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

**Bakalářská práce**

# Analýza DB knihoven a technologií pro Javu

Plzeň, 2015 Jaroslav Medek

# Prohlášení

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

V Plzni dne 25. června 2015

Jaroslav Medek

# Abstrakt

Práce je zaměřena na analýzu databázových systémů, které je možné používat jako embedded (in-process) v jazyce Java. Čtenář by měl nejprve získat základní přehled o existujících datových modelech, jejich základních odlišnostech, výhodách případně nevýhodách plynoucích z použití dané technologie. V další části je uveden přehled 18 vybraných databázových systémů s embedded funkcionalitou a představení jejich základních charakteristik – funkce udávané tvůrci systému, podpora, cena, historii vývoje a podobně. Vybrány byly 4 open-source databázové systémy – SQLite, Apache Derby, H2 a HyperSQL, jejichž možnosti jsou podrobněji rozebrány v další části práce. Pro tyto čtyři databázové systémy je dále připraven základní benchmark pro otestování, jak rychlé jsou. Benchmark je založen na vybraných základních vlastnostech TPC-C transakčního testu.

# Abstract

The work is focused on analysis of database systems that supports embedded (in-process) funcionality for Java programming language. The readers should gain an overview of existing data models their fundamental differences, advantages, disadvantages possibly arising when using them. The next section provides an overview of 18 selected database systems with embedded functionality and their basic characteristics - function declared by the creators of the system , production support , price , history of development etc. four open-source database systems were selected for detailed comparsion SQLite, Apache Derby, H2 a HyperSQL. Their functionality is discused in the second section of this work. These four systems were also tested and compared by basic transctional TPC-C like test implemented in this work.

# Obsah

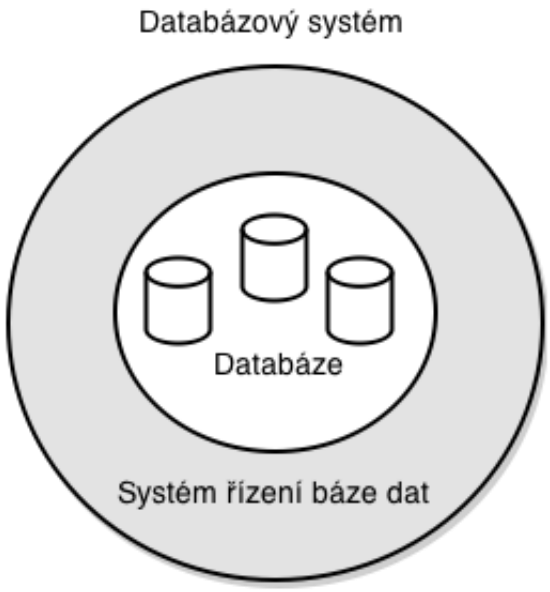
# 1. Úvod

Cílem této práce je dohledat co největší množství knihoven, které umožňují tvorbu embedded databází v Javě, nebo databázových serverů se kterými Java může komunikovat přímo - jinými metodami než přes ODBC / JDBC. Nejprve by měly být uvedeny jen funkce udávané tvůrci systému, podpora, cena, historii vývoje a podobně. U několika vybraných systémů bude provedeno podrobnější srovnání a dále připraven základní benchmark pro otestování, jak rychlé jsou. Benchmark by měl být založen na technologiích, které jsou používány pro testování výkonnosti databázových systémů. V závěru by měl být připraven stručný návod pro použití jednotlivých databází v Javě. Cílem není rozhodnout o tom, který databázový systém je nejlepší pro konkrétní použití, ale spíše poskytnutí základních informací o možnostech, které se nabízejí. Výstup práce je určen každému, kdo hledá jednoduchý, snadno použitelný embedded databázový systém a nemá dostatečnou představu o možnostech, které se v dané oblasti nabízejí.

# 2. Základní pojmy a teorie

## 2.1 Databázový systém

Pojem databázový systém bývá nesprávně zaměňován s databází. Databáze (označovaná také báze dat, nebo zkratkou db) je množina souvisejících, logicky uspořádaných dat uložených v paměti počítače. Aby mohli uživatelé s daty manipulovat, musí databázový systém obsahovat potřebné programové vybavení (například dotazovací jazyk). Takové softwarové vybavení, které umožňuje vkládat, mazat, editovat a dále zpracovávat data uložená v databázi, se nazývá systém řízení báze dat (SŘBD). Každý databázový systém je tvořen nejen vlastní databází, ale také systémem řízení báze dat. [Ott12]




***Obrázek 2.1:*** *Databázový systém je složen z databáze systému řízení báze dat (vlastní zpracování podle [Ott12]).*

## 2.2 Transakce

*„Databázová transakce je jistá posloupnost nebo specifikace posloupnosti akcí, jako jsou čtení, zápis, výpočet, se kterou se zachází jako s jedním celkem“* [Pok97]. Počet a rozsah operací proveditelných v rámci jedné transakce není omezen. Od provádění činnosti celého programu, provedení posloupnosti několika, nebo jednoho SQL příkazu. Transakce se v databázových systémech používají, pokud je nutné provést více příkazů pro SŘBD v jednom nedělitelném celku. Klasickým příkladem je převod peněz mezi dvěma bankovními účty. Vždy musí být zajištěno, že dojde k odečtení částky z jednoho účtu a následně přičtení stejné částky na druhý účet. Nikdy nesmí dojít k situaci, že by

peníze byly z prvního účtu odečteny a na druhý nepřičteny. Pokud bude převod peněz vykonán v rámci databázové transakce, změny proběhnou na obou účtech, nebo neproběhnou vůbec a databáze bude navrácena do stavu před započetím transakce [Man05] [Pok97].

### 2.1.1 ACID

Aby nedošlo k porušení konzistence dat, musí každá databázová transakce splňovat tyto vlastnosti:

- *Atomicita*: Transakce musí být provedena jako nedělitelná (atomická) operace. Operace v rámci transakce jsou provedeny všechny, nebo se neprovede žádná z nich.

- *Konzistence*: Databáze se vždy nachází v konzistentním stavu, během vykonávání transakce nesmí dojít k porušení žádného z integritních omezení. Pokud transakce selže, nesmí být uloženy žádné změny.

- *Izolovanost*: Pokud probíhá více transakcí souběžně, jsou na sobě nezávislé. Výsledek paralelního zpracování transakcí bude stejný, jako když proběhnou sekvenčně.

- *Trvalost*: Po úspěšném průběhu transakce jsou provedené změny uloženy v databázi natrvalo. Je zaručeno, že nedojde ke ztrátě nebo poškození dat v důsledku selhání aplikace, nebo databázového systému.

Transakce, které tyto vlastnosti splňují, se obvykle označují zkratkou ACID (Atomicity, Consistency, Isolation, Durability). Databázový systém, který takovéto transakce podporuje, se označuje za „silně konzistentní" [Pok97] [Staff 14].

## 2.3 Relační databáze

Model relační databáze vychází z práce Dr. E. F. Codda[1], který v roce 1970 publikoval článek „A relational data model for large shared data banks", v němž popsal databázový model založený na operacích relační algebry. [Sum07] [Codd70]. Základním prvkem relační databáze jsou uspořádané n-tice řádků (záznamů) a sloupců (atributů). Takto uspořádané n-tice se označují jako relace. Relaci si můžeme představit jako tabulku, kde každý řádek uchovává data o jednom objektu a každý atribut odpovídá jednomu sloupci. V praxi je potřeba mít nějaký způsob, jak jednoznačně identifikovat záznamy každé tabulky. Docílit toho lze použitím tzv. primárního klíče relace. Jde o atribut, nebo množinu atributů, jejichž hodnota je pro každý řádek tabulky unikátní. Dalším

[1] Anglický matematik a programátor. Pracoval v laboratořích IBM, kde se zabýval především výzkumem v oblasti relačního modelu dat. Je po něm pojmenována jedna z normálních forem – Boyce-Coddova normální forma. [Wik15]

speciálním atributem relace je cizí klíč, který se používá pro zachycení vztahů mezi tabulkami, a stejně jako primární klíč, může být složen z hodnot více sloupců. Pro vytvoření vazby mezi dvěma záznamy z různých tabulek musí být hodnota cizího klíče relace stejná, jako hodnota primárního klíče v „domovské" relaci [Sum07] [Šar03] [Con05]. Další vlastnosti relačního modelu dat jsou následující:

- V jedné databázi nesmí být dvě tabulky se stejným názvem. Požadavek na jedinečnost názvu platí také pro názvy atributů (sloupců) v rámci jedné tabulky [Sum07].
- V relaci nezáleží na pořadí jednotlivých řádků ani sloupců. Jejich záměnou vznikne ta samá relace.
- Každý atribut má předem určenou množinu hodnot, kterých může nabývat. V této souvislosti se používá termín doména atributu [Šar03].
- Hodnota atributu ve sloupci je dále nedělitelná (atomická).
- Primární klíč nemusí být v tabulce obsažen. Pokud ho tabulka obsahuje, musí být jeho hodnota pro každý řádek unikátní, a žádný z klíčových atributů nesmí obsahovat hodnotu NULL [Con05].

V sedmdesátých letech proběhl v IBM výzkum zaměřený na možnosti využití relačních databází, jehož součástí bylo vytvoření speciálního neprocedurálního jazyka pro práci s daty pojmenovaného SEQUEL (Structured English Query Language). SEQUEL byl později přejmenován na SQL (Structured Query Language) a díky vzrůstajícímu využití relačních databází došlo v roce 1986 k jeho standardizaci[2]. [Dat04][Šim99] Příkazy jazyka SQL se dělí do čtyř základních skupin [Con05]:

- DDL (Data Definition Language) – Příkazy sloužící k definici dat, tvorbě a úpravě struktury databáze (například: CREATE, DROP, ALTER).
- DML (Data Manipulation Language) – příkazy pro základní manipulaci s daty. Jedná se především o čtení, zápis, editaci a mazání dat, v literatuře označované Anglickou zkratkou CRUD (Create, Read, Update, Delete).
- DCL (Data Control Language) – Příkazy pro správu uživatelských rolí a oprávnění (například: GRANT, REVOKE).
- TCL (Transaction Control Language) – příkazy sloužící ke správě databázových transakcí (například: COMMIT, ROLLBACK).

Konkrétní možnosti jazyka SQL závisí na tom, jaký standard nebo jeho části jsou v databázovém systému implementovány a mohou se výrazně lišit. Nejčastěji jsou podporovány starší standardy SQL-92 a SQL-99, které bývají doplněny o některé

[2] První byl standard SQL86 vydaný organizací ANSI. Aktuálně již osmou verzí jazyka SQL je standard ISO/IEC 9075:2011, vydaný v prosinci roku 2011. Úplný popis zmíněného standardu lze zakoupit na stránkách organi-zace ISO. Více na: http://www.iso.org/iso/home/search.htm?qt=9075&sort=rel&type=simple&published=on

vybrané funkce z novějších verzí jazyka SQL. Mnoho SŘBD obsahuje také řadu vlastních příkazů, které nejsou součástí SQL standardů. Odlišnosti SQL dialektů v závislosti na výrobci databáze způsobují problémy s přenositelností dotazů, potažmo celé aplikace mezi jednotlivými SŘBD. [Dat04] [Šar03]

Důležitou roli při udržování databáze v konzistentním stavu hraje správný návrh databáze. Jednou z technik používaných při návrhu databáze je normalizace dat, při které zkoumáme funkční závislosti mezi jednotlivými atributy s cílem nalezení optimální struktury jednotlivých relací (tabulek). Správně provedená normalizace snižuje nebezpečí vzniku nekonzistence, jako důsledku redundance dat, nebo jiných tzv. anomálií, ke kterým může docházet při manipulaci s daty. Existuje celkem šest stupňů normalizace, které se souhrnně označují jako normální formy. Doporučena je normalizace alespoň do třetí normální formy, která je dostačující pro eliminaci aktualizačních anomálií. Transformace do vyšších normálních forem většinou není nutná, neboť řeší pouze velmi vzácně se objevující situace [Šar03] [Con05]. Změna rozložení databáze, aby splňovala vyšší normální formy, navíc vede ke složitějším dotazům – selecty pak musí spojovat víc a víc tabulek. Některé databáze proto nesplňují ani první normální formu. Porušení hned první normální formy se často týká i databází obsahujících sloupce s datovým typem BLOB (Binary Large OBject – např. XML soubory, obrázky, videa, pokud jejich obsah není zpracováván jako atomická hodnota.

Během návrhu relačního modelu určujeme povolené hodnoty atributů, vztahy mezi tabulkami, nebo primární klíč každé relace. Při manipulaci s daty je potřeba zajistit, že navržené schéma nebude porušeno vložením nesprávných dat nebo poškozením, případně ztrátou dat stávajících. K tomu se používá sada pravidel, která se nazývají integritní omezení. [Con05][Dat04]

V rámci *referenční integrity* jsou kontrolovány hodnoty cizích klíčů relace. Existuje-li v relaci cizí klíč, musí jeho hodnota odpovídat primárnímu klíči v „domovské" relaci, nebo musí být hodnota všech atributů cizího klíče NULL. *Entitní integrita* musí zajistit jedinečnost primárního klíče relace. Toto pravidlo říká, že žádný z atributů primárního klíče nesmí obsahovat hodnotu NULL. Každý atribut může obsahovat pouze hodnoty z určitého intervalu, definovaného při návrhu databáze. Toto pravidlo se označuje jako *Doménová integrita.* [Con05] [Ott04]

Bez dodržení těchto pravidel je v praxi téměř nemožné udržet databázi v konzistentním stavu. Většina relačních databází kontroluje dodržení definovaných integritních omezení automaticky, je však vhodné si ověřit, zda není kontrola integritních omezení ve výchozím nastavení databázového systému vypnuta. V takovém případě totiž platí, že SŘBD definovaná omezení ignoruje. Bohužel, ne všechny databázové systémy je podporují v plném rozsahu, a tak musí být implementovány v rámci aplikace. [Con05]

Relační model dat poskytuje díky pevně stanovené datové struktuře, ACID transakcím a mocným nástrojem v podobě integritních omezení konzistentní, spolehlivý databázový systém. Existence standardizovaného dotazovacího jazyka a velkého množství sofistikovaných nástrojů pro zálohování, reportování, administraci, migraci dat, jsou další z důvodů, proč jsou relační databáze nejrozšířenějším datovým uložištěm.

Použití relačních databází je však spojeno s řadou nevýhod. Mezi ty nejčastěji zmiňované patří zejména následující:

- Špatná reprezentace objektů reálného světa [Tiw11].
- Homogenita ukládaných dat. Každý řádek musí obsahovat stejné atributy, hodnoty každého sloupce jsou omezeny doménou atributu [Con05].
- Problematické změny databázového schématu. Provedení takových změn se často neobejde bez výrazných zásahů do celé aplikace [Tiw11].
- Nedostatečná podpora integritních omezení ze strany poskytovatelů relačních databázových systémů. Kontrolu integrity dat je potřeba zajistit v rámci aplikace [Con05].
- špatná škálovatelnost [Tiw11].

Kvůli výše zmíněným a některým dalším nedostatkům byly vytvořeny další databázové modely, které jsou alternativou v případech, kdy klasické relační databáze neposkytují dostatečnou flexibilitu a škálovatelnost [Tiw11].

## 2.4 Objektové databáze

Na začátku 90. let došlo ke změně programovacího paradigmatu, a ve vývoji aplikací postupně převládly objektově orientované jazyky. Pro ukládání dat jsou však téměř výhradně používány relační databáze, které mají velké množství nedostatků. Některé z nich byly již zmíněny – obtížné modelování komplexních vztahů mezi objekty reálného světa, homogenní struktura ukládaných dat, omezené datové typy apod. Problematické je samotné použití relačních databází s objektově orientovanými jazyky, protože se často neobejde bez implementace složitého mechanismu mapování mezi záznamy v tabulce a objekty příslušných tříd. Nákladnost vývoje takového procesu byla hlavním impulzem pro vznik objektových databázových systémů (ODBS), které umožňují ukládat data do objektů přímo tak, jak s nimi pracujeme. Kromě toho nabízí všechny základní principy objektově orientovaného programování, jako jsou dědičnost, abstrakce či polymorfismus, které navíc doplňují o možnosti dotazování, transakčního zpracování apod. [Con05] [Bat] [Zím14] [Šve03] [Mer04]

Objektové databáze jsou složeny ze dvou základních prvků – objektů a literálů. Objekty jsou instance určité třídy, která definuje společnou množinu vlastností (atributů) a operací (metod), které lze nad objekty příslušné třídy provádět. Každý objekt má systémem přidělený, unikátní identifikátor (OID – Object IDentifier). OID funguje jako ukazatel na objekt, jeho hodnota je neměnná a naprosto nezávislá na atributech objektu. Vztahy mezi objekty mohou být stejně jako v relačních databázích pouze binární – 1:1, 1:N a M:N. Vytvářejí se pomocí referenčních atributů, typicky s využitím OID. Literál je datová struktura, která může být atomickou hodnotou (atomický literál), kolekcí literálů stejného typu, nebo komplexní strukturou více elementů, kde každým typem může být opět literál, případně objekt. Základní rozdíl mezi objekty a literály je ten, že literál nemá vlastní identitu. Hodnota datových složek

literálu je konstantní a nelze jej jednoznačně identifikovat. Proto se literály nepoužívají samostatně, ale pouze jako součást objektů. [Šve03] [Con05] [Mer04] [Cat00]

Od objektových databází se očekávalo, že postupně nahradí zastaralý relační model. Nestalo se tak především z důvodu vysokých nákladů s tím spojených, a také faktu, že pro relační databáze existuje řada sofistikovaných nástrojů umožňujících zálohování, reportování, migraci dat, administraci apod., které objektové databáze většinou nenabízejí. Dalším problémem objektových databází je nedostatečná podpora integritních omezení. Většina ODBS neumožňuje vytváření a automatickou kontrolu integritních omezení v hierarchii dědičnosti a u kompozitních objektů. [Zaq08]

I přes tyto problémy je použití objektových databází v řadě aplikací velmi výhodné. Jedná se především o systémy, které se vyznačují vysokou komplexitou zpracovávaných dat, jako například geografické informační systémy (GIS), multimediální systémy, CAD systémy apod. [Con05]

## 2.5 Objektově-relační databáze, ORM

Jak již bylo zmíněno, entity objektového programovacího jazyka nelze ukládat do relační databáze přímo. Je potřeba implementovat proces mapování objektů příslušné třídy na záznam relační tabulky a zajistit perzistenci (trvalé uložení) dat. Proto se na trhu objevily objektově-relační databázové systémy (ORDBS), které zajišťují mapovací mechanismus automaticky. Základem ORDBS je klasický relační model dat, doplněný o vybrané funkcionality objektových databází jako OID, dědičnost, polymorfismus, abstraktní datové typy, zapouzdření a další. K dotazování se používá jazyk SQL, rozšířený o možnosti práce s objekty. [ODBMS05]

Nevýhodou ORSŘBD je složitost celé technologie a s tím spojené vyšší náklady. Navíc platí, že proces převodu objektů do relační databáze je spojen se značnou režií, a může snižovat výkonnost databázového systému. [ODBMS05]

Druhou možností, jak zajistit automatické mapování záznamů relační databáze na objekty a jejich perzistenci jako objektu příslušné třídy, jsou nástroje pro objektově relační mapování (ORM). Použitím ORM nástrojů lze dosáhnout odstínění programátorů i uživatelů od konkrétní relační databáze. Díky tomu je zajištěna bezproblémová přenositelnost aplikace mezi podporovanými SŘBD. Nejpopulárnějším ORM nástrojem je Hibernate, který je v této oblasti považován defacto za standard. Aktuální verze podporuje mapování a perzistenci objektů pro zhruba 30 relačních databází. Hibernate disponuje vlastním objektovým dotazovacím jazykem Hibernate Query Language (HQL), který je syntakticky velmi podobným jazyku SQL. Použití HQL usnadňuje přenositelnost aplikace mezi jednotlivými databázovými systémy, které mohou používat odlišné dialekty jazyka SQL. [Len09] [Hib15]

Nevýhody nasazení ORM jsou podobné jako v případě ORDBMS. Komplexní nástroje jako Hibernate vyžadují další náklady spojené s osvojením a nasazením technologie. [Len09] [Hib15]

## 2.6 Nosql databáze

Každé dva roky se objem vznikajících dat přibližně zdvojnásobí, přičemž v roce 2020 bude až 33 % dat (oproti dnešním cca 25%) obsahovat informace, které by mohly být komerčně využity. [Gan12] Problémem je, že pro zpracování takového objemu často nestrukturovaných, komplexních a značně propojených dat, nejsou relační databáze vhodné; z důvodu špatné škálovatelnosti a požadavku na (alespoň částečnou) předvídatelnost a strukturovanost dat. První zareagovala společnost Google, která nejprve v roce 2003 zveřejnila principy souborového systému pro ukládání distribuovaných dat GFS (Google File System), a poté v roce 2006 představila sloupcové úložiště BigTable. [Chan08] U projektu BigTable se inspiroval internetový prodejce Amazon a v roce 2007 zveřejnil koncept key/value databáze DynamoDB, [Dec07] kterou začal využívat ve svých distribuovaných systémech. [Pok12]

Na základě zveřejněných konceptů projektu BigTable a Dynamo vzniklo velké množství open-source databázových systémů, pro které se dnes používá označení Nosql (Not only sql) [Neu10]. Hlavní charakteristiky Nosql databázových systémů jsou následující:

- Podporují nerelační datový model (key/value, sloupcové, dokumentové, grafové) [Neu10].
- ACID model konzistence je většinou podporován pouze částečně. Místo něj je podporován alternativní model zajištění konzistence dat – BASE (Basic Availability, Soft-state, Eventual consistency).
- Jsou primárně určeny pro použití v distribuovaných systémech (obsahují podporu pro snadnou replikaci a distribuci dat) [Tiw11].
- Jsou vysoce horizontálně škálovatelné [Neu10].
- K dotazování nepoužívají jazyk SQL (práce s daty je řešena vlastním jednoduchým API, někdy je k dispozici vlastní dotazovací jazyk, který může syntakticky vycházet z SQL) [Tiw11].
- Nevyžadují dodržení pevného schéma databáze (umožňují ukládání nestrukturovaných dat) [Tiw11].

### *2.1.2 BASE*

Nosql databáze jsou primárně určeny pro použití v distribuovaných databázových systémech, kde je většinou preferována výkonnost a zajištění vysoké dostupnosti dat na úkor konzistence dat. Místo silně konzistentních systémů s ACID transakcemi se zde často setkáváme s přístupem označovaným zkratkou BASE [Bro09] [Staff14]:

- *Basic Availability*: Data jsou neustále k dispozici, systém je vysoce odolný proti poruchám.

- *Soft-state*: Data nemusí být v každém okamžiku konzistentní, zajištění konzistence dat je zodpovědností vývojářů klientské aplikace a neměla by být řešena databázovým systémem.
- *Eventual Consistency*: Konzistence není zajištěna okamžitě, ale je garantováno, že se v budoucnosti databáze dostane do konzistentního stavu.

### 2.1.3 Klíč/hodnota databáze

Nejjednodušším typem Nosql databází jsou databáze klíč/hodnota (key/value), které fungují na stejném principu jako hash tabulky. Záznamy tvoří dvojice klíč/hodnota, kde klíč (nejčastěji řetězec) slouží k jednoznačné identifikaci hodnoty v databázi. Hodnota může být libovolná, strukturovaná i nestrukturovaná (typicky BLOB). Key/value databáze nevyžadují pevné schéma dat, takže dvojice klíč-hodnota nemusí být stejných typů a není nutné ukládat hodnoty NULL (na rozdíl od záznamů v relačním modelu). Záznamy jsou většinou ukládány v pořadí seřazeném podle hodnoty klíče, pro zrychlení vyhledávání v databázi. Možnosti pro další zpracování dat jsou v key/value databázích velmi omezené. Typicky jsou implementovány pouze operace pro ukládání, získávání a odstranění hodnoty na základě znalosti klíče – put(key), get(key), delete(key). [Pok12] [Dec07] [Tiw11]

Většina dnes používaných klíč/hodnota databází vychází z distribuovaného databázového systému DynamoDB, navrženého pro potřeby internetového obchodu Amazon. Nejčastěji se využívají ve webových aplikacích pro správu informací u jednotlivých spojení (nastavení uživatelských profilů, monitorování aktivity, ukládání nákupních košíků v on-line obchodech apod.). V těchto případech nabízí klíč/hodnota databáze vysoký výkon a škálovatelnost i při desítkách milionů současně připojených uživatelů. Nevýhodou key/value uložišť je příliš jednoduchý datový model a s tím spojené omezené možnosti práce s daty. [Dec07] [Chan08] [Ree13]

### 2.1.4 Sloupcové databáze

Datový model je velmi podobný relačním úložištím, ale záznamy jsou na místo po řádcích ukládány po jednotlivých sloupcích. Každý záznam je složen z dvojice (klíč, hodnota), kde klíč udává název sloupce a hodnota může být libovolná, jako je tomu v key/value úložištích. Často se spolu s každým záznamem uchovává také časový údaj o době vzniku, nebo poslední editaci záznamu. Sloupce se souvisejícími hodnotami jsou slučovány do skupin a každá skupina sloupců obsahuje množinu unikátních klíčů, které zajišťují jednoznačnou identifikaci každého řádku ve skupině sloupců. Datový model nevyžaduje dodržování pevného schématu – sloupce nemusí být definovány předem a řádky v rodině sloupců mohou obsahovat různé sloupce. [Tiw11] [Pok12] [Bry14]

Výhodou organizace dat po sloupcích je možnost pracovat přímo pouze s daty z vybraných sloupců. To výrazně zvýší výkonnost zejména těch aplikací, které často čtou pouze malé množství sloupců z velké tabulky, protože nemusí vybírat celé řádky a požadované sloupce z nich extrahovat. Vzhledem k velkému množství shodných

hodnot v jednotlivých sloupcích provádí většina sloupcových databázových systémů kompresi dat. Kompresí lze výrazně snížit velikost dat a urychlit tím jejich čtení z disku. Běžný je kompresní poměr 5:1, v některých případech je možné dosáhnout i 10ti násobného zmenšení velikosti dat oproti jejich nekomprimované podobě. Většinu sloupcových databází tvoří klony distribuovaného úložiště BigTable od společnosti Google. Využívají se především v datových skladech a analytických systémech, které potřebují velké množství dat. [Tiw11] [Swa13] [Bry14] [Staff14]

### *2.1.5 Dokumentové databáze*

Datový model je velmi podobný s databázemi klíč/hodnota, ale hodnotou je zde semistrukturovaný dokument. Dokumenty jsou obvykle složeny z párů klíč-hodnota, mohou obsahovat indexy, případně do nich mohou vnořeny další dokumenty. Nejčastěji se dokumenty ukládají ve formátech JSON, XML, nebo jako objekt. Některé dokumentové databáze používají vlastní formáty pro reprezentaci dat, například databázový systém MongoDB převádí ukládané dokumenty do interního formátu BSON. Dokumenty obsahující související data se obvykle seskupují do tzv. kolekcí, které si lze představit jako ekvivalent tabulky v relačních databázích. V rámci kolekce mohou mít jednotlivé dokumenty vzájemně odlišnou strukturu, která nemusí být definována předem a lze ji kdykoliv změnit. [Tiw11] [Bry14] [Pok12]

Velkou výhodou dokumentových databází je právě formát JSON. Do něj lze snadno převést objekty většiny programovacích jazyků, a vkládat je přímo do databáze, bez složitého mapování mezi záznamy a objekty příslušných tříd, jako je tomu v případě relačních databází. Využití dokumentových databází je obdobné jako v případě key/value databází, ale dokumenty lze snadno využít pro ukládání velkého množství dat, a pracovat s nimi pomocí jediného klíče. Například se může jednat o ukládání informací o konkrétním produktu, uchovávání logovacích dat, údaje z uživatelských profilů apod. [Tiw11] [Pok12] [Bry14]

### *2.1.6 Grafové databáze*

Výše zmíněné Nosql databáze typicky pracují s obrovským množstvím nestrukturovaných dat, které je potřeba co nejrychleji ukládat a následně ve stejné podobě získat. Pokud však potřebujeme analyzovat vztahy mezi daty, neobejdeme se bez dalšího zpracování. Proto je výhodné ukládat data do grafových struktur, které poskytují náhled do propojenosti dat přímo, bez komplikovaných analytických výpočtů. [Bach14] V grafových databázích jsou dva druhy entit – uzly (vrcholy grafu) a hrany. Nejčastěji pracují s tzv. grafem vlastností (property graph), kde každý vrchol obsahuje další údaje objektu, který reprezentuje, a hrany obsahují popis vztahů mezi vrcholy, které propojují. Data v uzlech a hranách jsou typicky ukládány jako páry klíč/hodnota a mohou být prohledávány indexovaným způsobem jako v relačních databázích. Po nalezení požadovaného vrcholu lze přistupovat ke všem jeho sousedům v konstantním čase, bez ohledu na rozsáhlost grafu. Při průchodu grafem je možné využívat množství grafových algoritmů. Například algoritmus vyhledávání nejkratších cest mezi uzly, Dijkstrův algoritmus, algoritmus minimální kostry grafu a řadu dalších. [Hol12] [Bach14] [Rob13]

Grafové databázové systémy jsou v současnosti velice populární, rychle se rozšiřující databázovou technologií. Ředitel společnosti Neo Technology Emil Eifrem předpokládá, že do konce roku 2020 bude nejméně čtvrtina z 2000největších společností světa využívat ve svých systémech grafové databáze. Typické příklady pro využití grafové databáze jsou doporučovací systémy a sociální sítě. Například obchodní řetězec Wall-Mart využívá grafovou databázi pro analýzu nákupního chování zákazníků; oblíbenost jednotlivých výrobků u specifických skupin zákazníků, jaké kombinace produktů jsou nakupovány společně atd. Získané údaje se pak využívají k cílené reklamě. Naopak pro obyčejné ukládání velkého objemu agregovaných dat, nejsou grafové struktury vhodné. V těchto případech je výhodnější použít jiné Nosql databáze, s jednodušším schématem, které poskytují lepší výkonnost a škálovatelnost. [Hol12] [Bach14] [Woo15]

# 3. Představení vybraných databázových systémů

Následuje stručný popis 18 vybraných SŘBD, které je možné používat jako embedded (in-process) v jazyce Java. Srovnání některých parametrů a další doplňující informace jsou uvedeny v souhrnných tabulkách v příloze této práce. Další databázové systémy (nejen pro jazyk Java), lze dohledat například na webu db-engines.com[3].

