Univerzita Hradec Králové
Fakulta informatiky a managementu
Název katedry

Big data a jejich využití

Bakalářská práce

Autor: Martin Chaloupka
Studijní obor: AI3

Vedoucí práce: Ing. Barbora Tesařová, Ph.D.
Katedra informatiky a kvantitativních metod

Hradec Králové Srpen 2017

Prohlášení:

Prohlašuji, že jsem bakalářskou práci na téma „Big data a jejich využití“ zpracoval samostatně pod vedením Ing. Barbory Tesařové, Ph.D. a s použitím uvedené literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použité literatury.

vlastnoruční podpis

V Hradci Králové dne 16.8.2017 Jméno a Příjmení

Poděkování:
Děkuji vedoucí bakalářské práce paní Ing. Barboře Tesařové, Ph.D. za metodické vedení práce, trpělivost, poskytnuté odborné znalosti, konzultace a nápady na tvorbu této odborné práce.

Anotace

V dnešní době je cena hardwarových součástek výrazně levnější, než tomu bylo v minulosti. Z tohoto důvodu lze ukládat stále větší objemy dat. Tento jev odstartoval výzkum a využití aktuálního fenoménu zvaného „Big Data“. Čím více dat máme, tím přesnější a lepší analýzy se z nich dají udělat. V této práci jsou popsány struktury NoSQL databáze a funkčnost databáze VoltDB, která se zařazuje mezi nejnovější struktury databází zvané NewSQL.

Annotation

Nowadays, the hardware parts prices are distinctively cheaper than it used to be in the past. This is the reason why it is possible to save more and more data. This effect started research and usage of present phenomenon called „Big Data“. More data we have, more accurate and better analysis can be made. In this paper, there will be described the structure of NoSQL databases and the VoltDB database functionality, which is the part of the newest database structure called NewSQL.

Title: Big Data and its usage

Klíčová slova:

Big Data, Databáze, Relační databáze, NoSQL databáze, NewSQL databáze, CAP teorém, VoltDB databáze

Key words:

Big Data, Database, Relational database, NoSQL database, NewSQL database, CAP theorem, VoltDB database

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Množství dat u některých databází přesahuje možnosti relačních databází. Problém nastává převážně v rychlosti zpracování dat. V dnešní době jsou hardwarové způsoby uložení dat výrazně levnější, než tomu bylo v minulosti. Z tohoto důvodu lze shromažďovat, ukládat a zpracovávat či jinak využívat daleko větší množství datových záznamů. Tento jev odstartoval výzkum a využití dnešního fenoménu zvaného „Big Data“. Často řešeným problémem je, že zastaralé „klasické“ databáze na uložení a hlavně rychlé zpracování takového množství dat nemusí stačit. Z tohoto důvodu vznikají nové metody tvorby databázových struktur.

Tato práce slouží k představení a lepšímu pochopení fenoménu Big Data a alternativních způsobů ukládání dat v databázích. Strukturu této bakalářské práce tvoří teoretická a praktická část. Teoretická část popisuje základní funkcionality a jednotlivé druhy těchto nových struktur databází. Praktická část pak navazuje na znalosti prezentované v teoretické části o vytvoření NewSQL databáze v programu VoltDB. Z vytvořené struktury byly provedeny dotazy v jazyku SQL, které posloužily k testování rozdílnosti časů zpracování MySQL a VoltDB databáze.

Po krátkém představení tématu je v kapitole č. 3.1 pojednáno o zjednodušeném úvodu do relačních databází tak, aby byl snadno pochopitelný rozdíl a důvod výzkumu nových způsobů tvorby databází.

Kapitola č. 3.2 je zaměřena na tyto otázky: Co jsou to Big Data? Kde jsou takto rozměrná data využívána? Jakým způsobem jsou tato data získávána a ukládána? Jak jsou data využívána? Které výhody a nevýhody mají jednotlivé způsoby ukládání dat a jednotlivé typy databází? Poté následuje rozbor vlastností jednotlivých typů NoSQL databází oproti relačním databázím. Dále je prezentována další evoluce relačních databází zvaných NewSQL a také jejich rozdíly oproti klasickým relačním a NoSQL databázím. Zde jsou popsány i konkrétní NewSQL databáze VoltDB.

Kapitola č. 3.3 představuje tzv. CAP teorém a jeho důležitost při výběru vhodné databáze pro konkrétní systém.

V poslední teoretické kapitole č. 3.4 je poukázáno na různé alternativy řešení nedostatku výkonu starých relačních databází. K tomuto tématu budou zmíněny zápory i klady u každé z možností.

2 Cíl práce

Cílem bakalářské práce je představit hlavní aspekty přístupu k managementu dat, provést rozbor NoSQL databáze a zdůraznit rozdíly a unikátní vlastnosti oproti relačním databázím. V práci je proveden popis jednotlivých typů nerelačních databází a jejich porovnání s nejnovější databázovou strukturou NewSQL. Tato práce dále upozorňuje na důležité preference, které je zapotřebí si ujasnit před tvorbou databáze. V praktické části bude posouzeno změřením doby zpracování několika příkazů, zda jsou NewSQL databáze vhodné jako náhrada za staré relační databáze.

3 Teoretická část

3.1 Shrnutí vlastností relačních databází

Relační databáze se začaly používat k ukládání a zpracování dat v 70. letech 20. století a využívají se dodnes. A právě kvůli ochranným vlastnostem ACID (Atomicity, Consistency, Isolation, Durability - viz níže podkapitola 3.1.3) lze říci, že jsou spolehlivé. Jejich síla je využívána převážně v běžných databázích, kam se ukládají strukturovaná data.

3.1.1 Relational database management system (RDBMS)

Každá relační databáze má mechanismus pro manipulaci s daty. Tento mechanismus je nazýván RDBMS. Umožňuje vytvářet a manipulovat s tabulkami dat, určovat přístup jednotlivým uživatelům nebo jejich skupinám k datům v tabulkách, apod. Tabulky se skládají ze sloupců a řádků, kde každý sloupec je atribut a řádek je právě jeden záznam. V průsečíku řádku se sloupcem je hodnota atributu.

Aby bylo možné usnadnit vyhledávání, tabulky mohou mít jeden a více unikátních atributů, které spolu tvoří primární klíč. Pokud tuto referenci na objekt používá jiná tabulka, hovoří se o cizím klíči. Další urychlení vyhledávání dat je možné pomocí vytváření indexů. Tyto indexy mohou, ale nemusí být unikátní.

3.1.2 Požadavky pro relační databáze

Lze tedy říci, že databázový systém je relační, pokud splňuje několik integritních omezení:

- **Entitní omezení** – Tabulka nesmí mít duplicitní řádky.
- **Doménové omezení** – Pro každý sloupec musí být určený datový typ, formát a množina hodnot, kterých bude nabývat.
- **Referenční integrita** – Řádky, které jsou využívány ostatními záznamy, nemohou být smazány.
- **Uživatelem definovaná omezení** – Prosazuje některé obchodní podmínky, které nespadají pod předchozí omezení.

3.1.3 ACID

Každé jednotlivé operace s daty jsou nazývány transakce. Ty musí splňovat tzv. transakční vlastnosti ACID, aby bylo dosaženo spolehlivosti. Tyto vlastnosti jsou typicky běžné pro relační databáze. Jejich akronym je:

- **Atomicity** – *„Buď se udělá celá transakce, nebo nic“*. Pokud by se stalo, že v nějaké části by se transakce nezdařila, potom by selhala celá úloha. Nemůže se stát, že by část dat byla úlohou zpracována (změněna) a část ne.
- **Consistency** – Všechna data musí být v souladu se všemi pravidly nebo limity databáze.
- **Isolation** – Vnitřní operace uvnitř jedné transakce nezasahují do jiných operací. To je důvod, proč ostatní transakce zůstanou nedotčeny, když jedna transakce selže.
- **Durability** – Pokud je transakce úspěšná, musí tak zůstat. Nesmí se stát, že by se data ztratila (1).

3.2 Big Data

V dnešní době lze sbírat data z různých zdrojů (např. ze sociální sítě, z webové stránky, z lokální firemní databáze, z různých měřicích a záznamových zařízení, atd.). Vzhledem k současným hardwarovým technologiím je možné tato data i uložit a poměrně snadno a levně zpracovat. Pojem Big Data byl poprvé představen v roce 2005. Tento fenomén řeší problém, který nastává, když je potřeba zpracovat nestrukturovaná data, nebo když zpracováváme tak velké množství dat, že je dnes běžně používané relační databáze nedokáží zpracovat v reálném čase (2). Big Data definicí je mnoho, do určité míry se mezi sebou liší. Podle mého názoru jsou nejvýstižnější následující dvě definice:

Firmy IBM a Oracle, které pečují o jedny z nejrozsáhlejších databází v Evropě, vidí Big Data jako souhrn čtyř aspektů:

- **Volume** – Odkazuje na množství nahromaděných dat.
- **Velocity** – Odkazuje na časový limit, ve kterém mají být data zpracována. Některá data musí být zpracována v reálném čase.
- **Veracity** – Odkazuje na stupeň důvěryhodnosti dat. Většinou jde hlavně o zdroj, ze kterého jsou data získána.
- **Variety** – Odkazuje na způsob uložení dat, jsou-li data strukturována nebo nestrukturována (3).

Další výstižná definice je, že *„Big Data mohou být popsána jako data, která přesahují možnosti zpracování konvenčních databázových systémů. Data jsou příliš velká, často se měnící nebo nejsou uzpůsobena pro strukturu dané databázové architektury“* (2).

To je vzhledem ke způsobu sběru dat nejčastější výskyt Big Dat. Většina firem hromadí data i mimo svůj systém, ať už z internetu, od uživatelů nebo klientů. Tato data jsou pak ve velkém množství, nestrukturovaná. To znamená, že bude-li zapotřebí taková data analyzovat, musí být nejprve upravena do patřičného formátu. Databáze používající nestrukturovaná data budou popsány níže.

