XSEM-H diagramů. Ale je možné je použít i jinde.

6.7 Shrnutí

Výhody:

- výrazné zvýšení rychlosti dotazů na spojovací klastr oproti uložení spojovacího klastru v relační tabulce, zrychlení je až 20násobné
- méně tabulek, tedy i méně spojení při dotazech
- využití stabilní a efektivní technologie uložení dat do relačních tabulek spolu s přirozeným využitím nativního uložení XML dat pro data se smíšeným obsahem

Nevýhody:

- nutnost podpory uložení nativního XML od databázového systému
- `NULL` hodnoty v tabulce komponentního klastru
- složité a těžko optimalizovatelné dotazy

Kapitola 7

Prototypová implementace

Pro implementaci bylo vybráno druhé výrazně lepší řešení. Cílem této ukázkové implementace především bylo ověřit správnost tvorby XQuery dotazů, které vygenerují XML dokumenty, jejichž struktura je popsaná v XSEM-H diagramech. V této kapitole bude popsáno ovládání programu a představena základní architektura řešení.

Pro zápis kódu programu byl vybrán jazyk Java. Jeho hlavní výhodou je jednoduchá práce s řetězci, na které je celý program založen. Program nemá žádné grafické rozhraní a ovládá se pomocí parametrů zadaných při spouštění. Program je zkompilován ve verzi 1.6 jazyku Java.

7.1 Ovládání programu

Program se spouští z příkazového řádku pomocí příkazu:

```
java -jar SchemaAndQueryGeneration.jar input.xml [-o OutputDir]
```

Parametr `input.xml` je povinný a tento soubor musí obsahovat serializovanou podobu XSEM konceptuálního schématu. Tento soubor musí být validní vůči navrženému schématu uvedenému v příloze C. Validita vůči schématu však nezaručuje úplnou správnost přepisu XSEM diagramu do XML. Zejména správnost odkazů, které jsou řešené pouze pomocí atributů `ID` a `IDREF`, je velmi důležitá. Při nesprávném vstupu není chování programu specifikované.

Parametr `-o OutputDir` je nepovinný a je pomocí něj možné určit adresář, do kterého se budou ukládat soubory vygenerované programem. Tyto soubory tvoří výstup programu. Pokud adresář s udaným jménem neexistuje, program ho vytvoří. Jestliže není parametr `-o` uveden, jsou výstupní soubory uloženy do adresáře, ze kterého je program spuštěn.

Výstupem programu je několik souborů. Jejich počet záleží na počtu namodelovaných XSEM-H pohledů. Hlavním souborem je soubor `DatabaseSchema.sql`. Ten obsahuje SQL příkazy pro vytvoření tabulek výsledného logického schématu. Navíc jsou v něm obsaženy příkazy, které vytvoří indexy na všech cizích klíčích v tabulkách. Jednotlivé příkazy jsou oddělené ‘`;`’. Tento soubor je možné přímo použít k vytvoření tabulek v databázi.

Dalšími výstupy jsou soubory pojmenované `schemaXX.xsd`, kde `XX` je nahrazeno číslicí. To jsou XML schémata napsaná v jazyce XML Schema pro sloupce

datového typu `XML`. V jejich těle je v komentáři uvedena přesně tabulka a sloupec, kterému odpovídají.

Poslední skupinou jsou soubory, jejichž jména jsou shodná s názvy namodelovaných XSEM pohledů, pouze je jim přidána přípona `.xq`. To jsou XQuery dotazy odpovídající jednotlivým pohledům. Tyto dotazy nejsou nijak omezeny a vypisují kompletní obsahy tabulek. Možnost přidání selekce je krátce popsána v 6.6.

