UNIVERZITA PALACKÉHO V OLOMOUCI

PEDAGOGICKÁ FAKULTA

Katedra technické a informační výchovy

Bakalářská práce

Jakub Štos

Porovnání klient/server databázových systémů s vloženou databází SQLite z hlediska výkonu a funkcionality

Vedoucí práce: Mgr. Jan Kubrický, Ph.D

Olomouc 2017

Čestné prohlášení

Prohlášení

Prohlašuji, že jsem bakalářskou práci na téma: „Porovnání klient/server databázových systémů s vloženou databází SQLite z hlediska výkonu a funkcionality“ vypracoval samostatně pod odborným dohledem vedoucího práce a uvedl jsem všechny použité podklady a literaturu.

V Olomouci dne. …….. Podpis ……

Poděkování

Děkuji panu Mgr. Janu Kubrickému, Ph.D. za cenné rady a připomínky během tvorby této bakalářské práce.

OBSAH

ÚVOD

S určitou formou databáze se dnes můžeme setkat v mnoha sférách lidského života a často si to ani neuvědomujeme. Bezprostředně nás obklopují při nakupování v obchodech, výběru z bankomatu nebo při cestování na nádraží. Dá se říci, že bychom se v dnešní době bez jejich využití neobešli. Základním konceptem databází je snaha data uspořádat a dále s nimi pracovat. Takto uspořádáná data nám pak v kontextu poskytují informace. S rozvojem počítačových technologií vzrostl význam informací navíc i z technického hlediska, a tak se tento proces začal postupně rozšiřovat a zjednodušovat.

Než začneme se samotnou prací, musíme si nejdříve ujasnit terminologii. Pojem databáze je často používaný spíše ve významu databázový sytém. Ani to však není úplně přesné; ve svém plném znění se jedná o tzv. systém řízení báze dat. Rozdíl mezi těmito pojmy je důležité zmínit především z toho důvodu, abychom si uvědomili, že se jedná o velice komplexní systém a ne jen o pouhou sběrnu dat.

Správná volba DBMS je velice důležitým a často opomíjeným faktorem při zakládání databáze. Mezi nejznámější databázové systémy patří například MySQL, Oracle, Microsoft SQL server nebo PostgreSQL. Všechny spojuje způsob uložení dat, ke kterému využívají relační databázový model. Na přelomu tisícelití se začalo pracovat na dalším systému využívajícího implementaci toho modelu - SQLite. Od svých populárnějších konkurentů se ale zásadně liší ve své archtektuře a funkčnosti. Jak již z názvu vyplývá, jedná se o „odlehčenou verzi“, která namísto klasického modelu klient/server pracuje jako vložená knihovna přímo v aplikaci. Zcela tak odpadá nutnost použití serveru, což s sebou nese jak nesporné výhody, tak i určitá úskálí.

CÍLE PRÁCE

Hlavním cílem této práce je představení systému SQLite a jeho srovnání s ostatními klient/server systémy. Práce je soustředěna především na architekturu a výkon. Podává ucelené informace o tom, kdy a z jakého důvodu je vhodnější použít vloženou databázi SQLite a kdy je lepší sáhnout po klasickém systému na klient/server bázi jako MySQL nebo Oracle.

V první části se seznámíme s nezbytně nutnými pojmy týkajících se databází a rozebereme jednotlivé vlastnosti všech systémů. V závěrečné části tyto znalosti aplikujeme a ověříme testováním. Srovnání bude probíhat na základě rychlosti zpracování operací, způsobu implementace databáze a funkčnosti.

1 ZÁKLADNÍ POJMY

1.1 Databáze a databázový systém

Pojem databáze je často v širším významu chápán jako softwarová aplikace. Abychom ale byli přesní, samotná databáze je „*…uspořádaná kolekce dat. Nezáleží na tom, zda ke sběru a ukládání dat používáte papír nebo počítačový program. Pokud sbíráte a ukládáte data organizovaným způsobem za určitým účelem, pak máte databázi.“* [1, s. 42] Z této definice explicitně vyplývá, že se může jednat jak o obyčejnou kartotéku, tak i o složitější systémy zpracované v elektronické podobě. V našem případě se jedná o počítačový program.

Pro správu celé databáze slouží aplikační program pojmenovaný poměrně složitým názvem systém řízení báze dat[1]. Pod správu spadá veškerá manipulace s daty, tedy jejich získávání, zpracování, editace i mazání.

Drtivá většina těchto systémů pracuje na bázi klient/server. Uživatel zde figuruje v roli klienta používajícího aplikaci, která svá data získává ze serveru. Tam jsou uloženy vešekeré součásti DBMS, které zajišťují správu a přístupnost dat klientovi. Zatímco se server stará o vše potřebné, klient nemusí mít ani tušení o jeho existenci.

Mezi další velmi důležité vlastnosti DBMS můžeme zařadit autentizaci uživatelů, současný přístup více uživatelů najednou, ochranu dat a s tím spojené transakce. Základní kategorizace probíhá na základě databázového modelu a dotazovacího jazyka. Nejčastěji se setkáme s relačním modelem a strukturovaným jazykem SQL.

1.2 Databázové modely

Vývoj databázových modelů začal okolo roku 1970 a od té doby se neustále měnil a vyvíjel. Mezi vůbec první patřily hierarchický a síťový model a oba se výrazně podílely na vzniku relačního modelu.

1.2.1 Hierarchický databázový model

V tomto typu databáze jsou data strukturována jako obrácený strom. Diagram tabulek tvoří hierarchické struktury začínající od tzv. kořene, čili logicky nadřazené entity. Vztahy jsou určeny termíny rodič a potomek. Tato metoda vyžaduje perfektní

[1] dále zkracováno jen na DBMS z anglického *Database Management System*

znalost databáze, protože uživatel se přes kompletní strukturu musí k hledaným datům dostat od kořenové tabulky.

Jeho obrovskou nevýhodou je právě struktura rodič-potomek. Každý potomek musí být ve vztahu alespoň k jednomu rodiči. Má-li potomek více rodičů, nastává problém. Jako řešení se nabízí přidání neexistujícího duplicitního záznamu pro splnění podmínky relace, pak ale nastává problém s redundancí a nekonzistentností dat [1, s. 43].

1.2.2 Síťový databázový model

Vztah rodičů a potomků byl nahrazen termíny uzel a množinová struktura. Uzel reprezentuje soubor záznamů a množinová struktura reprezentuje a zřizuje vztah v síťové databázi. Uzel se dále dělí na dva typy - prvek a vlastník. Důležité je také zmínit, že uzly bylo možné spojovat do dalších struktur s jinými uzly v databázi [1, s. 45].

Největším problémem síťového modelu byla nemožnost snadno změnit strukturu databáze bez větších následků na databázovou aplikaci a její konzistenci. Použití relací se sice dočkalo výrazného zlepšení, ale stále se nejednalo o úplně ideální řešení.

1.2.3 Relační model

Základy teorie relačních databází definoval Dr. E. F. Codd roku 1970 v tzv. podmínkách minimálního relačního modelu. Jako základní stavební kámen této teorie si uvedeme následující tři pravidla.

- Všechny údaje v databázi jsou uloženy v tabulkách.
- Fyzická struktura údajů a jejich uložení jsou nezávislé a pro uživatele naprosto odstíněné. (Z hlediska uživatele neexistují žádné viditelné přístupové cesty).
- Pro práci s údaji v databázi předpokládáme existenci databázového jazyka, který umožňuje realizovat minimálně operaci výběru, omezení, projekce a spojení [2, s. 13].

Tabulka je v podstatě reprezentací vztahů a můžeme si ji jednoduše představit podobně jako excelovský list – obsahuje sloupce a řádky reprezentované jednotlivými buňkami. Sloupce reprezentují vlastnosti entity v tabulce a nazýváme je atributy. „*Každý sloupec ve správně navržené databázi obsahuje právě jednu hodnotu a jeho jméno identifikuje typ údaje, který je v něm uložen.“* [1, s. 72] Jako příklad si můžeme

uvést pole jméno a příjmení, která nám jasně dávají najevo, jaký typ záznamů v nich bude uložen. V řádcích potom uchováváme data, která se vážou k určité nadřazené entitě a ta pak označujeme je jako záznam. Důležité je si uvědomit, že jako záznam je brán celý řádek, který je odlišen od ostatních řádků pomocí jednoznačného identifikátoru.

id_osoba	jmeno	prijmeni	tel_cislo	datum_narozeni
1	Jan	Meca	777225355	12-05-1975
4	Aleš	Orság	605055917	22-11-1981
8	Tomáš	Húsek	737379940	15-01-1968
9	Milan	Žáha	604510823	16-09-1977

Tabulka 1 Ukázka záznamů v tabulce

U předchůdců bylo nutné znát dokonale strukturu databáze, aby byl uživatel schopný se k požadovaným záznamům dostat. „*Relační databáze ukládá data ve vztazích, které uživatel vidí jako tabulky.“* [1, s. 43]

Vazbu mezi tabulkami určují tři různé typy vztahů: 1:1, 1:N a M:N. „*Dvojice tabulek je spolu svázána vztahem typu 1:1, pokud je každý záznam v první tabulce svázán s právě jedním záznamem druhé tabulky*.“ [1, s. 227] Jedná se o přímou vazbu a často by se relace 1:1 dala vyjádřit i v rámci jedné tabulky.

Vztah, kdy jedna položka tabulky odpovídá více (tedy *n*) položkám druhé tabulky, nazýváme 1:N. Jako příklad si můžeme představit motocyklového závodníka, který může závodit pouze pro jeden tým. Za tento konkrétní tým ale závodí více jezdců, což odpovídá *n* záznamům. Jedná se o nejčastější typ vztahu mezi dvěma tabulkami. Hlavní výhodou je, že eliminuje redundanci dat na minimum.

To, co dost dobře nefungovalo ani v hierarchickém a ani v síťovém modelu, se zde tvoří mnohem přívětivějším způsobem. Řeč je o relaci typu M:N. Tato vazba vzniká za předpokladu, že více záznamům v jedné tabulce odpovídá více záznamů v tabulce druhé. Nejjednoduším provedením této relace je vytvoření tzv. spojovací tabulky.

Při tvorbě ukázek byly tabulky spojovány pomocí svých identifikátorů, kterým se také říká klíče. Pro naše účely a základní seznámení budeme používat dělení na klíče

primární a cizí[2]. Primárním klíčem je záznam nebo jejich kombinace, který jednoznačně určuje unikátnost záznamu a předchází tím jeho duplicitě. Cizí klíč je potom primární klíč druhé tabulky, který funguje na principu odkazu.

1.2.4 Objektově-orientovaný databázový model

Ačkoli je jasné, že svým provedením není relační model vhodný pro všechny typy databází, objektový model jej pravděpodobně nahrazovat nebude. Důvodem vzniku byl především vzestup objektově orientovaného programování (OOP) v 80. Letech, a tak na sebe totžný model nenechal dlouho čekat i v oblasti databází. Základem bylo usnadnění práce s daty v OOP.

Důvodem, proč se objektový model nikdy neuchytil a nebyl přijat jako standard byl především ten, že předělání milionů stávajících relačních databází na objektový model by přineslo více práce než užitku. Další nevýhodou byl fakt, že neposkytoval žádnou zásadní výhodu pro většinu komerčních databází, a tak se postupem času umístil do pozadí. Zásadní roli zde hrál také jazyk SQL, který i přes nesporné výhody objektového přístupu k programování jednoduše databázím vyhovoval více [3, s. 22].

1.3 SQL

Strukturovaný jazyk SQL (*Structured Query Language*) slouží k manipulaci a administraci dat v databázích. Jedná se o nejznámější dotazovací jazyk vůbec. Mezi jeho největší výhody patří především univerzálnost použití. Velmi dobře spolupracuje s ostatními programovacími jazyky jako C++, C#, Visual Basic či PHP [4, s. 104].

Již na první pohled lze vidět určitou transparentnost základních příkazů. Ty se dále dělí podle toho, jakou operaci v databázi provádí.

- Jazyk pro definici dat DDL (*Data Definition Language*)
- Jazyk pro manipulaci s daty DML (*Data Manipulation Language*)
- Příkazy pro řízení přístupu DCL (*Data Control Language*)
- Příkazy pro řízení transakcí TCC (*Transaction Control Commands*) [2, s. 29]

Kategorie DLL obsahuje příkazy k základním operacím při vytváření databáze a manipulaci s tabulkami. Řadíme zde CREATE, DROP a ALTER. Přidáme-li k nim ještě

[2] primární klíč se zkracuje jako PK a cizí klíč jako FK, z anglického *Foreign Key*

objekt, máme kompletní a funkční dotaz. Základem pro jakoukoli práci v rámci databáze jsou příkazy CREATE DATABASE a CREATE TABLE. Pokud bychom je přeložili, doslova tím říkáme „vytvoř databázi“ (respektive tabulku). Samotné dotazy jsou tedy intuitivní, což práci velmi usnadňuje.

Do kategorie DML, jak vyplývá z názvu, patří příkazy pro manipulaci s daty. Pod tím si můžeme představit vkládání, mazání, úpravu a výběr dat v databázi, tedy příkazy INSERT, DELETE, UPDATE a SELECT. Jejich syntaxe vypadá následovně:

```
INSERT INTO nazev_tabulky VALUES (hodnota1, hodnota2,…)
```

Abychom si příklad vysvětlili, opět stačí přeložit příkaz do českého jazyka. Dostáváme tak „vlož do tabulky hodnoty“, my už jen musíme určit jaké hodnoty a především do jaké tabulky. Ostatní zmíněné příkazy fungují na podobném principu:

```
DELETE FROM nazev_tabulky WHERE nazev_sloupce = hodnota
```

```
UPDATE nazev_tabulky SET nazev_sloupce = hodnota WHERE nazev_sloupce = hodnota
```

Příkazy UPDATE a DELETE se v syntaxi liší klíčovým slovem SET. Do něj napíšeme upravenou hodnotu. Co mají naopak oba společné je klauzule WHERE, určující záznam, který má být změnou ovlivněn. Pokud by byla část dotazu za podmínkou WHERE vynechána, změna by ovlivnila všechny záznamy [5].

```
SELECT nazev_sloupce FROM nazev_tabulky
```

```
SELECT * FROM nazev_tabulky
```

Výběr požadovaných dat uskutečníme příkazem SELECT. Základní konstrukce dotazu obsahuje výběr sloupců z tabulky, ale můžeme jej i dále větvit a rozvíjet, především podmínkovou strukturou WHERE. Pokud bychom nechtěli zbytečně vypisovat všechny atributy tabulky, stačí zadat symbol hvězdičky, který slouží jako klíčové slovo „vše“[3].