3.2.1 Chybovost – nepřesnost měření

Dokud potřeby průmyslového zpracování dat nepřesahovaly určitá množství, používaly se k jejich zpracování malé databáze, v nichž logicky bylo zapotřebí omezit chyby na minimum a tím zvýšit důvěryhodnost analýzy dat. Výsledkem bylo dosažení realistických výsledků. Když ovšem výsledky analýz nebudou vyžadovat maximální přesnost, budou zmenšeny nároky na přesnost zpracování a připustí se větší tolerance k chybám, může být nasbíráno mnohem více dat. Při ukládání a zpracování těchto dat může být využito několik levnějších (tj. i méně přesných) senzorů, spíše než jeden kvalitní a tudíž velmi drahý. To je velké plus pro zpracování dat statistickými metodami. V mnoha případech využívání Big Dat bylo potvrzeno, že *„Neplatí jen „raději více dat než méně“, ale někdy dokonce platí, že „raději více horších dat než méně lepších“.“* (4). Pokud se jedná o zpracování Big Dat, výsledky analýzy dat nemusí být vysoce přesné, ale spíše pravděpodobné. To je u většiny analýz více než dostačující.

Hlavní metodou statistického předvídání budoucích jevů je hledání korelací. Tato metoda je tím účinnější, čím větší vzorek dat zkoumáme. Korelace prezentuje měření síly vztahu mezi dvěma jevy (jaká je pravděpodobnost a za jakých podmínek se tyto jevy vyskytují současně) (4).

3.2.2 Příklady využití Big Dat

Příklady využití Big Dat jsou všude kolem nás. Lidé je využívají každý den, aniž by o nich něco věděli. V této kapitole bude rozebráno několik příkladů využití Big Dat v reálném životě.

Jedním z nejpopulárnějších programů, který využívá zpracování Big Dat, je překladač od společnosti Google. Tento překladač je doposud nejlépe fungujícím programem sloužícím k přeložení textových záznamů z a do mnoha jazyků. Předchozí světové pokusy o vytvoření takového překladače byly založeny na tom, že obsahovaly miliony poctivě přeložených vět. Google pro tvorbu tohoto překladače využil celý globální internet doplněný o další zdroje. Nakonec se společnosti Google povedlo nahromadit do databáze tohoto programu obsahově 95 miliard anglických vět s pochybnou úrovní překladu. Tady na tom příkladu je vidět množstevní rozdíl. Překladač od Googlu nefunguje líp proto, že má chytřejší

algoritmy, ale proto, že platí, čím více dat se využije, tím přesnější bude výsledek (4).

Další příklad využití Big Dat lze vidět u společnosti Microsoft. U nejvíce využívaného editoru textových dokumentů zvaného „Microsoft Office Word“ se nachází funkce kontrola gramatiky. Tvůrci této funkce převzali čtyři běžné algoritmy strojového učení a poté do databáze vložili až miliardu slov. Výsledky všech algoritmů byly znatelně účinnější, než tomu bylo doposud (4).

Dalším případem je zahraniční nákupní centrum Walmart. Nákupní gigant nahlédl do záznamů svých obchodních transakcí, aby našel zajímavé korelace. Zaměřil se na jednotlivé položky a cenu celkového nákupu. Dalšími hledanými kritérii byly například venkovní počasí a denní doba nákupu. Touto analýzou bylo zjištěno, že dojde ke zvýšení prodeje baterek a sladkých sucharů „Pop-Tarts“ právě když lidi očekávají bouřku nebo hurikán. Walmart tedy umístil krabice s daným zbožím blíže k pokladnám, když bylo předpovězeno špatné počasí. Tím podpořil svůj marketing a ulehčil nákup svým zákazníkům.

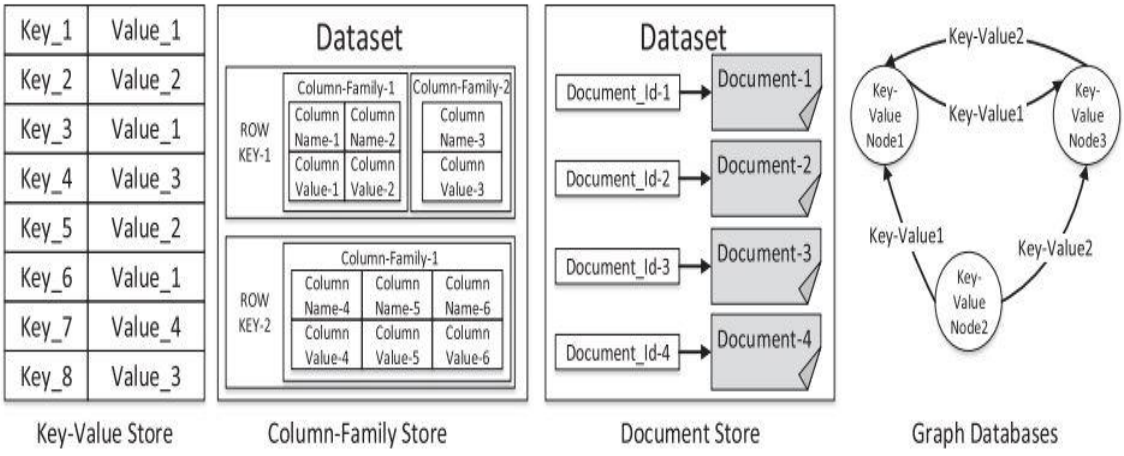
Posledním zde zmíněným případem je využití Big Dat u kurýrní firmy UPS. Firma UPS zavedla prediktivní analýzu tak, že nasadila na jednotlivé díly v autě měřící senzory. Odborníci totiž vycházeli z logického uvažování, že obvykle jednotlivé díly v automobilu selhávají postupně a ne najednou. Z velkého množství dostupných dat zjištěných z dlouhodobých sledování lze zjistit signálové výchylky, ze kterých se může nalézt pravděpodobnost poruchovosti součástky v blízké době. Z tohoto důvodu je v dnešní době firemní technik schopen vyměnit danou součástku u konkrétního automobilu ještě předtím, než dojde k poruše. Firma UPS je tak schopna předejít mnoha přepravním problémům a doručit zásilku včas.

3.2.3 NoSQL databáze

Je-li zmíněn pojem Big Data ve stylu nestrukturovaných dat, tak se většinou při jejich ukládání a zpracovávání využívá právě NoSQL struktura databází. Z důvodu pokročilé technologie lze libovolná data tímto způsobem ukládat a udržovat velmi levně. Jejich vývoj a využití bylo podmíněno tím, že většina dříve užívaných relačních databází bohužel není schopna zpracovávat nestrukturovaná data nebo strukturovaná data v požadovaném čase. Odpovědí na tento problém jsou právě NoSQL databáze. Tato zkratka je ale matoucí. Ve skutečnosti je myšleno „Not Only SQL“, což znamená využití nejen dotazovacího jazyka SQL. V tom spočívá další přednost tohoto typu databází. NoSQL databáze pracují s jinými než tabulkovými strukturami, proto existuje mnoho případů, kde nahradí relační databáze mnohem efektivněji.

NoSQL se dělí na čtyři kategorie, které jsou popsány níže. První tři z nich sdílejí vlastnost tzv. „*Aggregate orientation*“, což lze přeložit jako orientace podle celků. Celkem je myšlen souhrn dat, se kterým se provádí interakce jako s objektem (1).

Na obrázku č. 1 lze názorně vidět typy nerelačních databází.



Obrázek 1 - Různé typy nerelačních databází. Zdroj: (5)

3.2.3.1 Klíč - hodnota databáze

Jednou z nejčastějších implementací NoSQL principu je databáze typu klíč - hodnota. V úložném prostoru takové databáze jsou data uložena jako hashová

tabulka, kde každý klíč je unikátní a hodnota může být číselná i textová (například ve formátu JSON nebo BLOB) (1).

Strukturu takové databáze si lze představit jako asociační mapu nebo slovník. Klíč jednoznačně ukáže na hodnotu a je používán pro ukládání nebo získávání informace v ní obsažené. V hodnotě může být uložen libovolný datový formát (například integer, string, array a mnoho dalších). Taková hodnota garantuje schématem otevřený datový model (schéma neurčuje datové formáty hodnot). Tento typ databáze není vhodný pro scénář zahrnující relace nebo struktury. V takovém případě by muselo obojí být implementováno v klientské aplikaci, která by komunikovala s úložištěm typu klíč-hodnota. (5)

Klient může provádět operace, jako například získat hodnotu pro primární klíč, přidat hodnotu nebo smazat klíč z úložiště. Tato hodnota je pak uložena jako binární celek bez jakékoliv vědomosti, co obsahuje. Pochopení obsahu hodnoty má na starost aplikace (6).

3.2.3.2 Dokumentové databáze

Dokumenty jsou v podstatě složeny jako shluky párů databází typu klíč-hodnota. Databáze pak ukládá a obnovuje dokumenty jako hodnoty, které mají například formát XML, JSON a další. K těmto dokumentům je pak přidělen identifikační klíč, podle kterého se přistupuje k identifikaci (a vyhledávání) jejich pozice v úložišti. Největší výhodou tohoto typu databáze je, že nemá předdefinované schéma. Každý dokument může mít jiné atributy, a přesto mohou tyto dva dokumenty patřit do stejné kolekce. To by byl v relačních databázích problém, protože by bylo nutné vyplňovat i nepotřebné atributy jako hodnotu „null“ (1).

