Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Použití Oracle RDBMS jako XML Databáze

Plzeň, 2016 Radek Petruška

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne _____ _____

podpis

Abstrakt

Cílem diplomové práce je najít a porovnat možné způsoby zpracování XML dokumentů, které nabízí Oracle databáze.

Práce se zaměřuje především na porovnání efektivity jednotlivých způsobů zpracování XML dokumentů v databázi na základě rychlosti ukládání a vyhledávání dat.

Výsledkem diplomové práce je knihovna, která poskytuje všechny možné implementace zpracování XML dokumentů ze strany klienta a jejich nahrání do databáze.

Abstract

The goal of this thesis is to find and compare all of possible ways how to work with the XML documents in the Oracle database.

The work focuses on comparing the effectiveness of various methods of processing XML documents in a database based on speed of saving and searching.

The main result of this thesis is a library which contains several ways how to process XML document on client side and inserting to database.

Obsah

1 Úvod

Hlavní cílem diplomové práce je zrychlit přenos záznamů mezi systémy SIMS (Sdružené Informace Matrik Studentů) a IS/STAG (Informační Systém Studijní Agendy). Systém SIMS pravidelně poskytuje informace o studentech v podobě XML souborů. Tyto soubory se poté musejí ukládat do informačního systému IS/STAG. V současné době je přibližná doba ukládání vstupních souborů 20 až 30 minut. Hlavním cílem je tuhle dobu zkrátit na minimum.

Diplomová práce se zaměřuje především na možnosti zpracování XML souborů v rámci databáze Oracle 12c za použití nástroje Oracle XML Database. Jedná se o skupinu základních funkcí, která je přímo navržena pro zpracování XML dokumentů. Oracle XML Database poskytuje funkce pro vytvoření základních datových struktur i funkce pro efektivní práci s nimi. Cílem je tyto jednotlivé možnosti zpracování XML souborů mezi sebou porovnat a navrhnout nejvhodnější řešení.

Teoretická část diplomové práce obsahuje definici základních pojmů, popis základních datových struktur a ukázky spolu s popisem jednotlivých funkcí pro práci s XML dokumenty, které poskytuje databáze Oracle 12c. Teoretická část se dále věnuje analýze dostupných technologií pro zpracování XML souborů z pohledu klienta a souhrnu všech testovaných kritérií pro porovnávání jednotlivých řešení s ohledem na různé typy XML dokumentů.

Realizační část se zaměřuje na popis všech implementací použitých v diplomové práci. Tato část popisuje aplikaci pro generování testovacích dat a aplikaci pro testování ukládání a vyhledávání v databázi. Dále obsahuje ukázku implementace vlastního modulu pro IS/STAG, který je založen na upravené testovací aplikaci pro ukládání dat.

Kapitoly Výsledky porovnávaných uložišť a Výsledky modulu se zaměřují na porovnání a vyhodnocení naměřených dat. Kapitoly se soustředí na podrobné znázornění naměřených dat v podobě tabulek a grafů.

Poslední kapitola diplomové práce je kapitola Závěr. Ta obsahuje obecné shrnutí získaných poznatků.

2 Teoretická část

2.1 XML

2.1.1 Obecné informace

Extensible Markup Language dále pod zkratkou XML je univerzální značkovací jazyk. Tento jazyk byl vyvinut konsorciem W3C jako zjednodušená podoba staršího jazyka SGML. Jazyk XML se používá především pro univerzální komunikaci (výměnu dat) mezi různými systémy. Tato forma výměny informací je mnohem univerzálnější a praktičtější než používání různých binárních formátů jakými jsou doc, xls a jiné.

Jazyk XML jako takový neobsahuje žádné konkrétní elementy. Všechny elementy jazyka jsou definovány pro konkrétní potřeby. Definice těchto elementů můžou být uloženy v souborech DTD a XSD. Tyto soubory jsou podrobně popsány v následujících kapitolách.

Hlavní výhody jazyka oproti jiným formátům používaných pro přenos textové informace jsou:

- platformní nezávislost,
- standardizace,
- podpora různých národních kódování
- a převod do různých formátů.

Samozřejmě není tento jazyk vhodný pro všechny typy informací. Například pro multimediální informace jakými jsou hudba a video je XML úplně nevhodným řešením. Jak z pohledu místa tak i přehrávání [1] [2]. Ukázka základního XML kódu.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <osoba>
    <jmeno>Radek</jmeno>
    <prijmeni>Petruška</prijmeni>
    <datumNarozeni>1991-05-02</datumNarozeni>
  </osoba>
```

2.1.2 Základní XML struktura

Základním stavebním kamenem každého XML dokumentu jsou tagy (textové značky). Existuje několik pravidel pro pojmenovávání tagů. Jméno tagu může obsahovat číslice, pomlčku „-“, tečku „.“, podtržítko „_“ a jakékoliv písmeno včetně písmen s diakritickým znaménkem. Dále jméno tagu nesmí začínat číslicí a pro jednotlivá jména tagu je rozlišována velikost písmen (jsou case sensitive). Obecné doporučení je jednotlivé tagy pojmenovávat významově s pomocí neakcentovaných znaků. Další

prvkem XML dokumentu je element. Ten se skládá z počátečního a koncového tagu a informace mezi nimi. Tato informace může být i prázdný řetězec. V XML dokumentu existuje několik možných zápisů elementů. Všechny tyto zápisy jsou validní. Každý element může mít nula až N atributů. Atributy se vždy uvádí v počátečním tagu daného elementu. Pořadí atributů daného elementu je libovolné. Atribut se skládá z dvojice název a hodnota, kdy název atributu musí být unikátní pro daný element a hodnota atributu musí být uzavřena v uvozovkách. Dalším prvkem, který se často vyskytuje v XML dokumentu, je komentář. Komentáře jsou v dokumentu zapisovány pomocí následující direktivy: *<!-- text komentáře -->*. Jediné pravidlo pro práci s komentáři je, že se nesmí vnořovat do jména tagu [1].

```
<!-- ukázka zápisu elementu s neprázdným řetězcem -->
<jmeno>Radek</jmeno>

<jmeno>
Radek
</jmeno>


<!-- ukázka zápisu s prázdným řetězcem -->
<jmeno></jmeno>

<jmeno/>

<jmeno />


<!-- ukázka atributu -->
<rychlost jednotka="KM">50</rychlost>


<!-- ukázka chybného komentáře -->
<jmeno <!-- chybný komentář -->>
```

2.1.3 Validace XML dokumentu

Existuje několik možných kontrol XML dokumentu. Základní kontrola testuje strukturu samotného XML dokumentu. Pokud tato kontrola proběhne v pořádku, můžeme o XML dokumentu tvrdit, že je správně strukturován (well-formed). Pravidla pro kontrolu správnosti struktury [1]:

1. musí existovat právě jeden kořenový element,
2. každá počáteční značka má odpovídající koncovou značku,
3. elementy se nesmějí překrývat (křížit),
4. hodnoty atributů musí být v uvozovkách (nebo v apostrofech),
5. element nesmí mít stejně pojmenované atributy,

6. komentáře nesmí být vnořené a ani uvnitř značek
7. a ve znakových datech nejsou znaky **<** a **&**.

Existuje několik dalších kontrol, které lze aplikovat na XML dokument. Tyto kontroly jsou již závislé na dalších souborech, které v sobě můžou obsahovat popis struktury a datových typů jednotlivých elementů validovaného XML dokumentu. Soubory jsou definovány podle různých schémových jazyků. Nejstarším používaným jazykem je Document Type Definition (DTD). V tuto chvíli je pravděpodobně nejpoužívanější a nejznámější jazyk XML Schema Definition (XSD). Dalšími méně známými jazyky jsou Relax NG[1] (RNG), který je přímým konkurentem XSD, a Schematron[2]. V diplomové práci jsou podrobněji rozebrány pouze jazyky DTD a XSD.

2.1.4 Document Type Definition [DTD]

Patří do obecné skupiny pro popis schémových jazyků. Jedná se o první schémový jazyk. DTD je textově orientovaný dokument. Protože je DTD jeden z nejstarších schémových jazyků, každý XML parser ho podporuje. DTD může obsahovat pouze čtyři typy deklarací [1]:

1. elementy,
2. atributy,
3. entity
4. a notace.

Element je deklarován pomocí názvu a popisu obsahu. Název elementu odpovídá názvu tagu. Obsahem může být prázdný řetězec, jiný element, text, nebo text spolu s elementy (smíšený obsah).

Deklarace atributu se skládá ze čtyř částí, kterými jsou název elementu, název atributu, typ atributu a povinnost výskytu. Existuje pět atributových typů:

- CDATA – hodnota atributu je obecný text,
- NMTOKEN – hodnota atributu je jedno slovo,
- NMTOKENS – hodnota atributu je několik slov oddělených mezerami,
- ID – hodnota atributu musí mít jedinečnou hodnotu v rámci celého XML dokumentu a hodnota musí začínat písmenem (A123)
- a uvedením výčtu (brambora | mrkev | rajče).

Schémový jazyk DTD má pro atributy tři typy výskytu:

- #REQUIRED – povinný výskyt atributu, pokud není pro atribut výskyt v DTD uveden, automaticky se předpokládá povinný výskyt,
- #IMPLIED – atribut je volitelný

[1] http://relaxng.org/
[2] http://www.schematron.com/

- a standardní implicitní hodnota, která se používá u výčtového typu.

Deklarace entity se vyskytuje pouze zřídka. Entity pomáhají v DTD dokumentu minimalizovat opakování stejného zdrojového kódu a to například při používání společných atributů pro různé elementy.

Notace většinou slouží k deklaraci externího obsahu, který není přímo uložen v XML dokumentu. Externím obsahem mohou být například obrázky. Notace přesně specifikuje typ a umístění daného obrázku. Díky této informaci aplikace, která bude zobrazovat XML dokument, může spolu s textovými daty zobrazit i externí obrázek [1] [3].

Ukázka vnořeného DTD do XML souboru

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE osoba [
<!ELEMENT osoba (jmeno,prijmeni,datumNarozeni)>
<!ELEMENT jmeno (#PCDATA)>
<!ELEMENT prijmeni (#PCDATA)>
<!ELEMENT datumNarozeni (#PCDATA)>
]>

<osoba>
  <jmeno>Radek</jmeno>
  <prijmeni>Petruška</prijmeni>
  <datumNarozeni>1991-05-02</datumNarozeni>
</osoba>
```

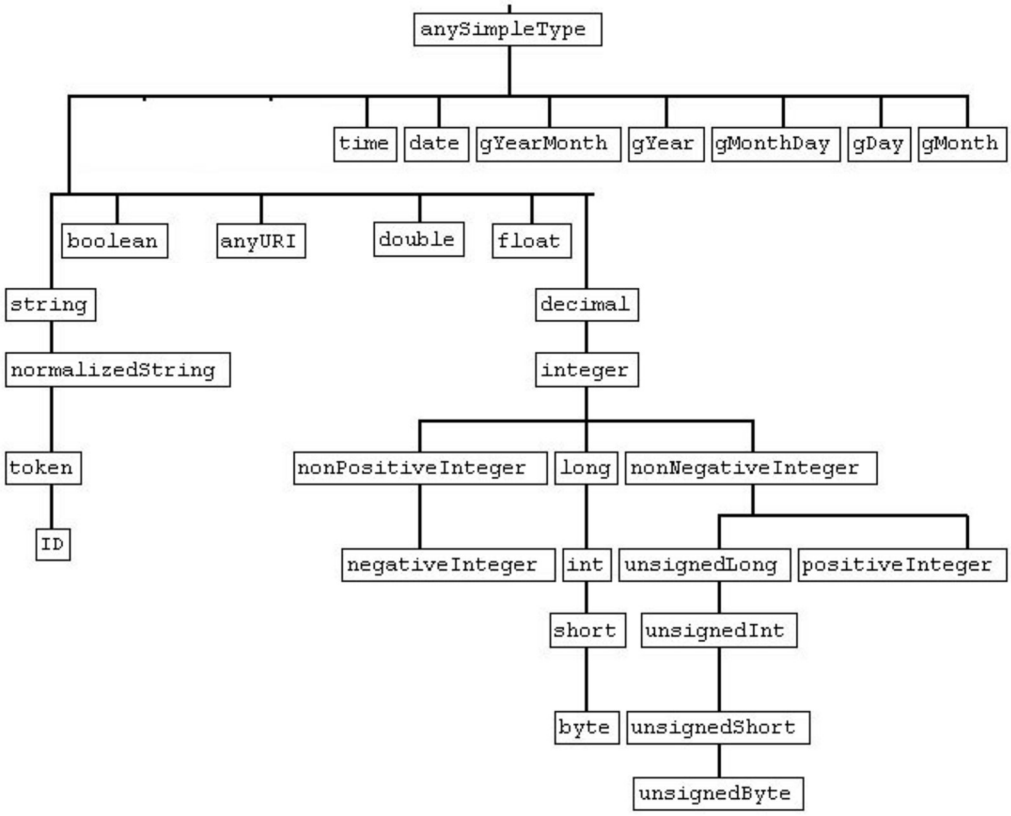
2.1.5 XML Schema Definition [XSD]

Tento schémový jazyk byl roku 2001 uznán jako standard a je všeobecně podporován. Mezi firmy, které ho aktuálně podporují, patří i Oracle. Oracle XML databáze využívá XSD pro vytváření pomocných datových struktur. Tyto struktury pak může používat pro práci se samotnými XML dokumenty. Využití pomocných datových struktur je blíže popsáno v kapitole 2.3.3 Strukturované uložiště.

XSD soubory dodržují XML konvenci. To znamená, že XSD soubory mohou být validovány stejně jako klasický XML dokument. Dále umožňují využívat jmenné prostory, vytvářet vlastní datové typy a umožnují nastavování specifických restrikcí pro určité elementy a atributy.

Jmenné prostory slouží především pro používání různých XSD souborů v rámci jednoho XML dokumentu. V hlavičce XML dokumentu jsou nadeklarované jednotlivé jmenné prostory spolu s jejich prefixy. Teoreticky může totiž docházet ke konfliktům jmen jednotlivých tagů. Díky použití prefixů se těmto konfliktům můžeme úplně vyhnout a používat stejně pojmenované tagy z různých XSD souborů. Tahle vlastnost se především využívá, když s jedním XML dokumentem pracuje více lidí a každý do něj definuje určitou část.

Schémový jazyk XSD dále umožňuje definovat vlastní datové typy. Ty jsou odvozené od základních datových typů (viz Obrázek 2-1 Běžné datové typy, zdroj obrázku [1]) spolu s některými specifickými restrikcemi.



Obrázek 2-1 Běžné datové typy

V XSD na rozdíl od DTD byly restrikce rozšířeny o možnost určovat konkrétní počet výskytu daného elementu. V DTD je tento typ restrikcí pouze předefinovaný. DTD má předdefinované tyto restrikce:

- otazník „**?**“ počet výskytů [0 – 1],
- hvězdičku „*“ počet výskytů [0 – N]
- a plus „+“ počet výskytů [1 – N].

XSD umožňuje pro jednotlivé základní datové typy nastavovat specifické restrikce. Například pro datový typ string můžeme nastavit minimální a maximální délku řetězce. Z takto vytvořených datových typů můžeme dále sestavovat komplexní datové typy, ve kterých nastavujeme konkrétní pořadí a výskyty elementů.

Ukázka XSD souboru

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          elementFormDefault="qualified">
 <xs:element name="osoba">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="jmeno"/>
    <xs:element ref="prijmeni"/>
    <xs:element ref="datumNarozeni"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="jmeno" type="xs:string"/>
 <xs:element name="prijmeni" type="xs:string"/>
 <xs:element name="datumNarozeni" type="xs:date"/>
</xs:schema>
```

Ukázka použití XSD v XML souboru