Ve výčtu kategorií jazyka SQL byly zmíněny ještě dvě další skupiny. Vzhledem k tomu, že běžný uživatel se s nimi tolik nesetká a jedná se spíše o pokročilé příkazy, zmíníme je zde jen v krátkosti. První z nich řídí přístup (DCL). Patří mezi ně například CREATE, ALTER a DROP USER. Jejich účelem je administrace databáze z pohledu přístupových práv uživatelů. Druhou skupinou jsou příkazy pro řízení transakcí (TCC):

[3] za předpokladu, že je naším cílem vypsat z tabulky všechny záznamy

SET TRANSACTION, COMMIT nebo ROLLBACK [2, s. 29-30]. Transakcím se budeme věnovat blíže v následující podkapitole.

1.4 Transakce

V řeči databází se dá transakce zjednodušeně popsat jako konzistentní změna stavu. Data se do databáze zapíší pouze v případě, proběhne-li přenos dat úspěšně. Dojde-li k jakémukoli problému mezi výchozím a konečným stavem, databáze se vrátí do původního stavu, v jakém se nacházela před začátkem transakce.

Transakce napomáhají obrovskou mírou k integritě dat. Ta je zajištěna především sadou charakteristických vlastností známé pod zkratkou ACID. Atomicita, tedy nedělitelnost, zajišťuje provedení buď všech, a nebo žádné z operací. Nesmí se stát, že by se v půlce přenosu dat operace přerušila a data zůstala přepsána. Další vlastností je konzistence (z anglického *Consistency*), která má za úkol zajistit, aby stav databáze zůstal konzistentní jak před, tak i po dokončení transakce. Další míru zabezpečení integrity dat přidává izolovanost. Je definována tak, že každá transakce je izolovaná od ostatních a ty se tak nemohou navzájem žádným způsobem ovlivnit. Poslední vlastností zkratky ACID je trvalost (z anglického *Durability*). Jedná se o určitou formu zabezpečení v podobě zálohování v průběhu operace, aby se předešlo ztrátě dat při nedokončení transakce [6].

1.5 Pohledy

Jednou z nejdůležitějších součástí databází jsou pohledy. Jedná se o „virtuální tabulku“ složenou z polí jedné, nebo více tabulek. Slovíčko virtuální zde hraje klíčovou roli, protože na rozdíl od samotných tabulek pohledy neukládají data, ale pouze je zobrazují. To vede ke snížení paměťové náročnosti. „*Kdykoli k pohledu nějakým způsobem přistupujete, databázový systém jej znovu obnoví a naplní daty*.“ [1, s. 300]

Pohled sestává pouze z vybraných polí tabulek a dává nám tedy možnost odizolovat data, jež nechceme uživatelům, či skupinám uživatelů vůbec zobrazit. Tento fakt přináší zvýšení zabezpečení a ochrany dat

Zdaleka nejpodstatnější je možnost využití pohledu složeného z polí více tabulek. Jedinou podmínkou přitom je, aby byla požadovaná data ve vzájemné relaci. V případě zvláště složitých pohledů však může docházet k delší přístupové době k základním tabulkám.

2 NEJPOUŽÍVANĚJŠÍ DATABÁZOVÉ SYSTÉMY

V několika následujících podkapitolách se budeme blíže věnovat vybraným databázovým systémům, které patří mezi nejznámější a nejpopulárnější dostupná řešení. Konkrétně se jedná o MySQL a Oracle Database. Před tím se blíže podíváme na to, jaká kritéria hrají při výběru takového systému roli a proč tomu tak je. Následně probereme do větší hloubky funkcionalitu a architekturu obou klient/server zástupců, což nám pomůže lépe specifikovat a pochopit výhody a nevýhody popisovaných systémů.

2.1 Kritéria výběru DBMS

Vzhledem k tomu, že RDBMS byly vytvořeny již před třiceti lety, začaly tolik rozšířené systémy s relačním modelem narážet na limitace spojené s modernizací počítačových technologií. V dnešním světě je důraz kladen především na co možná nejrychlejší zpracování obrovských objemů dat s co nejmenší přístupovou dobou. RDBMS v tomto ohledu nepatří k těm úplně nejvhodnějším systémům, a tak není divu, že se uvolnil prostor pro nová řešení. Jedním z takovýchto konceptů je NoSQL [7].

NoSQL je zde uveden záměrně, jelikož výborným způsobem shrnuje nedostatky klasických RDBMS, kterými se v této práci zabýváme. V moderní době je zdaleka nejdůležitějším faktorem rychlá odezva a provedení transakcí. Zatímco SQL funguje na bázi relací a striktně tyto vztahy dodržuje, NoSQL se na první pohled může zdát jako chaos. Místo toho, aby byla data uspořádána v tabulce, tak jsou spíše nahromaděna na jednom místě (například JSON dokumentu[4]) a určit vztahy mezi položkami je obtížným úkolem. Navíc zde neexistují žádná fixní pravidla zajišťující referenční integritu; její zajištění závisí pouze na typu projektu jako takovém [8]. Na druhou stranu je mnohem jednodušší upravovat strukturu databáze a to pomocí tzv. denormalizace. To v praxi znamená, že klasické vytváření vztahů pomocí odkazovacích tabulek[5] v normalizované databázi je často spíše nežádoucí, a proto je potřeba použití reverzního postupu. Data, která by normálně byla ve zmíněné odkazovací tabulce, jsou přidána k původnímu záznamu, čímž dochází k redundanci dat. Zároveň ale díky tomu může systém k datům přistupovat daleko rychleji, protože k nim nevedou žádné cesty přes odkazy z všemožných tabulek a systém se nemusí zdržovat náročnými dotazy obsahující klauzuli JOIN [9].

[4] *JavaScript Object Notation* – datový formát vhodný pro přenos informací
[5] za použití primárních a cizích klíčů

Z popsaných vlastností NoSQL můžeme snadno formulovat jeho výhody a nevýhody. Svobodou v oblasti celkového přístupu k záznamům se ztrácí referenční integrita, ale systém jako takový je neuvěřitelně flexibilní a dokáže rychle zpracovávat požadované transakce. Relační databáze se pyšní především spolehlivostí a stabilitou, ke které obrovskou měrou přispívají vlastnosti ACID. Do jisté míry se dá tvrdit i to, že jsou RDBMS přizpůsobivé, ne ale ve stejném smyslu jak je tomu u NoSQL. To je dáno tím, že jsou stále považovány za ověřené funkční řešení.

Dalším výrazným faktorem při výběru databázového systému je, zdali ho bude možné použít na požadované platformě. Přestože mnohá řešení nejsou podporována na úplně všech dostupných platformách, přední RDBMS se s tímto problémem vypořádají bez větších obtíží. Co už s sebou ale nese určitá úskalí, je výběr licence. Zatímco open source projekty obecně nabízí velikou flexibilitu a možnost téměř okamžitého startu, zároveň také přinášejí výrazné nedostatky; a to ve formě technické podpory a dlouhodobé správy [7].

Jako významný hráč vstupuje do hry samozřejmě cena. Je jasné, že za stabilitu a spolehlivost si uživatel připlatí. Ať už zvolíme jakoukoli variantu, vždy je nejdůležitější se první zeptat, jaký je charakter a rozsah projektu a co po něm budeme požadovat. Existuje nepřeberné množství řešení a každé je vhodné použít na jiný typ problému.

2.2 MySQL

2.2.1 Základní popis

„MySQL platí za nejpopulárnější open source databázi SQL. Open source znamená, že software může každý používat a měnit. MySQL si můžete stáhnout z internetu a používat. Každý, koho to zajímá, může studovat zdrojový kód a podle potřeby ho měnit. Pokud chcete MySQL použít pro komerční účely, je potřeba získat verzi s licencí.“ [4, s. 107] K tomuto popisu jednoduše není co dodat.

V současné době je tento systém vyvíjen a vydáván s plnou podporou firmy Oracle Corporation. Přestože ze začátku po odkoupení databázovým gigantem panovaly obavy o budoucnost MySQL, čas ukázal, že to tomuto open source projektu naopak pomohlo v žebříčcích směrem vzhůru [10].

2.2.2 Formát úložiště dat v databázi

Dříve než se vrhneme na architekturu systému, tak musíme probrat problematiku tzv. databázových úložišť[6]. Jedná se o součást RDBMS, která se stará o údržbu dat a práci s nimi. Určuje obecný charakter úložiště ve fyzické vrstvě a dále také to, jakým způsobem budou probíhat základní operace [11]. Zjednodušeně řečeno se jedná o mozek systému práce s daty.

Výběr typu úložiště záleží na tom, na jaký účel bude tabulka využita a jaká data bude obsahovat. Důležité faktory jsou především:

- Podpora transakcí
- Podpora cizích klíčů
- Indexování
- Typ zálohování

Tabulky podporující transakce jsou přirozeně pomalejší, protože musí vykonat větší počet operací. Používají se na ukládání dat, se kterými je často manipulováno. Pro archivaci dat slouží naopak tabulky bez transakcí, které k datům dokáží přistupovat efektivněji [11]. Cizí klíče jsou podporovány u transakčních tabulek proto, aby nemohlo dojít k nechtěné ztrátě integrity dat při vykonávání operací. Vykonávají totiž funkci pojidla mezi tabulkami.

Mezi primárně podporované databázové úložiště patří InnoDB, MyISAM, Memory, CSV, Archive, Blackhole a NDB [12]. Na oficiálních stránkách nalezneme jak stručný popis, tak i přehlednou tabulku s jejich vlastnostmi. V praxi se setkáme hlavně s prvními dvěma zmíněnými typy.

Výchozím enginem pro MySQL je InnoDB, které nahradilo původní MyISAM. Vytvořením nové tabulky v databázi, bez toho, aniž by došlo ke specifikaci typu úložiště klíčovým slovem ENGINE, vybere databáze automaticky právě InnoDB. Hlavními vlastnostmi jsou podpora transakcí, cizích klíčů a uzamykání dat v řádcích. Od verze 5.6.6 je automaticky nastavena vlastnost *file-per-table*, která vytvoří pro každou tabulku zvlášť soubor s koncovkou *ibd*. Dříve byla data ukládána do společného tabulkového prostoru v souboru *ibdata1*. Tímto způsobem lze tabulky v rámci MySQL databáze jednodušeji přesouvat [13]. Celou strukturu souborů všech systémů probereme

[6] z anglického *Storage Engines*

v kapitole 4.4.1. Důležitým faktorem výběru typu tabulkového enginu je jeho velikost a samozřejmě také výkon nad operacemi.

MySQL obsahuje mnoho funkcí pro zálohu a obnovu dat. V případě hardwarového či softwarového problému na straně serveru se datové soubory obnoví do původního stavu. Záleží na tom, jestli se provedené změny přepsaly do datových souborů či nikoliv a od toho se odvíjí, co bude z tzv. *redo logu* obnoveno do původního stavu [14]. Jak vyplývá z jeho názvu, tento soubor má za úkol uchovávat originální data.

Rychlost přístupu k datovým strukturám a indexům je vyřešena cache[7] pamětí zvanou *buffer pool*. Ta funguje na jednoduchém principu; na předních pozicích uchovává ta data, která jsou využívána nejvíce [15]. Dalším krokem k vylepšení je zmíněné uzamykání řádků a úprava čtení při probíhajících transakcích. Výsledkem je potom zlepšení současného přístupu uživatelů. To je důležité zejména u větších aplikací, kde se očekává častý provoz uživatelů.

Tabulky MyISAM byly nahrazeny jako výchozí formát od verze 5.5 [16]. Základním rozdílem oproti InnoDB je to, že nepodporují transakce a cizí klíče. Každá tabulka je fyzicky uložena na disku ve třech souborech. Ty nesou název tabulky a liší se pouze v příponách.

Tabulky je možné komprimovat a zmenšit jejich velikost až o 70% nástrojem *myisampack*. Po zmenšení se tabulky stávají přístupnými pouze pro čtení a jediná možnost, jak data upravovat je tabulku dekomprimovat do původní podoby a proces znovu opakovat [17]. MyISAM se využívají hlavně v databázích s vysokými nároky na čtení záznamů. Mimo již zmíněnou komprimační tabulku podporují ještě statický a dynamický formát.