7.2 Základní architektura

Celé schéma namodelované v XSEM modelu a uložené do XML se v programu načítá do instancí tříd. Ty jsou navzájem propojeny odkazy, které nahrazují hrany jak v XSEM-ER schématu, tak v XSEM-H pohledech. Pro každý objekt XSEM modelu je vytvořena speciální třída, která je potomkem abstraktní třídy `Node`. Všechny tyto třídy jsou uzavřeny v balíčku `cz.cuni.xsemstore.diagrams`.
Seznam tříd a odpovídajících objektů XSEM modelu:

Třída	**Objekt XSEM modelu**
`Entity`	Silný entitní typ
`WeakEntity`	Slabý entitní typ
`ComponentCluster`	Komponentní klastr
`ConnectionCluster`	Spojovací klastr
`Relationship`	Vztahový typ
`DataNodeType`	Typ datového uzlu
`HNode`	Uzel XSEM-H diagramu
`ClusterOfEdges`	Klastr hran
`ClusterOfNodes`	Klastr uzlů

Běh programu lze rozdělit do tří základních částí. Načtení a rozparsování vstupních dat, vytvoření logického schématu databáze a konstrukce XQuery dotazů.

Pro načítání vstupu z XML dokumentu je použito rozhraní DOM. Každý vstup je validován vůči schématu. Další kontrola správnosti vstupních dat není prováděna a nesprávná data mohou v dalším průběh zpracování zapříčinit nesprávnou funkčnost případně pád programu.

Tvorba schématu databáze je vcelku jednoduchá a je pouze přepisem algoritmu, který byl uveden v části 6.3. V této části jsou také generována schémata pro sloupce datových typů `XML`.

Poslední a nejsložitější částí je tvorba XQuery dotazů. Ta probíhá rekurzivním voláním jedné procedury, která generuje metodou *inorder* výsledný dotaz. V proceduře je běh programu řízen typem zpracovávané hierarchické projekce, přesně podle pravidel popsaných v 6.5.3.

Kapitola 8

Shrnutí a závěr

Základním úkolem této práce bylo navrhnout pravidla pro převod konceptuálního schématu vytvořeného pomocí XSEM-H modelu do logického schématu databáze. Nabízelo se několik možností jak data ukládat. Data popsaná konceptuálním schématem mohla být ukládána v XML dokumentech v nativních XML databázích, v relačních tabulkách v (objektově-)relačních databázích případně mohlo být využito obou možností, které dnešní vyspělé databázové systémy dokáží propojit v jedné databázi.

Po prozkoumání a posouzení možností bylo rozhodnuto data neukládat přímo v XML dokumentech. Uložení v XML má řadu nevýhod pro dotazování, protože je pevně daná hierarchická struktura dat a její přetransformování do jiné struktury je nesmírně složité a časově náročné. Taktéž jakákoliv optimalizace, kdy by se opakující časti struktury ukládaly odděleně do XML dokumentů a zamezilo se tím velké redundanci dat, je příliš složitá. Proto bylo rozhodnuto ukládat data ne do hierarchické struktury, ale do „ploché“ struktury a podle potřeby je do hierarchické struktury přetransformovat. Velkou inspirací v tomto byl klasický ER model a jeho způsob návrhu logického databázového schématu z konceptuálního schématu. Navrženy byly dvě možnosti tvorby logického schématu.

První popsaná možnost pro uložení dat využívá pouze čistě relační technologie. Objekty, které jsou v XSEM-ER modelu převzaté z ER modelu, jsou ukládány stejně jako u klasického ER modelu. Tedy obecně je pro každý objekt vytvořena tabulka, jejíž sloupce tvoří atributy objektu. Tabulky objektů jsou vzájemně propojeny cizími klíči. Navíc je v každé tabulce sloupec umělého klíče, který umožňuje identifikaci instancí objektů. Konstrukce pro smíšený obsah a nepravidelnou strukturu, které XSEM-ER model přidává, jsou taktéž uloženy v relačních tabulkách. Jejich nevýhodu představuje velké množství `NULL` hodnot.