3.2.3.3 Sloupcové databáze

Většina sloupcových databází je odvozena z tzv. „Google Bigtable“, ve které jsou data uložena jako sloupce. Celek je pak složen z několika řádků, kde každému je přiřazeno unikátní číslo řádku. Každý řádek v sobě obsahuje více tzv. „column families“ (v češtině to lze chápat jako skupiny sloupců) a také každý řádek může mít jinou skupinu sloupců. Struktura takové databáze je tedy podobná jako u

databází klíč-hodnota, jen s hlubším rozdělením. Číslo řádku je klíč a skupiny sloupců představují hodnotu. Ale každá skupina sloupců je dalším klíčem pro další sloupce, které jsou v nich obsaženy jako hodnoty. V podstatě se jedná o strukturovanou několikaúrovňovou databázi typu „klíč-hodnota“ uvnitř databáze stejného typu. Obecně lze říci, že sloupcové databáze poskytují efektivnější indexování a dotazování než jiné typy databází. (5)

3.2.3.4 Grafové databáze

Grafové databáze jsou oproti ostatním typům NoSQL databázím speciální. Umožňují ukládat entity a vztahy mezi nimi. V obrazovém prostoru jsou entity zobrazeny jako uzly, a hrany jako vzájemné vztahy těchto entit. Uzly i hrany mohou mít další rozšiřující vlastnosti. Hrany jsou směrově orientovány, díky čemuž se poté zorganizují uzly. Z toho pak lze vyčíst nové vztahy mezi uzly, které na první pohled nebyly patrné (1).

3.2.3.5 BASE

Nerelační databáze běžně nepodporují výše zmíněné transakční vlastnosti ACID. Jejich alternativou jsou BASE vlastnosti pro takové typy databází. Jejich akronym znamená:

- **Basically Available** – Databáze využívající vlastnosti BASE podporují dostupnost dat i přes výskyt několika chyb. Jinými slovy to je taková databáze, která jako celek je přístupná vždy, když je k ní vyžadován přístup, přestože v ní mohou být i nepřístupné části (lze přistupovat k datům, která ještě nejsou zcela v konzistentním stavu).
- **Soft state** – Data v databázi mohou být na určitý čas nekonzistentní.
- **Eventually consistent** – Zaručuje, že po určité době datové úložiště přejde do konzistentního stavu.

Avšak i zde existují výjimky. Například databázový systém CouchDB patří mezi NoSQL databáze, a přesto podporuje ACID vlastnosti (5) .

3.2.3.6 Nevýhody nerelačních databází

NoSQL databáze ve velké většině implementací využívají transakční pravidla BASE. To znamená, že fungují na úkor konzistence. Ta je ovšem pro velké firemní databáze prioritou. Při přístupu více lidí k databázi (například v účetnictví) je nutné mít důležitá data v konzistentním stavu. Naopak v aplikacích sbírajících velké množství informací sloužící k analýze dat je možné oddálení konzistentního stavu dovolit. Z důvodu možnosti ukládání nestrukturovaných a schématem neovlivněných dat je další nevýhodou například doba vyhledání jediného výskytu z milionu případů uložených v datovém úložišti. Mezi dočasné nevýhody se řadí doba vzniku nerelačních databází. Jelikož je využití těchto databází relativně nové, existuje pouze jen hrstka expertů, kteří tvorbu takových databází ovládají a několik málo kvalitních publikací o této problematice (7).

3.2.3.7 Výhody nerelačních databází

Hlavní výhodou Big Dat a současně i NoSQL databází je potenciální zpracování a maximalizace efektivity v kontextu využití velkého množství dat, v množství o několik řádů vyšším než u klasických databází, které lze zpracovávat v různých formátech. Pokud bude dosaženo efektivního zpracování velkého množství získaných a uložených dat, mohou být díky využití statistických metod předpovězeny nejrůznější situace, na které lze včas zareagovat a předejít tím např. selhání některých prvků, které mohou vzniknout v reálných situacích.

Z důvodu schématu otevřené struktury je ukládání nových dat oproti relačním databázím velmi rychlé. Například když je potřeba v relačních databázích uložit jiný případ než je předdefinovaným schématem navržený, musí se zavolat příkaz alter table, který vytvoří hodnotu atributu pro všechna data uložená v této tabulce. V NoSQL databázi se prostě takový případ přidá, aniž by se někde musela vyplňovat hodnota „null“ do přebytečného parametru (7).

Efektivita NoSQL databází je patrná především tehdy, je-li například taková databáze základem samoučícího expertního systému pro diagnostiku základních problémů, kde klienti zadávají do databáze co nejvíce parametrů a systém v reálném čase navrhne řešení podle úspěšnosti nejvíce podobného v minulosti

řešeného problému. Ať je řešení úspěšné nebo neúspěšné, případ se zapíše do databáze a může sloužit k dalšímu využití.

Jiným případem pro využití NoSQL databází může být např. zkoumání a měření mnoha fyzikálních vlastností určité hardwarové součástky a podle chování součástky v minulých případech se zjistí, zda nebude ohrožena funkčnost zařízení v blízké budoucnosti (4).

3.2.4 NewSQL

Je-li potřeba zpracovat strukturovaná Big Data v reálném čase, lze využít nejnovější strukturu databází zvanou NewSQL. NewSQL databáze vznikly jako odpověď na nedostatky relačních databází oproti NoSQL databázím. Zatím není mnoho příkladů jejich využití v reálném životě, jelikož se jedná zatím o nejnovější model databází.

Z předchozího textu je známo, že NoSQL databáze nepodporují vlastnosti ACID, místo kterých se stačilo spokojit s eventuální konzistencí dat. Cílem NewSQL databází je využít spolehlivost garantovanou vlastnostmi ACID, při zachování rychlosti zpracování dat nerelačními databázemi. Jde tedy o kombinaci spolehlivosti relačních databází spojenou s rychlostí zpracování dat NoSQL databází. Tato nová struktura databází se tváří jako relační databáze. A přestože komunikace s databází probíhá za pomoci SQL jazyka a klienti využívají tabulek a vztahů mezi nimi, opravdová vnitřní reprezentace může mít podobu NoSQL struktury (8).

Jednou z předních vlastností, kterou NewSQL sdílejí s NoSQL databázemi je horizontální škálovatelnost. Tím je myšlena schopnost zlepšování výkonnosti systému jednoduchým přidáváním uzlů do clusteru. Většinou to vypadá tak, že jeden z uzlů slouží jako řídící a rozděluje práci mezi ostatní uzly ve společném clusteru. Novější optimalizace relačních databází však mají optimalizace, které se vlastnosti této škálovatelnosti vyrovnají minimálně na malých databázích (například centralizované datové úložiště).

	Starší relační databáze	NoSQL	NewSQL
Podpora relací	Ano	Ne	Ano
ACID transakce	Ano	Ne	Ano
SQL komunikace	Ano	Ne	Ano
BASE transakce	Ne	Ano	Ne
Podpora Big Dat	Ne	Ano	Ano
Předem nedefinované schéma	Ne	Ano	Ne
Horizontální škálovatelnost	Ne	Ano	Ano

Tabulka 1 – Přehled rozdílných vlastností různých databázových struktur.

3.2.4.1 VoltDB databáze

Databáze VoltDB je představitelem nejnovějšího vyvinutého modelu pro tvorbu databázové struktury zvané NewSQL. Architektura této databáze zvládá zpracovat miliony operací za sekundu s velmi krátkou dobou čekání. Jde o tzv. „in-memory“ databázi. To znamená, že databáze přistupuje k datům v paměti místo na disku. Vzhledem k tomu, že přístup k paměti je daleko rychlejší než přístup k mechanickému disku, a také z důvodu, že paměti RAM už nejsou až tak cenově nákladné, zdá se toto řešení být velmi účinné.

VoltDB databáze je vhodná pro aplikace, které potřebují:

- Mnohonásobně větší výkonnost
- SQL jako základní jazyk pro zjednodušení vývoje a komunikace
- Vlastnosti ACID pro zajištění konzistence dat
- Vestavěnou vysokou dostupnost a toleranci k chybám (nutný předpoklad pro stabilní databázový systém)
- Podporu pro replikaci dat mezi datovými centry z důvodu nutné obnovy databáze = tím je myšlena jak obnova dat, tak i databázové struktury.

3.2.4.1.1 Představení funkcionality VoltDB

Jak už bylo zmíněno výše, jedná se o databázi, která zpracovává a udržuje data v paměti. Tato databáze využívá nesdílených clusterů („share nothing cluster“). Nesdílený cluster je soběstačný a nesdílí s ostatními clustery žádný prostor v paměti či na disku a tím zabraňuje chybám, které by mohly zapříčinit totální výpadek celého systému. Proto VoltDB vytváří automatické replikace dat uvnitř a vně clusteru. Díky tomu dosahuje velké dostupnosti dat. Tato data jsou synchronně zpracována do replikovaných částí v clusteru před tím, než se dokončí celá transakce (vykoná příkaz „commit“). Metoda automatické replikace dat poskytuje větší odolnost vůči chybám v případě struktury jednoho uzlu. Kromě selhání jednoho uzlu zde může nastat i výskyt chyb celého clusteru. Aby se zabránilo nevratné chybě clusteru, transakce jsou asynchronně zpracovány do replikovaného clusteru, který se fyzicky musí nacházet v jiné geografické lokaci. VoltDB automaticky garantuje, že každý replikovaný cluster si pamatuje pořadí transakcí, ve kterém mají být zpracovány. Nastane-li chyba jednoho uzlu, VoltDB se automaticky obrátí na replikovaný uzel. Spustí se obnova uzlu tak, že obnovený uzel si sám spustí všechny nutné příkazy, které potřebuje pro obnovu dat. Výhoda této metody je taková, že si samotná aplikace nevšimne nastoleného problému a tím aplikace nedosáhne fatální chyby (je dosažena větší odolnost vůči výpadkům systému) (9).