## 3.1 Relační databáze

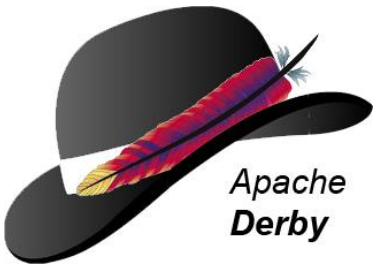
### 3.1.1 Firebird


Projekt Firebird[4] vznikl v roce 2000 po uvolnění zdrojových kódů databázového systému InterBase 6.0 od společnosti Borland. Prvotní verze byla napsána v jazyce C, ale od verze 1.5 byl kód Firebirdu z velké části přepsán do C++. Firebird je k dispozici ve čtyřech variantách: Classic, Superclassic, Superserver a Embedded. Hlavní rozdíl mezi nimi je ten, že Classic používá pro každé spojení samostatný, nezávislý proces, zatímco Superserver a Superclassic sdílí svoji cache paměť se všemi spojeními, které jsou obsluhovány v samostatných vláknech. Verze Superclassic a Classic je možné použít také jako embedded server spravující lokální databázi. Připojit se k databázi lze pomocí nativního

[3] http://db-engines.com/en/ranking
[4] http://www.firebirdsql.org/

API, dostupné jsou ovladače pro JDBC, ODBC a rozhraní pro použití Firebirdu v .NET, PHP, Python a dalších jazycích. Dostupné jsou oficiální i neoficiální nástroje pro správu zabezpečení, replikaci, administraci databáze a další. [Can10] [Fir14] [Fir11]

### 3.1.2 Apache Derby




Apache Derby[5] je relační databáze, vycházející ze zdrojového kódu projektu Cloudscape společnosti IBM. Na vývoji Apache Derby se podílí také společnost Oracle, která jí pod názvem JavaDB distribuuje jako součást standardních JDK od verze 1.6. Derby je napsána kompletně v Javě a je možné ji používat pouze s jazyky, které pracují v Javovském virtuálním stroji. Připojení k databázi je možné pomocí JDBC ovladače, a také jednoduchým nástrojem ij, se kterým lze spouštět SQL dotazy a skripty z příkazové řádky. V případě potřeby sdílení databáze s více uživateli může být databázový stroj spuštěn v režimu server, komunikující se vzdálenými klientskými aplikacemi pomocí TCP/IP a protokolu DRDA[6]. Databáze může být perzistentně uložena na lokální diskové úložiště, nebo je možné použít in-memory databázi, která uchovává data v operační paměti pouze po dobu běhu programu. Transakce splňují ACID vlastnosti a mohou probíhat vícevláknově s použitím zamykání na úrovni řádků, nebo tabulek. Použitý dialekt jazyka SQL splňuje všechny požadavky standardu SQL-92. Z ostatních standardů je implementována pouze část vybraných funkcionalit[7]. Přístup k datům uloženým na disku je možné šifrovat s využitím algoritmů z knihovny JCE (Java Cryptography Extension). [Apa14]

### *3.1.3 Hyper SQL DataBase*

HyperSQL

Relační databázový systém napsaný v Javě, dostupný pod open-source BSD licencí. Využívá se ve více než 1700 open-source projektech a řadě komerčních produktů. Mezi nejznámější patří výpočetní software Mathematica, balík kancelářských programů OpenOffice a distribuovaná relační databáze Voltdb, která používá HyperSQL při zpracování SQL dotazů. Databázový stroj může být spuštěn jako samostatně běžící server obsluhující klientské aplikace (Client-Server), nebo jako součást Javovské aplikace (embedded mód). V embedded režimu lze pracovat s více databázemi spuštěnými v jedné JVM. V HyperSQL je možné uchovávat data pouze dočasně, po dobu spuštění aplikace (TEMP tabulky), nebo perzistentně s ukládáním na disku

[5] http://db.apache.org/derby/
[6] https://collaboration.opengroup.org/dbiop/
[7] http://wiki.apache.org/db-derby/SQLvsDerbyFeatures

(CACHED a TEXT tabulky), případně uložené v operační paměti (MEMORY tabulky). U MEMORY tabulek je perzistence dat zajištěna použitím souboru s příponou .script, který obsahuje definici tabulek a všechny další SQL dotazy provedené v databázi. Po opětovném spuštění databáze se MEMORY tabulky z tohoto souboru znovu vytvoří. Tento logovací soubor lze využívat také v kombinaci s CACHED a TEXT tabulkami, kde slouží především pro obnovení databáze do konzistentního stavu v případě chyby. Na rozdíl od většiny relačních databází podporuje HyperSQL mnoho standardů jazyka SQL, včetně posledního standardu SQL:2011. Transakce splňují ACID vlastnosti a mohou být zpracovávány vícevláknově. Součástí aplikačního balíku HyperSQL je knihovna SQLTool, umožňující provádění SQL dotazů z příkazového řádku a nástroj Database Manager, poskytující grafické rozhraní pro správu databáze. Obě tyto knihovny mohou být použity pro správu libovolného databázového systému, který umožňuje připojení přes JDBC rozhraní. [Sim14]

### *3.1.4 H2*


H2 (Hypersonic2) pochází od stejného autora jako HyperSQL, nicméně zdrojový kód je napsán zcela od začátku. Z hlediska funkcí jde o velmi podobný SŘBD, doplněný o možnost nativního nebo Apache Lucene fulltextového vyhledávání a přímou podporou clusterování i replikace databáze. Stejně jako v HyperSQL může být databáze spuštěna jako samostatný server, nebo být součástí stejného procesu jako klientská aplikace. Kromě toho je možné pracovat v tzv. mixed módu, kdy lokální aplikace pracuje s embedded databází a zároveň spustí serverový proces pro obsluhu vzdálených spojení. Kromě Javy může být h2 použita i na platformě .NET pomocí rozhraní ADO .NET z projektu h2sharp[8]. Transakce jsou atomické, konzistentní a podporují tři různé úrovně izolace. Nicméně, trvalost dat v případě výpadku napájení není zaručena. Při paralelním zpracování transakcí je izolovanost zajištěna exkluzivním zamykáním tabulek pro zápisové operace, alternativně je možné použít mechanismus MVCC, který umožňuje sdílení zámků u vybraných zápisových transakcí, nebo zamykání pouze na úrovni řádků. Databáze může spuštěna v tzv. compatibility módu, kdy se emuluje chování některých vybraných SŘBD s cílem poskytnout přenositelnost aplikace pracující s H2 databází na jiný databázový systém. Podporovány jsou režimy kompatibility pro HyperSQL, Derby, DB2, MySQL, PostreSQL, Oracle a MS SQL. Kompatibilita však rozhodně není absolutní a pokrývá pouze malé množství rozdílných funkcionalit, zejména v oblasti SQL jazyka jednotlivých SŘBD. S databází H2 je k dispozici také nástroj H2 Console. Funguje jako samostatná aplikace a obsahuje vlastní webový server s databází H2 přístupný přes libovolný webový prohlížeč. Aplikaci H2 Console je možné používat také pro přístup k H2 databázi v embedded režimu, nebo libovolné databázi, která podporuje JDBC rozhraní. [H2d15]

---

[8] https://code.google.com/p/h2sharp/

### 3.1.5 SQLite


SQLite[9] je malá databázová knihovna, napsaná v jazyce C, kterou stačí přilinkovat k aplikaci a pomocí jednoduchého rozhraní s ní pracovat. Rozhraní pro přístup k databázi je implementováno pro jazyky C++, Java, Perl, PHP, C# a řadu dalších. Pro použití v Javě je nejvhodnější použít JDBC ovladač xerial[10], který je dostatečně otestován a pravidelně aktualizovaný. Celá databáze je uložena v jediném souboru, který má univerzální formát a může být přenášen mezi libovolnými operačními systémy i nezávisle na endianitě použitého stroje. Podporována je velká část symtaxe standardu SQL-92, indexované vyhledávání, triggery i ACID transakce. Mezi relačními SŘBD uvedenými v této práci je SQLite jediným systémem, umožňujícím sdílení jedné databáze více procesy (v embedded režimu). Nicméně, zápisová operace vyžaduje exkluzivní zamčení databáze pro všechny ostatní transakce včetně čtecích, což se nehodí v aplikacích, které požadují zpracování velkého množství zápisových transakcí. Nasazení SQLite v takových aplikacích se ani nepředpokládá, a proto ani neumožňuje klient-server režim využití databáze. Nejčastěji se používá ve webových aplikacích a embedded zařízeních[11]. Je součástí webového prohlížeče Firefox a operačního systému Android [Sqlite15].

## 3.2 Objektové databáze

### 3.1.6 Perst

Open-source, embedded objektová databáze s duální licencí, distribuovaná samostatně pro jazyk Java a platformu .NET, která je konverzí Java verze do jazyka C#. Pro Javu je na výběr ze tří verzí: Perst 1.5 pro JDK 1.5 a vyšší, Perst 1.4 používaná s JDK 1.4, a Perst Lite, určená pro Java ME[12]. Persistence objektů je zajištěna vlastním API, které implementuje část standardu JDO. Perst[13] Nabízí fulltextové vyhledávání, XML import/export objektů, master-slave replikaci i šifrování databáze. K dotazování používá objektový jazyk JSQL, syntakticky vycházející z SQL. Více transakcí může být zpracováváno současně ve vlastních vláknech, umožněno je i sdílení jedné transakce více vlákny. Izolace transakcí je dosažena zamykáním na úrovni objektů. Vyhledávání v

---

[9] https://sqlite.org/
[10] https://bitbucket.org/xerial/sqlite-jdbc
[11] https://www.sqlite.org/famous.html
[12] Java Micro Edition – Verze Javy určená primární k vývoji softwaru pro zařízení s omezenými prostředky.
[13] http://www.mcobject.com/perst_eval

databázi je založeno na indexovacích algoritmech. Při prohledávání vícerozměrných dat je použit R-tree algoritmus, pokud je potřeba prohledávání podle více kritérií, používá se K-dimensional tree algoritmus. Nejčastěji se databáze Perst využívá v mobilních zařízeních s OS android, nebo webových aplikacích. Společnost McObject nabízí výukové kurzy i placenou technickou podporu. [McO14] [Mco12]

### *3.1.7 Objectivity DB*


Distribuovaný, objektový databázový systém plně splňující požadavky standardu ODMG 3.0. K databázi se přistupuje rozhraním ODBC, podporovány jsou jazyky C++,C#, Java, Python, Smalltalk. Objectivity/DB[14] nepracuje jako databázový server. Jedná se o aplikační knihovnu a dva server procesy, které zajišťují obsluhu zámků a manipulaci s daty. K dotazování v jazyce Java je možné použít nativní API, případně samostatně licencovatelný engine pro paralelní zpracování dotazů. Konzistence databáze při souběžném přístupu více uživatelů je zajištěna exkluzivním zamykáním při modifikaci objektů, čtení obsahu objektu může probíhat souběžně z více transakcí. V nabídce je pouze komerční verze systému, jehož možnosti lze otestovat zdarma dostupnou, 60denní trial verzí. V rámci samostatných licencí je nabízena řada sofistikovaných nástrojů např. pro usnadnění vývoje, optimalizaci výkonu, import/export dat mezi databází a XML soubory, replikaci databáze, a mnoho dalších. Zdarma je k dispozici rozsáhlá dokumentace, včetně množství ukázkových příkladů. Mezi společnosti využívající Objectivity/DB se řadí například Ericsson, Siemens nebo Lockheed Martin. [Objectivity15]

### *3.1.8 ObjectDB*

V Javě napsaný objektový databázový systém, podporující klient-server, i embedded režim provozu. Nabízena je pouze komerční licence, nicméně v omezené podobě může být využívána bezplatně pro libovolné účely, s omezením velikosti databáze na maximální počet deseti entitních tříd a jednoho milionu objektů. Pro výzkumné a studijní účely lze získat plnou verzi zdarma. ObjectDB umožňuje přístup k datům pomocí rozhraní JDO, nebo JPA. Obě rozhraní nabízí ACID transakce i souběžný přístup více uživatelů. K izolaci transakcí se využívá optimistický / pesimistický model zamykání objektů. Použití rozhraní JPA je výhodné s ohledem na přenositelnost aplikace. Stejné rozhraní totiž typicky implementují ORM nástroje, a tak lze v případě nutnosti snadno přejít na relační databázi. Vzhledem k objektovému modelu odpadá režie spojená s mapováním Java objektů, které provádí ORM nástroje, díky čemuž

[14] http://www.objectivity.com/products/objectivitydb/

dosahuje ObjectDB výrazně lepších výkonnostních výsledků, než ostatní poskytovatelé JPA.[15] Dotazovací jazyk záleží na použitém rozhraní. Pro JDO se nabízí JDOQL, s rozhraním JPA je spojen jazyk JPQL. [Obj15] [Obje15]

Hlavní nevýhody spojené s použitím ObjectDB jsou následující:

- Chybějící podpora integritních omezení. Je na zodpovědnosti programátora zajistit kontrolu integrity databáze na straně aplikace.
- Absence oficiální dokumentace. Na webových stránkách projektu je pouze stručný popis systému, a elementárních příkladů užití.
- Podporovány jsou pouze JVM programovacími jazyky – Java, Groovy, Scala, Jruby apod. Ve vývoji je verze pro platformu .NET. [Obj15] [Obje15]

# 3.3 Key/Value databáze

### *3.1.9 Oracle Berkeley DB Java Edition*

Původní verze byla vydána v roce 1994 na univerzitě v Berkeley, a je napsána kompletně v jazyce C. V současnosti patří Oracle Berkeley DB pod společnost Oracle, která ji nabízí ve třech různých verzích – původní Oracle Berkeley DB, v Javě napsanou verzi Oracle Berkeley DB Java Edition a Oracle Berkeley DB XML, která je určena pro práci s XML dokumenty. Všechny tři verze jsou pro nekomerční využití dostupné pod open-source licencí. K použití v rámci proprietárního softwaru je nutné koupit licenci, která se pohybuje od 26000 Kč/procesor[16].

Pro práci s databází jsou v Java Edition (JE) určena dvě různá rozhraní:

- Direct Persistence Layer (DPL) pro práci s POJO objekty.
- Rozhraní pro práci s libovolnými daty jako s páry klíč/hodnota.

Přenositelnost databázových souborů mezi JE a původní verzí Berkeley DB není možná z důvodu jejich odlišných formátů. Nicméně, součástí API jsou funkce umožňující import/export dat mezi nimi. [Oracle15]

---

[15] Srovnání výkonu ObjectDB s vybranými relačními databázemi a ORM nástroji: http://www.jpab.org/.

[16] https://shop.oracle.com/pls/ostore/f?p=dstore:2:0::NO:RIR,RP,2:PROD_HIER_ID:4509887109971805720003

### 3.1.10 BangDB

Open-source key/value úložiště od společnosti Iqlect, kompletně napsané v jazyce C++. Klient Pro použití v jazyce Java je zatím dostupný pouze pro platformu Linux. BangDB je nabízena ve dvou rozdílných verzích – BangDB embedded, která může fungovat jako perzistentní, nebo cache úložiště a BangDB Client-server, pro síťové sdílení databáze. API pro práci s daty podporuje základní CRUD operace, tvorbu dotazů na základě omezení rozsahem vybrané hodnoty (range query), třídění a další zpracování výsledků dotazů. Součástí API jsou různé možnosti pro analýzu dat, například třídění podle doby vzniku, sledování TopK[17] položek a některé další. Transakce splňují ACID vlastnosti a mohou probíhat paralelně ve více vláknech. Izolovanost souběžně zpracovávaných transakcí zajišťuje optimistický model zamykání (OCC). Pro zvýšení odolnosti vůči ztrátě dat je možné zapisovat veškeré změny v databázi, před jejich provedením, do speciálního souboru, který se použije pro obnovu konzistence dat v případě, že dojde k nečekané havárii databáze. Obě verze BangDB jsou dostupné zdarma v rámci BSD licence. Do budoucna plánuje společnost Iqlect vydání placené verze databázového systému. [Iqlect14]

### 3.1.11 LevelDB


Embedded key/value uložiště od společností Google, která jej v roce 2011 uvolnila pod BSD licencí. [Dea11] Využívá se například v prohlížeči Google Chrome, pro ukládání bitcoinových transakcí (verze LevelDB pro Node.js), a také v key/value databázi Riak. Level DB je napsána v jazyce C++. Pro použití s Javou je nutné použít dostupné Java API[18], nebo do Javy přepsaný port Dain Leveldb[19], který ale neposkytuje všechny funkcionality originálu. Data jsou ukládána v seřazeném pořadí podle klíče a automaticky kompresována použitím kompresního algoritmu z knihovny Snappy[20]. Pro manipulaci s daty je určeno vlastní jednoduché API, LevelDB nemá žádný dotazovací jazyk a neumožňuje indexaci (ani automatickou). Databáze nemůže být sdílena více procesy, nicméně transakce mohou probíhat souběžně v různých vláknech. Automatická synchronizace vláken je prováděna pouze pro čtecí a některé zápisové operace. U ostatních zápisových operací musí být zajištěna externě. [LevelDB] [Sin12] [Dea]

---

[17] Analýza „top“ položek za vybrané časové období. Například 10 nejčastěji vykonávaných dotazů, uživatelé s největším počtem položek v nákupním košíku, nejčastěji vyhledávané produkty apod. [Iqlect14]

[18] https://github.com/fusesource/leveldbjni

[19] https://github.com/dain/leveldb

[20] https://code.google.com/p/snappy/

## 3.4 Sloupcové databáze

### *3.1.12 HBase*

HBase se nejčastěji využívá v kombinaci s HDFS[21] a dalšími nástroji z frameworku Apache Hadoop[22], jako distribuovaná databáze. Kromě toho může být využita i jako embedded databáze (stand-alone mod), kdy se místo HDFS používá lokální datové úložiště. HBase je napsána pro použití v jazyce Java, procesy z jiných programovacích jazyků se mohou k databázi připojit prostřednictvím rozhraní Apache Thrift[23]. V distribuované verzi je podporována master-slave replikace i sharding[24] databáze mezi jednotlivé uzly clusteru. K manipulaci s daty slouží čtyři základní operace – get, put, scan, delete. SQL jazyk není podporován, nicméně je možné použít například Apache Hive[25], umožňující automatický převod SQL dotazů v rámci platformy Apache Hadoop. Hbase je použita v celé řadě známých projektů, například Facebook ji využívá při uchovávání zpráv (SMS, emaily, chatovací systém[26]). Vyhledávač Yahoo! u systému na detekci duplicitních dokumentů, Twitter pro vyhledávání uživatelů, zálohu dat a některé další operace. [Apa15] [ApaW15]

### *3.1.13 Cassandra*

Cassandra původně vznikla pro sociální síť Facebook, kde se využívala při vyhledávání v příchozích zprávách (později nahrazeno HBase). [Str11] V roce 2008 byl zdrojový kód zveřejněn a nyní je k dispozici v rámci open-source licence Apache[27]. Mimo to existují také dvě distribuce od společnosti Datastax – základní Community Edition a komerční Enterprise Edition, která je doplněná o řadu sofistikovaných nástrojů pro analýzu dat a optimalizaci výkonu[28]. Společnost Datastax nabízí také výukové kurzy a další komerční podporu pro svoji Entreprise distribuci Cassandry.

---

[21] Hadoop Distributed Filesystem. Souborový systém pro distribuované zpracování souborů. Je součástí frameworku Apache Hadoop.
[22] https://hadoop.apache.org/
[23] http://wiki.apache.org/hadoop/Hbase/ThriftApi
[24] Používá se při horizontálním škálování databáze s cílem rozložit zátěž mezi více serverů a zlepšit tak rychlost přístupu k datům. Na rozdíl od replikace má každý uzel pouze část dat, nikoliv celou kopii databáze. [CodeF14]
[25] https://hive.apache.org/
[26] https://code.facebook.com/posts/321111638043166/hydrabase-the-evolution-of-hbase-facebook/
[27] http://cassandra.apache.org/
[28] http://www.datastax.com/download/dse-vs-dsc

Datový model Cassandry je kombinací mezi key/value a sloupcovým úložištěm, inspirovaný distribuovanými systémy Amazon Dynamo a Google BigTable. Jednotlivé uzly v clusteru mohou být použity i jako embedded server[29], s perzistentním ukládáním dat na lokální disk nebo jejich udržováním v operační paměti. Cassandra nemá ACID transakce, místo toho se zaměřuje na eventuální konzistenci, která poskytuje lepší škálovatelnost. Používá vlastní dotazovací jazyk Cassandra Query Language (CQL), syntakticky vycházející z SQL. V současnosti patří Cassandra mezi nejpopulárnější distribuované databázové systémy. Je využívána více než 1500[30] společnostmi, mezi nimiž jsou například eBay, Netflix, Spotify nebo Adobe. [Str11] [Dat15]

# 3.5 Grafové databáze

### *3.1.14 Neo4j*

Společnost NeoTechnology je s produktem Neo4j jedničkou na poli grafových databázových systémů. Do vývoje technologie bylo investováno přes 20 milionů dolarů, počet komerčních i open-source projektů využívajících Neo4j roste. Odhaduje se, že v roce 2017 bude Neo4j využívat alespoň 25 % všech globálních společností. Na oficiálních stránkách NeoTechnology jsou ke stažení dvě edice Neo4j – Community a Enterprise. Obě jsou dostupné zdarma pod open-source licencí. Použití Neo4j Enterprise pro komerční účely vyžaduje zakoupení licence, zdarma lze vyzkoušet 30denní trial verzi. Ukládání dat je ve formě tzv. „property graphs“, kde každý vrchol i hrana může obsahovat další údaje (vlastnosti) a může mít přidělen typ (label), který se využívá například pro indexované prohledávání grafu. Maximální velikost grafu je omezena na 34 miliard uzlů, stejný počet hran a 68 miliard vlastností. Databázový systém může být spuštěn jako server přístupný přes REST API rozhraní, nebo v embedded režimu pro lokální připojení k databázi. V distribuovaném systému může každý stroj v Neo4j clusteru fungovat jako Client-server, nebo embedded instance. Transakce provádějící čtení mohou probíhat souběžně ve více vláknech bez použití zámků. Zapisovací transakce používají pro zajištění konzistence dat zamykání vrcholů nebo hran, které modifikují. Čtecí transakce mohou přistupovat k objektům grafu i v případě, že jsou uzamčeny pro zápis. Pro práci s grafem lze použít nativní deklarativní jazyk Cypher, jehož syntaxe se velmi podobá jazyku SQL. Kromě jednoduchých CRUD operací se Cypher používá pro hledání vzorů v grafové struktuře (pattern matching dotazy), správu indexů a integritních omezení. Druhou možností je imperativní jazyk Gremlin, který využívá rozhraní Blueprints[31] a je tak nezávislý na použité grafové databázi. Celý graf je uložen perzistentně na disku, v operační paměti jsou udržovány nejčastěji zpracovávané části grafu. Transakce

[29] http://wiki.apache.org/cassandra/Embedding
[30] http://planetcassandra.org/companies/
[31] https://github.com/tinkerpop/blueprints

probíhají celé v paměti, změny se zapisují na disk až v případě úspěšného dokončení transakce. Z tohoto důvodu je dosažení optimální výkonnosti podmíněno dostatečným množstvím operační paměti v závislosti na velikosti grafu. V knihovně Neo4j jsou implementovány grafové algoritmy pro hledání nejkratších cest mezi vrcholy, včetně Dijkstrova i A* algoritmu. Hlavní využití Neo4j je v oblasti sociálních sítí a doporučovacích systémů. V komerčních projektech ji používají společnosti eBay, Wall-Mart, TomTom, nebo Cisco. [Neo4j15]

### 3.1.15 OrientDB


OrientDB[32] je hybridní datové úložiště, kombinující grafový, dokumentový a objektový model databáze. Je napsána kompletně v Javě a dostupná v open-source verzi – Community edition, nebo placené – Enterprise edition, která navíc obsahuje nástroje pro optimalizaci dotazů, online zálohování, a podporu vývoje. Každý záznam v databázi je buď dokument, uzel grafu, nebo hrana grafu, s automaticky generovaným recorID, které slouží jako ukazatel na fyzické umístění záznamu. Ke každému záznamu se ukládá také verze, která se automaticky zvyšuje s každou editací záznamu. Verzování záznamů slouží ke zjišťování konfliktů mezi transakcemi a umožňuje použití optimistického modelu transakcí, kde není potřeba zamykání záznamů při paralelním zpracování transakcí. V Javě jsou pro práci s databází k dispozici tři různé API – Graph API, které umožňuje používat OrientDB jako grafovou DB, dále document API, jehož použití umožňuje pracovat se záznamy jako s dokumenty ve formátu JSON a Object API, které provádí transparentní mapování POJO objektů na záznamy v DB, které jsou uloženy jako dokument. Ačkoliv může být OrientDB použita i jako dokumentová, nebo objektová databáze, primárně se jedná o grafový SŘBD a například Object API nebylo od verze 1.5 nijak updatováno. K dotazování se používá jazyk SQL doplněný o podporu práce s grafovými strukturami. Obsaženy jsou i grafové algoritmy pro hledání nejkratší cesty a průchod grafem. Pro pokročilejší grafové dotazy a algoritmy je nutné použít grafový dotazovací jazyk Gremlin. [Ori15]

### 3.1.16 Sparksee

Sparksee (do roku 2014 vydávaný pod názvem DEX) je komerční, bezserverový grafový databázový systém implementovaný v jazyce C++. Open-source licence není dostupná, nicméně je možné využívat Sparksee zdarma pro libovolné účely, s omezením velikosti databáze na 1 milion objektů. Společnost Sparksity nabízí zdarma plnou verzi bez omezení také pro studijní a vědecké účely. Dostupná jsou rozhraní pro použití Sparksee v jazycích C#, Java, C++, Objective-C a Python. Sparksee pracuje

[32] http://orientdb.com/

stejně jako neo4j s orientovanými multigrafy, kde každý uzel i hrana mohou obsahovat další údaje a mohou mít přiděleny typ, který se používá například při indexovaném vyhledávání. Vrcholy a hrany grafu jsou ukládány v bitmapové struktuře spolu s automaticky generovaným, unikátním identifikátorem (OID), který odpovídá pozici objektu v bitmapě. Ukládání v bitmapových strukturách umožňuje efektivní práci s grafem pomocí binárních operací, díky čemuž dosahuje Sparksee velmi dobrých výkonnostních výsledků v porovnání například s neo4j a dalšími vybranými Nosql databázovými systémy[33]. Použití bitmapových struktur je výhodné pro úsporu paměťového prostoru, neboť dosahují vysoké míry komprese dat. Manipulace s daty probíhá pomocí nativního API prostřednictvím OID objektů. Součástí jsou grafové algoritmy pro průchod grafem do šířky i do hloubky a vyhledávání nejkratší cesty s Dijkstrovým algoritmem. Databáze obsahuje rozhraní Blueprints, které poskytuje možnost práce s dotazovacím jazykem Gremlin. [Mar12] [Spark14]

# 3.6 Dokumentové databáze

### *3.1.17 iBoxDB*

iBoxDB[34] je malý databázový systém, který může být spuštěn jako in-memory databáze bez perzistentního ukládání dat, nebo s trvalým ukládáním na lokální disk. Je možné ji používat i jako TCP/IP server, který může být součástí distribuovaného systému s master-slave, nebo master-master replikací. Databázová knihovna je napsána v jazyce Java a samostatně také v C# pro použití na platformě .NET. Datový model je kombinací mezi objektovou, dokumentovou a relační databází. Objekty se ukládají do tabulek s dynamickými sloupci, databázový systém transparentně zajišťuje jejich perzistenci i objektově relační mapování. Součástí je vlastní jednoduché API umožňující základní CRUD operace s vlastním dotazovacím jazykem, který je syntakticky podobný SQL. Podporovány jsou ACID transakce a jednoduché i složené indexy.

Vhodným místem použitím pro iBoxDB jsou mobilní zařízení (platforma Android) a webové aplikace (používá se například ve webových hrách). Bohužel, není k dispozici žádná oficiální dokumentace, ani komerční podpora ze strany vývojářů iBoxDB. Na oficiálních stránkách projektu je pouze stručný návod k nasazení databáze a popis funkcionalit databázového systému. [iBo15] [uni13]

[33] http://dl.acm.org/citation.cfm?doid=2351476.2351489
[34] http://www.iboxdb.com/

### 3.1.18 EJDB

Dokumentová databáze implementačně vycházející z key value uložiště Tokyo Cabinet[35]. Jazyk implementace je C, existují oficiální rozhraní pro použití EJDB v Javě, .NET, Nodejs, Python, Ruby a některá další od třetích stran. Pro přístup k databázi z příkazové řádky je možné nainstalovat modul ejdb-node. EJDB ukládá JSON dokumenty do tzv. kolekcí. Každá kolekce je tabulková databáze, uložená v samostatném souboru, kde jsou záznamy ukládány po řádcích. Každý řádek obsahuje unikátní, systémem generovaný identifikátor UUID a vlastní JSON dokument, který je při ukládání převeden do interního BSON formátu. Zbylé sloupce řádku mohou být volitelně použity pro ukládání metadat sloužících pro zrychlení indexovaného vyhledávání, optimalizaci dotazů, nebo specifikaci přístupových práv k jednotlivým dokumentům. Dotazovací jazyk vychází ze stejných principů jako v dokumentové databázi MongoDB, díky čemuž je zajištěna přenositelnost aplikace mezi MongoDB a EJDB. Umožňuje provádět CRUD operace nad kolekcí dokumentů, obsahuje funkce pro vyhledávání řetězců, práci s regulárními výrazy a tvorbu pohledů. Transakce jsou atomické a zaručují trvalost dat. Nad kolekcí dokumentů nemůže probíhat více transakcí souběžně. Aktuální verze 1.2.7 zatím nepodporuje všechny funkce dostupné v Tokyo Cabinet. Chybí například podpora složených indexů, omezené jsou možnosti pro využití metadat. [Soft14].

