Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Srovnání ukládání dat – XML versus databáze

Petr Hahn

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Srovnání ukládání dat – XML versus databáze" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 28. 3. 2013 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Alexandru Vasilenkovi za pomoc při vypracování této práce.

Srovnání ukládání dat – XML versus databáze

Comparsion XML and common database in data store operation

Souhrn

Diplomová práce se zabývá problematikou ukládání dat. Konkrétně srovnává ukládání dat v XML souborech a v databázi. Jako databáze je v práci zvolena MySQL. Teoretická část mapuje současný stav databáze MySQL a značkovacího jazyka XML. Dále jsou popsány dotazovací jazyky SQL a XQuery a deklarační nástroj Relax NG. V praktické části je srovnána výkonost mezi SQL a XQuery u jednotlivých dotazů, velikost databáze a XML dokumentu, rozdíly zabezpečení, omezení datových typů a možnosti zálohování. Všechny testy jsou prováděny ve stejném hardwarovém a softwarovém prostředí za pomoci skriptů napsaných v jazyce PHP. Závěrečná část shrnuje výsledky a udává výhody a nevýhody implementace databáze nebo XML souborů.

Klíčová slova: Databáze, XML, MySQL, SQL, XQuery, ukládání dat, PHP, Relax NG, Zorba

Summary

This thesis deals with the issue of data storage. It compares saving of data in XML files and database. There is selected MySQL as the database in this thesis. The theoretical part deals with the current state of the database MySQL and marking language XML. It describes the query languages SQL and XQuery and declarative tool RelaxNG. The practical part compares the production rate between SQL and XQuery for individual queries, the database and the XML document size, differences of protection, limitation of data types data types and backup options. All tests are performed in the same hardware and software environment using scripts written in language PHP. The final section summarizes the results and shows the advantages and disadvantages of implementation of a database or XML files.

Keywords: Database, XML, MySQL, SQL, XQuery, data storage, PHP, Relax NG, Zorba

Obsah

1. Úvod

Počítače se za více jak půl století své existence stávají stále důležitější součástí našeho života. Za tuto dobu se výrazně změnily jejich parametry – jedná se zejména o obrovské zrychlení počítačů, nárůst jejich paměti či zmenšení rozměrů počítačů. Spolu s rozšiřováním počítačové gramotnosti rostou požadavky na získávání a ukládání informací. Ať už se jedná o osobní nebo veřejné informace, je nutné je ukládat v určité formě dat na úložné medium. Tato data musí být zpětně prezentována v takové podobě, v jaké byla uložena.

Z počátku o interpretaci dat rozhodoval vývojář aplikace. Tento způsob vedl k nekompatibilitě mezi aplikacemi a k prakticky nulové přenositelnosti dat. S příchodem propojení více počítačů a možností spolupráce mezi nimi, bylo nutné vytvořit způsob ukládání dat, který bude schopen moci zpracovávat více aplikací najednou. Nezávisle na sobě začaly vznikat první databáze a formáty značkovacích jazyků. Jako první standardizovaný značkovací jazyk vnikl jazyk SGML v roce 1960, ze kterého evolučně vznikl formát XML. XML je dnes základním prostředkem, na kterém stojí provoz služeb poskytovaných na internetu. Jeho velkou výhodou je nezávislost na platformě a otevřenost. Tyto vlastnosti umožňují, aby se tento jazyk dobře uplatnil při vývoji a standardizaci protokolů používaných pro webové služby.

První relační databáze, která pohlíží na data jako na tabulky, vznikla v roce 1970 a čtyři roky na to se začala vyvíjet první verze dotazovacího jazyka SQL. V současné době má SQL pevný základ, ale každá firma vyvíjející databáze jazyk rozšiřuje a přidává tak další možnosti. Databáze jsou vyvíjeny s ohledem na zpracování velkého množství dat, dotazů a obsluhování více uživateli v reálném čase.

Rozšiřování XML formátu vedlo k vytvoření samostatného dotazovacího jazyka nad XML daty. Tento jazyk přebral spoustu vlastností z již fungujícího a ustáleného jazyka SQL a převedl ho na fungování nad hierarchickými daty uloženými ve formátu XML. Verze XQuery 3.0, která zatím není ve finální podobě, již umožňuje provádět většinu operací jako SQL. Nabízí se otázka, v jakých případech je vhodné použít formát XML a kdy nasadit databázové zpracování.

2. Cíl práce a metodika

Práce je zaměřena na problematiku rozdílu ukládání dat v souborech využívající vlastností značkovacího jazyka XML a ukládání dat v databázové podobě, konkrétně v databázi MySQL. Práce se zaměřuje na rozdíly ve výkonnosti zpracování mezi jazyky XQuery a SQL a okrajově srovnává zabezpečení, velikost, integritu dat, možnosti zálohování a přenositelnost mezi XML dokumenty a MySQL databází.

Informace použité v diplomové práci byly získány z tištěných a online dostupných zdrojů. Teoretický základ z tištěné literatury byl doplněn o aktuální obsah z internetových zdrojů. Teoretické předpoklady v rešeršní části práce byly doplněny o praktické příklady, které usnadňují pochopení ne vždy zcela lehce představitelných struktur či definic.

V praktické části bylo provedeno výkonové srovnání mezi SQL a XQuery za pomoci skriptů napsaných v PHP jazyce. Skripty byly spouštěny na webovém serveru, který měřil čas potřebný na zpracování. Pro variabilní vykonání skriptů byly vygenerovány databáze o různých velikostech. Ostatní aspekty byly hodnoceny dle teoretických východisek.

V závěru práce je uvedeno zhodnocení stávající situace popsané problematiky s odhadem vývoje do blízké budoucnosti.

3. Teoretická část

3.1. Databáze

Databáze je určitá uspořádaná množina informací (dat) uložená na paměťovém médiu. Součástí databáze jsou i softwarové prostředky, které umožňují manipulaci s uloženými daty a přístup k nim. Data jsou uložena v paměti nezávisle na programech, které je používají. Softwarové prostředky pro správu a přístup k uloženým datům jsou označovány jako Systém řízení báze dat (DBMS) a slouží pro řízení přístupu k datům a správu reprezentace dat na úrovni paměťového média. DB a DBMS tvoří databázový systém.

Databázový systém zahrnuje

- datové prvky – elementární hodnoty
- vztahy mezi prvky dat – složitější datové struktury, které tvoří jednotlivé záznamy
- integritní omezení – podmínky, které musí data uložená v databázi splňovat
- logické schéma – popis záznamů a jejich vazeb z uživatelského hlediska
- fyzické schéma – řeší uložení dat na paměťovém médiu [1]

3.1.1 Základní normy tvorby databází

Při vytváření moderních databází je nutné dodržovat základní normy. Tyto normy mají za úkol:

- omezit redundanci - každý údaj má být v databázi jen jednou
- omezit složitost, tj. dekomponovat složité relace na dvojrozměrné tabulky s atomickými hodnotami polí
- zabránit aktualizačním anomáliím

První normální forma

Tabulka splňuje tuto podmínku, pokud hodnoty, zadané do všech sloupců, jsou atomické, tzn. dál nedělitelné. Jeden sloupec tak nesmí obsahovat více druhů dat. Hodnota sloupce nesmí být relace.

Druhá normální forma

Tuto podmínku splňuje tabulka tehdy, pokud splňuje 1. NF a každý sloupec (kromě primárního klíče) je zcela závislý na primárním klíči. 2. NF se týká pouze tabulek, které obsahují více primárních klíčů.

Třetí normální forma

Tato podmínka platí, pokud platí 2. NF a současně neexistují závislosti neklíčových sloupců tabulky.

Čtvrtá normální forma

Tabulka je ve 4. NF tehdy, pokud je ve 3. NF a popisuje jediný fakt nebo souvislost.

Pátá normální forma

Tabulka je v 5. NF tehdy, pokud je ve 4. NF a není možno do ní přidat sloupec nebo skupinu sloupců, aniž by se rozpadla tato tabulka na několik dílčích tabulek.

Relační databázové systémy používají tabulky a spojují je pomocí relací, z čehož vnikl jejich název. Jako příklad lze uvést databázový systém MySQL, Oracle nebo Microsoft SQL server. Tyto systémy v sobě zahrnují programy jak pro bezpečné skladování dat, tak pro jejich vyhledávání, třídění, mazání, přepisování a vkládání nových dat. Pro připojení k databázovému serveru existují databázoví klienti. Každý klient má určená práva pro práci s databází a má přístup jen k povoleným operacím.

Naproti relačním databázovým systémům existují objektově orientované databáze. Ty jsou z hlediska používání a vývoje mladší a doposud méně využívané. Tyto databáze ukládají samostatné objekty místo tabulek a poté se přistupuje přímo k jednotlivým objektům. Příkladem jsou databáze db4o, CouchDB, ObjectStore nebo MongoDB. [1]

3.2. Relační databáze

Tabulky v relačních databázích představují struktury pro ukládání dat. Každý řádek v tabulce je jednotlivý datový záznam a každý takovýto záznam má definovaný svůj jedinečný klíč, který jednoznačně odkazuje pouze na jeden záznam. Takto definovaný klíč se nazývá primární klíč a je základem pro vytváření relací mezi tabulkami. Sloupce

představují jednotlivá pole záznamu. Každý sloupec má svůj název a typ ukládané informace. Získávání dat probíhá pomocí SQL dotazů, které umožňují přesně definovat, jaká data se mají vybrat. Mezi základní operace patří projekce, selekce a spojení. Projekce vybírá sloupce, operace selekce vybírá řádky (záznamy) a operace spojení provádí spojení dvou tabulek přes zvolenou položku nebo skupinu položek. Dotazy jsou standardizovány, ale každý výrobce rozšiřuje a specifikuje své vlastní pomocné syntaxe. [2]

3.3. Databáze MySQL

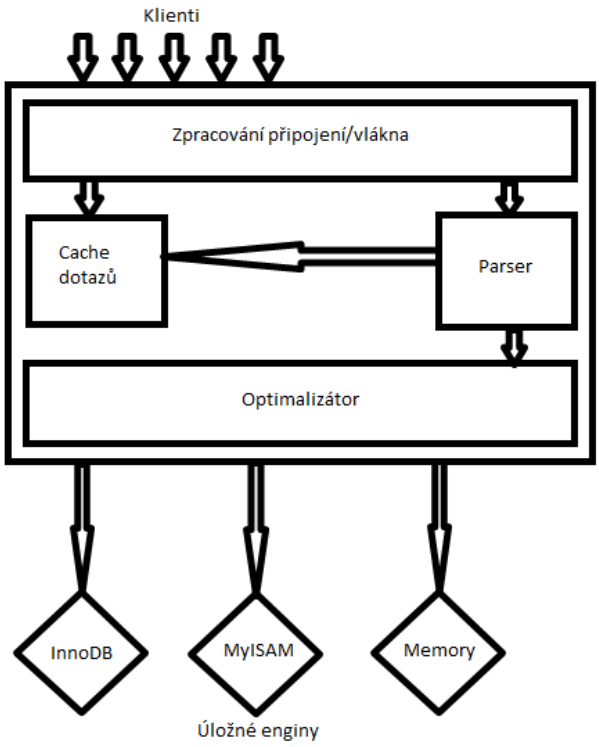
MySQL je relační databázový systém, vytvořený švédskou firmou MySQL AB, nyní vlastněný společností Sun Microsystems. MySQL byl od počátku optimalizován především na rychlost, a to i za cenu některých zjednodušení: má jen jednoduché způsoby zálohování, a až donedávna nepodporoval pohledy, triggery, a uložené procedury. Rozšířenost MySQL je velká díky své bezplatné licenci GPL, podpoře několika jazyků (mezi které lze zařadit PHP, Perl, Python, Ruby, Java, C#), podpoře více platforem (nejvíce využíváno na Linuxu spolu s Apache a PHP)

3.3.1 Vlastnosti MySQL

- Architektura Klient-server – Systém používá databázový server, na který se připojuje množství klientů
- Kompatibilita s SQL – MySQL využívá standardní jazyk pro manipulaci s daty.
- Vnořené dotazy – dotazy lze do sebe libovolně vnořovat
- Pohledy – fiktivní tabulky umožňující pohled na část databáze
- Uložené procedury – programy v SQL uložené v databázi
- Triggery – příkazy pouštěné automaticky při určitých operacích
- Unicode – podpora všech znakových sad
- Transakce – provedení několika operací v jednom bloku, který se buď provede celý, nebo se žádná transakce nezapíše.
- Omezení cizího klíče – zajištění odkazování na existující záznam.
- Nezávislost na platformě – podpora MacOS, Linux, Windows [1]

3.3.2 Architektura MySQL

Vrstva, která je úplně nahoře, obsluhuje většinu potřebných nástrojů klient/server, které jsou založeny na síti. Ve druhé vrstvě se nachází nástroje pro rozbor, analýzu, optimalizaci a pro všechny zabudované funkce. Na této úrovni se nachází veškerá funkcionalita, která se poskytuje prostřednictvím úložných enginů. Třetí vrstva obsahuje úložné enginy. Ty mají na starosti ukládání a získávání všech dat uložených v MySQL. Server komunikuje s úložnými enginy prostřednictvím API úložných enginů. Toto rozhraní skrývá rozdíly mezi jednotlivými úložnými enginy. Úložné enginy nedělají rozbor SQL a nekomunikují mezi sebou (pouze odpovídají na požadavky serveru). [1]



Obrázek 1 - Schéma architektury MySQL

3.3.3 Úložné enginy

Při vytváření tabulky je nutností specifikovat její úložný engine. MySQL podporuje několik typů tabulek a mezi nejpoužívanější se řadí MyISAM a InnoDB (dále pak Memory, Archive, CSV, Falcon, SolidDB a další). Který z těchto typů použít, závisí na několika hlediskách při výběru. Mezi hlediska lze zahrnout podporu transakcí, souběžnost vkládání a čtení, zálohování nebo zotavení po havárii. [2]

3.3.3.1 Úložný engine MyISAM

Engine MyISAM je výchozím enginem MySQL. Je založený na starším a již nedostupném ISAM enginu, který rozšiřuje o nové možnosti. Je jednoduchý, stabilní a rychlý. Tabulky ukládá ve třech souborech, kdy každý soubor se jmenuje podle tabulky a jeho přípona určuje typ souboru:

.frm – ukládá formát tabulky

.MYD (MYData) – samotný datový soubor

.MYI (MYIndex) – soubor s indexy

Tabulky se dělí na statické a dynamické, kdy statický typ se použije v případě, kdy všechny sloupce mají pevně danou velikost, zatímco dynamický typ se využívá v případě datových typů různých velikostí, jako jsou VARCHAR, TEXT nebo BLOB. MyISAM uzamyká celé tabulky. Klienti obdrží sdílené zámky na tabulky, ze kterých potřebují číst. Engine MyISAM nepodporuje omezení cizích klíčů, transakce, ani uzamykání na úrovni řádků.

Limity MyISAM

- Maximální počet řádků v tabulce je limitován počtem 2^{64} (1.844E+19).
- Maximální počet indexů v jedné tabulce je 64.
- Maximální počet sloupců na jeden index je 16.
- Maximální délka klíče je 1000 bajtů.

3.3.3.2 Úložný engine InnoDB

InnoDB je mladší alternativa podporující nové funkce. Mezi tyto funkce patří podpora transakcí, zamykání na úrovni záznamů, omezení cizího klíče a body obnovení databáze. Tento engine je navrhnut pro velká datová úložiště. Engine ukládá tabulky a indexy ve speciálním tabulkovém prostoru, který se může skládat z několika souborů. InnoDB tabulky tak mohou být velmi velké i na operačních systémech, kde je velikost souboru omezena na 2 GB. Od MySQL verze 5.5.5 je nastaven jako výchozí engine. Nevýhodou jsou větší paměťové požadavky na ukládání dat, nepodpora fulltextových indexů a vyšší cena licence při komerčním použití.

Limity InnoDB

- Maximální počet sloupců v tabulce je 1000.
- Tabulka může obsahovat maximálně 64 indexů.
- Maximální délka klíče je 3500 bajtů.
- Minimální velikost tabulkového souboru je 10MB a maximální 64TB.

3.3.3.3 Úložný engine Memory

Tabulky typu MEMORY (synonymum je HEAP) existují pouze v paměti počítače. V paměti jsou uložena jen data, definice tabulky je uložena na disku. To znamená, že při restartu PC je obsah tabulek HEAP ztracen. Díky tomu, že jsou tabulky celé v paměti, je práce s HEAP enginem velmi rychlá. Tabulky typu HEAP mohou být ideálními kandidáty na dočasné tabulky. HEAP tabulku lze vytvořit jako kopii existující trvalé tabulky a pracovat s ní po dobu chodu serveru. [2]

3.4. Datové typy MySQL

Pro každý sloupec v tabulce je definován jeden datový typ. Datových typů v MySQL existuje několik a použití určitého typu ovlivňuje výkon databáze. Použití zbytečně velkého datového typu může mít za následek snížení výkonu, zatímco využití optimálně velikého typu databázi ulehčí a zvýší výkon. Základním pravidlem je použití co nejmenšího a nejjednoduššího možného typu. Datové typy definují ukládaný obsah do databáze, a při špatně zadaných datech sama databáze rozezná chybu a záznam neuloží. [1]

3.4.1 Datový typ – číslo

Existují dva druhy čísel pro uložení v databázi, a to celočíselná a čísla s desetinnou částí. Celočíselné typy se ukládají do celočíselných typů INT a při využití atributu UNSIGNED, který zamezuje ukládání záporných čísel, se horní mez zdvojnásobuje. Reálná čísla se ukládají do typů FLOAT a DOUBLE, kde jde o neexaktní (přibližné číslo), a do typu DECIMAL, kde jde o exaktní (přesné) číslo.