Data a procesy s nimi spojené jsou rozděleny na části a distribuovány jednotlivým jádrům procesoru neboli virtuálním uzlům (tzv. „virtual nodes“ v jednom hardwarovým clusteru). Pro jednotlivé transakce pracuje každé VoltDB jádro samostatně, bez pomoci nebo zasahování ostatních jader. Využívá sekvenční frontu transakcí (každá transakce se zpracovává postupně), čímž ruší potřebu zamykání nebo blokování tabulek, jak tomu bylo dříve v relačních databázích. Pro zpracování vnitřních transakcí (nastává, pokud jedna transakce obsahuje více menších transakcí) vždy jedno jádro distribuuje a koordinuje pracovní činnost pro ostatní jádra, proto nemůže nastat, že by si více jader pracovalo na stejném oddílu dat. Data a procesy jsou zpracovány jedním vláknem a poté distribuovány mezi ostatní jádra procesoru v serverech tvořící jeden cluster (9).

Každá tabulka je zde buď rozdělena na části mezi virtuální uzly, nebo je mezi tyto uzly klonována. Rozdělení tabulky na části se používá pro rozsáhlé, často aktualizované nebo přistupované tabulky, zatímco klonování se používá na malé, často čtené tabulky (9).

Bez neustálého čekání na disk a s vyjmutím složitosti zamykání a odemykání dat nebo tabulek se může výkon databáze plně zaměřit na zpracování požadavků a tím urychlit jejich zpracování (9).

3.2.4.1.1.1 Replikace a rozdělování tabulek

Jednou z dalších výhod zrychlení databáze je využití rozdělení tabulek na menší části. Tabulky jsou rozděleny tak, že se hodnoty primárních klíčů vloží do „hash“ tabulky. Největší optimalizace je dosaženo, pokud jsou k rozdělení tabulky vybrány pouze klíče, ke kterým se přistupuje nejčastěji. Pokud jsou nalezeny menší tabulky, které slouží spíše pro čtení, je efektivnější takové tabulky replikovat do ostatních uzlů v clusteru, spíše než je rozdělovat. Tím se omezí mezioddílové spojovací („join“) operace. Replikace a rozdělení tabulek bude rozebráno na příkladu praktické části v kapitole č. 4.1.2 (9).

3.2.4.1.1.2 Obnova dat

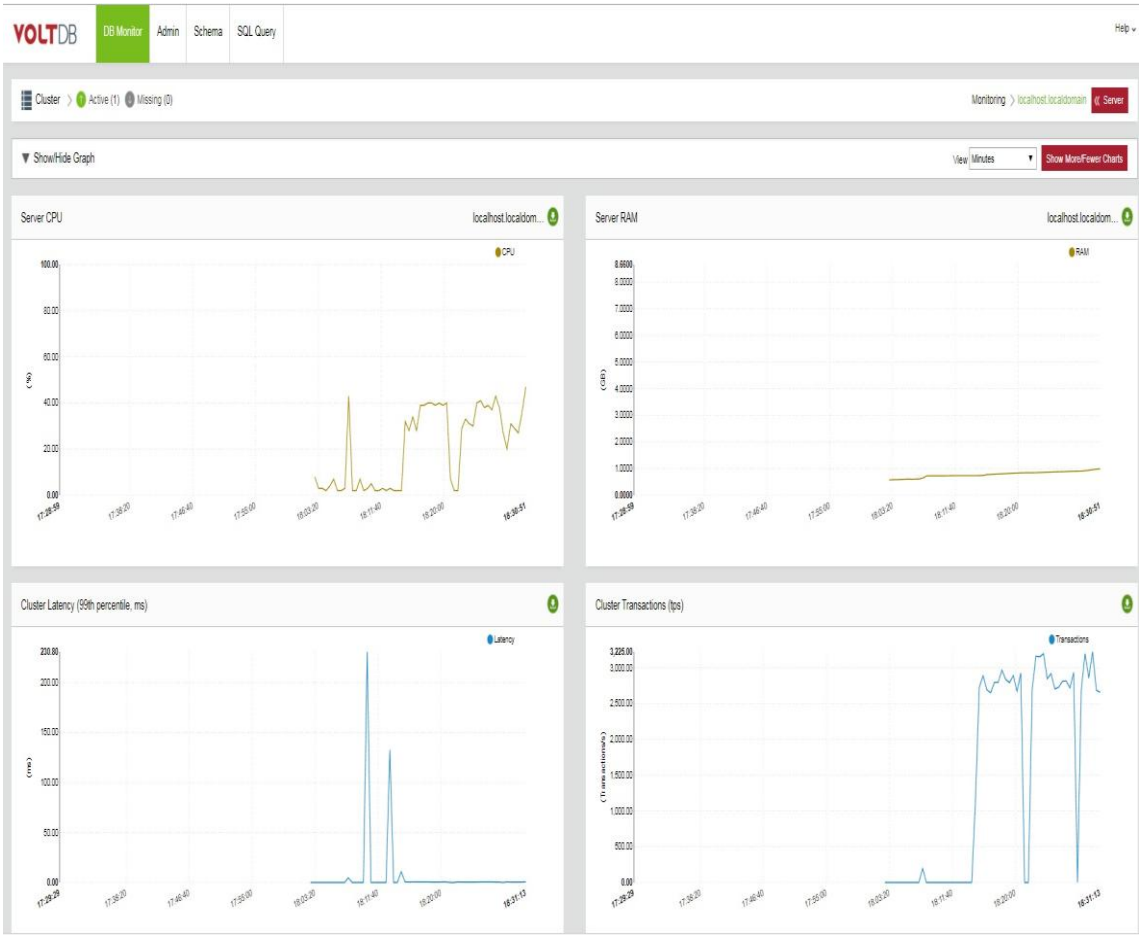
VoltDB obsahuje dva typy zálohování dat pro případ fatálního výpadku nebo vymazání dat z paměti počítače (serveru, hardwarové stroje, apod.). Při vypnutí počítače nebo ukončení práce s databází (ukončení programu) dojde k vymazání dat i databáze v RAM. Obnovu dat proto využívá databáze při každém novém spuštění počítače i při novém spuštění databáze. K vrácení databáze do funkčního stavu VoltDB využívá:

- **Snímky („Snapshots“)** – V definovaných intervalech si VoltDB zapisuje obrazy databáze na disk. Při vytvoření obrazů je zaručeno, že databáze je v konzistentním stavu. Z těchto obrazů je možná obnova do konzistentního stavu před výskytem fatální chyby systému. Obrazy představují minimální využití výkonu databáze a jejich vytvoření může probíhat v naplánovaných intervalech (například po každé provedené transakci).

- **Záznamy příkazů („Command logging“)** – Aplikace, která vyžaduje ještě lepší obnovu dat, než lze udělat ze snímků, může být podpořena záznamy příkazů. Jedná se o odolné úložiště každé procedury, která byla vykonána mezi jednotlivými snímky databáze. V případě fatálního výpadku může být databáze obnovena ze snímků a přehrání jednotlivých zaznamenaných příkazů. Administrátor si může vše podstatné nastavit (například synchronní nebo asynchronní vytváření záznamů nebo frekvenci zapisování těchto záznamů na disk) (9).

3.2.4.1.2 Rozhraní VoltDB

V této podkapitole bude rozebráno webové rozhraní VoltDB databáze.