```
<?xml version="1.0" encoding="UTF-8"?>
 <osoba
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="myXsd.xsd">
  <jmeno>Radek</jmeno>
  <prijmeni>Petruška</prijmeni>
  <datumNarozeni>1991-05-02</datumNarozeni>
 </osoba>
```

2.2 Oracle databáze

V roce 1979 firma Relational Software, Inc. (RSI), která je nyní známá jako Oracle Corporation, představila první komerční verzi své databáze Oracle V2. Jednalo se o RDBMS databázi. V následujících letech firma Oracle Corporation postupně vydávala další verze své databáze. Poslední dostupnou verzí je Oracle Database 12c, která byla vydána v roce 2013.

Databáze	Rok vydání	Změny
Oracle 2	1979	První komerční verze
Oracle 3	1983	První RDBMS na mainframu, C, platformě nezávislá
Oracle 4	1984	Podpora čtecích transakcí
Oracle 5	1985	Podpora distribuovaného DB systému
Oracle 6	1988	PL/SQL, zamykání řádků, obnova, záloha

Databáze	Rok vydání	Změny
Oracle 7	1992	PL/SQL, triggery
Oracle 8	1997	ORDBMS, large table
Oracle 8i	1999	Podpora internetových protokolů a podpora jazyku Java
Oracle 9i	2001	Oracle XML DB
Oracle 10g	2003	Grid computing, Oracle ASM
Oracle 11g	2007	Přidáno mnoho nástrojů pro automatickou správu DB
Oracle 12c	2013	Navržena pro Cloud, přidána podpora JSON

Tabulka 2-1 Přehled historických verzí databáze

Všechny historické údaje byly získány z oficiálních webových stránek firmy Oracle Corporation [4].

2.2.1 Oracle DB 12c Enterprise Edition

Jedná se o poslední dostupnou verzi Oracle databáze. V této verzi došlo k několika změnám, které mají dopad na jednotlivé nástroje Oracle XML databáze. Těmito změnami jsou:

- aktualizace ASM (Automatic Storage Management)
- aktualizace Secure File,
- a nahrazování SQL funkcí pro XML funkcemi XQuery.

ASM zajišťuje kompletní správu diskového prostoru dané databázové instance. ASM byl rozšířen o nový filtr, který zajišťuje, že do databázových souborů může zapisovat pouze instance dané databáze a nikdo jiný.

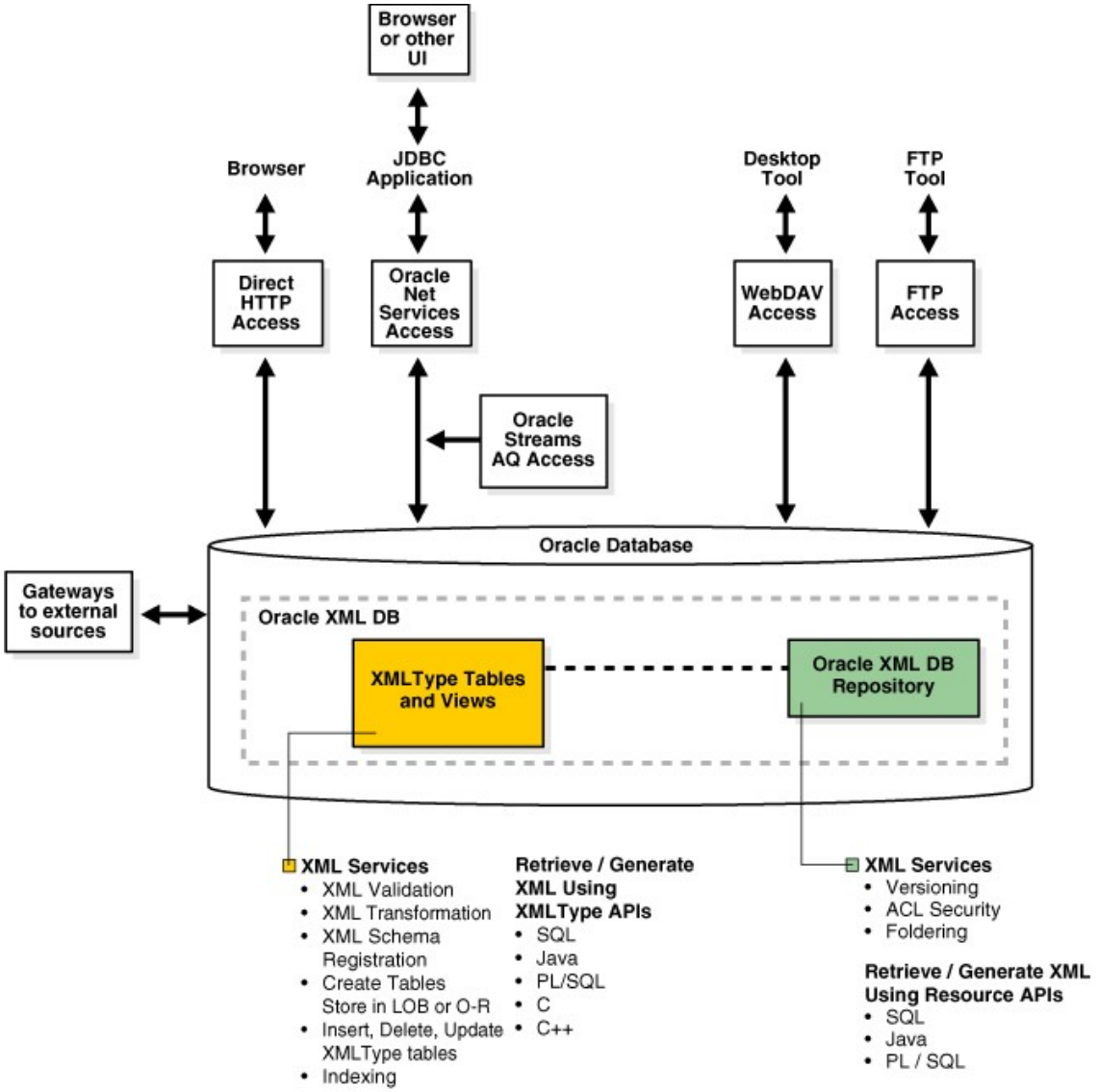
Secure File je nové paradigma pro zpracování a ukládání binárních souborů jakými jsou hudba, video, fotografie a textové soubory (datový typ CLOB) do databáze. Tento přístup funguje na jednoduchém principu. Vedle databáze je vytvořen další soubor, který slouží jako paměťový prostor pro danou tabulku s těmito binárními záznamy. Díky tomu je umožněna snadná deduplikace a komprimace binárních dat, což má za následek zvýšení výkonu samotného zpracování binárních souborů. Tato nová vlastnost byla představena v databázi Oracle verze 11g. V základním nastavení databáze byla tato vlastnost nastavena jako volitelná. Ve verzi 12c v základním nastavení je už použití Secure File přednastaveno.

V aktuální verzi databáze Oracle 12c jsou postupně SQL funkce pro práci s XML daty (updateXML, insertChildXML, insertChildXMLbefore, insertChildXMLafter, …) označovány jako DEPRECATED a nahrazovány pomocí XQuery a XPath výrazů. Tyto nové funkce by měly být efektivnější než jejich předchůdci.

2.2.2 Oracle XML databáze

Oracle XML databáze je soubor technologií, který se zaměřuje na maximálně efektivní práci s XML dokumenty v rámci Oracle databáze. Nejedná se tedy o samostatnou databázi, která by byla implementována pomocí XML souborů. V dalším textu je tento soubor technologií označován zkratkou XML DB.

XML DB podporuje standardní model XML dokumentu, schématu a metody pro práci s XML dokumenty. Tyto všeobecně uznávané standardy vytvořilo konsorcium WC3. Architektura XML DB je přehledně zobrazena na následujícím obrázku. Obrázek byl převzat ze zdroje [5].



Obrázek 2-2 Oracle XML DB architektura

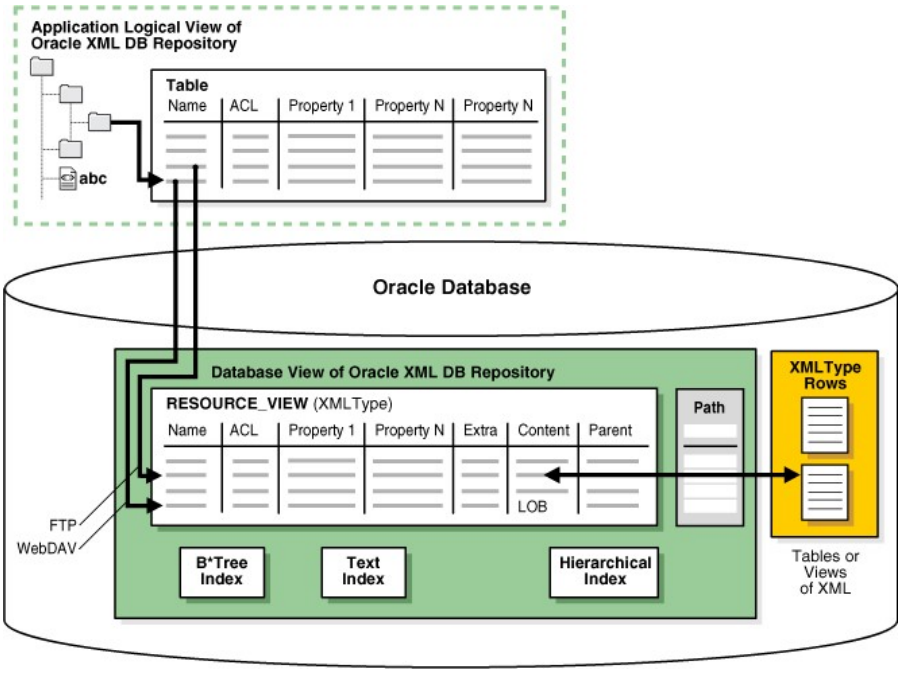
XML DB umožňuje dva základní přístupy pro práci s XML dokumenty. Tyto přístupy se mohou navzájem křížit.

1. Přístup Oracle XML DB repository je určitý druh pokročilého souborového systému, který umožňuje silné hierarchické ukládání XML dokumentů.
2. Standardní přístup využívá abstraktní datový typ XMLType. XML dokument je zde uložen do předem připravených datových struktur (tabulek).

Přístup Oracle XML DB repository vychází z myšlenky, že jednotlivé repositáře obsahují prvky. Prvkem může být složka nebo samostatný soubor. Ke každému prvku jsou zaznamenávány klíčová meta-data:

- název prvku a jeho umístění,
- obsah prvku (může a nemusí se jednat o XML data),
- systémové informace (vlastník, datum uložení, …),
- uživatelské informace (klíčová slova, popis, …)
- a oprávnění (ACL – Access Control List).

Další velmi zajímavou vlastností, kterou tento přístup podporuje, je verzování pro jednotlivé soubory, kdy předchozí verze dokumentů jsou zachovávány podle standardu IETF WebDAV. Schématické znázornění přístupu Oracle XML DB repository je zobrazeno na následujícím obrázku [5].



Obrázek 2-3 Oracle XML DB repository

Přímým využitím standardního přístupu s ohledem na maximální rychlost zápisu a vyhledáváním konkrétního XML dokumentu se zabývá tato diplomová práce. Tento přístup je blíže rozveden a detailně popsán v následujících kapitolách.

2.3 Oracle XMLType

XMLType je abstraktní datový typ, který umožňuje efektivně pracovat s XML daty. Pro efektivní zpracování dat poskytuje několik možných řešení s ohledem na rychlost ukládání, načítání a prohledávání již uložených záznamů. Každé z těchto řešení má svoje kladné a záporné stránky. Neexistuje tedy jedno univerzální řešení pro všechno. Nástin všech dostupných řešení je popsán v následujících kapitolách.

Ukázka použití datového typu XMLType pro různé případy užití byla převzata a přeložena z následujícího zdroje [6]. Tento dokument popisuje nejvhodnější způsoby využití datového typu XMLType.

	Data-Centric		**Document-Centric**	
Případy užití	XML schéma s minimálními změnami	XML schéma + nestrukturované části	Podobná XML schémata se stejným základem	Různá XML schémata
Příklad	Záznamy o studentech (matrika)	Záznamy o studentech + výsledky různých hodnocení	Technické články (autor, datum, …)	Webové stránky
Uložiště	Strukturované	Hybridní	Nestrukturované (CLOB) nebo Binární	
Index	B-tree index	Index pro strukturovanou + Index pro nestrukturovanou část	Funkční index	XML index

Tabulka 2-2 Ukázka použití XMLType

- Data-Centric - tímto pojmem jsou označena silně strukturovaná data. Struktura takovýchto dat se v čase téměř nemění. Tento druh dat je typický pro relační databáze.
- Document-Centric - pojem označuje skupinu dat s podobnou strukturou. Každý záznam může mít svoji strukturu nějakým způsobem lehce odlišnou od přechozích záznamů. Typickým příkladem jsou webové stránky.

2.3.1 Binární uložiště

Binární uložiště ukládá data ve specifickém formátu, který byl přímo navržen pro potřeby binárního uložiště. V původním anglickém textu je tento formát označován jako „post-parsed binary format“. Výhody a nevýhody binárního uložiště jsou uvedeny v následujícím výpise, který byl převzat ze zdroje [6].

Výhody

- Nezávislý na XML schématu,
- uložení libovolného XML záznamu
- a lepší paměťová náročnost než nestrukturované uložiště.

Nevýhody

- Pomalejší vyhledávání než strukturované uložiště.

Ukázka vytvoření Binárního uložiště bez použití Secure File

Jedná se o zastaralý přístup. Ve verzi databáze Oracle 12c je stále možné takto vytvořit binární uložiště, ale nejedná se o doporučený způsob. Takto vytvořené uložiště by teoreticky mělo být pomalejší než uložiště vytvořené pomocí Secure File.

```
ALERT SYSTEM SET DB_SECUREFILE = 'ignore';
CREATE TABLE table_name OF XMLTYPE
 XMLTYPE STORE AS BINARY XML;
```

Ukázka vytvoření Secure File spolu s Binárním uložiště

Tento přístup doporučuje oficiální dokumentace databáze Oracle 12c.

```
--vytvoření Secure File
CREATE BIG FILE TABLESPACE
 bigtbs_01
 DATAFILE 'bigtbs_f1.dat'
 SIZE 20M
 AUTOEXTEND ON;
```

- bigtbs_01 – DB název Secure File
- datafile – cesta k Secure File
- size – nastavení počáteční velikosti souboru
- autoextend – povolení automatické expanze souboru

```
--vytvoření tabulky
CREATE TABLE table_name OF XMLTYPE
 XMLTYPE STORE AS SECURE FILE BINARY XML
 DEDUP_COMP_LOB(
      TABLESPACE bigtbs_01
      COMPRESS HIGH
      RETENTION MIN 3600
      DEDUPLICATE
      CACHE READS
      NO LOGGING);
```

- dedup_comp_lob – nastavení použitého Secure File
- tablespace – název použitého Secure File
- compress – použití komprimace
- retention – doba po kterou DB udržuje spojení do daného Secure File
- deduplicate – použití deduplikace dat
- cache reads – využití cache pamětí pro čtení
- no logging – zákaz logování přístupů

2.3.2 Nestrukturované uložiště

Abstraktní datový typ XMLType je v nestrukturovaném uložišti reprezentován pomocí datového typu CLOB. Díky tomu lze do nestrukturovaného uložiště ukládat libovolné XML dokumenty bez závislosti na XML schématu. Takto uložené dokumenty jsou naprosto identické s originálními dokumenty včetně bílých znaků. Proto je také toto uložiště nejvíc náročné na diskový prostor. Podle oficiálních materiálů je nestrukturované uložiště teoreticky nejlepší pro případy, kdy chcete pracovat s celými XML dokumenty. Shrnutí výhod a nevýhod Nestrukturovaného uložiště bylo převzato ze zdroje [6].

Výhody

- Nezávislý na XML schématu
- a zachování identické bitové kopie záznamu.

Nevýhody

- Pomalé vyhledávání,
- velká paměťová náročnost
- a nelze provádět update části záznamu.

Ukázka SQL kódu pro vytvoření Nestrukturovaného uložiště

```
CREATE TABLE table_name OF XMLTYPE XMLTYPE STORE AS CLOB;
```

2.3.3 Strukturované uložiště

Strukturované uložiště je přímo závislé na XML schématu. Na základě XML schématu si databáze vytvoří pomocné datové tabulky a objekty. Každý ukládaný XML dokument je validován proti danému XML schématu. Shrnutí výhod a nevýhod Strukturovaného uložiště bylo převzato ze zdroje [6].

Výhody

- Nejrychlejší možné prohledávání,
- minimální paměťová náročnost
- a podporuje B-tree index.

Nevýhody

- Závislý na XML schématu
- a pouze záznamy s odpovídajícím schématem lze uložit.

Ukázka registrace schématu

```
DBMS_XMLSCHEMA.registrSchema(
 schemaurl => 'vystupy.xsd',
 schemadoc => bfilename('MYDIR'[3], 'vystupy.xsd'),
 CSID => nls_charset_id('AL32UTF8'),
 local => true,
 genTypes => true,
 getTables => true,
 genBean => false,
 force => false,
 owner => 'SYSTEM');
```

- local – ovlivňuje viditelnost schématu pro ostatní uživatele databáze TRUE = schéma je přístupné pouze vlastníkovi
- genTypes – DB vygeneruje potřebné objekty pro podporu schématu
- genTables – DB vygeneruje potřebné pomocné tabulky
- genBean – DB vygeneruje procedury pro registraci specifického schématu
- force – Pokud vzniknou chyby při registrovaní schématu, DB je ignoruje a stále se pokouší registrovat dané schéma
- owner – vlastník schématu (jakýkoliv uživatel DB)

[3] lokální složka DB instance. Registrace složky:
CREATE OR REPLACE DIRECTORY directory_name AS ‘file_path’

Ukázka vytvoření Strukturovaného uložiště

```
CREATE TABLE table_name OF XMLTYPE
 XMLTYPE STORE AS OBJECT_RELATIONAL
 XMLSCHEMA "vystupy.xsd"
 ELEMENT "SIMS";
```

- element – název elementu, který chci zpracovávat

2.3.4 Shrnutí dostupných uložišť

Každé z výše uvedených uložišť má svoje specifické výhody a nevýhody oproti ostatním. Obecné doporučení firmy Oracle jak jednotlivá uložiště používat je uvedeno v tabulce Tabulka 2-2 Ukázka použití XMLType v kapitole 2.3 Oracle XMLType.

Uložiště	Výhody	Nevýhody
Binární	Nezávislý na XML schématu	Pomalejší vyhledávání než strukturované uložiště
	Uložení libovolného XML záznamu	
	Lepší paměťová náročnost než nestrukturované uložiště	
Nestrukturované	Nezávislý na XML schématu	Pomalé vyhledávání
	Zachování identické bitové kopie záznamu	Velká paměťová náročnost
		Nelze provádět update části záznamu
Strukturované	Nejrychlejší možné prohledávání	Závislý na XML schématu
	Minimální paměťová náročnost	Pouze záznamy s odpovídajícím schématem lze uložit
	Podporuje B-tree index	

Tabulka 2-3 Shrnuti dostupných uložišť

2.4 Oracle indexy

V následující kapitole jsou popsány indexy, které podporuje Oracle databáze. Příklady SQL kódů jsou uvedeny pouze u indexů, které byly použity v rámci diplomové práce.

2.4.1 XML index

XML index byl navržen tak, aby urychloval práci s nestrukturovanými daty, která jsou uložena v binárním a nestrukturovaném uložišti. Největší problém nastává, pokud chceme pracovat pouze s částí XML dokumentu. Za předpokladu, že chceme pouze vyhledávat a pracovat vždy s celými XML dokumenty, tak se teoreticky můžeme bez XML indexu obejít a vytvářet klasické funkční indexy na určité elementy nebo atributy daného XML dokumentu [7]. Příklad jak vytvořit takový index je uveden v kapitole 2.4.3 Funkční index. Ale pro efektivní zpracování jednotlivých částí XML dokumentu se bez XML indexu neobjedeme.

Základní funkcí XML indexu je, že indexuje vnitřní strukturu daného XML dokumentu. Díky tomu můžeme velmi efektivně prohledávat XML dokumenty. V dokumentu vyhledáváme pomocí XPath výrazů. Ukázky a popis základních XPath výrazů jsou uvedeny v kapitole 2.5 XQuery. XML index se využívá především v SQL/XML funkcí, kterými jsou XMLQuery, XMLTable, XMLExists, a XMLCast. Použití XML indexu není v SQL dotazu vázáno čistě na část s podmínkou (WHERE), ale můžeme ho použít i na jiných místech. Vlastní XML index má následující strukturu.

- Path Index – indexuje XML tagy v dokumentu. Pro každý tag existuje záznam, který popisuje úroveň zanoření.
- Order Index – udržuje informace o interní struktuře. Vztahy mezi jednotlivými tagy (předchůdce, rodič, potomek a sourozenec).
- Value Index – indexuje hodnoty dané XML části. Tento index se uplatňuje především v podmínkové části dotazu.

Ukázka XML indexu v praxi

```
<!-- 1. záznam v tabulce => ROW ID = R1 -->
<student>
 <jmeno>Radek</jmeno>
 <prijemni>Petruška</prijmení>
 <adresa>
      <ulice>Rudolfovská 509</ulice>
      <mesto>České Budějovice</mesto>
 </adresa>
</student>
```