TINYINT(m)	8bitové celé číslo (-128 až 127), hodnota m určuje šířku sloupce ve výsledcích SELECT dotazů
SMALLINT(m)	16bitové celé číslo (-32 768 až 32 767)
MEDIUMINT(m)	24bitové celé číslo (-8 388 698 až 8 388 607)
INT(m)	32bitové celé číslo (-2147483648 až 2147483647)
BIGINT(m)	64bitové celé číslo
FLOAT(m, d)	desetinné číslo, 8místná přesnost, parametry m a d určují počet čísel před a za desetinnou čárkou
DOUBLE(m, d)	desetinné číslo, 16místná přesnost
DECIMAL(p, s)	desetinné číslo s pevnou řádovou čárkou, uložené jako textový řetězec, libovolný počet číslic

Tabulka 1 - Číselné typy dat v MySQL

3.4.2 Datový typ – datum, čas

Datum a čas má v MySQL několik typů pro uložení samostatného data, času nebo roku a dva typy pro uložení celého data s časem a to DATETIME a TIMESTAMP. Rozdíl mezi nimi je v uložení. TIMESTAMP udává, kolik sekund uplynulo od 1. ledna 1970 GMT (4bajty úložního prostoru) a DATETIME ukládá celý datum ve formátu RRRRMMDDHHMMSS (8bajtů úložného prostoru)

DATE	datum ve tvaru RRRR-MM-DD
TIME	časový údaj ve tvaru HH:MM:SS
YEAR	Rok ve tvaru RRRR
DATETIME	kombinace DATE a TIME ve tvaru RRRR-MM-DD HH:MM:SS
TIMESTAMP	speciální časový záznam, který se aktualizuje vždy při změně záznamu

Tabulka 2 - Časové typy dat v MySQL

3.4.3 Datový typ – textový řetězec

Hlavní dva řetězcové typy jsou CHAR a VARCHAR, do nichž se ukládají znakové hodnoty. Rozdíl mezi nimi je v délce řetězce, kdy u typu CHAR je pevně daná a u typu VARCHAR se mění podle počtu uložených znaků. Typy TEXT a BLOB ukládají velké objemy dat.

CHAR(n)	textový řetězec pevné délky (max 255 znaků)
VARCHAR(n)	textový řetězec proměnné délky (max 65 536 znaků)
TINYTEXT	textový řetězec proměnné délky (max 255 znaků)
TEXT	textový řetězec proměnné délky (max 2^{16}-1 znaků)
MEDIUMTEXT	textový řetězec proměnné délky (max 2^{24}-1 znaků)
LONGTEXT	textový řetězec proměnné délky (max 2^{32}-1 znaků)
BIT(n)	binární data, kde n značí počet bitů (do 64 bitů)
TINYBLOB	binární data proměnné délky (max 255 bajtů)
BLOB	binární data proměnné délky (max 2^{16}-1 znaků)
MEDIIUMBLOB	binární data proměnné délky (max 2^{24}-1 znaků)
LONGBLOB	binární data proměnné délky (max 2^{32}-1 znaků)

Tabulka 3 - Textové datové typy v MySQL

3.4.4 Ostatní datové typy

EMUN	výčet textových řetězců
SET	kombinace textových řetězců
GEOMETRY, POINT	geometrický objekt

Tabulka 4 - Ostatní datové typy v MySQL

3.5. Dotazy v MySQL

SQL příkazy se dělí do čtyř základních skupin.

Příkazy pro manipulaci s daty

Jsou to příkazy pro získání dat z databáze a pro jejich úpravy (DML – Data Manipulation Language)

SELECT – vybírá data z databáze, umožňuje výběr podmnožiny a řazení dat.

INSERT – vkládá do databáze nová data.

UPDATE – mění data v databázi (editace).

DELETE – odstraňuje data (záznamy) z databáze.

EXPLAIN PLAN FOR – speciální příkaz, který zobrazuje postup zpracování SQL příkazu.

SHOW – méně častý příkaz, umožňující zobrazit databáze, tabulky nebo jejich definice.

Příkazy pro definici dat

Těmito příkazy se vytvářejí struktury databáze – tabulky, indexy, pohledy a další objekty. Vytvořené struktury lze také upravovat, doplňovat a mazat. (DDL – Data Definition Language).

CREATE – vytváření nových objektů.

ALTER – změny existujících objektů.

DROP – odstraňování objektů.

Příkazy pro řízení dat

Do této skupiny patří příkazy pro nastavování přístupových práv a řízení transakcí. (DCL – Data Control Language)

GRANT – příkaz pro přidělení oprávnění uživateli k určitým objektům.

REVOKE – příkaz pro odnětí práv uživateli.

BEGIN – zahájení transakce.

COMMIT – potvrzení transakce.

ROLLBACK – zrušení transakce, návrat do původního stavu.

Ostatní příkazy

Do této skupiny patří příkazy pro správu databáze. Pomocí nich lze přidávat uživatele, nastavovat systémové parametry (kódování znaků, způsob řazení, formáty data a času apod.). Tato skupina není standardizována a konkrétní syntaxe příkazů je závislá na databázovém systému. [2]

3.5.1 Vytvoření databáze a tabulky

Struktura databáze a tabulky se vytváří za pomoci DDL. Příkaz *CREATE DATABASE nazev_databaze* vytvoří databázi s názvem nazev_databaze. Příkaz *USE* database zvolí tuto databázi pro další používání. Nové tabulky se vytvářejí příkazem *CREATE TABLE nazev_tabulky* a výpisem všech sloupců a jejich datových typů. Pro změnu struktury existuje příkaz *ALTER TABLE* a parametry *ADD* pro přidání sloupce a indexu, *CHANGE* pro změnu názvu sloupce, *DROP* pro smazání sloupce a indexu, *ENGINE* pro změnu typu enginu tabulky. Smazání databáze nebo tabulky provede příkaz *DROP*. [5]

3.5.2 Přidělení práv k databázi

Pro správu práv k databázi je určený DCL. Práva se přidělují příkazem *GRANT právo ON tabulka TO uživatel*. Uživateli mohou být přidělena všechna práva příkazem *ALL* nebo pouze výčet použitelných příkazů. Mezi tato oprávnění patří Select, Insert, Update, Delete, Create, Alter, Index, Drop, Shutdown a Grant option. Pro odebrání práv slouží příkaz *REVOKE*. [5]

3.5.3 Výběr dat

Pro manipulaci s daty se používají příkazy DML. Základní konstrukce dotazu pro výběr dat je následující: *SELECT sloupce FROM tabulka WHERE podmínka ORDER BY sloupec LIMIT číslo*. Příkaz *SELECT* vrací záznamy z tabulky, které vyhovují daným podmínkám. Mohou být seřazeny podle jednoho nebo více sloupců. Počet výsledků ovlivňuje příkaz

LIMIT. Příkazy *WHERE, ORDER BY a LIMIT* jsou nepovinné a bez jejich použití vybere příkaz všechna data seřazená podle uložení v databázi. [5]

3.5.4 Vestavěné funkce

Výstup nemusí obsahovat pouze vyprané sloupce a záznamy. MySQL disponuje mnoha vestavěnými funkcemi, které umožňují zpracovat vrácená data z vybraných sloupců. Mezi nejpoužívanější patří agregační funkce *SUM, AVG, COUNT, MIN, MAX*. Agregační funkce pracují s číselnými daty a vrací vždy jednu hodnotu.

SUM – sečtení hodnot

AVG – průměr z hodnot

COUNT – počet řádků v tabulce

MIN, MAX – vrátí nejmenší (největší) číslo z hodnot

MySQL nabízí ještě celou řadu dalších funkcí. Za zmínku stojí funkce pro práci s řetězci *TRIM, LENGHT, SUBSTRING*, funkce pro zpracování čísel *CEIL, FLOOR, ROUND, RAND* a funkce pro zpracování data a času *CURDATE(), DATE_ADD, YEAR()*.

TRIM – odstranění mezer, formátování

LENGHT – vrací délku řetězce

SUBSTRING – extrakce obsahu řetězce

CEIL, FLOOR, ROUND – zaokrouhlovací funkce

RAND – vrací náhodné číslo

CURDATE() – vrací čas provedení příkazu

DATE_ADD – umožňuje sčítat kalendářní data[5]

3.5.5 Aktualizace dat

Změna dat se provádí pomocí dotazů *INSERT, UPDATE* nebo *DELETE*. Pro vkládání dat slouží příkaz *INSERT INTO tabulka VALUES (hodnoty)*. Příkaz vkládá data do tabulky postupně do polí podle toho, jak jsou seřazena. Příkazem *UPDATE* lze měnit jednotlivé sloupce v tabulce. Syntaxe příkazu je *UPDATE tabulka SET sloupec=hodnota WHERE podmínka*. Podmínka *WHERE* je zde nepovinná a bez použití příkaz změní sloupec u celé tabulky. Pro smazání záznamů slouží příkaz *DELETE*. Syntaxe *DELETE FROM tabulka WHERE podmínka*. Pokud není stanovena podmínka, jsou smazána všechna data z tabulky. [5]

3.5.6 Spojování tabulek

Teoreticky je možné všechna data ukládat do jedné tabulky. Tento přístup však odporuje všem normám a konvencím používaných v SQL jazyce. Tabulka by obsahovala spoustu redundantních dat, zabírala mnoho místa a výkonově zaostávala. V praxi téměř neexistuje databáze, která by obsahovala data uložená pouze v jedné tabulce. V SQL existuje příkaz *JOIN*, který tabulky spojuje efektivně. Samotný příkaz *JOIN* má několik variant. Nejpoužívanější je příkaz *INNER JOIN* neboli vnitřní spojení tabulek, dále *LEFT* a *RIGHT JOIN* spojující tabulky zleva nebo zprava a dvě méně časté varianty spojení *CROSS JOIN*, který vrátí kartézský součin tabulek, a *NATURAL JOIN*. [5]

INNER JOIN zobrazuje pouze data, která si v obou tabulkách přesně odpovídají. Vnitřní spojení se dá přepsat v MySQL pomocí podmínky, která spojí tabulky podle cizích klíčů. [5]

3.6. Jazyk XML

Značkovací jazyk XML (eXtensible Markup Language) je určený pro značkování textů. Značky slouží k vyznačení určitých částí textu na jednotky. Takto označené části pak mohou mít nadefinovaný svůj vlastní význam. Názvy značky jsou zcela volitelné a tím pádem dávají autorovi volnost k vytvoření jakéhokoli značkování dokumentu. XML data lze rozdělit na dokumentově orientovaná, kdy není přesně dána struktura dat, a na datově orientovaná, kdy je známa přesná struktura dat dokumentu. Značky do sebe zapadají hierarchickým způsobem a XML lze tedy dobře využít pro data, která jsou hierarchická a vyžadují jisté uspořádání. Díky tomu lze XML použít pro databázově orientované ukládání dat.

Formát XML je definován konsorciem W3C jako formát pro přenos obecných dokumentů a dat. Jazyk XML vychází ze staršího standardu SGML, z kterého taktéž vycházel i formát dokumentů HTML. Na rozdíl od HTML rozlišuje XML malá a velká písmena. XML implicitně využívá kódování Unicode, což umožňuje ukládat data v jakémkoli písmu, respektive používat všechny znaky této znakové tabulky. Lze použít i jiná kódování jako windows-1250 nebo iso-8859-2, přičemž jiné než implicitní kódování musí být v dokumentu určeno.

Jazyk se používá převážně k výměně elektronických dat. Za elektronická data se v tomto případě považují zejména html soubory, dále pak vektorová grafika, textové dokumenty, strukturovaná data pro samostatný software, RSS (Really Simple Syndication) data a další. [3]

Požadavky na formát XML

- Formát XML musí být použitelný v rámci internetu
- Formát XML by měl podporovat širokou škálu aplikací
- Formát XML musí být kompatibilní s formátem SGML
- Musí být snadné vytvářet programy, které manipulují s dokumenty v XML
- Množství variant XML by mělo být minimální
- XML dokumenty by měly být čitelné a pochopitelné i pro člověka

3.6.1 Struktura XML

XML dokument se skládá z posloupnosti jednotlivých prvků. Každý prvek je definován svojí značkou a má svůj vlastní význam. Značky si volí autor dokumentu sám, ale musí dodržovat jisté konvence jazyka. Značky mají tvar obecných závorek < >. Otevírací značka je ve tvaru <nazev_znacky> a uzavírací značka má tvar </nazev_znacky>. Jednoduchý příklad pro popsání města.

```
<mesto>Karlovy Vary</mesto>
```

Každý element má svoji otevírací značku <mesto> a zavírací značku </mesto>. Vše, co se nachází mezi těmito značkami, tvoří samotný obsah elementu. V tomto případě jde o textový řetězec „Karlovy Vary“.

Pokud by element neobsahoval žádný text, lze využít dva způsoby zápisu. Celý zápis elementu bez zapsání obsahu <mesto></mesto> nebo také zjednodušený zápis <mesto />. Oba zápisy znamenají to samé a výsledkem je element město, který neobsahuje žádný text. Takto vytvořený element se nazývá prázdný element.

Dokumenty se ukládají s příponou .xml pro určení typu dokumentu. Název dokumentu je zcela libovolný, ale standardně by měl vyjadřovat obsah dokumentu. [3]

3.6.2 Značky XML

Značky (také tagy) elementů jsou case sensitive neboli rozlišují mezi velkými a malými písmeny. XML má striktní syntaxi a nedodržení vede ke špatně formulovanému dokumentu. Každá značka musí mít svoji ukončovací značku. Tyto dvě značky musí mít stejný název, přičemž záleží na velikosti písmen. To znamená, že značka <mesto> musí být ukončena značkou </mesto> a ne značkou </Mesto>, která má jiný význam.

Názvy XML elementů mohou obsahovat alfanumerické znaky obsažené v tabulce znaků, která je nastavena pro daný dokument a dále pak znaky:

. tečka

- pomlčka

_ podtržítko

Ostatní znaky jako mezery, uvozovky, tabulátory, apostrof, středník nebo symbol procenta jsou striktně zakázány. Speciálním případem je symbol dvojtečka, která značí jmenné prostory. Jména značek musí začínat pouze písmeny nebo podtržítkem. Pokud značka začíná otazníkem, jde o procesní instrukci a ne o vyznačení elementu. [3]

3.6.3 Hierarchie XML

Každý element může obsahovat jiný element. Takto lze elementy vnořovat do jakékoli úrovně. Každý vnořený element se nazývá potomek a jemu nadřazený element se nazývá předek. Pokud má jeden rodič více potomků. XML dokument má právě jeden kořenový element (root element), který nemá žádného rodiče. Každý další element musí mít právě jednoho rodiče a libovolné množství potomků. V následujícím dokumentu je definován kořenový prvek adresa, který má dva potomky město a psc. Prvek mesto a psc jsou sourozenci a oba mají stejného rodiče prvek adresa.

```
<adresa>
     <mesto>Karlovy Vary</mesto>
     <psc>222 22</psc>
</adresa>
```

Křížení jednotlivých prvků je zakázáno. To znamená, že nelze značky prohazovat. Následující zápis je neplatný:

```
<adresa>
     <mesto><psc>Karlovy Vary</mesto>222 22</psc>
</adresa>
```

3.6.4 Atributy

Atributy představují doplňkovou informaci k elementům. Element může obsahovat více různých atributů. Atributy se zapisují do počátečního tagu elementu ve tvaru jmeno_atributu = ”hodnota_atributu”. Na hodnotu atributu se dají použít buď složené, nebo jednoduché uvozovky. Požadavky na názvy atributů se shodují s požadavky na názvy značek elementů. Atribut je vyhodnocen stejně jako element a zápis z předchozího příkladu lze zapsat pomocí atributů takto:

```
<adresa mesto=”Karlovy Vary” psc=”222 22”>
</adresa>
```

Výsledek dokumentu pak podává stejnou informaci jako výsledek z předešlého příkladu. Neexistuje pravidlo na to, zda použít atribut nebo vnořit další element. Atributy jsou využívány převážně v html, kde je to již standardem. V datových XML dokumentech se používají zřídka a převážně pro doplnění informace o elementu jako například id, jednotka, typ apod. Příklad použití atributu v datovém xml dokumentu:

```
<adresa id=”34”></adresa>
<vaha jednotka=”kg”></vaha>
<pocet typ=”number”></pocet>
```

Některé problémy s používáním atributů

- Atributy nemohou obsahovat složené hodnoty (potomci ano).
- Atributy nejsou jednoduše rozšiřitelné pro budoucí změny.
- Atributy nemohou popisovat struktury (potomci ano).
- Atributy jsou těžkopádnější při editaci či při čtení programovým kódem. [3]

3.6.5 Entitní reference

Entita je prvek zastupující část textu v dokumentu. V obsahu elementu jsou zakázány některé znaky, které se dají nahradit právě entitami. Je možné definovat také vlastní entity pro často využívaný textový řetězec. Vlastní entity se deklarují v definiční struktuře dokumentu a deklarace vypadá následovně:

```
<!ENTITY nazev_entity "textkteryzastupuje">
```

Vyvolání entity v obsahu elementu se provede znakem ampersandu a ukončí středníkem: &nazeventity;

```
<element>&nazev_entity;</element>
```

Předdefinované entity pro XML dokumenty zastupující znaky, které jsou v elementu zakázané:

&lt;	<	symbol menší než
&gt;	>	symbol větší než
&amp;	&	ampersand
&apos;	'	apostrof, jednoduché uvozovky
&quot;	"	dvojité uvozovky

Parser dokumentu při obdržení znaku & zjistí, že jde o entitu a tu pak nahradí textovým řetězcem nebo znakem, který nebylo možné do obsahu vepsat samostatně. [3]

3.6.6 Komentáře v XML

Komentáře se využívají pro zanechání poznámky nebo dočasného skrytí XML kódu. I přes to, že XML je samo o sobě velmi dobře čitelné a strukturované, je zde možnost komentáře využívat. Komentář je parsery ignorován a nemá pro dokument žádnou informační hodnotu. Komentáře se zapisují mezi znaky <!-- a --> a nesmí být součástí ostatního značkování (např. nelze „zakomentovat" atributy). [3]

```
<!-- toto je komentář -->
```

3.6.7 Sekce CDATA

Pokud je potřeba do dokumentu vložit delší text, který obsahuje větší množství zakázaných xml znaků jako < > &, vloží se do sekce CDATA. Tento text pak může obsahovat jakékoli znaky, je správně interpretován a nenarušuje strukturu dokumentu. Obecně se sekce zapisuje jako <![CDATA[text]]>. Tato konstrukce zabraňuje použití znaků]]> v obsaženém textu, protože je tím sekce CDATA ukončena. [3]

3.6.8 Instrukce pro zpracování

Dokumenty XML mohou být zpracovány různými programy, které vyžadují dodatečné informace. K tomu slouží v XML zpracovací instrukce. Tyto instrukce parser ignoruje a

přebírá je samotný program. Instrukce má ve značce na prvním a posledním místě symbol otazníku a její syntaxe je tedy:

```
<?instrukce data?>
```

Pomocí instrukcí se hned na první řádce deklaruje XML dokument a připojují se tak k dokumentu styly dokumentu. [3]

3.6.9 Struktura dokumentu

Každý XML dokument musí začínat prologem. Prolog je instrukce pro zpracování s názvem xml. Zpravidla mívá tři atributy a to version, encoding a standalone.