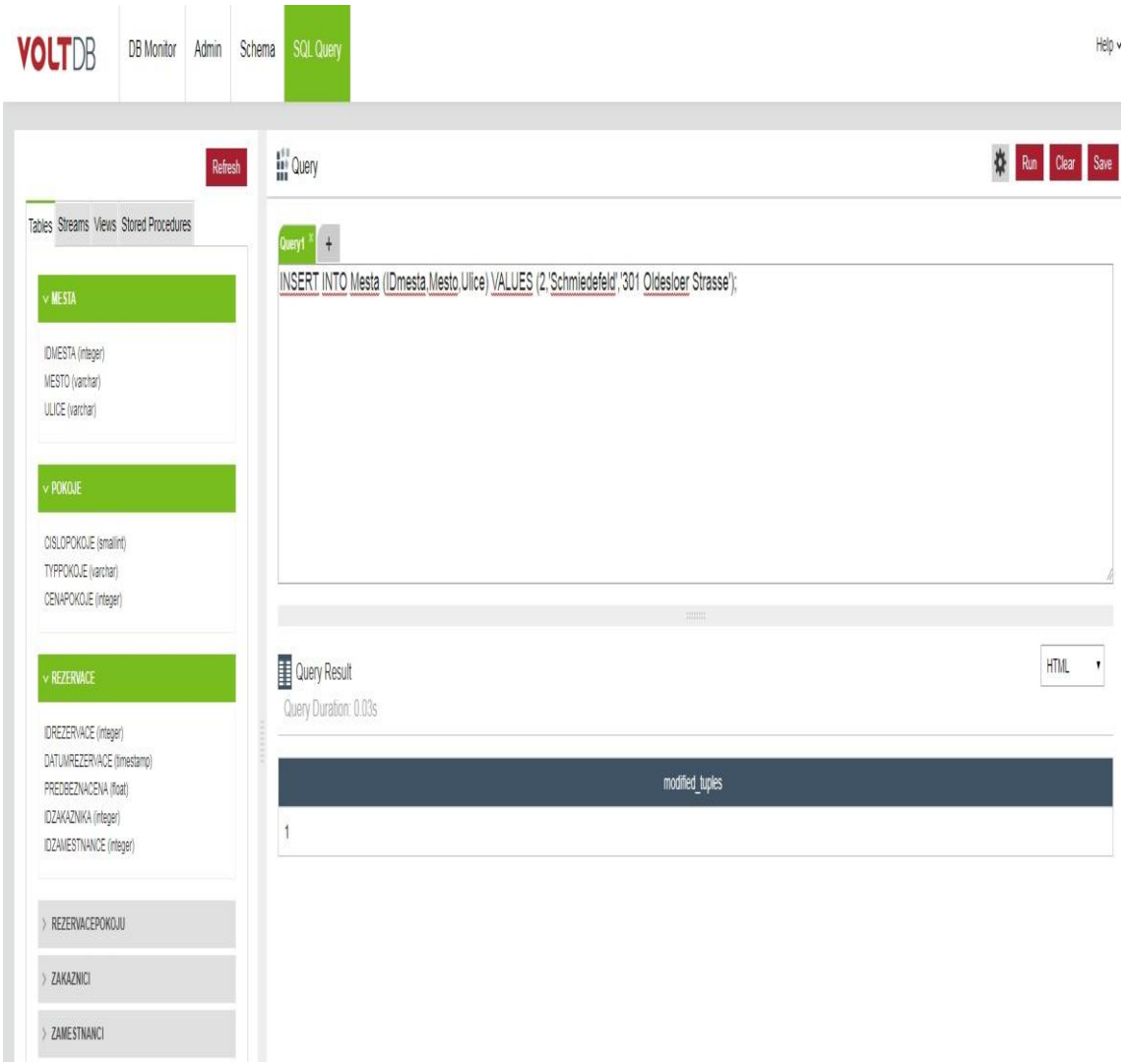


Obrázek 2 – VoltDB rozhraní pro sledování vytíženosti clusteru. Zdroj: Vlastní

Obrázek 3 – VoltDB rozhraní tabulek. Zdroj: Vlastní

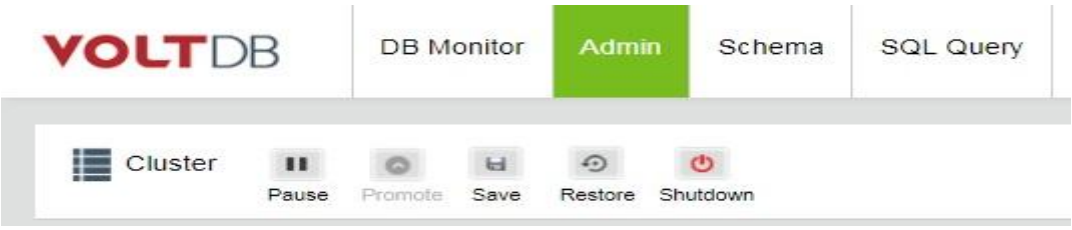
Na obrázku č. 2 lze vidět zatížení clusteru při zpracování SQL příkazů. V horní levé části snímku obrazovky se nachází graf vytíženosti procesoru a v pravé části vytížení paměti RAM. V dolní části obr. č. 2 je zobrazena odezva clusteru a počet zpracovaných transakcí za sekundu. Intervaly těchto údajů lze přepínat (například

dny, minuty, sekundy). Na obrázku č. 3 je přehled tabulek v databázi a jejich druh optimalizace (zda jsou rozdělené nebo replikované).

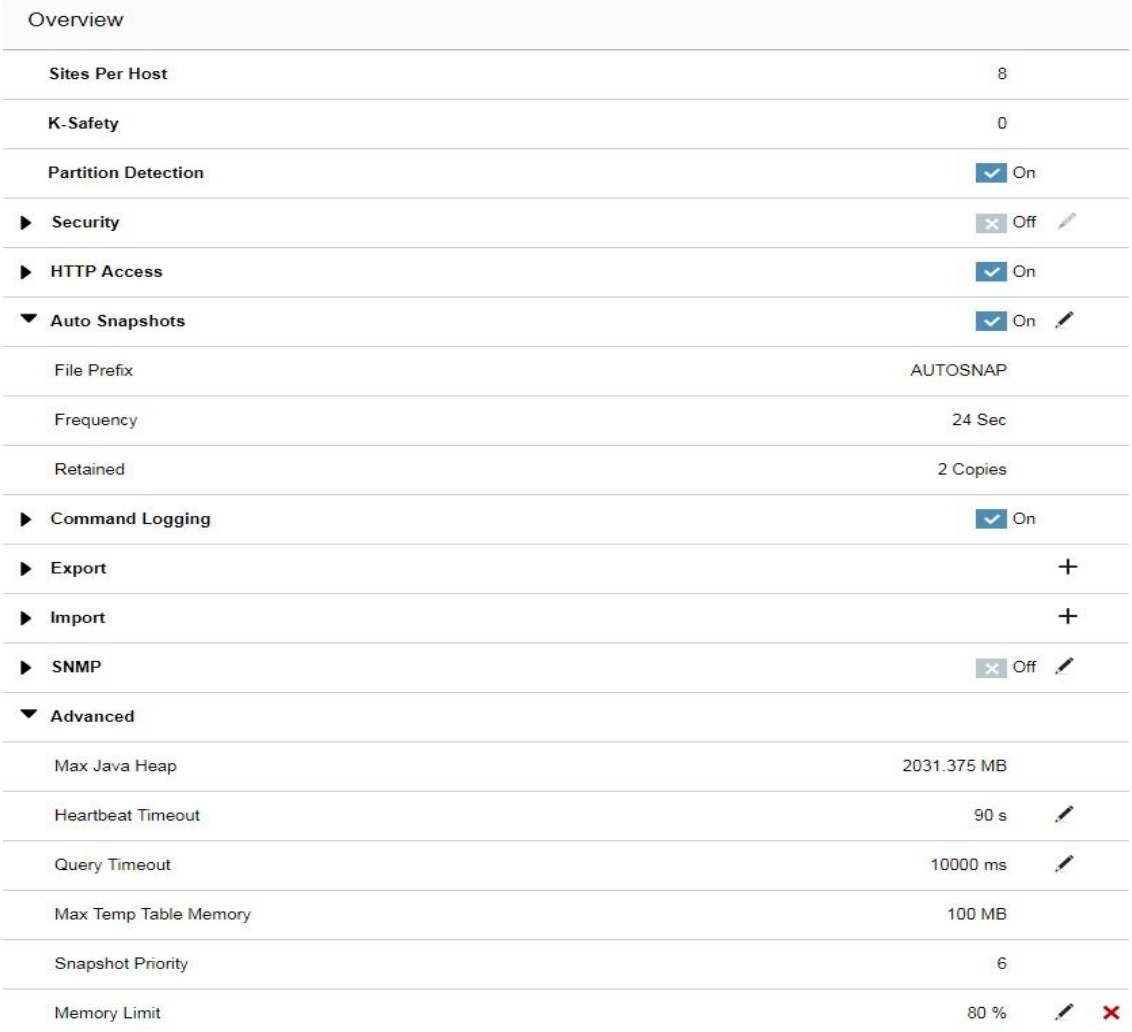


Obrázek 4 – VoltDB rozhraní pro zadávání SQL příkazů. Zdroj: Vlastní

Na obrázku č. 4 lze vidět hlavní výhodu oproti NoSQL databázím, kterou je zobrazení SQL komunikačního příkazového řádku. Pod příkazovým řádkem lze vidět úspěšnost zpracovaného „insert“ příkazu s celkovou dobou trvání zpracování. V levé části se pak nachází struktura tabulek v databázi, nebo vytvořené pohledy či vnořené procedury.



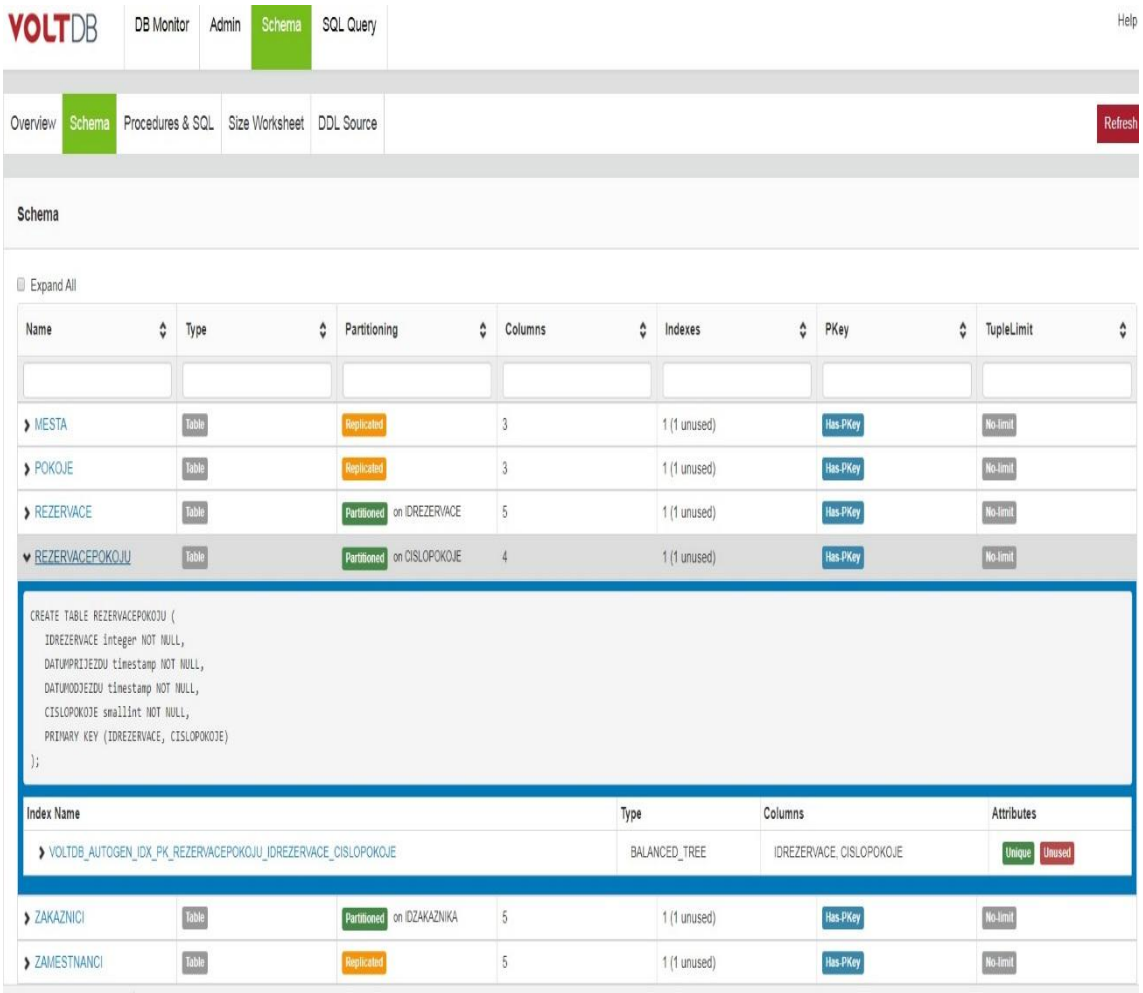
Obrázek 5 – VoltDB administrátorské rozhraní. Zdroj: Vlastní