# 4. HyperSQL

## 4.1 Databáze

HyperSQL pracuje s třemi typy databází:

- Souborové databáze, které jsou (částečně, nebo kompletně) uloženy na disku.
- Neperzistentní (in-memory) databáze, udržované pouze v operační paměti počítače. Databáze je přístupná pouze aplikaci, která jí vytvořila. Po ukončení spojení data zaniknou.
- databáze typu res jsou určeny pouze pro čtení. Ukládají se do jar, nebo zip souborů a distribuují jako součást Javovské aplikace.

V případě souborové databáze je na disku uloženo až 6 souborů v závislosti na konfiguraci a použitém typu tabulek. Například v adresáři s databází testovaciDB mohou být tyto soubory:

[35] http://fallabs.com/tokyocabinet/

- *testovaciDB.data* – soubor s daty uloženými v perzistentních CACHED tabulkách. Pokud v databázi nejsou žádné CACHED tabulky, soubor s příponou .data v adresáři nebude.

- *testovaciDB.properties* – soubor obsahující nastavení databázového systému. Tento soubor bude v adresáři u každé souborové databáze.

- *testovaciDB.script* – Tento typ souboru se v adresáři nachází vždy. Obsahuje definici tabulek a dalších databázových objektů, a také data MEMORY tabulek. Soubor je na disku uložen v textové podobě. Je možné jej editovat, a změnit definici schématu tabulek, nebo obsah databáze. Nicméně, při editaci souboru je potřeba dávat pozor, aby nebyla porušena integrita databáze.

- *testovaciDB.log* – obsahuje poslední změny provedené v databázi (všechny DML a DDL SQL příkazy). Pokud během transakce dojde k chybě, použije se k obnovení databáze do konzistentního stavu (provede se ROLLBACK posledních změn). Pokud k žádné chybě nedojde, jsou soubory typu .log smazány po ukončení práce s databází.

- *testovaciDB.backup* – Komprimovaná záloha posledního konzistentního stavu dat ze souboru testovaciDB.data. Používá se jen tehdy, pokud jsou v databázi nějaké tabulky typu CACHED.

- *testovaciDB.lobs* – slouží pro ukládání obsahu sloupců s datovým typem BLOB a CLOB. Pokud se žádný takový sloupec v databázi nenachází, soubor v adresáři nebude.

Mimo těchto souborů mohou existovat ještě soubory s daty pro textové tabulky (typicky CSV). Ty mohou být uloženy kdekoli na disku a přilinkovány k databázi. [Sim14]

## 4.2 Možnosti provozu

Připojení k databázi se realizuje prostřednictvím JDBC ovladače, který je součástí knihovny s databázovým systémem. Aplikace se může připojit buď k databázi v embedded režimu, nebo je databázový systém spuštěn jako samostatný serverový proces, umožňující sdílení databáze mezi více JVM procesy. [Sim14]

### *Embedded*

Embedded režim je nejrychlejší, protože poskytuje přímý přístup k databázi místo síťové komunikace. Nevýhodou je, že neumožňuje souběžné sdílení databáze více procesy. Nelze ani prohlížet databázové soubory pomocí nástroje DatabaseManager, pokud ve stejnou chvíli pracuje s databází jiný program. Je však možné spustit databázi v režimu pouze pro čtení, který dovoluje prohlížení databázových souborů z více procesů současně i v embedded režimu. Pokud je embedded databáze spuštěna v neperzistentním (in-memory) režimu, ostatní procesy databázi ani neuvidí.

### *HyperSQL Server*

Tento typ serverového procesu je autory doporučován pro běžné využití. Používá vlastní komunikační protokol, který je rychlejší než HTTP komunikace použitá v ostatních serverových režimech provozu.

### *HyperSQL HTTP Server*

Webový server, ke kterému se JDBC klienti připojují prostřednictvím protokolu HTTP. Používá se v situacích, kdy není možné (například kvůli nastavení firewallu) použít proprietární komunikační protokol. V ostatních případech je jeho použití nevýhodné z důvodu vyšší režie HTTP komunikace oproti klasickému HyperSQL serveru.

### *HyperSQL HTTP Servlet*

Tato metoda přístupu se používá, pokud je přístup k databázi řízen speciálním servletem, nebo aplikačním serverem (například Apache Tomcat ). Komunikační protokol je opět HTTP.

V případě serverových procesů je vhodné zabezpečit databázi proti nechtěnému přístupu. HSQL umožňuje vytvoření seznamu IP adres povolených pro připojení k serveru (ACL), a také šifrování síťové komunikace protokolem TLS. Server může být spuštěn samostatně z příkazové řádky, nebo je možné vytvořit a spustit instanci serveru v rámci aplikace. Podrobnější informace o spouštění a konfiguraci serveru obsahuje kapitola 13 oficiální dokumentace HyperSQL.

## 4.3 Tabulky

SQL standard rozlišuje dva druhy tabulek – dočasné (temporary) tabulky, jejichž obsah je uchováván pouze po dobu spojení s databází a trvalé tabulky, uchovávající data i po ukončení spojení. HSQL pracuje s těmito typy tabulek [Sim14]:

### *Dočasné tabulky*

Existují dva různé typy dočasných tabulek – lokální a globální. Rozdíl mezi nimi je ten, že definice schématu globální tabulky je v databázi uložena trvale, zatímco lokální tabulky po ukončení spojení zaniknou. Každé spojení má vlastní kopii dočasné tabulky, ve které vidí pouze vlastní data. Po ukončení spojení je obsah dočasných tabulek vymazán. Ve výchozím nastavení jsou záznamy dočasných tabulek uchovávány pouze v operační paměti. Tuto vlastnost je možné změnit nastavením parametru SESSION RESULT MEMORY ROWS, jehož hodnota udává maximální počet řádků uložených v operační paměti. Každý další řádek dočasné tabulky bude uložen na disku.

### *Perzistentní tabulky typu MEMORY*

Jsou výchozím typem tabulek, pokud použijeme standardní SQL příkaz CREATE TABLE. Všechna jejich data jsou uložena v operační paměti a po ukončení spojení zanikají. Persistence dat je zajištěna uložením definice tabulky do souboru *.script a zaznamenáním všech nad nimi provedených SQL dotazů do souboru s příponou .log. Po každém otevření databáze jsou tyto soubory použity pro znovuvytvoření a naplnění MEMORY tabulek. Stejně tak při každém ukončení spojení musí být do těchto souborů zaznamenány všechny provedené změny. Z tohoto důvodu může být spouštění a ukončování databáze, která obsahuje tento typ tabulek, výrazně pomalejší (v závislosti na velikosti MEMORY tabulek).

### *Perzistentní tabulky typu CACHED*

Pouze část dat a indexů je udržována v paměti, zbytek je uložen na disku v souboru s příponou .data. Ve výchozím nastavení je v operační paměti udržováno maximálně 50 000 řádek, pokud nepřesáhnou velikost 10 000 kB. Maximální počet řádků v paměti lze změnit SQL příkazem SET FILES CACHE ROWS, maximální celkovou velikost ovlivňuje příkaz SET FILES CACHE SIZE. Použití CACHED tabulek je výhodné použít pouze v případě, že jsou příliš velké a nelze je udržovat v operační paměti. Práce s nimi je výrazně pomalejší oproti MEMORY tabulkám (s výjimkou spuštění a ukončení databáze). CACHED a MEMORY tabulky mohou být použity společně v jedné databázi. Z hlediska rychlosti práce s databází je vždy výhodné definovat malé tabulky jako MEMORY a pro ostatní použít typ CACHED. V in-memory modu pracuje databáze s CACHED i MEMORY tabulkami, jako s neperzistentní verzí MEMORY tabulek (nejsou zaznamenávány prováděné změny do souboru *.log a tabulky nelze po opětovném připojení k databázi obnovit).

Ve výchozím nastavení má HyperSQL výkonnostní problémy při zapisování do větších databází s CACHED tabulkami. Při snaze naplnit databázi o velikosti přibližně 1GB s pěti milióny řádků, se nepodařilo naplnit tabulky ani po 7 hodinách čekání, přestože například SQLite zvládne vytvořit stejnou souborovou databázi do dvou minut. Možností jak zrychlit zápisy do CACHED tabulek je zvýšení hodnoty parametru NIO. Ten udává maximální velikost souborové databáze [MB], pro kterou bude HSQL pracovat s databází pomocí metod z rozhraní Java NIO[36]. Výchozí nastavení je 256 MB, a když databáze tuto velikost přesáhne, rychlost zápisových operací dramaticky klesá. Po zvýšení hodnoty parametru NIO na 8192 trvalo vytvoření shodné databáze necelých 7 minut. Další zrychlení lze dosáhnout dočasným vypnutím kontroly referenční integrity, případně zakázáním paralelního zápisu změn v databázi do souboru .log nastavení parametru FILES LOG na FALSE. Pro nastavení parametru NIO nad hodnotu 4192, je vyžadován 64bitový Javovský virtuální stroj.

[36] http://docs.oracle.com/javase/7/docs/technotes/guides/io/

***Perzistentní tabulky typu TEXT***

Jako zdroj dat používají CSV soubory (nebo jiné soubory s hodnotami oddělenými specifickým symbolem). Při práci s textovými tabulkami udržuje databázový systém v paměti všechny jejich indexy a malou část dat. U každé textové tabulky je možné nastavit maximální počet řádků udržovaných v paměti (proměnná *cache_rows*), případně maximální celkovou velikost tabulky v paměti (proměnná *cache_size*). Ve výchozím nastavení databáze není možné používat textové tabulky v in-memory režimu. Práce s textovými tabulkami je podrobně popsána v oficiální dokumentaci v kapitole 5.

## 4.4 Další nástroje

V archivu s databázovým systémem HyperSQL je také aplikace sqltool, která poskytuje jednoduché rozhraní pro provádění SQL dotazů a skriptů z příkazové řádky. Sqltool umožňuje univerzální připojení k databázi přes JDBC rozhraní, a může být použit s libovolným databázovým systémem, který umožňuje JDBC spojení. V kombinaci s HSQL lze sqltool používat v embedded (in-memory i perzistentní) i klient-server režimu databáze. Pokud je knihovna hsqldb.jar umístěna ve stejném adresáři s knihovnou sqltool.jar, stačí pro připojení k databázi zadat správnou JDBC adresu.

Přímou součástí knihovny hsql je GUI nástroj Database Manager. Má podobné funkcionality jako Sqltool a existuje ve dvou různých verzích – první je vytvořena s použitím nástrojů knihovny AWT, ta druhá používá novější knihovnu SWING. Hlavní rozdíl mezi nimi je, že verze s knihovnou AWT může být použita jako součást Java appletu, který je přístupný přes webový prohlížeč.

Hsqldb nabízí ještě dva další užitečné nástroje – grafický Transfer Tool umožňující přesun dat mezi libovolnými databázemi s JDBC rozhraním a pomocný program testUtil obsahující skripty předpřipravené skripty k testování funkcí databázového systému. [Simp15]

## 4.5 Víceuživatelský přístup

V embedded režimu se k databázi může připojit pouze jeden proces (pokud není databáze spuštěna pouze v režimu pro čtení), ale může provádět paralelní transakce v různých vláknech. V serverovém režimu je možné sdílet databázi více procesy a ty mohou být rovněž vícevláknové. HSQL podporuje tři modely řízení transakcí.

Výchozí transakční model 2PL (two-phase-locking), používá zámky na úrovni tabulek. Pro čtení stačí získat sdílený zámek, který může být držen současně více vlákny nebo procesy. Modifikace tabulky vyžaduje exkluzivní zamčení tabulky během provádění zápisové transakce. Druhý transakční model – MVLOCKS znovu používá

zámky na úrovni tabulek. Zápisové transakce mohou probíhat současně s čtecími, protože každá čtoucí transakce vidí vlastní „snímek“ databáze, obsahující aktuální konzistentní data před začátkem transakce, zatímco zápisové transakce mohou modifikovat původní data. Třetím použitelným modelem je MVCC (Multiversion concurrency cotrol). V tomto případě nejsou po čtecích operacích vyžadovány žádné sdílené zámky. Zápisové transakce stále potřebují exkluzivní zámky, ale pouze na úrovni modifikovaného záznamu. Prohlížení i modifikace tabulky může probíhat souběžně. Zapisování do jedné tabulky může také probíhat souběžně, pokud se netýká stejného řádku. [Sim 14]

# 5. SQLite

## 5.1 Dotazování

Dotazovací jazyk v SQLite nabízí většinu funkcí ze standardu SQL-92. Nepodporované jsou následující [Sqlite15]:

- Pravé vnější spojení a úplné vnější spojení tabulek (RIGHT OUTER JOIN a FULL OUTER JOIN). Naopak levé vnější spojení (LEFT OUTER JOIN) implementováno je.

- Většina variant příkazu ALTER TABLE. SQLite umí pouze přejmenování tabulky (RENAME) a přidání nového sloupce do existující tabulky (ADD COLUMN).

- Triggery (spouštěče) mohou být přiřazeny pouze na úrovni řádků (FOR EACH ROW), nikoliv pro celé příkazy (FOR EACH STATEMENT)[37].

- Databázové pohledy jsou určeny pouze pro čtení a nemohou na ně být použity příkazy DELETE, UPDATE, INSERT. Nicméně, je možné vytvořit spouštěče reagující na pokus o modifikaci pohledů a potřebné příkazy v nich vykonat.

- Příkazy GRANT a REVOKE. Vzhledem k tomu, že SQLite nemůže pracovat v režimu klient-server a k databázi se tak dostanou pouze lokální aplikace, nejsou příkazy pro přidávání a odebírání práv potřeba.

## 5.2 Datové typy

Na rozdíl od většiny ostatních relačních databází, používá SQLite dynamické datové typy. Do každého sloupce (s výjimkou primárního klíče) lze uložit libovolnou hodnotu, bez ohledu na datový typ uvedený při vytváření tabulky. Vkládaná hodnota je pak převedena na příbuzný (afinní) datový sloupec, kterých SQLite rozeznává celkem pět. Afinní datové typy nejsou pevným omezením množiny hodnot přípustných v daném sloupci, ale pouze doporučenou hodnotou. Databázový systém se automaticky pokusí převést vkládanou hodnotu na příbuzný datový typ (viz Tabulka 5.1), ale pokud hodnota nemůže být převedena, uloží ji v nezměněné podobě. Autoři SQLite považují tuto vlastnost za přednost, nicméně pro zachování konzistence dat a přenositelnosti aplikace

[37] Rozdíl je ten, že spouštěče na úrovni řádků jsou aktivovány samostatně pro každý záznam tabulky (například pokud je v rámci jednoho SQL příkazu manipulováno s tisícem řádků, spustí se tisíckrát), zatímco spouštěč na úrovni příkazu je aktivován vždy jen jednou, bez ohledu na počet zpracovávaných řádků.

je nutné vkládat do jednotlivých sloupců pouze předem definované datové typy. SQLite také neomezuje délku datových typů, která se uvádí v závorce za jejich názvem (například do sloupce s datovým typem VARCHAR(20), povolí uložení řetězce s délkou přesahující 20 znaků).

| Příklady SQL datových typů | Afinní typ SQLite |
|---|---|
| INT, INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT UNSIGNED BIG INT, INT2, INT8 | INTEGER |
| CHARACTER(20), VARCHAR(255),VARYING CHARACTER(255), NCHAR(55), NATIVE CHARACTER(70), NVARCHAR(100), TEXT, CLOB | TEXT |
| BLOB | NONE |
| REAL, DOUBLE, DOUBLE PRECISION, FLOAT | REAL |
| NUMERIC, DECIMAL(10,5), BOOLEAN, DATE DATETIME | NUMERIC |

***Tabulka 5.1**: Převody mezi datovými typy v SQLite (Zdroj: [Sqlite15]).*

### *Datum a čas*

Datový typ pro uložení data a času SQLite neobsahuje. Nabízí pouze funkce pracující s datem a časem uloženým jako číslo v afinních datových typech REAL, INTEGER, nebo ve formě textového řetězce (datový typ TEXT).

## 5.3 Zamykání

SQLite umožňuje sdílení databáze více vlákny v jednom procesu i současné připojení více procesů. Aby se předešlo poškození dat při souběžném zpracování transakcí, používá databázový systém zamykání databáze. Z hlediska procesů je rozlišováno pět různých stavů, ve kterých se databáze může nacházet [Sqlite15]:

- UNLOCKED – V databázi nejsou žádné procesy, je odemčená pro libovolnou operaci.
- SHARED – Sdílený zámek umožňuje procesům číst databázi, ale nesmí jí modifiko-vat. Libovolné množství procesů může držet sdílený zámek souběžně.
- RESERVED – Tento typ zámků používají procesy, které v budoucnu plánují modifikaci databáze, ale momentálně provádí čtecí operace. RESERVED se od PENDING liší tím, že další procesy mohou získat sdílený zámek a číst databázi.
- PENDING – Proces chce provést modifikaci databáze a čeká na dokončení probíhajících zápisových transakcí, aby mohl zamknout databázi výhradně pro sebe. V tomto stavu je databáze uzamčena pro přístup všech dalších transakcí.

- EXCLUSIVE – Databáze je exkluzivně zamčena pro zápis. Všechny ostatní procesy musí čekat.

### *Žurnál*

Pro zajištění konzistence databáze používá SQLite žurnálovací soubor, do kterého každá transakce nejprve uloží původní obsah databáze, než jej začne modifikovat. Pokud v průběhu transakce dojde k chybě, bude žurnál označen jako „hot“. Před čtením databáze každá transakce nejprve zkontroluje stav žurnálu, a pokud nalezne „hot“ žurnál, použije ho pro vrácení všech změn v databázi do původního stavu. Pokud transakce proběhne úspěšně, je žurnál smazán.

### *Write-Ahead-Log*

Hlavním problémem spojeným s použitím žurnálovacího souboru je nemožnost čtení dat, pokud jiný proces do databáze zapisuje. Toho lze docílit přepnutím databáze do write-ahed-log (WAL) modu, kdy se místo žurnálu se zálohou konzistentního stavu databáze používá tzv. write-ahead-log, do kterého se zapisují plánované změny v databázi. Transakce modifikující databázi zapisují vždy do WAL souboru, zatímco původní databázový soubor je ostatním transakcím dostupný pro čtení. Po určité době (standardně po každých tisíci stránek paměti) proběhne tzv. checkpoint, kdy je obsah všech transakcí WAL souboru uložen do databáze.

WAL režim je výrazně rychlejší pro zápisové operace, protože nejsou blokovány transakcemi, které prohlížejí databázi. Naopak čtecí operace jsou pomalejší, protože kromě databáze musí prohledávat také WAL soubor. Změny jsou vždy zapisovány na konec WAL souboru, takže jeho čtení i modifikace může probíhat souběžně. Zapisovat do WAL souboru může vždy jen jeden proces.[Sqlite15]

## 5.4 Pragma příkazy

Speciální příkazy umožňující prohlížení a nastavení specifických proměnných knihovny SQLite, jejichž prostřednictvím lze ovládat vlastnosti databázového systému. Existují dva druhy PRAGMA příkazů – trvalé, které jsou uloženy v databázi a ovlivňují každé spojení a dočasné, které trvají pouze po dobu připojení k databázi a musí se po každém připojení provádět znovu. Celkem obsahuje knihovna SQLite asi 60 různých PRAGMA příkazů. Některé z nich jsou zachovány pouze z důvodu kompatibility a jejich použití v nových aplikacích se nedoporučuje. Seznam všech PRAGMA příkazů používaných SQLite včetně vysvětlení a příkladů lze nalézt v oficiální dokumentaci[38]. Zde uvádím pro příklad následující [Sqlite15]:

[38] https://www.sqlite.org/pragma.html#pragma_automatic_index

- Nastavení case-sensitive porovnávání řetězců u dotazů s klauzulí LIKE. (Výchozí nastavení nerozlišuje velikost písmen):

```
PRAGMA case_sensitive_like = [true | false];
```

- Zapnutí podpory cizích klíčů:

```
PRAGMA foreign_keys = [ON | OFF];
```

- Výpis všech databázových indexů vytvořených pro konkrétní tabulku:

```
PRAGMA database.index_list(table-name);
```

***Cizí klíče***

Ve výchozím nastavení nekontroluje databázový systém dodržení referenční integrity i přesto, že jsou cizí klíče definovány při vytváření tabulek. Automatickou kontrolu cizích klíčů je nutné aktivovat výše uvedeným PRAGMA příkazem. Tento příkaz není v databázi uložen trvale a je nutné jej opakovat po každém připojení. Rovněž není možné aktivovat kontrolu cizích klíčů v rámci transakce s více příkazy – v takovém případě je zmíněný PRAGMA příkaz ignorován. Bez kontroly cizích klíčů je v praxi téměř nemožné udržet integritu databáze, proto je vhodné provádět zmíněný příkaz ihned po připojení k databázi.

***Wal***

Ve výchozím nastavení používá SQLite pro zajištění konzistence dat žurnálovací soubor. Přepnutí do WAL režimu je možné provést PRAGMA příkazem:

```
PRAGMA journal_mode=WAL
```

## 5.5 Nástroje

Z oficiálních stránek je možné stáhnout program sqlite3 (dostupný pro OS Windows i Linux), který umožňuje připojení k databázi z příkazového řádku a provádění SQL dotazů. Obsahuje také sadu speciálních příkazů, například pro zálohování databáze, import/export tabulek z/do CSV souborů, změnu výstupního formátu dotazů a některé další, dohledatelné v oficiální dokumentaci[39]. Tyto příkazy vždycky začínají tečkou a na rozdíl od SQL dotazů se neukončují středníkem [Sqlite15].

Pro připojení k existující databázi slouží příkaz ve tvaru sqlite3 *název databáze.* Pokud databáze zadaného názvu neexistuje, program automaticky vytvoří novou databázi zadaného názvu.

[39] https://www.sqlite.org/cli.html

```
$ sqlite3 ex1
SQLite version 3.8.5 2014-05-29 12:36:14
Enter ".help" for usage hints.
sqlite> create table tbl1(one varchar(10), two smallint);
sqlite> insert into tbl1 values('hello!',10);
sqlite> insert into tbl1 values('goodbye', 20);
sqlite> select * from tbl1;
hello!|10
goodbye|20
sqlite>
```

**Příklad 5.1:** *Práce s SQLite databází pomocí nástroje sqlite3*

Je potřeba dát pozor na to, že ve výchozím nastavení pracuje program sqlite3 s in-memory databázemi, takže veškeré změny jsou uloženy pouze v operační paměti. Perzistentní uložení na disk zajistí příkaz .save *název souboru*

Uživatelé, kteří neovládají dotazovací jazyk SQL, mohou pracovat s SQLite databázovými soubory s GUI nástrojem SQLite Database browser[40]. Je dostupný zdarma v rámci GNU GPL licence a umožňuje vytváření, modifikaci a prohledávání databáze, import/export tabulek mezi CSV soubory, vše pomocí jednoduchého grafického rozhraní. Může být použit na platformě Windows, Mac i Linux

[40] http://sqlitebrowser.org/

# 6. Derby

### *5.1.1 Distribuce*

Již bylo zmíněno, existují dvě různé distribuce databázového systému Derby. Původní Apache Derby vycházející ze zdrojového kódu projektu Cloudscape společnosti IBM. A dále distribuce na jejímž vývoji se podílí společnost Oracle, a která je součástí standardního JDK od verze 1.6, kde je distribuována pod názvem JavaDB. V této práci používám distribuci získanou z oficiálních stránek projektu Softwarové nadace Apache.[41] Obě distribuce by měly být z hlediska funkcionality naprosté shodné, výhodou distribuce od společnosti Oracle je, že k ní lze získat placenou uživatelskou podporu v rámci předplacené podporu vývoje k balíku JDK.

### *5.1.2 Možnosti provozu*

Databázový stroj může fungovat na principu architektury Klient-server, kdy se aplikace prostřednictvím JDBC ovladače připojuje k síťovému databázovému serveru, který obsluhuje jednotlivé požadavky. Síťová komunikace s klientskými aplikacemi probíhá pomocí TCP/IP a protokolu DRDA Druhou možností fungování je embedded architektura, kdy je databázový systém spuštěn ve stejném Javovském virtuálním stroji ve kterém běží aplikace, která s ním pracuje. Vlastní komunikace je opět realizována přes JDBC rozhraní. V klient-server i embedded režimu může být databáze perzistentně uložena na lokální diskové úložiště, nebo jsou data udržována pouze v operační paměti a zanikají po ukončení spojení s databází. [Apa14]

### *5.1.3 Výkonnost systému*

Databáze Derby je obecně považována za málo výkonnou a to při většině prováděných operací. Nicméně existuje několik možností, kterými je možné výkonnost výrazně ovlivnit. [San07]:

- Změnou hodnoty parametru *pageCacheSize*, který udává velikost stránek paměti používaných databázovým systémem pro udržování nejčastěji zpracovávaných záznamů v RAM paměti. Standardní nastavení je 1000 stránek paměti, každá o velikosti 4KB. Celková počáteční velikost paměťového bufferu, je tedy pouze 4 MB, což může být pro mnoho aplikací nedostačující. Pokud chceme dosáhnout lepší výkonnosti během zpracování dat a máme k dispozici dostatečné množství operační paměti, je vhodné nastavení parametru upravit.

41 http://db.apache.org/derby/derby_downloads.html#Latest+Official+Releases

- Použití předkompilovaných SQL dotazů pomocí třídy *PreparedStatements()*. To je vhodné zejména v aplikacích, které vykonávají velké množství stejných SQL dotazů, častá kompilace dotazů může výrazně snižovat výkonnost systému.
- Použití obsáhlejších transakcí místo jednotlivých SQL dotazů a deaktivace automatického COMMITU změn v databázi, který je ve výchozím nastavení prováděn po každém příkaz. Databáze Derby je při častém provádění COMMIT výrazně pomalejší.
- Pokud je to možné měla by být prováděna indexace tabulek. Především u rozsáhlých tabulek může použití indexů zrychlit vyhledávání v tabulce o několik řádů. Na druhé straně, udržování velkých indexů vyžaduje dostatek operační paměti

## 6.1 Nástroje

Kromě JDBC ovladače se lze k databázovému systému připojit také jednoduchým nástrojem *ijtool,* se kterým lze spouštět SQL dotazy a skripty z příkazové řádky. Knihovna s nástrojem *ijtool, Derbytools.jar* je součástí Apache distribuce Derby a je umístěna v adresáři */lib* stejně jako knihovna s databázovým systémem. Pokud máme obě knihovny ve stejném adresáři, provede se připojení k databázi testovaciDB v Linuxu následovně:

```
export DERBY_INSTALL=/home/pc/NetBeansProjects/JavaApplication1/dist/
lib/Derby

#nastavení absolutní cesty ke knihovně derby.jar a derbytools.jar
export CLASSPATH=/home/pc/NetBeansProjects/JavaApplication1/dist/
lib/Derby/lib/derby.jar:/home/pc/NetBeansProjects/JavaApplication1/dist
/lib/Derby/lib/derbytools.jar:

#spuštění nástroje ijtool a připojení k databázi zadání JDBC adresy
.java org.apache.derby.tools.sysinfo
ij> connect 'jdbc:derby:testovaciDB';
```

**Ukázka 6.1:** Připojení k databázi testovaciDB pomocí nástroje ijtool

Podrobné příklady a možnosti práce s nástrojem ijtool lze dohledat v oficiální dokumentaci.[42]

Další podrobnosti o možnostech použití a fungování databázového systému Apache Derby, jsou uvedeny v souhrnné tabulce relačních databázových systémů A.1, která je součástí přílohy A.

[42] http://db.apache.org/derby/papers/DerbyTut/ij_intro.html

# 7. TPC-C test

Benchmarkovací program realizovaný v rámci této práce vychází z transakčního testu TPC-C. Jedná se o jeden z mnoha testů připravených konsorciem TPC (Transaction Processing Performance Council), které se specializuje na vytváření objektivně porovnatelných testů pro hodnocení výkonnosti systémů během transakčního zpracování dat [TPC15] [Krá13]. Kompletní test je značně rozsáhlý, proto se omezím pouze na popis základních vlastností testu, zejména na ty, které jsou implementovány v testovacím programu. Podrobnější informace lze získat z oficiální specifikace, která je hlavním zdrojem informací obsažených v této kapitole [TPC10].

## 7.1 Neimplementované vlastnosti

- TPC-C Benchmark simuluje víceuživatelský přístup k systému zpracovávajícímu objednávky. Ty jsou náhodně zadávány z určitého množství terminálů, přičemž platí, že každý terminál má pevně přiřazeno číslo skladu, pro který realizuje objednávky. Parametr udávající počet nakonfigurovaných terminálů (a tedy počet transakcí, které mohou probíhat současně), je zadán jako vstupní parametr spolu s počtem skladů. Protože embedded databáze typicky neumožňují paralelní transakce z více procesů (v in-memory režimu to ani není možné), je testovací program omezen pouze na sériové zpracování transakcí (terminály nejsou použity vůbec).

- Vstupy a výstupy na terminál není potřeba implementovat, protože nemají vliv na výsledky testu. Požadované zpoždění mezi jednotlivými transakcemi způsobené simulací zadávání vstupních a výstupních údajů na terminál, také nemá smysl neboť připravený test má za cíl měřit maximální možnou propustnost TPC-C.