- Atribut version – určuje verzi použitého XML
- Atribut encoding – určuje použité kódování dokumentu, kdy implicitně je nastaveno kódování UTF-8 Unicode.
- Atribut standalone – udává, zda je dokument složen z jednoho nebo více souborů. Má dvě přípustné hodnoty yes a no

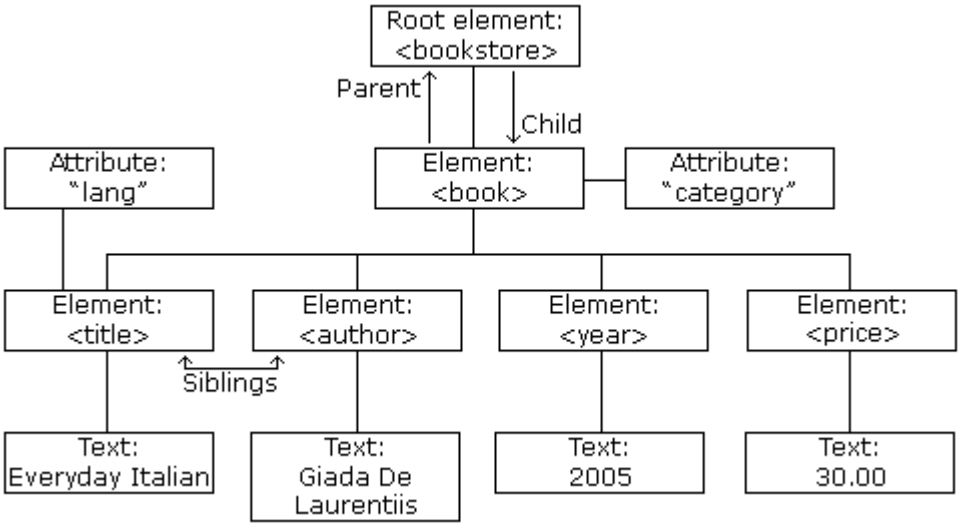
Dále mohou následovat xml instrukce pro připojení stylů k dokumentu.

```
<?xml-stylesheet href="file" type="text/css" ?>
```

Další součástí prologu může být schéma dokumentu. Schéma definuje, jaké elementy mohou být v dokumentu použity, jaké lze využít atributy a jaké mají být hodnoty elementů nebo atributů. Podle tohoto schématu se také určuje validita XML dokumentu. Tato deklarace typu dokumentu může být uvedena lokálně přímo v XML dokumentu nebo může odkazovat na externí soubor se schématem.

```
<!DOCTYPE adresa SYSTEM "adresa.dtd">
```

Bezprostředně po prologu následuje kořenový element dokumentu, který se vyskytuje v dokumentu pouze jednou. Kořenový element může obsahovat libovolný počet hierarchicky řazených elementů. Na konci musí být kořenový element uzavřen a tím je dokončen celý XML dokument. [3]



Obrázek 2 - Schéma struktury XML dokumentu

3.6.10 Definice typu dokumentu

Za základní definici dokumentu lze považovat DTD (Document Type Definition - definice typu dokumentu). Popisuje, jak mohou být značky navzájem uspořádány a vnořovány. Vymezuje atributy pro každou značku a typ těchto atributů. DTD je poměrně starý a málo expresivní jazyk. Jeho další nevýhoda je, že DTD samotný není XML soubor.

3.6.10.1 Deklarace typu elementu

```
<!ELEMENT odesílatel (#PCDATA)>
```

Element může obsahovat znaková data.

```
<!ELEMENT odesílatel(jméno | text)>
```

Element obsahuje další dva dané elementy a nic jiného

```
<!ELEMENT odesílatel EMPTY>
```

Element je prázdný (neobsahuje text ani další elementy nebo atributy)

3.6.10.2 Regulární výrazy v DTD

, sekvence složek

| možnost výběru z alternativ

? nepovinnost složky

+ možnost opakování složky

* možnost libovolného opakování složky, včetně žádného

3.6.10.3 Deklarace atributů elementu

```
<!ATTLIST odesílatel email CDATA #REQUIRED>
```

Element odesílatel obsahuje atribut email, který je tvořen řetězcem dat a který je povinně vyžadován. U každého atributu lze stanovit, zda je povinný #REQUIRED nebo nepovinný #IMPLIED. Třetí možností je pevně stanovený neměnný atribut #FIXED, který již dále v dokumentu nelze měnit.

3.6.10.4 Deklarace entity

Entity jsou v XML využívány jako konstanty, které nahrazují text v dokumentu. Většinou se používají pro speciální znaky. [3]

```
<!ENTITY apos "&#39;">
```

3.7. Relax NG

Pro popis struktury dokumentu lze využít silnější jazyky jako například XML Schema, Relax NG, Schematron nebo DSD, které jsou již samy o sobě ve formátu XML. Tato práce bude využívat jazyka Relax NG, který je jedním ze standardů ISO. Relax NG využívá jednoduššího zápisu než XML Schema a oproti DTD podporuje datové typy.

Relax NG je samostatný XML soubor. Kořenový element dokumentu má vždy název element a obsahuje alespoň jeden atribut se jménem. Dalším atributem je použitý jmenný prostor Relax NG a knihovna typů dat.

```
<?xml version="1.0" encoding="UTF-8"?>
<element name="nazev"
xmlns="http://relaxng.org/ns/structure/1.0"
 datatypeLibrary="http://www.w3.org/2001/XMLSchema-
datatypes">
</element>
```

Celé Relax NG schéma se skládá ze vzorů. Mezi dva základní vzory patří <element name="nazev"> a <attribute name="nazev">. Atribut name u vzorů je povinný a určuje název definovaného elementu nebo atributu. Vzory lze hierarchicky strukturovat. Pro určení počtu opakování jsou vymezeny vzory <optional> <oneOrMore> <zeroOrMore>. Optional omezuje počet opakování na žádné nebo jednou, oneOrMore na jednou a více a zeroOrMore na jakékoli opakování.

```
<zeroOrMore>
    <element name="nazev">
      <text/>
    </element>
</zeroOrMore>
```

Relax NG sám o sobě podporu datových typů nenabízí. Nabízí však mechanismus, jak jej používat s libovolnou externí knihovnou datových typů. Díky tomu je možné i v Relax NG kontrolovat, zda element nebo atribut obsahuje hodnotu vyhovující podmínkám nějakého datového typu.

```
<element name="nazev">
    <data type="double"/>
</element>
```

Všechny datové typy jsou přebírány z externí knihovny a záleží tak na použité knihovně, jaké typy mohou být používány. Základní knihovna XMLSchema disponuje všemi běžně používanými datovými typy a pro běžné účely je dostačující. Výpis datových typů je dostupný na adrese http://www.w3.org/2001/XMLSchema-datatypes.

U každého datového typu je možné určit parametry, které jej dále upřesňují. Při použití datových typů z XMLSchema je tak možné přímo v Relax NG nastavit většinu integritních omezení. Pro nastavení parametrů se používá element param s atributem name.

```
<param name="minLength">5</param>
<param name="maxLength">10</param>
```

Hodnota minLength a maxLength vymezuje délku textového řetězce.

```
<param name="minInclusive">100</param>
<param name="maxExclusive">1000</param>
```

Hodnota minExclusive a maxExclusive vymezuje interval čísla

Použití integritního omezení enumeration je zakázáno, protože Relax NG nabízí vlastní prostředky pro definování seznamu hodnot přípustných pro nějaký atribut nebo element. Pro určení přípustné hodnoty se používá vzor value.

```
<choice>
    <value>hodnota1</value>
    <value>hodnota2</value>
    <value>hodnota3</value>
</choice>
```

Relax NG podporuje kompaktní syntaxi. Ta umožňuje zapsat schéma v textové syntaxi, která je mnohem úspornější než ta založená na XML. Problémem této syntaxe bývá nepodpora u validačních nástrojů. Tyto nástroje zpravidla podporují jen plnou XML verzi Relax NG. [5] [7] [8]

3.8. XPath

XPath (XML Path Language) je počítačový jazyk, pomocí kterého lze adresovat části XML dokumentu. Pomocí tohoto jazyka lze z XML dokumentu vybírat jednotlivé elementy a pracovat s jejich hodnotami a atributy. Jazyk XPath je standardem vydaným organizací W3C. Nejdůležitější konstrukcí jazyka XPath je cesta k XML uzlu. Cesta obsahuje několik kroků, přičemž každý z nich sestává z identifikátoru osy, testu uzlu a podmínky.

3.8.1 Oddělování kroků

Lomítko (/)

Stejně jako u adresářové struktury, jednotlivé kroky v XPath cestě lze oddělovat lomítky. Pokud je lomítko na začátku výrazu, znamená to, že výraz není vztažen k aktuálnímu elementu, ale počítá se od kořene dokumentu.

auto/barva - Všechny elementy barva, které jsou přímými potomky elementu auto.

/auta - Kořenový element dokumentu.

Dvě lomítka (//)

Dvě lomítka bezprostředně za sebou slouží k překonání víceúrovňové struktury. Pokud jsou dvě lomítka na začátku výrazu, opět se jedná o absolutní cestu od kořene dokumentu, takže lze snadno vybrat libovolný element odkudkoli.

auto//barva - Funguje, i když elementy barva nejsou přímými potomky elementu auto.

//auto - Vrací všechny elementy daného jména v jakémkoli kontextu.

//* - Všechny elementy XML dokumentu.

3.8.2 Identifikátory osy

Testu uzlu může předcházet takzvaný identifikátor osy. Ten určuje směr procházení XML dokumentu, tedy říká, odkud se uzly k vyhodnocení budou vytahovat. Pokud identifikátor osy není uvedený, použije se child::

- child:: - Přímí potomci aktuálního uzlu.
- descendand:: - Všichni potomci aktuálního uzlu.
- descendand-or-self:: - Aktuální uzel a všichni potomci.
- self:: - Aktuální uzel.
- ancestor-or-self:: - Aktuální uzel a všichni jeho předci.
- ancestor:: - Všichni předci aktuálního uzlu.
- parent:: - Rodič aktuálního uzlu.
- following:: - Uzly, které se v XML dokumentu nacházejí za aktuálním uzlem.
- preceding:: - Uzly, které se v XML dokumentu nacházejí před aktuálním uzlem.
- attribute:: - Atributy aktuálního uzlu.
- namespace:: - Deklarované jmenné prostory.

3.8.3 Uzel určený názvem

Je to jeden z nejzákladnějších testů uzlu. Vybere všechny XML elementy s daným názvem. Tento test se zapisuje jednoduchým zápisem testovaného názvu elementu.

auto - Všechny podřízené elementy s daným názvem.

auta - Kořenový element dokumentu.

3.8.4 Uzel určený typem

Tento test uzlu vybírá pouze určitý typ. Test se zapisuje typem uzlu následovaným prázdnými kulatými závorkami. Existují tyto možnosti: comment(), text(), processing-instruction() a node().

//text() - Vybere všechny textové uzly v dokumentu.

3.8.5 Podmínky

Do hranatých závorek se zapisují další podmínky, které zužují výsledky předchozího vyhodnocení. Pokud je místo podmínky zapsáno pouze číslo, vybere podmínka pouze uzel umístěný na pozici s daným pořadovým číslem. Protože některé konstrukce se používají velice často, existují pro ně zkrácené tvary.

Hvězdička (*) - Hvězdička použitá jako test uzlu vybírá všechny poskytnuté elementy.

Tečka (.) - Tečka nahrazuje odkaz na sebe sama.

Dvě tečky (..) - Dvě tečky umožní pohyb o jednu úroveň výše.

Zavináč (@) - Zavináč nahrazuje identifikátor osy attribute::.

3.8.6 Operátory

V XPath se používají operátory obvyklé ve všech programovacích jazycích. Využívají se především v podmínkách. Pořadí vyhodnocení je možné upravovat závorkami.

| - Sjednocuje výsledky více výrazů

+, -, *, div, mod - Matematické operátory

=, !=, <, <=, >, >= - Relační operátory

and, or - Logické operátory

3.8.7 Funkce

XPath obsahuje celou řadu užitečných funkcí, které lze využít hlavně při tvorbě podmínek.

position(), last(), count()

Pořadové číslo aktuálního XML uzlu, pořadové číslo posledního uzlu a počet uzlů v daném kontextu.

name(), namespace-uri(), local-name()

Úplné kvalifikované jméno, název jmenného prostoru a lokální název aktuálního XML uzlu.

string(), number(), boolean()

Převede libovolný objekt na řetězcovou, číselnou a logickou hodnotu.

floor(), ceiling(), round()

Zaokrouhlení čísla dolů, nahoru a na nejbližší celé číslo.

sum()

Objekty v parametru jsou zkonvertovány na čísla a funkce vrátí jejich součet.

not(), true(), false()

Negace logického výrazu, logická jednička a logická nula.

starts-with(), contains(), substring-before(), substring-after(), substring(), string-length(), normalize-space(), translate()

Funkce pracující s řetězci. [9]

3.9. XQuery

Jazyk Xpath je standardem pro dotazování XML dat a je zaměřen na adresaci a výběr určitých částí XML dokumentu. Oproti tomu jazyk XQuery je více zaměřený na potřeby SŘBD. Využívá jazyka Xpath a rozšiřuje ho. XQuery podporuje cesty, množinová porovnávání, aritmetické výrazy a rozšiřuje konstrukci o FLWOR výrazy, třídění, podmínění výrazy nebo konstruktory.

Cesty se vyjadřují podobně jako v Xpath, je však nutné doplnit vstupní funkci

```
Doc("knihy.xml")/knihovna/knihy
```

3.9.1 Výrazy FLWOR

FLWOR výrazy jsou základní konstrukcí jazyka XQuery. FLWOR je zkratka pro for-let-where-order by-return. Tato syntaxe je převzatá z SQL jazyka, kde je využíváno konstrukce SELECT-FROm-WHERE-ORDER BY.

- Klauzule for sváže jednu nebo více proměnných s požadovanými výrazy
- Klauzule let váže k proměnné celý výsledek výrazu, ke kterému se proměnná vztahuje
- Klauzule where funguje jako filtr na jednotlivé posloupnosti a ponechává jen ty vyhovující daným podmínkám
- Klauzule order by řadí vzniklé posloupnosti podle zadaného kritéria
- Klauzule return vrací výsledek FLWOR výrazu

FLWOR výraz musí mít alespoň jednu for nebo let klauzuli a musí končit return klauzulí. Where a order by jsou nepovinné.

```
for $k in doc(knihy.xml)//kniha
where $k/@rok_vydani = 2003
return $k/nazev
```

Pro spojení více dokumentů se využívá více příkazů for.

```
For $a in doc(knihy.xml)//kniha,
      $b in doc(knihy2.xml)//pujcovna/kniha
Where $a/nazev = $b/nazev
Return {$a/nazev/text()}
{$b/cena/text()}
```

3.9.2 Kvantifikátory

XQuery podporují jak existenční tak obecný kvantifikátor. Tyto kvantifikátory lze využít ke zjednodušení složitějších dotazů. Kvantifikátory mají syntaxi some a every následovanou slovem satisfied

```
For $kniha in doc(knihy.xml)//kniha
Where some $autor in $kniha//autor satisfies $autor/jmeni –
"jan"
Return $kniha
```

Tato syntaxe vyhledá všechny knihy, u kterých se nějaký z autorů jmenuje Jan.