Obrázek 6 – VoltDB rozhraní pro nastavení databáze. Zdroj: Vlastní

Karta „Admin“ zobrazená na obrázku č. 5 slouží pro základní nastavení funkčnosti VoltDB databáze. Na této záložce lze vidět jednoduché menu pro změnu stavu databáze. Takto je možné jednoduše vypnout nebo jenom pozastavit databázi. Jedna z možností je také manuálně vytvořit zálohovací snímek a případně i z něho obnovit konzistentní stav databáze bez nutnosti použití příkazového řádku. Druhá

část stejné karty je zobrazena na obrázku č. 6. Zde je patrné základní nastavení databáze. Za zmínku stojí určitě poukázat na nastavení automatických zálohovacích snímků nebo příkazových záznamů a jejich intervaly. Časové intervaly v nabídce jsou v sekundách, minutách nebo hodinách. Dále lze změnit časový limit pro dokončení příkazu nebo maximální využití paměti serveru.



Obrázek 7 – VoltDB rozhraní schématu. Zdroj: Vlastní

V kartě „Schema“ je několik dalších záložek. Nejdůležitější je však zobrazení struktury tabulek představené v obrázku č. 7. Schéma ukazuje typ tabulky, zda je určená pro replikaci nebo bude rozdělena na části. Dále v detailu tabulky je vidět SQL formát vytvořené tabulky a primární klíč s ní spojený.

Name	Type	Count	Row Min	Row Max	Index Min	Index Max	Table Min	Table Max
mesta	Table	100000	40	548	44	44	8,400 K	59,200 K
pokoje	Table	20000	34	118	42	42	1,520 K	3,200 K
rezervace	Table	1000000	28	28	44	44	72,000 K	72,000 K
rezervacepokoju	Table	1000000	22	22	46	46	68,000 K	68,000 K
zakaznici	Table	1000000	125	357	44	44	169,000 K	401,000 K
zamestnanci	Table	100000	105	357	44	44	14,900 K	40,100 K

Obrázek 8 – VoltDB rozhraní pro výpočet potřebné paměti. Zdroj: Vlastní

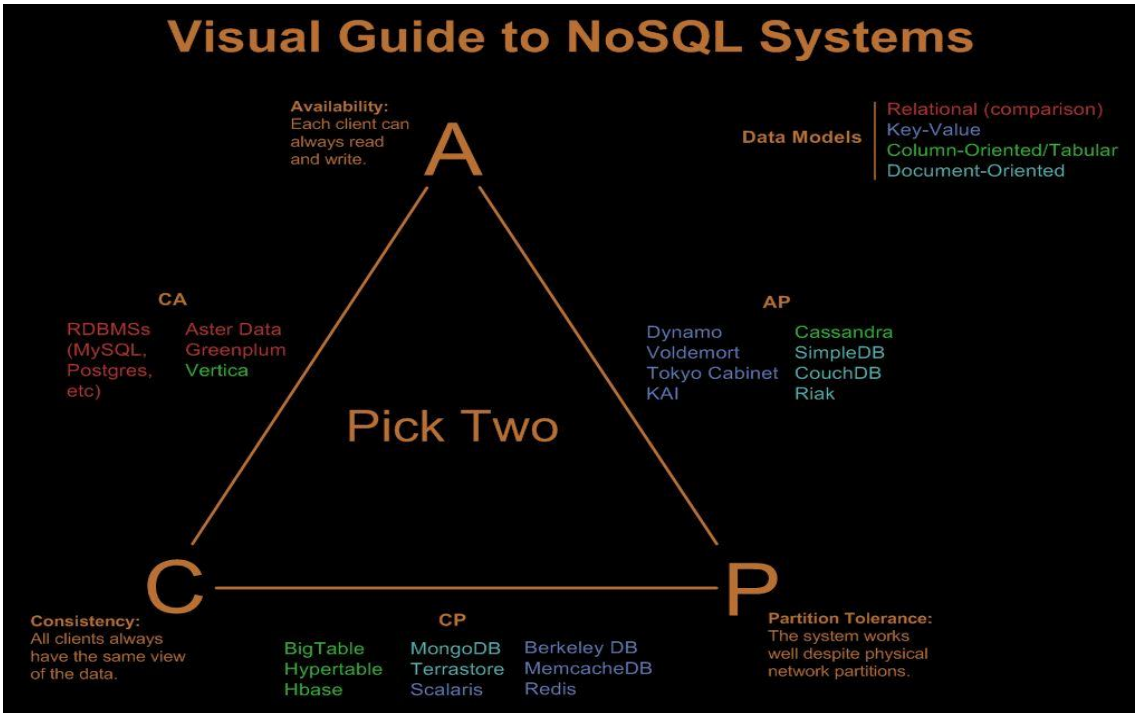
Na obrázku č. 8 je představena tabulka, která je částí karty schématu. Tato tabulka umožňuje uživateli zadat předpokládaný počet záznamů v tabulce a VoltDB mu automaticky vypočítá, kolik by si měl vymezit paměti. Velikost se udává v bytech.

3.3 CAP teorém

Tato zkratka vypovídá o třech vlastnostech databáze, které jsou zapotřebí:

- **Consistency (konzistence)** – Všichni klienti mají vždy stejný pohled na aktuální data. Tato definice má odlišný význam a dalo by se napsat, že je to podmnožina konzistence zmíněné výše v kapitole ACID.
- **Availability (dostupnost)** – Každý klient může vždy z databáze číst a zapisovat do ní.
- **Partition toleration (částečná tolerance)** – Systém funguje, přestože přišla doplňková zpráva o ztrátě nebo výpadku části systému (například zpoždění dat v síti).

Teorém, který říká, že v jakýchkoliv distribuovaných systémech mohou být vybrány pouze dvě z těchto vlastností v daný čas. Relační databáze zajišťují pouze konzistenci a dostupnost. U NoSQL je většinou možné nastavit výběrem implementace, které dvě vlastnosti bude databáze využívat (3).
Moderní definice je taková, že vzhledem k velké fyzické vzdálenosti mezi servery se může stát, že dojde ke zpoždění nebo chybě v síťové komunikaci. V takovém případě je systém zaměřen buď na zachování konzistence kombinací vlastností CP (Consistency – Partition toleration) nebo na podporu dostupnosti využitím AP (Availability – Partition toleration) (7). Na obrázku níže si lze prohlédnout databáze a jejich podporu CAP vlastností.



Obrázek 9 - Vizualizace CAP teorému, Zdroj: (10)

3.4 Alternativy řešení nedostatku výkonu relačních databází

Pokud již používaná relační databáze výkonnostně nestačí, existuje několik alternativ, které uživatel může zvolit pro rozšíření výkonu:

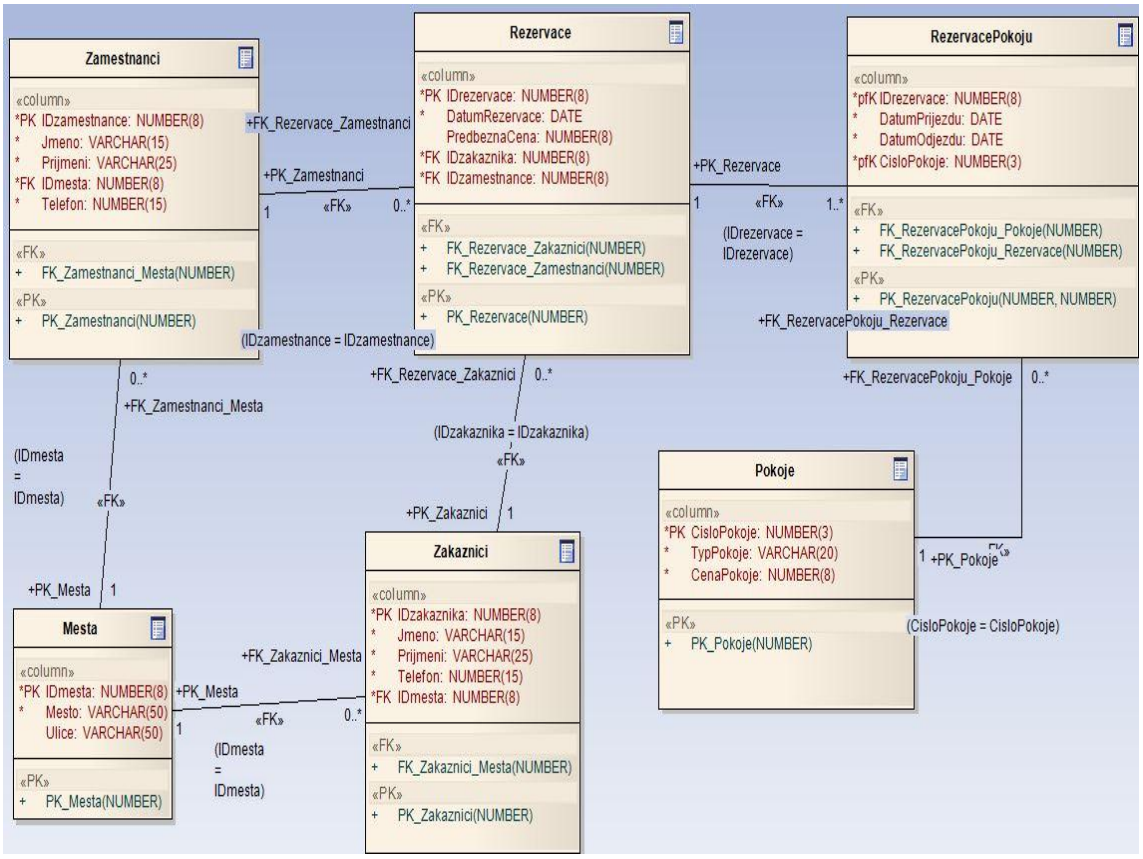
- Uživatel může nakoupit drahé vylepšení hardwaru (např. rychlejší disky, výkonnější procesor, nebo dokonce celý server). To by bylo velmi nákladné řešení.
- Zjednodušení databáze rozdělením na menší části, které poběží na jiných serverech. S tím je spojena nutnost úpravy aplikace pro komunikaci se všemi zařízeními (nakoupení nového zařízení, zaplacení úpravy aplikace…). To je však rovněž velmi složité a nákladné řešení.
- Přechod na NoSQL databázi, která odepře uživateli transakční vlastnosti ACID a SQL komunikaci. To by znamenalo ztrátu záruky konzistence a bylo by těžké najít správce takové databáze, vzhledem k malé rozšířenosti.
- Přechod z volně stažitelné databáze na komerční placený produkt. To by bylo opět velmi nákladné řešení, ale se zárukou stability a zkušené uživatelské podpory.
- Přechod na NewSQL databázi, která může zachovat ACID vlastnosti a SQL komunikaci z předešlého modelu. Tím by byl zaručen velký nárůst výkonu, ale opět bez rozšířené podpory z důvodu novoty tohoto modelu. Další nevýhodou je, že pokud bude mít uživatel databázi o velikosti petabytů a vybral by si výše zmíněnou VoltDB databázi, je zde problém (vzhledem k tomu že to je databáze pracující v paměti) nedostatku místa. Proto VoltDB nabízí integrovaný podsystém pro analýzu dat, který zvládá přesunout data do NoSQL databáze Hadoop, kde lze analýzu provádět.