- TPC-C test předpokládá ověření ACID vlastností databázového systému. Jedná se o poměrně náročnou sadu testů, která navíc ACID vlastnosti neprokáže, protože to ani není s konečnou množinou testů možné. Pro potřeby implementovaného testu je třeba se spokojit s deklarací autorů jednotlivých SŘBD, že jejich transakce splňují ACID vlastnosti. (V případě databáze H2 nejsou transakce plně ACID, neboť není trvalost dat v případě výpadků napájení.)

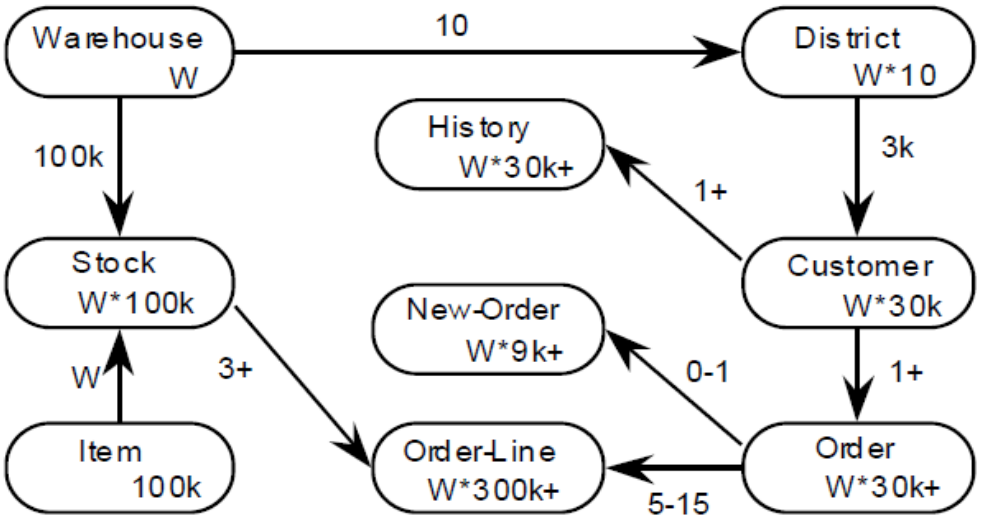
- TPC-C benchmark se dále zabývá také hodnocením pořizovací ceny systému, s čímž souvisí množství dalších požadavků (například požadavek na výpočet 60denního prostoru, kdy je vyžadováno, aby systém byl schopen obsáhnout množství dat odpovídající 8hodinovým 60 dnům běhu systému).

- Transakce doručení by měla být vykonávána po dávkách v tzv. odloženém modu (deferred mode). Tuto funkcionalitu většina open-source databází ani nepodporuje.

## 7.2 Testovací databáze

Testovací databáze obsahuje celkem devět tabulek a je plně škálovatelná. Hlavní jednotkou ovlivňující velikost výchozí databáze je počet záznamů v tabulce Warehouse. Velikosti všech ostatních tabulek, jsou závislé na hodnotě parametru W. Jedinou výjimkou je tabulka, která obsahuje vždy 100 000 záznamů. Požadavky na počet záznamů v jednotlivých tabulkách ilustruje následující E-R diagram. Symbol (+) u kardinality některých vztahů a tabulek označuje, že kardinalita vztahů a tabulek může je ovlivněna náhodným generováním a může v každé počáteční databázi mírně lišit.

Počet atributů tabulek a jejich datové typy je zachován podle požadavků specifikace. Liší se ovšem způsob generování testovacích dat, protože specifikace testu vyžaduje dodržení maximálních odchylek středních hodnot při náhodném generování dat. V implementaci programu je pro generování pseudonáhodných čísel použita třída *java.util.Random*, která generuje pseudonáhodné posloupnosti s rovnoměrným rozdělením pravděpodobnosti – požadované odchylky proto nebudou vždy dodrženy. Pro některé typy dat je ve specifikaci požadováno generování s nerovnoměrným rozdělením pravděpodobnosti. I v tomto případě je v programu použit generátor s normálním rozdělením.




***Obrázek 6.2.1**: E-R diagram počáteční databáze TPC-C testu. (Zdroj: [TPC10])*

Úplný ERA diagram testovací databáze je umístěn v příloze C.

## 7.3 Business Transakce

TPC-C test je simuluje systém zachycující průběh zpracování nové objednávky od jejího zadání náhodným zákazníkem až po konečné doručení objednávky. V testu je použito celkem 5 různých typů transakcí, které se označují jako business transakce. Každá z nich obsahuje několik SQL dotazů a představuje určitou část procesu zpracování nové objednávky. Protože v implementaci testu nejsou simulovány terminály obsluhující dotazy na jednotlivé sklady, jsou čísla skladu generována náhodně v rozsahu parametru W testovací databáze.

- *Transakce „Nová objednávka“*
  Jedná se o velmi často prováděnou transakci, která představuje hlavní zátěž systému. Provádí čtecí i zápisové operace a Je navržena aby způsobila nerovnoměrné zatížení systému, které je typické v systémech pro online zpracování transakcí.
- *Transakce „Doručení“*
  Dávkové zpracování deseti nových objednávek. Transakce by podle specifikace měla být zpracována v odloženém modu.
- *Transakce „Množství na skladě“*
  Zjišťuje úroveň zboží na skladě u položek, které byly z poslední doručených objednávek, s cílem zjistit jestli se na skladě nachází zboží pod minimální hranicí zásob. Transakce neprovádí žádné zápisové operace.
- *Transakce „Stav objednávky“*
  Opět pouze čtecí transakce, která má za úkol zjistit, jestli objednávka náhodně vybraného zákazníka byla vyřízena, či nikoliv
- Transakce „Platba“
  Realizuje platbu nové objednávky od náhodně vybraného zákazníka. Měla by se vykonávat

## 7.4 Metrika

Výstupem Transakčního testu je údaj o propustnosti systému MQTh udávaný v jednotkách TpmC. Jedná se o průměrný počet Business transakcí nová objednávka, které systém zvládne zrealizovat za minutu. Počet ostatních Business transakcí není relevantní.

# 8. Implementace a průběh testu

Připravený benchmark otestuje vstupní relační databázi s využitím výše popsaných vlastností transakcí TPC-C testu. Testovací data jsou vytvořeny automaticky na základě šklálovacího parametru W před započetím měření. Způsobu spouštění aplikace a popisu struktury naměřených výsledků je věnována příloha D.

## 8.1 Fungování testovací aplikace

Spuštění programu je vhodné provádět pomocí připravených bash skriptů, které umožňují automatické spouštění většího množství předpřipravených testů. Více o vstupních parametrech a možnostech spouštění je uvedeno v příloze D. Po spuštění programu a připojení k databázi, musí být nejprve vytvořena testovací data, podle schématu databáze uvedeného v příloze B. Pokud se test spouští v již existující databázi, jsou všechny obsažené tabulky z předchozích TPC-C testů smazány a vytvořeny znovu, aby byl zaručen stejný počáteční stav databáze v každém testu. Doba Plnění testovací databáze (konkrétně doba trvání metody *executeBatches()*, která provádí dávkové zpracování jednotlivých INSERT příkazů, je změřena s přesností na milisekundy a zapsána do logovacího souboru *InsertsLog.csv* spolu s dalšími údaji odlišujícími jednotlivá měření.

Po úspěšném vytvoření testovacích dat se automaticky spustí transakční test. Ten vždy nejprve vygeneruje TPC-C transakci která se má vykonat (viz. transakční mix) a poté zavolá příslušnou metodu obsluhující průběh zvolené transakce. Každá z TPC-C Business transakcí je tvořena více SQL příkazy, přičemž trvalé uložení změn do databáze příkazem COMMIT má být prováděno vždy až po dokončení celé jedné business transakce. Z tohoto důvodu je potřeba zajistit aby v JDBC ovladači nebyl nastaven automatický COMMIT změn po každém SQL dotazu.

Po provedení každých deseti business transakcí je ověřováno, jestli neuplynul čas vyhrazený pro běh testu. Interval ověřování času po každých deseti transakcích byl zvolen pozorováním. Jedná se o hodnotu, při které v žádném měření nedocházelo k nepřesným výsledkům v podobě nulových časů, které by byly zaznamenány v případě, že by databázový systém vykonal posloupnost transakcí příliš rychle (tj. nebylo by možné zachytit čas vykonávání s přesností na milisekundy). V průběhu testu jsou zaznamenávány dvě hodnoty – průběžná hodnota počtu realizovaných nových objednávek za každou minutu, která se zapisuje do souboru *IntervalLog.csv*, a celkový počet transakcí „Nová objednávka“ po skončení testu, který je ukládán do souboru *TpccLog.csv*. Po skončení programu ještě proběhne zpracování zmíněných logů skriptem *ProcessLog.sh*, který spočte souhrnné výsledky souvisejících měření (například průměrnou hodnotu MQTh u všech měření HyperSQL databáze se stejnou

hodnotou škálovatelného parametru W). Výstupy programu jsou dále popsány v příloze D.

## 8.2 Testovací konfigurace

Benchmark byl spouštěn v konzolovém režimu operačního systému Linux, který dovoluje větší kontrolu nad množstvím běžících procesů a je celkově mnohem méně náročný na systémové zdroje oproti OS Windows. Měření bylo provedeno na notebooku s následující HW a SW konfigurací:

| | |
|---|---|
| PC | Notebook Acer Aspire Aspire-5750G |
| Operační systém | GNU/Linux Lubuntu 13.1 64bitový |
| Procesor | Intel Core i7 2630QM s frekvencí 2,0 GHz, 6MB L2 |
| HDD | 750 (TOSHIBA MK7559GSXP, 5400rpm, 8MB cache) |
| RAM | 4096 DDR3-1066 (1x4096) |

***Tabulka 7.2.1:*** *Testovací konfigurace*

Všechny databázové systémy byly testovány ve výchozím nastavení. Nebyla prováděna indexace tabulek, ani další změny, které by mohly ovlivnit dosažené výsledky s výjimkou následujících:

- V souborovém režimu HyperSQL jsou použity CACHED tabulky místo výchozích memory tabulek. Ty totiž udržují data v paměti a na disk zapisují pouze SQL dotazy potřebné k jejich znovuvytvoření. Pro srovnání výkonových výsledků s ostatními SŘBD, které ukládají data perzistentně na disk, se memory tabulky nehodí.
- V souborovém režimu HyperSQL je nastavena hodnota parametru NIO na 8192. Je to z důvodu nepřijatelné doby vytváření větších databází při výchozím nastavení parametru více viz 4.3.
- Dále je v souborovém režimu HyperSQL zakázáno vytváření souboru .log. I tato volba výrazně zrychluje vytvoření testovací databáze
- V obou testovaných konfiguracích je po připojení k databázi SQLite zapnuta kontrola referenční integrity.

## 8.3 Parametry testování

V rámci benchmarku byly databázové systémy podrobeny transakčnímu testu vycházejícímu ze specifikace TPC-C testu. Měřeny jsou hodnoty propustnosti MQTh, udávané jako průměr počtu přijatých nových objednávek za minutu [TpmC]. U všech databázových systémů byly samostatně testovány dvě různé konfigurace – in-memory režim databáze, kdy jsou veškerá data udržována pouze v operační paměti a souborový

(perzistentní) režim, kdy databáze pracuje s daty uloženými trvale na lokálním diskovém úložišti. Jako doplňující testové kritérium byla měřena rychlost operace vkládání dat při plnění testovací databáze. Opět zvlášť v in-memory a souborovém režimu.

Test je škálovatelný parametrem *W*, který ovlivňuje počáteční velikost testovací databáze. V obou konfiguracích testu byly měřeny hodnoty MQTh pro čtyři různé hodnoty parametru *W*. Pro zajištění co nejvyšší porovnatelnosti naměřených výsledků byla pro každý test použita stejná hodnota parametru *SEED*, která se používá pro inicializaci generátoru pseudonáhodných čísel. Stejná hodnota zajistí identickou posloupnost generovaných pseudonáhodných dat (tj. u všech porovnávaných testů měly všechny SŘBD naprosto shodná počáteční data, nad kterými probíhala identická posloupnosti transakcí).

### *8.3.1 Postup a vyhodnocení transakčního testu*

V souborovém režimu databáze probíhalo měření každého systému nepřetržitě 120 minut. V případě in-memory databází, bylo z důvodu vysoké paměťové náročnosti měření rozděleno na dvanáct desetiminutových intervalů. V každém z měřených intervalů byla použita jiná hodnota vstupního parametru *SEED*. Naopak v rámci jedné sady testů prováděné na různých SŘBD byla použita stejná posloupnost hodnot vstupního parametru *SEED* (tj. byla zajištěna shodná vstupní data). Desetiminutový interval jednotlivých měření byl zvolen proto, aby předcházel častým Garbage kolekcím, ke kterým docházelo při delším testování databází s větší počáteční velikosti, což výrazně ovlivňovalo výsledky testu.

### *8.3.2 Postup a vyhodnocení testu rychlosti naplnění databáze*

Měření opět proběhlo samostatně pro dvě konfigurace databázových systémů – in-memory a souborové databáze. V obou případech proběhlo vždy deset měření pro každý databázový systém a jednu hodnotu škálovatelného parametru *W*. Měřeny byly stejné velikosti databází jako v případě transakčního testu tedy postupně $W = 1,2,3,4$ pro in-memory režim databáze a $W = 1,2,5,10$ u souborových databází. Při všech měřeních byla v tomto případě hodnota parametru *SEED* nastavena na 1. Z výsledků měření byla pro další porovnání použita průměrná hodnota vložených záznamů za 1 milisekundu a také převrácená hodnota v podobě celkového času potřebného pro naplnění testovací databáze v sekundách.

# 9. Výsledky měření

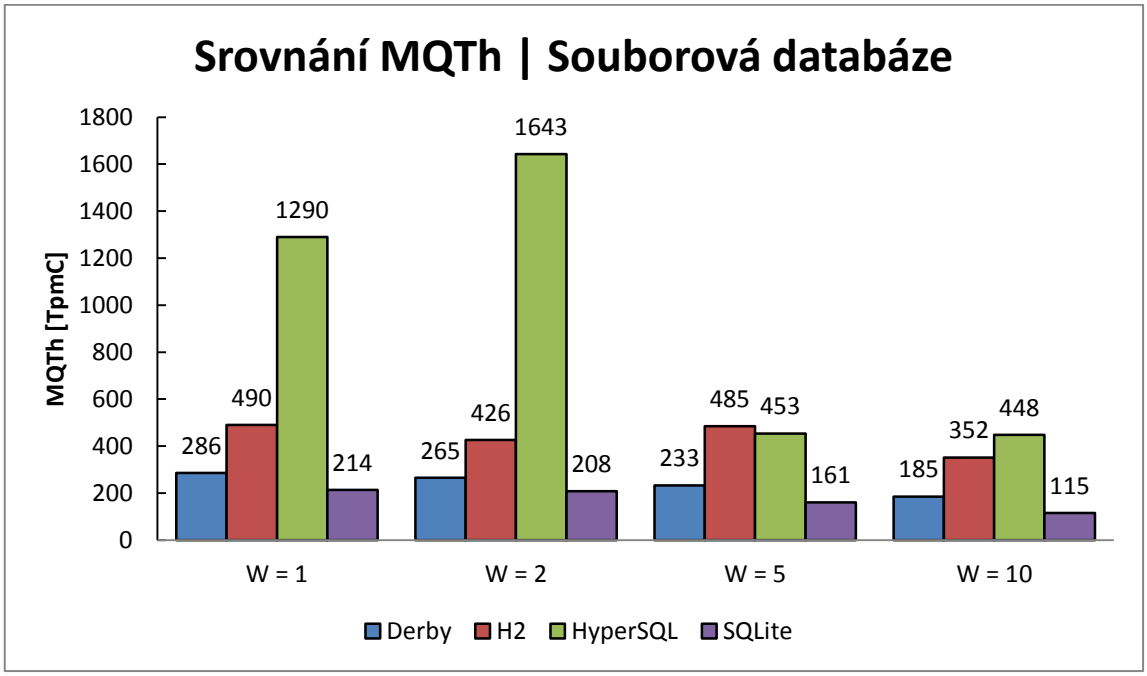
V této části jsou graficky zpracovány naměřené hodnoty propustnosti TPCC-C testu a doby plnění počáteční testovací databáze. Grafy jsou rozděleny podle režimu provozu databázového systému, pro každou z měřených veličin jsou uvedeny dva grafy

s totožnými hodnotami – v prvním jsou výsledky seskupeny podle parametru velikosti počáteční databáze *W*, druhý graf obsahuje výsledky seskupené podle databázového systému, řazené vzestupně dle parametru *W*. Zatímco v případě grafů znázorňujících výsledky dosažené v TPC-C testu je žádoucí co nejvyšší hodnota udávaná v TPm, v případě porovnání časů plnění testovací databáze je tomu naopak – větší hodnota znamená delší dobu potřebnou k naplnění databáze, a tedy horší výsledek.

Další grafy znázorňující vývoj měřené propustnosti TPC-C u souborových databází v čase, jsou uvedeny v příloze E.

## 9.1 Souborový režim databáze

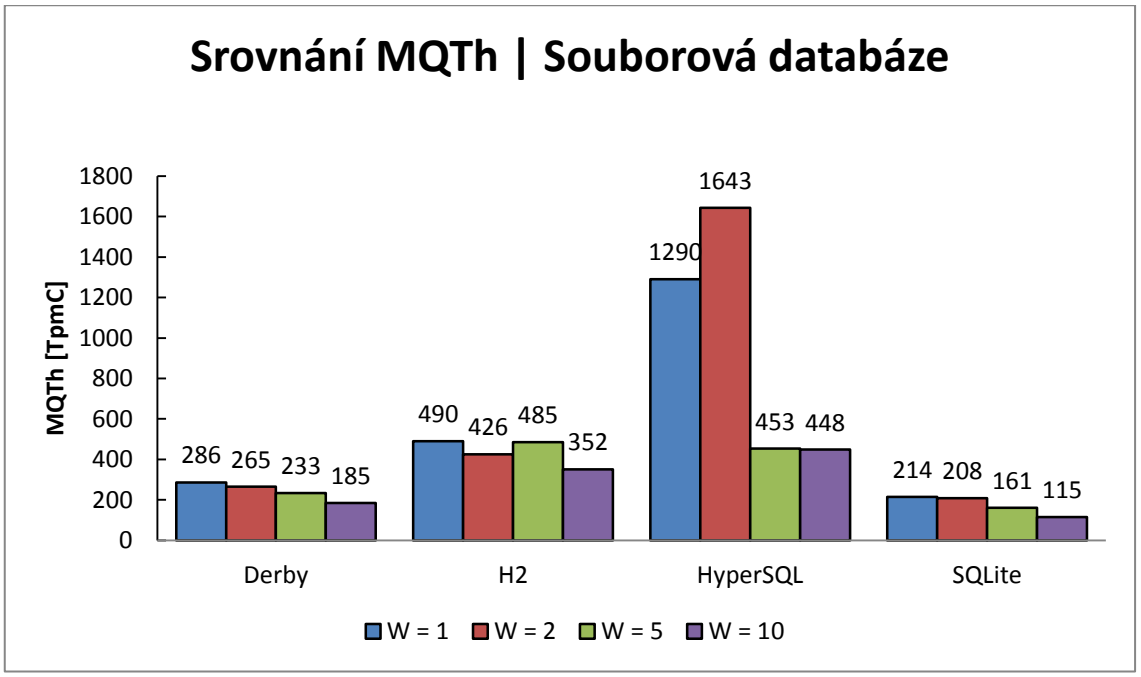
Graf 8.1.1 Ukazuje výsledky TPC-C testu naměřené v souborovém režimu databáze. Nejlepších výsledků dosahuje databázový systém HyperSQL , který zejména v prvních dvou případech měření dosahuje jasně nejvyšší hodnoty MQTh. Zvláštní je dramatický výkonnostní propad mezi výsledky dosaženými ve druhém a třetím měřením databáze HyperSQL. Ten je patrný i z grafu znázorňujícího vývoj propustnosti v čase, který je uveden v příloze E a ukazuje značně nestabilní výsledky naměřených hodnot propustnosti v průběhu celého testu. Naopak jasně nejhorších výsledků dosáhli SŘBD SQLite a Derby. Na rozdíl od prvních dvou systémů však dosahují stabilních výsledků během celé doby testu jak je vidět z příslušných grafů vývoje propustnosti v již zmiňované příloze.




***Graf 9.1.1:*** *Porovnání propustnosti SŘBD Derby, H2, HyperSQL a SQLite v režimu s perzistentním ukládáním dat na disk – výsledky seřazené podle škálovatelného parametru velikosti databáze.*

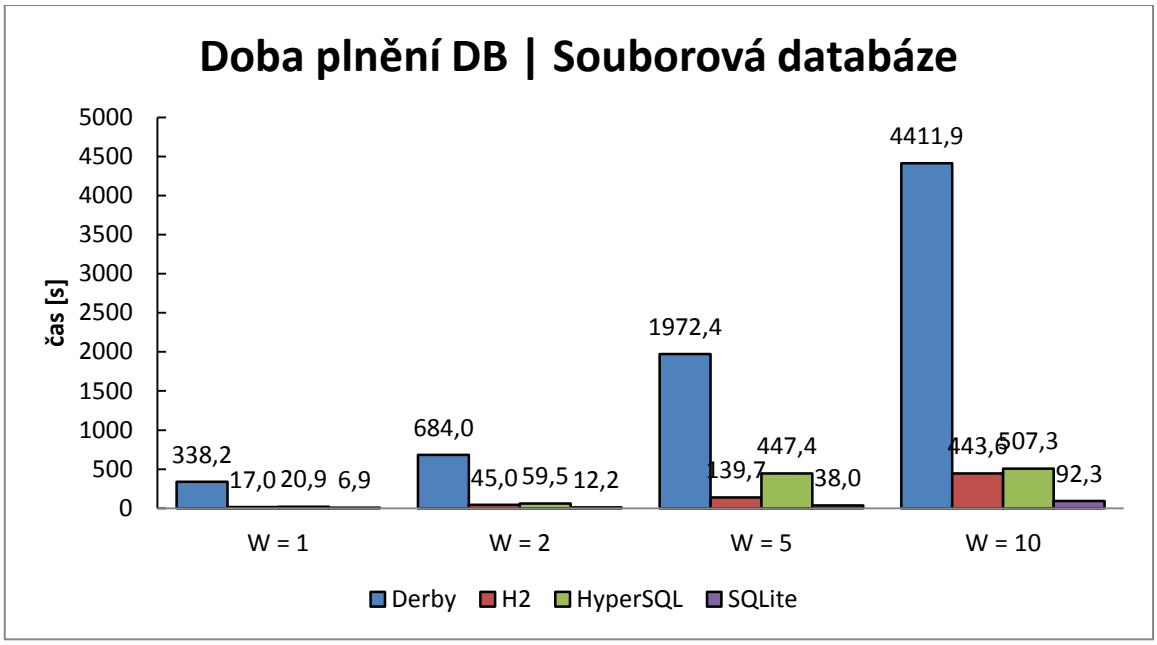
Pokud v grafu 8.1.2 srovnáme stejné výsledky z hlediska výkonnostních rozdílů v závislosti na parametru W, opět nejvíce překvapí již zmiňovaný dramatický propad

výsledků mezi druhým a třetím testem databáze HyperSQL. V případě H2 je vidět výraznější zhoršení výsledků v posledním testu, naopak v prvních třech měřeních dosahovala srovnatelných hodnot. Zajímavé jsou grafy vývoje propustnosti v čase, kde v případě H2 dochází v prvních třech po určité době ke značnému výkonnostnímu propadu až na třetinu původně dosahovaných hodnot. Databázové systémy Derby a SQLite také výkonově zpomalují v závislosti na rostoucím parametru W, ale na rozdíl od HyperSQL a H2 dosahují po celou dobu testu stabilních výsledků, jak ukazují příslušné grafy vývoje propustnosti v čase.



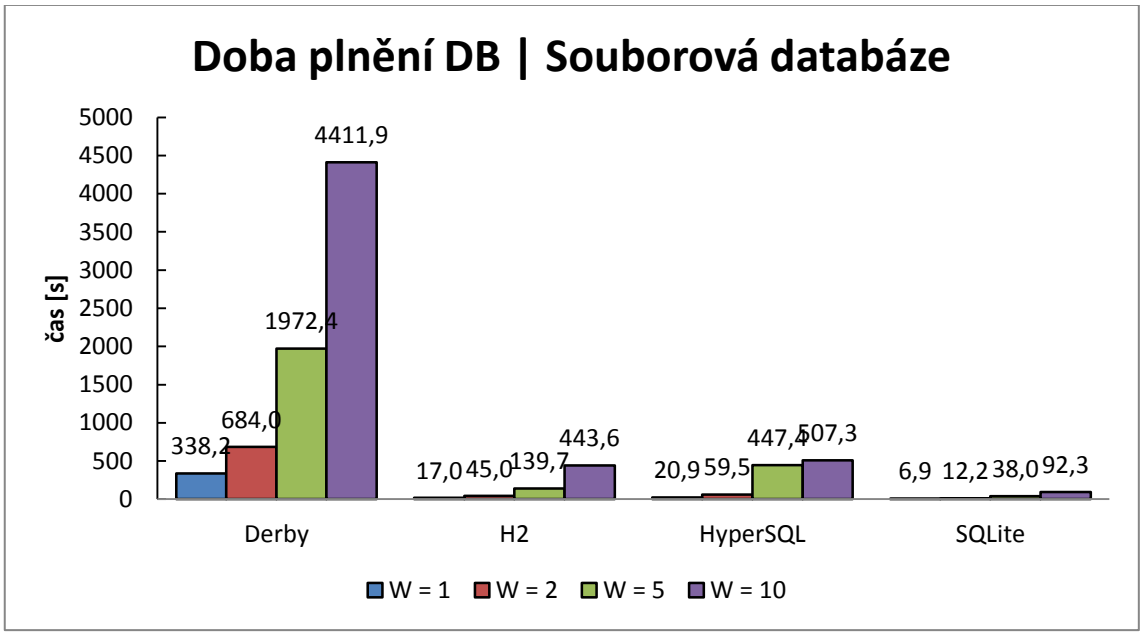

***Graf 9.1.2:*** *Porovnání propustnosti SŘBD Derby, H2, HyperSQL a SQLite v režimu s perzistentním ukládáním dat na disk – výsledky seřazené podle databázových systémů.*

Z porovnání časů potřebných k vytvoření výchozí testovací databáze (tedy v podstatě rychlost SQL operace INSERT) v grafu 8.1.3 vidíme, že jasně nejhorších výsledků bylo dosaženo s databází derby, která potřebovala k vytvoření největší databáze více než hodinu času. Naopak jasně nejlepších výsledků dosahuje SQLite, která je nejrychlejší ve všech čtyřech měřeních.




**Graf 8.1.3:** *Porovnání časů plnění výchozí testovací databáze v SŘBD Derby, H2, HyperSQL a SQLite v souborovém režimu – výsledky seřazené podle škálovatelného parametru velikosti databáze.*

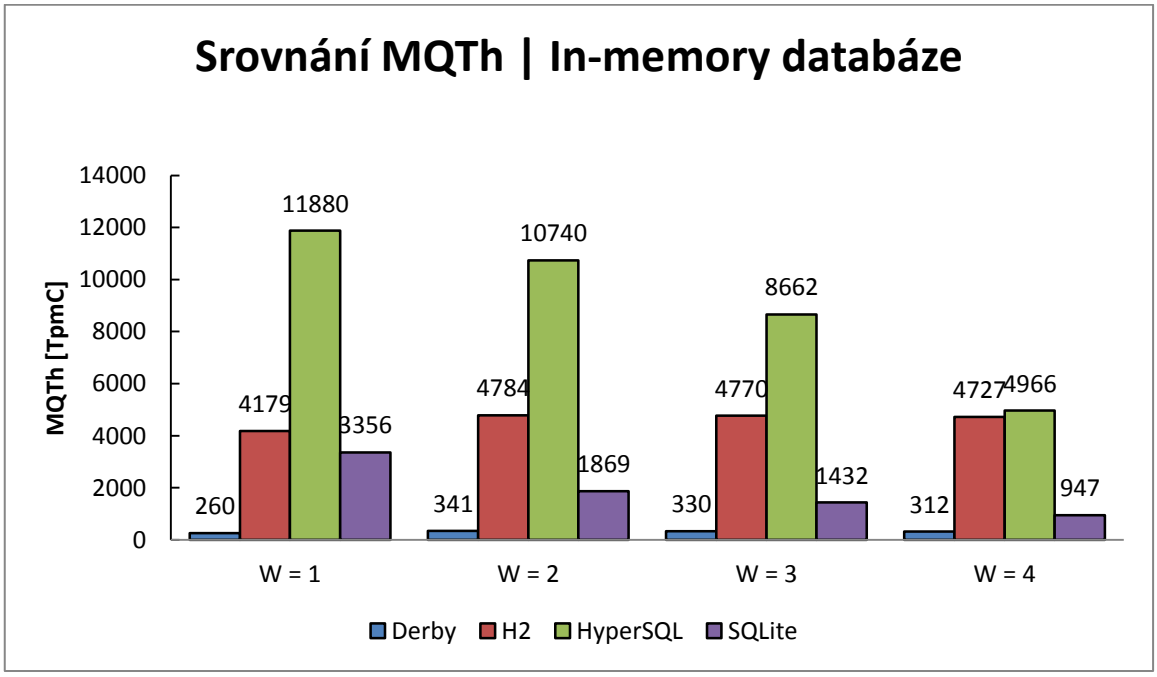
Při srovnání vlivu parametru W na dobu trvání testy je vidět, vysoký nárůst mezi druhým a třetím testem u HyperSQL. Ta má obecně problém při vytváření databáze od velikosti parametru W = 5 při použití CACHED tabulek a dosahuje značně nerovnoměrných výsledků při opakovaném měření za stejných podmínek. Nárůst mezi jednotlivými časy v případě databáze H2 je také značný, ale na rozdíl od HyperSQL byly výsledky podobné i při dalších opakování měření. Naopak SQLite dokázala databázi naplnit velice rychle i při velikosti parametru W = 10.