Druhá metoda tvorby logického schématu vychází z první. Změnu představuje pouze jiné uložení konstrukce pro smíšený obsah. Ta se ukládá přímo do formátu XML. Toto uložení je přirozenější a lépe odpovídá struktuře dat. Pro takto navržené logické schéma byl také popsán algoritmus, jak data převést do hierarchické struktury. Převod probíhá pomocí XQuery dotazu, který je vytvářen na základě navržených XSEM-H pohledů. Ty jsou nedílnou součástí konceptuálního schématu.

Obě možnosti tvorby logického schématu byly otestovány pomocí XQuery dotazů vytvořených na základě XSEM-H pohledů. Jako výrazně lepší se ukázalo řešení využívající nativního uložení XML dat. Dotaz pro pohled obsahující smí-

šený obsah proběhl pro toto řešení 20krát rychleji než pro první navržené řešení. Z toho důvodu byla pro zkušební implementaci vybrána druhá navržená možnost tvorby logického schématu.

V průběhu práce se objevilo množství problémů. Některé byly vyřešeny, některé zůstávají nadále otevřeny. Jedním z nich je identifikace instancí objektů. Ta je u dat ve formátu XML složitější než u dat navržených pomocí klasického ER modelu. V této práci je vše řešeno pomocí uměle generovaných identifikátorů a formalismus identifikace objektů popsaný v XSEM modelu je trochu opomíjen. Na funkčnost navrženého řešení to však nemá žádný vliv. Dalším problémem je zajištění integrity dat. V současné době neexistují nástroje pro zajištění integrity mezi daty uloženými ve formátu XML a daty uloženými ve sloupcích jednoduchých datových typů. Integrita se tedy musí zajistit již při vkládání dat do databáze. Tím se však tato práce nezabývala a je to jeden ze směrů kudy se mohou ubírat další práce založené na XSEM modelu.

Závěrem lze říct, že všechny vytyčené cíle práce byly splněny. Podařilo se navrhnout algoritmus převodu konceptuálního schématu do logického schématu. Vytvořené logické schéma je dostatečně efektivní a snaží se využít veškeré v současnosti běžně dostupné databázové technologie. Ty však jsou, zejména v oblasti ukládání XML dat, zatím bohužel poněkud omezené. Jejich vývoj je ale nyní v popředí zájmu všech velkých producentů databázových systému a v průběhu vzniku této práce se objevilo několik vylepšení, která velmi usnadnila a zefektivnila navržená řešení.

Literatura

[1] Bourret, R.:*XML and Datatbases*, 2005 [http://www.rpbourret.com/xml/XMLAndDatabases.htm]

[2] *DB2 9.5 Information Center for Linux, Unix, and Windows* [http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp]

[3] Mlýnková, I. - Pokorný, J.: *XML in the World of (Object-)Relational Database Systems*, Springer Science+Business Media, Inc., 2005 [http://kocour.ms.mff.cuni.cz/~mlynkova/doc/ISD2004.pdf]

[4] Nečaský, M.: *XSEM - A Conceptual Model for XML Data*, APCCM 2007: Proc. Fourth Asia-Pacific Conference on Conceptual Modelling, Ballarat, Australia [http://necasky.net/content/publications/necasky_apccm2007.pdf]

[5] *OMG XMI Specification* [http://www.omg.org/technology/documents/formal/xmi.htm]

[6] *XML Schema* [http://www.w3.org/XML/Schema]

[7] *XQuery* [http://www.w3.org/XML/Query/]

Příloha A

Návod pro otestování výstupu programu

Na přiloženém CD je několik utilit, které pomohou při prvotním seznamování a testování výstupů generovaných programem `SchemaAndQueryGeneration`. Zde bude popsán postup, jak tyto utility využít, a zároveň bude na konkrétním příkladě ukázáno využití vygenerovaného schématu a dotazů.