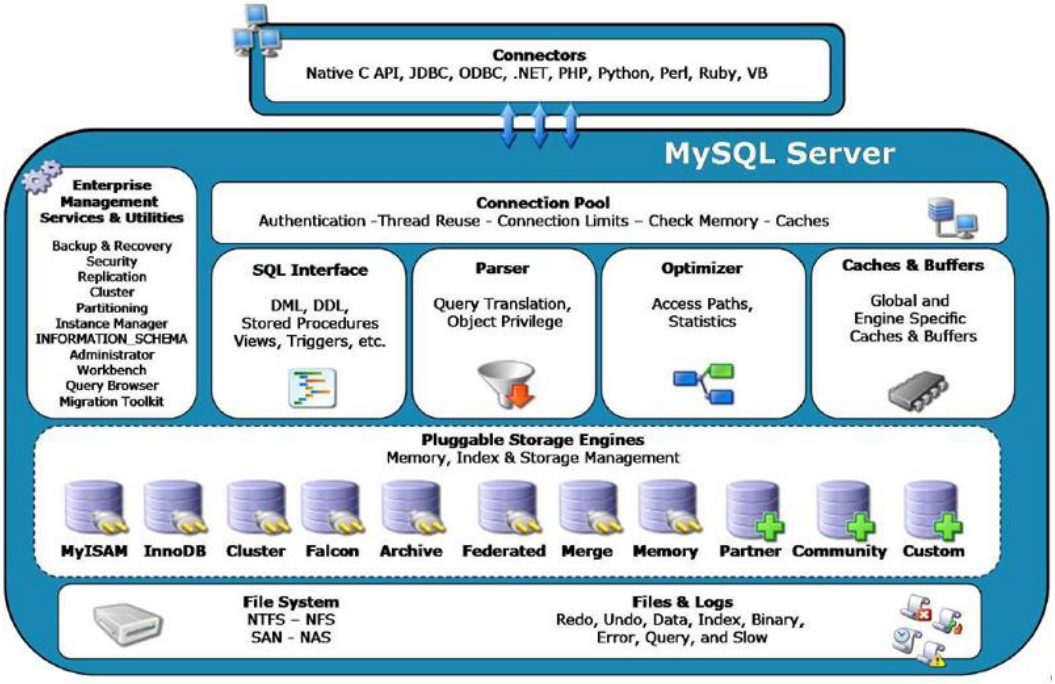
Zásadní rozdíl je jak v rychlosti přístupu k datům a velikosti souborů, tak v možnosti obnovení poškozených dat. Tabulky s dynamickou délkou umožňují zvětšit diskový prostor automaticky dle potřeby. Při přesažení nastaveného limitu se data ukládají po částech do jiných souborů a dochází tak k fragmentaci. Pokud se data jakýmkoli způsobem poškodí a bude je potřeba dát zpět do původní podoby, bude mít systém mnohem složitější práci při hledání ztracených částí [18].

[7] vyrovnávací paměť (také zvaná mezipaměť), která uchovává důležitá data pro snadný přístup

2.2.3 Architektura

Typ databázových tabulek hraje velice důležitou roli u architektury MySQL. Co ji dělá tak unikátní je možnost zapojení více druhů enginů v rámci jedné databáze. V podstatě každá tabulka může mít v případě potřeby jiný typ úložiště. Tím dochází k využití potenciálu správy databáze na maximum.

Vezmeme-li strukturu architektury popořadě, musíme začít tzv. aplikační vrstvou. Jedná se o velmi důležitou pasáž textu celé práce a bude na ni často navazováno. Tato vrstva je u všech klient/server databázových systémů prakticky identická. Jedná se o klasické služby zahrnující připojení k RDBMS, autentizaci, zabezpečení a tak dále. Ke komunikaci s klientem je využíváno jedno vlákno, uložené na straně serveru. Aby k připojení vůbec došlo, je nutné zadat uživatelské jméno, název hostitele a heslo. Poté si databáze v rámci zabezpečení zkontroluje, jaká práva má připojený uživatel v rámci celého systému [19, s. 1].



Obrázek 1 Architektura MySQL [11]

Následuje logická vrstva, která se stará o funkčnost a kompletní správu databázového serveru. Nachází se v ní rozhraní pro komunikaci s SQL, syntaktický analyzátor, optimalizátor a vyrovnávací paměti. Mimo jiné jsou zde uloženy všechny procedury, triggery a pohledy. Do poslední vrstvy patří databázové úložiště, které jsme probrali hned na začátku kapitoly.

K architektuře patří neodmyslitelně současný přístup uživatelů. MySQL řeší tento problém na dvou úrovních. O prvním z nich jsme si řekli v předchozí kapitole. Řeč je o uzamykání řádků v tabulce, technologii poskytované databázovým enginem. Zámky se dělí na dva druhy: sdílený a exkluzivní. U operace čtení se využívá sdílený zámek, aby uživatelé mohli ze stejného řádku číst ve stejný okamžik. Exkluzivní zámek je nadřazený, zablokuje přístup jak ke čtení, tak i pro další zápis. Je to logické, protože hlavním cílem je zpřístupnit zápis pouze jednomu uživateli ve stejný čas, aby mohla být data v pořádku změněna. Problémem je, že neustálé zamykání, kontrola a odemykání ovlivňuje výrazným způsobem výkon databáze. Z tohoto důvodu dává MySQL uživatelům na výběr z různých typů zámků [19, s. 4].

2.2.4 Edice

- MySQL Community Edition: verze ke stažení zdarma, vyvíjena komunitou pod licencí GNU/GPL.
- MySQL Classic Edition: verze dostupná pouze jako již vložená databáze do aplikací, určená pro distributory softwaru. Pracuje s MyISAM tabulkami a je určena pro aplikace zatížené častým čtením dat.
- MySQL Standard Edition: nejlevnější placená verze, určená pro náročný provoz online transakcí.
- MySQL Enterprise Edition: zahrnuje mnoho pokročilých funkcí a nástrojů pro správu. Samozřejmostí je neustálá technická podpora.
- MySQL Cluster Carrier Grade Edition: nejdražší ze všech verzí, určená pro nejnáročnější aplikace jako sociální sítě, komunikační servery a cloudová řešení [20].

2.2.5 Podporované systémy

Systém MySQL běží v současné době na většině dostupných platforem. V seznamu nechybí tituly jako Windows, MAC OS, distribuce Linuxu či Solaris. Kompletní seznam všech podporovaných platforem a jejich architektur lze nalézt v souhrnné tabulce na oficiálních stránkách [21].

2.2.6 Využití

MySQL je nedílnou součástí populárního balíčku internetových služeb LAMP. Ten se skládá primárně z open source aplikací. Mimo MySQL je to operační systém Linux, webový server Apache a poslední písmeno reprezentuje programovací jazyk PHP. Nejuniverzálnějším balíčkem je multiplatformní XAMPP, ve kterém bylo na pozici RDBMS nahrazeno MySQL za svou čistě open source verzi MariaDB.

Mnoho českých i zahraničních hostingů webových služeb nabízí možnost založení databáze právě v systému MySQL. Důvodů je samozřejmě více než jen nulové náklady na provoz. Dalšími faktory jsou popularita a relativně jednoduché a intuitivní rozhraní. V neposlední řadě je jedním z důvodů také integrace s redakčními systémy. MySQL můžeme najít v open source CMS [8] Wordpress, Drupal či Joomla.

Své využití najde tento open source projekt i v komerční sféře. Na svých oficiálních stránkách se pyšní implementací v aplikacích organizací Facebook, Google, Adobe, Alcatel Lucent nebo Zappos [22]. Většina těchto organizací spolupracuje na projektu WebScaleSQL, který se snaží o maximalizaci využití možností nabízených v MySQL a na jeho snazší nasazení na požadovanou platformu [23]. Zároveň je nutné dodat, že tyto gigantické firmy zcela nepochybně využívají více typů databázových systémů v závislosti na charakteru ukládaných dat.

2.2.7 Výhody

Největším kladem je bezesporu open source licence. Na internetu nalezneme spousty návodů a jeho instalace se tak stává otázkou několika desítek minut. Obsahuje spoustu užitečných funkcí pro menší a středně velké aplikace. Navíc spolupracuje se širokou škálou programovacích jazyků. Zajímavou možností je podpora různých druhů tabulkových enginů v rámci jedné databáze. Samozřejmostí je transakční přístup, tvorba pohledů, procedur, triggerů a podpora GIS systémů nebo třeba JSON dokumentů.

2.2.8 Nevýhody

Volná licence je zároveň i prakticky největším mínusovým bodem. Pro plnou podporu si již případní zájemci musí zakoupit placenou edici. Při použití jiného systému pro ukládání než InnoDB, nemusí MySQL fungovat ve svém plném potenciálu. Další nevýhodou jsou limitace uložených procedur a triggerů [24]. Mezi omezené funkce patří i pohledy.

[8] redakční systémy, z anglického *Content Management Systems*

2.3 Oracle Database

2.3.1 Základní popis

Oracle je naprostou špičkou mezi relačními databázovými systémy, což dokládá i jeho popularita. Společně s MySQL se jedná o nejpopulárnější databázový systém vůbec [25]. Obsahuje nepřeberné množství vysoce optimalizovaných funkcí a služeb, které cílí primárně na komerční trh.

Oracle přidává k označení svých verzí ještě písmeno, které reprezentuje oblast zaměření. Verze 8 a 9 obsahovaly v názvu písmeno *i*, indikující zaměření na služby spojené s *internetem*. Oracle *10g* a *11g* představovaly posun k tzv. *grid computingu*, jehož cílem je využití architektury pro více serverů, raději než jednoho výkonného zařízení. Nejnovější verze vyšla v roce 2013 a jedná se o *12c*, směřující veškerou pozornost k využití *cloudových služeb* [26].

2.3.2 Architektura

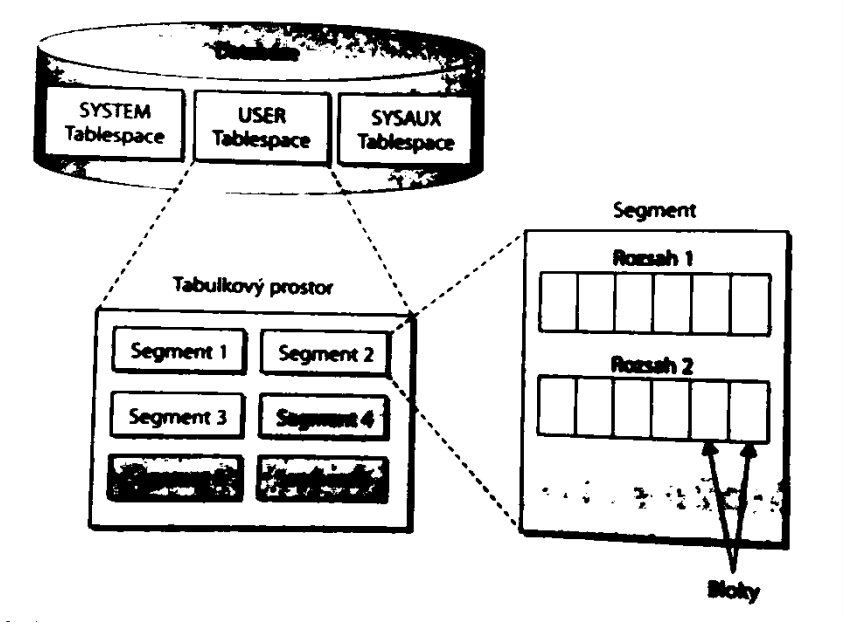
Samotná databáze je u Oracle uložena na pevném disku serveru a sestává z několika databázových souborů. Důležitou entitou je instance, která se skládá ze sdíleného paměťového bloku, uloženého v tzv. systémové globální oblasti[9] a systémových procesů, běžících na pozadí. Úkolem instance je správa databázových souborů [27, s. 27]. Pojmy instance a databáze jsou často zaměňovány, ale konkrétně v případě Oracle je nutné je rozlišovat. Pokud zatím není rozdíl úplně patrný z předešlé definice, postupem času bude jeho význam v práci ještě osvětlen.

Úložné datové struktury se v Oracle dělí na dva typy. První z nich jsou fyzické struktury, které se starají o uložení a správu dat. Tyto soubory se vytvoří společně se zadáním příkazu CREATE DATABASE.

- Datové soubory: obsahují veškerá data o logické struktuře databáze, jako například tabulky a indexy.
- Řídící soubory: obsahují tzv. metadata, tedy upřesňující informace o fyzické struktuře databáze. Najdeme zde informace o názvu databáze, umístění souborů a také nastavení typu zálohování.
- Redo log: uchovává data o provedených změnách, minimálně do dvou souborů [27, s. 48-49].

[9] zkráceně SGA, z anglického *System Global Area*

Druhým typem úložných struktur jsou logické struktury. Celá databáze je rozdělena do tabulkových prostorů, obsahující datové soubory fyzické struktury. V tabulkových prostorech jsou uchovávány bloky dat tvořící segmenty, které dále reprezentují databázové objekty, jako jsou třeba tabulky nebo indexy [27, s. 28-29].



Obrázek 2 Logické úložné prostory Oracle [27, s. 28]

Ke správě databáze slouží mimo SGA také procesy. Jakmile se uživatel připojí k databázi, spustí se instance a dochází ke spuštění dalších procesů. Ty se dále dělí dle toho, k jakému účelu slouží. Klientské a serverové procesy se starají o spravování připojené aplikace a o komunikaci s klientem. Na zbylé úkoly dohlíží procesy na pozadí [28].

Oracle nabízí mnoho funkcí pro lepší a rychlejší přístup k datům. *„Technologie Real Application Cluster (RAC) umožňuje více instancím na oddělených serverech přístup ke stejným databázovým souborům.“* [27, s. 66] Architektura rozdělení správy operací a přístupu na základě požadavků databázového systému je podstatou *grid computing* technologie [29], hlavní funkcí Oracle ve verzích 10g a 11g. Hlavním úkolem je rovnoměrné rozdělení úkolů mezi více strojů a jejich spolupráce k dosažení rychlejšího výsledku – v rámci databází lepší zpracování, správa a přístupnost dat. Dochází tak k oproštění se od „ultra“ výkonných a drahých zařízení.

Od verze *12c* přichází Oracle s funkcí *pluggable database* pod záštitou tzv. *multitenant* architektury. Ve zkratce se jedná o odlehčení určitých částí databáze do

přenositelné databáze (*Portable Database*, PDB). Veškerá funkční část zůstává v tzv. kontejneru (*Container Database,* CDB), tedy něčeho, co na první pohled pro Oracle rozhraní vypadá jako kompletní databáze. Díky tomu je možné spojit databázové struktury dohromady, bez toho aniž by se navzájem ovlivňovaly a sdílely svá data. Dochází k ušetření hardwarových prostředků, ke zrychlení přístupu a usnadnění celkové správy. Další nespornou výhodou je oblast zálohování, protože stačí zálohovat pouze CBD [30].