```
<!-- 2. záznam v tabulce => ROW ID = R2 -->
<student>
 <jmeno>Franta</jmeno>
 <prijmeni>Omáčka</prijmeni>
 <adresa>
      <ulice>U Buku</ulice>
      <mesto>Plzeň</mesto>
 </adresa>
</student>
```

XML index pro ukázková data má následující strukturu.

Path Index

Path ID	Indexed XPath
1	/student
2	/student/jmeno
3	/student/prijmeni
4	/student/adresa
5	/student/adresa/ulice
6	/student/adresa/mesto

Tabulka 2-4 Ukázka Path indexu

Order Index + Value Index

Path ID	Row ID	Order Index	Value Index
1	R1	1	RadekPetruškaRudolfovská 509České Budějovice
2	R1	1.1	Radek
3	R1	1.2	Petruška
4	R1	1.3	Rudolfovská 509České Budějovice
5	R1	1.3.1	Rudolfov
6	R1	1.3.2	České Budějovice
1	R2	1	FrantaOmáčkaU BukuPlzeň
2	R2	1.1	Franta
3	R2	1.2	Omáčka
4	R2	1.3	U BukuPlzeň
5	R2	1.3.1	U Buku
6	R2	1.3.2	Plzeň

Tabulka 2-5 Ukázka Order indexu a Path indexu

Z příkladu je patrně, že velikost Value indexu pro větší XML dokumenty, může neúměrně narůstat. Proto je zde zásadní omezení na velikost tohoto pole. Maximální velikost je 4000 bajtů. Část řetězce, která přesáhne maximální velikost je uříznuta [8].

2.4.2 B-tree index

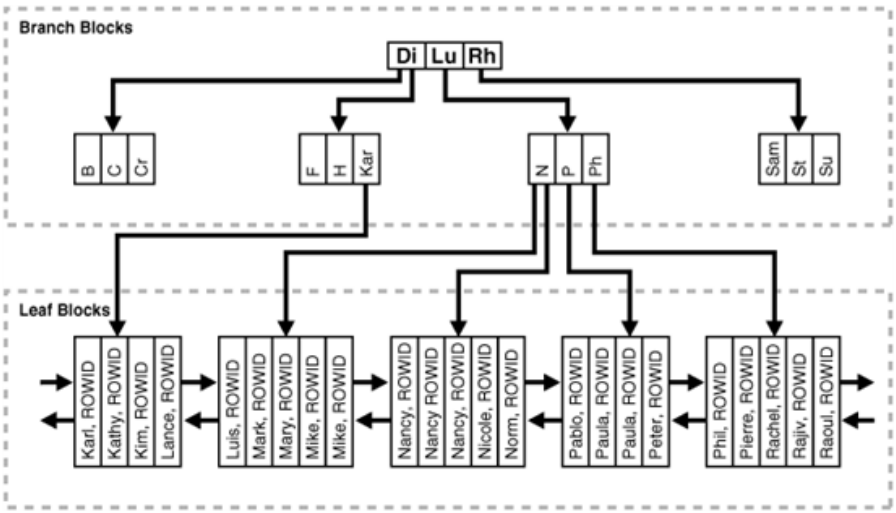
B-tree index je standardně používán pro indexování strukturovaných dat. Proto se dá v případě abstraktního datového typu XMLType využít pouze ve strukturovaném uložišti.

B-tree index jak už název napovídá, funguje na principu B-tree. Jedná se tedy o vyvážený strom, kde platí základní pravidlo, že klíče jsou seřazené a pro každý klíč platí, že v jeho levém podstromu jsou pouze prvky menší a v pravém podstromu jsou prvky větší. Umístění prvků, které mají stejnou velikost jako daný klíč, je závislé na konkrétní implementaci.

Oracle si pro efektivnější prohledávání upravil strukturu vyváženého stromu tak, že každý list navíc obsahuje ukazatel na předešlý a následující list. Struktura Oracle B-tree indexu se pak skládá z následujících prvků [9].

- List [Leaf node] – obsahuje v sobě 1 – N záznamů. Každý záznam se skládá z indexované hodnoty a ukazatele na data (ROW_ID). Dále v sobě obsahuje ukazatele na předešlý a následující list.
- Uzel [Branch node] – obsahuje ukazatele na listy spolu s klíči, podle kterých probíhá vyhledávání ve vyváženém stromu.

Ilustrace struktury B-tree indexu byla převzata ze zdroje [9].



Obrázek 2-4 B-tree index

```
--vytvoření B-tree indexu
CREATE INDEX index_name ON table_name(column_name)
```

2.4.3 Funkční index

Funkční index byl představen v databázi Oracle 8i. Důvod proč tento typ indexu vznikl je, že v mnoha případech klasický index prostě nestačí. Funkční index nám umožnuje vytvořit index přes jedno nebo více polí na základě SQL funkce [10].
Například máme tabulku studentů, kdy každý student má uloženo jméno a příjmení. Problémem je, že v sloupci příjmení mají někteří studenti své příjmení napsáno velkými písmeny a jiní pouze s velkým počátečním písmenem. Požadavkem je najít všechny záznamy s příjmením Petruška, tudíž nám v tomto případě na sloupec příjmení nepomůže klasický index. Vyhledávání by tedy probíhalo nad celou tabulkou (full scan). Proto nad daným sloupcem vytvoříme Funkční index, který bude používat funkci UPPER.

Ukázka SQL dotazu

```
SELECT *
FROM STUDENTS
WHERE UPPER(STUDENTS.PRIJMENI) = 'PETRUŠKA'
```

Ukázka Funkčního indexu

```
CREATE INDEX idx_prj_up ON STUDENTS (UPPER(PRIJMENI))
```

Po vytvoření tohoto indexu už bude nad tabulkou probíhat vyhledávání za pomoci indexu (range scan).

Stejně jako funkci UPPER můžeme pro vytvoření Funkční indexu používat i funkce z XML DB pro práci s XML dokumenty.

2.4.4 Bitmap index

Bitmap index pracuje na principu dvourozměrného pole, kde každý řádek pole odpovídá řádku indexované tabulky a každý sloupec odpovídá jedné unikátní hodnotě [11]. Ukázku zjednodušené struktury Bitmap indexu si můžeme prohlédnout na následujícím příkladu.

Tabulka dat

ROW_ID	JMENO	POHLAVI
1	Radek	muž
2	Veronika	žena
3	Hana	žena
4	Jan	muž
5	Lukáš	muž

Tabulka 2-6 Ukázková data

Bitmap Index pro sloupec POHLAVI

ROW_ID	muž	žena
1	1	0
2	0	1
3	0	1
4	1	0
5	1	0

Tabulka 2-7 Struktura Bitmap indexu

Hlavní výhodou tohoto typu indexu je fakt, že je velmi komprimovaný a umožňuje velmi rychlé prohledávání dat. Bohužel tento index není vhodný pro všechny typy záznamů. Celková velikost indexu roste s počtem unikátních hodnot a kvůli tomu potom klesá i rychlost prohledávání. V následujícím výčtu je uveden souhrn vlivu počtu unikátních hodnot na rychlost prohledávání [12].

- 1 – 7 unikátních hodnot. Index dosahuje nejvyšší rychlosti prohledávání.
- 8 – 100 unikátních hodnot. S rostoucím počtem unikátních hodnot začíná pomalu klesat rychlost prohledávání.
- 100 – 10000 unikátních hodnot. Rychlost prohledávání rapidně klesá.
- Přes 10000 unikátních hodnot. Rychlost prohledávání je 10-krát pomalejší než pro index se 100 unikátními hodnotami.

Bitmap index se musí přepočítávat vždy po každé aktualizaci dat (UPDATE, INSERT, DELETE) a je tedy vhodný spíše pro statická data. Nejčastěji se tento index používá v datových skladech, kde je po každé pravidelné aktualizaci dat dostatek času k přepočítání indexů.

Ukázka SQL kódu pro vytvoření Bitmap indexu

```
CREATE BITMAP INDEX idx_name ON table_name(column)
```

2.4.5 Oracle Text

Oracle Text byl ve starších verzích Oracle databáze označován jako interMedia a ConText [13]. Oracle Text poskytuje soubor technologií k vytváření různých typů indexů. Tento typ indexů je vhodný pro full-textové vyhledávání v různých textových dokumentech. Oracle Text poskytuje následující typy indexů.

- CONTEXT – používá se pro indexování různých typů dokumentů. Například MS Wordu, HTML a prostého textu (CLOB).
- CTXCAT – vhodný pro indexování malých dokumentů nebo textových fragmentů spolu s jednotlivými sloupci tabulky.

- CTXRULE – index pro klasifikaci dokumentů.
- CTXXPATH – tento typ indexu slouží pro urychlení vyhledávání pomocí funkce existsNode (funkce pro vyhledávání v abstraktním datovém typu XMLType). Tato funkce spolu s indexem CTXXPATH je označena v aktuální verzi jako zastaralá (DEPRECATED).

2.5 XQuery

XQuery je všeobecný standard, který byl vytvořen konsorciem W3C. Tento standard popisuje obecné funkce pro procházení a úpravy XML dokumentu. Aktuálně existuje více jak 40 různých implementací tohoto standardu [14]. Při vyhledávání dostupných prostředků pro práci s XML dokumenty je velmi často zmiňován termín XPath jazyk. XPath jazyk je pouze podmnožinou celého jazyka XQuery.

XPath jazyk se zaměřuje na navigaci a prohledávání XML dokumentu. XPath byl navržen na základě myšlenky, že XML dokument je převeden do stromové struktury. V této stromové struktuře jsou uvedeny jednotlivé elementy daného XML dokument. Za pomoci základních funkcí pak tento strom procházíme a vyhledáváme potřebné informace.

Značka	Popis
/	Označuje kořen stromu nebo se používá jako oddělovač pro identifikaci rodič-potomek.
//	Zobrazení všech následovníků od daného uzlu.
*	V anglickém originálu je tato značka označována jako „wild card“. Používá se pro nahrazení jmen elementů v XPath výrazu. Značka zastupuje jakýkoliv element v daném výrazu.
[]	Závorky mohou označovat index elementu nebo se používají pro vytváření logických podmínek.
Funkce	XPath jazyk disponuje mnoha pomocnými funkcemi. Například last(), first() a jiné.

Tabulka 2-8 XPath značky

XML DB implementuje XQuery standard se všemi jeho doporučeními. Takto implementovaný standard má pár zásadních odlišností oproti klasickému chování SQL dotazů. Největší rozdíly lze pozorovat ve výsledných množinách jednotlivých dotazů. Pokud klasický SQL dotaz nenajde žádná data, tak uživateli vrátí prázdnou výstupní množinu. Naopak když XQuery dotaz nenajde žádná data, která by splňovala zadané

podmínky, tak vrací uživateli množinu s prázdným řetězcem. Na tento zásadní rozdíl je potřeba brát zřetel při implementaci jakékoliv aplikace, která používá klasické a XQuery dotazy.

V následujících kapitolách jsou uvedeny některé ze základních funkcí, které poskytuje XML DB. Tyto funkce jsou definovány standardem SQL/XML a slouží jako základní rozhraní mezi SQL a XML jazyky. Díky těmto funkcím můžeme efektivně pracovat s XML daty, dále tyto funkce umožňují XML data zobrazovat jako klasická relační data v tabulkách. Jednotlivé popisy funkcí byly vytvořeny na základě oficiální dokumentace databáze pro verzi 12c [15]. Ukázky SQL dotazů využívající XML DB funkce byly provedeny nad XML tabulkou A_JINY_STUDENT_STRUC.

Ukázka datové struktury záznamů

```
<SIMS>
  <Vystup>
    <Student>
      <Jmeno></Jmeno>
      <Prijmeni></Prijmeni>
      <Studia>
            ...
      </Studia>¨
    </Student>
 </Vystup>
</SIMS>
```

2.5.1 XMLQuery

Funkce XMLQuery se nejčastěji v SQL dotazu vyskytuje za klíčovým slovem SELECT. Tato funkce zajišťuje, že výsledná množina dat bude zobrazena ve formátu XML. Například máme XML tabulku, kde každý řádek reprezentuje jednoho studenta. Úkolem je vytvořit z každého řádku XML tabulky jednoduchý XML dokument, který bude obsahovat celé jméno daného studenta.

Ukázka SQL dotazu

```
SELECT XMLQuery(
'<celeJmeno>{//Jmeno/text()}&#32;{//Prijmeni/text()}<
/celeJmeno>'
passing OBJECT_VALUE
RETURNING CONTENT) "Celá jména"
FROM A_JINY_STUDENT_STRUC
```

SQL dotaz obsahuje XQuery výraz, který je tučně vyznačen. Tento výraz je vyhodnocen pro každý řádek XML tabulky. Výsledná množina dat SQL dotazu obsahuje pouze první a druhý sloupec. Třetí sloupec je přidán jako náhled textového zobrazení obsahu daného dokumentu.

	Celá jména	Obsah XMLType
1	(XMLType)	<celeJmeno>Radek Petruška</celeJmeno>
2	(XMLType)	<celeJmeno>Franta Omáčka</celeJmeno>

Tabulka 2-9 XMLQuery výsledek dotazu

2.5.2 XMLCast

Funkce XMLCast umožňuje převést abstraktní datový typ XMLType na některý ze skalárních datových typů: NUMBER, VARCHAR2, CHAR, CLOB, BLOB, REF XMLTYPE a jakýkoliv SQL DATE. Oracle DB rozděluje datové typy do několika kategorií. Následující tabulka zobrazuje seznam kategorií podle oficiální dokumentace Oracle DB. Názvy jednotlivých kategorií jsou ponechány v anglickém znění.

Kategorie	Popis
Scalar	Základní datové typy bez vnitřních komponent
Composite	Prvky, které obsahuji vnitřní komponenty. Například kolekce.
Reference	Ukazatel na datový prvek. Například kurzor.
Large Object (LOB)	Ukazatel na velký datový prvek. Prvek je ukládán do vlastního uložiště (Secure File). Například multimediální soubory.

Tabulka 2-10 Oracle kategorie datových typů

Obvykle je funkce XMLCast používána v SQL dotazu za klíčovým slovem SELECT. Na rozdíl od standardu SQL/XML má implementace funkce v XML DB určitá omezení. Funkce neumožnuje převádět XML na XML a skalární datové typy na XML.

Ukázka použití funkce XMLCast
Jedná se o modifikaci SQL dotazu z předchozí kapitoly 2.5.1 XMLQuery.

```
SELECT XMLCast(XMLQuery(
'<celeJmeno>{//Jmeno/text()}&#32;{//Prijmeni/text()}<
/celeJmeno>'
passing OBJECT_VALUE
RETURNING CONTENT) as Varchar2(200)) as "Celé Jméno"
FROM A_JINY_STUDENT_STRUC
```

Výsledná množina SQL dotazu bude vypadat následovně. Sloupec Celá jména je reprezentován datovým typem VARCHAR2.

	Celá jména
1	<celeJmeno>Radek Petruška</celeJmeno>
2	<celeJmeno>Franta Omáčka</celeJmeno>

Tabulka 2-11 XMLCast výsledek dotazu

2.5.3 XMLTable

Funkce XMLTable byla do Oracle DB přidána ve verzi 10g [16]. Funkce nad XML daty vytvoří virtuální tabulku. Ta zobrazuje XML data jako by se jednalo o klasická data v relační databázi. To znamená, že na základě XQuery výrazu jsou jednotlivé elementy reprezentovány jako sloupce tabulky.

Ukázka použití funkce XMLTable

```
select vTable.*
from A_JINY_STUDENT_STRUC,
XMLTable(
 '//Student'
 passing OBJECT_VALUE
 columns "Jméno"     VARCHAR2(30) path 'Jmeno',
         "Příjmení"  VARCHAR2(30) path 'Príjmení'
 ) vTable;
```

Výsledná množina SQL dotazu.

	Jméno	Příjmení
1	Radek	Petruška
2	Franta	Omáčka

Tabulka 2-12 XMLTable výsledek dotazu

2.5.4 XMLExists

Funkce XMLExists nahrazuje zastaralou funkci existsNode. Funkce existsNode je v aktuální verzi databáze označena jako DEPRECATED. Podle oficiální dokumentace by měla být nová funkce XMLExists rychlejší a lépe optimalizovaná než původní funkce existsNode. V SQL dotazu se funkce XMLExists používá za klíčovým slovem WHERE. Tato funkce zpracuje XQuery výraz a pokud XQuery výraz vrací neprázdnou množinu, tak funkce vrací hodnotu true jinak vrací hodnotu false.

Ukázka použití funkce v SQL dotazu

Cílem je najít XML záznam se jménem studenta Radek Petruška a zobrazit jeho rodné číslo jako klasické pole tabulky.

```
select
 XMLCast(
  XMLQuery('$p//Student/@RodneCislo'
   passing OBJECT_VALUE as "p"
   RETURNING CONTENT) as VARCHAR2(30))
 as "Rodné číslo"

from A_JINY_STUDENT_STRUC
where

 XMLExists('declare default element namespace
"https://sims.msmt.cz";
$p/SIMS/Vystup/Student[Prijmeni/text() = "Petruška"]'
 PASSING OBJECT_VALUE as "p")

AND

 XMLExists('declare default element namespace
"https://sims.msmt.cz";
$p/SIMS/Vystup/Student[Jmeno/text() = "Radek"]'
 PASSING OBJECT_VALUE as "p");
```

Výsledná množina SQL dotazu

	Rodné číslo
1	123456789

Tabulka 2-13 XMLExists výsledek dotazu

2.6 Programovací jazyk

Samotné zadání diplomové práce nespecifikuje konkrétní programovací jazyk. Oficiálně Oracle XML DB poskytuje API pro Javu, C a C++. S ohledem na to, že výsledný modul bude jednou integrován do univerzitního systému IS/STAG, byla jako programovací jazyk zvolena Java. Celý modul byl vyvíjen a testován v Javě verze 1.8.0_73 64bit.

2.6.1 Knihovny třetích stran

Pro vytvoření modulu (knihovny) jsou zapotřebí čtyři knihovny třetích stran. Základní knihovnou je **OJDBC**. Tato knihovna zprostředkovává připojení do databáze. Dalšími třemi knihovnami jsou **XDB, XMLParserV2 a ORAI18N.**

OJDBC celým názvem Oracle Java DataBase Connectivity je volně dostupná na oficiálních webových stránkách firmy Oracle Corporation.

Knihovna XDB obsahuje především definici abstraktního datového typu XMLType. Dále obsahuje různé pomocné prvky pro práci s XML DB. Tato knihovna je volně dostupná na webových stránkách společnosti.

Knihovna XMLParserV2 obsahuje parser, který umožňuje vytvářet abstraktní datový typ XMLType. Oficiálně je knihovna XDB součástí instalace Oracle databáze. Po instalaci databáze má uživatel přístup k archivu všech potřebných knihoven. Na oficiálních stránkách společnosti tuhle knihovnu nenajdete.

Knihovna ORAI18N slouží k převádění znakových sad a je volně dostupná na oficiálních webových stránkách společnosti.

Umístění archivu knihoven na testovací databázi

```
/sw/product/12.1.0.2/rdbms/jlib/...
```

Neoficiálně může být knihovna XMLParserV2 stažena i z internetu.

2.7 Java možnosti zpracování XML

V rámci programovacího jazyku Java existuje mnoho způsobů jak zpracovávat XML soubory. Od standardních způsobů až k použití knihoven třetích stran. V následujících kapitolách jsou představeny standardní možnosti zpracování XML souboru.

2.7.1 SAX

Oficiální anglický název je Simple Api for XML. Jedná se o proudové zpracování XML dokumentu parserem typu push-parser. V rámci vlastního parseru uživatel poté musí implementovat několik základních metod. Při čtení XML souboru jsou pak vyvolávány předem definované události, které jsou zpracovávány těmito metodami.

Seznam metod

- startDocument() a endDocument() – volají se na začátku a konci XML souboru,
- startElement a endElement() – volají se na začátku a konci každého elementu,
- a characters() – slouží k načtení informace mezi úvodním a koncovým tagem.

Výhody	Nevýhody
Efektivní práce s pamětí	Nedokáže se při čtení vrátit
Rychlejší než DOM	Nemá objektový model. Uživatel ho musí implementovat sám.
Podporuje validaci podle schématu	Nepodporuje XPath

Tabulka 2-14 SAX výhody a nevýhody

Na základě informací z stackoverflow.com jsem došel k závěru, že SAX je v Javě už jenom kvůli zpětné kompatibilitě. Dále už na něm neprobíhají žádné optimalizace.

2.7.2 DOM

Zkratka DOM v anglickém originále znamená Document Object Model. DOM je standardem konsorcia W3C. Na rozdíl od SAXu DOM API načte celý XML soubor do stromové struktury v paměti počítače což je důvod, proč je paměťově velmi náročný. Obecně je používán na XML soubory menší než 10Mb. Při použití DOM API uživatel následně prochází načtenou stromovou strukturu. Velkou výhodou je možnost snadno vytvářet a upravovat XML dokumenty.

Výhody	Nevýhody
Objektový model. (Node)	Pomalý
Umožňuje se při čtení vracet.	Paměťově velmi náročný.
Podporuje validaci podle schématu	
Podpora XPath	

Tabulka 2-15 DOM výhody a nevýhody

2.7.3 StAX

StAX v anglickém originále Streaming API for XML. Jedná se o tzv. pull-parser, který poskytuje rychlé a jednoduché rozhraní pro čtení a zápis XML dokumentů. Tento přístup využívá výhody ze SAX a DOM API. Umožnuje náhodné čtení XML souboru s maximální paměťovou efektivitou. Stejně jako v SAX API musí i zde uživatel implementovat základní metody pro zpracování událostí. Názvy těchto metod jsou identické jako u SAX API.

Výhody	Nevýhody
Efektivní práce s pamětí.	Nepodporuje validaci podle schématu
Velmi rychlý.	Nepodporuje XPath

Tabulka 2-16 StAX výhody a nevýhody

2.7.4 JAXB

JAXB v anglickém originále Java Architecture for XML Binding. JAXB nabízí metody pro konverzi XML dat na objekty a naopak. Tyto konverze probíhají na základě JAXB anotací[4]. API umožňuje snadný zápis a čtení XML z mnoha zdrojů, například ze souboru a streamu. Dále JAXB nabízí nástroj XJC, který na základě XML schématu vygeneruje datové struktury s potřebnými anotacemi. Následně uživatel takto vygenerované struktury vloží do své aplikace a velmi snadno může pracovat s konkrétním typem XML souborů. Nástroj XJC je od Javy verze 1.6 součástí instalace SDK. Tento nástroj naleznete ve složce bin.

Ukázka generování datových struktur nástrojem XJC