***Graf 8.1.4:*** *Porovnání časů plnění výchozí testovací databáze v SŘBD Derby, H2, HyperSQL a SQLite v souborovém režimu – výsledky seřazené podle databázových systémů*

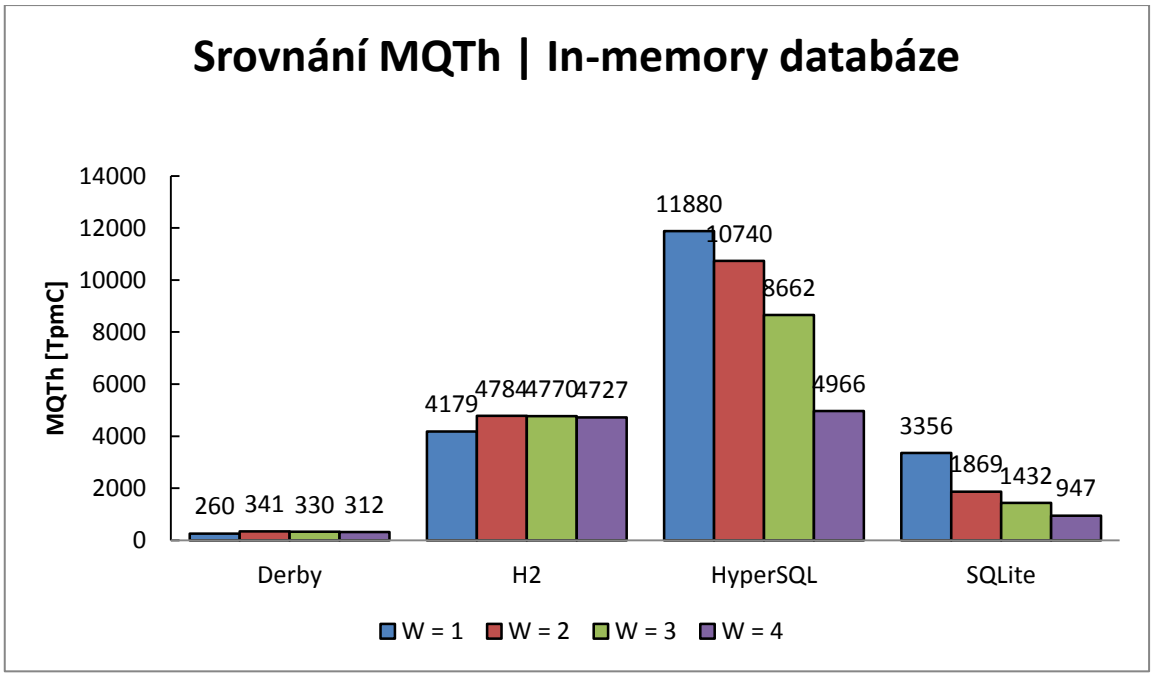
## 9.2 In-memory režim databáze

Pokud databáze zpracovává data uložená pouze v operační paměti je z grafu 8.1.5 patrné, že nejlepších výsledků propustnosti dosahuje jednoznačně opět HyperSQL. Naopak nejhorší je databáze Derby, která se zrychlila pouze minimálně oproti výsledkům dosaženým v souborovém režimu.



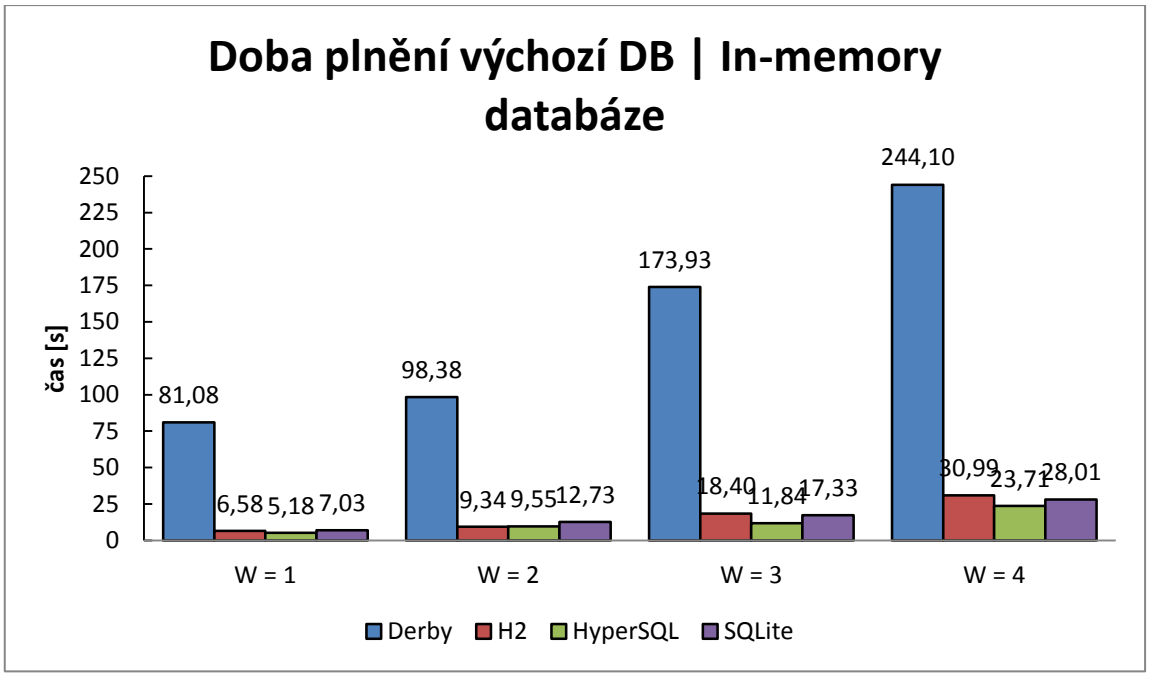

***Graf 8.1.5:*** *Porovnání propustnosti SŘBD Derby, H2, HyperSQL a SQLite v režimu s neperzistentním ukládáním dat – výsledky seřazené podle škálovatelného parametru velikosti databáze.*

Zajímavé jsou výsledky databáze H2 z pohledu srovnání vlivu velikosti databáze v grafu 8.1.6, které jsou velmi podobné ve všech čtyřech testech, zatímco HyperSQL a SQLite zpomalují v závislosti na zvětšujícím se parametru W. Databáze Derby v testu dosahovala stabilních, ale velmi nízkých hodnot propustnosti, jako ostatně ve všech měřeních.



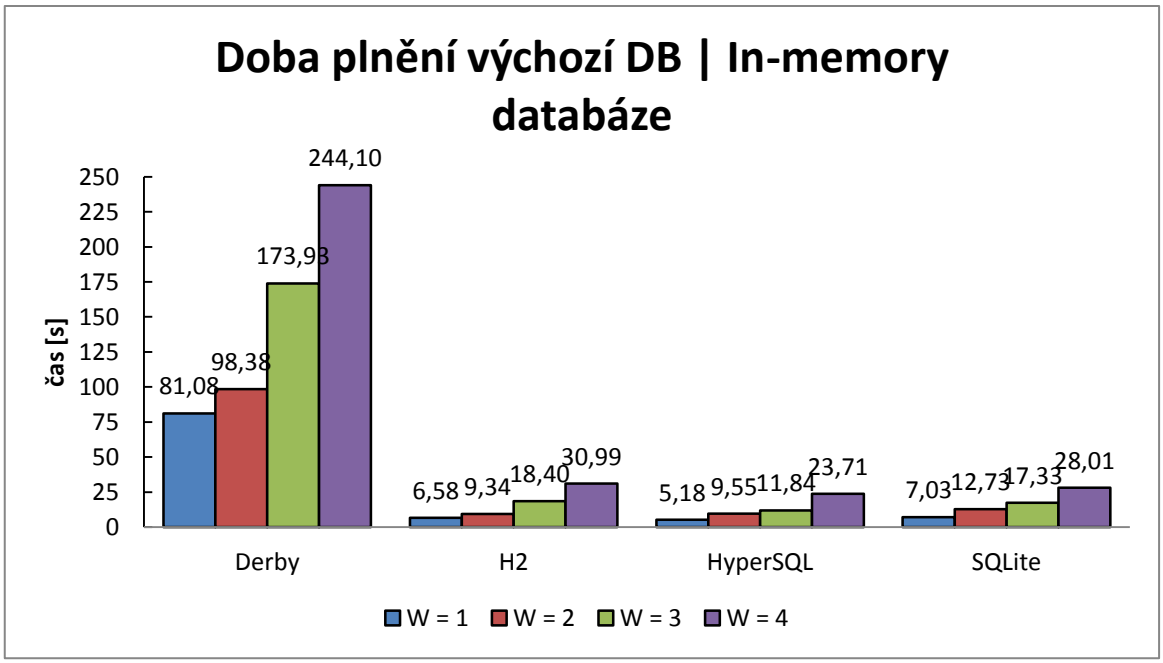

**Graf 8.1.6:** *Porovnání propustnosti SŘBD Derby, H2, HyperSQL a SQLite v režimu s neperzistentním ukládáním dat – výsledky seřazené podle databázových systémů.*

Z hlediska výsledného umístění jednotlivých databází při porovnávání doby potřebné k naplnění testovací databáze uvedené v grafu 8.1.7 se opakují výsledky dosažené jednotlivými SŘBD v souborovém režimu. Znovu se jako výrazně nejpomalejší ukazuje databáze Derby, naopak nejlepších výsledků opět dosahuje SQLite.




**Graf 8.1.7:** *Porovnání časů plnění výchozí testovací databáze v SŘBD Derby, H2, HyperSQL a SQLite v režimu s neperzistentním ukládáním dat – výsledky seřazené podle škálovatelného parametru velikosti databáze*

Při pohledu na výsledky seřazené podle jednotlivých SŘBD v grafu 8.1.8, se v in-memory režimu zjistíme, že se zde neobjevují tak výrazné výkonnostní propady u vytváření větších databází jako tomu bylo v souborovém režimu zejména u HyperSQL, ale také u H2 databáze.




***Graf 8.1.8:*** *Porovnání časů plnění výchozí testovací databáze v SŘBD Derby, H2, HyperSQL a SQLite v režimu s neperzistentním ukládáním dat – výsledky seřazené podle databázových systémů*

# 10. Závěr

Svou práci bych rozdělil na dvě základní části. V první jsem se rozhodl nejprve poskytnout stručný teoretický základ o současných možnostech databázových technologií, zaměřený především na popis používaných datových modelů. Představeny byly relační, objektové, objektově-relační, klíč/hodnota, dokumentové, sloupcové a grafové databáze. U každé kategorie je kladen důraz zejména na důvodu vedoucí k použití příslušného datového modelu, klady a zápory plynoucí z jeho nasazení a konkrétní příklady použití. Následuje kapitola 3, kde jsou stručně představeny vybrané databázové systémy splňující základní požadované kritérium – umožnují embedded připojení s programovacím jazykem Java. Celkem je v práci uvedeno 18 různých SŘBD. Doplňkovým kritériem výběru bylo, aby mezi vybranými SŘBD byly zástupci každého z popsaných datových modelů. Druhá část práce se soustředí na podrobnější analýzu databázových systémů HyperSQL, H2, Apache Derby a SQLite pro které byl připraven základní benchmark ke srovnání výkonových parametrů. Benchmmark byl připraven na základě specifikace transakčního TPC-C testu. Měřeny jsou hodnoty propustnosti testu (MQTh) udávané v počtu realizovaných nových objednávek. Výsledné hodnoty jsou graficky zpracovány a stručně zhodnoceny.

# Seznam zkratek

| | |
|---|---|
| ACID | Atomicity, Consistency, Isolation, Durability |
| ACL | access control list |
| AES | Advanced Encryption Standard |
| API | Aplication Programming Interface |
| AWT | Abstract Window Toolkit |
| BASE | Basically Available, Soft-state, Eventually Consistent |
| BLOB | Binary Large OBject |
| BSD | Berkeley Software Distribution |
| BSON | Binary JSON |
| CAD | Computer Aided Design |
| CLI | Command Line Interface |
| CQL | Cassandra Query Language |
| CRUD | Create, Read, Update, Delete |
| DCL | Data Control Language |
| DDL | Data Definition Language |
| DML | Data Manipulation Language |
| DPL | Direct Persistence Layer |
| DRDA | Distributed Relational Database Architecture |
| GFS | Google File System |
| GIS | Geografický informační systém |

| | |
|---|---|
| GPL | General Public license |
| GUI | Graphical User Interface |
| HDFS | Hadoop Distributed Filesystem |
| HQL | Hibernate Query Language |
| HTTP | Hypertext Transfer Protocol |
| IDPL | Initial Developer's Public License |
| IDE | Integrated development enviroment |
| JCE | Java Cryptography Extension |
| JDBC | Java Database Connectivity |
| JDK | Java Development Kit |
| JDO | Java Data Objects |
| JDOQL | Java Data Object Query Language |
| JPA | Java Persistence Aplication Programming Interface |
| JPQL | Java Persistence Query Language |
| JSON | JavaScript Object Notation |
| JSQL | Java Structured Query Language |
| JVM | Java Virtual Machine |
| kB | Kilobyte |
| MPL | Mozilla Public License |
| MVCC | Multiversion concurrency control |
| MQTh | |
| Nosql | Not Only Structured Query Language |

| | |
|---|---|
| OCC | Optimistic Concurrency Control |
| ODBC | Open Database Connectivity |
| ODBS | Objektové databázové systémy |
| ODMG | Object Data Management Group |
| OID | Object IDentifier |
| ORDBS | objektově-relační databázové systémy |
| ORM | objektově-relační mapování |
| PHP | Hypertext Preprocessor |
| POJO | plain old Java object |
| REST | Representational State Transfer |
| SEQUEL | Structured English Query Language |
| SHA | Secure Hash Algorithm |
| SQL | Structured Query Language |
| SŘBD | Systém řízení báze dat |
| SSL | Secure Sockets Layer |
| TCL | Transaction Control Language |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| TLS | Transport Layer Security |
| TPC | Transaction Processing Council |
| UUID | universally unique identifier |
| XML | Extensible Markup Language |

# Literatura


[Ott12] OTTE, Lukáš. *Databázové systémy*. Vysoké škola Báňská – Technická univerzita Ostrava. [online]. 2012 [cit. 2015-04-25]. Dostupné z: http://homel.vsb.cz/~ott007/Predmety/databaze/Doprovodny_text_DB.pdf

[Sum07] SUMATHI, S. a S. ESAKKIRAJAN. *Fundamentals of Relational Database Management Systems*. Berlín:Springer-Verlag, 2007, 776 s. ISBN 3-540-48397-7.

[Cod70] CODD, E. F. A relational model of data for large shared data banks. *Communication ACM 13*. 1970-06-06, s. 377-387. [cit. 2015-04-25]. DOI=10.1145/362384.362685. Dostupné z: http://doi.acm.org/10.1145/362384.362685

[Wik15] Edgar F. Codd. *Wikipedia* [online]. 15. 3. 2015 [cit. 2015-04-25]. Dostupné z: http://en.wikipedia.org/wiki/Edgar_F._Codd

[Dat04] DATE, C. J. *An introduction to database systems 8th edition*. Pearson Education, Inc. 2004, 983 s. ISBN 0-321-18956-6.

[Šar03] ŠARMANOVÁ, Jana. *Teorie zpracování dat*. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava. 2003.

[Con05] CONNOLLY, Thomas a Carolyn BEGG. *Database Systems: A Practial Approach to Design, Implementation and Management*.foruth edition. Pearson education, 2005, 1374 s. ISBN 0-321-21025-5.

[Ott04] OTTA, Max. Druhy integrity dat. *Západočeská univerzita v Plzni* [online]. 2004 [cit. 2015-04-25]. Dostupné z: http://www-kiv.zcu.cz/~otta/vyuka/db2/cviceni05.html

[Pok97] POKORNÝ, Jaroslav. Základy implementace souborů a databází. 1. vyd. Praha: Karolinum, 1997, 196 s. ISBN 80-7184-472-1.

[Tiw11] TIWARI, Shashank C. Professional nosql. 1. vydání. Indianapolis: Wiley Pubishing, Inc., 2011. ISBN 04-709-4224-X.

[Bat] BATKO, Michal. Relační vs. objektově-relační vs. objektové databáze. *Masarykova Univerzita v Brně* [online]. [cit. 2015-04-25]. Dostupné z: http://www.fi.muni.cz/~xbatko/oracle/compare.html

[Zím14] ZÍMA, Martin. KIV/DB2 - Objektově relační mapování: Hibernate – model. *Západočeská Univerzita v Plzni*. [online]. 14. 3. 2014 [cit. 2015-04-25]. Dostupné z: http://www.kiv.zcu.cz/~zima/vyuka/db2/cviceni-orm-01.html

[Šve03] ŠVEC, Martin. *Objektové databáze*. Vysoké učení technické v Brně. [online]. 1. 7. 2003 [cit. 2015-04-25]. Dostupné z: http://www.fit.vutbr.cz/study/courses/VPD/public/0203VPD-Svec.pdf

[Mer04] MERUNKA, Vojtěch. Objektový přístup v databázové technologii. *Celostátní konference tvorba softwaru 2004*. (26-28.5 2004). VŠB-TU, Ostrava, 152-162. [cit. 2015-04-25] Dostupné z: http://cev.cemotel.cz/programovani_a_tvorba_sw_1975-2004/2004/152.pdf

[Cat00] CATTELL, R. G. The Object Data Standard: ODMG 3.0. 1. vyd. San Francisco: Morgan Kaufmann, 2000, 273 s. ISBN 1558606475.

[Zaq08] ZAQALBEH, Belal a Essam AL DAOUD. The Constraints of Object-Oriented Databases. *International Journal of Open Problems in Computer Science and Mathematics*. [online]. 2008-07, Vol. 1 No., s. 11-17. [cit. 2015-04-25] Dostupné z: http://www.ijopcm.org/files/IJOPCM2.pdf

[Len09] LENC, Ladislav *Dokumentační databáze pro ČTK*.Plzeň, 2009. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

[Hib15] Hibernate ORM documentation. *Hibernate.org* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://hibernate.org/orm/documentation/

[Šim99] ŠIMŮNEK, Milan. SQL kompletní kapesní průvodce.1. vyd. Praha: Grada, 1999, 248 s. ISBN 80-7169-692-7.

[ODBMS05] Overview RDBMS-ORDBMSOODBMS. *ODBMS.org* [online]. 2005 [cit. 2015-04-25]. Dostupné z: http://www.odbms.org/wp-content/uploads/2013/11/013.01-Chountas-RDBMS-versus-ORDBMS-versus-OODBMS-August-2005.pdf

[Neu10] NEUBAUER Peter, Graph Databases, NOSQL and Neo4j. *InfoQ.com* [online]. 12. 3. 2010 [cit. 2015-04-25]. Dostupné z: http://www.infoq.com/articles/graph-nosql-neo4j

[Gan12] GANTZ John a David REINSEL THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadow s, and Biggest Grow th in the Far East. *emc.com* [online] prosinec 2012 [cit. 2015-04-25]. Dostupné z:

http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf

[Bro09] BROWNE Julian. Brewer's CAP Theorem. *emc.com* [online] 11.1 2009 [cit. 2015-04-25]. Dostupné z: http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

[Staf14] STAFFOVÁ, Barbora *Správce databází pro vybraný Nosql databázový systém.* Plzeň, 2009. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

[Pok12] POKORNÝ, Jaroslav. NoSQL databáze: současný stav vývoje. In: Moderní databáze 2012: Zpracování velkých objemů dat a transakcí. Praha: KOMIX s.r.o., 2012, s. 11. ISBN 978-80-905231-0-4. Dostupné z: http://www.plustek.cz/sitecore/content/Komix_cs CZ/Home/Produkty/MD/2012/~/media/Komix/Downloads/Konference _Moderni_databaze/Moderni_databaze_2012/Sbornik_konference_Moderni_databaze_2012.ashx

[Dec07] DECANDIA, Giuseppe, Deniz HASTROUN, Madan JAMPANI, Gunavardhan KAKULAPATI, Avinash LAKSHMAN, Alex PILCHIN, Swaminathan SILVASUBRAMANIAN, Peter VOSSHALL, a Werner VOGELS. 2007. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles* (SOSP '07). ACM, New York, NY, USA, 205-220. DOI=10.1145/1294261.1294281. Dostupné z: http://doi.acm.org/10.1145/1294261.1294281

[Ree13] REEVE, April. Big Data Architectures – NoSQL Use Cases for Key Value Databases. *Emc.com* [online]. 25. 11. 2013 [cit. 2015-04-25]. Dostupné z https://infocus.emc.com/april_reeve/big-data-architectures-nosql-use-cases-for-key-value-databases/

[Chan08] CHANG Fay, Jeffrey DEAN, Sanjay GHEMAWAT, Wilson C. HSIEH, Deborah A. WALLACH, Mike BURROWS, Tushar CHANDRA, Andrew FIKES, a Robert E. GRUBER. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4 (June 2008), 26 pages. DOI=10.1145/1365815.1365816. Dostupné z: http://doi.acm.org/10.1145/1365815.1365816

[Swa13] SWANHART, Justin. Introduction to column stores. *Linkedln corporation* [online]. 04. 2013 [cit. 2015-04-28]. Dostupné z: http://www.slideshare.net/MySQLGeek/intro-to-column-stores?next_slideshow=1

[Bry14] BRYDEN, James. Guide to wide column stores – Nosql explained. *Bryden web solutions* [online]. 2014 [cit. 2015-04-28]. Dostupné z: http://nosqlguide.com/column-store/nosql-databases-explained-wide-column-stores/

[Bryd14] BRYDEN, James. Guide to document databases – Nosql explained. *Bryden web solutions* [online]. 2014 [cit. 2015-04-28]. Dostupné z: http://nosqlguide.com/document-store/nosql-databases-explained-document-databases/

[Woo15] WOODIE, Alex. Graph Databases Everywhere by 2020, Says Neo4j Chief. *Datanami.com* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://www.datanami.com/2015/01/15/graph-databases-everywhere-2020-says-neo4j-chief/

[Hol12] HOLÝ, Josef. Grafové databáze a Neo4j. *Youtube* [online]. 31. 12. 2012 [cit. 2015-04-25]. Dostupné z: https://www.youtube.com/watch?v=lLuqIcYvWUQ

[Bach14] BACHMAN, Michal. Informatický večer FIT - Grafové databáze první oficiální Neo4j Meetup. *Youtube* [online]. 16. 1. 2014 [cit. 2015-04-25]. Dostupné z: https://www.youtube.com/watch?v=lLuqIcYvWUQ

[Rob13] ROBINSON, Ian, Jim WEBBER a Emil EIFREM. Graph databases. 1. vyd. O'Reilly Media, Inc., 2013, 208 s. ISBN 978-1-449-35626-2.

[Oracle15] Getting Started with Berkeley DB Java Edition. *Oracle* [online]. 5. 9. 2015 [cit. 2015-04-25]. Dostupné z: https://docs.oracle.com/cd/E17277_02/html/GettingStartedGuide/BerkeleyDB-JE-GSG.pdf

[Iqlect14] BangDB Manual – version 1.5. *Iqlect* [online]. 2014 [cit. 2015-04-25]. Dostupné z: http://www.iqlect.com/developer.php

[Sin12] SINHA, Sachin. Performance Data For LevelDB, Berkley DB And BangDB For Random Operations. *High scalability* [online]. 29. 11. 2012 [cit. 2015-04-25]. Dostupné z: http://highscalability.com/blog/2012/11/29/performance-data-for-leveldb-berkley-db-and-bangdb-for-rando.html

[Dea11] DEAN, Jeff a Sanjay GHEMAWAT. LevelDB: A fast persistent key-value store. *Google Open-source Blogspot* [online]. 27. 7. 2011 [cit. 2015-04-25]. Dostupné z: http://google-opensource.blogspot.cz/2011/07/leveldb-fast-persistent-key-value-store.html

[LevelDB] LevelDB Features. *Leveldb.org* [online]. [cit. 2015-04-25]. Dostupné z: http://leveldb.org/

[Dea] DEAN Jeff, Sanjay GHEMAWAT. LevelDB docs. *Googlecode.com* [online]. [cit. 2015-04-25]. Dostupné z: http://leveldb.googlecode.com/git-history/1.17/doc/index.html?r=1.17

[Apa15] Apache HBase reference guide. *The Apache Software Foundation* [online]. 11. 4. 2015 [cit. 2015-04-25]. Dostupné z: http://hbase.apache.org/book.html

[CodeF14] Database sharding. *CodeFutures* [online]. 2014 [cit. 2015-04-25]. Dostupné z: http://codefutures.com/database-sharding/

[ApaW15] Hbase powered by. *Wiki.Apache.org* [online]. 6. 3. 2015 [cit. 2015-04-25]. Dostupné z: http://wiki.apache.org/hadoop/Hbase/PoweredBy

[Str11] STRAUCH, Christof. *NoSQL Databases*. Stuttgart, 2011. Dostupné z: http://www.christof-strauch.de/nosqldbs.pdf

[Dat15] Cassandra: Getting Started Documentation. *DataStax* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://docs.datastax.com/en/cassandra/2.1/cassandra/gettingStartedCassandraIntro.html

[Sqlite15] SQLite documentation. *SQLite.org* [online]. 2015 [cit. 2015-04-25]. Dostupné z: https://www.sqlite.org/docs.html

[Sim14] SIMPSON, Blaine a Fred TOUSSI. HyperSQL User Guide: HyperSQL Database Engine (HSQLDB) 2.3. *hsqldb.org* [online]. 13. 2. 2014 [cit. 2015-04-25]. Dostupné z: http://hsqldb.org/doc/2.0/guide/guide.pdf

[Simp15] SIMPSON, Blaine a Fred TOUSSI. HyperSQL Utilities Guide. *hsqldb.org* [online]. 13. 2. 2014 [cit. 2015-04-25]. Dostupné z: http://hsqldb.org/doc/2.0/util-guide/util-guide.pdf

[H2d15] Reference: H2 Database Engine. *h2database* [online]. 10. 4. 2015 [cit. 2015-04-25]. Dostupné z: http://www.h2database.com/h2.pdf

[Apa14] Apache Derby: Documentation. *Apache Software Foundation* [online]. 27. 8. 2014 [cit. 2015-04-25]. Dostupné z: http://db.apache.org/derby/manuals/index.html#docs_10.11

[Can10] CANTU, Carlos. Poznejte Firebird za 2 minuty. *Firebirdnews* [online]. 2010 [cit. 2015-04-25]. Dostupné z: http://www.firebirdnews.org/docs/fb2min_cz.html

[Fir14] Historical reference. *Firebirdsql* [online]. 2014 [cit. 2015-04-25]. Dostupné z: http://www.firebirdsql.org/en/historical-reference/

[Fir11] Firebird 2.5 Quick Start Guide. *IBPhoenix, Firebird Project* [online]. 26. 9. 2011 [cit. 2015-04-25]. Dostupné z: http://www.firebirdsql.org/file/documentation/reference_manuals/user_manuals/Firebird-2.5-QuickStart.pdf

[Neo4j15] The Neo4j Manual v2.2.2. *Neo4j* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://neo4j.com/docs/2.2.2/

[Ori15] OrientDB Manual – version 2.0. *orientdb* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://orientdb.com/docs/2.0/index.html

[Mar12] MARTÍNEZ-BAZAN, M. Norbert, Ángel ÁGUILA-LORENTE, Victor MUNTÉS-MULERO, David DOMINGUEZ-SAL, Sergio GÓMEZ-VILLAMOR, a Josep-L. LARRIBA-PEY. 2012. Efficient graph management based on bitmap indices. *Proceedings of the 16th International Database Engineering & Applications Sysmposium* (IDEAS '12). ACM, New York, NY, USA, 110-119. [cit. 2015-04-25] DOI=10.1145/2351476.2351489 Dostupné z: http://doi.acm.org/10.1145/2351476.2351489

[Spark14] Sparksee user manual. *Sparksity Technologies* [online]. 2014 [cit. 2015-04-25]. Dostupné z: http://www.sparsity-technologies.com/UserManual/Index.html

[iBo15] Fast transactional tablestyle document NoSQL application database. *iBoxDB* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://www.iboxdb.com/

[uni13] iBoxDB -Lightweight Embedded Database. *unity3d.com* [online]. 2013 [cit.2015-04-25]. Dostupné z: http://www.iboxdb.com/

[Soft14] EJDB Documentation. *Softmotions Ltd.* [online]. 2014 [cit. 2015-04-25]. Dostupné z: http://ejdb.org/index.html#

[McO14] Perst – Datasheet. *McObject LLC* [online]. 2014 [cit. 2015-04-25]. Dostupné z: http://www.mcobject.com/index.cfm?fuseaction=download&pageid=459&sectionid=140

[Mco12] Perst Introduction and Tutorial: Java and Java ME (Perst Lite) edition. *McObject LLC* [online]. 10. 10. 2012 [cit. 2015-04-25]. Dostupné z: http://www.mcobject.com/index.cfm?fuseaction=download&pageid=458&sectionid=139

[Obj15] ObjectDB Object Database Features. *ObjectDB Software* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://www.objectdb.com/object/db/database/features

[Obje15] ObjectDB 2.5 Developer's Guide. *ObjectDB Software* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://www.objectdb.com/object/db/database/overview

[Objectivity15] Documentation – Objectivity/DB 11.0. *Objectivity Inc.* [online]. 2015 [cit. 2015-04-25]. Dostupné z: http://support.objectivity.com/docs/objectivity/11_0_0

[Man05] Teorie databází: transakce. *Manuály.net* [online]. 22. 12. 2005 [cit. 2015-06-10]. Dostupné z: http://www.manualy.net/article.php?articleID=8

[TPC 15] About the TPC. Transaction Processing Performance Council. 2015 [cit. 2015-06-24]. Dostupné z: http://www.tpc.org/information/about/abouttpc.asp

[Krá15] KRÁL, Jakub. Porovnání open source databázových systémů s využitím TPC-C testu: *Bakalářská práce*. Praha : Vysoka škola ekonomická v Praze, Fakulta informatiky a statistiky, 2013. 97 l., 25 l. příl. Vedoucí bakalářské práce práce Dušan Chlapek.