3.9.3 Vstupní funkce

Vstupní funkce definují data, ve kterých se později dotazuje. Všechny tyto funkce berou jako parametr řetězec lokace URI dokumentu.

Doc() – vrací dokument identifikovaný svým URI

Collection() – vrací kolekci dokumentů

Root() – vrací kořen aktuálního dokumentu [9][11]

3.10. Praktické využití XML

OOXML (Office open XML)

Specifikuje souborový formát pro ukládání dokumentů kancelářských balíků. Lze ukládat text, tabulky nebo prezentace. Tento formát navrhla společnost Microsoft a poprvé byl použit v aplikaci Microsoft Office 2007. Výhoda oproti předchozím verzím je otevřenost a menší velikost.

RSS (Resource Description Framework)

RSS je XML formát, určený pro čtení novinek na webových stránkách. Technologie RSS umožňuje uživatelům Internetu přihlásit se k odběru novinek z webu, který nabízí RSS zdroj (RSS feed, též RSS kanál). Tento zdroj se většinou vyskytuje na stránkách, kde se obsah mění a přidává velmi často

RSS formát poskytuje obsah celého článku, příp. jeho část, odkaz na původní článek a také jiná metadata. Tyto informace poté zpracovává RSS čtečka.

MATHML

Pomocí MathML lze vytvářet různé matematické konstrukce, rovnice, funkce apod. Specifikace verze 1.01 byla uveřejněna v červenci 1999 a v únoru 2001 se objevila verze 2.0. V listopadu 2003 bylo zveřejněno druhé vydání MathML verze 2.0 a byla označena jako konečná verze matematické pracovní skupiny konsorcia W3C. MathML bylo navrženo s několika základními cíli:

- možnost konvertovat existující dokumenty do MathML
- možnost připojit MathML k HTML a interpretovat ho prohlížeči
- možnost získání vzorce ze zdrojového zápisu, který může být určeným programem interpretován a vyhodnocen

SVG

Jazyk SVG popisuje dvourozměrnou vektorovou grafiku. V budoucnu by se měl stát základním otevřeným formátem pro internetovou grafiku. Může obsahovat i rastrovou grafiku avšak při vektorovém vykreslování zachovává stejnou kvalitu při zvětšování obrazu. Grafika SVG neobsahuje obrazová data pixel po pixelu, ale seznam svých součástí – grafických objektů, pomocí kterých lze obrázek vykreslit. SVG je ideální pro jednoduchou grafiku. Mezi výhody patří

- velikost výsledného souboru
- nezávislost na platformě
- snadná přenositelnost
- je čitelný jak pro počítač, tak pro člověka
- obsahuje-li text, je možné jej vyhledávat

DOCBOOK

DocBook umožňuje psaní vlastní dokumentace. Lze jej využít na psaní knih, článků, prezentací nebo na tvorbu webů. Formát DocBook lze snadno konvertovat do jiných, běžně využívaných formátů jako HTML, PDF nebo RFT. DocBook je jazyk XML. Ve své současné verzi (5.0), jazyk DocBooku je formálně definován pomocí Relax NG XML schématem s integrovanými pravidly Schematron. [4]

4. Praktická část

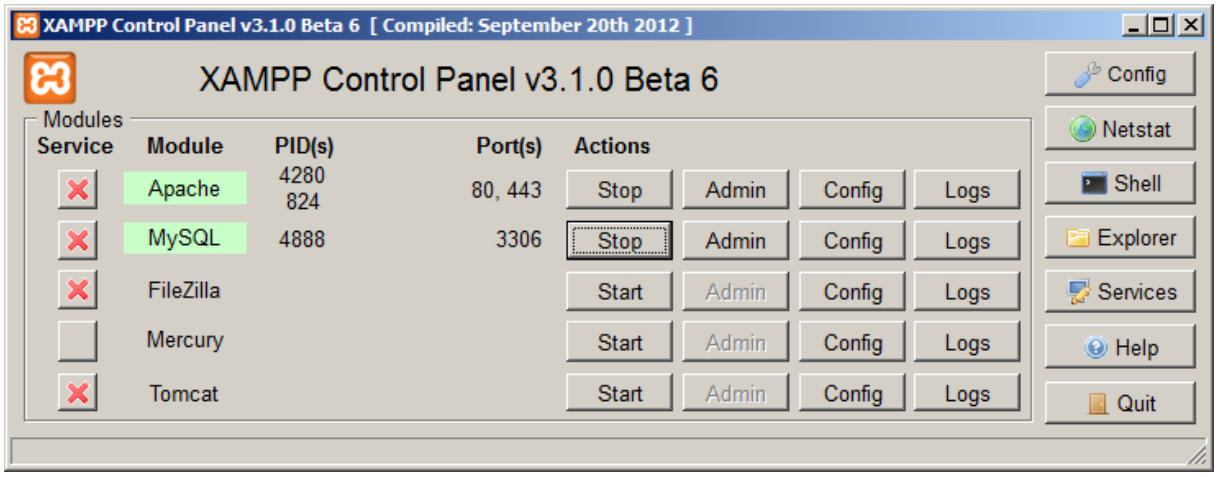
4.1. Softwarové prostředí

K porovnání různých aspektů mezi databázovým systémem a XML dokumenty je zvoleno stejné prostředí se stejnými podmínkami. Použitý operační systém je Windows 7 x64 SP1. Zvolený jazyk pro vykonávání jednotlivých příkazů je skriptovací jazyk PHP, který je hojně využíván právě s MySQL databází a podporuje i zpracování XML souborů. K jednoduché instalaci těchto nástrojů jsou zdarma dostupné komplexní instalační balíky WampServer, XAMPP a EasyPHP, všechny obsahují totožné verze Apache, PHP a MySQL.

V této práci je použit XAMPP for Windows 1.8.1. XAMPP obsahuje následující verze modulů:

- Apache 2.4.3
- MySQL 5.5.27
- PHP 5.4.7
- phpMyAdmin 3.5.2.2
- XAMPP Control Panel 3.1.0

Celá aplikace lze jednoduše ovládat pomocí Control Panelu [12]



Obrázek 3 - Kontrolní panel XAMPP

Pro tuto práci jsou stěžejní moduly Apache – webový server, MySQL – relační databáze a PHP – skriptovací jazyk. XAMPP dále obsahuje FTP server FileZilla, poštovního klienta Mercury a Tomcat jako aplikační server. [13]

Jazyk PHP podporuje operace nad XML soubory, ale nepodporuje XQuery. XQuery lze v PHP využívat za pomoci knihovny Zorba (aktuálně Zorba 2.8.0 – Hermes). Knihovna Zorba není v základním balíčku XAMPP a je nutné ji dodatečně nainstalovat. [14][15]

4.2. Hardwarové prostředí

Samotný XAMPP má minimální nároky na hardwarové požadavky:

- 64 MB RAM
- 350 MB místa na disku
- Windows NT, 2000, XP, VISTA, 7

Testování v této práci probíhalo na průměrně výkonném PC s těmito parametry:

- Procesor – i5 3350
- Paměť RAM – 8GB 1600MHz
- HDD – 3TB 7200ot, SATA III

Výkonově je nejslabším článkem pevný disk, který zvládá zápis/čtení dat rychlostí okolo 100MB/s. Pokud by databáze nebo XML soubory zabíraly místo v řádech gigabajtů, trvalo by načíst celou databázi nebo soubor desítky sekund. Vhodnější je proto SSD disk s několikanásobnou rychlostí čtení a menší přístupovou dobou.

Z paměti RAM samotný běh systému spotřebuje kolem 2GB a na ostatní aplikace zbývá 6GB. V současné době je to na běh ostatních programů u osobního PC více než dostatečná velikost. Ovšem jak vyplyne z výsledků práce, na parsování XML dokumentu je vyžadováno velké množství paměti a proto jsou u serverových PC nároky na paměť velmi vysoké.

Čtyř jádrový procesor z řady Ivy Bridge běží na frekvenci 3,1 GHz. Zpracování každého PHP skriptu vytěžuje však jen jedno jádro současně. Výkonově se procesor nevyrovná běžným serverovým procesorům. Na zpracování testovacích databází však dostačuje.

4.3. Aspekty k porovnání

4.3.1 Rychlost

Nejdůležitější vlastností každého databázového systému je rychlost. Jednotlivé příkazy musí být vykonány co nejrychleji a pokud možno s co nejmenšími nároky na hardwarové zdroje. V současné době jsou procesory tak výkonné, že rozdíly ve zpracování elementárních operací na úrovni relačních databází jsou minimální.

Při zpracovávání XML souborů záleží na programovacím jazyku. Jazyky parsují XML soubory jinak a to se odráží na rychlosti zpracování. XML dokument je rozložen na jednotlivé Nody, se kterými se dále pracuje jako se záznamy.

Rychlost zpracování lze srovnávat při každé operaci, která je nad daty prováděna. V PHP je rychlost měřitelná příkazem microtime(). Tento příkaz vrací aktuální čas v mikrosekundách, který uběhl od 1. ledna 1970 0:00:00 GMT. Rozdíl mezi počátečním časem při začátku vykonávání operace a časem na konci vykonání operace je pak výslednou dobou zpracování příkazu.

```
$time_start = microtime(true);
Vykonávaný příkaz
$time_end = microtime(true);
$time = $time_end - $time_start;
echo "$time sekund\n";
```

4.3.2 Velikost

Velikost je druhá nejdůležitější vlastnost při ukládání informací. Malé databáze zabírají desítky MegaBytů a zálohování, přenositelnost nebo údržba jsou záležitostí několika sekund. Zatímco velké databáze zabírající místo v řádech TeraBytů jsou náročné jak časově tak finančně na udržování záloh a archivaci. Velikost DB je ovlivněna použitým databázovým softwarem, dále úložným enginem v samotném softwaru a v neposlední řadě také správnou volbou typu ukládaných dat.

U XML souborů je velikost ovlivňována každým písmenem (bajtem) napsaným v dokumentu. To znamená, že každá značka ovlivňuje velikost souboru. Značky se

s každým záznamem přidávají znovu, a pokud by značka měla dlouhé jméno, soubor by zbytečně nabýval na velikosti. Nejkratší možnou značkou je jedno písmeno (nebo znak), to je však v rozporu se snadnou čitelností souboru. U extrémně velkých XML souborů je možné nahradit značky jedním znakem a tím ušetřit na velikosti souboru.

4.3.3 Integritní omezení dat

Databázové systémy mají vlastní datové typy. MySQL podporuje poměrně velkou škálu datových typů (uvedeno v teoretické části). Datové typy se deklarují při vytváření tabulek a systém řízení báze dat si pak sám hlídá omezení a do tabulky nedovolí zapsat jiný datový typ.

XML soubory nemají interní datová omezení a pro integritu dat jsou využívána definiční schémata, která datové soubory vyhodnocují a určují validitu dat.

4.3.4 Bezpečnost

Každý DB systém má své zabezpečení dat. MySQL používá tabulku se jmény, hesly a právy, podle které přiděluje práva připojeným uživatelům k příkazům vykonávaným nad databází.

XML dokumenty nemají řešený přístup k datům. Pokud má uživatel přístup k souboru, má pak přístup ke všem datům v něm obsaženým. Přístup ke XML datům lze řešit pomocí programovacích jazyků, ve kterých je soubor nejdříve zpracován a poté jsou data prezentována uživateli, který na ně má právo.

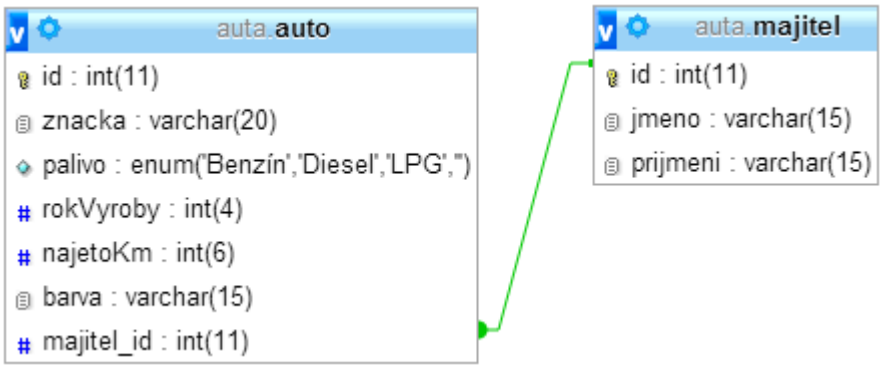
4.3.5 Zálohování a přenositelnost dat

Data v DB systému jsou uložená podle použitého enginu. Přenositelnost jednotlivých souborů je závislá na verzi databáze a enginu. Přenášení dat v MySQL proto probíhá pomocí exportu databáze a následného importu na jiné úložiště.

XML dokumenty jsou navrhovány pro přenos a jejich přenositelnost je také jejich základní vlastností. Jednotlivé soubory lze otevřít kdekoli a prakticky pomocí jakéhokoli editoru.

4.4. Vytvoření tabulek databáze

Pro spouštění dotazů jak nad databází, tak nad XML soubory, je nutné, aby databáze obsahovala nějaká data. K srovnání dotazů, velikostí, rychlostí a ostatních aspektů byla vygenerována data dle velmi jednoduchého schématu dvou tabulek, které mezi sebou mají vazbu 1:N. Data jsou vygenerována ve třech verzích. První verzí je malá databáze o velikosti 1000 záznamů. Druhou verzí je střední databáze o velikosti 100 000 záznamů a poslední verzí je velká databáze o velikosti 1 000 000 záznamů. Tyto objemy dat jsou použity pouze u jedné tabulky. Druhá tabulka je pomocná ke srovnání dotazů nad více tabulkami a obsahuje konstantní počet 100 záznamů. V databázi je použit nejnovější engine InnoDB. Tabulky jsou vytvořeny pomocí XAMPP rozšíření phpMyAdmin, které dovoluje pohodlně zadávat SQL příkazy.



Obrázek 4 - Diagram databázových tabulek

```
CREATE TABLE `auto` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `znacka` varchar(20) COLLATE utf8_czech_ci NOT NULL,
  `palivo` enum('Benzín','Diesel','LPG','') NOT NULL,
  `rokVyroby` int(4) NOT NULL,
  `najetoKm` int(6) NOT NULL,
  `barva` varchar(15) COLLATE utf8_czech_ci NOT NULL,
  `majitel_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `majitel_id` (`majitel_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci
AUTO_INCREMENT=1;
```

Tabulka „auto“ obsahuje hlavní datové záznamy. Záznamy ukládají informace o vozech a tabulka má celkem sedm sloupců. Sloupec id je jednoznačný číselný identifikátor vozu. Je nastaven na primární klíč a dále na hodnotu auto_increment, která zaručuje při každém přidání záznamu zvýšení hodnoty o jedna a tím i unikátnost hodnoty v tomto sloupci. Sloupec znacka zaznamenává značku automobilu. Je nastaven na variabilní textovou hodnotu o velikosti 20 znaků. Sloupec palivo určuje typ paliva vozu. Sloupec je typu enum se třemi hodnotami Benzín, Diesel a LPG. Sloupce rokVyroby a najetoKm ukládají číselné hodnoty typu int. Sloupec barva ukládá informace o barvě automobilu v textové podobě o maximální velikosti 15 znaků. Poslední sloupec majitel_id je klíčový sloupec, který odkazuje na záznam v tabulce majitel. Všechny sloupce mají nastavenou vlastnost NOT NULL a kódování pro textová pole je utf8_czech_ci.

```
CREATE TABLE `majitel` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `jmeno` varchar(15) COLLATE utf8_czech_ci NOT NULL,
  `prijmeni` varchar(15) COLLATE utf8_czech_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_czech_ci
AUTO_INCREMENT=1;
```

Tabulka „majitel“ je pomocná tabulka, která má pouze tři sloupce. Sloupec id jako jednoznačný identifikátor a sloupce jmeno a prijmeni. Tyto dva sloupce mají oba textovou hodnotu o maximální velikosti 15 znaků.

```
ALTER TABLE `auto`
  ADD CONSTRAINT `auto_ibfk_1` FOREIGN KEY (`majitel_id`)
REFERENCES `majitel` (`id`) ON DELETE RESTRICT ON UPDATE
RESTRICT;
```

Poslední příkaz při vytváření tabulek upravuje tabulku auto. Sloupci majitel_id přiděluje hodnotu cizího klíče s odkazem na záznam v tabulce majitel. Příkaz ON DELETE RESTRICT zakazuje smazání majitele, pokud na něj odkazuje nějaký automobil.