Základní prerekvizitou je nainstalovaný systém IBM DB2 Express-C. Ten je volně ke stažení na adrese `http://www-306.ibm.com/software/data/db2/express/`. Návod k instalaci a konfiguraci je obsažen v dokumentaci k tomuto systému [2]. Pro správnou práci utilit je nutné přidat do systémové proměnné `CLASSPATH` cestu k Java knihovnám nainstalovaných spolu se systémem DB2. Podrobný návod je taktéž obsažen v dokumentaci systému.

Na CD je pro testování k dispozici serializovaná podoba XSEM schématu představeného v této práci na obrázcích 4.1, 4.2, 4.3 a 4.4. Toto schéma je uloženo v souboru `FilmSchema.xml`. K tomuto schématu je také k dispozici generátor dat.

Všechny následující postupy platí pro výstupy vygenerované ze souboru `FilmDiagram.xml` pomocí programu `SchemaAndQueryGeneration`.

A.1 Vytvoření databáze a nagenerování dat

Aby bylo možné použít vygenerovaný soubor `databaseSchema.sql`, je nejprve nutné vytvořit databázi. To se udělá pomocí příkazu

```
CREATE DATABASE jmeno_databaze;
```

Poté již stačí zkopírovat všechny příkazy ze souboru `databaseSchema.sql` do Editoru příkazů a spustit jejich vykonání. Po tomto kroku bude databáze připravena pro generování dat. Schémata pro sloupce datového typu `XML` není nutné využívat, protože generátor generuje validní data.

Generátor dat je uložen v souboru `FilmGen.java`. Nelze ho použít přímo, ale je nutné nastavit v jeho zdrojovém kódu adresu databáze, uživatelské jméno a heslo. Tyto údaje se nastavují do řetězcových proměnných `dbUrl`, `dbUser` a `dbPasswd`. Tento soubor je poté nutné překompilovat kompilátorem `javac` a spustit. Vygenerovaná databáze obsahuje řádově desítky tisíc záznamů. Počet záznamů je možné nastavit změnou konstant uložených v souboru `FilmGenConsts.java`.

A.2 Spouštění dotazů

Všechny vygenerované dotazy jsou uloženy v souborech s příponou `.xq`. Je možné je spouštět přímo z Editoru příkazů nebo lze využít utility `QueryBechmark.java`. V jejím zdrojovém kódu je opět nutné nastavit uživatelské jméno, heslo a adresu zde nikoliv databáze, ale pouze databázového systému, tedy bez názvu databáze na konci. K tomu opět slouží proměnné `dbUser`, `dbPasswd` a `dbUrl`. Dále je nutné nastavit počet opakování každého dotazu, a zda se má vypisovat výstup generovaný dotazem, proměnné `pocetOpakovani`, `printOutput`. Poslední proměnnou, kterou je nutné nastavit je `inputFile`. V ní je uložena cesta k souboru s dotazy, který má přesně specifikovaný obsah.

První řádek souboru s dotazy vždy obsahuje název databáze, do které se dotazy dotazují. Druhý řádek má tvar

```
---
```

Následuje první dotaz, ten může být rozdělen do několika řádků. Každý dotaz musí být ukončen řádkem

```
--===
```

Pokud chceme testovat pouze několik prvních dotazů v souboru, stačí uvést za posledním dotazem, který má ještě být vykonán, řádek

```
+++
```

Ten způsobí konec načítání dalšího obsahu souboru.

Utilita `QueryBenchmark` se hodí spíše k testování rychlosti dotazů, než k ověřování správnosti výstupu dotazu. Ještě je nutné upozornit, že Editor příkazů nezobrazuje korektně smíšený obsah XML dokumentu vytvořeného pomocí XQuery dotazu a je tedy nutné takovýto dotaz otestovat jiným způsobem, např. pomocí `QueryBenchmark`.

Příloha B

Obsah přiloženého CD

Na CD přiloženém k této diplomové práci se nachází prototypová implementace, její zdrojové kódy a elektronická podoba textu této práce.