Je tedy na každém uživateli, pro jakou volbu se bude chtít rozhodnout. A protože popularita NewSQL a NoSQL databází stále roste, nedostatek podpory již v budoucnu nemusí být problém.

4 Praktická část

4.1 Vytvoření databázové struktury v programu VoltDB

VoltDB databáze má oproti nerelačním databázím komunikační výhodu. Uživatel se může dotazovat pomocí jazyka SQL. Jelikož SQL není přímým předmětem této práce, bude zde detailně rozebrána tvorba pouze jedné tabulky s hlavními rozdíly oproti klasické relační databázi.



Obrázek 10 – Návrh rezervačního systému hotelu pro praktickou ukázku. Zdroj: Vlastní

Pro tuto praktickou ukázku bylo navrženo schéma z obrázku č. 7. Jedná se o návrh rezervačního systému pro hotelový řetězec.

4.1.1 VoltDB omezení

Klasicky jako pro každou jinou databázi je zde list unikátních podporovaných datových typů, které lze nalézt v tutoriálech na internetových stránkách VoltDB.

Většina omezení je vyřešena uloženými procedurami. Mezi důležitější limitace patří:

- Kontrolní omezení („check constraint“) a cizí klíče („foreign keys“) zatím nejsou podporovány
- Automatická inkrementace sloupcových hodnot není podporována (například pro automatické zvyšování identifikačního čísla)
- Každý sloupec má maximální velikost 1 MB a každá tabulka se všemi sloupci dohromady musí mít méně než 2 MB
- Při použití sloupce k rozdělení tabulky musí mít daný sloupec vždy nenulové hodnoty (musí být specifikován jako „not null“) (9).

4.1.2 VoltDB optimalizace

Schéma databáze není jediná věc, kde lze ušetřit nebo rozdělit výkon. Důležitým předpokladem kvalitního architekta databáze je schopnost předvídat vytížení na určitých tabulkách v jednotlivých sloupcích. Z obrázku č. 10, Návrhu pro rezervaci pokojů v hotelovém řetězci lze říci, že nejčastější operací bude vyhledání obsazenosti pokojů v daném termínu. Hned za tím následuje operace vytvoření rezervace a jako třetí bude operace zrušení rezervace. Vytvoření zákazníka nebude na stejné úrovni jako vytvoření rezervace z důvodu provázanosti tabulky rezervace s ostatními.

4.1.2.1 Rozdělení tabulek

Teď když bylo rozebráno, které operace budou mít největší zátěž, může být provedeno rozdělení tabulky. K tomu je nutné vybrat sloupec z tabulky, který může VoltDB unikátně identifikovat a distribuovat jeho řádky do různých částí. K vyhledání volných pokojů je přistupováno přes tabulku rezervaci pokojů s využitím sloupce *„CisloPokoje“*, proto se to zdá tento parametr být jasným kandidátem pro rozdělení tabulky. SQL příkaz by pak vypadal: ***„PARTITION TABLE RezervacePokoju ON COLUMN CisloPokoje;“***.

Víme z předchozí kapitoly, že u rozdělení tabulky sloupcem musí být hodnoty v daném sloupci unikátní. Když by byla rozdělena tabulka zákazníků, hodnoty sloupců jména a příjmení nemusí být unikátní. Z toho důvodu je lepší zvolit

„IDzakaznika“ jako rozdělovací sloupec. Příkaz v tomto případě by vypadal: ***„PARTITION TABLE Zakaznici ON COLUMN IDzakaznika“***.

Pro rozdělování tabulek je nutno respektovat několik pravidel:

- V každé tabulce může být pouze jeden rozdělovací sloupec.
- Pokud jsou dvě rozdělené tabulky a uživatel nad nimi bude chtít provést operaci spojení tabulek („join“), pak obě tyto tabulky musí být rozděleny nad stejným sloupcem.
- Pokud rozdělovaná tabulka obsahuje primární klíč, pak rozdělovací sloupec musí být zahrnut do daného primárního klíče.
- Jakýkoliv sloupec obsahující integer nebo string hodnoty (v databázi často označován jako varchar) může být vybrán pro rozdělení tabulky.
- Hodnoty rozdělovacího sloupce nesmí být „null“.
- Tabulka, která již obsahuje data a je rozdělena sloupcem, nemůže být předělána na replikovanou nebo nelze učinit změnu sloupce, kterým by měla být daná tabulku rozdělena (9).

4.1.2.2 Replikace tabulek

Pokud se ve schématu nachází tabulka, která není vhodná pro rozdělení, může být replikována mezi ostatní uzly v clusteru. V daném schématu je to například tabulka s přehledem pokojů. Tato tabulka se dá označit za malou s odhadovaným maximálním výskytem tisíce záznamů. Dále lze říci, že tato tabulka bude sloužit hlavně pro čtení. Je to ideální kandidát pro replikaci do ostatních uzlů v clusteru. Dalším příkladem by byla tabulka zaměstnanců, kde se neočekává časté zapisování (např. změny zaměstnanců) ani přílišná velikost tabulky.

Ve VoltDB není explicitní SQL příkaz pro replikaci tabulek. Pokud není řečeno, že daná tabulka má být rozdělena na části, pak tato tabulka bude automaticky replikována díky interní logice.

4.2 Dotazování pomocí jazyka SQL

V této kapitole bude provedeno několik SQL dotazů na zjištění rychlosti zpracování VoltDB databáze a MySQL databáze. VoltDB databáze byla vytvořena v minulé kapitole. Pro tyto účely byla vytvořená i relační databáze MySQL se stejným schématem. Jediný rozdíl je, že v MySQL databázi byly vytvořeny referenční integrity s indexy na primárních a cizích klíčích oproti rozdělení a replikací tabulek u VoltDB.

SQL příkaz číslo 1 vrátí počet rezervací, které byly vytvořeny jednotlivými zaměstnanci:

```
SELECT COUNT(IDrezervace), z.Jmeno, z.Prijmeni
FROM rezervace as r, zamestnanci as z
WHERE r.IDzamestnance = z.IDzamestnance
GROUP BY z.jmeno, z.prijmeni
ORDER BY COUNT(IDrezervace) DESC;
```

SQL příkaz číslo 2 vybere zákazníky, kteří pochází z Berlína:

```
SELECT z.Jmeno, z.Prijmeni
FROM mesta as m, zakaznici as z
WHERE z.IDmesta = m.IDmesta AND m.Mesto LIKE '%Berlin%'
GROUP BY z.jmeno, z.prijmeni;
```

SQL příkaz číslo 3 analyzuje oblíbenost pokojů v počtu rezervací:

```
SELECT COUNT(IDrezervace), r.CisloPokoje
FROM RezervacePokoju as r, Pokoje as p
WHERE r.CisloPokoje = p.CisloPokoje
GROUP BY r.CisloPokoje
ORDER BY COUNT(IDrezervace) DESC;
```

SQL příkaz číslo 4 vyčíslí celkovou předběžnou sumu, zpracovanou jednotlivými zaměstnanci:

SELECT SUM(PredbeznaCena) cena, z.jmeno, z.prijmeni
FROM rezervace as r, zamestnanci as z
WHERE r.IDzamestnance = z.IDzamestnance
GROUP BY z.jmeno, z.prijmeni
ORDER BY cena DESC;

Poslední příkaz SQL číslo 5 zobrazí všechny rezervace vytvořené v roce 2016:
*SELECT **
FROM rezervace
Where YEAR(DatumRezervace) = 2016;

4.2.1 Výsledky a porovnání s relační databází MySQL

Všechny SQL příkazy byly spuštěny na obou databázích přes příkazový řádek tak, aby nevznikla žádná odchylka způsobená rozhraním databáze. Obě databáze fungovaly individuálně na stejném hardwarovém stroji. Ani jedna databáze nebyla testována v plném rozsahu. Například MySQL databáze neměla k dispozici diskové pole a VoltDB neměla více uzlů ke spolupráci.