4.5. Vygenerování dat

K vygenerování dat je použit jazyk php. Jako první je nutné se připojit k databázi. Pro přístup k databázi je použito rozšíření MySQLi. Umožňuje přístup k novým funkcím. MySQLi podporuje jak objektově orientovaný, tak i procedurální styl programování V této práci je použit procedurální styl.

```
$mysqli = @mysqli_connect("localhost", "root", "", "auta");
if (mysqli_connect_errno()){
    die('Připojení se nezdařilo: ' .
mysqli_connect_error());
    exit();
}
mysqli_set_charset($mysqli, "utf8");
```

Příkazem se připojí k databázovému serveru na adrese localhost, jako uživatel root, s prázdným heslem a vybrání databáze auta. Pokud připojení bude neúspěšné, vypíšou se případné chyby. Funkce mysqli_set_charset() nastavuje kódování použité v databázi.

Po připojení k databázi je spuštěn příkaz na generování dat. Jelikož se data generují v několika variantách, tak bude vytvořeno více tabulek o různých velikostech. Nejdříve je nutné vyplnit tabulku majitel, protože na ní odkazuje tabulka auto a musí tak odkazovat na již existující záznamy. Tabulka majitel obsahuje 100 vygenerovaných záznamů.

```
$nameArray = array("Jiří","Jan","Petr "…);
$surnameArray = array("Novák","Svoboda","Novotný"…);
for($i=0 ; $i<100; $i++)
     {
     $name = $nameArray[rand(0, 12)];
     $surname = $surnameArray[rand(0, 11)];
     $query = "INSERT INTO majitel VALUES (NULL,
'".$name."','".$surname."')";
     mysqli_query($mysqli, $query);
}
```

Majitel má náhodně generováno jméno a příjmení z pole hodnot stanovených v proměnných nameArray a surnameArray. Cyklus for proběhne stokrát a přiřadí

proměnným name a surname náhodnou hodnotu z polí hodnot. Proměnná query je řetězec obsahující SQL příkaz, který je dále předán proceduře mysqli_query a zpracováván databází. Vygenerování takto malé databáze trvá v rozmezí 1-2 sekund.

Další generované tabulky budou tabulky automobilů. Generování většího objemu dat je výpočetně náročné a tak základní nastavení pro php max_execution_time = 30 sekund nebude stačit a je nutné zvýšit maximální výpočetní čas.

```
$signArray = array("BMW","Citroen","Dacia"…);
$fuelArray = array("Diesel","LPG","Benzín");
$colorArray =
array("červená","žlutá","zelená","modrá","bílá","černá");
for($i=0 ; $i<1000; $i++)
{
     $sign = $signArray[rand(0, 17)];
     $fuel = $fuelArray[rand(0, 2)];
     $year = rand(1980,2012);
     $distance = rand(0,500000);
     $color = $colorArray[rand(0, 5)];
     $owner = rand(1,100);
     $query = "INSERT INTO auto VALUES (NULL, '".$sign."'
,'".$fuel."' ,'".$year."' ,'".$distance."'
,'".$color."','".$owner."')";
     mysqli_query($mysqli, $query);
}
```

Vygenerování databáze o velikosti 1000 záznamů trvalo 15 sekund. Střední databáze o velikosti 100 000 záznamů bude generována 100krát déle tzn. 25 minut a velká databáze o velikosti 1 000 000 záznamů pak 250 minut.

Po naplnění databáze daty jsou vytvořeny XML soubory, které obsahují totožná data jako ta uložená v databázi. Pro každou tabulku je vhodnější vytvořit samostatný XML soubor než ukládat všechny tabulky do jednoho souboru. Kořenový element souboru nese název podle názvu databáze, což je v tomto případě auta. Každý záznam je pak element s názvem tabulky, v tomto případě auto. Element má jeden atribut id, který odpovídá

primárnímu klíči v tabulce a obsahuje šest dalších elementů odpovídajících sloupcům tabulky.

```
$file = fopen("auto.xml", "a");
$header = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
$header .= "<auta>\n";
fwrite($file, $header);
$query = "SELECT * FROM auto";
$result = mysqli_query($mysqli, $query);
while($result_array = mysqli_fetch_assoc($result))
{
     $xml = "\t<auto id=\"".$result_array['id']."\">";
     $xml .= "\n";
     foreach($result_array as $key => $value)
     {
           if($key != "id")
           {
                $xml .= "\t\t<$key>";
                $xml .= "$value";
                $xml .= "</$key>\n";
           }
     }
     $xml.="\t</auto>\n";
     fwrite($file, $xml);
}
fwrite($file, "</auta>");
fclose($file);
```

Skript vytvoří soubor auto.xml a uloží do něj hlavičku potřebnou pro identifikaci XML souboru. Dále jsou vybrány všechny záznamy z tabulky auto a po jednom přepisovány do specifikované XML formy. Každý záznam je pak připsán na konec souboru. Předposlední řádek uzavírá kořenový element xml souboru. Výsledkem je pak kopie tabulky převedená do zformátovaného xml souboru viz obrázek.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<auta>
    <auto id="1">
        <znacka>Renault</znacka>
        <palivo>Benzín</palivo>
        <rokVyroby>1982</rokVyroby>
        <najetoKm>72723</najetoKm>
        <barva>modrá</barva>
        <majitel_id>55</majitel_id>
    </auto>
    <auto id="2">
        <znacka>Opel</znacka>
        <palivo>LPG</palivo>
```

Obrázek 5 - XML soubor

Druhá tabulka je převedena stejným skriptem se změněnými proměnnými, dle názvu tabulky.

4.6. Srovnání datových operací

4.6.1 Vytvoření tabulky/xml souboru

Vytvoření tabulky představuje dotaz CREATE TABLE následovaný pojmenováním jednotlivých sloupců a určení jejich datových typů. Tímto je definována celá struktura tabulky a tabulka je připravena na vkládání dat.

```
$query = "CREATE TABLE table (id INT NOT NULL)";
mysqli_query($mysqli, $query);
```

Vytvoření xml souboru představuje založení souboru běžně s příponou .xml a deklarování verze xml a kódování v prvním řádku dokumentu. Dále by mělo následovat schéma dokumentu, které definuje logickou strukturu dat v dokumentu. A nakonec kořenový element. Tímto je xml dokument připraven na vkládání dat.

```
$file = fopen("auta.xml", "w");
$header = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
$header .= "<auta>\n";
$header .= "</auta>\n";
fwrite($file, $header);
fclose($file);
```

Velké množství tabulek databázi značně zatěžuje, zatímco xml soubory jsou na sobě nezávislé a jejich počet výkon nijak neovlivňuje. I přes to, že k vytvoření základního xml souboru je potřeba vykonat více příkazů, je vytváření mnohem rychlejší. Posílání příkazů do databáze a zpracování samotnou databází spotřebuje více prostředků na vykonání.

4.6.2 Vkládání dat

Data jsou do tabulek vkládána příkazem INSERT INTO. Tento příkaz vloží buď jeden, nebo více záznamů najednou do tabulky. U xml souborů se data vkládají za pomoci XQuery příkazu insert element/attribute. Příkazy pro vkládání dat musí respektovat stanovené datové typy. Pokud by byl v příkazu jiný datový typ, než jaký je určen strukturou, bude vložena buď základní hodnota, hodnota null nebo se takový příkaz nevykoná vůbec.

```
$query = "INSERT INTO auto VALUES (NULL, '1' ,'Citroen'
,'2000' ,'5000' ,'černá','5'),(NULL,…)";
mysqli_query($mysqli, $query);
```

První hodnota v příkazu je NULL, protože databáze sama určuje hodnotu sloupce id a při každém záznamu je o jednu zvýšena. Další hodnoty jsou proměnné uchovávající textové řetězce nebo čísla. Poslední hodnota je číselná a odkazuje na id záznamu v tabulce majitel.

```
$xquery = new XQueryProcessor();
$query = <<<'XQ'
declare variable $doc as document-node() external;
insert node element auto {
  attribute id { 1 },
  element znacka { "Citroen" },
  element palivo { "Benzín" },
  element rokVyroby { 2000 },
  element najetoKm { 5000 },
  element barva { "černá" },
  element majitel_id { 5 }
} as last into $doc/auta;
```

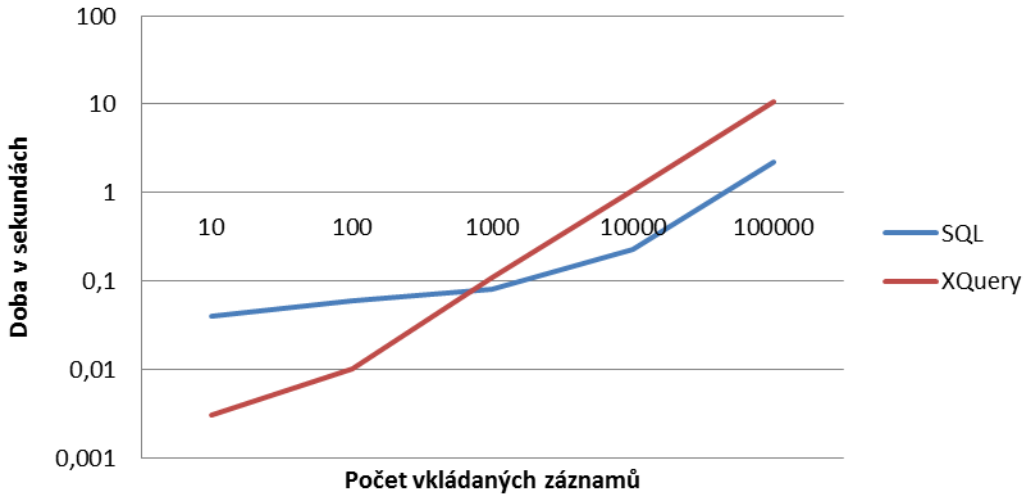
```
insert node element auta { …}
$doc
XQ;

$xquery->importQuery($query);
$doc = simplexml_load_file("auta.xml");
$xquery->setVariable("doc", $doc);
$newXML = $xquery->execute();
```

Vložení nového záznamu do XML souboru je trochu složitější. Jako první je potřeba vytvořit novou instanci třídy XQueryProcessor, která je definována v rozšíření Zorba. Druhým krokem je vytvoření dotazu. Dotaz vytváří element auto, který obsahuje dalších šest elementů a jeden atribut. Nový element auto je vždy přidán nakonec. V dotazu je definována proměnná doc, která odkazuje na soubor s xml dokumentem. Po vykonání dotazu je celý soubor uložen.

Při vkládání nových dat mají databáze integrované funkce pro autoinkrementaci pole id. U XML souborů je nutné nejprve tuto hodnotu získat, poté zvýšit o jedna a nakonec vložit nový záznam. Následující tabulka udává dobu potřebnou ke zpracování vkládání dat:

Počet vkládaných záznamů	SQL (v sekundách)	XQuery (v sekundách)
10	0,04	0,003
100	0,06	0,01
1000	0,08	0,11
10000	0,23	1,05
100000	2,18	10,61

Tabulka 5 - Doba vkládání záznamů v dávce



Obrázek 6 - Graf doby vkládání záznamů v dávce

Z grafu je viditelný rozdíl ve zpracování pří stoupajícím objemu zpracovávaných dat. XQuery předběhne zpracování SQL okolo 1000 zpracovávaných záznamů. Takovýto výsledek byl dosažen, pokud byly všechny příkazy sepsány najednou a předány databázi nebo Zorba executoru. Problém nastává pří vkládání dat po jednom záznamu. V tomto případě má databázové zpracování značnou výhodu v integrované autoinkrementaci. Při vkládání záznamů do XML souboru je nutné soubor nejprve načíst, poté zjistit id posledního záznamu a zvýšit o jedna, následně vytvořit nový záznam a nakonec celý soubor uložit.

```
$xquery = new XQueryProcessor();

$query = <<<'XQ'
declare variable $doc as document-node() external;
variable $lastid := $doc/auta/auto[last()]/@id/number();
$lastid := $lastid + 1;

insert node element auto {
  attribute id {$lastid},
  element znacka { "Citroen" },
  element palivo { "Benzín" },
  element rokVyroby { 2000 },
```

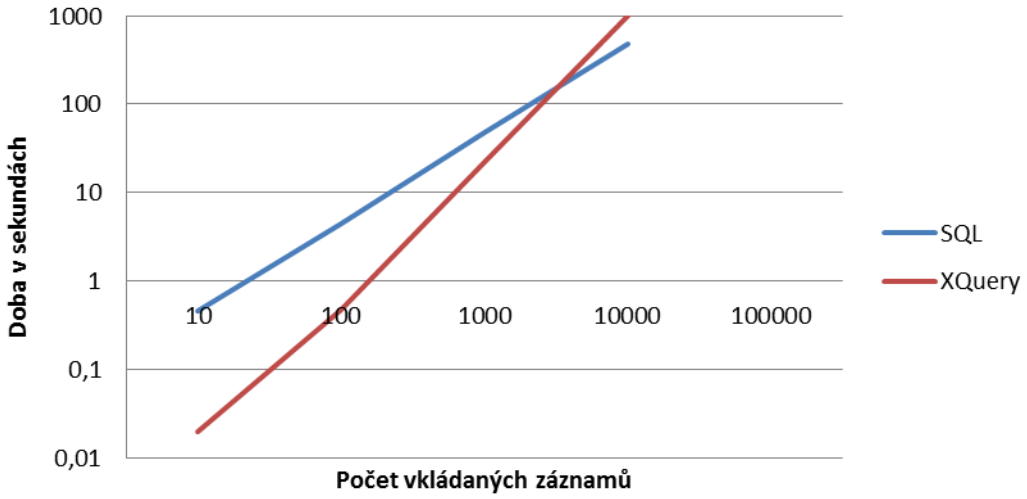
```
  element najetoKm { 5000 },
  element barva { "černá" },
  element majitel_id { 5 }
} as last into $doc/auta;
$doc
XQ;

$xquery->importQuery($query);
$doc = simplexml_load_file("auta.xml");
$xquery->setVariable("doc", $doc);
$newXML = $xquery->execute();
```

V případě SQL je syntaxe jednoduchá a do databáze jsou odesílány po jednom příkazy INSERT INTO.

```
$query = "INSERT INTO auto VALUES (NULL,
'Citroen','Benzín',2000,5000,'černá','5');";
mysqli_query($mysqli, $query);
```

Počet vkládaných záznamů	SQL (v sekundách)	XQuery (v sekundách)
10	0,45	0,02
100	4,43	0,47
1000	47,45	22,51
10000	480,56	1000+
100000	1000+	1000+

Tabulka 6 - Doba vkládání záznamů po jednom



Obrázek 7 - Graf doby vkládání záznamů po jednom

Z tabulky lze vyčíst, že SQL INSERT INTO příkazy vykonávané po jednom rostou téměř konstantně, zatímco vkládání dat do XML souborů má exponenciální růst časové náročnosti. To je dáno neustálým načítáním a parsováním dokumentu.

4.6.3 Změna dat

V MySQL se změna dat provádí příkazem UPDATE SET. Takto lze upravit všechny záznamy v tabulce nebo po přidání selektoru WHERE pouze záznamy splňující danou podmínku. XQuery syntaxe používá replace value of node with. Změny lze provádět pouze na vybraném nodu. Hromadná změna pak lze provést pomocí for příkazu.

Změna jednoho záznamu v SQL:

```
$query = "UPDATE auta SET najetoKm = najetoKm + 1000 WHERE
id=1";
```

Změna jednoho zánamu v XQuery:

```
$query = <<<'XQ'
declare variable $doc as document-node() external;
replace value of node
$doc/auta/auto[@id=1]/najetoKm/number()
with $doc/auta/auto[@id=1]/najetoKm/number() + 1000;
$doc XQ;
```

SQL syntaxe je jednodušší a čitelnější oproti složitější XQuery syntaxi. SQL vykoná příkaz UPDATE nad tabulkou auta a nastaví (SET) položku najetoKm na hodnotu o 1000 větší ve všech záznamech, které splňují kritérium (WHERE) hodnota sloupce id je rovna 1. XQuery vykoná příkaz replace value nad elementem auta/auto[@id=1]/najetoKm. Tento XPath výraz filtruje elementy a odkazuje na všechny elementy auto s atributem id nastaveným na hodnotu 1, v tomto případě na jediný element. Dále nastaví číselnou hodnotu number() na stejnou hodnotu zvýšenou o 1000.

V případě upravování více hodnot naráz je nutné v XQuery použít více volání příkazu replace a odkazovat na každý element samostatně. Je vhodné využít proměnnou, která uloží cestu k aktuálně upravovanému elementu.

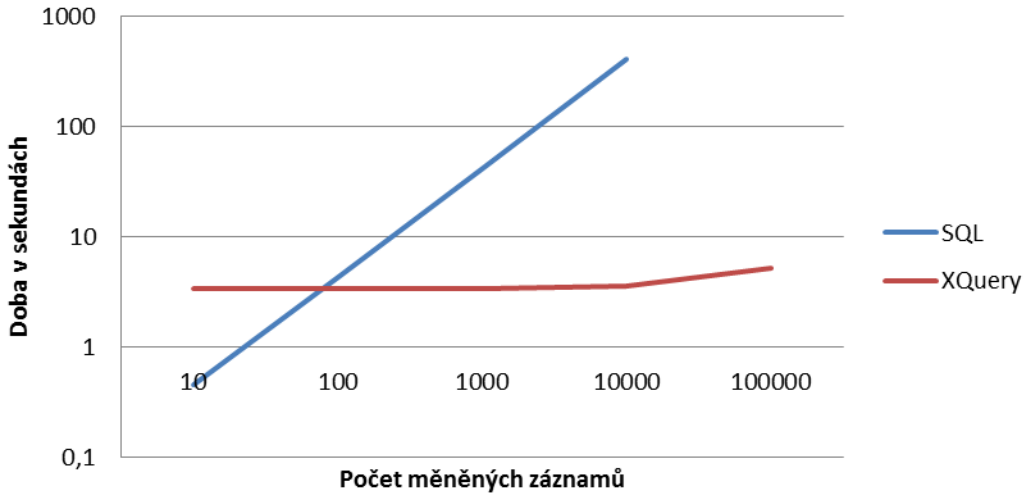
```
$query = <<<'XQ'
declare variable $doc as document-node() external;
declare variable $auto := $doc/auta/auto[@id=1];

replace value of node $auto/najetoKm/text()
  with "10000";
replace value of node $auto/barva/text()
  with "modrá";
replace value of node $auto/majitel_id/text()
  with "23";

$doc
XQ;
```

V případě SQL je původní dotaz rozšířen o další pole oddělené čárkou.