Současný přístup více uživatelů řeší Oracle pomocí transakcí a uzamykání objektů. Podporuje tzv. *multiversion consistency,* model s návratovou technologií pro konzistentní čtení záznamů. Pokud provádí příkaz jakékoli změny dat, jsou původní data zaznamenána do návratového segmentu a kdykoli připravena na obnovu do počátečního stavu transakce [27, s 237]. Oracle podporuje tři transakční izolační stupně ochrany, přičemž výchozím stupněm je *read-committed*. Dalšími úrovněmi jsou *serializable* a *read-only*, poskytující větší ochranu proti zobrazení neprovedených operací nad daty [31].

Pro lepší správu dat Oracle inovoval i další oblasti. PL/SQL přináší procedurální přístup do strukturovaného jazyka SQL. Skládá se z bloků, dělících se na tři sekce: deklarace proměnných, výkonná sekce a oblast pro ošetření výjimek. Je do něj možné aplikovat pouze některé z příkazů SQL. Samotné procedury a funkce jsou uloženy v datovém slovníku, okamžitě dostupné pro více uživatelů naráz. Volání procedur je mnohem rychlejší, než standardní SQL příkazy. PL/SQL má sdílené jádro s SQL, tedy přístup ke všem novinkám implementovaných do SQL [27, s. 46].

Verze *12c* se nese ve znamení *cloudových služeb*. Oracle nabízí své služby v rámci softwaru jako služby SaaS (*Software as a Service*), platformy jako služby PaaS (*Platform as a Service*) a infrastruktury jako služby IaaS (*Infrastructure as a Service*), souhrnně nazývanými *cloud computing*. Uživatelé platí za poskytování přístupu k software či hardware v cloudovém řešení Oracle. Mezi těmito funkcemi nalezneme Oracle Database Cloud Service. Jedná se o nasazení databáze na cloudové servery, s plnou podporou a optimalizací Oracle [32].

2.3.3 Edice

- Oracle Database Enterprise Edition: nejlepší a nejdražší dostupná verze. Nemá žádné hardwarové omezení. Podporuje všechny dostupné funkce

Oracle, spousta z nich je však ve formě přídavných balíčků za další poplatek. Jednou z nich je i *multitenant* architektura. Enterprise edice je plně škálovatelná a vhodná především pro obrovské projekty.

- Oracle Database Standard Edition 2: levnější varianta, určená pro menší a středně velké aplikace. Podporuje méně funkcí, což nabízí o něco jednodušší správu. Do verze *12c* Oracle nabízel ještě edice *Standard Edition* a *Standard Edition 1*. Všechny zmíněné varianty se lišily hlavně v omezení licencované instalace na stanovený počet CPU socketů[10]. SE2 je možné nainstalovat na systémy maximálně se dvěma sockety.
- Oracle Database Express Edition: verze zdarma, podléhající mnoha omezením. Expres edice dovoluje provádět operace pouze na jednom CPU na serveru a využít jen 1GB operační paměti. Maximální velikost dat je 11GB. Podobně jako u open source projektů, Oracle u expres edice nenabízí plnou technickou podporu [33].

2.3.4 Podporované systémy

Oracle lze nainstalovat na operační systémy Windows, Mac OS X, Solaris, HP-UX, AIX a Linux. Oficiální podporované distribuce Linuxu jsou Red Hat, Oracle Linux 7 a SUSE [34].

2.3.5 Využití

Primární trh Oracle jsou komerční produkty. Hlavní výsadou tohoto databázového systému je škálovatelnost, vlastnost zaměřená na velké a expandující firmy. Systém podporuje nepřeberné množství optimalizovaných funkcí a technologií pro lepší správu a chod zatíženého systému. Zmínil bych především technologii RAC a *multitenant* architekturu, shrnuté v podkapitole 2.3.3.

Oracle je vhodné použít jak pro velké databáze VLDB, tak pro online transakční systémy[11]. Oba spojuje funkce *partitioning*, která dělí databázové objekty na menší části, čímž dává prostor rychlejšímu a snadnějšímu přístupu k těmto objektům [35].

[10] patice procesoru
[11] VLDB = *Very Large Database*, OLTP = *Online Transaction Processing*

2.3.6 Výhody

Oracle v podstatě udává, jakým směrem se bude trh ubírat. V každé z verzí se zaměřuje na určitou vlastnost, u které vylepšuje funkcionalitu. Nespornou výhodou je technická podpora a bohatá funkcionalita. Škálovatelnost pro něj není nejmenší problém a je proto vhodný pro velké aplikace. Plusem je expresní edice na odzkoušení systému.

2.3.7 Nevýhody

Limitace v rámci jednotlivých edic je velmi znatelná. Neomezené možnosti hardware má bohužel jen verze Enterprise, jejíž cena je velmi vysoká. V neprospěch Oracle hraje také celková náročnost správy.

3 SQLITE

3.1 Vložená databáze

Systém SQLite je v pravém smyslu slova spíše knihovna, než databázový systém. Nejedná se totiž o samostatný program, ale o vloženou databázi přímo do aplikace. Jeho použití spadá pod licenci *Public Domain* a je tedy kompletně zdarma i v komerčních aplikacích. Jedinou výjimkou jsou tzv. rozšíření, která jsou za poplatek [36].

3.2 Historie

Nápad vložené databáze dostal autor projektu, Richard Hipp, v roce 2000. Původní myšlenkou byla odlehčená a přenositelná databáze, která by byla jednoduchá na správu a nemusela se složitě instalovat.

První aktualizací prošel změnou správce dat. Volně dostupné GDBM[12], umožňující nasazení bez instalace, nahradil komplexnější algoritmus B-Strom.

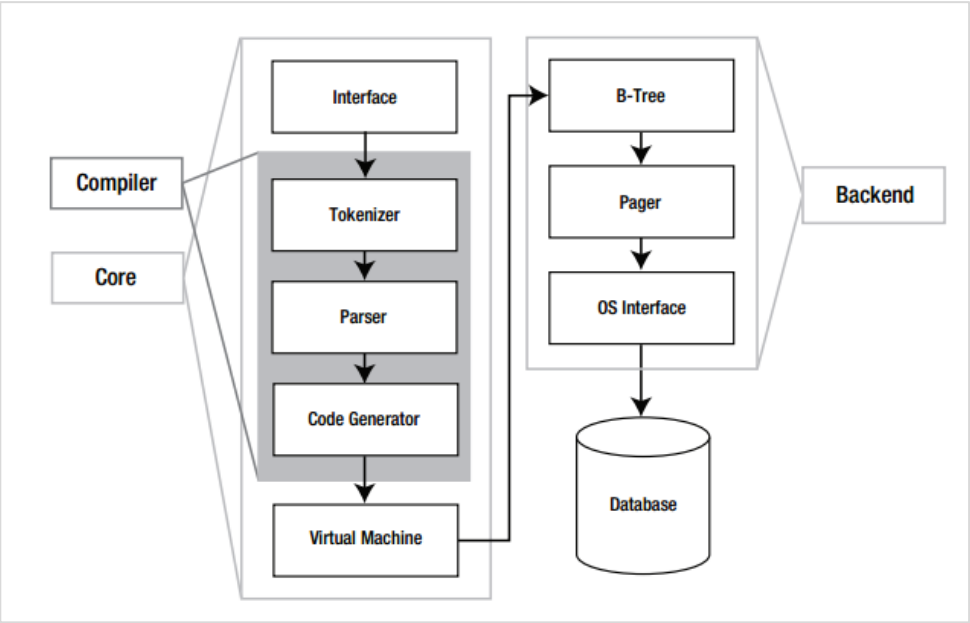
Významný update přišel v roce 2004 na verzi 3. Ta přinesla spoustu vylepšení jako třeba podporu kódování UTF-8 a 16, kompaktnější databázový soubor, podporu datového typu BLOB[13] nebo třeba zlepšení současného přístupu uživatelů [37, s. 4].

V roce 2012 přišla verze *sqlite4*. Její účel není nahradit *sqlite3*, funguje spíše jako její alternativa. Přidává nové možnosti práce s běhovým prostředím, zjednodušené řazení dat v souboru a také dává větší váhu primárnímu klíči [38].

3.3 Architektura

Celková architektura sestává z osmi modulů seskupených do třech subsystémů. Vrchní část se stará o kompilaci kódu, ve střední části jsou příkazy provedeny a spodní část se stará o ukládání dat a komunikaci s operačním systémem.

[12] GNU *Database Manager*
[13] *Binary Large Object* – ukládá data bez jasně určeného typu, nejčastěji soubory



Obrázek 3 Architektura SQLite [37, s. 5]

Vrchní část je na obrázku označena jako jádro (core). Prvním z modulů je rozhraní sloužící ke komunikaci se systémem. Pod ním se nachází překladač, zodpovědný za zpracování kódu.

Do střední části patří virtuální databázový stroj[14], který funguje jako oddělená jednotka a zajišťuje tak nezávislost na operačním systému nebo použité architektuře aplikace. Jeho primárním úkolem je zpracování dat.

Poslední částí je tzv. *back-end*, v informačních technologiích obecně známý jako „dění za oponou“. B-strom má na starosti organizaci dat. Uchovává je ve stromové struktuře, optimalizované pro vyhledávání. Pager mezitím rozhoduje, které ze stránek s daty budou uchovány v cache paměti pro rychlejší přístup. Dále jsou v kompetenci pageru transakce, uzamykání řádků a obnova dat při pádu systému [37, s. 5-7].

3.3.1 Transakce

SQLite patří mezi RDBMS, což znamená podporu vlastností ACID. V průběhu transakce je možné data číst, ale zapisovat může současně pouze jeden uživatel. Systém při operaci zápisu uzamkne celou databázi. Předtím než se data přepíšou, dojde k vytvoření souboru se zálohou zvaného *rollback journal* a zkopíruje do něj celý obsah databáze. Poté se uzamkne databáze i pro čtení a změny se přepíší. Následně je z disku smazán *rollback journal* a databáze je znovu přístupná všem uživatelům [39].

[14] VDBE, *Virtual Database Engine*

Alternativou je použití *write-ahead logu* (WAL). Ten funguje přesně na opačném principu, než *rollback journal*. Data jsou při transakci modifikována ve WAL, zatímco původní databáze zůstává nezměněna. Při čtení se nejdříve zkontroluje poslední provedená změna ve WAL. Pokud se požadovaná stránka ve WAL nenachází, její data ještě nebyla upravena a jsou přečtena z původní databáze. Ke zrychlení přístupu slouží wal-index, usnadňující vyhledávání. Nachází se ve sdílené paměti, což znamená, že využít rychlého vyhledávání lze pouze na stejném stroji [37, s. 319].

3.3.2 Souborový systém

Celá databáze a vše s ní spojené je uloženo pouze v jednom souboru. Mimo to je její součástí ještě soubor se záložními daty *rollback journal*, popřípadě *write-ahead log*. DB soubor se skládá ze stránek:

- Hlavička: Prvních 100B stránky je určeno pro informace o databázi.
- Lock-Byte: Stránka určená pro zpětnou kompatibilitu se staršími OS.
- Freelist: Sdružuje nevyužité stránky databáze. Více níže.
- B-strom: Algoritmus starající se o ukládání a organizaci dat do stránek.
- Stránky s přesahem: Spojují přesahující data stránek.

Stránky mají fixní velikost od 512B do 65535B. Maximální velikost DB souboru je 140TB. Standardně se velikost pohybuje v řádech GB [39].

SQLite nabízí možnost spojit databáze v rámci jednoho připojení příkazem:

```
ATTACH DATABASE filename AS database_name;
```

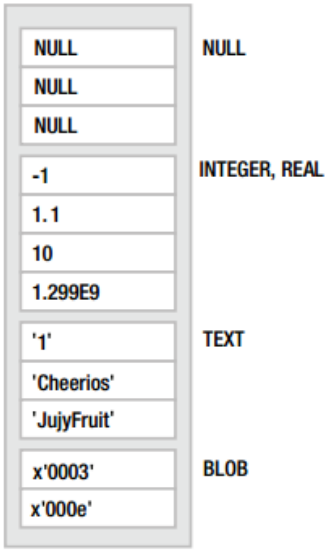
Dovoluje tak pracovat s daty obou databází. Transakce v rámci ATTACH jsou atomické, změny proběhnou buď ve všech databázích, nebo ani v jedné. To neplatí, pokud se při spojení používá WAL namísto *rollback journal*. Tehdy je možné, že se data přepíší pouze v jedné z DB [40].

Zajímavé je přidělování diskového prostoru pro databázi. Smazáním dat nedojde ke zmenšení databázového souboru. Místo toho je tato část disku připravena pro další zápis v části zvané *freelist*. Pokud chceme prostor vrátit zpět operačnímu systému, je nutné použít příkaz VACUUM. Tato funkce není podporována při použití WAL databáze [41].

3.3.3 Datové úložné třídy

Datové typy v SQLite fungují na odlišném principu než u klasických RDBMS. Na disku jsou data uložena pod známými datovými typy, ale při zpracovávání je s nimi nakládáno pod zjednodušenými úložnými třídami. Patří mezi ně TEXT, NUMERIC, INTEGER, REAL a BLOB.

Kvůli kompatibilitě s ostatními systémy obsahuje každá z tříd standardní datové typy. Například INTEGER obsahuje INT, TINYINT, BIGINT a další standardní číselné typy [42]. Kompletní seznam je uveden v přílohách. SQLite umožňuje porovnání i v rámci různých úložných tříd. Pro tyto účely slouží následující reprezentace pravidel na obrázku 4.