Obsah CD:

- `/bin` — adresář obsahující JAR soubor `SchemaAndQueryGeneration.jar` prototypové implementace
- `/schema_film` — adresář obsahující serializovanou podobu XSEM schématu pro databázi filmů představenou v textu práce `FilmSchema.xml` a XML schéma `schema.xsd` pro serializaci XSEM diagramů
- `/src` — adresář se zdrojovými kódy implementace
- `/utilites` — adresář obsahující utility k testování
- `obsah.txt` — soubor s popisem obsahu CD
- `ukladani_XML_dat.pdf` — text diplomové práce ve formátu pdf

Příloha C

Schéma pro serializaci diagramů

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://xsem.cuni.cz/DiagramSerialization"
      xmlns="http://xsem.cuni.cz/DiagramSerialization"
      elementFormDefault="qualified">

  <!-- Kostra XML dokumentu -->
  <xs:element name="diagram">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="er_diagram" type="er_diagram_type" minOccurs="
           1" maxOccurs="1"/>
        <xs:element name="h_diagram" type="h_diagram_type" minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="integrity_constraints" type="
           integrity_constraints_type" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Ulozeni ER diagramu -->
  <xs:complexType name="er_diagram_type">
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="entity" type="er_entity_type"/>
        <xs:element name="relationship" type="er_relationship_type"/>
        <xs:element name="weak_entity" type="er_weak_entity_type"/>
        <xs:element name="component_cluster" type="
           er_component_cluster_type"/>
        <xs:element name="connection_cluster" type="
           er_connection_cluster_type"/>
        <xs:element name="data_node_type" type="er_data_node_type"/>
      </xs:choice>
      <xs:element name="association" type="er_association_type"
         minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
```

```
</xs:complexType>

<!-- Entitni typ -->
<xs:complexType name="er_entity_type">
  <xs:sequence>
    <xs:element name="attribute" type="er_attribute_type" minOccurs="0
        " maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>

<!-- Vztahovy typ -->
<xs:complexType name="er_relationship_type">
  <xs:sequence>
    <xs:element name="attribute" type="er_attribute_type" minOccurs="0
        " maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>

<!-- Slaby entitni typ -->
<xs:complexType name="er_weak_entity_type">
  <xs:sequence>
    <xs:element name="attribute" type="er_attribute_type" minOccurs="0
        " maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>

<!-- Komponentni klastr -->
<xs:complexType name="er_component_cluster_type">
  <xs:attribute name="id" type="xs:ID" use="required"/>
</xs:complexType>

<!-- Spojovaci klastr -->
<xs:complexType name="er_connection_cluster_type">
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="ordered" type="xs:boolean" use="required"/>
</xs:complexType>

<!-- Typ datovy uzel -->
<xs:complexType name="er_data_node_type">
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="data_type" type="data_types" default="STRING"/>
</xs:complexType>

<!-- Asociace == hrany v diagramu -->
```

```
<xs:complexType name="er_association_type">
  <xs:sequence>
    <!-- odkaz na element ER diagramu -->
    <xs:element name="begin" type="xs:IDREF"/>
    <!-- odkaz na element ER diagramu -->
    <xs:element name="end" type="xs:IDREF"/>
  </xs:sequence>
  <xs:attribute name="oriented" type="xs:boolean" use="required"/>
</xs:complexType>

<!-- Atribut entity, relace -->
<xs:complexType name="er_attribute_type">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="data_type" type="data_types" use="required"/>
</xs:complexType>

<!-- Mozne datove typy atributu -->
<xs:simpleType name="data_types">
  <xs:restriction base="xs:string">
    <xs:enumeration value="BOOLEAN"/>
    <xs:enumeration value="INTEGER"/>
    <xs:enumeration value="DOUBLE"/>
    <xs:enumeration value="STRING"/>
    <xs:enumeration value="CHAR"/>
    <xs:enumeration value="DATE"/>
  </xs:restriction>
</xs:simpleType>