```
xjc.exe -d C:\jaxb\gen -p sims.students C:\jaxb\*.xsd
```

Výhody	Nevýhody
Umožňuje se při čtení vracet.	Dokáže zpracovat pouze validní XML dokument.
Efektivnější práce s pamětí než DOM	
Objektový model je vygenerován z XML schématu.	

Tabulka 2-17 JAXB výhody a nevýhody

2.7.5 Použité možnosti zpracování XML

V diplomové práci jsou používány technologie StAX, DOM a JAXB. Tyto technologie jsem si vybral na základě jejich výše uvedených vlastností.

V knihovně, která je zodpovědná za ukládání dat do databáze, byly použity technologie StAX a DOM. Cílem knihovny je rozdělit vstupní soubor se všemi záznamy o studentech na jednotlivé studenty a ty pak uložit do databáze. Protože knihovna zpracovává velké XML soubory (až stovky Mb), je zbytečné celé tyto soubory načítat do paměti. Proto není vhodné řešení pomocí DOM a JAXB. Jako nejlepší řešení se tedy ukazuje použití SAX nebo StAX. Při rozhodování, která technologie bude zvolena, jsem narazil na fórum[5], kde uživatelé porovnávali výkonnost jednotlivých přístupů. Z naměřených údajů vyplývalo, že StAX má pomalejší inicializaci vnitřních objektů, ale v celkovém čtení je trochu rychlejší než SAX. Proto jsem se rozhodl pro načítání zvolit StAX přístup. DOM přístup je pak v knihovně použit pro vytváření XML dokumentu, které jsou nakonec ukládány do databáze.

[4] Anotace http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/package-summary.html
[5] Fórum https://developer.jboss.org/thread/151665?_sscc=t

V rámci aplikace pro generování testovacích dat bylo nejsnazším řešením použít technologii JAXB. Stačilo pouze z XML schémat za použití nástroje XJC vygenerovat potřebné datové typy.

2.8 Kritéria testování

Cílem diplomové práce je zjistit, které z možných uložišť je nejvhodnější pro rychlé ukládání a načítání dat. Proto byl do všech uložišť opakovaně nahráván stejný vstupní soubor.

V rámci testování efektivity ukládání dat do konkrétního uložiště rozhodují další faktory. Těmito faktory jsou použitá implementace v Javě, rychlost sítě (internetu), struktura nahrávaných data, zatížení databáze a použité databázové indexy.

Pro testování byla použita data, která byla vygenerována podle XML schémat. Tato schémata pochází ze systému SIMS (Sdružené Informace Matrik Studentů). Schémata popisují dva typy XML dokumentů.

- SIMS_matricniVystupy.xsd
- a SIMS_studiaJinychVSVystupy.xsd.

Protože testy efektivity ukládání dat mohou být silně ovlivněny rychlostí sítě, tak byly jednotlivé série testů prováděny ze stejného internetového připojení.

Dalším důležitým faktorem v reálném prostředí je vlastní zatížení databázového serveru. Jelikož se všechny testy prováděly proti testovací databázi, která nebyla jakkoliv jinak zatížená, tak výsledky testů nemůžou přesně reflektovat hodnoty reálného prostředí.

Posledním faktorem, který ovlivňuje výsledky testů je použitá implementace v Javě. V rámci Javy jde především o to, kde bude ve skutečnosti vytvořen abstraktní datový typ XMLType. Jestli se tento typ bude vytvářet na straně klienta (knihovny) nebo až v databázi ze surových dat. Vytváření datového typu XMLType na straně klienta by teoreticky mělo snižovat režii databáze na uložení záznamu.

Během testování vyhledávání dat v databázi je pak rozhodující použití indexů a způsob vyhledávání. Opět dalšími faktory, které mohou ovlivnit efektivitu prohledávání, jsou rychlost sítě a zatížení databáze.

Hlavní cílem diplomové práce je na základě získaných poznatků z testování vytvořit a otestovat modul (knihovnu), který bude v pravidelných intervalech data ze SIMS ukládat do univerzitního systému IS/STAG.

2.8.1 Vliv XML struktury na testování

Pro ověření vlivu XML struktury na efektivitu zpracování dat je zapotřebí zkoušet ukládat XML dokumenty s různě složitou strukturou. Jako reprezentant dokumentů s triviální strukturou byl zvolen dokument vygenerovaný na základě schématu SIMS_studiaJinychVSVystupy.xsd. Struktura XML dokumentů může obsahovat až 17

unikátních elementů s maximálním zanořením sedmi elementů (počítáno včetně kořenového elementu).

Komplexní strukturu XML dokumentů reprezentují dokumenty vygenerované ze schématu SIMS_matricniVystupy.xsd. Struktura dokumentů může obsahovat až 67 unikátních elementů s maximálním zanořením devíti elementů.

Podle vlastností jednotlivých uložišť by komplexnost struktury měla hlavně ovlivňovat 2.3.3 Strukturované uložiště. Toto uložiště totiž dál rozkládá XML dokument a ukládá ho do předem definovaných objektů (tabulek). Binární i nestrukturované uložiště XML dokument ukládají jako celek.

2.8.2 Vliv implementace na rychlost ukládání dat

Testování rychlosti ukládání jednotlivých uložišť není závislé pouze na typu použitého uložiště, ale i na typu použité implantace. OJDBC knihovna umožňuje několik možných implementací:

- ukládání XMLType jako String,
- ukládání XMLType jako CLOB,
- ukládání XMLType jako Stream
- a ukládání XMLType jako XMLType.

Rozdíly ve vlastní implementaci jsou popsány v kapitole 3.2 Aplikace pro testování ukládání.

Vlastní aplikace pro testování rychlosti ukládání následně vytvoří připojení do databáze a začne postupně ukládat jednotlivé studenty. Po zpracování XML dokumentu aplikace zobrazí naměřené statistiky. Výsledky určují průměrné doby potřebné na zpracování jednoho studenta. Naměřené výsledky obsahují hodnoty:

- počet uložených záznamů,
- doba běhu celé aplikace [s],
- průměrná doba zpracování XML dokumentu [ms],
- průměrná doba přípravy dotazu spolu s daty [ms]
- a průměrná doba vykonání příkazu [ms].

2.8.3 Vliv uložiště na testování

Největší podíl na rychlosti ukládání XML dokumentu má typ zvoleného uložiště. Typ zvoleného uložiště dále velmi ovlivňuje ostatní práci s XML dokumenty. Ne všechny uložiště poskytují všechny typy indexů a umožnují všechny typy operací. Pro porovnání obecné efektivity uložišť pro abstraktní datový typ XMLType bylo navrženo referenční řešení. Popis tohoto řešení naleznete v kapitole 3.3 Referenční řešení.

Cílem diplomové práce je porovnat jednotlivá uložiště mezi sebou a vybrat to, které bude nejvíce vyhovovat požadavkům na modul. Hlavní požadavky na knihovnu jsou:

- rychlé ukládání XML dokumentů
- a rychlé vyhledávání konkrétního studenta.

2.8.4 Vliv indexu na testování

Databázový index ovlivňuje hlavně rychlost vyhledávání, ale i ukládání dat. Obecně platí, že čím je nad danou tabulkou více indexů, tím je ukládání pomalejší, protože se musí jednotlivé indexy aktualizovat. Toto obecné pravidlo neplatí pro Bitmap index. Na rozdíl od ostatních indexů má Bitmap index specifické použití. Popis Bitmap indexu naleznete v kapitole 2.4.4 Bitmap index.

Efektivita databázového indexu je silně ovlivněna stavem indexu. Obecně platí, že index, který byl postupem času iterativně aktualizován, má menší efektivitu než aktuálně vytvořený a přepočítaný index. Proto jsou v rámci testování testovány dva základní případy rychlosti vyhledávání konkrétního studenta za pomoci postupně aktualizovaného indexu a rychlosti vyhledávání za pomoci aktuálně vytvořeného a přepočítaného indexu.

Dalším velmi zajímavým testem je porovnání vyváženosti B-tree indexu v rámci databáze. Protože B-tree index podporuje pouze strukturované uložiště, tak tento test není objektivní pro porovnávání uložišť jako takových, ale může naznačit, jak často je vhodné nechat index přepočítat. Základní myšlenkou testu je, že s největší pravděpodobností nebude postupně aktualizovaný index tak dobře vyvážený jako aktuálně přepočítaný index. Toto tvrzení můžeme zkusit dokázat tak, že budeme v databázi náhodně vyhledávat záznamy za pomoci postupně aktualizovaného indexu. Následně pak index přepočítáme a náhodné vyhledávání provedeme znovu. Za předpokladu, že by byl index vždy ideálně vyvážen, vyhledání každého záznamu by mělo trvat průměrně stejnou dobu.

2.8.5 Testování rychlosti vyhledávání

Na testování rychlosti vyhledávání v rámci jednotlivých uložišť byla vytvořena aplikace, která se snaží maximálně napodobit reálné vyhledávání v databázi.

Tato aplikace vyhledává studenty podle rodného čísla. Aby se vyhledávání blížilo realitě, jsou studenti vyhledávání náhodně. Test, ve kterém by se studenti načítali od prvního do posledního studenta, nám nedá objektivní výsledek ohledně efektivity hledání. Takový test by mohl v podstatě reflektovat průměrnou rychlost exportu dat z dané tabulky.

Proto byla aplikace navržena tak, aby z množiny o velikosti N náhodně vybírala rodná čísla. Dále v aplikaci není úmyslně použit předpřipravený dotaz, aby nedocházelo ke zkreslování výsledků díky optimalizaci, kterou předpřipravený dotaz poskytuje. Optimalizace předpřipraveného dotazu spočívá v tom, že je dotaz převeden do

binárního tvaru a je dočasně uložen v databázi. Tento fakt pak značně urychluje provádění dotazu. Aplikace po vyhledání 1000 náhodných studentů zobrazí naměřenou statistiku. Statistika zobrazuje průměrné doby na zpracování jednoho studenta. Statistika obsahuje následující hodnoty:

- doba běhu celé aplikace [s],
- průměrná doba vytvoření dotazu [ms],
- průměrná doba vyhledávání v databázi [ms]
- a průměrná doba zpracování výsledku dotazu [ms].

3 Realizační část

3.1 Aplikace pro generování testovacích dat

Aplikace pro generování testovacích dat dále pouze jako generátor byla vytvořena z důvodu ochrany osobních údajů jednotlivých studentů. Během zpracovávání diplomové práce jsem neměl nikdy přístup ke skutečným datům. V rámci aplikace IS/STAG jsem dostal přístup jako demo uživatel. Tento přístup umožňuje generovat XML dokument s názvem Sber. Tento dokument obsahuje náhodně promíchaná data studentů univerzity. Struktura dokumentu je velkou podmnožinou XML schématu SIMS_matricniVystupy.xsd. Na základě XML dokumentu Sber byly vygenerovány vstupní soubory, které slouží jako datový podklad pro generování testovacích dokumentů. Jednotlivé XML dokumenty jsou generovány JAXB API. Struktura vstupních souborů je popsána v následující kapitole.

3.1.1 Struktura vstupních souborů

Vstupní soubory generátoru testovacích dat tvoří šest nezávislých souborů ve formátu CSV. Tento formát byl zvolen úmyslně pro jeho velmi snadný způsob zpracování.

Vstupní soubory a popis jejich vnitřní struktury

- adresa_data.csv
  - okres; obec; castObce; ulice; uliceCislo; psc; stat
- aprobace_data.csv
  - aprobace
- etapa_data.csv
  - platnostOd; obcanstviKvalifikator; obcanstviStat; pobytVcr; jazykVyuky; mistoVyuky; formaStudia; financovani; preruseniStudia; platnostDo
- obory_data.csv
  - obor
- student_data.csv
  - jmeno; prijmeni; rodnePrijmeni; titulPred; titulZa; rodinnyStav; stredniSkola; rokMatZkousky
- studium_data.csv
  - vsFakulta; studijniProgram; zapisDoStudia; delkaStudia; novePrijaty; navazujiciStudProgram; predchoziVzdelani; odstudovanaCastStudia; vs; vos; czv; zahranici; ubytovaniVKoleji; datum; zpusob; udelenyTitul

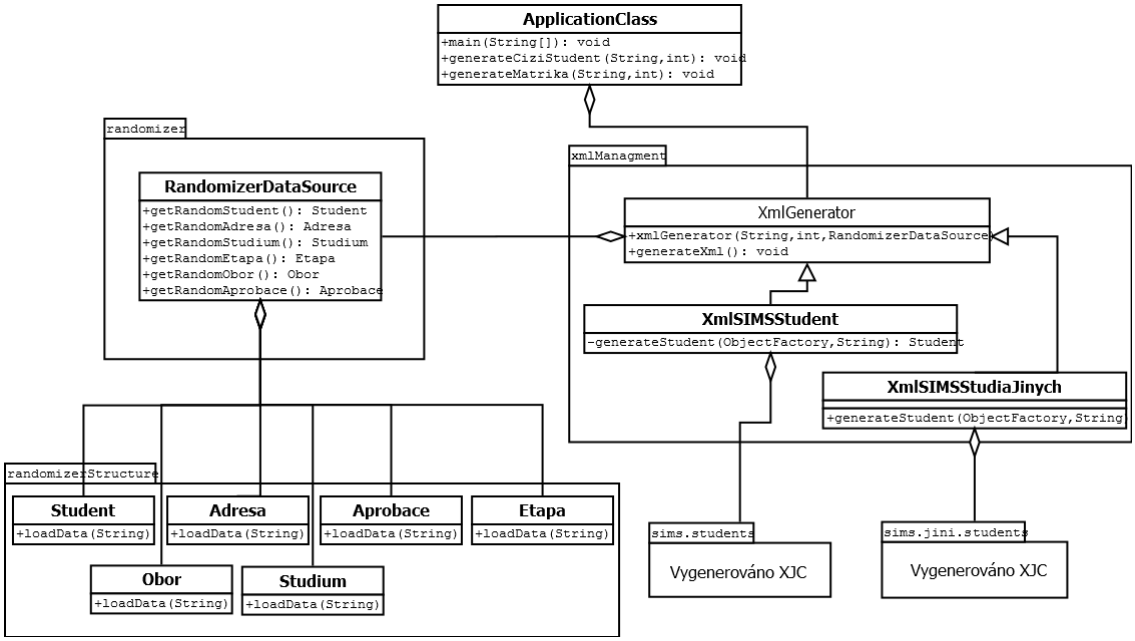
3.1.2 Výstupní data generátoru

Generátor vytváří dva typy XML souborů libovolné velikosti. Velikost souborů je určena počtem generovaných studentů. Tyto soubory svojí strukturou odpovídají schématům SIMS_matricniVystupy.xsd a SIMS_studiaJinychVSVystupy.xsd. XML struktura vygenerovaných souborů přesně odpovídá definovaným schématům (XML struktury a typy). Datová část dokumentů je náhodně vytvořená na základně vstupních souborů. Proto můžou vygenerované XML dokumenty obsahovat nelogické informace. Například student odmaturoval později, než získal vysokoškolský titul.

3.1.3 Implementace generátoru

Vlastní aplikace se skládá ze dvou částí. První část poskytuje datové struktury a metody pro získávání náhodných dat a druhá část poskytuje datové struktury pro generování XML dokumentů. Datové struktury pro vytváření XML dokumentů byly vygenerovány pomocí nástroje XJC.

Zjednodušený UML diagram aplikace



Obrázek 3-1 UML diagram generátoru

Implementace generátoru má jednu nevýhodu a to, že ve skutečnosti nedokáže vytvořit libovolně velké XML soubory. Protože generátor používá JAXB API, tak celý výstupní soubor je před uložením na disk dočasně uložen v paměti počítače. Proto při pokusu vytvořit extrémně velký výstupní soubor může dojít k selhání aplikace.

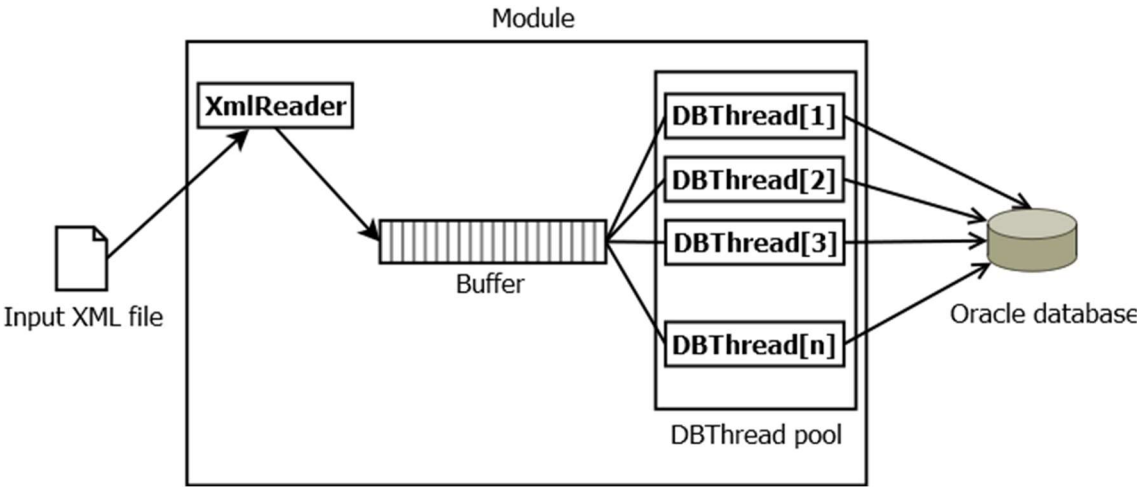
V rámci testování byly generátorem vytvořeny soubory s maximální velikostí až 1 GB. Přibližná velikost souboru ze systému SIMS je 300 MB.

3.2 Aplikace pro testování ukládání

Zadáním diplomové práce je seznámení se a otestování možných způsobů ukládání XML dokumentu do databáze Oracle 12c. Za tímto účelem byla vytvořena aplikace, která toto testování umožňuje. Aplikace obsahuje tři základní balíčky a objekty, které umožnují mapování jednotlivých XML dokumentů. Těmito balíčky jsou:

- dbManagment – obsahuje třídy pro práci s databází,
- xmlManagment – obsahuje třídy pro vytváření a načítání XML dokumentů,
- internalObjects – obsahuje obecné třídy a rozhraní pro zapouzdření načítaných XML dokumentů
- a baličky datových tříd pro mapování konkrétních XML dat.

Obecný návrh potom umožňuje snadno rozšiřovat aplikaci o zpracování jiných typů XML dokumentu nebo přidávání dalších implementací pro ukládání dat do databáze. Obecné schéma testovací aplikace je znázorněno na následujícím obrázku.



Obrázek 3-2 Obecné schéma testovací aplikace

Jedná se o více vláknovou aplikaci, která je založena na návrhovém vzoru producent-konzument. Aplikace používá jednoho producenta a N konzumentů. Producent je zastoupen třídou pro načítání a zpracování XML dokumentu. Naopak konzumenti jsou vlákna, která jsou zodpovědná za uložení XML dokumentu do databáze.

3.2.1 Abstraktní třída StaxReader

Potomci abstraktní třídy StaxReader načítají konkrétní XML soubory. Samotná třída StaxReader je oděděna od třídy Thread. Třída StaxReader implementuje následující metody:

- run() – obecná metoda pro spuštění vlákna,
- startParsing() – metoda inicializuje čtení pomocí StAX API,
- putToBuffer(MyXmlObject) – metoda vkládá již načtené záznamy (studenty) do bufferu,
- abstraktní metoda parseStudent(XmlStreamReader) – tuhle metodu musí implementovat každý potomek abstraktní třídy, metoda je zodpovědná za parsování konkrétního XML dokumentu
- a různé gettry (metody, které vrací hodnoty konkrétního atributu).

3.2.2 Abstraktní třída DBThread

Abstraktní třida DBThread má pět potomků. Každý její potomek reprezentuje jednu z možných implementací pro uložení XML dokumentu do databáze. Třída DBThread je stejně jako abstraktní třída StaxReader oděděna od třídy Thread a implementuje tyto metody:

- run() – obecná metoda pro spuštění vlákna,
- startDBConnection() – metoda vytvoří připojení do databáze,
- endDBConnection() – metoda uzavře spojení do databáze a uvolní objekt předpřipraveného dotazu,
- storeStatistics() – metoda zaznamená statistiky do sdíleného objektu Statistic
- a abstraktní metoda insertToDatabase(MyXmlObject) – každý potomek třídy musí implementovat tuhle metodu. Implementace této metody určuje použitý způsob pro uložení XML dokumentu do databáze.

V následujících kapitolách jsou popsáni jednotliví potomci třídy DBThread spolu s ukázkou implementace.

3.2.3 Insert XMLType jako String

Název třídy, která implementuje tento způsob ukládání je DBThreadClob. Tato implementace zvyšuje režii práce na straně databáze, protože vlastní převod na abstraktní datový typ XMLType dělá až databáze. Další velkou nevýhodou tohoto způsobu je fakt, že maximální velikost SQL dotazu je v základním nastavení Oracle databáze 4000 znaků. Toto omezení lze podle oficiální dokumentace v aktuální verzi databáze přenastavit.

Ukázka SQL dotazu a implementace abstraktní metody insetToDatabase