Obrázek 4 Pořadí při porovnání úložných tříd SQLite [37, s. 104]

3.3.4 Výkon

Vzhledem k architektuře je zpracování základních operací velmi rychlé. Na rozdíl od klasických RDBMS se nemusí vypořádávat se síťovou komunikací se serverem a řešit složitou autentizaci. Také musí provést méně operací k tomu, aby došlo k samotnému výpočtu dotazu či transakce. Problém nastává až v případě složitých dotazů, kde hraje největší roli optimalizace. Potom už na „odlehčení“ tolik nezáleží, klient/server databáze v takovém případě poskytnou lepší výkon [37, s. 11].

3.3.5 Limitace

Systém SQLite je navržen k co nejjednodušší implementaci a nemá žádná hardwarová omezení. Při testování horních hranic systému běžnými uživateli se přišlo na to, že docházelo k nestandardnímu chování, a tak je v současné době každá verze v tomto ohledu před vydáním řádně testována. Na základě testů má SQLite navrženy výchozí hranice. V případě potřeby mají uživatelé možnost měnit vlastnosti systému přes rozhraní *sqlite3_limit()* [43].

Ve většině případů jsme zvyklí spíše na opačnou situaci, kdy systém limituje uživatele. SQLite není výjimkou, nepodporuje hned několik funkcí jazyka SQL.

- FULL a RIGHT OUTER JOIN
- ALTER TABLE (podporuje RENAME TABLE a ADD COLUMN)
- Triggery podporují jen FOR EACH ROW
- Pohledy jsou pouze pro čtení
- GRANT a REVOKE [44]

V dalších kapitolách si vysvětlíme, že některá omezení jdou buď snadno nahradit, anebo jsou z knihovny funkcí vynechány záměrně. Například příkazy GRANT a REVOKE jsou naprosto zbytečné, protože přístupová práva jsou řízena samotnou aplikací.

Problémy mohou nastat také v případě cizích klíčů, protože explicitně neexistuje žádné omezení pro porušení integrity vztahů. Pro vynucení pravidel vztahů zahrnuje *sqlite3* funkci *foreign key constraints*, která je ale ve výchozím stavu vypnutá [45]. Názorná ukázka chování cizích klíčů je součástí kapitoly 4.6.

Problematika klíčů se týká i těch primárních. SQLite v databázi vyhledává na základě automaticky vytvořeného indexu zvaného *rowid*, který funguje stejně jako primární klíč. Vyhledávání je díky tomu mnohem rychlejší, než by bylo v případě klasicky definovaného PK. Na druhou stranu je jeho použití v podstatě jen kosmetické. V případě, že použijeme pro definování PK datový typ INTEGER, stává se automaticky aliasem[15] indexu *rowid*. Jinak zůstává obyčejným sloupcem v tabulce [46].

[15] odkazem

Databázi SQLite není vhodné využívat pro VLDB nebo OLTP hned ze dvou důvodů. Prvním je současný přístup uživatelů, zmíněný v podkapitole o transakcích u SQlite. Zatímco všichni připojení uživatelé mohou data číst, zapisovat může současně jen jeden. Zámek platí na úplně celou databázi.

Druhým problémem je dostupnost na internetu přes protokol NFS (*network filesystem*). V případě přístupu mnoha klientů přímo k databázi se projeví vyšší odezva a tím pádem i nižší výkon [37, s. 11-12]. Zmíněné vlastnosti vyplývají z designu celého systému, který byl navržen jako samostatná a odlehčená databáze. SQLite najde lepší využití v jiném typu aplikací.

3.4 Podporované platformy

SQLite je podporováno na operačních systémech Windows, Mac OS X, BSD, Linuxu, Solaris, HP-UX, AIX a na vestavěných platformách jako Windows CE, Symbian, Palm OS, QNX či VxWorks. Velkou výhodou SQLite je přenositelnost. To v praxi znamená, že databáze vytvořená na Linuxu bude bez problémů a jakéhokoli nastavení fungovat také na ostatních OS [47].

3.5 Využití

„SQLite se nedá přímo srovnávat s klasickými klient/server systémy jako MySQL, Oracle, PostgreSQL nebo SQL Server, protože je určen k jiným účelům.“ [48] Systém se hodí pro malé a středně velké aplikace, k čemuž je také optimalizován.

Své místo si najde zejména v mobilních zařízeních, operačních systémech a internetových prohlížečích. SQLite můžeme najít téměř v každém zařízení s Andoid, iOS, Mac a Windows 10. Dále je součástí webových prohlížečů Firefox, Google Chrome a Safari. Jeho služby využívá i Dropbox, Skype a iTunes [49].

Důvodem, proč je tak rozšířený, je variabilita použití. Hlavním rozdílem oproti klasickým klient/server systémům je, že server i klient běží ve stejném procesu. Vše potřebné pro běh relační databáze je vloženo přímo do kódu programu společně s tímto „odlehčeným“ systémem splňujícím všechny základní předpoklady RDBMS. SQLite často slouží jako vyrovnávací paměť s konfiguračními daty aplikací. Na webu se stará například o uchování dat internetových relací. Samozřejmě jej lze využít i pro ukládání dat menších webů, kde nepočítáme s příliš velkým současným přístupem uživatelů.

Zjednodušeně řečeno se SQLite hodí všude, kde je použití DBMS jako Oracle či MySQL až příliš a kde postačí jednodušší řešení. [37, s. 1-2]

Do jisté míry lze využít databázi SQLite jako aplikační formát. V jeho prospěch hrají vlastnosti jako snadný přístup, minimální nároky, současný přístup více uživatelů nebo to, že je uložen v jednom souboru a přístupný na mnoha platformách. Schopnost uložit a uchovávat data různého typu na jednom místě umožňuje datový typ BLOB [50].

3.6 Přehled rozdílů s klient/server databázemi

	SQLite	MySQL	Oracle
Licence	• Public Domain	• Open Source	• Komerční
Využití	• Aplikace na stejném zařízení jako DB • Nenahrazuje klasické klient/server systémy	• Menší a střední aplikace klient/server	• Velké aplikace • VLDB • OLTP
Přístup	• Databáze vložena do aplikace • Přístup závislý na aplikaci	• Spojení s uživatelem pomocí vlákna	• Spojení pomocí instancí • Řízen serverovými procesy
Správa dat	• Data uložena ve stránkách v jediném souboru • Správa pomocí algoritmu B-strom	• Pluggable Storage Engine • Řídí celou správu dat	• Data uložena v tabulkovém prostoru • Logické a fyzické datové struktury
Datové typy	• Úložné třídy	• Standardní datové typy	• Nestandardní • Mapování[16]
SQL jazyk	• SQLite dot příkazy	• SQL	• PL/SQL
Obnova dat	• Rollback journal • WAL	• Redo log • Undo log	• Redo log • Undo log

[16] změna datového typu na odpovídající typ v Oracle

Současný přístup uživatelů	• V případě zápisu zámek na celou databázi • Nevhodné při časté manipulaci	• Zámky na úrovni řádků • Negativně ovlivní výkon • Možnost výběru zámku	• Multiversion consistency model • Automatické zámky • Izolační úrovně transakcí
Další	• PK/rowid • FK constraints	• Komerční použití za poplatek	• Pluggable Database PDB

Tabulka 2 Porovnání důležitých aspektů SQLite s MySQL a Oracle

4 POROVNÁNÍ V PRAXI

V následujících podkapitolách si vyzkoušíme reálné použití systémů. Podíváme se zblízka na problematiku probíranou v předchozích kapitolách. Nejdříve se seznámíme s nástroji, se kterými budeme pracovat a jak probíhá implementace databáze na jednotlivé systémy. Poté se podíváme na strukturu souborů, ve kterých jsou data uložena. Shrneme tak architekturu, ze které dále vyplývají vlastnosti týkající se využití RDBMS.

Na závěr si v praxi vyzkoušíme vybrané funkční vlastnosti a otestujeme výkon. Zátěžový test bude probíhat v rámci základních operací databázových systémů – čtení, zápis, modifikace a mazání záznamů, známé pod zkratkou CRUD[17].

4.1 Databáze

Na jednotlivé systémy budeme implementovat jednoduchou databázi s tématem MotoGP, jejíž diagram je uveden v příloze 1. Požadovaným výstupem jsou výsledky závodu z tabulky „vysledky“. Před tím je potřeba vyplnit databázi údaji o motocyklech, jezdcích a v jakých týmech jezdí. Dalším krokem je vyplnění okruhů a velkých cen. Informace jsou spojeny v tabulce „zavod“, kde určíme, který ze závodníků se zúčastnil jednotlivých velkých cen. Pak už jen zbývá vyplnit, kdo se jak umístil v tabulce „vysledky“.

Databázový diagram je vytvořen v aplikaci *Enterprise Architect*. Velkou výhodou je funkce generování kódu SQL zadaných tabulek a to i se specifickou syntaxí v rámci RDBMS. V praxi se tedy vytvoří i cizí klíče, které budou testovány na nepovolené vkládání neexistujících hodnot rodičovské tabulky.

4.2 Instalace systému

4.2.1 MySQL

Ke správě databázového systému MySQL budeme používat oficiální nástroj *MySQL Workbench*, konkrétně ve verzi 6.3. Jedná se o grafické prostředí obsahující mnoho pokročilých funkcí pro správu DB. Nejdříve si stáhneme kompletní balíček služeb *MySQL Installer*, který mimo ostatní aplikace obsahuje jak *MySQL Server*, tak i

[17] zkratka, která vyjadřuje základní operace databází Create, Read, Update a Delete

samotný *Workbench*[18]. Instalaci a nastavení serveru zde nebudeme příliš rozvádět, kompletní návod lze najít na oficiálních stránkách[19].

4.2.2 Oracle

Abychom se dostali k instalačnímu souboru systému *Oracle 11g Express Edition*, je nutné se nejdříve na oficiálních stránkách zaregistrovat[20]. Instalační soubor je stejně jako v případě MySQL relativně velký. Navíc je edice XE hardwarově omezena, jak jsme si řekli v kapitole 2.3.4. Ke správě databáze využijeme grafické rozhraní *SQL Developer Tool* ve verzi 4.2, dostupné na oficiálních stránkách[21].

Vytvoření nového uživatele lze provést v nástroji *SQL Developer* a nebo k tomu můžeme využít SQL konzole.

```
CREATE USER jmeno IDENTIFIED BY <heslo>;
```

Při vytváření uživatele je nutné zadat i jeho práva v databázi. Už při instalaci vidíme kontrastní rozdíl oproti SQLite, který příkaz *GRANT* nepodporuje.

```
SQL> grant CREATE SESSION, ALTER SESSION, CREATE DATABASE
LINK, CREATE MATERIALIZED VIEW, CREATE PROCEDURE, CREATE PUBLIC
SYNONYM, CREATE ROLE, CREATE SEQUENCE, CREATE SYNONYM, CREATE
TABLE, CREATE TRIGGER, CREATE TYPE, CREATE VIEW, UNLIMITED
TABLESPACE - to jmeno;
```

I přesto, že se jedná o nejodlehčenější edici Oracle, samotná konfigurace je poměrně náročná. K našim účelům se snažíme implementovat opravdu pouze jednoduchou databázi. Na druhou stranu jak *MySQL Workbench* tak i *SQL Developer* obsahují mnoho pokročilých funkcí a možností správy databáze.

4.2.3 SQLite

Instalační archiv aktuální verze 3.18 nalezneme na oficiálních stránkách v sekci download. Ke správné funkčnosti na systému Windows je nutné stáhnout:

- *sqlite-dll-win32-x86-3180000.zip*
- *sqlite-tools-win32-x86-3180000.zip*

[18] https://dev.mysql.com/downloads/installer/
[19] https://dev.mysql.com/doc/refman/5.7/en/introduction.html
[20] http://www.oracle.com/technetwork/database/database-technologies/express-edition/overview/index.html
[21] http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html

V prvním archivu se nacházejí dva soubory obsahující všechny funkce SQLite. Ve druhém archivu nalezneme příkazový řádek pro práci s SQLite a dva volitelné ladící programy. Jedním z nich je *Analyzer*, který měří efektivitu využití místa databází. Druhý je *sqldiff*, který porovnává vlastnosti vytvořených databází.

Dalším krokem je vytvořit složky v systému pro snadný přístup a do něj rozbalit obsah stažených archivů. Poté spustíme příkazový řádek a ověříme si funkčnost balíčku. V něm nastavíme umístění do složky s SQLite soubory a zkontrolujeme funkčnost následujícím příkazem:

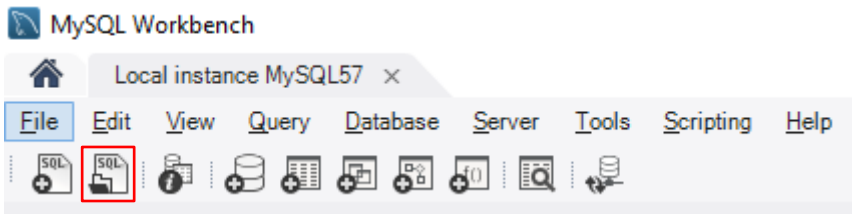
```
C:\sqlite>sqlite3
```

Pokud jsme vše provedli správně, vypíše se nám aktuální verze příkaz pro nápovědu a další informace. K práci se systémem budeme využívat grafické rozhraní, konkrétně open-source nástroj *SQLiteStudio*[22].

4.3 Implementace databáze

4.3.1 MySQL a Oracle

Implementace je na obou systémech téměř identická, a tak ji nastíním v jedné kapitole. Díky tomu, že již máme vytvořený databázový diagram, stačí vygenerovaný SQL kód importovat do naší databáze. Tu v *MySQL Workbench* vytvoříme jako tzv. „schema“, pojmenování přejaté od Oracle. K implementaci stačí kliknout na položku „Open a SQL script“ a vybrat soubor s kódem.