[TPC10] TPC Benchmark C: Standard specification. *Transaction Processing Council* [online]. 02 2010 [cit. 2015-06-25]. Dostupné z: http://www.tpc.org/information/current_specifications.asp

[SAN07] SANDSTA, Olav. Configuring Apache Derby for performance and durability. *Sun Microsystems* [online]. 2007 [cit. 2015-06-25]. Dostupné z: http://db.apache.org/derby/binaries/DerbyPerfDurability-2006.pdf

# A. Srovnání parametrů vybraných databázových systémů

| Název | Vývojářská společnost | Jazyk implementace | První vydání | Poslední stabilní verze | Open-source licence | Komerční licence | Trial verze |
|---|---|---|---|---|---|---|---|
| SQLite | Hipp, Wyrick & Company, Inc. | Ansi C | 2005 | 3.8.7 (2014) | Public domain | Ne | - |
| HSQL | hsql development group | Java | 2001 | 2.3.2 (2014) | BSD | Ne | - |
| Derby | Apache foundation, Oracle | Java | 1997 | 10.10.1.2 (2014),(JDK 1.8) | Apache license 2.0 | Ne | - |
| H2 | Thomas Mueller | Java | 2005 | 1.3.176 (2014) | MPL 2.0 | Ne | - |
| Firebird | Firebird foundation Incorporated | C, C++ | 2000 | 2.5.3 (2014) | IDPL 1.0 | Ne | - |

| Název | Možnosti provozu | Přístupová rozhraní | Podporované jazyky | Nástroje | Použití v projektech | Dokumentace | Komerční podpora |
|---|---|---|---|---|---|---|---|
| SQLite | Embedded<br>In-memory | JDBC (neoficiální)<br>ODBC (neoficiální) | Java, C, C++, C#,<br>Perl, PHP, Python… | SQL CLI | Mozilla Firefox,<br>Dropbox, Adobe | Online, často neaktuální<br>Podrobnější v knize | Ne |
| HSQL | Embedded<br>Klient-server<br>In-memory | JDBC<br>ODBC | Java,<br>další via ODBC | SqlTool (SQL CLI)<br>Database Manager (GUI) | Open office 3.0,<br>Onepoint,<br>Mathematica | Online, podrobná dokumentace | Ano (3rd party) |
| Derby | Embedded<br>In-memory<br>Klient-server | JDBC<br>ODBC (neoficiální) | Java | Ij (SQL CLI) | ? | Online, podrobná dokumentace od Apache i Oracle | Oracle (v rámci JDK) |
| H2 | Embedded<br>Klient-server<br>Mixed (Embedded + server)<br>In-memory | JDBC<br>ODBC<br>ADO .NET (neoficiální) | Java<br>C#<br>další via ODBC | H2 Console | ? | Online, podrobná dokumentace | Ano (3rd party) |
| Firebird | Embedded server<br>Klient-server<br>In-memory | JDBC (Jaybird), ODBC,<br>C/C++ API, ADO .NET | C,C++,Java,Ruby,PHP,<br>Python… | Isql (SQL CLI)<br>Gsec (CLI administrace zabezpečení db) | Steamsoft,<br>U.S. Navy | Online, stručná, neaktuální<br>Podrobná dokumentace v knize | Ano (3rd party) |

| Název | Transakce | SQL standard | Souběžný přístup (v embedded provozu) | Řízení souběhu | Možnosti zabezpečení | Některé další funkce |
|---|---|---|---|---|---|---|
| SQLite | ACID | SQL-92 (pouze částečně) | Více vláken<br>Více procesů | Zamykání DB pro<br>Zápis (N čtenářů 1 písař) | Žádné<br>Nelze ani určovat role a práva uživatelů | Triggery,<br>Dynamické datové typy |
| HSQL | ACID | 92, 1999, 2003, 2008, 2011 | Více vláken<br>jeden proces | M čtenářů N písařů<br>Zámky (tabulky)<br>MVCC (zámky pro řádky)<br>MVLOCKS | ACL<br>Šifrování Klient-server komunikace (SSL)<br>šifrování db (pomocí JCE) | Triggery |
| Derby | ACID | SQL-92 (téměř vše)<br>Částečně: SQL1999, 2003, 2008 | Více vláken<br>Jeden proces | Zamykání řádků,<br>Zamykání tabulek | Role a práva pro jedn. uživatele<br>Šifrování db (JCE)<br>Šifrování Klient-server komunikace (SSL,TSL) | Triggery<br>Master-slave replikace |
| H2 | Atomické<br>Konzistentní | SQL-92,1999,2003<br>Režimy kompatibility s vybranými SŘBD | Více vláken<br>Jeden proces | Zámky<br>MVCC | Role a práva pro jedn. uživatele<br>Šifrování db (AES 128)<br>Šifrování hesla uživatelů (SHA-256)<br>Šifrování Klient-server komunikace (TLS) | Fulltextové vyhledávání (nativně nebo Lucene)<br>Triggery |
| Firebird | ACID | SQL-92, SQL-99 | Více vláken (každé vlastní připojení)<br>Jeden proces | MVCC<br>Zámky | V embedded verzi žádné | Triggery |

**Tabulka A.1**: Porovnání relačních databázových systémů SQLite, HyperSQL, Apache Derby a H2. [Can10] [Fir14] [Fir11] [Apa14] [H2d15] [Sim14] [Sqlite15]

| Název | ObjectDB | Perst | Objectivity DB |
|---|---|---|---|
| Vývojářská společnost | ObjectDB software Ltd | McObject LLC | Objectivity, Inc. |
| Jazyk implementace | Java | Java | ? |
| První vydání | ? | ? | 1990 |
| Poslední stabilní verze | 2.5.6 (17. 6. 2014) | 4.36 (25. 8. 2014) | 11.2. (5. 9. 2013) |
| Open-source licence | Pouze komerční, zdarma pro studijní a vědecké účely | GNU GPL 3.0 | Ne |
| Komerční licence | £1,800 pro celou firmu £300 za jeden server | Ano, individuální podmínky | Ano, individuální podmínky |
| Trial verze | Libovolné využití, omezená velikostí db | - | 60 denní |
| Možnosti provozu | Embedded Klient-server In-memory | Embedded In-memory | Embedded server In-memory Distribuovaná db |
| Přístupová rozhraní | JDO JPA | Java API, .NET API | ODBC |
| Podporované jazyky | JVM jazyky | Java, C# | Java, C,C++,C# |
| Další nástroje | Replikace, clustering, zálohování, obnova db, správa serveru | | Zálohování, obnova, správa db a serverů, clustering, paralelní zpracování dotazů |
| Použití v projektech | ? | Vhodné pro OS Android | Lockheed Martin, Siemens, Ericsson |
| Dokumentace | Pouze stručný popis systému s jednoduchými příklady | Online, stručný popis + možnosti použití | Online, podrobná včetně příkladů |
| Komerční podpora | V rámci komerční licence | V rámci komerční licence | V rámci komerční licence |
| Transakce | ACID | ACID | ACID |
| Souběžný přístup | Více vláken Více procesů | Více procesů Více vláken | Více vláken Více procesů |
| Řízení souběhu | Zamykání objektů (čtení/zápis, optimistické / pesimistické) | Optimistické zamykání pesimistické zamykání (databáze, souboru, objektů) | Exkluzivní zamykání objektu pro čtení i zápis |
| Možnosti zabezpečení | Šifrování Klient-server komunikace (SSL) | Šifrování db | Šifrování objektů |
| Dotazovací jazyk | JDOQL, JPQL | JSQL, vlastní API | SQL++, vlastní API |
| Možnosti indexování | Jednoduché indexy i Složené indexy (B-tree) | Jednoduché i složené B-tree, KD-tree, R-tree, T-tree | Jednoduché indexy Vícerozměrné indexy |
| Některé další funkce | Master-slave replikace, triggery (lifecycle events) | XML import/export Fulltextové vyhledávání (Lucene) | Master-slave replikace |

**Tabulka A.2**: Porovnání objektových databázových systémů Perst, ObjectDB a Objectivity/DB [McO14] [Mco12] [Objectivity15] [Obj15] [Obje15]

| Název | iBoxdb | EJDB |
|---|---|---|
| Vývojářská společnost | ? | Softmotions |
| Jazyk implementace | Java<br>C# (konverze Java verze) | C |
| První vydání | ? | 2012 |
| Poslední stabilní verze | iBoxDB Java v1.8.1 (2014) | 1.2.6 (15.4.2015) |
| Open-source licence | Free Software Redistributable | GNU GPL v2.1 |
| Komerční licence | Ne | Ne |
| Trial verze | - | - |
| Možnosti provozu | Embedded<br>In-process<br>Embedded server | Embedded<br>In-process |
| Přístupová rozhraní | Vlastní Java API, nebo vlastní .NET API | Java API |
| Podporované jazyky | Java, C# | Java, C#, Ruby, Nodejs, Python<br>Pro další jsou neoficiální API |
| Další nástroje | Ne | CLI API (Nodejs) |
| Použití v projektech | Používána s OS Android<br>Webové aplikace (unity plugin) | Vhodné pro menší projekty, které nepotřebují síťový přístup k db například PC hry |
| Dokumentace | Není (pouze stručný popis) | Není (pouze stručný popis) |
| Komerční podpora | Není nabízena | Není nabízena |
| Transakce | ACID | Atomické<br>Trvalé |
| Souběžný přístup | Více vláken<br>Jeden proces | Pouze jeden proces v db<br>Více vláken |
| Řízení souběhu | Zamykání záznamu pro zápis | Zamykání kolekce dokumentů pro čtení i zápis.<br>Nad kolekcí může probíhat pouze jedna transakce |
| Zabezpečení | | Připravuje se podpora pro určování rolí a práv uživatelů na úrovni dokumentů |
| Dotazovací jazyk | Vlastní API (SQL-like jazyk) | Vlastní API (obdobné dotazování jako v MongoDB) |
| Možnosti indexování | Jednoduché i složené indexy<br>Primární klíč | Index pro každý dokument (Rid),<br>Vlastní indexy (hash mapy, B+tree) |
| Další funkce | Master-slave, Master-master replikace | Dotazy kompatibilní s MongoDB<br>(možnost přenosu aplikace) |

**Tabulka A.3**: Porovnání dokumentových databázových systémů iBoxDB a EJDB. [iBo15] [uni13] [Soft14]

| Název | Cassandra | Apache HBase |
|---|---|---|
| Vývojářská společnost | Apache Software Foundation, DataStax | Apache Software Foundation |
| Jazyk implementace | Java | Java |
| První vydání | 2008 | 2008 |
| Poslední stabilní verze | 2.1.0 (24. 10. 2014) | 0.98.7 (09.10.2014) |
| Open-source licence | Apache license 2.0 | Apache license 2.0 |
| Komerční licence | DataStax Enterprise Edition | Ne |
| Trial verze | Ne | - |
| Možnosti provozu | Embedded (server) distribuovaná db | Stand-alone (embedded server)<br>Pseudo-distributed<br>Fully-distributed |
| Přístupová rozhraní | CQL API (Java), Velké množství dalších Java API (Thrift, ODBC, JDBC, a další od třetích stran) Rest API, PHP…. | Java API<br>REST API<br>Thrift |
| Podporované jazyky | JVM jazyky, DataStax nabízí ovladače pro C,C++, C#, Ruby, Nodejs | JVM jazyky, další prostřednictvím Thrift API, případně Rest API |
| Další nástroje | Velké množství nástrojů pro správu db je součástí distribuce od společnosti DataStax | Velké množství neoficiálních nástrojů pro správu HBase |
| Použití v projektech | Ebay, Instagram, Netflix, Spotify, více jak 1500 společností | Facebook, Yahoo!,<br>Adobe a mnoho dalších |
| Dokumentace | Online, podrobná od společnosti DataStax | Obsáhlá dokumentace online |
| Komerční podpora | Ano dostupná pro DataStax Cassandra Enterprise Edition<br>Nabídka výukových kurzů | Poskytována společností Cloudera |
| Transakce | BASE | ACID (pouze pro řádky) |
| Souběžný přístup | Více vláken i procesů | Více procesů i vláken |
| Řízení souběhu | možnosti výběrů z různých úrovní konzistence dat pří čtení a zápisu | MVCC, zamykání řádků |
| Možnosti zabezpečení | Ověřování uživatelů<br>Přístupová práva uživatelům na úrovni objektů.<br>Šifrování databáze (uzlu clusteru) | Šifrování síťové komunikace<br>Ověřování uživatelů (ACL) |
| Dotazovací jazyk | CQL (Cassandra Query Language) | Vlastní API operace get, put, scan, delete |
| Možnosti indexování | Primární indexy (row key)<br>Uživatelsky definované indexy<br>Indexace podle názvů sloupců a další možnosti | Primární indexy (row key)<br>Jiné indexy pouze formou speciální tabulky nebo externích nástrojů například Apache Lucene |
| Další funkce | Sharding databáze, triggery, různé modely replikování db, | Sharding databáze, triggery, Master-slave replikace db |

**Tabulka A.4**: Porovnání sloupcových databázových systémů Cassandra a Apache HBase. [iBo15] [uni13] [Soft14]

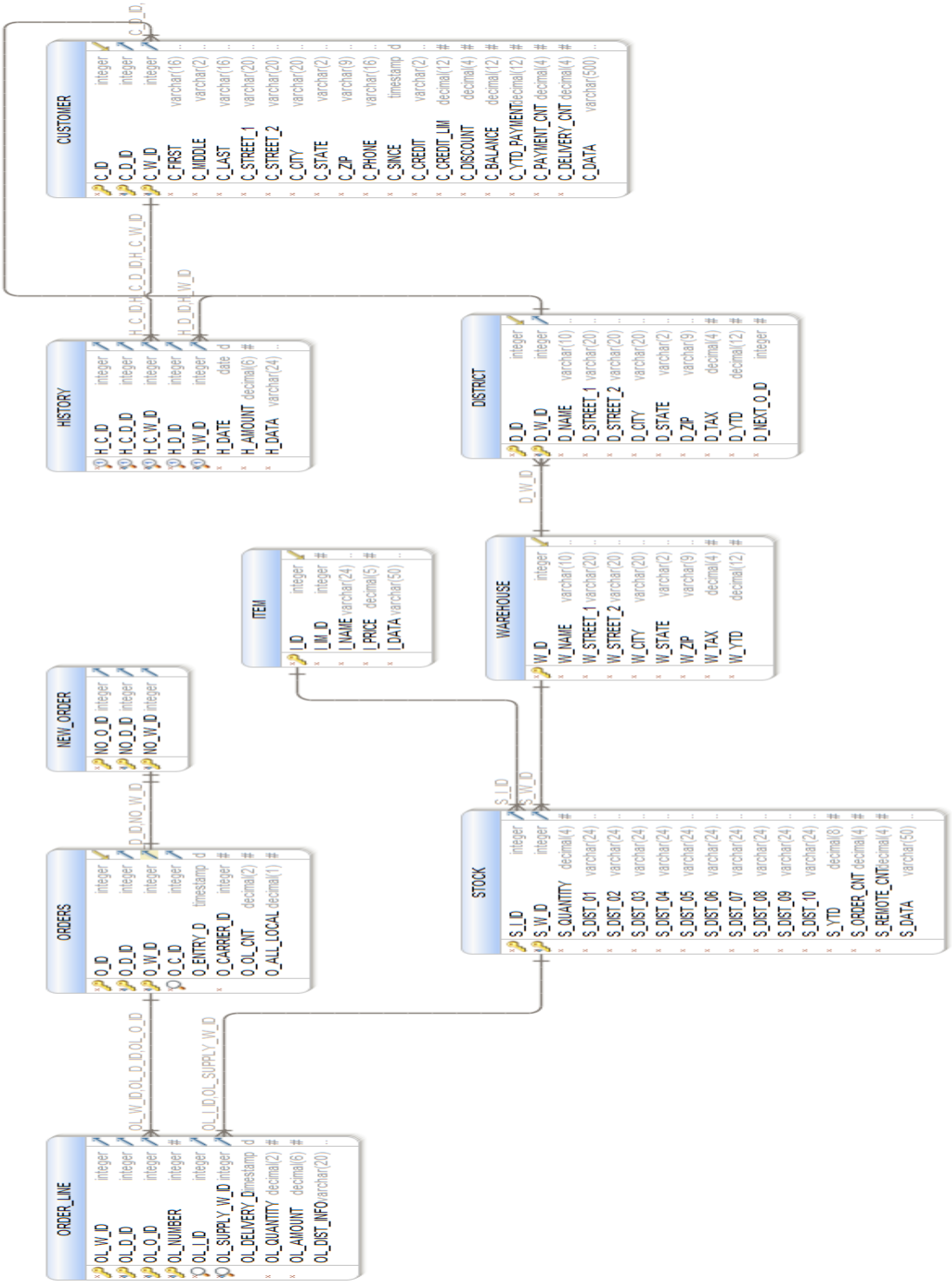
| Název | Oracle Berkeley DB Java edition | BangDB | LevelDB |
|---|---|---|---|
| Vývojářská společnost | Oracle | Iqlect | Google |
| Jazyk implementace | Java | C/C++ | C++ |
| První vydání | 1994 | ? | 2011 |
| Poslední stabilní verze | 6.2.7 (2014) | 1.5 (2014) | 1.8 (2013) |
| Open-source licence | Ano (Oracle) | BSD | BSD |
| Komerční licence | Ano | Připravují se | Ne |
| Trial verze | Ne | - | - |
| Možnosti provozu | Embedded<br>Client-server<br>Distribuovaná db | Embedded<br>In-memory<br>Klient-server<br>Distribuovaná db | Embedded |
| Přístupová rozhraní | Java API (Direct persistence Layer, Javacollection API, key/value API) | Vlastní API | C++, Nodejs<br>C# (neoficiální)<br>Java (neoficiální) |
| Podporované jazyky | Java | Klient pro C++, Java, C# | C++, Java, C# |
| Další nástroje | Správa db z příkazové řádky | ? | Neoficiální CLI a GUI pro přístup k db |
| Použití v projektech | ? | ? | Google Chrome, Riak, Bitcoin (Node.js) |
| Dokumentace | Oficiální dokumentace zdarma online | Online, pouze základní popis funkcí a příklady | Není, pouze stručný popis jednotlivých rozhraní |
| Komerční podpora | V rámci komerčních licencí | Ano | Ne |
| Transakce | ACID | ACID | ACID |
| Souběžný přístup | Více vláken (čtení, zápis)<br>Více procesů (čtení)<br>Jeden proces (zápis) | Více vláken i procesů (i v embedded verzi) | Pouze jeden proces<br>Více vláken |
| Řízení souběhu | Zamykání záznamů | Optimistický model zámků, zamykání pro zápis | Zamykání db (zápis i čtení) |
| Replikace DB | Master-slave | Master-slave | Ne |
| Možnosti zabezpečení | Šifrování db zřejmě nelze<br>Rozlišování uživatelů také ne | V embedded edici není | Není součástí LevelDB knihovny |
| Dotazovací jazyk | Vlastní API (SQL-like) | Vlastní API (CRUD, Částečná podpora SQL) | Vlastní API |
| Možnosti indexování | Primární index<br>Uživatelsky def. Indexy | Primární indexy i uživatelské indexy (B-tree, hash) | Indexy nejsou podporovány<br>Data jsou automaticky seřazena podle klíče |
| Další funkce | | Součástí API jsou nástroje pro analýzu dat<br>Fulltext search | Komprese dat |

**Tabulka A.5**: Porovnání key/value databázových systémů Oracle Berkeley DB JE, BangDB a LevelDB. [Oracle15] [Iqlect14] [LevelDB] [Sin12] [Dea]

| Název | Neo4j | Sparksee | OrientDB |
|---|---|---|---|
| Vývojářská společnost | Neo technology Inc | Sparksity Technologies | Orient Technologies LTD. |
| Jazyk implementace | Java, Scala | C++ | Java |
| První vydání | 2007 | 2008 | 2010 |
| Poslední stabilní verze | 2.1.5 (30. 9. 2014) | 5.1 (2014) | 1.7.9(22. 9. 2014) |
| Open-source licence | GNU GPL 3.0 | Pouze komerční, zdarma pro vědecké a studijní účely. | Community edition Apache 2.0 |
| Komerční licence | Ano (Neo4j Enterprise) | Ano | Ano (OrientDB Enterprise) |
| Trial verze | 30 denní u Enterprise edice | Omezení velikosti DB | |
| Možnosti provozu | Embedded<br>Klient-server | Embedded | Embedded server<br>Klient-server<br>In-memory |
| Přístupová rozhraní | Java API, JDBC,<br>Rest API | Nativní API pro C#, Java, C++, Objective-C a Python | Java API, Rest API, JDBC |
| Podporované jazyky | Java, .Net, Clojure, Go, Groovy, JavaScript, Perl, PHP, Python, Ruby, Scala | C#, Java, C++, Objective-C a Python | Java, .Net, Clojure, Go, Groovy, JavaScript, Perl, PHP, Python, Ruby, Scala |
| Další nástroje | Nástroje pro správu Neo4j serveru (GUI i CLI), import dat, podpora nástrojů pro práci s grafy z frameworku Tinkerpop | Možnost použití nástrojů pro práci s grafy z frameworku Tinkerpop | podpora nástrojů pro práci s grafy z frameworku Tinkerpop, webové rozhraní pro přístup k serveru, Console tool |
| Použití v projektech | Ebay, Cisco, Walmart | ? | Cisco, Lufthansa<br>Warner music group |
| Dokumentace | Podrobná dokumentace online | Online, není příliš podrobná | Podrobná dokumentace online |
| Komerční podpora | U Enterprise edice, placené kurzy | Ano | U Enterprise edice, placené kurzy |
| Transakce | ACID | ACID | ACID |
| Souběžný přístup | Jeden proces (embedded)<br>Více vláken | Pouze jeden proces<br>Více vláken | Jeden proces (embedded)<br>Více vláken |
| Řízení souběhu | Zamykání uzlů/hran pro zápis | Zamykání celé db při zápisu | MVCC |
| Replikace DB | Master-slave | Ne | Multi-master |
| Možnosti zabezpečení | Přístup k serveru via HTTPS, podpora JVM standardů pro šifrování, ACL na úrovni částí grafu. | Vytváření uživatelů s různými právy není podporováno | Šifrovaný přístup k serveru (SSL),<br>Nastavení rolí uživatelů<br>Nastavení zabezpečení na úrovni záznamů |
| Dotazovací jazyk | Cypher, Gremlin (via blueprints API) | Vlastní API, Gremlin (via blueprints API) | SQL rozšířený o práci s grafy, Gremlin (via blueprints API) |
| Možnosti indexování | Nativní podpora indexace vrcholů hran i atributů,<br>Indexace pomocí Lucene | Indexace atributů i hran | SB-tree, hash, indexace pomocí Lucene |
| Model grafu | Property graph | Property graph | Property graph |
| Grafové algoritmy | Hledání nejkratší cesty (Dijkstra,A*), všechny cesty | Prohledávání grafu do hloubky a do šířky, Dijkstra | Dijkstra, hledání nejkratších cest |
| Další funkce | Triggery, sharding db (cache) | | Triggery |

**Tabulka A.5** Porovnání grafových databázových systémů Neo4j, OrientDB a Sparksee [Neo4j15] [Ori15] [Mar12] [Spark14]

# B. ERA diagram testovací databáze




Generated using DbSchema

# C. Připojení k testovaným SŘBD

Zde je uveden stručný návod k použití testovaných SŘBD v jazyce Java. Zaměřuje se především na to jak vytvořit spojení embedded připojení s příslušnou databázovou knihovnou. Některá další specifika (například vybrané konfigurační parametry nebo popis dostupných nástrojů pro správu databáze) jsou zmíněny v kapitolách věnovaných analýze konkrétního systému.

Ve druhé části je podrobněji popsán postup připojení k databázovému systému HyperSQL, spolu s několika dalšími příklady. Protože všechny čtyři zkoumané databázové systémy mohou komunikovat přes jednotné JDBC rozhraní, liší se jednotlivé postupy zejména tvarem JDBC adresy potřebné pro připojení přes JDBC ovladač. Základní spojení s embedded databází disponující JDBC ovladačem se skládá z následujících kroků:

### *Stažení příslušné databázové knihovny*

| SŘBD | Adresa pro stažení |
|---|---|
| **HyperSQL** | *http://sourceforge.net/projects/hsqldb/files/* |
| **SQLite** | *https://bitbucket.org/xerial/sqlite-jdbc/downloads*<br><br>*Poznámka:* Protože je SQLite knihovna napsána v ANSI C, musí být pro přístup z Javy použito JDBC rozhraní nebo jiný databázový wrapper. Na oficiálních stránkách projektu je uvedeno množství dostupných řešení pro Javu a řadu dalších programovacích jazyků. Pro použití s Javou se nejlepší variantou zdá být JDBC ovladač xerial, který obsahuje sqlite knihovnu 3.8.7, kterou obaluje JDBC rozhraním. |
| **Apache Derby** | *http://db.apache.org/derby/derby_downloads.html#Latest+Official+Releases*<br><br>*Poznámka:* Knihovnu s distribucí Apache Derby je možné nalézt v adresáři JavaDB v rámci JDK 1.7 a JDK 1.8. Alternativně je možné stáhnout kompletní distribuci z oficiálních webových stránek projektu. |
| **H2** | *http://www.h2database.com/html/download.html* |

**Tabulka C.1:** Přehled oficiálních webových stránek porovnávaných SŘBD

### *Nastavení cesty k databázové knihovně*

Cestu je možné trvale přidat do proměnné classpath, nebo pokud je program spouštěn z příkazové řádky, je možné dočasné nastavení proměnné classpath při spouštění programu. Uvedený příklad spustí benchmarkovací proceduru pro databázový systém SQLite:

```
Java –classpath build/classes:dist/lib/sqlite/sqlite-jdbc-
3.8.7.jar TpccBenchmark.DbBenchmark <parametry testu>
```

Uvedený příklad bude hledat knihovnu sqlite-jdbc-3.8.7.jar v adresáři *dist/lib*, a vlastní spustitelný soubor programu *TpccBenchmark.DbBenchmark* v adresáři *build/classes*. Pokud jsou potřebné soubory umístěny v jiných adresářích, musí být cesty náležitě změněny. Seznam a popis vstupních parametrů zadávaných při spouštění testu je uveden v příloze D.

Pokud je program spouštěn z vývojového prostředí, mohou být požadované knihovny součástí projektu. Například v Netbeans IDE lze přidat knihovny do existujícího projektu následovně : kliknout pravým tlačítkem na příslušný projekt –> zvolit *„Properties“*–> ve zobrazeném vybrat položkou *„Libraries“* –> záložka *„Compile“*–> tlačítko *„Add JAR/Folder“* –> výběr požadovaného jar souboru.