```
sql = "INSERT INTO TABLE VALUES (XMLType(?))";
sql = sql.replace("TABLE", config.getTableName());

if(this.pstmt == null)
{
  this.pstmt = (OraclePreparedStatement)
               this.connection.prepareStatement(sql);
}

this.pstmt.setString(1, xmlString);
myObject.prepareData.setEndTime();

myObject.dbExecution.setStartTime();
this.pstmt.execute();
myObject.dbExecution.setEndTime();
```

Při prvním průchodu vlákna je vytvořen předpřipravený dotaz do databáze. Tento dotaz se pak už neustále používá. Při vytváření předpřipraveného dotazu je vždy z konfiguračního souboru načteno jméno tabulky, do které se bude ukládat. Způsob vytvoření předpřipraveného dotazu mají všichni potomci třídy DBThread identický.

3.2.4 Insert XMLType jako Clob

Stejně jako předchozí způsob ukládání XML dokumentu do databáze i tento způsob zvyšuje režii na straně databáze. Název třídy, která implementuje tento způsob ukládání je DBThreadClob. Během implementace ukládání se projevilo několik problémů.

Knihovna OJDBC poskytuje předdefinovaný datový typ CLOB a dále poskytuje i metody pro vytvoření tohoto datového typu z řetězce. Problém nastává, když chceme používat objekt CLOB opakovaně v rámci předpřipraveného dotazu. Protože po vytvoření objektu CLOB se mi už nepovedlo jeho obsah nahradit jiným. Proto jsem musel vždy pro každý další Insert vytvořit nový objekt. Tento fakt způsoboval zbytečné zpomalení na straně testovací aplikace. Kvůli tomu bylo vytváření CLOB objektu a jeho vkládání nahrazeno metodou setStringForClob. Tato metoda je standardně poskytována knihovnou OJDBC od verze 6 a nezpůsobuje zbytečné zpomalení jako opakované vytváření CLOB objektu.

Ukázka zdrojového kódu

```
sql = "INSERT INTO TABLE VALUES (XMLType(?))";
sql = sql.replace("TABLE", config.getTableName());

if(this.pstmt == null)
{
 this.pstmt = (OraclePreparedStatement)
              this.connection.prepareStatement(sql);
```

```
}

this.pstmt.setStringForClob(1, xmlString);
myObject.prepareData.setEndTime();

myObject.dbExecution.setStartTime();
this.pstmt.execute();
myObject.dbExecution.setEndTime();
```

Při porovnání s ukázkou kódu z předchozí kapitoly 3.2.3 Insert XMLType jako String jsou v podstatě zdrojové kódy stejné. Jediný rozdíl tvoři použitá metoda pro vložení dat do předpřipraveného dotazu (setString vs setStringForClob).

3.2.5 Insert XMLType jako XMLType

Vytvořením abstraktního datového typu XMLType přímo na straně testovací aplikace se šetří celková režie databáze na uložení záznamu, protože databáze už nemusí sama převádět vstupní data do XMLType. Pro vytvoření XMLType objektu přímo v testovací aplikaci jsou zapotřebí tři knihovny třetích stran **XDB.jar, XMLParserV2.jar a ORAI18N.jar**. Popis těchto knihoven je uveden v kapitole 2.6.1 Knihovny třetích stran. Název třídy implementující tento způsob ukládání je DBThreadXmlType.

Ukázka zdrojového kódu

```
sql= "INSERT INTO TABLE VALUES (?)";
sql= sql.replace("TABLE",this.config.getTableName());

if(this.pstmt == null)
{
 this.pstmt = (OraclePreparedStatement)
              this.connection.prepareStatement(sql);
}

XMLType myXML = null;
myXML= XMLType.createXML(this.connection, xmlString);

this.pstmt.setObject(1,myXML);
myObject.prepareData.setEndTime();

myObject.dbExecution.setStartTime();
this.pstmt.execute();
myObject.dbExecution.setEndTime();
```

V ukázce kódu je použita statická metoda XMLType.createXML, která vytvoří XMLType objekt ze vstupního řetězce. Tato metoda nekontroluje validitu XML dokumentu, ale pouze převede vstupní řetězec na datový typ XMLType. O tom jestli je vkládaný XML dokument správně strukturovaný rozhoduje databáze. Stejně tak databáze rozhoduje, zda vkládaný soubor odpovídá registrovanému schématu. Toto ověření probíhá pouze ve strukturovaném uložišti. To znamená, že v aplikaci si uživatel

může vytvořit XMLType objekt z jakéhokoliv řetězce, ale až databáze rozhodne, zda jej uloží nebo ne. XMLType objekt je potom metodou setObject použit v předpřipraveném dotazu.

3.2.6 Insert XMLType jako Stream

Podle oficiální dokumentace je tento způsob ukládání dat obecně vhodný pro větší soubory. Název třídy, která implementuje tento typ ukládání je DBThreadStream. Existuje několik různých typů streamu, které se liší v určitých vlastnostech, například zda převádí znakovou sadu vstupních dat na znakovou sadu, kterou používá databáze. Podporované typu streamů jsou:

- asciiStream,
- binaryStream,
- characterStream
- a nCharacterStream.

Při testování byl použit characterStream.

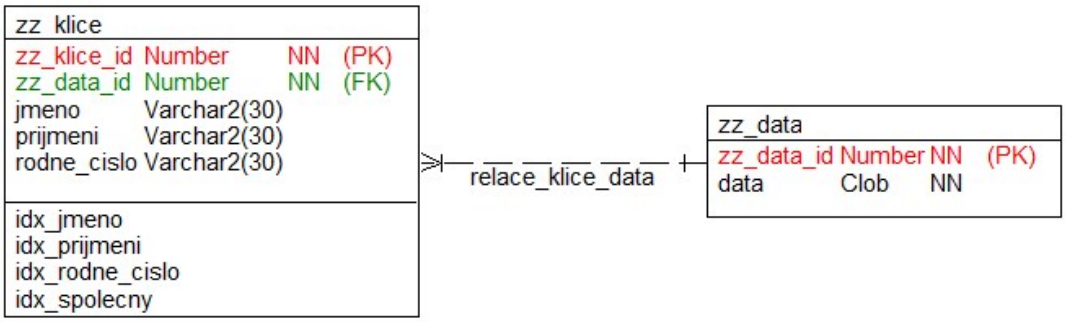
Ukázka zdrojového kódu

```
sql = "INSERT INTO TABLE VALUES (XMLType (?))";
sql= sql.replace("TABLE",this.config.getTableName());

if(this.pstmt == null)
{
 this.pstmt = (OraclePreparedStatement)
              this.connection.prepareStatement(sql);
}

Reader reader = new StringReader(xmlString);
this.pstmt.setCharacterStream(1, reader,
   xmlString.length());

myObject.prepareData.setEndTime();
myObject.dbExecution.setStartTime();
this.pstmt.execute();
myObject.dbExecution.setEndTime();
```

3.3 Referenční řešení

Pro testování obecné efektivity abstraktního datového typu XMLType bylo navrženo referenční řešení. Toto řešení velmi zjednodušeně a napodobuje chování nestrukturovaného uložiště. Řešení se skládá ze dvou tabulek. První tabulka s názvem zz_klice obsahuje vyhledávací pole, cizí klíč na konkrétní XML data a indexy. Druhá tabulka s názvem zz_data ukládá XML dokumenty za pomoci datového typu CLOB.

Obě tabulky používají prefix zz. Tento prefix byl použit pouze z praktických důvodů, aby byly tabulky umístěny na konci seznamu všech tabulek v rámci databáze. Referenční řešení pokrývá pouze zlomek toho, co umožnuje provádět nestrukturované uložiště. V žádném případě nelze XML dokumenty jakkoliv zpracovávat pomocí XQuery výrazů. Databázové schéma je zobrazeno na následujícím obrázku.



Obrázek 3-3 DB schéma referenčního řešení

Porovnání efektivity jednotlivých uložišť s referenčním řešením teoreticky dokáže, zda se při práci s XML dokumenty vyplatí používat abstraktní datový typ XMLType a jestli je řešení pomocí datového typu XMLType vždy vhodné.

Pro maximální usnadnění práce s referenčním řešením byla do databáze připravena procedura, které zajišťuje ukládání. Název procedury je zz_insert_data. SQL skripty pro vytvoření referenčního řešení včetně procedury jsou uvedeny v příloze A.

3.3.1 Implementace referenčního řešení

Referenční řešení, stejně jako implementace jednotlivých způsobů ukládání, vychází ze třídy DBThread. Název třídy, která implementuje referenční řešení je DBThreadMyClob. Tato třída, stejně jako ostatní potomci třídy DBThread, pouze implementuje abstraktní metodu insertToDatabase. V rámci této metody je naprogramován mechanismus pro vytvoření předpřipraveného dotazu a vlastní volání procedury zz_insert_data pro uložení XML záznamů do databáze.

Ukázka části zdrojového kódu metody insertToDatabase

```
String command = "{call ZZ_INSERT_DATA(?, ?, ?, ?)}";

if(this.pstmt == null)
{
 this.pstmt = (OraclePreparedStatement)
           this.connection.prepareStatement(command);
}

this.pstmt.setString(1, student.jmeno);
this.pstmt.setString(2, student.prijmeni);
this.pstmt.setString(3, student.rodneCislo);
this.pstmt.setStringForClob (4, xmlString);
myObject.prepareData.setEndTime();

myObject.dbExecution.setStartTime();
this.pstmt.execute();
myObject.dbExecution.setEndTime();
```

3.4 Aplikace pro testování vyhledávání

Pro testování rychlosti vyhledávání v rámci různých uložišť byla vytvořena aplikace. Jedná se o jedno vláknovou aplikaci, která provádí testování ve dvou krocích.

První krokem je načtení vstupního souboru, který byl uložen do databáze. Aplikace nenačítá celý vstupní souboru, ale načte z něj pouze hodnoty, podle kterých se bude vyhledávat.

Druhým krokem je samotné vyhledávání záznamů v databázi, které probíhá náhodně. Tento postup byl zvolen z důvodů, které jsou popsány v kapitole 2.8.5 Testování rychlosti vyhledávání. Aby vyhledávání mohlo skutečně probíhat náhodně, je před provedením každého Selectu z pole klíčových hodnot náhodně vybrána jedna hodnota. Tato hodnota je pak z pole odstraněna. Náhodný výběr z pole klíčových hodnot je prováděn na základě vygenerovaného indexu pomocí třídy Random. Po vyhledání konkrétního záznamu je tento záznam načten do proměnné XMLType.

Ukázka SQL dotazu pro vyhledávání v databázi

```
SELECT OBJECT_VALUE
FROM table_name
WHERE
  XMLExists('
   declare default element namespace
                                "https://sims.msmt.cz";
   /SIMS/Vystup/Student[@RodneCislo="123456789"]'
   PASSING OBJECT_VALUE);
```

3.5 Modul pro IS/STAG

Modul pro informační systém IS/STAG byl navržen na základě dosažených výsledků při porovnávání různých typů implementací. Modul využívá ukládání za pomoci datového typu XMLType, tak jak je popsáno v kapitole 3.2.5 Insert XMLType jako XMLType. Vlastní modul je vytvořen z Aplikace pro testování ukládání. Na žádost zadavatele diplomové práce modul obsahuje všechny původní třídy, které implementují jednotlivé možnosti, jak ukládat XML dokument do databáze.

Modul k ukládání dat do databáze nepoužívá třídu DBThreadXMLType ale místo ní používá rozšířenou třídu DBThreadBatchInsert, která navíc využívá dávkového zpracování dat. Dávkové zpracování dat umožnuje vložit do databáze více záznamů najednou. Díky tomu se ukládání řádově urychlí, protože se zmenší potřebná režie pro uložení každého záznamu. Třída DBThreadBatchInsert není potomek abstraktní třídy DBThread. Jedná se o potomka třídy Thread. Tento postup byl zvolen proto, aby bylo možné snadno naimplementovat zpracování SQL dotazů pomocí dávek. Ukládání do databáze za použití třídy DBThreadBatchInsert se skládá ze dvou opakujících se kroků.

První krokem je vytvoření předpřipraveného dotazu a postupné připravování SQL dávky. Údaj o velikosti dávky stejně jako údaj o počtu databázových vláken[6] nebo jménu tabulky, do které mají být ukládány XML dokumenty, je uveden v konfiguračním souboru **module.properties**.

Dávka je poslána do databáze a zpracována poté, co dosáhla požadované velikosti nebo už pro ni nejsou žádná další vstupní data. Po zpracování dávky jsou uloženy statistické údaje do instance třídy Statistics.

Dalším rozšířením oproti testovací aplikaci pro ukládání, je přidání třídy, která obsahuje statické metody pro ukládání XML souborů, které odpovídají XML schématům SIMS_matricniVystupy.xsd a SIMS_studiaJinychVSVystupy.xsd. Pro použití vytvořené knihovny stačí pouze zavolat tyto předpřipravené metody. Těmito metodami jsou:

- BasicMethods.insertMatricniVystupy(inputFile, configFile)
- a BasicMethos.insertStudiaJinychVSVystupy(inputFile, configFile).

Po zavolání předpřipravené metody se modul pokusí uložit vstupní XML soubor do databáze. Pokud dojde k nějakému problému, modul vyhodí výjimku a je na uživateli, jak se zachová. Pro správnou funkčnost knihovny musí uživatel ve svém projektu (aplikaci) mít připojené knihovny třetích stran viz kapitola 2.6.1 Knihovny třetích stran.

[6] Databázová vlákno je vlákno modulu (knihovny), které je připojeno do databáze.

Ukázka zdrojového kódu pro vytvoření SQL dávky

```
private void insertToBatch(MyXmlObject myObject)
            throws TransformerException, SQLException

{
 String xmlString;
 xmlString=this.xmlCreator.createXMLStructureToString
                                          (myObject.data);

 XMLType myXML = null;
 myXML=XMLType.createXML(this.connection,xmlString);

 this.pstmt.setObject(1,myXML);
 this.pstmt.addBatch();
 this.pstmt.clearParameters();
}
```

Velmi důležité je po každém vložení nového SQL dotazu do dávky zavolat metodu clearParameters tak, aby mohl být vložen další SQL dotaz.

Ukázka zdrojového kódu pro zpracování SQL dávky