```
$query = "UPDATE auto SET najetoKm = 10000, barva = 'modrá',
majitel_id = 23 WHERE id=1";
```

Počet měněných záznamů	SQL (v sekundách)	XQuery (v sekundách)
10	0,45	3,34
100	4,3	3,35
1000	41,12	3,37
10000	405,8	3,53
100000	1000+	5,21

Tabulka 7 - Doba změny záznamů



Obrázek 8 - Graf doby změny záznamů

XML Dokument má stále stejnou velikost a jeho načítání a ukládání trvá stejnou dobu při každé změně záznamu. Databáze přijímá a updatuje data taktéž konstantní rychlostí, ale doba na zpracování je delší než v případě XQuery. Při aktualizování dat v XML souborech je doba updatu přímo úměrná velikosti souboru. Čím více soubor obsahuje dat, tím déle trvá jeho načítání a ukládání. Aktualizace v SQL se do databáze odesílá po jednom příkaze, proto je čas na zpracování delší. Změna nodů v XML dokumentu je hromadná operace a je proto rychlejší.

4.6.4 Mazání dat

Oba jazyky používají k mazání příkaz delete. Rozdíl je v zápisu a aplikaci příkazu. SQL příkaz DELETE FROM smaže záznam nebo záznamy dle podmínek výběru. XQuery příkaz maže samotné nody ve struktuře. Syntaxe XQuery je tak flexibilnější a dovoluje smazat jakýkoli element v souboru. XQuery delete je volán na XPath výraz a maže vše, na co výraz referuje.

SQL dotaz smazání všech záznamů

```
$query = "DELETE FROM auto";
```

XQuery dotaz na smazání všech záznamů

```
$query = <<<'XQ'
declare variable $doc as document-node() external;
delete node $doc/auto;
$doc
XQ;
```

Dotaz maže všechny elementy auto v souboru, ale zanechá v něm kořenový element auta.

V případě smazání posledního záznamu by příkaz vypadal následovně:

```
$query = <<<'XQ'
declare variable $doc as document-node() external;
delete node $doc/auto[last()];
$doc
XQ;
```

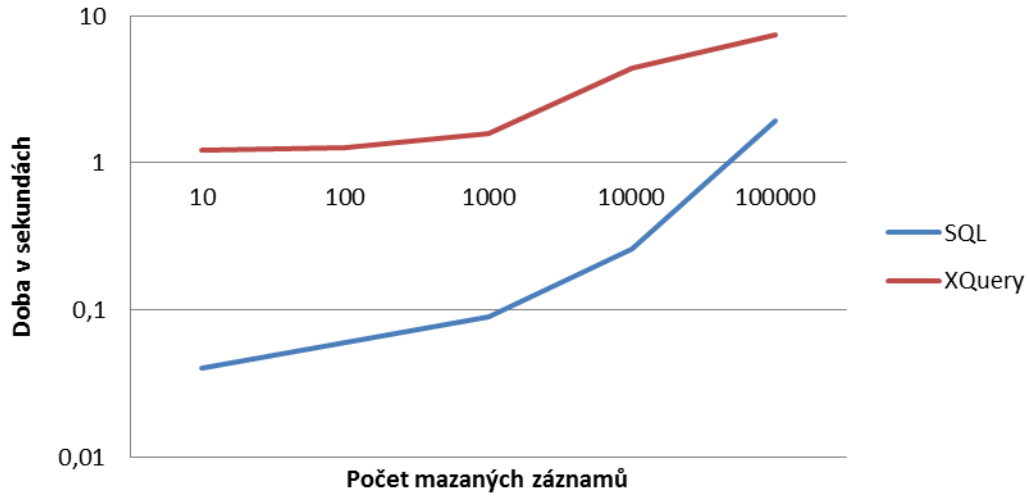
V SQL neexistuje pomocná funkce last() a příkaz na smazání posledního záznamu musí obsahovat pomocnou podmínku:

```
$query = "DELETE FROM auta ORDER BY id DESC LIMIT 1";
```

V databázi o jednom milionu záznamů vypadá tabulka mazání záznamů následovně:

Počet mazaných záznamů	SQL (v sekundách)	XQuery (v sekundách)
10	0,04	1,22
100	0,06	1,26
1000	0,09	1,57
10000	0,26	4,42
100000	1,95	7,39

Tabulka 8 - Doba zpracování smazání záznamů



Obrázek 9 - Graf doby zpracování smazání záznamů

4.6.5 Jednoduchý výběrový dotaz

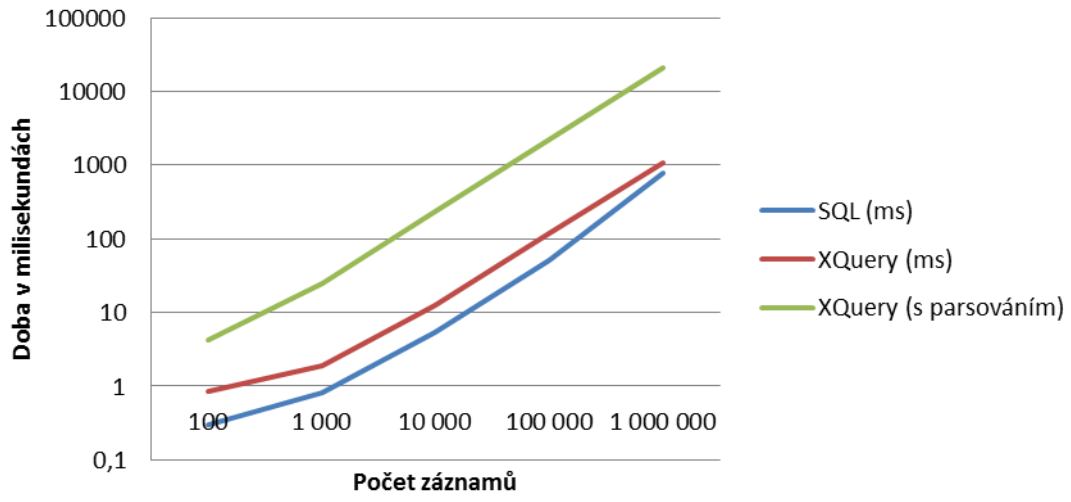
Selekce dat je prováděna v jazyce SQL příkazem SELECT. Příkaz dále může být rozvinut o podmínky, které data musí splňovat, řazení, podle vybraného sloupce nebo sloupců, seskupování a limitování počtu výsledků. V XQuery lze data selektovat obdobným způsobem za použití kompletního FLOWR výrazu. Syntaxe je oproti SQL složitější, zato mnohem flexibilnější a výsledek dotazu může mít svůj vlastní formát dat. V příkazu je nutné použít jednu klauzuli for nebo let a příkaz vždy zakončit formulí return, která vrátí naformátovaná výsledná data.

Dotaz, který vypíše všechny automobily značky Mercedes, vyrobené po roce 2008, a který vypisuje pouze ID automobilu má SQL syntaxi:

```
SELECT id FROM auto WHERE znacka="Mercedes" AND rokVyroby > 2008
```

V případě XQuery je syntaxe následující:

```
variable $doc := doc("D:/auta1000.xml");
for $x in $doc/auta/auto
where $x/znacka = "Mercedes" and $x/rokVyroby > 2008
return $x/@id/string()
```



Obrázek 10 - Graf doby zpracování jednoduchého výběrového dotazu

Počet záznamů	SQL (ms)	XQuery (ms)	XQuery (s parsováním)
100	0,30	0,86	4,24
1 000	0,80	1,87	24,85
10 000	5,35	12,51	230,01
100 000	50,28	118,89	2254,14
1 000 000	789,31	1091,818	20605,364

Tabulka 9 - Doba zpracování jednoduchého výběrového dotazu

Jednoduchý výběrový dotaz je silně závislý na velikosti XML souboru, pokud je započítávána doba na parsování souboru. Bez započítání doby na parsování je dotaz zpracován téměř stejně rychle jako v případě SQL. Do dalších výběrových dotazů již nebude započítáváno parsování.

4.6.6 Souhrnný a řadící dotaz

Oba dva jazyky podporují seřazování pomocí příkazu order by. V nové verzi XQuery 3.0 byla přidána podpora příkazů group by a count. Díky těmto příkazům je nyní snadnější vytvářet souhrnné dotazy a sčítat počet položek výsledku. SQL provádí seskupování příkazem group by. Dotaz, který seskupí položky podle vybraných sloupců a seřadí podle sloupců, vypadá v SQL následovně:

```
SELECT znacka, barva
FROM auto
GROUP BY znacka, barva
ORDER BY znacka, barva
```

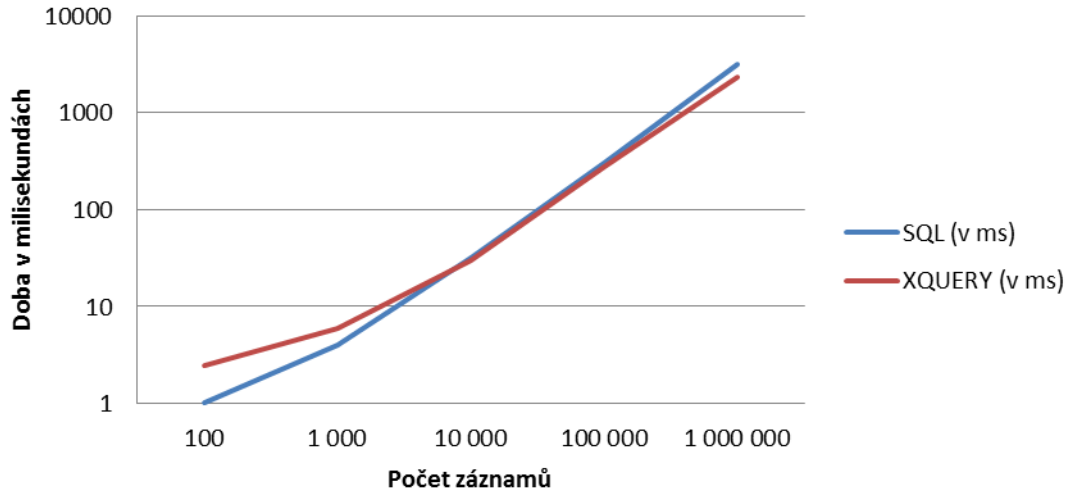
XQuery zápis má tvar:

```
$query = <<<'XQ'
declare variable $doc as document-node() external;
for $x in $doc/auta/auto
group by $z := $x/znacka, $y := $x/barva
order by $z, $y
return <skupina>
<znacka>{$z}</znacka>
<barva>{$y}</barva>
</skupina> XQ;
```

V případě XQuery je nutné přidat další proměnné pro možnost seskupování. Proměnné lze deklarovat přímo ve funkci group by a není nutné použít klauzuli let. Výstup je možné formátovat jakýmkoli způsobem (zde jako element skupina obsahující elementy znacka a barva).

Počet záznamů	SQL (v ms)	XQUERY (v ms)
100	1,03	2,49
1 000	4,06	5,93
10 000	31,28	29,71
100 000	298,68	275,44
1 000 000	3145,78	2345,23

Tabulka 10 - Doba zpracování souhrnného dotazu



Obrázek 11 - Graf doby zpracování souhrnného dotazu

V případě souhrnného dotazu je z počátku rychlejší SQL ale při větších objemech začne výkonově značně ztrácet nad XQuery. Přelom nastává okolo 10 000 záznamů, kdy je doba zpracování velmi podobná. Při počtu jeden milion záznamů je XQuery téměř o vteřinu rychlejší než SQL.

4.6.7 Spojovací dotaz

Tabulky jsou v databázi spojovány relacemi, které umožňují vytvářet dotazy nad více než jednou tabulkou. Na každém konci relace musí být deklarovaný stejný datový typ, aby bylo možné určit přesné propojení záznamů. Ukládací enginy pak používají tyto relace a za pomoci cizích klíčů nedovolí mazání záznamů, pokud je na tabulku odkazováno. V případě

XML jsou soubory na sobě nezávislé a ověřování referencí na jiné záznamy je nutné provádět dotazováním nad druhým dokumentem s vypsáním možných hodnot. Spojení tabulek je prováděno příkazem JOIN ON a parametry spojující relace. Dotaz spojující dvě tabulky a vypisující majitele automobilů má v SQL zápis:

```
SELECT sum(auto.najetoKm), majitel.jmeno, majitel.prijmeni
FROM auto INNER JOIN majitel ON majitel.id = auto.majitel_id
GROUP BY majitel.jmeno, majitel.prijmeni
```

V XQuery je zápis výrazně složitější a spojují se v něm dva XML soubory:

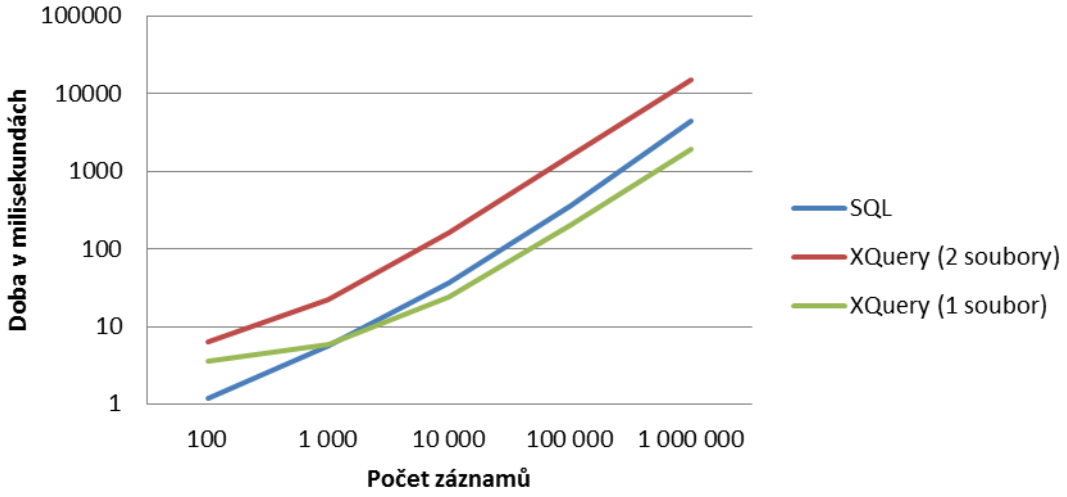
```
declare variable $doc as document-node() external;
declare variable $doc2 as document-node() external;
for $x in $doc/auta/auto,
$y in $doc2/majitele/majitel
where $x/majitel_id = $y/@id
group by $z := $y/jmeno, $zz := $y/prijmeni
return <vysledek>
<jmeno>{$z}</jmeno>
<prijmeni>{$zz}</prijmeni>
<najetoKm>{sum($x/najetoKm)}</najetoKm>
</vysledek>
```

Parsování dvou souborů je výpočetně a paměťově náročné. Po zparsování je dále nutné oba dokumenty spojit podmínkou where a až poté seskupovat.

Jiná situace nastává v případě jednoho hierarchicky složeného dokumentu, kde rodičovským elementem je majitel a každý element majitel má za své potomky všechny jeho automobily. V tomto případě je možné odebrat element <majitel_id>, protože na něj lze přistupovat přes vazbu rodič-potomek.

Počet záznamů	SQL (v ms)	XQuery (v ms) (2 soubory)	XQuery (v ms) (1 složený soubor)
100	1,17	6,46	3,52
1 000	5,72	21,93	5,84
10 000	36,11	162,25	24,46
100 000	363,20	1564,14	205,68
1 000 000	4509,85	14882,02	1912,24

Tabulka 11 - Doba zpracování spojovacího dotazu



Obrázek 12 - Graf doby zpracování spojovacího dotazu

Pokud je spojováno více tabulek respektive více XML souborů, je XQuery výrazně pomalejší. Při jednom milionu záznamů trvá spojení a výběr 14 sekund. Takto dlouhá doba zpracování je prakticky nepoužitelná. Ovšem pokud jsou dva XML datové soubory vhodně hierarchicky spojeny, tak již při tisíci záznamech je doba zpracování XQuery stejná jako SQL a při stoupajícím počtu záznamů se snižuje. Čím více tabulek je spojováno, tím geometricky výpočetní náročnost vyřízení dotazu. Hierarchie v XML umožňuje spojení obejít a tak ušetřit dobu na spojování datových objektů.

4.6.8 Výběrový dotaz s použitím vestavěných funkcí

Oba jazyky disponují množstvím vestavěných funkcí. Mezi nejčastěji používané patří funkce agregační. V následujícím testu jsou použity právě tyto funkce.

Počítací funkce COUNT v SQL zápisu:

```
SELECT COUNT(id) AS pocet_aut
FROM auto
```

XQuery zápis:

```
let $z := $doc//auto
return count($z)
```

Sumarizační funkce SQL zápis:

```
SELECT SUM(najetoKm) AS celkove_najeto
FROM auto
```

XQuery zápis:

```
let $z := $doc//auto
return sum($z/najetoKm)
```

Funkce průměru SQL zápis:

```
SELECT AVG(rokVyroby) AS prumerny_RV
FROM auto
```

XQuery zápis:

```
let $z := $doc//auto
return avg($z/rokVyroby)
```

Funkce vracející největší hodnotu SQL zápis:

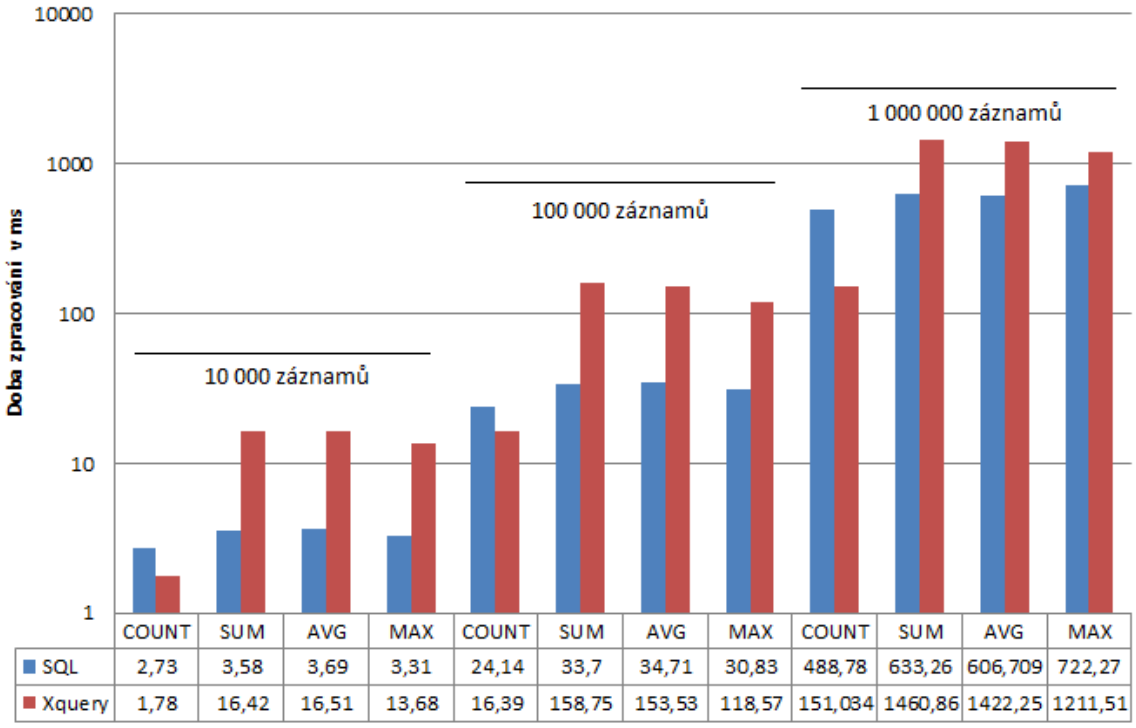
```
SELECT MAX(najetoKm) AS maximalne_najeto
FROM auto
```

XQuery zápis:

```
let $z := $doc//auto
return max($z/najetoKm)
```

Počet záznamů	Funkce	SQL (v ms)	XQuery (v ms)
10 000 záznamů	COUNT	2,73	1,78
	SUM	3,58	16,42
	AVG	3,69	16,51
	MAX	3,31	13,68
100 000 záznamů	COUNT	24,14	16,39
	SUM	33,7	158,75
	AVG	34,71	153,53
	MAX	30,83	118,57
1 000 000 záznamů	COUNT	488,78	151,034
	SUM	633,26	1460,862
	AVG	606,709	1422,252
	MAX	722,27	1211,509

Tabulka 12 - Doba zpracování vestavěných funkcí



	COUNT	SUM	AVG	MAX	COUNT	SUM	AVG	MAX	COUNT	SUM	AVG	MAX
SQL	2,73	3,58	3,69	3,31	24,14	33,7	34,71	30,83	488,78	633,26	606,709	722,27
Xquery	1,78	16,42	16,51	13,68	16,39	158,75	153,53	118,57	151,034	1460,86	1422,25	1211,51

Obrázek 13 - Graf doby zpracování vestavěných funkcí

Jedinou funkcí, kterou zpracuje XQuery rychleji je paradoxně funkce count. I přes indexování primárního klíče v MySQL pracuje XQuery rychleji. U ostatních funkcí se s rostoucím počtem záznamů časy zpracování pomalu přibližují. Zatímco u 10 000 záznamů je XQuery 4x pomalejší, tak 1 000 000 záznamů je to již jen dvojnásobně. Větší soubory bohužel nemohly být v rámci testu kvůli hardwarovým omezením otestovány a tak lze pouze předpokládat, že při stejném tempu by XQuery rychlostně dorovnalo SQL kolem 10 mil. záznamů.

4.7. Zabezpečení

Zabezpečení dat proti neoprávněnému přístupu patří v MySQL mezi to nejdůležitější. Základní funkcí řízení přístupu je rozpoznat uživatele přistupujícího z daného počítače a přidělit mu příslušná práva nad databázemi. Informace o uživatelích a právech jsou uloženy v tabulkách "user", "db", "host", "tables_priv" a "columns_priv" patřících do databáze "mysql". Tato databáze byla vytvořena již během instalace. Kontroluje se platnost celé identifikační trojice (uživatel - heslo - počítač). Všechny uživatele lze vypsat pomocí příkazu

```
select * from mysql.user;
```

Po instalaci se vytvoří tabulky privilegií a do nich je zaznamenán uživatel root, který má neomezená privilegia na všechny operace. Hned po instalaci je nutností nastavit heslo pro uživatele root a to příkazem

```
 UPDATE user SET Password=PASSWORD('heslo') WHERE
user='root';
```

Další uživatele je možné přidávat do databáze pomocí příkazu GRANT. Přidání uživatele, který bude mít přístup pouze z místního počítače a právo používat jen příkaz select na všechny databáze

```
GRANT SELECT ON *.* TO user@"%" IDENTIFIED BY 'heslo';
```

V MySQL lze přiřadit práva každému uživateli přesně podle potřeby a omezit přihlašování pouze na určitý počítač respektive IP adresu. Mělo by být dodrženo několik základních pravidel:

- Pouze uživatel "root" může přistupovat k databázi "mysql".
- Vždy udělovat jen ta práva, která jsou nutná pro daný úkol.
- Povolit přístup jen z minima počítačů.
- Všichni mají zavedené heslo pro přihlašování.

V případě XML jsou možnosti zabezpečení značně omezenější. Jelikož je XML soubor obsahující znaky nějaké kódovací sady, je možné ho otevřít v kterémkoli textovém editoru. Takto má přístup k datům jakýkoli uživatel, který má práva na čtení souboru. V případě uložení souboru na webovém serveru apache se přístup k souborům omezí nastavením práv ke složce, ve které jsou soubory umístěny. Základní nastavení pro přístup pouze z určité adresy nastavují příkazy allow a deny. Nastavení přístupu k souborům pouze z místního PC se uloží do souboru .htaccess do složky s xml soubory.

```
order deny,allow
deny from all
allow from localhost
```

Pro zpřístupnění souboru jednotlivým uživatelům je vytvořen druhý soubor .htpasswd obsahující uživatele a heslo ve tvaru uživatel:heslo. Obsah htacces odkazuje na soubor s uživateli a požaduje tím platného přihlášeného uživatele.

```
AuthUserFile C:/xampp/web/.htpasswd
AuthName "Prosím zadejte vaše jméno a heslo"
AuthType Basic
require valid-user
```

Toto jednoduché zabezpečení zamezuje pouze přístup k datovým souborům xml. Jednotlivá práva na XQuery příkazy se musí specifikovat ve skriptovacím jazyce PHP.

Mezi další možnosti zabezpečení XML patří digitální podpis a šifrování dat. Jde zašifrovat celý soubor, jednotlivý element, obsah dokumentu. Šifrovaný dokument může vypadat následovně:

```
<?xml version='1.0' ?>
<auta>
<EncryptedData
Type='http://www.w3.org/2001/04/xmlenc#Element'
 xmlns='http://www.w3.org/2001/04/xmlenc#'>
<EncryptionMethod

Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
 <CipherData>
 <CipherValue>C5X1I65RCX…</CipherValue>
 </CipherData>
 </EncryptedData>
</auta>
```

4.8. Omezení dat

V MySQL databázi při vytváření tabulky uživatel musí specifikovat, jaká data má tabulka obsahovat. Tyto hodnoty musí být atomické a pouze jednoho typu. Po vytvoření tabulky je možné vkládat data, která striktně dodržují specifikované datové typy a formáty. Pokud jsou vkládaná data jiného typu, je vložena základní hodnota nebo hodnota null. Databáze si takto sama vyhodnocuje, která data přijme. V případě pokusné databáze byly

použity datové typy int(11), varchar(20), enum('Be….'), int(4), int(6), int(11). Databáze nepřijme jiná data než číslici na pozici jedna, text na pozici dva, jeden z výčtu prvků na pozici tři a číslice na pozicích 4, 5 a 6. Poslední int(11) je referencí na identifikátor v jiné tabulce a databáze sama hlídá omezení a nedovolí zápis jiného čísla.

XML dokument nemá při samotném vytvoření specifikované datové typy a je možné ukládat jakákoli znaková data. Omezení je možné vytvořit až po samotném naplnění dat a poté zkontrolovat platnost dat. I přes existenci definičního schema dokument může obsahovat nesprávná data, ale stává se tak neplatným. V této práci je ke kontrole použito schema Relax NG.

```
<?xml version="1.0" encoding="UTF-8"?>
<element name="auta"
xmlns="http://relaxng.org/ns/structure/1.0"
 datatypeLibrary="http://www.w3.org/2001/XMLSchema-
datatypes">
  <oneOrMore>
    <element name="auto">
      <attribute name="id"><data type="integer"><param
name="minExclusive">0</param></data></attribute>
      <element name="znacka"><data type="string"><param
name="maxLength">20</param></data></element>
      <element name="palivo">
        <choice>
          <value>Benzín</value>
          <value>Diesel</value>
          <value>LPG</value>
        </choice>
      </element><element name="rokVyroby"><data
type="gYear"/></element>
      <element name="najetoKm"><data
type="integer"></data></element>
      <element name="barva"><data type="string"><param
name="maxLength">15</param></data></element>
      <element name="majitel_id"><data
type="int"/></element>
```

```
    </element>
  </oneOrMore>
</element>
```

Toto schéma odpovídá specifikovaným datových typům v MySQL databázi. Jediným problémem je odkazování na referenční hodnoty ze souboru majitele.xml, které v Relax NG nelze definovat. Možnost odkazování na jiné elementy je možná pouze v rámci jednoho souboru za použití datových typů ID a IDREF. Kontrola, zda je xml dokument validní**,** je provedena php skriptem

```
$rngschema = file_get_contents("relax.xml");
$dom_xml = new DomDocument;
$dom_xml->loadXML(file_get_contents("auta10.xml"));
$dom_xml->relaxNGValidateSource ( $rng )
```

Skript načte soubor s datovým schématem a poté vytvoří objekt DomDocument, do kterého zparsuje samotný datový soubor. PHP funkce relaxNGValidateSource spuštěná na objekt DomDocument s parametrem obsahujícím Relax NG schema ověří validitu dokumentu a vypíše, zda je dokument validní nebo ne. Pro správnou funkčnost je nutností použít plnou xml syntaxi Relax NG schema a ne pouze kompaktní zápis.

4.9. Velikost

Srovnání velikostí mezi databázovým uložením a XML dokumentem podle počtu uložených záznamů představuje tabulka:

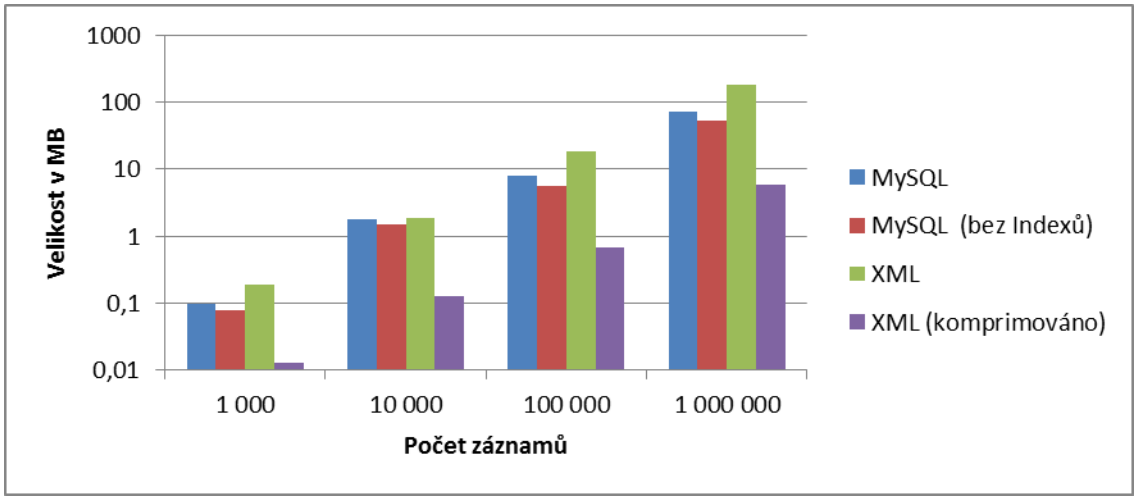
Počet záznamů	MySQL (v MB)	MySQL (v MB) (bez Indexů)	XML (v MB)
1 000	0,096	0,08	0,187
10 000	1,8	1,5	1,83
100 000	8	5,5	18,4
1 000 000	73,1	53,6	184

Tabulka 13 - Velikosti MySQL a XML

Z tabulky je patrný konstantní nárůst velikosti u XML dokumentů, zatímco uložení dat v MySQL má zpočátku nepravidelný průběh. Přibližně stejné místo na disku zabírá jak databáze**,** tak XML dokument o velikosti 10 tisíc záznamů. Při dalším zvětšování počtu záznamů se databáze chová asi 2,5krát úsporněji než XML. To je způsobeno opakováním značek v dokumentu, které pak zvyšují velikost. Tato nehospodárná vlastnost může být kompenzována výborným poměrem při komprimování XML dokumentů. Testovaný dokument o velikosti jeden milion záznamů má kompresní poměr 30. To znamená, že při komprimaci je snížena jeho velikost na 1/30 původní velikosti. Takto velký kompresní poměr je způsoben velkou redundancí dat, kterou XML dokumenty obsahují. Čím menší je dokument**,** tím menší je komprimační poměr, ale stále se komprimací velikost mnohokrát snižuje.

Počet záznamů	XML (v MB)	XML (v MB) (komprimováno)
1 000	0,187	0,013
10 000	1,83	0,13
100 000	18,4	0,67
1 000 000	184	5,81

Tabulka 14 - Velikost nekomprimovaného a komprimovaného XML



Obrázek 14 - Graf velikostí

4.10. Zálohování a přenositelnost dat

V případě zálohování databáze existuje několik možností zálohování. Je možné zálohovat

- Strukturu databáze
- Samotná data
- Část dat
- Celou databázi

Každá záloha je jinak časově náročná. Čím méně dat se zálohuje, tím je záloha rychleji hotova. V případě zálohování je nutné dobře zvolit, jaká data zálohovat, kdy je zálohovat a kdo se o samotný proces postará.

MySQL je ve většině případů používána jako webová databáze instalovaná na linuxový server s podporou jazyka PHP. Tato kombinace umožňuje zálohovat i omezenými právy. Velmi rozšířeným nástrojem pro kompletní správu databáze je PhpMyAdmin. PhpMyAdmin umí exportovat data z MySQL do několika nejběžnějších formátů.

- Textové soubory oddělené čárkou nebo středníkem.
- Příkazy SQL. Vznikne sice poněkud větší soubor, ale ten bude obsahovat přímo sadu příkazů INSERT s tím, že spuštěním tohoto souboru proti cílové databázi dojde k obnově dat.
- Soubory Microsoft Excel.
- Soubory Microsoft Word. Z hlediska zálohování se moc nehodí, neboť případná obnova dat z takového souboru by nebyla triviální.

PHPMyAdmin má mnoho možností nastavení a je téměř ideálním nástrojem jak pro rychlé zálohování, tak pro pokročilou archivaci dat. Tento nástroj je stále vyvíjen a má velkou podporu uživatelů. Pro zálohu MySQL dat je možné využít vlastních skriptů, které mohou být spouštěny z jakéhokoli prohlížeče nebo automaticky pomocí nástroje cron obsaženém v systému Linux.

Samotný PHPMyAdmin používá příkazu mysqldump. Tento příkaz vytvoří zálohu dat tak, že sestaví SQL příkazy, které povedou k vytvoření dané tabulky nebo tabulek. Je

vytvořena jak definice tabulky, tak i data. Produkuje výstup, který je čitelný, textový a jde dobře zkomprimovat.

Obnovení dat v nástroji PHPMyAdmin probíhá obdobně jako jejich záloha. Nástroj přijme soubor s příkazy a odešle je ke zpracování MySQL serveru. Pro obnovu je podporováno více formátů souboru. Nejběžnějším je SQL příkaz vytvořený právě příkazem mysqldump, dále jsou podporovány formáty CDV, DocSQL, OpenDocument a XML.

Zálohování XML dokumentů je podstatně jednodušší. Jelikož jde o textové soubory, je předmětem zálohy jednoduchá kopie souboru. Struktura dokumentu je obsažena v externím definičním souboru, který je nutné zálohovat spolu s XML dokumentem. I v tomto případě je možné zálohovat pouze část dokumentu a v případě obnovy spojit zálohy dohromady do jednoho souboru. Všechny zálohy XML souborů jsou prováděny pomocí PHP skriptů a XQuery jazyka. Kopie dokumentu slouží jako záloha pro obnovení dokumentu. Jednoduchý skript pro zálohu xml souborů:

```
$file = 'dokument.xml';
$newfile = 'dokument.xml.bak';
copy($file, $newfile);
```

4.11. Finanční hledisko

MySQL je v základní verzi distribuována zdarma. Zorba procesor s PHP je jako open sourcový projekt taktéž šířen zdarma. Náklady na pořízení jsou nulové a srovnávat lze měsíční náklady na provoz serveru, popřípadě náklady spojené s programováním aplikace dolující data z databáze nebo XML souborů.

Jelikož jsou si oba jazyky velmi podobné a prakticky dovolují vytvářet stejné výstupy, tak na dobu a složitost vytváření aplikace nemá jeden či druhý způsob vliv. V případě provozování aplikace na Apache serveru je dotaz zpracován buď databází, nebo Zorba procesorem a PHP skript dále pracuje s výslednými daty. V tomto případě je složité srovnávat časovou náročnost na vytvoření aplikace, neboť oba přístupy poskytují stejný výstup a programátor se o způsob zpracování dat nemusí starat.

U webových serverů nebývá k dispozici Zorba procesor a samotné PHP nepodporuje XQuery. Samotný běh Zorba procesoru vyžaduje aplikační server zatímco MySQL je dostupné u všech webových hostingů. Parsování XML dokumentů je paměťově náročné a s tím souvisí i nároky na aplikační server. Virtuální aplikační servery jsou oproti webovým hostingům dražší, ale je možné na nich provozovat vlastní aplikace, v tomto případě XML procesor Zorba. Následující tabulky udávají měsíční ceny webového hostingu a virtuálního serveru.

Prostor pro databáze	PHP memory_limit	PHP upload_max_filesize	Cena
1 GB	128 MB	32 MB	25 Kč
2 GB	256 MB	128 MB	63 Kč

Tabulka 15 - Webhosting od společnosti Wedos

HDD	RAM	Cena
15 GB	512 MB	100 Kč
30 GB	1 GB	200 Kč
60 GB	2 GB	400 Kč
90 GB	3 GB	600 Kč
120 GB	4 GB	800 Kč

Tabulka 16 - VPS od společnosti Wedos

Webhosting nabízí poměrně omezené místo pro ukládání dat do databáze o velikosti 1GB resp. 2GB a má taktéž nastaveny omezené limity pro PHP zpracování. Tomu ale odpovídá cena, která je měsíčně 25 Kč resp. 63 Kč bez DPH. U VPS je limit místa pro uložení dat násobně vyšší a je garantována velikost RAM, ale odpovídá tomu i vyšší měsíční cena. Pro zpracování XML dokumentu je potřeba velké množství paměti a nejlevnější varianta by byla i v případě testovacích dokumentů vytvořených v práci nedostačující.

Za zpracování XQuery by se v případě pořízení VPS od firmy Wedos platilo násobně více. Zde má MySQL výhodu v integraci do webového hostingu s PHP.

5. Zhodnocení výsledků a doporučení

Při hodnocení výkonnosti je nutné vzít v úvahu použitý hardware a software, a to, že výsledky mohou být ovlivněny použitou verzí MySQL nebo jazyka PHP. PHP je jazyk zaměřený právě na web se značnou podporou databáze MySQL. Naproti tomu podpora jazyka XQuery jako takového chybí. PHP disponuje knihovnami pro zpracování XML dokumentů jako DOM, libxml, SimpleXML, XML Parser, XML Reader nebo XSL. Použití těchto knihoven je však nevhodné pro práci s velkými XML soubory. Podpora XQuery byla do PHP přidána za pomoci externí knihovny Zorba, která je distribuována zdarma pod licencí Apache License.

Z výkonnostních výsledků je možné vypozorovat delší vývoj jazyka SQL oproti mladšímu, stále ve vývojovém stádiu, jazyku XQuery. Ve většině případů byly příkazy SQL provedeny rychleji a převážně u objemných dokumentů se projevily větší rozdíly. Především u běžného vyhledávání má databázový systém výhodu v indexaci položek. U XML dokumentu záleží na typu dotazu a struktuře dokumentu. Souhrnné a spojovací dotazy mohou být v XQuery zpracovány rychleji kvůli hierarchické struktuře dokumentu, díky které odpadá náročné spojování nebo seskupování položek.

Výkonově se jazyky liší v řádu několika procent ve prospěch jazyka SQL. Hlavním nedostatkem pro práci s XML dokumenty je nutné parsování souboru. Parsování extrémně prodlužuje dobu potřebnou ke zpracování dokumentu a je přímo závislé na velikosti dokumentu. Z výsledků vychází přibližný stav parsování 10MB souboru za 1 sekundu. Parsovaný dokument zabírá také velkou část místa v operační paměti a v případě, že se nevejde do paměti, nelze na něm provádět operace. V případě testovacích souborů byl přepočet přibližně desetinásobný, tzn. 10MB velký soubor zabíral v paměti 100MB. Pokud musí být dokument před každou operací znovu parsován, je výsledná výkonnost na nepřijatelné úrovni.

Výsledky ze zpracování XML dokumentu je vždy nutné uložit znovu do souboru, nebo s nimi dále pracovat. MySQL při provedení příkazů data ukládá hned sama. Tímto je XML dokument nepoužitelný pro práci více uživatelů najednou. Každý uživatel musí pracovat vždy s celým dokumentem, a dokud soubor neuloží, neměl by nikdo jiný mít k souboru

přístup. MySQL podporuje uzamykání na úrovni jednotlivých záznamů, čímž umožňuje práci více uživatelů nad jednou tabulkou.

Velikostně je ve výhodě MySQL. Velikost XML dokumentu roste konstantně s počtem záznamů. V MySQL je velikost ovlivněna ukládáním indexů. Pokud nejsou ukládány indexy, je velikost databáze menší, ale zpracování dotazů trvá déle. Při velikosti nad 10MB zabírá XML dokument třikrát více místa na disku než MySQL. XML dokument má velmi dobré komprimační vlastnosti a pro archivaci je tento formát vhodnější.

Bezpečnost MySQL je na vysoké úrovni a umožňuje velmi variabilní přístup k datům. U XML dokumentů musí být bezpečnost řešena na úrovni programovacího jazyka. XML lze zabezpečit šifrováním a digitálním podpisem.

SQL	XQuery
+ velké množství vývojářů	+ převzatá syntaxe z SQL
+ podporuje většina programovacích jazyků	+ podporováno Oraclem, IBM, Microsoftem
+ dlouho ustanovený standard	+ nově ustanovený standard
+ jednoduchá syntaxe	+ práce s hierarchickými daty
+ rychlé zpracování	+ velmi flexibilní formulace výstupů
- pouze atomické datové typy	- malý počet vývojářů
- složitě vyjadřuje hierarchické struktury	- složitá formulace dotazů
	- při špatně formulovaných dotazech nízká výkonnost

Tabulka 17 - Srovnání SQL a XQuery

Z výsledků práce vychází MySQL spolu s jazykem SQL jako lepší řešení pro webové aplikace. K webovým aplikacím přistupuje velké množství uživatelů, kdy každý může mít jiná přístupová práva. Pro tento model je SQL jazyk navržen a dobře funguje. Pokud by byl použit jako datové úložiště XML dokument pro velké množství záznamů a připojení většího množství uživatelů, byla by aplikace výkonnostně nepoužitelná a složitě kódovaná pro bezchybný provoz.

Pro lokální aplikace, kdy aplikaci používá málo uživatelů a všichni mají stejná práva je pak rozhodnutí, zda používat XML nebo MySQL na místě. MySQL vyžaduje běh serveru, na kterém databáze běží a server poskytuje data. XML dokumenty nic takového nevyžadují a aplikace je může používat jako takové. Dalším měřítkem je předpokládaná velikost databáze. XML dokument o větší velikosti než 100MB je již náročné zpracovat a vhodnější je pak databázové zpracování. Jako příklad aplikace, která používá tento přístup, je možné uvést MS Word 2007. Balíček docx obsahuje několik XML dokumentů, které aplikace při otevření zpracuje a zobrazí.

Mobilní aplikace využívají pro interní ukládání dat převážně XML dokumenty. Neplýtvají tak výkonem pro SQL server a data načítají přímo z dokumentů. Tyto aplikace nepředpokládají velké objemy souborů a zpracovávají tak data o velikosti desítek záznamů. Pokud aplikace vyžaduje ukládat větší objemy dat, je většinou použit externí databázový server, ke kterému se aplikace připojuje.

Relační databáze	XML dokument
+ více uživatelů	+ dokument čitelný v každém textovém editoru
+ rychlost zpracování	+ přenositelnost
+ velké objemy dat	+ není nutný server na zpracování
- nutnost serveru	+ hierarchická struktura
- spojování tabulek	- současně může zpracovávat jeden uživatel
- atomické hodnoty	- nutné parsování pro další použití
	- značně klesá výkonnost s objemem dat

Tabulka 18 - Srovnání relační databáze a XML dokumentu

Jelikož jazyk XQuery není tak rozšířený jako SQL, nemá velkou podporu v programovacích jazykách a ve většině případů je nutné použít externí knihovny nebo nástroje. V práci byl použit XQuery procesor Zorba, který je velmi rozsáhlý a jeho API podporuje jazyky C++, C, XQJ, Java, PHP, Python, C# a Ruby. Nutností pro použití je

nainstalování celého nástroje do systému a následné propojení API s programovacím jazykem.

Kdy použít databázi a kdy XML dokument záleží tedy na několika faktorech.

- S jak velkým objemem dat bude nakládáno

Databáze při středních velkých objemech, XML při menších objemech

- Kolik uživatelů bude k datům přistupovat

Databázový přístup pro více uživatelů, XML pro malý počet uživatelů

- Jaká je struktura dat

XML pokud struktura zasahuje hierarchicky velmi do hloubky

- Jak často budou data přenášena

Databáze pro přenos jednotlivých záznamů, XML pro přenos celých dokumentů

- Jaké zabezpečení je potřebné

Databáze pro vysoký stupeň zabezpečení s možností přidělovat práva, XML dokumenty pro možnost zobrazení kýmkoli.

6. Závěr

Diplomová práce zmapovala situaci v oblasti ukládání dat. Byly analyzovány dva způsoby a to databázové ukládání dat a ukládání dat pomocí XML dokumentů. V teoretické části byl vymezen aktuální stav databáze MySQL a XML jazyka. Praktická část se zaměřila na srovnání těchto dvou způsobů a vymezila několik hledisek vhodných ke srovnání.

Všechny výkonnostní testy probíhaly v testovací vygenerované databázi a příslušných XML dokumentech a to za pomoci skriptovacího jazyka PHP. Jelikož je jazyk XQuery stále ve vývoji, je možné počítat do budoucna se změnami ve výsledcích práce. Hlavně pak se zlepšením podpory a výkonnosti tohoto jazyka. Spolu s výkonovým srovnáním byla okrajově probrána velikost, bezpečnost, integrita a zálohování. Z vyhodnocení vyplývá, že ač jsou jazyky XQuery a SQL velmi podobné, není možné je používat ve stejných situacích. Oba jsou zaměřeny na zpracování dat a především na webové aplikace.

Nejrozšířenějším jazykem na webu je bezesporu PHP. Tento jazyk si velmi dobře rozumí s MySQL a zvládá i základní operace nad XML dokumenty. Avšak nejmladší na trhu jsou databáze objektové. Otázkou je, zda se do budoucna změní trend relačních databází a z větší části se začnou používat databáze objektové.

Snaha zjednodušit ukládání dat a objektů do databáze je stále větší. Vzniklo proto objektově relační mapování. Jedná se o vrstvu aplikační logiky, která se vloží mezi objektovou hierarchii v programu a relační databázi a ta zde automatizuje transformaci objektů používaných v programu do tabulek v databázi a naopak. Tímto způsobem se nasazení plně objektových databází více oddálilo a do budoucna není jasný jejich vývoj.

7. Seznam použitých zdrojů

1. SCHWARTZ. Baron. *MySQL profesionálně: optimalizace pro vysoký výkon* [překlad Jan Pokorný]. Brno: Zoner Press, 2009. 712 s. ISBN 978-80-7413-035-9

2. KOFLER, Michael. *Mistrovství v MySQL 5*. Computer Press, 2007; 808 s. ISBN 978-80-251-1502-2

3. HUB, Miroslav. *Technologie internetu - XML : distanční opora*. 1. vydání. Pardubice: Univerzita Pardubice, 2010. 84 s. ISBN 978-80-7395-306-5

4. MLÝNKOVÁ, Irena. *XML technologie: principy a aplikace v praxi*. Praha: Grada Publishing, 2008. 267 s. ISBN 978-80-247-2725-7

5. ZAJÍC. Petr. *MySQL tutoriály*. [on-line]. Dostupný z www: <http://www.linuxsoft.cz/mysql/>.

6. D. C. Fallside. *XML Schema Part 0*. W3C. 2001. [on-line]. Dostupný z www: <http://www.w3.org/TR/xmlschema-0/>.

7. VAN DER VLIST. Eric. *Relax NG. O'Reilly*. 2003. ISBN 0-596-00421-4. [on-line] <http://books.xmlschemata.org/relaxng/page1.html>.

8. CLARK. James. *Relax NG Tutorial*. 2001. [on-line]. Dostupný z www: <http://relaxng.org/tutorial-20011203.html>.

9. XPath Tutorial. [on-line]. Dostupný z www: <http://www.w3schools.com/xpath/>.

10. XQuery Tutorial. [on-line]. Dostupný z www: <http://www.w3schools.com/xquery/>.

11. XQuery FLWOR Expressions. [on-line]. Dostupný z www:
<http://www.w3schools.com/xquery/xquery_flwor.asp>.

12. XAMPP. [on-line]. Dostupný z www:
<http://www.apachefriends.org/en/xampp.html>.

13. PHP. [on-line]. Dostupný z www:
<http://php.net/>.

14. Zorba. [on-line]. Dostupný z www:
<http://www.zorba-xquery.com/>.

15. Zorba demo. [on-line]. Dostupný z www:
<http://www.zorba-xquery.com/html/demo>.

8. Přílohy

A. Seznam obrázků

B. Seznam tabulek

C. Použité zkratky a cizí slova

Apache	Softwarový webový server s otevřeným kódem pro GNU/Linux, Solaris, Mac OS X, Microsoft Windows a další platformy
API	Rozhraní pro programování aplikací
DBMS	Softwarové vybavení, které zajišťuje práci s databází. Tvoří rozhraní mezi aplikačními programy a uloženými daty
DOM	Objektově orientovaná reprezentace XML nebo HTML dokumentu
engine	Formát úložiště dat
GPL	Licence pro svobodný software
HTML	Značkovací jazyk pro hypertext. Hlavní jazyk pro publikování dokumentů na Internetu.
MySQL	Relační databázový systém založený na architektuře klient-server

Parser	Program, který převádí XML dokument na objekty
PHP	Skriptovací programovací jazyk. Je určený především pro programování dynamických internetových stránek a webových aplikací
phpMy Admin	Nástroj napsaný v jazyce PHP umožňující jednoduchou správu obsahu databáze MySQL prostřednictvím webového rozhraní
Relax NG	Předpis struktury a datových typů pro třídu dokumentů v XML
RSS	XML formáty určené pro čtení novinek na webových stránkách
SQL	Standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích
Trigger	V databázi definuje činnosti, které se mají provést v případě definované události nad databázovou tabulkou
Unicode	Unicode je tabulka znaků všech existujících abeced, která v současnosti obsahuje více než 110 000 znaků.
W3C	Mezinárodní konsorcium, jehož členové společně s veřejností vyvíjejí webové standardy pro World Wide Web
XAMPP	Webový server obsahující základní části Apache, MySQL, PHP a Perl
XML	Je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C
Xpath	Jazyk, pomocí kterého lze adresovat části XML dokumentu
XQuery	Dotazovací a funkční programovací jazyk navržený pro práci s XML dokumenty
Zorba	Open source XQuery procesor napsaný v C++, který implementuje několik XQuery a XML specifikací