### *Registrace JDBC ovladače*

Tento krok je povinný pouze pro starší JDBC ovladače. Novější ovladače verze 4.0 jsou přilinkovány automaticky, pokud jsou nalezeny v proměnné classpath. Z testovaných databázových systémů je registrace JDBC ovladače nutná pouze u Apache Derby. Provádí se voláním metody:

```
Class.forName.(<java.sql.Driver>);
```

Pro Apache Derby bude načtení JDBC ovladače vypadat následově:

```
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
```

### *Vytvoření spojení s databází*

Vlastní spojení s databází vytváří třída DriverManager metodou getConnection. Je potřeba zadat správnou JDBC adresu pro připojení k databázi. Adresa specifikuje typ režimu databáze (server, embedded, in-memory embedded), název databáze,

identifikační údaje uživatele apod. příklady základních tvarů adresy pro připojení k jednotlivým databázím v embedded režimu, uvádí následující tabulka.

| SŘBD | In-memory databáze | Souborová databáze |
|---|---|---|
| SQLite | *jdbc:sqlite::memory:* | *jdbc:sqlite:* |
| HyperSQL | *jdbc:hsqldb:mem:* | *jdbc:hsqldb:file:* |
| H2 | *jdbc:h2:mem:* | *jdbc:h2:file:* |
| Derby | *jdbc:derby:memory:* | *jdbc:derby:* |

**Tabulka C.2:** Příklady JDBC adres pro připojení k embedded databázi v souborovém a embedded režimu

## *HyperSQL*

Pro připojení k databázi (v embedded i server režimu) používá HSQL JDBC ovladač. Adresa pro připojení k embedded databázi má následující tvar:

```
jdbc:hsqldb:[typ_db:][<cesta_k_db>]<jméno_db>;<parametry_spojení>
```

HSQL automaticky vytvoří novou prázdnou databázi, pokud v příslušném adresáři žádná zadaného jména neexistuje. (platí pro klient-server i embedded databáze). Tuto vlastnost je možné zakázat předáním parametru *ifexists=true;* při vytváření spojení. Příkaz, který vytvoří spojení s databází *testovaciDB* uložené v adresáři */hsql/db*, bude vypadat následovně:

```
Connection c = DriverManager.getConnection("jdbc:hsqldb:file:
testovaciDB;ifexists=true", "SA", "");
```

Pro každé spojení je třeba specifikovat uživatelské jméno a heslo. Pokud tyto údaje nejsou při vytváření spojení s novou databází použity, automaticky se vytváří spojení pro standardního uživatele, který nemá žádné heslo a uživatelské jméno je SA. Tento přístup platí i pro spojení s již existující databází; pokud nejsou parametry zadány, SŘBD automaticky použije připojovací údaje standardního uživatele. Následující příkaz tedy vytvoří spojení se stejnou databází a stejnými vlastnostmi jako výše uvedený:

```
Connection c = DriverManager.getConnection( "jdbc:hsqldb:file:
testovaciDB;ifexists=true");
```

Některé další příklady tvaru adresy pro připojení k embedded databázi spolu se stručným popisem uvádí následující tabulka. Další, včetně adres pro připojení k databázovému serveru, lze dohledat v podrobné oficiální dokumentaci[43] v kapitole 12.

| Příklad adresy | Popis |
|---|---|
| jdbc:hsqldb:file:testovaciDB;ifexists=true<br>jdbc:hsqldb:mem: testovaciDB;ifexists=true<br>jdbc:hsqldb:res: testovaciDB;ifexists=true | Adresa pro připojení k souborové / neperzistentní (mem) / res databázi testovaciDB v embedded režimu. Pokud v pracovním adresáři aplikace databáze testovaciDB neexistuje, nebude automaticky vytvořena. Uživatel není uveden – hsql připojí standardního uživatele |
| jdbc:hsqldb:file:testovaciDB;ifexists=true;<br>user=User;password=pass | Adresa pro připojení uživatele se jménem *User* a heslem *pass*, k souborové databázi *testovaciDB*. Pokud v pracovním adresáři žádná databáze *testovaciDB* neexistuje, nebude automaticky vytvořena. |
| jdbc:hsqldb:mem: testovaciDB | Neperzistentní (mem) databáze jsou uloženy pouze v paměti, po dobu běhu programu. Cesta k databázi se proto sestává pouze z jejího názvu. |

***Tabulka C.3:*** *Příklady JDBC adresy pro připojení k HyperSQL databázi v embedded režimu*

# D. Ovládání testovacího programu

### *Předpoklady pro spuštění*

Aplikace je naprogramována a testována v 64bitové verzi operačního systému Linux, distribuce Lubuntu. Použití v operačním systému Windows se nepředpokládá, z důvodu obtížné kontroly velkého množství běžících procesů, které mohou mít vliv na výsledky benchmarku a celkově výrazně vyšší náročnosti na systémové zdroje oproti OS Lubuntu, který umožňuje pouze konzolový režim. Spouštění a zpracování výsledků programu je prováděno pomocí bash skriptů, které v OS Windows nebudou fungovat bez použití speciálního Linuxového emulátoru (například Cygwing[44]). S výjimkou knihovny obsahující JDBC ovladač a C knihovnu SŘBD SQLite, jsou všechny přibalené knihovny databázových systémů spustitelné i na 32bitové verzi systému. V případě SQLite je pro 32bitovou verzi systému nutné použít starší verzi *sqlite-jdbc-*

[43] http://hsqldb.org/doc/2.0/guide/guide.pdf
[44] https://www.cygwin.com/

*3.8.6.jar*. Vývoj probíhal v Javě s JDK 1.8 (distribuce Oracle), s použitím vývojového prostředí Netbeans.

### *Spouštění testu*

Ke spouštění jsou připraveny dva bash skripty – *BenchParam.sh* a *runTransactionTest.sh*. První skript provádí vlastní spuštění jednoho testu, vyžaduje tyto vstupní parametry:

| Parametr | Popis |
|---|---|
| **Typ testu** | *-m* pro testování databáze v in-memory režimu. Souborová databáze bude vytvořena volbou *-f*. |
| **DBMS** | Název testovaného databázového systému. Povolené hodnoty jsou sqlite, hsql, h2, derby. |
| **Název DB** | Název nově vytvořené testovací databáze. |
| **W** | Udává počet skladišť, pro která budou vygenerována vstupní data. Dovoluje škálovat počáteční velikost testovací databáze. |
| **Čas** | Délka Tpcc testu v minutách. |
| **Seed** | Parametr ovlivňující generátor náhodných čísel. Stejná hodnota zajistí přesnou shodu vygenerovaných vstupních dat během jednotlivých testů.<br><br>*Poznámka: pro zajištění maximální srovnatelnosti testu by měla být hodnota parametru seed nastavena shodně u všech testů které budou porovnávány.* |

**Tabulka D.1:** vstupní parametry programu

Skript *BenchParam.sh* je určen pro automatické provedení více testů za sebou. Obsah skriptu *BenchParam.sh*, který provede 120minutový transakční test všech podporovaných databázových systémů v režimu s persistentním ukládáním dat, při počáteční velikosti databáze W = 5 je následující:

```
#!/bin/bash

# Před každou sadou testů zapíše řádek s hlavičkou do výstupních souborů s výsledky testu
echo  "Date;DBMS;Mode;Test type;W;time;TotalTPCC;Total NewOrder
Transactions" >> TpccLog.csv
echo  "Date;DBMS;Mode;Test type;W;time;total inserts">> InsertsLog.csv
echo  "Date;DBMS;Mode;Test type;W;time;total inserts">>TpmLog.csv

#vlastní spouštění jednotlivých testů.
./runBench.sh -f derby derby.db  5 120 1
./runBench.sh -f sqlite sqlite.db 5 120 1
./runBench.sh -f hsql hsql.db 5  120 1
./runBench.sh -f hsql hsql.db 5 120 1
#Po dokončení všech testů zpracuje výstupní soubory skriptem ProcessLog.sh
./ProcessLogs.sh
```

***Ukázka D.1:*** *Příklad obsahu skriptu pro automatické spouštění testů*

### *Výstupy*

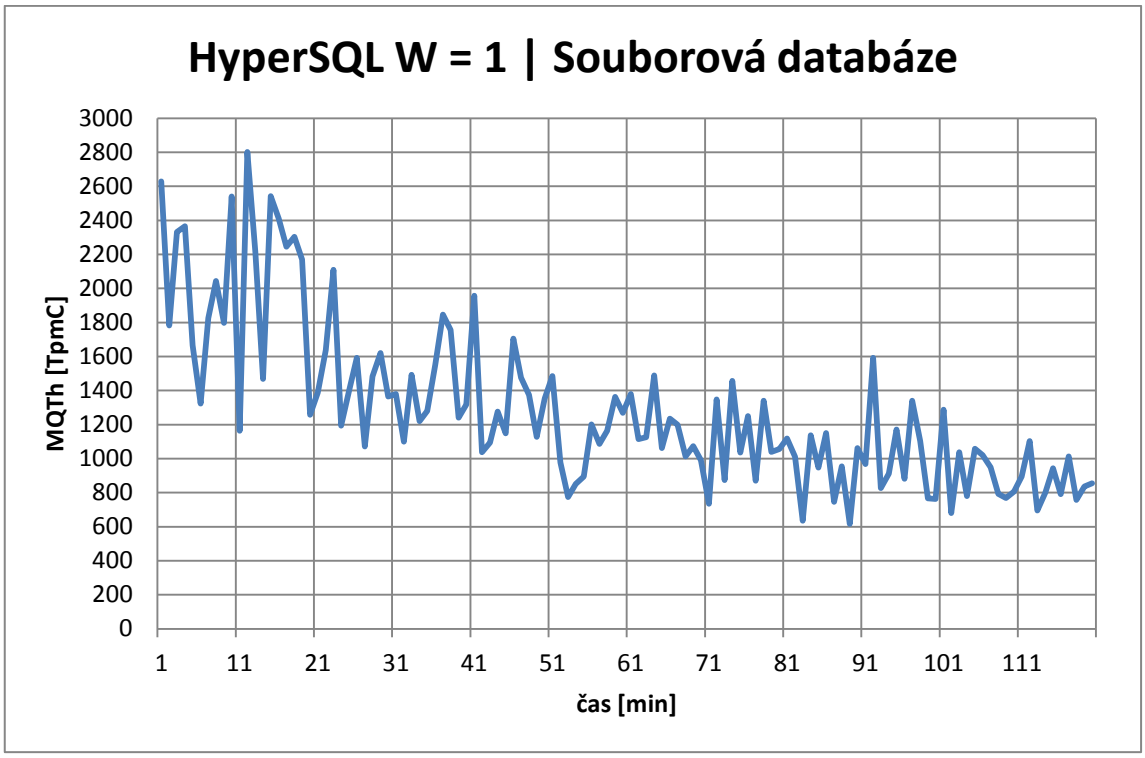
Pokud bude spuštěn test s parametry podle výše uvedeného příkladu skriptu *runBench.sh*, zůstanou po dokončení všech čtyř testů v adresáři, ze kterého byl skript spuštěn, tyto výstupní soubory:

| Soubor | Popis |
|---|---|
| **TpccLog.csv** | *Soubor s naměřenými výsledky v průběhu TPC-C testu.* Každý řádek (s výjimkou řádku s hlavičkou logu) odpovídá jednomu dokončenému testu. |
| **InsertsLog.csv** | Obsahuje výsledky měření rychlosti naplnění testovací databáze. Opět platí, že každý řádek mimo hlavičky logu patří k jednomu provedenému testu. Zaznamenána je celkový čas plnění databáze v milisekundách a počet vložených řádků. |
| **TpccSummary.csv** | Tento soubor vznikne zpracováním výstupních informací ze souboru *TpccLog.csv* skriptem *ProcessLog.sh,* který je volán po dokončení všech měření. V souboru jsou stejné výsledky měření, ale uvedené hodnoty jsou souhrnné pro všechny testy se shodnými parametry DBMS a W. (Například sečte dosažené výsledky testů provedených s databází SQLite u škály databáze W = 5. Dále je přidán nový sloupec TpmC, s vypočtenou průměrnou hodnotou dosažené propustnosti MQTh.<br><br>Poznámka: pokud nebude spuštěn skript ProcessLogs.sh, který je ve výše ukázce skriptu *BenchParam.sh* spuštěn po skončení všech testů, soubor TpccSummary nebude vytvořen.<br><br>*Poznámka2:* Měření by měla probíhat zvlášť pro databáze spuštěné v in-memory režimu a souborové databáze. Skript zpracovávající souhrnné výsledky měření nerozlišuje režim databáze, a v soubory by proto byly uvedeny špatné výsledky sečtené dohromady pro oba režimy databáze. |
| **InsertSummary.csv** | Stejný případ jako soubor TpccSummary.csv, ale jedná se o souhrnné výsledky časů vytváření počáteční databáze vygenerované skriptem ProcessLog.sh z InsertsLog.csv.<br><br>Poznámka: Opět platí, že soubor nebude vytvořen, pokud nebude spuštěn skript ProcessLog.sh. |
| **Graphs.pdf** | Pro přehlednější grafické zpracování je vhodné otevřít soubory se souhrnnými výsledky například v programu Matlab, nebo Excel, které umožňují snadné zpracování csv souborů. |

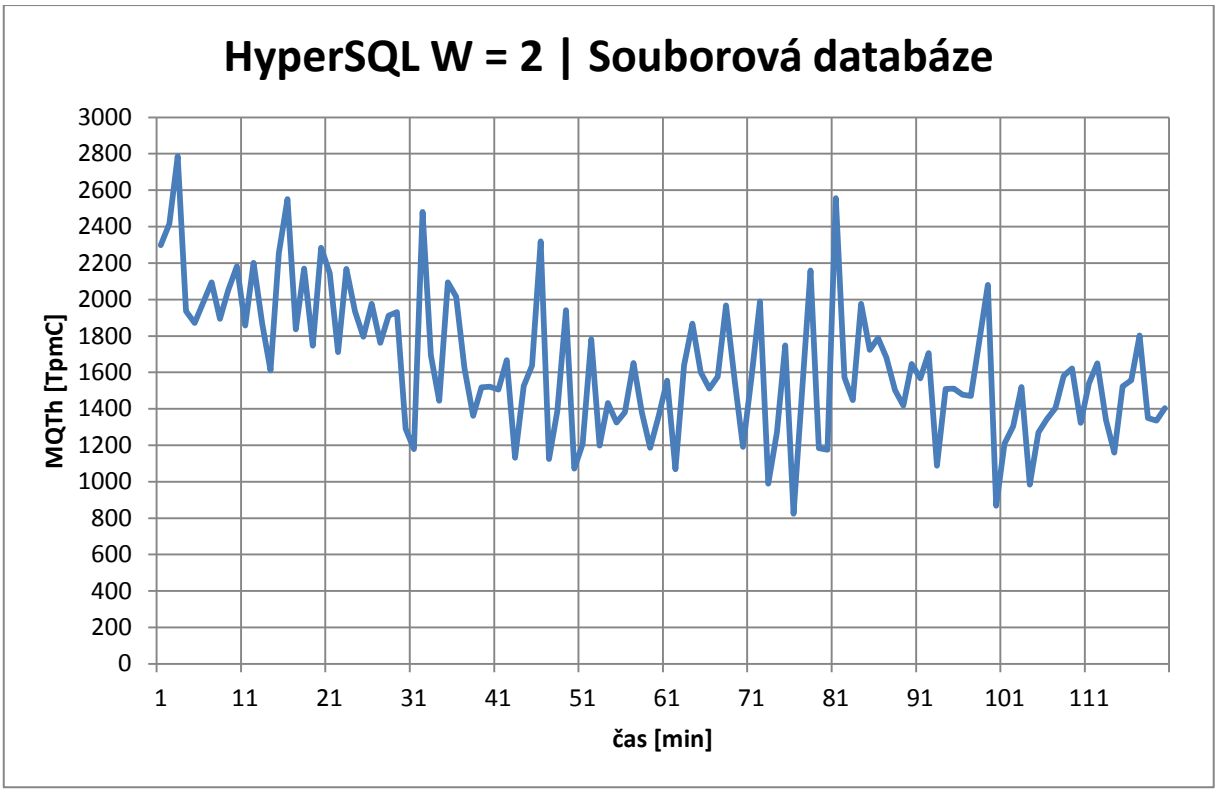
**Tabulka D.2:** vstupní parametry programu

# E. Naměřené hodnoty MQTh

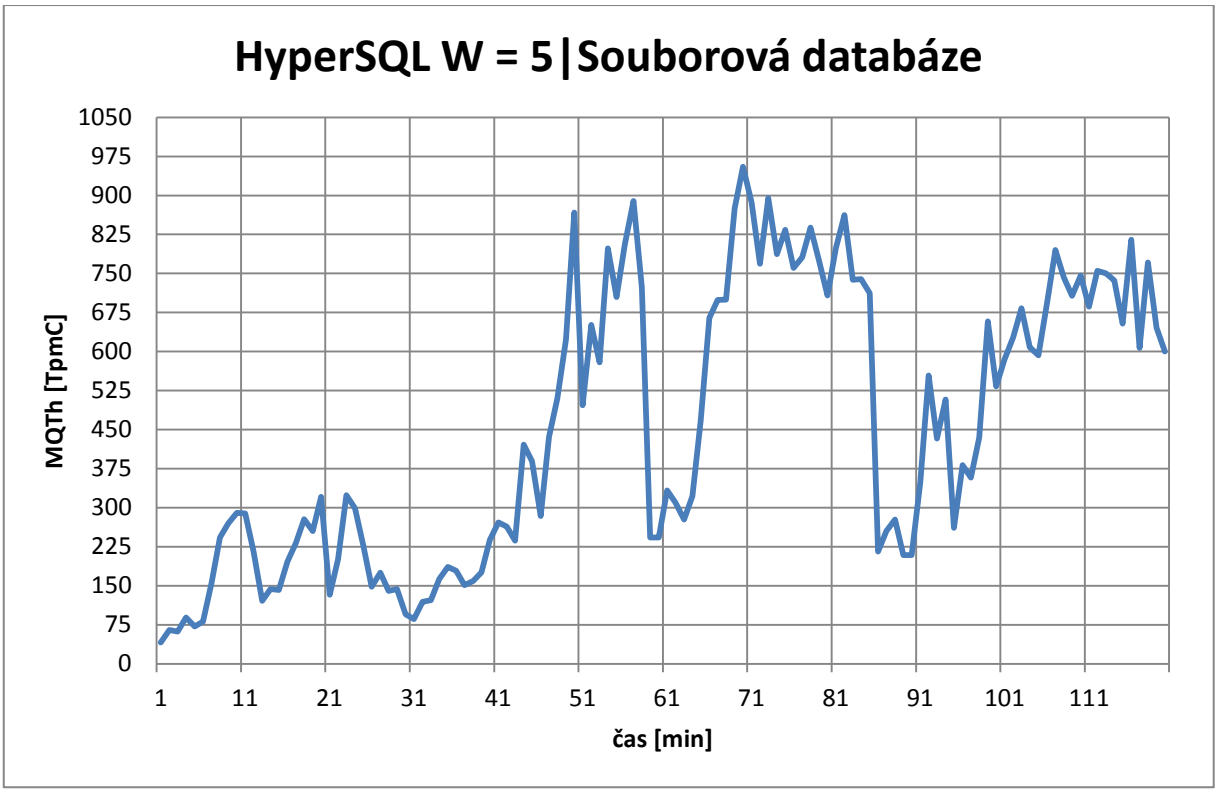
Následující grafy zachycují vývoj hodnoty propustnosti TPC-C testu MQTh [Tpm] během průběhu transakčního testu. Naměřené hodnoty znázorněné hodnoty vyjadřují počet „Nových objednávek“, které byl schopen databázový systém provést v konkrétní minutě testu. Graf zobrazuje chování databázových systému v čase. OsaZáznam chování databáze v čase



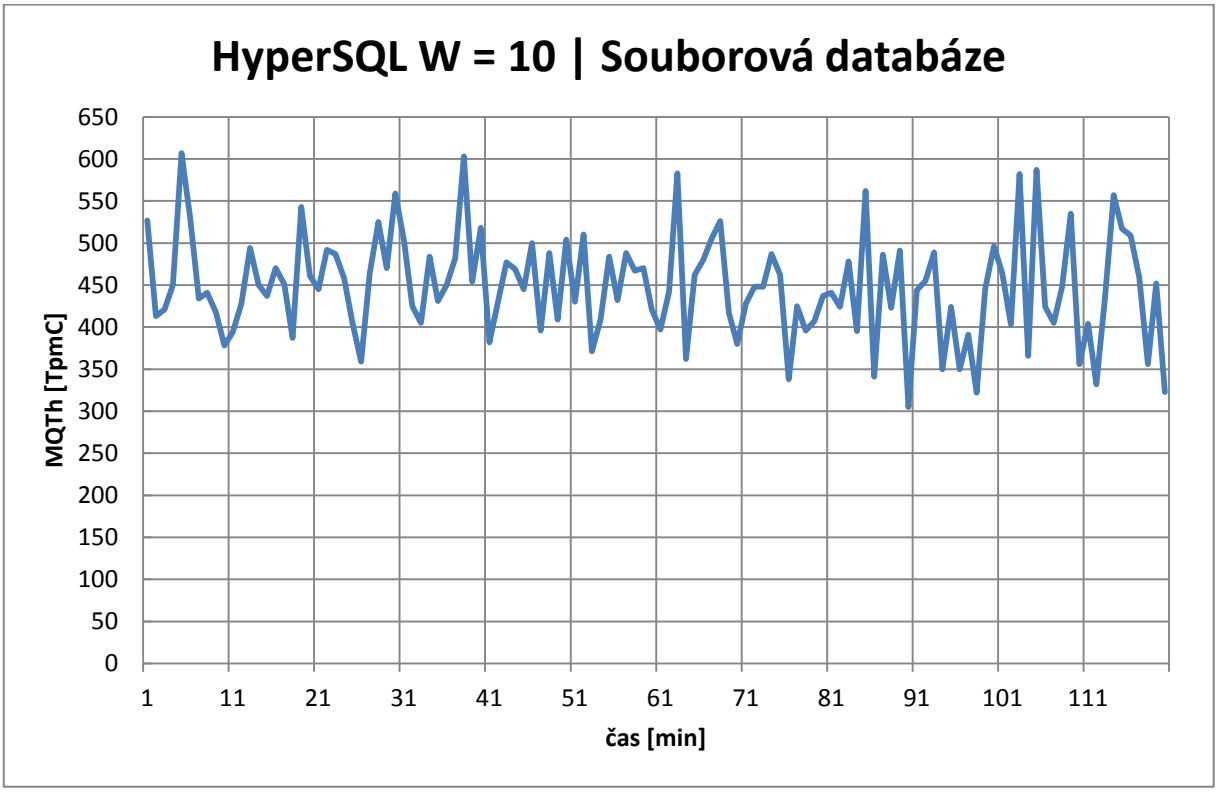

***Graf E.1:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD SQLite při počátečním parametru škály testovací databáze W = 1 v režimu s perzistentním ukládáním dat*



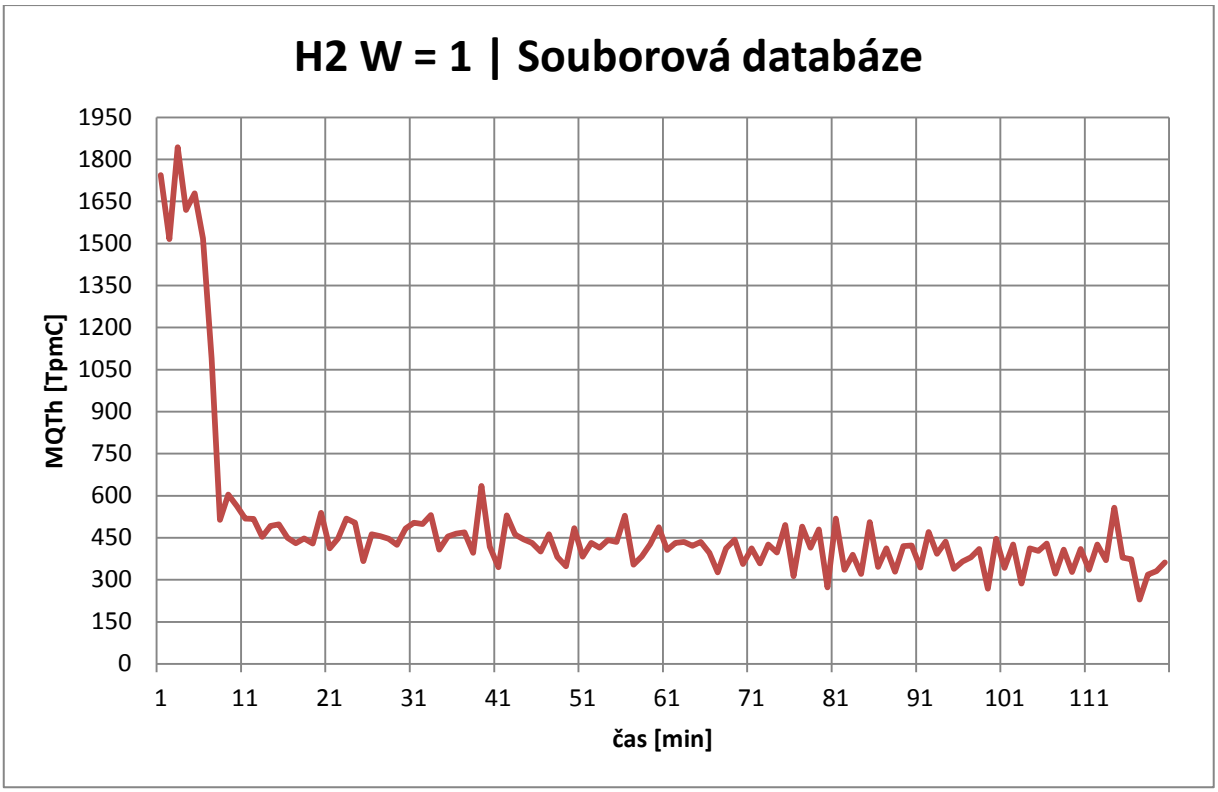

***Graf E.2:*** Vývoj hodnoty MQTh během transakčního testu SŘBD SQLite při počátečním parametru škály testovací databáze W = 2 v režimu s perzistentním ukládáním dat



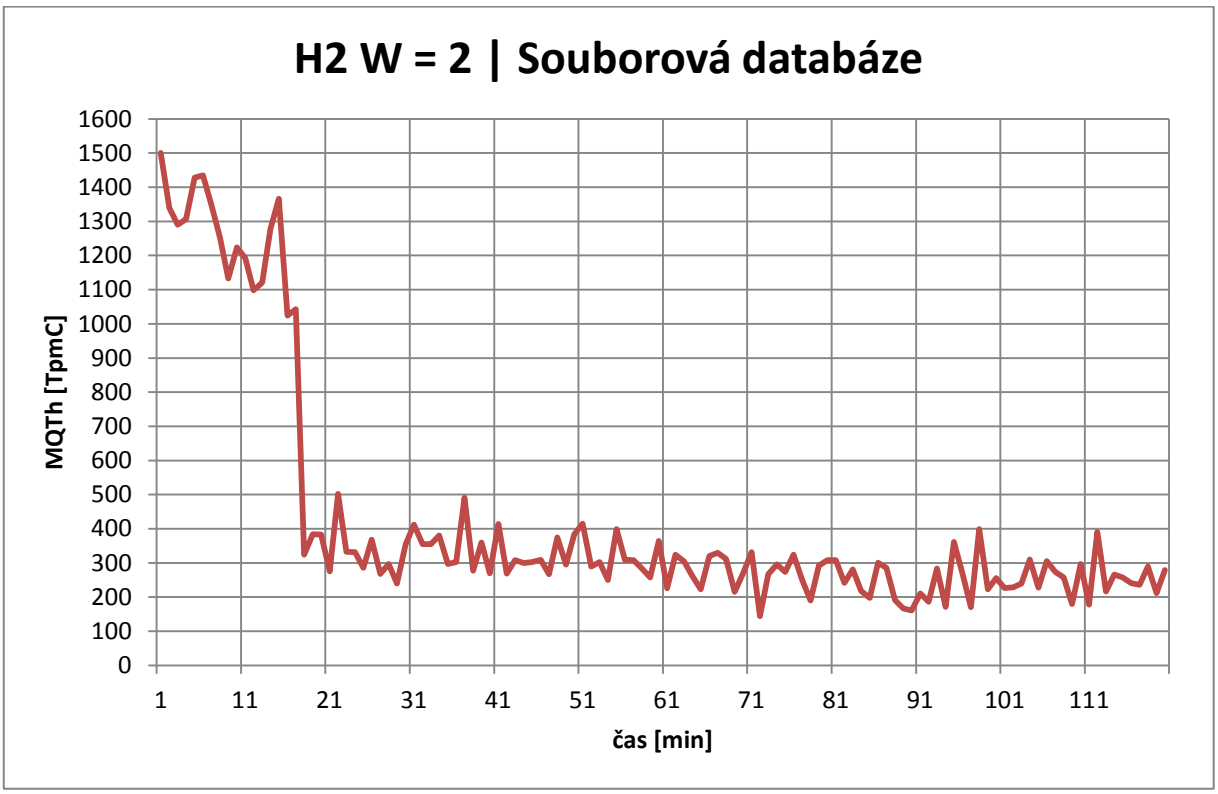

***Graf E.3:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD SQLite při počátečním parametru škály testovací databáze W = 5 v režimu s perzistentním ukládáním dat*



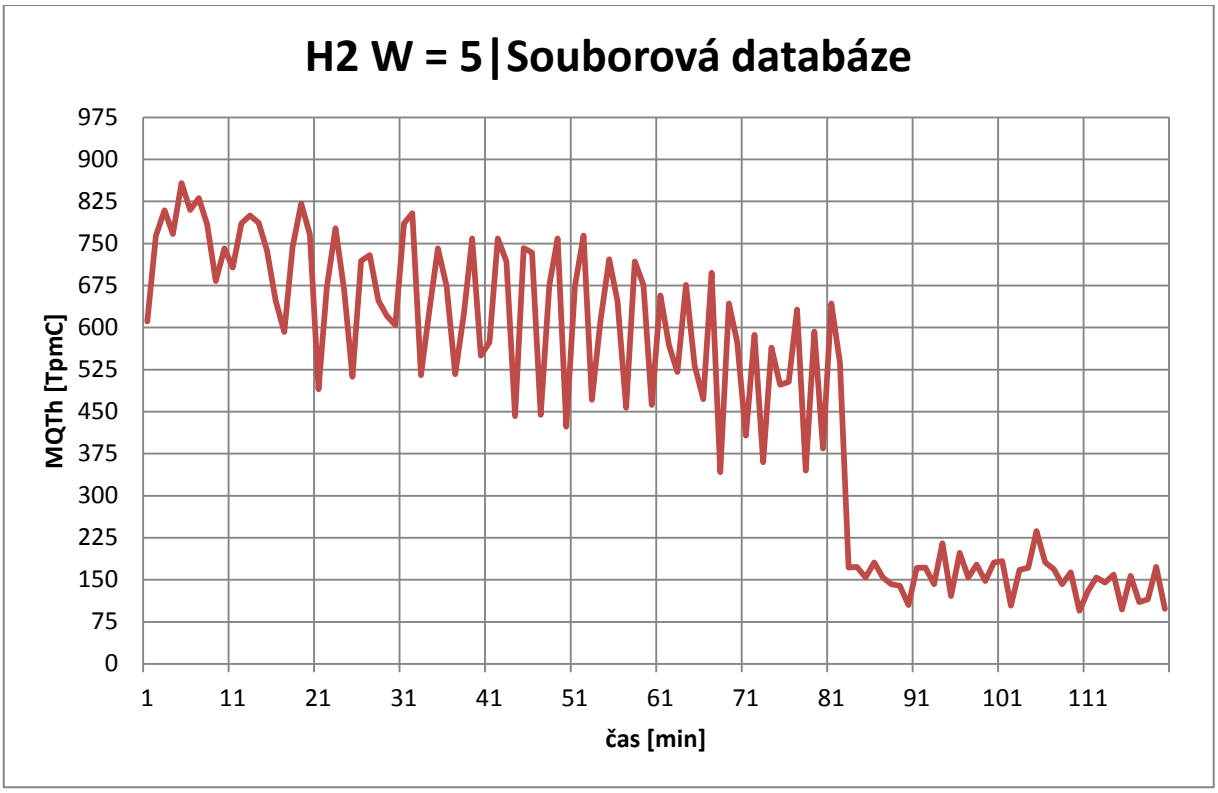

***Graf E.4:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD SQLite při počátečním parametru škály testovací databáze W = 10 v režimu s perzistentním ukládáním dat*