```
private void executeBatch() throws SQLException
{
  this.pstmt.executeBatch();
  this.pstmt.clearBatch();
}
```

Opět je velmi důležité po zpracování dávky zavolat metodu clearBatch tak, aby mohla být vytvořena a zpracována další dávka.

4 Výsledky porovnávaných uložišť

Tabulky uvedené v této kapitole obsahují zaokrouhlené hodnoty. Podrobné výsledky všech měření jsou uvedeny na přiloženém DVD.

4.1 Popis testovacích strojů

Pro testování jednotlivých implementací byl použit notebook Asus N56V. Konfigurace testovací notebooku:

- Procesor Intel® Core™ i7 3610QM Processor,
- RAM 8 GB,
- Disk Samsung SSD 850 Pro 256 GB,
- OS Windows 10 Home x64
- a Java verze 1.8.0_73 64bit.

Testovací databáze byla spuštěna na univerzitním serveru. Konfigurace testovací databáze:

- Procesor Intel® Xeon® CPU E5310 (4 jádra),
- RAM 4 GB,
- OS Debian 8.3 Jessie,
- Oracle DB verze 12c Enterprise
- a SGA 1.2 GB (kolik paměti RAM využívá databáze).

Testy pro porovnávání uložišť probíhaly přes internetové připojení od UPC s rychlostí 10/100 Mb/s.

4.2 Porovnání Binárních uložišť

V aktuální verzi databáze Oracle 12c lze Binární uložiště vytvořit dvěma způsoby. Podrobný popis spolu s SQL příkazy pro vytvoření obou typů Binárního uložiště naleznete v kapitole 2.3.1 Binární uložiště. Do každého uložiště byla nahrána stejná testovací data a byl vytvořen stejný index. Jako testovací soubor byl použit sber_velka.xml. Bližší popis testovacích dat je v následující kapitole 4.3.1 Vliv XML struktury na rychlost ukládání. Podle oficiální dokumentace by mělo být nové Binární uložiště rychlejší než původní staré. V následujících tabulkách je porovnání rychlosti nahrávání dat a jejich vyhledávání.

V tabulce 4-1 je uvedena průměrná doba přípravy dotazu, která je označena jako STMT, a průměrná doba vykonání dotazu, která je označena jako EXEC. Obě hodnoty byly zaokrouhleny na dvě desetinná místa a jsou uvedeny v milisekundách.

Implementace	Typ uložiště			
	Binární - starý typ		**Binární - nový typ**	
	STMT [ms]	EXEC [ms]	STMT [ms]	EXEC [ms]
String (3.2.3)	0,51	107,82	0,55	94,62
Clob (3.2.4)	0,46	94,99	0,47	93,97
XMLType (3.2.5)	0,61	96,92	0,68	96,39
Stream (3.2.6)	0,41	92,28	0,47	89,6

Tabulka 4-1 Binární uložiště – ukládání průměrné hodnoty

V tabulce 4-2 je uvedena celková doba potřebná pro uložení celého testovacího souboru. Hodnoty jsou zaokrouhleny na celé sekundy.

Implementace	Typ uložiště	
	Binární - starý typ	**Binární - nový typ**
String (3.2.3)	163 s	142 s
Clob (3.2.4)	143 s	141 s
XMLType (3.2.5)	136 s	135 s
Stream (3.2.6)	139 s	135 s

Tabulka 4-2 Binární uložiště - ukládání celková doba

V tabulce 4-3 je uvedena průměrná hodnota pro vyhledání jednoho studenta a celková doba náhodného vyhledávání tisícovky studentů. Průměrná hodnota vyhledávání je vyjádřena v milisekundách a zaokrouhlena na dvě desetinná místa. Celková dobá je zaokrouhlena na celé sekundy.

	Typ uložiště	
	Binární - starý typ	**Binární - nový typ**
Průměrná doba vyhledání 1 studenta [ms]	201,54	156,47
Celková doba [s]	203	158

Tabulka 4-3 Binární uložiště - vyhledávání

4.2.1 Shrnutí naměřených údajů

Z naměřených údajů je patrné, že nový typ Binárního uložiště je skutečně výkonnější než jeho předchůdce. V rámci ukládání dat je rozdíl mezi uložišti minimální. Hlavní rozdíl v rychlosti se projevuje při vyhledávání. Při náhodném vyhledávání tisícovky studentů byl rozdíl skoro minuta. A to je velmi dobré zlepšení oproti původnímu uložišti. V následujících testech je vždy použit nový typ Binárního uložiště.

4.3 Test – vliv XML struktury

Testování vlivu XML struktury na rychlost ukládání a vyhledávání XML dokumentu je testováno hlavně oproti Strukturovanému uložišti. Toto uložiště je ze své podstaty nejvíce ovlivněno strukturou XML dokumentu.

4.3.1 Vliv XML struktury na rychlost ukládání

Pro testování byly vygenerovány dva testovací soubory, které mají přibližně stejnou fyzickou velikost. Přibližná velikost testovacích souborů je 100 MB. První testovací soubor byl vygenerován na základě XML schématu SIMS_matricniVystupy.xsd a obsahuje 27 000 záznamů. Tento soubor reprezentuje složitou XML strukturu. Druhý testovací soubor byl vygenerován na základě schématu SIMS_studiaJinychVSVystupy.xsd a obsahuje 100 000 záznamů. Soubor reprezentuje XML dokumenty s triviální strukturou. Tato testovací data jsou opakovaně nahrávána na Strukturované uložiště za pomoci modulu pro IS/STAG. Konfigurace modulu:

- DB vlákna 10
- a velikost dávky 100.

V tabulce 4-4 jsou uvedeny statistické hodnoty naměřené během ukládání dat do databáze. Hodnoty, které jsou uvedeny v milisekundách, byly zaokrouhleny na dvě desetinná místa. Hodnota pole Celková doba byla zaokrouhlena na celé sekundy.

Typ uložiště – Strukturované uložiště		
	Triviální XML struktura	**Složitá XML struktura**
Celková doba [s]	106	97
Průměrné doby		
Parsování XML (1 student) [ms]	0,03	0,1
Příprava DB dávky [ms]	14,03	36,83

Typ uložiště – Strukturované uložiště		
	Triviální XML struktura	**Složitá XML struktura**
Vykonání DB dávky [ms]	960,17	2573,79

Tabulka 4-4 Strukturované uložiště a XML struktura (Insert)

4.3.2 Vliv XML struktury na rychlost vyhledávání

Pro porovnání vlivu XML struktury na vyhledávání byly vygenerovány dva testovací soubory, které obsahují stejný počet záznamů. Každý soubor obsahuje 10 000 záznamů a reprezentuje jednu ze struktur. Tento přístup byl zvolen proto, aby vyhledávání probíhalo vždy na tabulkách stejné velikosti (stejný počet záznamů). Nad každou tabulkou byl spuštěn test, který provede tisícovku náhodných vyhledávání a zobrazí výslednou statistiku.

Tabulka 4-5 obsahuje pole Celková doba, které je zaokrouhleno celé sekundy, a dále obsahuje pole s průměrnými hodnotami, které jsou uváděny v milisekundách a zaokrouhleny na dvě desetinná místa.

Typ uložiště – Strukturované uložiště		
	Triviální XML struktura	**Složitá XML struktura**
Celková doba [s]	38	43
Průměrné doby		
Příprava dotazu [ms]	0,11	0,12
Vykonání dotazu [ms]	36,98	41,86
Načtení výsledku dotazu [ms]	0,11	0,11

Tabulka 4-5 Strukturované uložiště a XML struktura (Select)

4.3.3 Shrnutí naměřených údajů

Z naměřených údajů je patrné že složitost struktury v rámci Strukturovaného uložiště ovlivňuje rychlost ukládání a vyhledávání XML dokumentů. Při ukládání složité XML struktury je průměrná doba zpracování jedné dávky více jak dvakrát větší než při zpracovávání SQL dávky s triviální strukturou. Podle mého názoru je to způsobeno tím, že Strukturované uložiště si vytváří pomocné datové struktury, do kterých ukládá

jednotlivé XML dokumenty. Čím je XML struktura složitější, tím více pomocných datových struktur a delší dobu k rozparsování ukládaného XML dokumentu do pomocných struktur Strukturované uložiště potřebuje.

Výsledky testu pro ověření vlivu XML struktury na rychlost vyhledávání dopadly podle mých očekávání. Rozdíl ve vyhledávání mezi složitou a triviální strukturou byl minimální a to proto, že obě struktury mají stejný základ a vždy se hledalo podle rodného čísla daného studenta. Rozdíl ve vyhledávání mezi triviální a složitou strukturou je 4.88 milisekund. Tento rozdíl je téměř zanedbatelný.

Vliv XML struktury na Binární a Nestrukturované uložiště byl podobný jako na Strukturované uložiště. Dokument se složitější XML strukturou (fyzicky větší) se ukládal o něco déle než ten s triviální strukturou. Vyhledávání trvalo přibližně stejnou dobu. Rychlost vyhledávání je v Binárním a Nestrukturovaném uložišti ovlivněna především kvalitou XML indexu tak, jak dokazuji v kapitole 4.6 Testování rychlosti vyhledávání.

4.4 Test – vliv počtu vláken na rychlost ukládání

Testovací aplikace pro ukládání byla navržena jako více vláknová. Stejně byl navržen i modul pro systém IS/STAG, který vychází z testovací aplikace. Každé DB vlákno vytváří nové připojení do databáze a tím zvyšuje celkovou zátěž databázového serveru. Proto byl proveden test, který ukazuje urychlení ukládání dat do databáze v závislosti na počtu použitých DB vláken. Pro test byl použit testovací soubor sber_velka.xml. Popis tohoto souboru je uveden v následující kapitole 4.5 Test – vliv použití OJDBC na rychlost ukládání. Test byl prováděn na všech typech uložiští. Pro testování byla použita implementace pomocí XMLType. Popis použité implementace naleznete v kapitole 3.2.5 Insert XMLType jako XMLType.

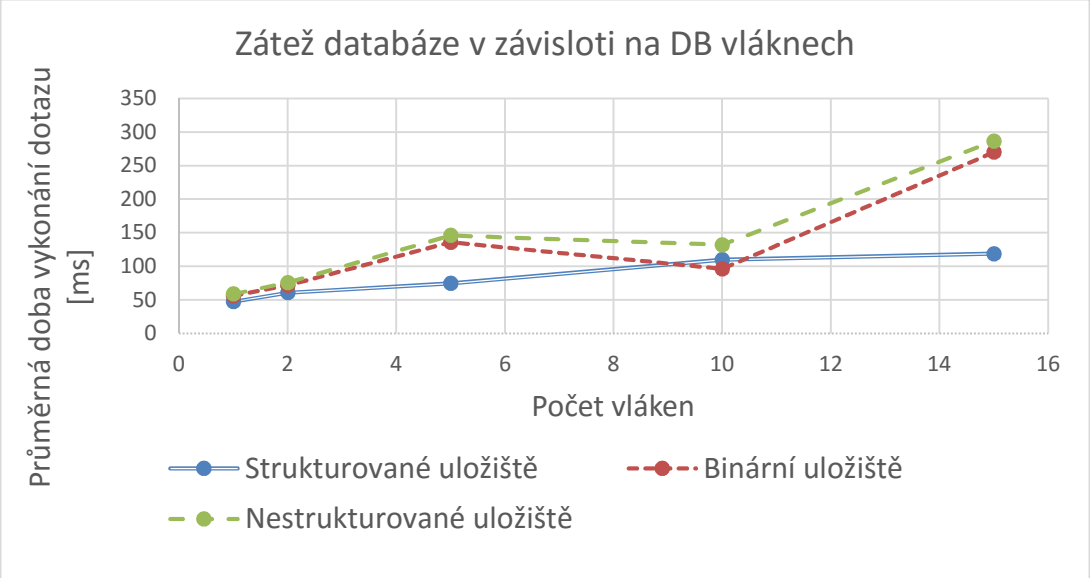
Naměřené výsledky jsou uvedeny v následující tabulce 4-6. Sloupec EXEC vyjadřuje průměrnou dobu provádění jednoho dotazu. Doba je uváděna v milisekundách a zaokrouhlena na dvě desetinná místa. Další sloupec DOBA vyjadřuje celkovou dobu potřebnou pro uložení celého testovacího XML souboru do databáze. Tato hodnota je zaokrouhlena na celé sekundy.

Počet vláken	Typ uložiště					
	Strukturované		**Binární**		**Nestrukturované**	
	EXEC [ms]	DOBA [s]	EXEC [s]	DOBA [s]	EXEC [s]	DOBA [s]
1	47,36	654	56,16	775	58,95	813
2	60,78	420	71,86	496	75,89	523

Počet vláken	Typ uložiště					
	Strukturované		**Binární**		**Nestrukturované**	
	EXEC [ms]	DOBA [s]	EXEC [s]	DOBA [s]	EXEC [s]	DOBA [s]
5	74,54	207	135,85	375	146,23	404
10	109,95	145	96,39	135	132,08	185
15	118,71	113	270,31	252	286,58	266

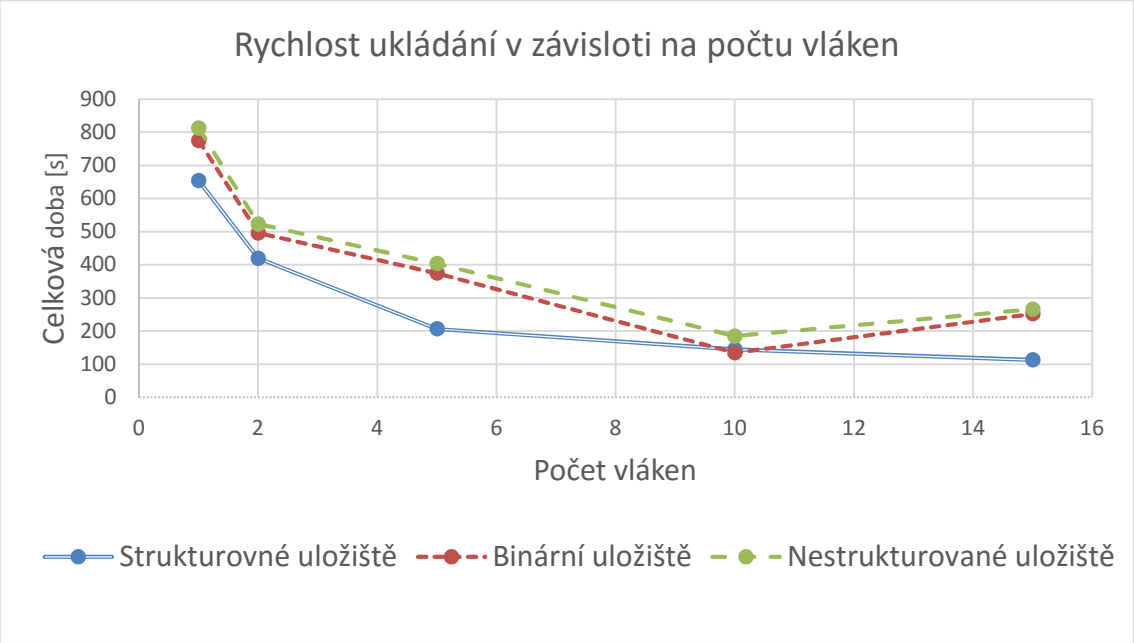
Tabulka 4-6 Vliv počtu vláken na dobu ukládání

Grafické znázornění vlivu počtu vláken na průměrnou dobu vykonání SQL dotazu



Obrázek 4-1 Graf – průměrná doba zpracování SQL dotazu

Grafické znázornění vlivu počtu DB vláken na celkovou rychlost ukládání dat do databáze



Obrázek 4-2 Graf - rychlost ukládání testovacího souboru

4.4.1 Shrnutí naměřených údajů

Z naměřených dat můžeme vidět, jak s rostoucím počtem DB vláken roste i celková rychlost ukládání dat do databáze. S rostoucím počtem vláken roste i průměrná doba potřebná pro zpracování jednotlivých SQL dotazů, protože každé nové vlákno si vytvoří vlastní připojení do databáze a tím se zvyšuje i zatížení databázového serveru. V rámci hledání optimálního počtu vláken byly testovány konfigurace: 1, 2, 5, 10 a 15 DB vláken.

Získaná data ilustrují moment, kdy počet používaných DB vláken překročí určitou mez a začne docházet ke zpomalování celého procesu ukládání. Pro Binární a Nestrukturovaná uložiště je tato mez 10 DB vláken, protože při použití 15 vláken prudce stoupne průměrná doba na zpracování jednoho SQL dotazu a tím i celková doba uložení testovacího souboru do databáze. Pro Strukturované uložiště je mezní počet vláken vyšší, protože při použití 15 DB vláken došlo k drobnému zlepšení.

Obecně se z naměřených údajů dá usuzovat, že optimální počet DB vláken je 10. Při použití více vláken je zrychlení už minimální a u Binárního a Nestrukturovaného uložiště dokonce dochází ke zpomalení.

4.5 Test – vliv použití OJDBC na rychlost ukládání

Existují minimálně čtyři způsoby, jak ukládat XML dokumenty pomocí knihovny OJDBC do abstraktního datového typu XMLType. Všechny způsoby použité v diplomové práci byly vždy testovány na třech typech uložišť (Strukturované, Binární a Nestrukturované). Pro testování byl použit soubor, který je vygenerován systémem IS/STAG. Tento soubor obsahuje 13 679 studentů a jeho přibližná velikost je 24 MB. Název souboru je sber_velka.xml, soubor je přiložen na DVD spolu s ostatními testovacími daty. Testovací soubor odpovídá svojí strukturou schématu SIMS_matricniVystupy.xsd. Všechny implementace při testování používaly stejnou konfiguraci 10 DB vláken a Single Query (= co dotaz, to uložení jednoho studenta).

Při porovnávání jednotlivých implementací byla průměrná doba zpracování XML dokumentu přibližně stejná, proto zde není uváděna. Hodnoty, které se naopak vlivem použité implementace a výběrem použitého uložiště měnily, byly: průměrná doba přípravy dotazu spolu s daty, průměrná doba vykonání příkazu a celková doba běhu testu.

V následující tabulce 4-7 jsou uvedeny průměrné doby přípravy a vykonávání SQL dotazu, které jsou ovlivněny použitou implementací a vybraným typem uložiště. Všechny hodnoty byly zaokrouhleny na dvě desetinná místa a jsou uvedeny v milisekundách. Průměrná doba přípravy dotazu je označena jako STMT a průměrná doba vykonání dotazu je označena jako EXEC.

Implementace	Typ uložiště					
	Strukturované		**Binární**		**Nestrukturované**	
	STMT [ms]	EXEC [ms]	STMT [ms]	EXEC [ms]	STMT [ms]	EXEC [ms]
String (3.2.3)	0,48	105,07	0,55	94,61	0,52	124,95
Clob (3.2.4)	0,49	104,96	0,47	93,97	0,53	126,1
XMLType (3.2.5)	0,65	109,95	0,68	96,39	0,71	132,08
Stream (3.2.6)	0,48	102,25	0,467	89,60	0,49	120,89

Tabulka 4-7 Vliv implementace na průměrnou dobu přípravy a vykonávání SQL dotazu

V tabulce 4-8, která je na následující stránce, jsou údaje o celkové době potřebné pro uložení celého testovacího souboru do databáze. Hodnoty jsou zaokrouhleny a uvedeny v celých sekundách.

Implementace	Typ uložiště		
	Strukturované	**Binární**	**Nestrukturované**
String (3.2.3)	158 s	143 s	190 s
Clob (3.2.4)	158 s	142 s	189 s
XMLType (3.2.5)	154 s	136 s	185 s
Stream (3.2.6)	154 s	135 s	181 s

Tabulka 4-8 Vliv implementace na dobu ukládání

4.5.1 Shrnutí naměřených údajů

Z naměřených údajů je patrné, že implementace ukládání na straně klienta má minimální dopad na celkovou dobu ukládání dat. Z výsledků pak dále vyplývá, že typ uložiště je hlavní faktor, který ovlivňuje celkovou dobu ukládání. Na návrh a implementaci modulu pro informační systém IS/STAG byl vybrán způsob implementace pomocí datového typu XMLType. Tento způsob implementace je pospán v kapitole 3.2.5 Insert XMLType jako XMLType. Implementace byla vybrána na základě předpokladu, že při ukládání XML dokumentu je menší režie na práci databázového serveru.

4.6 Testování rychlosti vyhledávání

V rámci základního porovnání uložišť byla použita stejná testovací data jako v kapitole 4.3.1 Vliv XML struktury na rychlost ukládání. Do Binárního a Nestrukturovaného uložiště byl přidán XML index. V následujících tabulkách jsou uvedeny průměrné doby potřebné na vyhledání jednoho studenta. Studenti jsou vyhledávání na základě rodného čísla. Všechny uvedené hodnoty jsou zaokrouhleny maximálně na dvě desetinná místa.

4.6.1 Obecný XML index

Test na vyhledávání za podpory obecného XML indexu byl proveden pouze na Binárním a Nestrukturovaném uložišti, protože na Strukturovaném uložišti nelze obecný XML index vytvořit a z podstaty vlastností Strukturovaného uložiště by to ani nedávalo smysl. Na Binárním i Nestrukturovaném uložišti byl obecný XML index vytvořen až po vložení záznamů, protože během práce s databází jsem si všiml, že obecný XML index, který byl vytvořen postupně tak, jak se přidávala data do tabulky, je úplně k ničemu. Tento index vůbec neurychluje vyhledávání dat a je potřeba ho přepočítat nebo smazat a vytvořit znova.

SQL kód pro vytvoření obecného XML indexu

```
/* obecný XML index */
CREATE INDEX index_name ON table_name (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLINDEX
```

V následující tabulce 4-9 je uvedena průměrná doba pro vyhledání studenta, doba pro vytvoření indexu a celková doba náhodného vyhledávání tisícovky studentů.

	Typ uložiště	
	Binární	**Nestrukturované**
Průměrná doba vyhledání 1 studenta [ms]	121,6	347,61
Vytvoření indexu [s]	31,26	50,04
Celková doba [s]	123	349

Tabulka 4-9 Doba vyhledávání s obecným XML indexem

4.6.2 Definovaný XML index

Při vytváření XML indexu je možné přesně určit XML strukturu, kterou bude daný index ve skutečnosti indexovat. Další zajímavostí je, že definovaný XML index netrpí stejnou degradací jako obecný XML index. To znamená, že pokud je index tvořen postupným přidáváním dat do tabulky, tak stále plní svoji funkci a urychluje vyhledávání.

SQL kód pro vytvoření přesně definovaného XML indexu

```
/* XML index s definicemi */
CREATE INDEX index_name ON table_name (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLIndex
PARAMETERS
 ('PATHS (INCLUDE (/SIMS/Vystup/Student/@RodneCislo)
 NAMESPACE MAPPING (xmlns="https://sims.msmt.cz"))');
```

V následující tabulce 4-10 jsou uvedeny naměřené hodnoty vyhledávání za podpory definovaného XML indexu, který byl tvořen postupným přidáváním dat. V tabulce je uvedena průměrná doba pro vyhledání jednoho studenta a celková doba náhodného vyhledávání tisícovky studentů.

	Typ uložiště	
	Binární	**Nestrukturované**
Průměrná doba vyhledání 1 studenta [ms]	156,47	158,02
Celková doba [s]	158	160

Tabulka 4-10 Doba vyhledávání s definovaným XML indexem

4.6.3 Přepočítaný XML index

Teorie tvrdí, že po přepočítání používaných indexů by se měla zlepšit rychlost vyhledávání. Konkrétní index můžeme nechat databázi kdykoliv přepočítat. Po zadání následujícího SQL příkazu začne databáze přepočítávat daný index. Doba, než je index přepočítán, závisí na počtu záznamů uložených v dané tabulce.