Obrázek 5 Import databáze v MySQL Workbench

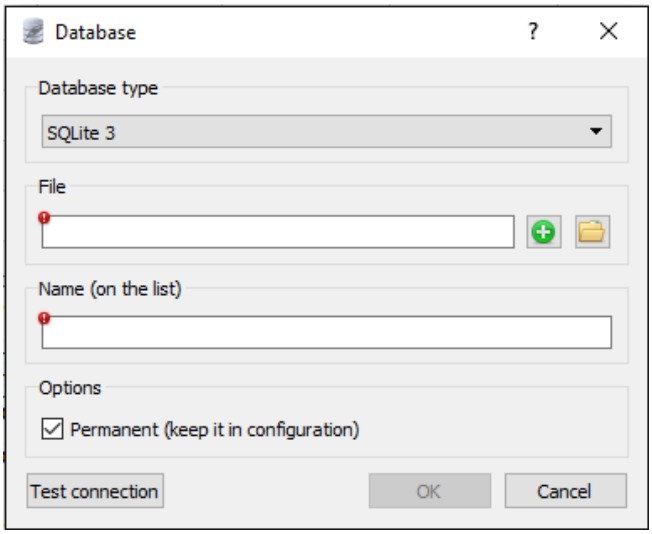
Postup u *SQL Developer* je prakticky stejný, jako v MySQL. Importovat soubor lze v záložce hlavní kontextové nabídky „File“, kde vybereme možnost „Open“. Pokud máme v databázi uložená data, nejlehčím způsobem přenesení je pomocí CSV souboru.

[22] https://sqlitestudio.pl/index.rvt

V nástroji *Workbench* rozklikneme databázi, vybereme požadovanou tabulku, pravým tlačítkem zvolíme možnost „Table data export wizard“ a průvodcem vytvoříme CSV soubor. V *SQL Developer* postupujeme následovně. Nejdříve si zvolíme databázi a tabulku, stejně jako u MySQL. Kliknutím pravého tlačítka vybereme možnost „Import data“ a spustí se průvodce vložení dat z externího souboru.

4.3.2 SQLite

Databázi založíme jednoduše tlačítkem pro přidání databáze. V okně vyplníme typ, fyzické místo uložení a název. Důležitou položkou je „permanent“. Ta určuje, jestli bude databáze permanentní nebo se po restartu SQLite ihned vymaže.



Obrázek 6 SQLiteStudio založení databáze

Pro importování databáze je nejdříve potřeba v *SQLiteStudiu* nastavit zpracování více než jednoho příkazu najednou. To uděláme v záložce „Tools“, „Open cofniguration dialogue“ a v menu „general“ odškrtneme položku „Execute only the query under the cursor“.

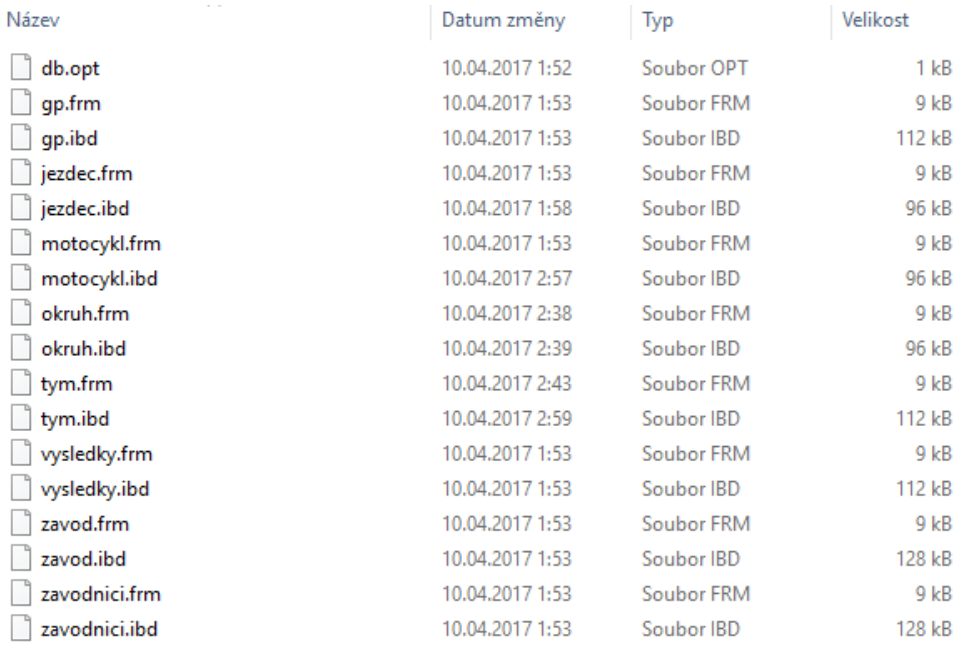
Importovat kód lze v liště pod hlavní nabídkou možností „Open SQL Editor“, kde vybereme „Load SQL from file“. V zásadě se jedná pouze o načtení kódu ze souboru.

4.4 Datová struktura

4.4.1 MySQL

K datové struktuře databáze MySQL serveru se dostaneme následující cestou[23]:

C:\ProgramData\MySQL\MySQL Server 5.7\data\nazevdb



Název	Datum změny	Typ	Velikost
db.opt	10.04.2017 1:52	Soubor OPT	1 kB
gp.frm	10.04.2017 1:53	Soubor FRM	9 kB
gp.ibd	10.04.2017 1:53	Soubor IBD	112 kB
jezdec.frm	10.04.2017 1:53	Soubor FRM	9 kB
jezdec.ibd	10.04.2017 1:58	Soubor IBD	96 kB
motocykl.frm	10.04.2017 1:53	Soubor FRM	9 kB
motocykl.ibd	10.04.2017 2:57	Soubor IBD	96 kB
okruh.frm	10.04.2017 2:38	Soubor FRM	9 kB
okruh.ibd	10.04.2017 2:39	Soubor IBD	96 kB
tym.frm	10.04.2017 2:43	Soubor FRM	9 kB
tym.ibd	10.04.2017 2:59	Soubor IBD	112 kB
vysledky.frm	10.04.2017 1:53	Soubor FRM	9 kB
vysledky.ibd	10.04.2017 1:53	Soubor IBD	112 kB
zavod.frm	10.04.2017 1:53	Soubor FRM	9 kB
zavod.ibd	10.04.2017 1:53	Soubor IBD	128 kB
zavodnici.frm	10.04.2017 1:53	Soubor FRM	9 kB
zavodnici.ibd	10.04.2017 1:53	Soubor IBD	128 kB

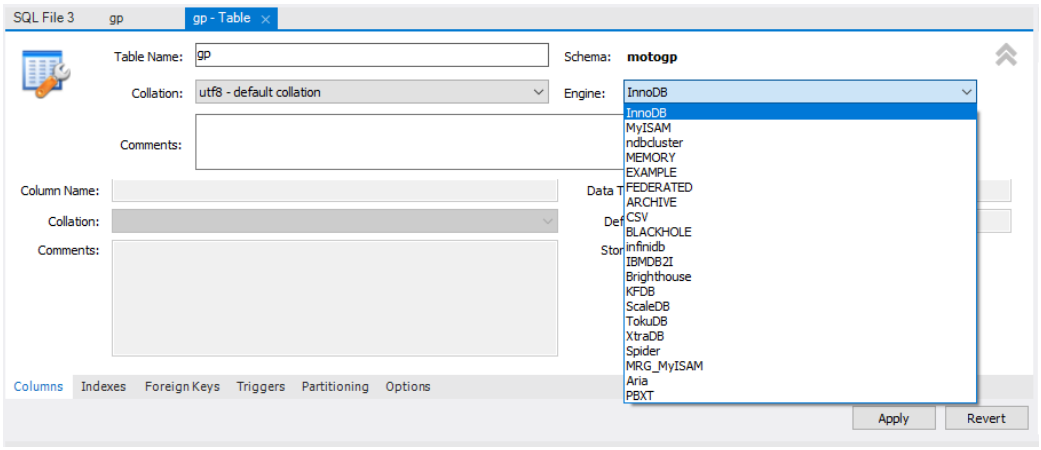
Obrázek 7 Struktura databáze MySQL

Prvním souborem struktury je *db.opt*, kde se nachází veškerá data o databázi. Dále zde nalezneme soubory tabulek InnoDB. MySQL má od verze 5.6.6. ve výchozím stavu zapnutou funkci *InnoDB file-per-table*. Ta umožňuje data ukládat do dvou souborů přímo svázaných s tabulkami, namísto jednoho tabulkového prostoru *ibdata1*. Zásadní rozdíl spočívá ve využití diskového prostoru po operacích TRUNCATE nebo DROP. V případě zapnutého *file-per-table* je možné přiřadit uvolněné místo zpět operačnímu systému. Pokud by byla funkce vypnutá, diskový prostor by zůstal alokovaný pro tabulkový prostor v *ibdata* a k dispozici by byl znovu pouze pro nová InnoDB data [13]. Funkčnost je tedy velmi podobná jako v SQLite, kde je k přidělování diskového prostoru využívaná stránka *freelist*.

Se strukturou souvisí i změna databázového enginu. Stačí si kurzorem najet na požadovanou tabulku a vybrat ikonku „klíče“. Ve sloupci „Engine“ můžeme měnit typ

[23] je-li balíček se serverem nainstalovaný do výchozí složky

úložiště. Pokud bychom vybrali typ MyISAM, došlo by k nahrazení souboru *ibd* tabulkovými soubory *MYD* a *MYI*.

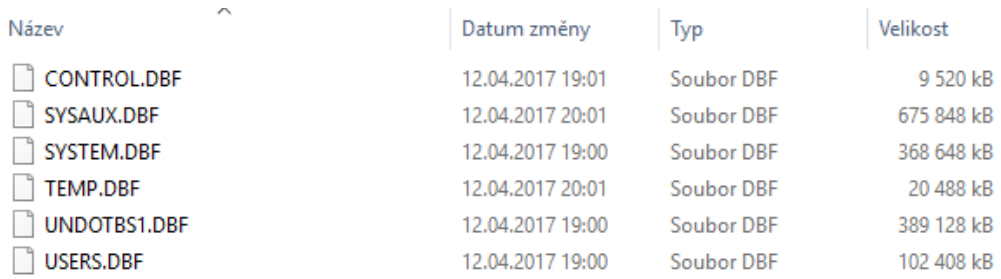


Obrázek 8 Změna typu tabulek v MySQL Workbench

4.4.2 Oracle

Ve výchozím nastavení obsahuje databáze Oracle šest souborů s tabulkovými prostory. Nalézt je můžeme ve složce:

```
C:\oraclexe\app\oracle\oradata\XE
```



Obrázek 9 Datová struktura Oracle 11g XE

Základním tabulkovým prostorem, vytvořeným při vytváření databáze, je *SYSTEM.DBF*. Jeho nejdůležitější součástí je datový slovník skládající se z tabulek, uchovávajících data o objektech celé databáze. *SYSTEM* ke správě využívá pomocného tabulkového prostoru *SYSAUX*.

TEMP.DBF slouží k uchování dočasných dat při dotazování a *UNDOTBS1.DBF* uchovává originální data při započetí transakce. Veškerá uživatelská data a objekty jsou uloženy v tabulkovém prostoru *USERS.DBF*.

Soubor *CONTROL.DBF* je řídícím souborem fyzické struktury databáze. Uchovává informace o všech ostatních souborech, které se v ní nachází [51].

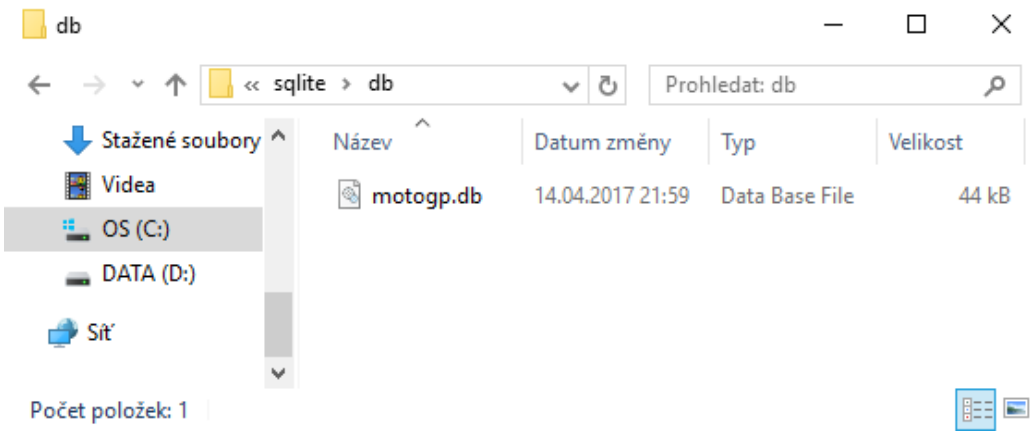
Kompletní výpis velikosti souborů lze nalézt také ve webovém rozhraní správy databáze pod tlačítkem „Storage“. V případě Oracle je využití diskového prostoru znatelně větší než u MySQL.

Tablespace	Free Space (MB)	Used Space (MB)	Percent Used	Maximum (MB)
SYSAUX	39	622		32,768
SYSTEM	1	360		600
UNDOTBS1	360	20		500
TEMP	15	5		32,768
USERS	97	3		11,264

Obrázek 10 Využití diskového prostoru Oracle 11g XE

4.4.3 SQLite

Odlehčená databáze SQLite obsahuje opravdu jen to nejnutnější. Všechna data jsou uložena v jednom souboru s příponou *db*, který se stejným množstvím dat jako v ostatních systémech zabírá pouhých 44kB. V případě, že nad databází provádíme transakci, je ve stejné složce ještě dočasně vytvořen *rollback journal*. Žádné další soubory nejsou potřeba.



Obrázek 11 Datová struktura SQLite

4.5 Výkon

Z hlediska výkonu budou systémy testovány v rámci základních operací CRUD, a to ve dvou testech. V první části testu se budeme věnovat pouze příkazu SELECT. Z vytvořené databáze MotoGP budeme chtít vypsat přehlednou tabulku s výsledky

vybraného závodu. Vzhledem k její velikosti se bude jednat spíše o orientační hodnoty porovnání čtení menšího množství dat.