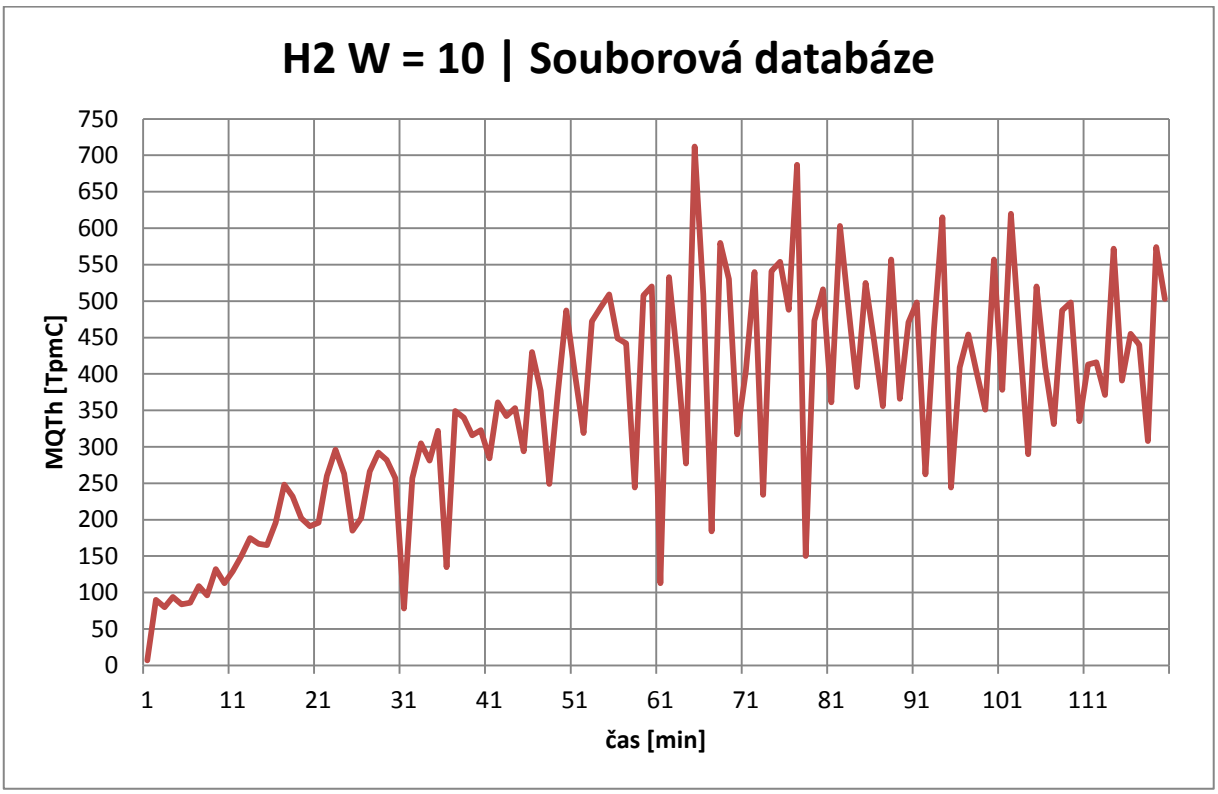

***Graf E.5:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD H2 při počátečním parametru škály testovací databáze W = 1 v režimu s perzistentním ukládáním dat*



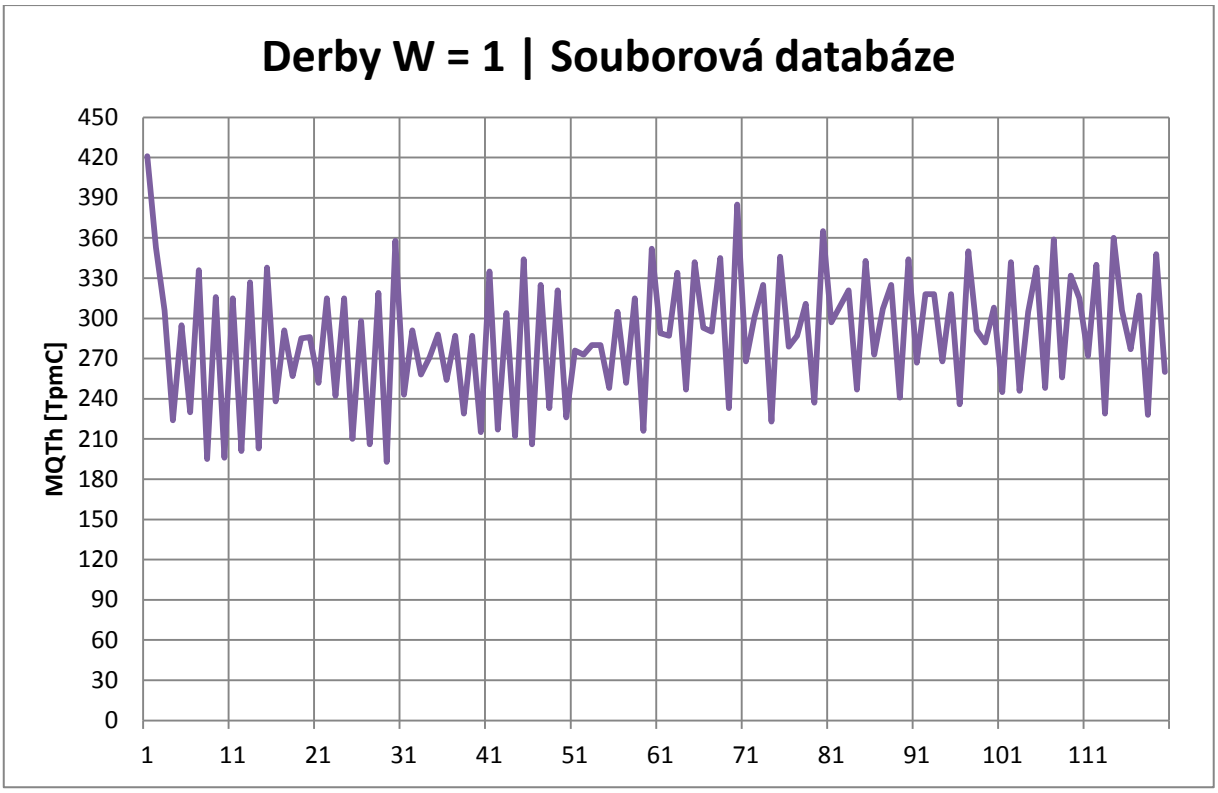

***Graf E.6:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD H2 při počátečním parametru škály testovací databáze W = 2 v režimu s perzistentním ukládáním dat*



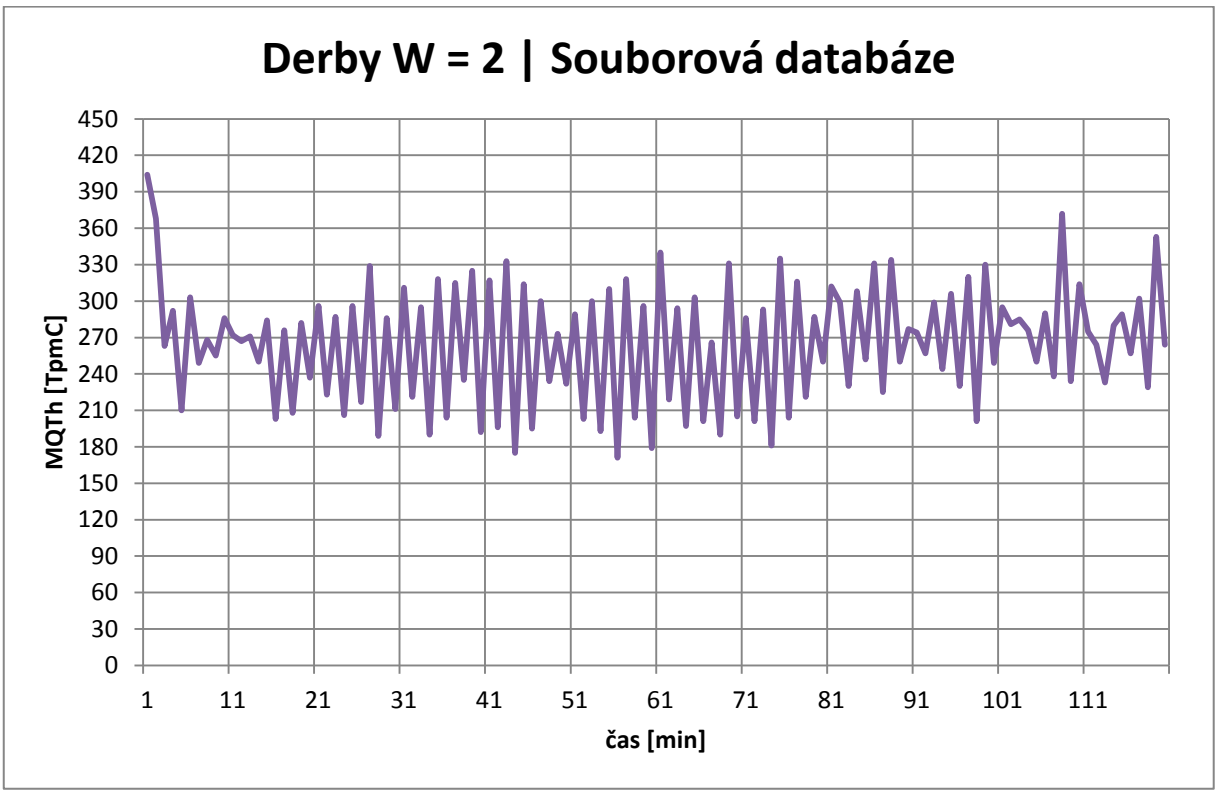

***Graf E.7:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD H2 při počátečním parametru škály testovací databáze W = 5 v režimu s perzistentním ukládáním dat*



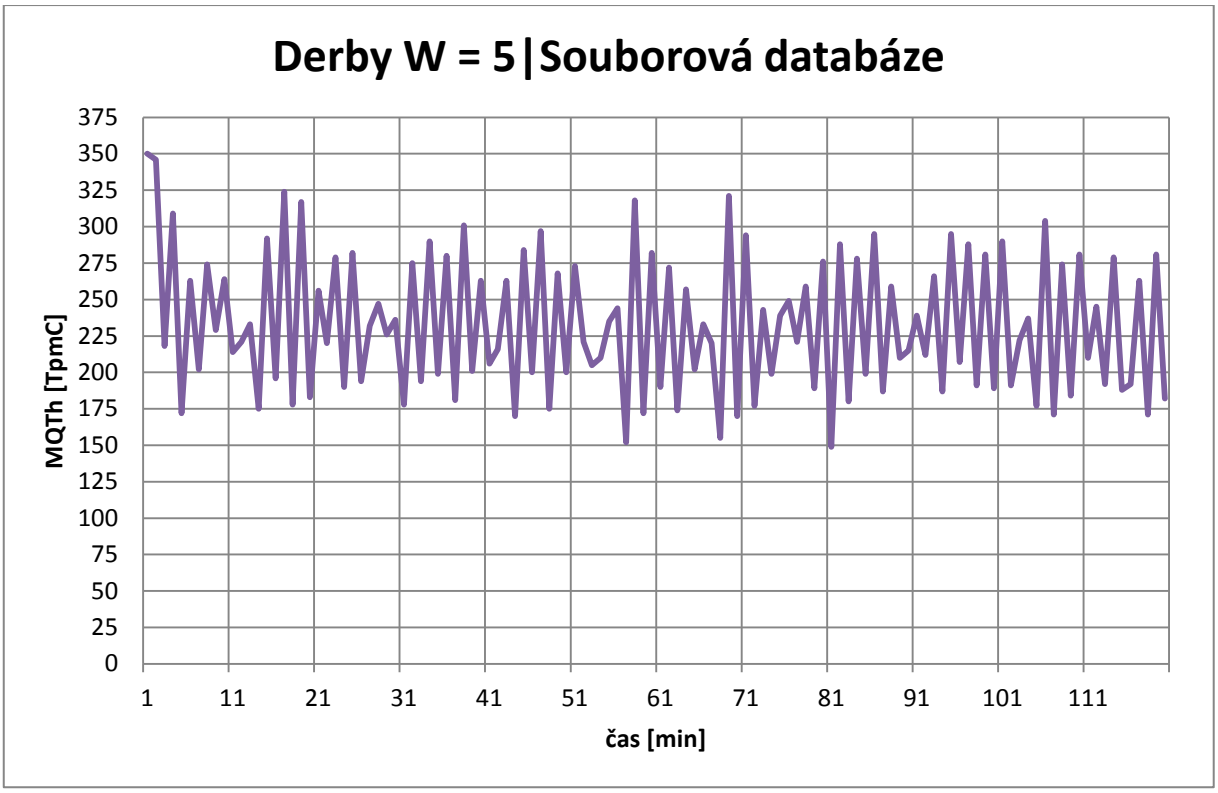

***Graf E.8:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD H2 při počátečním parametru škály testovací databáze W = 10 v režimu s perzistentním ukládáním dat*



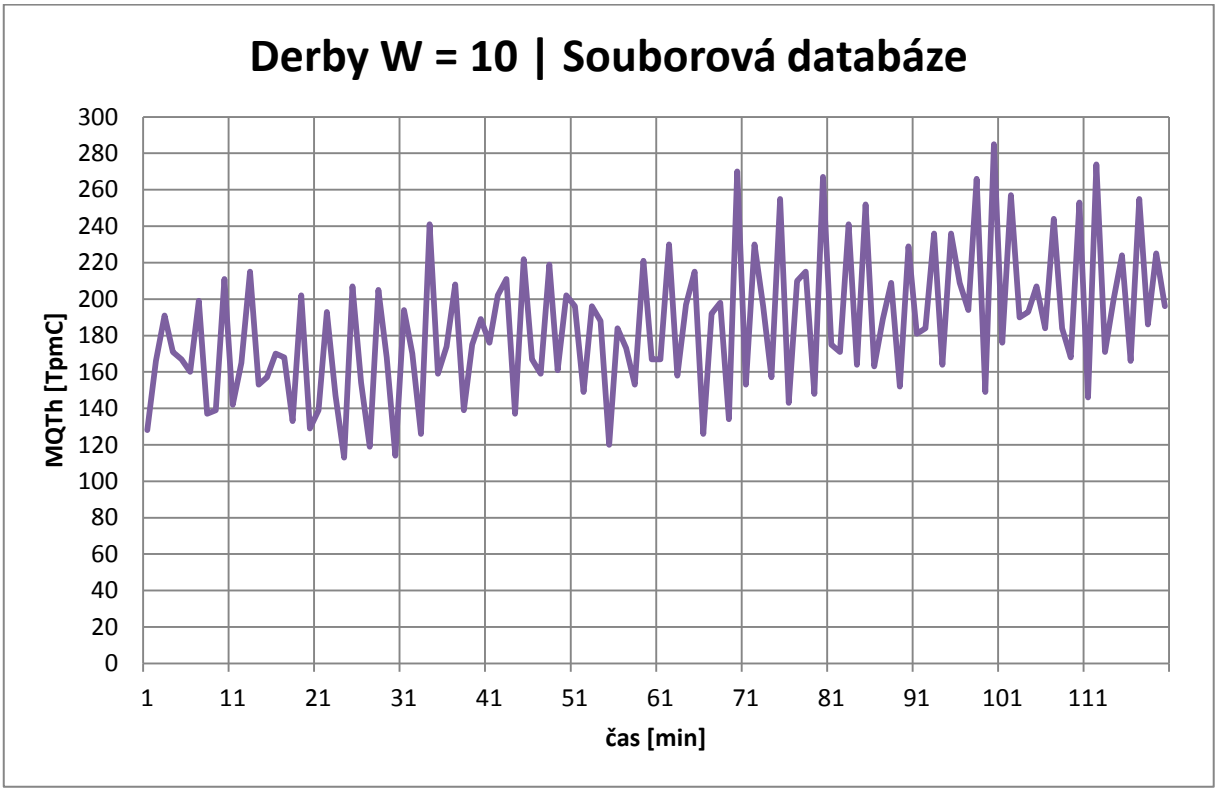

***Graf E.9:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD Derby při počátečním parametru škály testovací databáze W = 1 v režimu s perzistentním ukládáním dat*



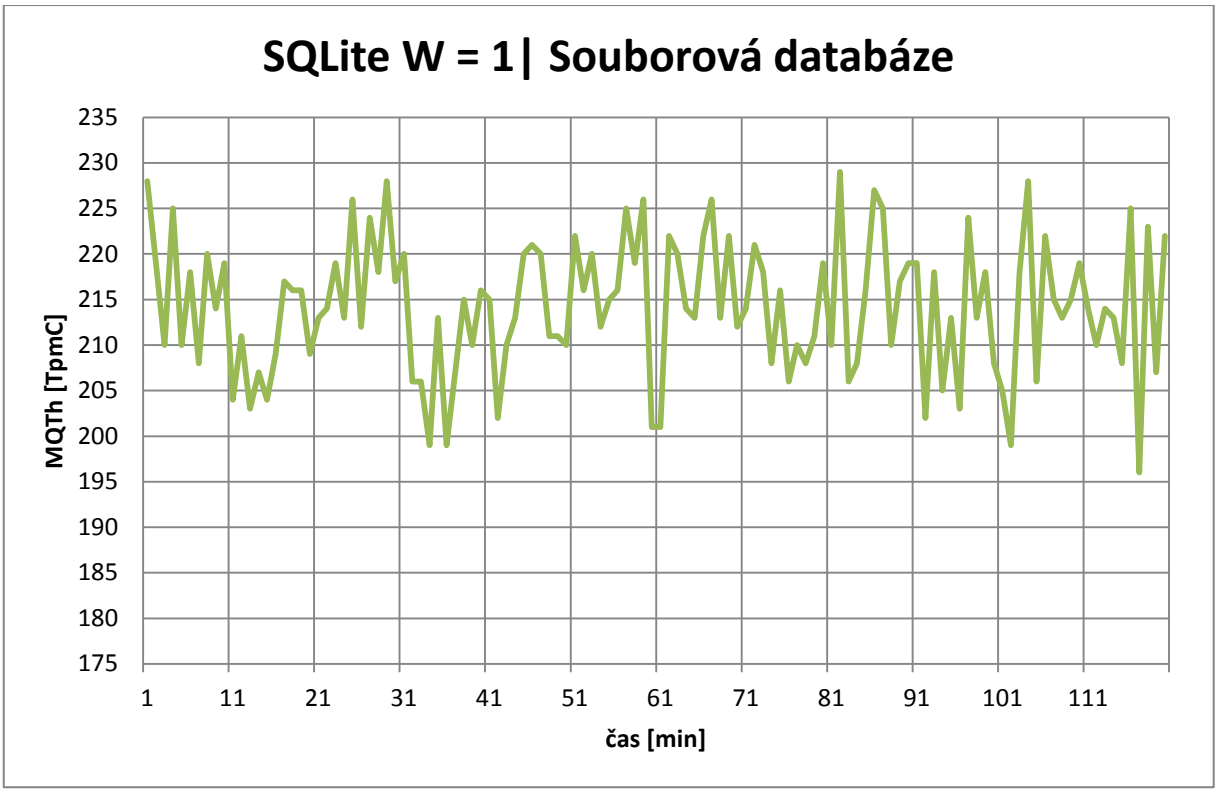

***Graf E.10:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD Derby při počátečním parametru škály testovací databáze W = 2 v režimu s perzistentním ukládáním dat*



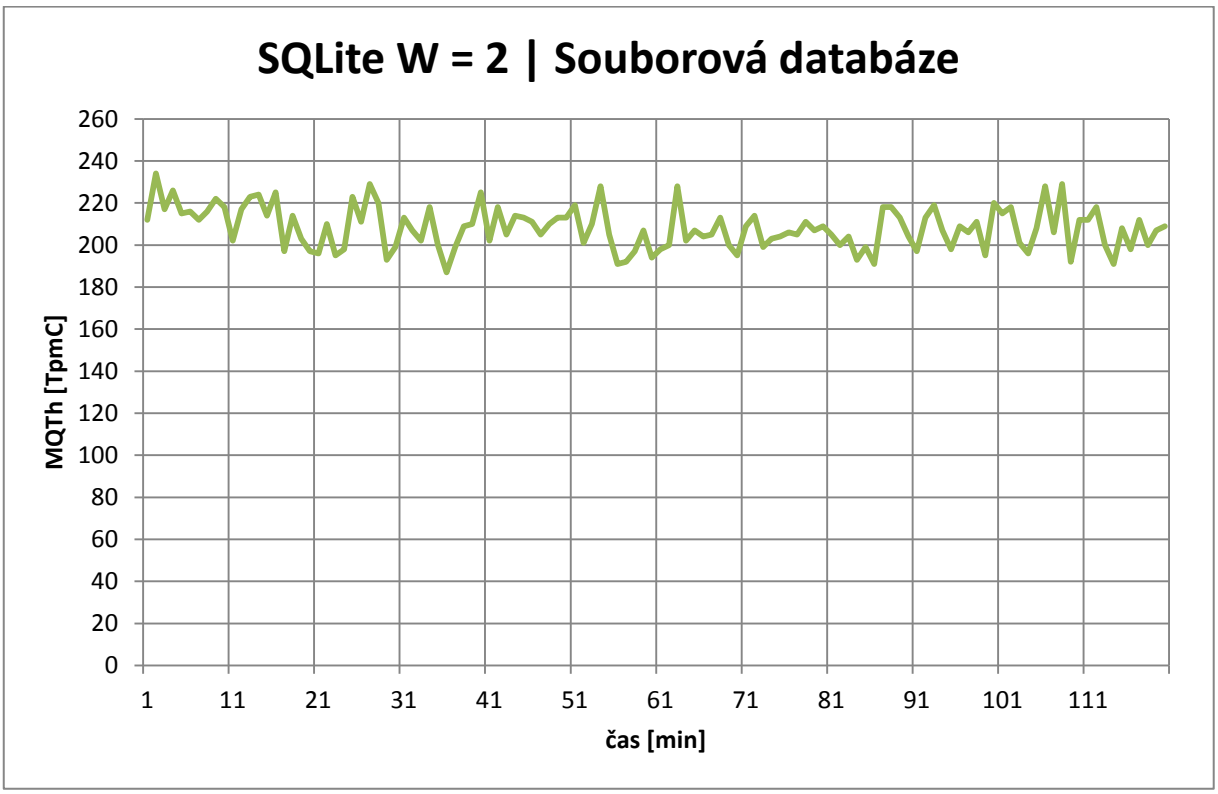

***Graf E.11:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD Derby při počátečním parametru škály testovací databáze W = 5 v režimu s perzistentním ukládáním dat*



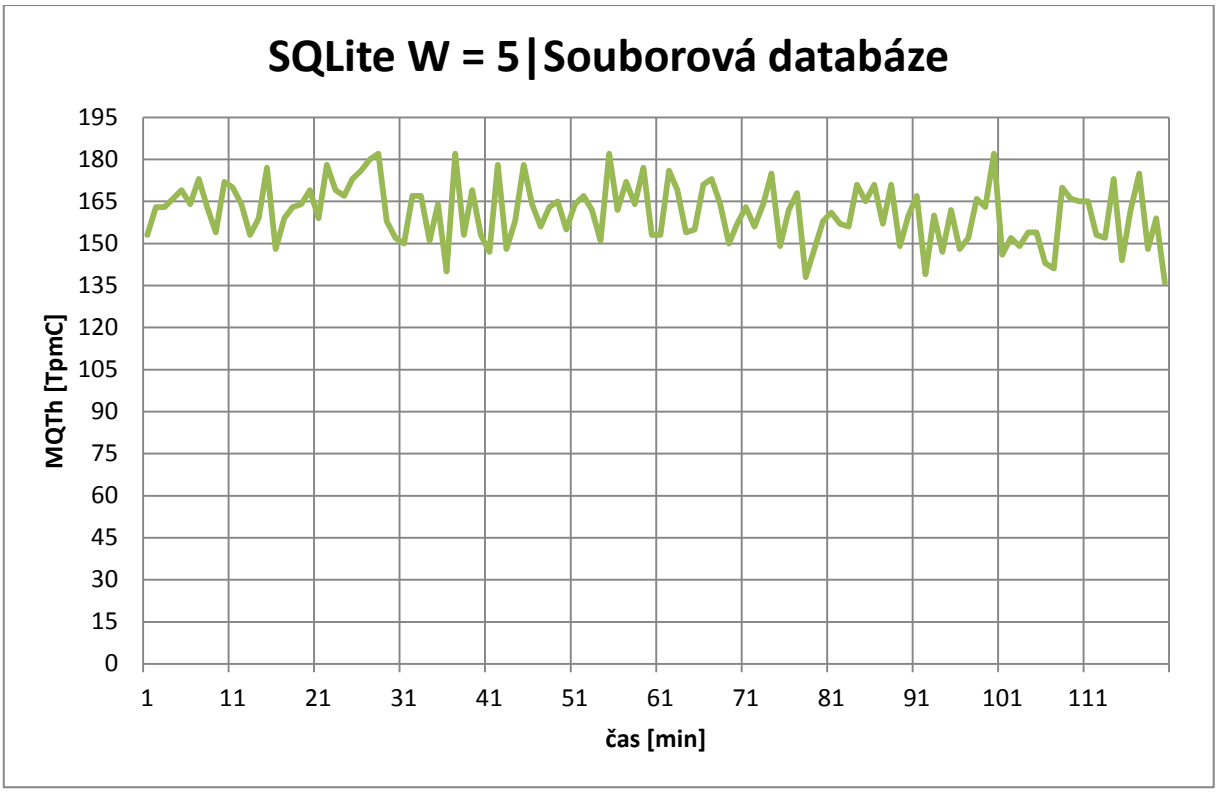

**Graf E.12:** Vývoj hodnoty MQTh během transakčního testu SŘBD Derby při počátečním parametru škály testovací databáze W = 10 v režimu s perzistentním ukládáním dat



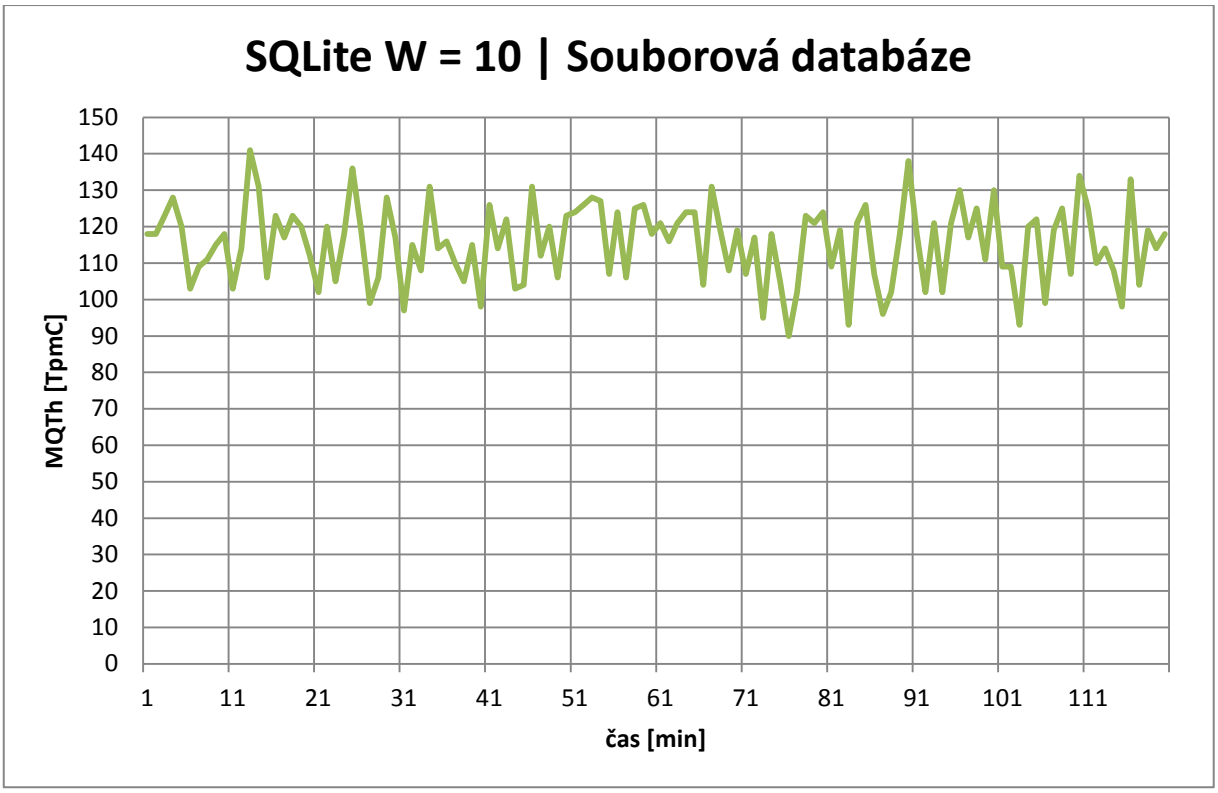

***Graf E.13:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD SQLite při počátečním parametru škály testovací databáze W = 1 v režimu s perzistentním ukládáním dat*




***Graf E.14:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD SQLite při počátečním parametru škály testovací databáze W = 2 v režimu s perzistentním ukládáním dat (Zdroj: vlastní)*




***Graf E.15:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD SQLite při počátečním parametru škály testovací databáze W = 5 v režimu s perzistentním ukládáním dat (Zdroj: vlastní)*




***Graf E.16:*** *Vývoj hodnoty MQTh během transakčního testu SŘBD SQLite při počátečním parametru škály testovací databáze W = 10 v režimu s perzistentním ukládáním dat (Zdroj: vlastní)*