```
ALTER INDEX index_name REBUILD;
```

Tabulka 4-11 zobrazuje naměřené hodnoty, které byly získány po přepočítání indexů a celkovou dobu potřebnou pro náhodné vyhledání tisícovky studentů.

	Typ uložiště			
	Strukturované	**Strukturované**[7]	**Binární**	**Nestrukturované**
Průměrná doba vyhledání 1 studenta [ms]	43,05	42,82	124,12	171.68
Celková doba [s]	45	44	126	173

Tabulka 4-11 Doba vyhledávání po přepočítání indexů

Při porovnání hodnot z předchozí kapitoly 4.6.2 Vliv XML struktury na rychlost vyhledávání je patrné zlepšení pro Binární uložiště, ale naopak v rámci Nestrukturovaného uložiště došlo ke značnému zhoršení. Proč k tomuto zhoršení došlo po přepočítání indexů, nedokážu vysvětlit. I po opakovaném pouštění testů jsem docházel k podobným výsledkům. Možné vysvětlení, které mě napadá je, že po

[7] Strukturované uložiště po přepočítání indexů.

přepočítání se změnila fyzická velikost indexu a index se začal hůře cachovat v paměti, ale toto vysvětlení je čirá spekulace.

V rámci Strukturovaného uložiště došlo po přepočítání indexu pouze k minimálnímu zlepšení v rychlosti vyhledávání. Při použití větší množiny testovacích dat by toto zlepšení bylo pravděpodobně výraznější. Pro množinu dat o velikosti 13 679 záznamů se index přepočítávat nevyplatí.

4.6.4 Porovnání uložišť

Pro všechny uložiště byly vybrány ty nejlepší výsledky testů. V následující tabulce 4-12 je uvedena průměrná doba vyhledání jednoho studenta a celková doba pro náhodné vyhledání tisícovky studentů.

	Typ uložiště		
	Strukturované	**Binární**	**Nestrukturované**
Průměrná doba vyhledání 1 studenta [ms]	42,82	121,6	158.02
Celková doba [s]	44	123	160

Tabulka 4-12 Porovnání uložišť

Z tabulky 4-12 je patrné, že strukturované uložiště je nejrychlejší možnou variantou pro vyhledávání XML dokumentů. Průměrně je až třikrát rychlejší než ostatní uložiště.

4.7 Porovnání uložišť s datovým typem CLOB

Cílem testu je zjistit, zda je použití abstraktního datového typu XMLType vždy vhodným řešením při zpracování XML dokumentů pomocí Oracle databáze. Proto bylo navrženo referenční řešení. Popis referenčního řešení je uveden v kapitole 3.3 Referenční řešení. Test byl proveden za pomoci testovací aplikace pro ukládání s použitím 5 DB vláken. Jako testovací množina dat byl použit soubor sber_velky.xml. Popis testovacích dat naleznete v kapitole 4.5 Test – vliv použití OJDBC na rychlost ukládání.

Porovnání rychlosti ukládání XML dokumentu mezi referenčním řešením a jednotlivými uložišti je uvedeno v následující tabulce 4-13. Hodnoty, které jsou uváděny v milisekundách, jsou zaokrouhleny na dvě desetinná místa.

		Typ uložiště[8]		
	ref. řešení	**Strukturované**	**Binární**	**Nestrukturované**
Celková doba [s]	211	207	375	404
Průměrné doby				
Parsování XML 1 student [ms]	0,14	0,13	0,13	0,13
Příprava DB dotazu [ms]	0,83	0,55	0,55	0,80
Vykonání DB dotazu [ms]	75,81	74,54	135,85	146,23

Tabulka 4-13 Porovnání ref. řešení s uložišti (Insert)

Rychlost vyhledávání byla standardně testována za pomoci tisícovky náhodných vyhledávání. Pro jednotlivá uložiště byly vybrány hodnoty po přepočítání indexu viz kapitola 4.6.3 Přepočítaný XML index.

		Typ uložiště		
	ref. řešení	**Strukturované**	**Binární**	**Nestrukturované**
Průměrná doba vyhledání 1 studenta [ms]	98,47	42,82	124,12	171,68
Celková doba [s]	100	44	126	137

Tabulka 4-14 Porovnání ref. řešení s uložišti (Select)

[8] Pro ukládání XML dokumentů byla použita implementace Insert XMLType jako XMLType.

4.7.1 Shrnutí naměřených údajů

Z naměřených údajů vyplývá, že Strukturované uložiště vždy poráží referenční řešení. Referenční řešení bylo navrženo jako odlehčená verze Nestrukturovaného uložiště a je, při porovnání výsledků, vždy lepší než jeho předloha. Nicméně v reálném prostředí by bylo více či méně k ničemu. Například při vytváření archivu XML dokumentů bych určitě šel cestou Binárního uložiště, než vytvářet nějakou relační obálku jako v případě referenčního řešení. Binární uložiště poskytuje celkem vyvážený poměr rychlostí (ukládání vs. vyhledávání). Tento test slouží jako důkaz, že je při práci s XML dokumenty vždy lepší využívat abstraktního datového typu XMLType, než se snažit vytvářet jeho náhrady za pomocí primitivních datových typů jakými jsou CLOB a jiné.

5 Výsledky modulu

5.1 Testování modulu s různými uložišti

Pro testování byl vygenerován soubor s testovacími daty. Tento soubor obsahuje 80 000 studentů a byl vytvořen na základě schématu SIMS_matricniVystupy.xsd. Velikost souboru je přibližně 320 MB. Tato velikost odpovídá reálné velikosti souboru z informačního systému SIMS.

5.1.1 Test rychlosti ukládání dat

Test rychlosti ukládání dat byl proveden v rámci univerzitní sítě Eduroam dne 14.4.2016. Univerzitní síť má přibližně 4 krát větší rychlost nahrávání (upload) než moje domácí sít. Pro testy byla zvolena následující konfigurace modulu:

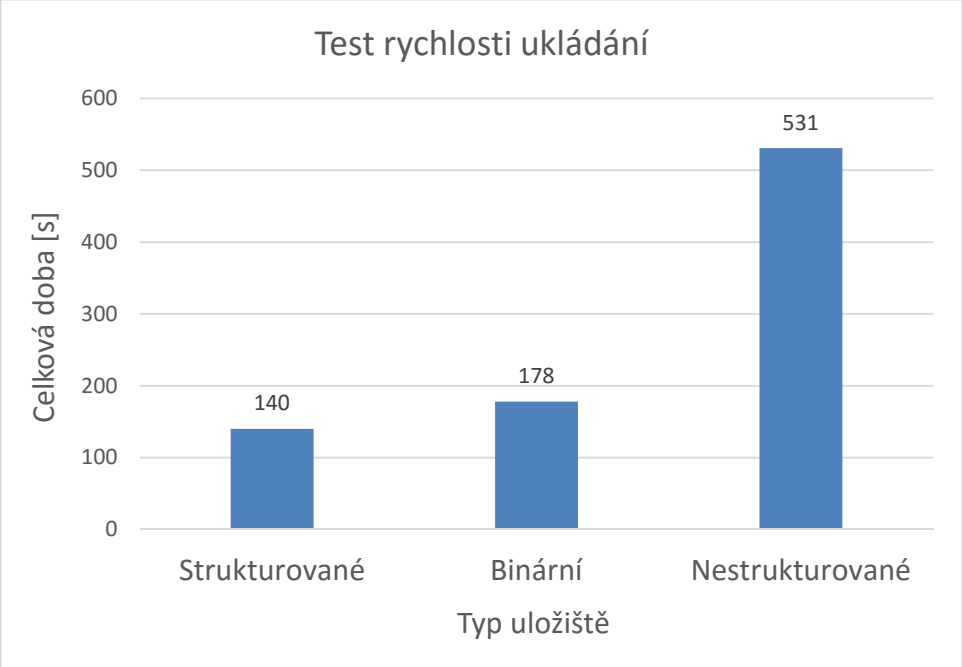
- počet vláken 10
- a velikost dávky 100.

Tabulka 5-1 zobrazuje naměřené časy dosažené při ukládání testovacího souboru do databáze. Hodnoty uvedené v milisekundách byly zaokrouhleny na dvě desetinná místa.

Typ uložiště	**Strukturované**	**Binární**	**Nestrukturované**
Celková doba [s]	140	178	531
Průměrné doby			
Parsování XML (1 student) [ms]	0,1	0,1	0,11
Příprava DB dávky [ms]	31,16	33,92	29,7
Vykonání DB dávky [ms]	1671,8	2149,96	6560,22

Tabulka 5-1 Modul - test rychlosti ukládání dat

Grafické znázornění dat



Obrázek 5-1 Graf - test rychlosti ukládádní

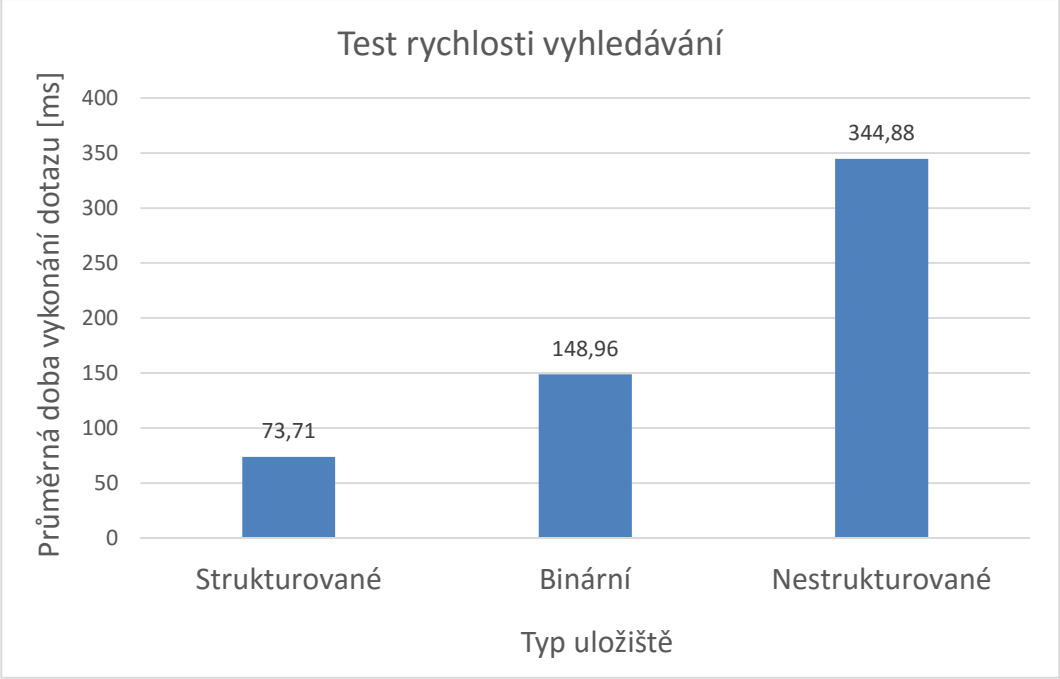
5.1.2 Test rychlosti vyhledávání dat

Pro testování rychlosti vyhledávání dat byla použita Aplikace pro testování vyhledávání. Popis aplikace naleznete v kapitolách 2.8.5 a 3.4. Při testování rychlosti vyhledávání byl z Binárního a Nestrukturovaného uložiště odstraněn obecný XML index a byl nahrazen XML indexem, ve kterém je definovaný jmenný prostor a konkrétní atribut. Jednotlivé XML dokumenty byly vyhledávány na základě rodného čísla. Výsledky testu jsou uvedeny v následující tabulce 5-2.

Typ uložiště	**Strukturované**	**Binární**	**Nestrukturované**
Celková doba [s]	75	151	347
Průměrné doby			
Příprava dotazu [ms]	0,095	0,125	0,138
Vykonání dotazu [ms]	73,71	148,96	344,88
Načtení výsledku dotazu [ms]	0,137	0,145	0,148

Tabulka 5-2 Modul - test rychlosti vyhledávání dat

Grafické znázornění dat



Obrázek 5-2 Graf - test rychlosti vyhledávání

5.1.3 Optimální konfigurace modulu

Při testování modulu pro IS/STAG byla zvolena konfigurace 10x100 (10 DB vláken s SQL dávkou velikostí 100). Hledání optimálního počtu vláken je uvedeno v kapitole 4.4 Test – vliv počtu vláken na rychlost ukládání. Optimální velikost SQL dávky je závislá na rychlosti sítě a výkonu databázového serveru. Během testování jsem pozoroval zatížení databázového serveru nástrojem EM Express. Tento nástroj slouží k monitorování Oracle databázových serverů. EM Express poskytuje mnoho různých pohledů na aktuální zatížení databáze. Pro mě bylo nejzajímavější aktuální vytížení databázových CPU jader, protože se zvětšující se dávkou roste i vytížení jader. Když jsem zkoušel různé kombinace počtu vláken a velikost SQL dávky od kombinací 1x1000 až k 10x200, tak jsem došel k závěru, že databázový server byl optimálně vytížen při kombinacích 10x100 nebo 5x200, kdy při tomto vytížení zbývalo serveru přibližně 40% volné kapacity.

6 Závěr

Výsledkem diplomové práce je knihovna a generátor testovacích dat. Knihovna pokrývá aktuálně dostupné implementace ukládání XML záznamů do Oracle databáze v rámci programovacího jazyka Java.

Hlavním cílem diplomové práce bylo vytvořit takové řešení, které urychlí proces ukládání XML souborů do databáze. Myslím si, že tento cíl jsem splnil. Teoreticky z původních 20 až 30 minut se doba potřebná pro uložení dat zkrátila přibližně na 3 minuty. Hotové řešení (Modul), které implementuje knihovnu, je součástí přiloženého DVD.

Dalším cílem práce bylo porovnat dostupné možnosti ukládání XML dokumentů do Oracle databáze. Momentálně existují 4 možnosti jak XML soubory do databáze uložit a to vytvořením vlastního řešení pomocí datového typu CLOB nebo využitím abstraktního datového typu XMLType (Strukturované, Binární a Nestrukturované uložiště). Osobně mě při porovnání uložišť velmi překvapilo, že Strukturované uložiště vždy dosahovalo nejlepších výsledků. Očekával jsem, že Strukturované uložiště bude vždy nejrychlejší v rámci vyhledávání dat, ale fakt, že je i nejrychlejší pro ukládání XML dokumentů, mě skutečně překvapil. Pokud bych měl volit mezi uložišti, která bych použil pro nějaký projekt, tak mými favority by jednoznačně byly Strukturované a Binární uložiště. Strukturované uložiště poskytuje výbornou rychlost pro ukládání a vyhledávání XML dokumentů, ale bohužel vyžaduje definici XML schématu zpracovaného XML dokumentu, takže do něj nelze ukládat různé XML dokumenty. Naopak Binární uložiště umožňuje zpracovávat různé XML dokumenty a poskytuje rozumnou rychlost ukládání a vyhledávání záznamů.

Při porovnávání vlivu složitosti XML struktury na rychlost ukládání a vyhledávání jsem dospěl k očekávánému závěru, že čím je XML dokument větší a složitější, tím déle trvají i prováděné operace.

V rámci diplomové práce jsem narazil na několik problémů. Nejvíce záludným problém asi bylo to, že pro některé XQuery výrazy databáze vracela nesmyslné chyby. Například, že v daném XML dokumentu neexistuje požadovaný element. Tyto chyby databáze vracela vždy pouze pro jeden typ uložiště, v ostatních uložištích nad stejnými testovacími daty XQuery výraz fungoval bez problémů. Nejprve jsem toto chování přisuzoval nějaké chybě databáze (bug), ale na internetu jsem nikde nenašel zmínky, že by měl stejný problém i někdo jiný. Nakonec jsem přišel na velmi důležitý fakt a to ten, že když XML dokument deklaruje svoje jmenné prostory, tak je musíte v XQuery výrazu deklarovat taky. Nejhorší na tom je, že přibližně 70% XQuery výrazů bez deklarace jmenných prostorů funguje správně a člověk si toho hned nevšimne. A v takové situaci se pak chyba celkem těžko hledá.

Přehled zkratek

Zkratka	Celé jméno
ASM	Automatic Storage Management
DB	databáze
DOM	Document Object Model
DTD	Document Type Definition
IETF	Internet Engineering Task Force
JAXB	Java Architecture for XML Binding
SAX	Simple Api for XML
SGA	System Global Area
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
StAX	Streaming API for XML
W3C	World Wide Web Consortium
WebDav	Web-based Distributed Authoring and Versioning
XML	eXtensible Markup Language
XSD	XML Schema

Literatura

[1] **Herout, Pavel.** *Přednášky z KIV/JXT.* Plzeň : Západočeká univerzita v Plzni, 2012.
[2] Wikipedia. *Wikipedia XML.* [Online] [Citace: 23. 03 2016.] https://cs.wikipedia.org/wiki/Extensible_Markup_Language.
[3] Document type definition. *Wikipedia.* [Online] [Citace: 31. 3 2016.] https://en.wikipedia.org/wiki/Document_type_definition.
[4] Introduction to Oracle Database. *Oracle Help Center.* [Online] [Citace: 23. 03 2016.] https://docs.oracle.com/database/121/CNCPT/intro.htm#CNCPT958.
[5] Introducing Oracle XML DB. *Oracle XML DB.* [Online] [Citace: 15. 03 2016.] https://docs.oracle.com/cd/A97630_01/appdev.920/a96620/xdb01int.htm.
[6] XMLChooseStorage. *oracle.* [Online] [Citace: 24. 03 2016.] http://www.oracle.com/technetwork/database-features/xmldb/xmlchoosestorage-v1-132078.pdf.
[7] Indexing XMLType Data (11g). [Online] [Citace: 03. 04 2016.] http://docs.oracle.com/cd/B28359_01/appdev.111/b28369/xdb_indexing.htm#CHDEADIH.
[8] Indexes for XMLType Data (12c). *Oracle.* [Online] [Citace: 04. 04 2016.] http://docs.oracle.com/database/121/ADXDB/xdb_indexing.htm#ADXDB0500.
[9] Oracle B-Tree Index. *Toad World.* [Online] [Citace: 03. 04 2016.] http://www.toadworld.com/platforms/oracle/w/wiki/11001.oracle-b-tree-index-from-the-concept-to-internals.
[10] **Niemiec, Richard.** About Oracle Function-based Indexes. *Logical Read.* [Online] [Citace: 27. 04 2016.] http://logicalread.solarwinds.com/oracle-11-g-function-based-indexes-mc02/#.VyBY9TCLTW8.
[11] **Niemiec, Richard.** Understanding Oracle Bitmap Indexes. *http://logicalread.solarwinds.com.* [Online] [Citace: 03. 04 2016.] http://logicalread.solarwinds.com/oracle-11g-bitmap-indexes-mc02/#.VwC3uaSLTW8.
[12] Oracle Bitmap Index Techniques. [Online] [Citace: 03. 04 2016.] http://www.dba-oracle.com/oracle_tips_bitmapped_indexes.htm.
[13] **Hall, Tim.** Full Text Indexing using Oracle Text. *Oracle-base.* [Online] [Citace: 03. 04 2016.] https://oracle-base.com/articles/9i/full-text-indexing-using-oracle-text-9i.
[14] W3C XML Query (XQuery). *W3C.* [Online] [Citace: 05. 04 2016.] https://www.w3.org/XML/Query/.
[15] SQL/XML Functions XMLQUERY, XMLTABLE, XMLExists, and XMLCast. *XML DB Developer's Guide.* [Online] [Citace: 10. 03 2016.] http://docs.oracle.com/database/121/ADXDB/xdb_xquery.htm#ADXDB5094.
[16] **Patel, Viral.** Oracle XMLTable Tutorial with Example. *VIRALPATEL.NET.* [Online] [Citace: 09. 04 2016.] http://viralpatel.net/blogs/oracle-xmltable-tutorial/.

Přílohy

Seznam příloh: Příloha A – SQL skripty referenčního řešení
Příloha B – SQL skripty uložišť
Uživatelská dokumentace Modul
Uživatelská dokumentace Generátor

Příloha A – SQL skripty referenčního řešení