Druhý test bude o něco průkaznější. Spočívá ve vkládání a editování velkého počtu záznamů, v našem případě to bude číslovka tisíc. Data jsou náhodně vygenerována z webové stránky *freedatagenerator.com*. Výsledky se budou výrazně lišit zvláště při použití transakcí.

4.5.1 Čtení z databáze

Operace SELECT má za úkol vypsat výsledky závodu a seřadit je vzestupně dle pořadí jezdce. Ke spojení cizích klíčů tabulek je použita klauzule WHERE a výsledkem dotazu je výpis osmnácti řádků. Syntaxe kódu vypadá následovně:

```
SELECT poradi_jezdce, pocet_bodu, jmeno, prijmeni, cislo,
nazev_tym, celkovy_cas

FROM vysledky, zavod, zavodnici, jezdec, tym, gp

WHERE vysledky.id_zavod=zavod.id_zavod

AND zavodnici.id_zavodnici=zavod.id_zavodnici

AND tym.id_tym = zavodnici.id_tym

AND jezdec.id_jezdec=zavodnici.id_jezdec

AND gp.id_gp = zavod.id_gp

AND zavod.rok = '2012'

AND gp.id_gp='1'

AND poradi_jezdce>='1'

ORDER BY poradi_jezdce ASC;
```

Oba nástroje klient/server zástupců obsahují analýzu dotazu. Uchovávají si statistiky provedených operací a jsou schopny vypsat jejich průměrný čas.

SQLiteStudio nabízí pouze standardní zaokrouhlený výpis doby trvání po provedení příkazu. Vzhledem k tomu, že ostatní použité nástroje, mimo analýzu, nabízí také standardní dobu provedení příkazu, výsledky testu jsou měřeny právě z těchto údajů. To zajišťuje jistou férovost – výsledkem je průměrná hodnota z deseti výpisů.

SELECT	Oracle	SQLite	MySQL
1	0,0040	0,0030	0,0160
2	0,0050	0,0050	0,0000
3	0,0040	0,0040	0,0000
4	0,0040	0,0040	0,0000
5	0,0050	0,0030	0,0160
6	0,0040	0,0060	0,0000
7	0,0060	0,0030	0,0000
8	0,0060	0,0040	0,0000
9	0,0040	0,0030	0,0000
10	0,0040	0,0040	0,0160
Průměr	**0,0046**	**0,0039**	**0,0044**
Analýza	0,0038	nenabízí	0,0032

Tabulka 3 Výsledky čtení z databáze MotoGP

Na základě jednoduchého výpisu jsou rozdíly opravdu pouze nepatrné. Největší změna přichází, pokud zabalíme dotaz do transakce příkazy BEGIN a COMMIT. V tom případě SQLite vykazuje konstantní dobu výpisu 0,001s.

4.5.2 Vkládání

Jak bylo řečeno na začátku kapitoly, v druhé části testu bude databáze vkládat, editovat a mazat rovný tisíc záznamů.

Důležitým poznatkem k SQLite je transakční přístup. Každý příkaz způsobující změnu záznamu je automaticky v transakci. Pokud jsou tyto dotazy pro každý záznam zvlášť, provedení trvá poměrně dlouhou dobu. Řešením je umístění celého bloku kódu do bloku jediné transakce. Normálně by systém SQLite musel počkat, než se data každého z příkazů řádně přepíší na disk. Obalením kódu do BEGIN a COMMIT dojde ke zpracování složeného dotazu velice rychle, protože systém není limitovaný rychlostí pevného disku [52].

Začněme operací vkládání. Příkaz v základní podobě vypadal takto:

```
INSERT INTO mytable VALUES (1, 'FRANKLIN', 'MEJIA',
'Franklin.MEJ1958@live.com', 'LSD MONTROSE AV', 'EUGENE');…
```

První test byl proveden vložením tisíce příkazů INSERT. SQLite dopadl téměř se 108 vteřinami nejhůř. Překvapivě rychle zvládl vložení do databáze Oracle. V druhém testu byl stejný počet separátních příkazů vložení obalen do jedné transakce klíčovými slovy BEGIN a COMMIT. Zde už si SQLite počínal opravdu rychle a provedení mu zabralo 0,347s. Zklamáním byl tentokráte Oracle, u kterého došlo k nejmenšímu rozdílu. Poslední test vkládá všech tisíc záznamů pouze v jednom příkazu INSERT. V této kategorii zpracoval dotaz nejrychleji systém MySQL.

INSERT	1000 příkazů (s)	1000 v transakci (s)	1 příkaz (s)
MySQL	29,154	2,376	0,473
Oracle	6,839	5,962	3,969
SQLite	107,923	0,347	0,961

Tabulka 4 Test 1000 záznamů INSERT

4.5.3 Editace

Příkaz pro operaci modifikace záznamů vypadal následovně:

```
UPDATE mytable SET street = 'JWZPHCPQHINL' WHERE id = '1';
```

V tomto testu proběhla pouze dvě měření, která byla založená na stejném principu jako u operace INSERT. Editace tisíce záznamů bez transakce trvala všem systémům dlouhou dobu, nejhůře opět dopadlo SQLite se 113 vteřinami. Obalením bloku kódu příkazy BEGIN a COMMIT se výsledky srovnaly.

UPDATE	1000 příkazů (s)	1000 v transakci (s)
MySQL	32,694	0,523
Oracle	9,137	0,370
SQLite	113,245	0,203

Tabulka 5 Test 1000 záznamů UPDATE

4.5.4 Mazání

Závěrečným výkonovým testem byla operace DELETE. Ke smazání záznamů v jednom příkazu byl použit dotaz se syntaxí:

```
DELETE FROM mytable WHERE id>0 AND id<1001;
```

Druhý zápis příkazu k mazání byl rozdělen do tisíce příkazů DELETE a mazal na základě podmínky id záznamu v tabulce.

```
DELETE FROM mytable WHERE id = '1';…

DELETE FROM mytable WHERE id = '1000';
```

Ani u mazání se nekonalo žádné překvapení. Při změně dat v databázi bylo znovu poznat, že obalení do bloku transakce udělá velký skok v konečných výsledcích. Bez transakce SQLite vykazoval opět hodnotu přes 100s a po použití byl příkaz proveden v rámci tisícin vteřiny. DELETE v jednom příkazu trval všem systémům krátce, nejrychlejší byl SQLite.

DELETE	1000 příkazů (s)	1000 v transakci (s)	1 příkaz (s)
MySQL	32,764	3,109	0,203
Oracle	7,35	0,13	0,058
SQLite	105,383	0,052	0,003

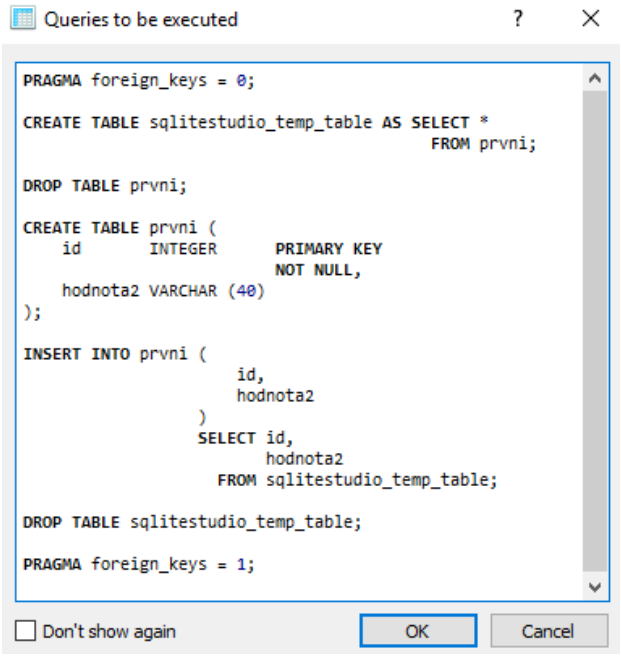
Tabulka 6 Test 1000 záznamů DELETE

4.6 Funckionalita

V poslední kapitole této práce se zaměříme na vybrané funkční vlastnosti systému SQLite. Po celou dobu jsme pracovali s jednotlivými systémy v nástroji určeném k jejich správě. Pracovat s databází lze i přes příkazový řádek a SQLite není výjimkou. Setkáme se u něj s tzv. dot příkazy, které slouží k ovládání systému. Na rozdíl od klasického SQL kódu nemají na konci středník.

Seznam nejdůležitějších dot příkazů i s vysvětlením vypíšeme pomocí *.help* v konzoli. Základem je otevřít databázi, čehož docílíme zadáním *.open FILENAME*. Pro výpis tabulek databáze slouží *.tables* a jejich strukturu zobrazíme přes *.schema TABLENAME*. SQL dotazy lze vkládat normálním zápisem nebo je můžeme importovat příkazem *.read FILENAME* [53].

Použití grafického rozhraní rozhodně usnadní práci. Řeší veškerou logickou funkčnost a přehledně nám vše zobrazí. V kapitole 3 jsem zmínil limitace SQL dotazů v SQLite. Mezi ně patří například smazání sloupce příkazem *DROP COLUMN*. *SQLiteStudio* řeší situaci zkopírováním dat do dočasné tabulky. Následně původní tabulku smaže a vytvoří znovu novou, tentokráte však bez smazaného sloupce.

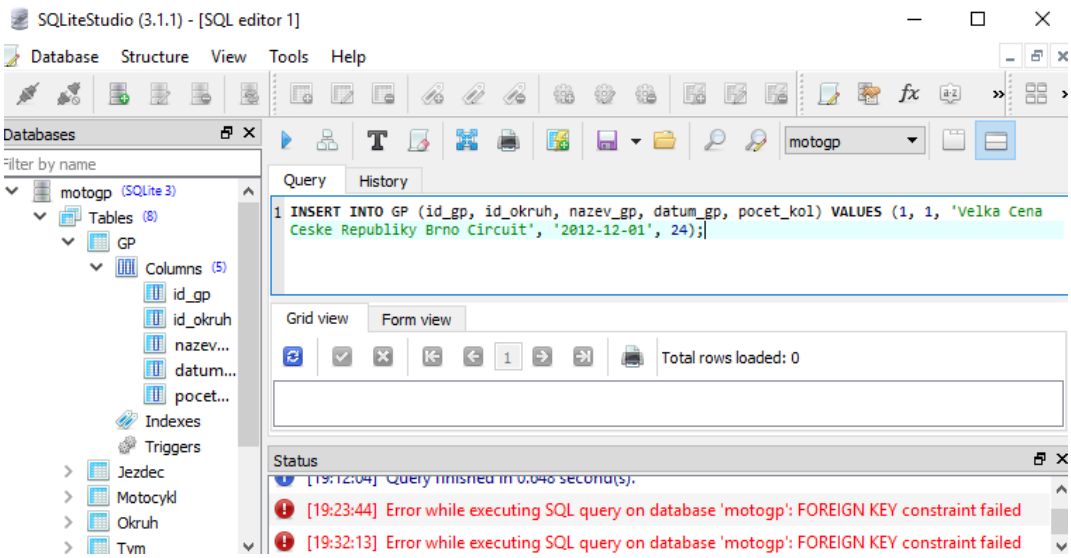


Obrázek 12 Příkaz DROP COLUMN v SQLiteStudio

Další probíranou limitací bylo nepovolené přidávání záznamů z cizí tabulky. SQLite tuto problematiku řeší funkcí *FK constraints* přidanou ve verzi 3.6.19 [45]. SQL syntaxe příkazu vypadá následovně:

```
CONSTRAINT 'FK_Okruh' FOREIGN KEY ('id_okruh') REFERENCES
'Okruh' ('id_okruh') ON DELETE Restrict ON UPDATE Restrict
```

Restrikce je určena pro příkazy vložení, mazání a editace. Když se pokusíme přidat neexistující záznam, vyskočí chybová hláška a vložení se neprovede.



Obrázek 13 SQLiteStudio foreign key constraint

ZÁVĚR

Předmětem této práce bylo porovnání vložené databáze SQLite s vybranými klient/server systémy. Zástupci z této skupiny byli Oracle Database a MySQL, v současné době dva nejpopulárnější dostupné RDBMS. Primárně jsem se v práci zabýval architekturou, funkcionalitou a výkonem zmíněných systémů.

V první kapitole jsem nastínil pojmy nezbytně nutné k pochopení databázové problematiky a hlouběji jsem popsal relační databázový model.

Druhá kapitola popisuje vybrané klient/server systémy a kritéria jejich výběru. Zaměřil jsem se především na architekturu, ze které plynou výhody a nevýhody použití v různých situacích.

Ve třetí kapitole se důkladně věnuji systému SQLite. Již z prvních odstavců je jasné, proč si toto téma vyžádalo celou práci. Přesto, že se jedná o plnohodnotný relační databázový systém, své využití najde spíše v jiných oblastech, než klasické RDBMS. Jeho cílem je nejjednodušší možná funkčnost a implementace. Zásadním rozdílem oproti MySQL a Oracle je, že klient i server běží ve stejném procesu. To s sebou nese spousty výhod i limitací. Na konci kapitoly jsem všechna podstatná hlediska shrnul v přehledné tabulce.

Praktická část je zaměřena na představení práce s SQLite. Nejdříve jsem v krátkosti popsal instalaci a grafická rozhraní, se kterými jsem pracoval. Poté následovala ukázka implementace jednoduché databáze na jednotlivé systémy. Další podkapitolou je datová struktura. Zde SQLite předvádí svou „lehkost“, kdy mu k plnohodnotné funkčnosti celé databáze stačí jeden malý soubor v řádu kilo bytů.