<!-- Ulozeni jednoho hierarchickeho pohledu -->
<xs:complexType name="h_diagram_type">
  <xs:sequence>
    <xs:sequence>
      <xs:element name="node" type="node_type" minOccurs="2" maxOccurs
          ="unbounded"/>
      <xs:element name="cluster_of_nodes" type="cluster_of_nodes_type"
           minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="cluster_of_edges" type="cluster_of_edges_type"
           minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:element name="edge" type="edge_type" minOccurs="1" maxOccurs="
       unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string"/>
</xs:complexType>

<!-- Uzel hierarchickeho pohledu -->
<xs:complexType name="node_type">
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string"/>
  <!-- odkaz na element ER diagramu ktery uzel predstavuje/
```

```
      reprezentuje -->
  <xs:attribute name="repr" type="xs:IDREF" use="required"/>
</xs:complexType>

<!-- Klastr uzlu -->
<xs:complexType name="cluster_of_nodes_type">
  <xs:sequence>
    <!-- seznam uzlu, ktere klastr seskupuje, hrany z klastru do jeho
       ucastniku se nereprezentuji jako edge -->
    <xs:element name="node" type="xs:IDREF" minOccurs="1" maxOccurs="
       unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <!-- odkaz na element ER diagramu -->
  <xs:attribute name="repr" type="xs:IDREF" use="required"/>
</xs:complexType>

<!-- Klastr hran -->
<xs:complexType name="cluster_of_edges_type">
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <!-- odkaz na element H diagramu -->
  <xs:attribute name="parent" type="xs:IDREF" use="required"/>
  <!-- odkaz na element ER diagramu -->
  <xs:attribute name="repr" type="xs:IDREF" use="required"/>
</xs:complexType>

<!-- Hrana hierarchickeho pohledu -->
<xs:complexType name="edge_type">
  <xs:sequence>
    <!-- projekce odpovidajici hrane H diagramu -->
    <xs:element name="projection" type="projection_type" minOccurs="0"
        maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID"/>
  <xs:attribute name="parent" type="xs:IDREF" use="required"/>
  <xs:attribute name="child" type="xs:IDREF" use="required"/>
</xs:complexType>

<!-- Projekce relace nebo slabe entity nebo typu datovy uzel-->
<xs:complexType name="projection_type">
  <xs:sequence>
    <xs:element name="relation" type="xs:IDREF"/><!-- odkaz do ER
       diagramu -->
    <!-- odkaz do ER diagramu -->
    <xs:element name="context" type="xs:IDREF" minOccurs="0" maxOccurs
       ="unbounded"/>
    <xs:element name="parent" type="xs:IDREF"/><!-- odkaz do ER
       diagramu -->
    <xs:element name="child" type="xs:IDREF"/><!-- odkaz do ER
       diagramu -->
  </xs:sequence>
```

```
    <xs:attribute name="id" type="xs:ID"/>
  </xs:complexType>

  <!-- seznam integitnich omezeni -->
  <xs:complexType name="integrity_constraints_type">
    <xs:sequence>
      <xs:element name="integrity_constraint" type="
          integrity_constraint_type" minOccurs="0" maxOccurs="unbounded"
          />
    </xs:sequence>
  </xs:complexType>

  <!-- Jedno integritni omezeni -->
  <xs:complexType name="integrity_constraint_type">
    <xs:sequence>
      <xs:element name="projection" type="projection_type"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="context" type="xs:IDREF" use="required"/>
    <xs:attribute name="lowerBound" type="xs:unsignedInt" use="required"
        />
    <xs:attribute name="upperBound" type="unsignedIntPlusStar" use="
        required"/>
  </xs:complexType>

  <!-- Datovy typ pro hodnotu integritniho omezeni,
  muze nabyvat hodnot nezaporne cele cislo nebo * -->
  <xs:simpleType name="unsignedIntPlusStar">
    <xs:union>
      <xs:simpleType>
        <xs:restriction base="xs:unsignedInt"/>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="*"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>
</xs:schema>
```