```
CREATE TABLE "SYSTEM"."ZZ_KLICE"
(
        "ZZ_KLICE_ID" NUMBER NOT NULL ENABLE,
        "JMENO" VARCHAR2(50 BYTE) NOT NULL ENABLE,
        "PRIJMENI" VARCHAR2(50 BYTE) NOT NULL ENABLE,
        "OSOBNICISLO" VARCHAR2(50 BYTE) NOT NULL ENABLE,
         CONSTRAINT "ZZ_KLICE_PK" PRIMARY KEY ("ZZ_KLICE_ID"));
CREATE INDEX "SYSTEM"."JMENO_STUDENTA" ON "SYSTEM"."ZZ_KLICE" ("JMENO") ;
CREATE INDEX "SYSTEM"."OSOBNI_CISLO" ON "SYSTEM"."ZZ_KLICE" ("OSOBNICISLO") ;
CREATE INDEX "SYSTEM"."PRIJMENI_STUDENTA" ON "SYSTEM"."ZZ_KLICE" ("PRIJMENI") ;

CREATE OR REPLACE NONEDITIONABLE TRIGGER "SYSTEM"."AI_ZZ_KLICE_ID"
before insert on "SYSTEM"."ZZ_KLICE"
for each row
begin
  if inserting then
    if :NEW."ZZ_KLICE_ID" is null then
      select ZZ_KLICE_SEQ.nextval into :NEW."ZZ_KLICE_ID" from dual;
    end if;
  end if;
end;

ALTER TRIGGER "SYSTEM"."AI_ZZ_KLICE_ID" ENABLE;

CREATE TABLE "SYSTEM"."ZZ_DATA"
(
        "ZZ_DATA_ID" NUMBER NOT NULL ENABLE,
        "XML_STUDENT" CLOB NOT NULL ENABLE,
         CONSTRAINT "ZZ_DATA_FK1" FOREIGN KEY ("ZZ_DATA_ID")
         REFERENCES "SYSTEM"."ZZ_KLICE" ("ZZ_KLICE_ID") ENABLE) ;

CREATE OR REPLACE PROCEDURE ZZ_INSERT_DATA
(
        INJMENO IN VARCHAR2,
        INPRIJMENI IN VARCHAR2,
        INOSOBNICISLO IN VARCHAR2,
        INXML_FILE IN CLOB
) AS
        myKey NUMBER;
BEGIN
  INSERT INTO ZZ_KLICE (JMENO, PRIJMENI, OSOBNICISLO) VALUES (INJMENO, INPRIJMENI,
                                                INOSOBNICISLO) RETURNING ZZ_KLICE_ID INTO myKey;
  INSERT INTO ZZ_DATA (ZZ_DATA_ID, XML_STUDENT) VALUES (myKey, INXML_FILE);
END ZZ_INSERT_DATA;
```

Příloha B – SQL skripty uložišť

Zdrojové SQL kódy pro vytvoření jednotlivých uložišť jsou uvedeny a podrobně pospány v diplomové práci kapitola 2.3 Oracle XMLType.

Uživatelská dokumentace - Modul

Obsah

1 Obecný popis modulu

Modul pro informační systém IS/STAG poskytuje pouze UI ke knihovně. Knihovna bude dále označována jako DP knihovna. DP knihovna jako taková je klíčovým prvkem. Tato knihovna zajišťuje parsování vstupních XML souborů, které odpovídají určitým XML schématům, a jejich nahrávání do Oracle databáze. Aktuálně podporovaná XML schémata:

- SIMS_matricniVystupy.xsd
- a SIMS_studiaJinychVSVystupy.xsd.

XML schémata nalezneme ve složce **modul/xsd**. Modul má přímo v sobě vložené všechny potřebné knihovny třetích stran. Tento modul nalezneme ve složce **modul/bin**.

Knihovny třetích stran spolu s DP knihovnou nalezneme v **modul/lib**.

1.1 Požadavky

1.1.1 Java

Modul i DP knihovna byly implementovány za pomoci programovacího jazyka Java. Pro vývoj a testování bylo použita aktuální verze Javy **1.8.0_73**. Pro spuštění modulu nebo pro práci se samotnou DP knihovnou musíme mít v počítači nainstalovanou Javu a správně nastavené proměnné prostředí (PATH).

1.1.2 Konfigurace modulu (DP knihovny)

Samotná funkce modulu není podmíněna pouze vstupními parametry při spuštění, ale také nastavením v konfiguračním souboru. Tento soubor musí být vždy ve stejné složce jako modul (DP knihovna). Název konfiguračního souboru je **module.properties.** Konfigurační soubor obsahuje následující nastavení.

```
dbUser=system
sizeOfBatch=100
dbPassword=tajne_heslo
countOfThread=10
tableNameStudiaJinych=A_JINY_STUDENT_STRUC
tableNameMatrika=A_STUDENT_STRUC
dbLink=jdbc\:oracle\:thin\:@avtest.zcu.cz\:1521\:xmltest
bufferSize=500
```

- dbUser – přihlašovací jméno do databáze
- dbPassword – databázové heslo

- sizeOfBatch – velikost SQL dávky (kolik se bude najednou vkládat záznamů)
- countOfThread – počet použitých DB vláken (kolik vláken najednou bude pracovat s DB, každé vlákno si vytváří vlastní připojení do DB a je samostatné)
- tableNameStudiaJinych - název tabulky pro ukládání XML dokumentů, které odpovídají XML schématu SIMS_studiaJinychVSVystupy.xsd
- tableNameMatrika - název tabulky pro ukládání XML dokumentů, které odpovídají XML schématu SIMS_matricniVystupy.xsd
- dbLink – cesta k databázi
- bufferSize – velikost vnitřního bufferu DP knihovny (do bufferu jsou ukládány XML dokumenty před uložením do DB)

1.1.3 Aktuální konfigurace modulu (DP knihovny)

Aktuální konfigurace modulu (DP knihovny) je nastavena pro testovací databázi, která byla poskytnuta zadavatelem diplomové práce. **Pro použití testovací databáze musíme být připojeni do univerzitní sítě.** Připojit se můžeme pomocí VPN nebo přímo přes univerzitní připojení například využitím sítě Eduroam. Aplikaci pro vytvoření VPN připojení nalezneme ve složce **sw/vpn**. Jedná se o aplikaci Cisco AnyConnect Secure Mobility Client.

V testovací databázi jsou připraveny následující tabulky:

- A_JINY_STUDENT_STRUC
  - Strukturované uložiště, schéma SIMS_studiaJinychVSVystupy.xsd
- A_STUDENT_BIN,
  - Binární uložiště, Basic File (starý způsob)
- A_STUDENT_BIN_SEC,
  - Binární uložiště, Secure File (nový způsob)
- A_STUDENT_CLOB
  - Nestrukturované uložiště
- a A_STUDENT_STRUC
  - Strukturované uložiště, schéma SIMS_matricniVystupy.xsd

1.1.4 Knihovny třetích stran

Modul (DP knihovna) využívá knihovny třetích stran. Tyto knihovny naleznete ve složce **module/lib**. Samotný modul má všechny potřebně knihovny přímo v sobě. Seznam použitých knihoven třetích stran:

- ojdbc6.jar,
- xdb6.jar
- a xmlparserv2.jar.

2 Práce s modulem

Základní spuštění modulu bez parametrů

```
C:\Users\radek\Desktop\2016-04-14\2016-04-14\app>java -jar modul.jar
###Napoveda###

Parametry
 -h                     Zobrazeni napovedy
 -f jmeno_souboru       Vstupni XML soubor
 -t typ                 Typ vstupnich dat (matrika || studiajinych)
 -c jmeno_souboru       Konfiguracni soubor
 -s                     Zobrazeni statistiky

Priklad pouziti
 java -jar aplikace.jar -t matrika -f matrika.xml -c module.properties -s
 java -jar aplikace.jar -t studiaJinych -f studiaJinych.xml -c module.properties -s

C:\Users\radek\Desktop\2016-04-14\2016-04-14\app>
```

Obrázek 2-1 Spuštění modulu

Pokud spustíme modul bez jakéhokoliv parametru, tak modul automaticky zobrazí nápovědu s popisem všech dostupných parametrů, jak vidíme na přiloženém obrázku.

2.1 Dostupné parametry

Modul má několik vstupních parametrů viz následující tabulka.

Parametr	Popis
-h	Zobrazení nápovědy
-f <název_souboru>	Umístění vstupního XML souboru
-t <typ>	Typ výstupního XML souboru (matrika || studiajinych)
-c <název_konfigurace>	Umístění konfiguračního souboru
-s	Zobrazení statistických údajů

Obrázek 2-2 Dostupné parametry modulu

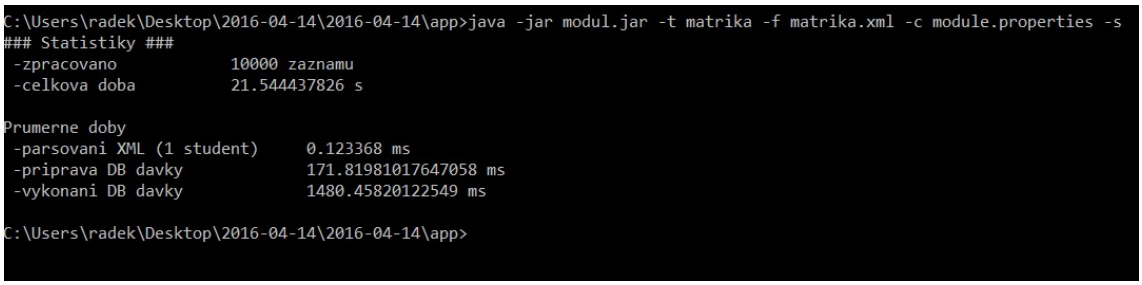
2.1.1 Ukázka použití modulu

Cílem je uložit vstupní soubor matrika.xml do databáze.

Příkaz

```
java -jar modul.jar -t matrika -f matrika.xml – c
module.properties -s
```

Ilustrační obrázek



Obrázek 2-3 Modul - ukázka statistických dat

Uživatelská dokumentace - Generátor

Obsah

1 Obecný popis aplikace

Jedná se o konzolovou aplikaci, která umožňuje generovat testovací data podle následujících XML schémat:

- SIMS_matricniVystupy.xsd
- a SIMS_studiaJinychVSVystupy.xsd.

Tyto schémata jsou uložena na DVD ve složce **generator/xsd**. Vlastní spustitelný soubor aplikace naleznete ve složce **generator/bin**.

1.1 Požadavky aplikace

1.1.1 Java

Pro spuštění aplikace musíme mít v počítači nainstalovanou Javu a nastavené proměnné prostředí (PATH). Aplikace byla vyvíjena a testována na aktuální verzi **1.8.0_73**.

1.1.2 Vstupní soubory

Další podmínkou pro správnou funkčnost aplikace je správné umístění a pojmenování vstupních souborů. Vstupní soubory v sobě obsahují datové podklady pro generování náhodných testovacích dat. Soubory naleznete ve složce **generator/bin/csv**. Složka se vstupními soubory musí být vždy ve stejné složce jako aplikace.

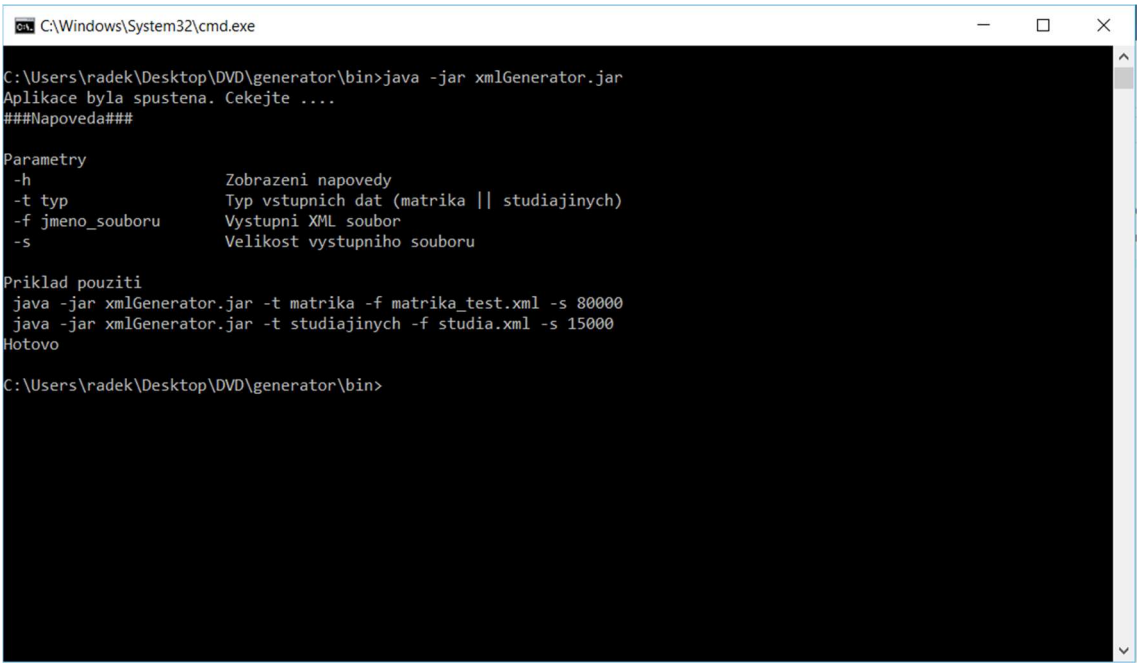
Struktura vstupních dat

- adresa_data.csv
  - okres; obec; castObce; ulice; uliceCislo; psc; stat
- aprobace_data.csv
  - aprobace
- etapa_data.csv
  - platnostOd; obcanstviKvalifikator; obcanstviStat; pobytVcr; jazykVyuky; mistoVyuky; formaStudia; financovani; preruseniStudia; platnostDo
- obory_data.csv
  - obor
- student_data.csv
  - jmeno; prijmeni; rodnePrijmeni; titulPred; titulZa; rodinnyStav; stredniSkola; rokMatZkousky

- a studium_data.csv
    - vsFakulta; studijniProgram; zapisDoStudia; delkaStudia; novePrijaty; navazujiciStudProgram; predchoziVzdelani; odstudovanaCastStudia; vs; vos; czv; zahranici; ubytovaniVKoleji; datum; zpusob; udelenyTitul

2 Práce s aplikací

Základní spuštění aplikace bez parametrů



Obrázek 2-1 xmlGenerator - spuštění

Pokud spustíme aplikaci bez jakéhokoliv parametru, tak aplikace automaticky zobrazí nápovědu s popisem všech dostupných parametrů, jak vidíme na přiloženém obrázku.

2.1 Dostupné parametry

Aplikace má několik vstupních parametrů viz následující tabulka.

Parametr	Popis
-h	Zobrazení nápovědy
-t <typ>	Typ výstupního XML souboru (matrika || studiajinych)
-f <název_souboru>	Umístění výstupního XML souboru
-s <číslo>	Velikost výstupního XML souboru

Tabulka 2-1 Vstupní parametry generátoru

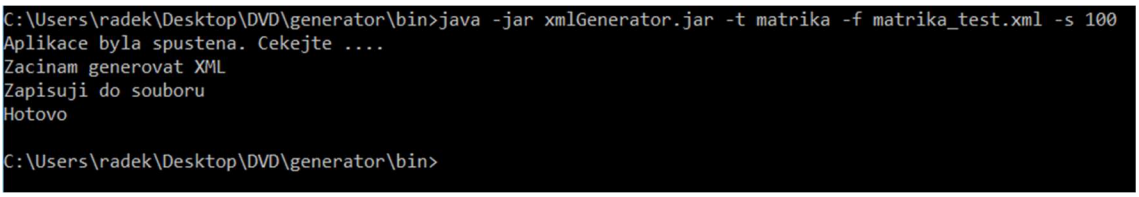
2.1.1 Ukázka generování testovacích dat

Vygenerování testovacích dat podle schématu SIMS_matricniVystupy.xsd. Úkolem je vygenerovat výstupní souboru se 100 studenty.

Ukázka příkazu

```
java -jar xmlGenerator.jar -t matrika -f test.xml -s
100
```

Ilustrační obrázek



Obrázek 2-2 Spuštění generátoru

Pro vygenerování testovacích dat podle schématu SIMS_studiaJinychVSVystupy.xsd stačí pouze změnit v ukázkovém příkladu typ **matrika** na **studiajinych**.