Zásadní podkapitolou je porovnání výkonu. Z testů vyplynulo, že při použití transakčního bloku je prakticky v každém ohledu nejrychlejší SQLite. Záleží ale na mnoha faktorech, blíže specifikovaných v teoretické části. Jedním z nich je počet současně přistupujících uživatelů. Dalším je rozsah a typ aplikace. SQLite je určeno jako vložená databáze do aplikace a do menších webových aplikací. Pro použití na serveru je ve většině případů lepší použít k tomu určené systémy, tedy klient/server.

Hlavním přínosem práce je rozšíření znalostí o systému SQLite jakožto plnohodnotného zástupce relačních databázových systémů. Práce shrnuje výhody a nevýhody nasazení systému na základě srovnání funkcionality a výkonu s MySQL a Oracle.

POUŽITÁ LITERATURA A ZDROJE

[1] HERNANDEZ, Michael J. *Návrh databází*. Praha: Grada, 2006, 408 s. Profesionál. ISBN 80-247-0900-7.

[2] LACKO, Ľuboslav. *SQL: kapesní přehled*. Brno: CP Books, 2005, 96 s. ISBN 80-251-0788-4.

[3] KROENKE, David M. a David J. AUER. *Database processing: fundamentals, design, and implementation*. Ed. 12. Boston: Pearson, 2012, 612 s. ISBN 978-013-2145-374.

[4] LEISS, Oliver a Jasmin SCHMIDT. *PHP v praxi: pro začátečníky a mírně pokročilé*. Praha: Grada, 2010, 256 s. Průvodce (Grada). ISBN 978-80-247-3060-8.

[5] SQL UPDATE Statement. W3schools [online]. Refsnes Data, 2017 [cit. 2017-04-17]. Dostupné z: https://www.w3schools.com/sql/sql_update.asp

[6] DBMS - Transaction. *Tutorialspoint* [online]. Tutorials Point, 2017 [cit. 2017-03-28]. Dostupné z: https://www.tutorialspoint.com/dbms/dbms_transaction.htm

[7] Evaluating the different types of DBMS products. *SearchDataManagement* [online]. Mullins, 2016 [cit. 2017-03-28]. Dostupné z: http://searchdatamanagement.techtarget.com/feature/Evaluating-the-different-types-of-DBMS-products

[8] NoSQL vs SQL. *Microsoft Azure* [online]. Gentz, 2017 [cit. 2017-04-02]. Dostupné z: https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql

[9] SQL vs NoSQL: The Differences. *Sitepoint* [online]. Buckler, 2015 [cit. 2017-03-29]. Dostupné z: www.sitepoint.com/sql-vs-nosql-differences/

[10] MySQL: Why the open source database is better off under Oracle. *ZDNet* [online]. Wolpe, 2014 [cit. 2017-04-02]. Dostupné z: http://www.zdnet.com/article/mysql-why-the-open-source-database-is-better-off-under-oracle/

[11] Sun and MySQL: How It Stacks Up for Developers. *Oracle* [online]. Palkovic, 2008 [cit. 2017-04-02]. Dostupné z: http://www.oracle.com/technetwork/articles/java/mysql-acq-139875.html

[12] Alternative Storage Engines. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/storage-engines.html

[13] InnoDB File-Per-Table Tablespaces. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/innodb-multiple-tablespaces.html

[14] MySQL Glossary. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/glossary.html#glos_crash_recovery

[15] MySQL Glossary. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/glossary.html#glos_buffer_pool

[16] The InnoDB Storage Engine. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://dev.mysql.com/doc/refman/5.5/en/innodb-introduction.html

[17] myisampack — Generate Compressed, Read-Only MyISAM Tables. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/myisampack.html

[18] Dynamic Table Characteristics. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/dynamic-format.html

[19] SCHWARTZ, Baron., Peter. ZAITSEV a Vadim. TKACHENKO. *High performance MySQL*. 3rd ed. Cambridge [Mass.]: O'Reilly, c2012, 771 s. ISBN 978-1-449-31428-6.

[20] MySQL Editions. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://www.mysql.com/products/

[21] Supported Platforms: MySQL Database. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://www.mysql.com/support/supportedplatforms/database.html

[22] Why MySQL? *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://www.mysql.com/why-mysql/

[23] What is WebScaleSQL? *WebScaleSQL* [online]. WebScaleSQL Contributors, 2016 [cit. 2017-04-02]. Dostupné z: http://webscalesql.org/

[24] Restrictions on Stored Programs. *MySQL* [online]. Oracle Corporation, 2017 [cit. 2017-04-02]. Dostupné z: https://dev.mysql.com/doc/mysql-reslimits-excerpt/5.6/en/stored-program-restrictions.html

[25] DB-Engines Ranking. *DB-Engines* [online]. Solid IT, 2017 [cit. 2017-04-03]. Dostupné z: http://db-engines.com/en/ranking

[26] Database Concepts: Brief History of Oracle Database. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-06]. Dostupné z: https://docs.oracle.com/database/122/CNCPT/introduction-to-oracle-database.htm#CNCPT88784

[27] BRYLA, Bob a Kevin LONEY. *Mistrovství v Oracle Database 11g*. Brno: Computer Press, 2009, 700 s. ISBN 978-80-251-2189-4.

[28] Database Instance Structures. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-05]. Dostupné z: https://docs.oracle.com/database/122/CNCPT/introduction-to-oracle-database.htm#CNCPT946

[29] Overview of Grid Computing. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-05]. Dostupné z: https://docs.oracle.com/cd/E11882_01/server.112/e40540/cmntopc.htm#CNCPT1958

[30] Introduction to the Multitenant Architecture. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-05]. Dostupné z: https://docs.oracle.com/database/121/CNCPT/cdbovrvw.htm#CNCPT89234

[31] Overview of Oracle Database Transaction Isolation Levels. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-05]. Dostupné z: https://docs.oracle.com/database/121/CNCPT/consist.htm#CNCPT621

[32] Oracle Cloud. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-05]. Dostupné z: https://www.oracle.com/cz/cloud/

[33] Oracle Database Editions. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-06]. Dostupné z: https://docs.oracle.com/database/121/DBLIC/editions.htm#DBLIC109

[34] Oracle Database Online Documentation 12c Release 1. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-06]. Dostupné z: https://docs.oracle.com/database/121/nav/portal_11.htm

[35] Database VLDB and Partitioning Guide. *Oracle* [online]. Oracle, 2017 [cit. 2017-04-06]. Dostupné z: https://docs.oracle.com/database/121/VLDBG/GUID-2C1ABA68-5671-410A-8F81-D4E4BB5F7137.htm#VLDBG006

[36] SQLite Support Options. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/support.html

[37] GRANT ALLEN a MIKE OWENS. *The definitive guide to SQLite: [take control of this compact but powerful tool to embed sophisticated SQL databases within your applications!]*. 2nd. ed. New York: Apress, 2010, 347 s. ISBN 978-1-4302-3225-4.

[38] The Design Of SQLite4. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/src4/doc/trunk/www/design.wiki

[39] Database File Format. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://www.sqlite.org/fileformat.html

[40] SQLite Query Language: ATTACH DATABASE. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/lang_attach.html

[41] SQLite Query Language: VACUUM. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/lang_vacuum.html

[42] Datatypes In SQLite Version 3. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/datatype3.html

[43] Implementation Limits For SQLite. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://www.sqlite.org/limits.html

[44] SQL Features That SQLite Does Not Implement. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://www.sqlite.org/omitted.html

[45] SQLite Foreign Key Support. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/foreignkeys.html

[46] SQLite Query Language: CREATE DATABASE. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/lang_createtable.html#rowid

[47] Features Of SQLite. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://www.sqlite.org/features.html

[48] Appropriate Uses For SQLite. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://www.sqlite.org/whentouse.html

[49] Most Widely Deployed and Used Database Engine. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/mostdeployed.html

[50] SQLite As An Application File Format. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/appfileformat.html

[51] Managing Database Storage Structures. *Oracle* [online]. Oracle Corporation, 2011 [cit. 2017-04-16]. Dostupné z: http://docs.oracle.com/cd/E25178_01/server.1111/e10897/storage.htm

[52] SQLite Query Language: BEGIN TRANSACTION. *SQLite* [online]. SQLite Consortium, 2017 [cit. 2017-04-16]. Dostupné z: https://sqlite.org/lang_transaction.html

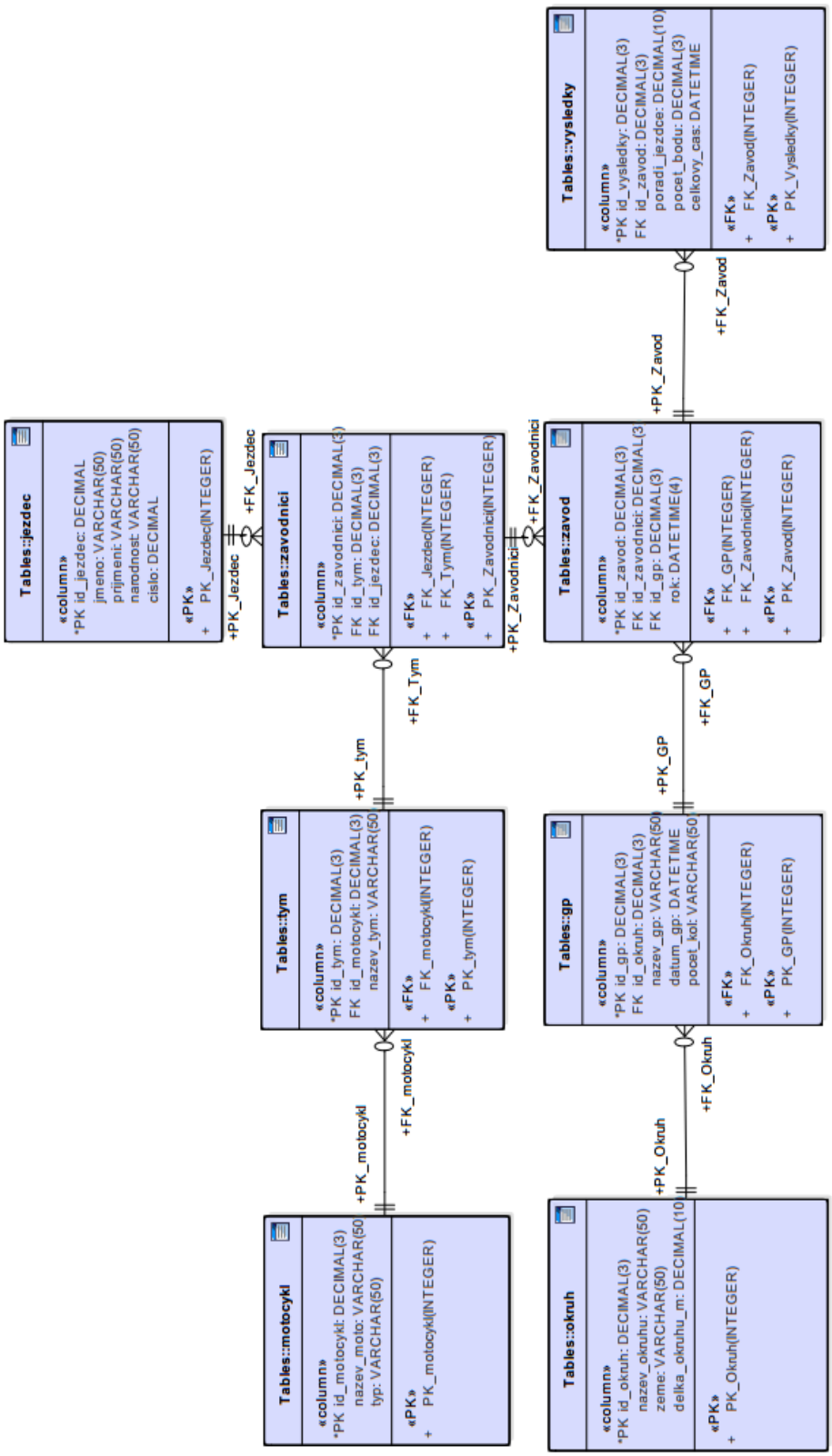
[53] SQLite Commands. *SQLite Tutorial* [online]. SQLite Tutorial, 2017 [cit. 2017-04-16]. Dostupné z: http://www.sqlitetutorial.net/sqlite-commands/

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

PŘÍLOHY

Příloha 1: Databáze MotoGP [vlastní]



Příloha 2: Úložné třídy SQLite a jejich datové typy [42]

Example Typenames From The CREATE TABLE Statement or CAST Expression	Resulting Affinity	Rule Used To Determine Affinity
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	1
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	2
BLOB *no datatype specified*	BLOB	3
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	4
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	5

ANOTACE

Jméno a příjmení:	Jakub Štos
Katedra:	Pedagogická fakulta
Vedoucí práce:	Mgr. Jan Kubrický, Ph.D
Rok obhajoby:	2017

Název práce:	Porovnání klient/server databázových systémů s vloženou databází SQLite z hlediska výkonu a funkcionality
Název v angličtině:	Comparison of client/server database management systems with an embedded SQLite in terms of performance and functionality
Anotace práce:	Tato bakalářská práce se zaměřuje na vloženou databázi SQLite jako zástupce relačních databázových systémů. Rozebírá přitom zásadní rozdíly oproti standardním klient/server systémům. Primárními aspekty porovnání jsou výkon a architektura, ze kterých dále plynou výhody a nevýhody nasazení v praxi.
Klíčová slova:	relační databázové systémy, vložená databáze, klient/server, SQLite, MySQL, Oracle
Anotace v angličtině:	This bachelor thesis focuses on an embedded database SQLite as a representative of relational database systems. At the same time, it describes crucial differences between SQLite and standard client/server systems. Main aspects of the comparison are performance and architecture, both further defining advantages and disadvantages of uses and deployment.
Klíčová slova v angličtině:	relational database systems, embedded database, client/server, SQLite, MySQL, Oracle
Přílohy vázané v práci:	Příloha 1: Databáze MotoGP Příloha 2: Úložné třídy SQLite a jejich datové typy
Rozsah práce:	55
Jazyk práce:	Český jazyk