Databáze/SQL	SQL1	SQL2	SQL3	SQL4	SQL5
VoltDB	1,34s	0,86s	0,42s	1,35s	0,1s
MySQL	2,45s	8,00s	0,87s	7,32s	0,53s
SQL příkaz/Databáze	VolTDB	MySQL			
SQL1	9546 řádků	9546 řádků			
SQL2	3557 řádků	3557 řádků			
SQL3	20 000 řádků	20 000 řádků			
SQL4	9546 řádků	9546 řádků			
SQL5	131 199 řádků	131 199 řádků			

Tabulka 2 – Výsledky SQL dotazů na VoltDB a MySQL databázi

Z tabulky číslo 2 lze přehledně vidět porovnání časů mezi databázemi MySQL a VoltDB. Za zmínku určitě stojí také čas vložení generovaných dat pro celou databázi. U VoltDB byl čas vložení SQL dat v rozmezí minut, zatímco u MySQL to bylo v hodinách. Dále maximálních transakcí za sekundu bylo dosaženo ve VoltDB 1300 a v MySQL 400. To už se určitě dá považovat za znatelný rozdíl výkonu. Databáze VoltDB je tedy pro zvolený příklad rezervace pokojů v systému hotelového řetězce výhodnější než databáze SQL.

5 Shrnutí a závěr

Z literárních údajů i praktických zkušeností s různými typy databázových systémů je zřejmé, že relační databáze jsou stále pro řadu případů dostačující volbou, budeme se s nimi jistě setkávat v různých aplikacích po řadu následujících let. Přesto je však řada případů, ve kterých je využívání nerelační databáze vhodnější volbou. Především se jedná o analýzy ohromného množství dat, nebo například webové aplikace, kde se nejčastěji nachází nestrukturovaná data, která se mohou často měnit. V budoucnu můžeme předpokládat vývojový souboj mezi NewSQL a NoSQL především v aplikacích, kde bude docházet k časté změně dat.

Jednotlivé databázové systémy využívají různých vlastností CAP teorému. Z tohoto důvodu je úlohou architekta databáze zjistit a rozvrhnout si, kterou z těchto tří vlastností lze oželet. Před tím, než se bude vybírat správný typ databáze, je zapotřebí si uvědomit, která data se budou ukládat v datovém úložišti a také které požadavky budou nezbytně zapotřebí, ať už je potřebná konzistence dat poskytována transakčními vlastnostmi ACID, nebo jestli budou stačit BASE. To všechno jsou základní kritéria pro výběr nejefektivnější databáze pro konkrétní aplikaci.

V textu práce byl zmíněn nejnovější typ databází NewSQL jako nové struktury relačních databází zachovávající vlastnosti ACID při doplnění nedostatků oproti nerelačním databázím. Do architektury NewSQL patří například VoltDB, jehož funkcionality byly detailně popsány. VoltDB je zajisté dobrá volba, jedná-li se o data, která jsou často aktualizovaná. Z důvodu rychlosti zpracování a uložení dat v paměti je výkonnost této databáze bezkonkurenční. Bohužel komunitní verze neumožňuje využití zajímavějších funkcí, jako například vytváření snímků databáze nebo záznamů příkazů. Byly ukázány mnohé vlastnosti této nové databáze. Hlavní výhodou takové databáze je jednoduchá tvorba databáze i snadná optimalizace velmi zvětšující účinnost zpracování transakcí.

Cílem bakalářské práce bylo představit hlavní aspekty novějších typů databáze a porovnat jejich rozdíly s relační databází. Snahou bylo především poukázat na přednosti těchto databází, které by vedly k dalšímu prohloubení a zvýšení obecného zájmu o nové metody architektur databáze. Výsledky provedených

měření prezentované v praktické části bakalářské práce ukazují rozdílné časy zpracování obou testovacích databází. Je možno konstatovat, že NewSQL databáze již v dnešní době jsou pro řadu případů vhodnou náhradou za zastaralé relační databáze.

6 Použitá literatura

1. **Mareček, Jakub.** *NoSQL databáze pro ukládání owl dat.* Brno : Fakulta informatiky, Masarykova univerzita, 2013. Diplomová práce.

2. **Ularu, Elena Geanina, a další.** Perspectives on Big Data and Big Data analytics. *Database systems journal vol. III.* [Online] Duben 2012. [Citace: 27. Srpen 2016.] http://dbjournal.ro/archive/10/10_1.pdf.

3. **Puget, Jean Francois.** Big Data for dummies. [Online] 22. Duben 2013. [Citace: 27. Srpen 2016.] https://www.ibm.com/developerworks/community/blogs/jfp/entry/big_data_for_dummies23?lang=en.

4. **Mayer-Schönberger, Viktor a Cukier, Kenneth.** *Big Data.* Brno : Computer press, 2014. 978-80-251-4119-9.

5. **Grolinger, Katarina, a další.** Data management in cloud environments: NoSQL and NewSQL data stores. 2013, Sv. Journal of cloud computing: advances, systems and applications, stránky 1-24.

6. **Sadalage, Paramod.** NoSQL databases: an overview. [Online] 2. Říjen 2014. [Citace: 3. Listopad 2016.] https://www.thoughtworks.com/insights/blog/nosql-databases-overview.

7. **Musil, Jan.** *Relační nebo NoSQL databáze? Nebo obojí?*: IBM Corporation, 2016. IBM Analytics.

8. **Gurevich, Yuri.** *Comparative survey of NoSQL/NewSQL DB systems.* místo neznámé, Israel : Department of mathematics and computer science, the open university of israel, Prosinec 2015. Final Paper.

9. **VoltDB, Inc.** Using VoltDB. [Online] 2008-2017. [Citace: 10. 8 2017.] https://docs.voltdb.com/UsingVoltDB/index.php.

10. **Hurst, Nathan.** Nathan Hurst's blog. *Thoughts on software, technology, and startups.* [Online] 15. Březen 2010. [Citace: 16. Květen 2017.] http://blog.nahurst.com/visual-guide-to-nosql-systems.

11. **VoltDB, Inc.** VoltDB technical overview. [Online] 2017. https://www.voltdb.com/wp-content/uploads/2017/04/VoltDB_Technical_Overview.pdf.

7 Příloha

První odstavec SQL vytváří strukturu pro MySQL databázi.

```
CREATE TABLE Mesta (
IDmesta INTEGER UNIQUE NOT NULL,
Mesto VARCHAR(50) NOT NULL,
Ulice VARCHAR(50),
PRIMARY KEY(IDmesta)
);
CREATE INDEX pk_IDmesta ON Mesta (IDmesta);

CREATE TABLE Zamestnanci (
IDzamestnance INTEGER UNIQUE NOT NULL,
Jmeno VARCHAR(15) NOT NULL,
Prijmeni VARCHAR(25) NOT NULL,
IDmesta INTEGER NOT NULL,
Telefon VARCHAR(25),
PRIMARY KEY(IDzamestnance),
FOREIGN KEY(IDmesta) REFERENCES Mesta(IDmesta)
);
CREATE INDEX pk_IDzamestnance ON Zamestnanci (IDzamestnance);
CREATE INDEX fk_IDmestazamestnancu ON Zamestnanci (IDmesta);
CREATE TABLE Zakaznici (
IDzakaznika INTEGER UNIQUE NOT NULL,
Jmeno VARCHAR(15) NOT NULL,
Prijmeni VARCHAR(25) NOT NULL,
Telefon VARCHAR(25) NOT NULL,
IDmesta INTEGER NOT NULL,
PRIMARY KEY(IDzakaznika),
FOREIGN KEY(IDmesta) REFERENCES Mesta(IDmesta)
);
CREATE INDEX pk_IDzakaznika ON Zakaznici (IDzakaznika);
```

```
CREATE INDEX fk_IDmestazakazniku ON Zakaznici (IDmesta);
CREATE TABLE Pokoje (
CisloPokoje SMALLINT UNIQUE NOT NULL,
TypPokoje Varchar(20) NOT NULL,
CenaPokoje INTEGER NOT NULL,
PRIMARY KEY(CisloPokoje)
);
CREATE INDEX pk_CisloPokoje ON Pokoje (CisloPokoje);
CREATE TABLE Rezervace (
IDrezervace INTEGER NOT NULL,
DatumRezervace TIMESTAMP NOT NULL,
PredbeznaCena FLOAT NOT NULL,
IDzakaznika INTEGER NOT NULL,
IDzamestnance INTEGER NOT NULL,
PRIMARY KEY(IDrezervace),
FOREIGN KEY(IDzakaznika) REFERENCES Zakaznici(IDzakaznika),
FOREIGN KEY(IDzamestnance) REFERENCES Zamestnanci(IDzamestnance)
);
CREATE INDEX pk_IDrezervace ON Rezervace (IDrezervace);
CREATE INDEX fk_IDzakaznika ON Rezervace (IDzakaznika);
CREATE INDEX fk_IDzamestnance ON Rezervace (IDzamestnance);

CREATE TABLE RezervacePokoju (
IDrezervace INTEGER NOT NULL,
DatumPrijezdu TIMESTAMP NOT NULL,
DatumOdjezdu TIMESTAMP NOT NULL,
CisloPokoje SMALLINT NOT NULL,
PRIMARY KEY(IDrezervace, CisloPokoje),
FOREIGN KEY(IDrezervace) REFERENCES Rezervace(IDrezervace),
FOREIGN KEY(CisloPokoje) REFERENCES Pokoje(CisloPokoje)
);
```

```
CREATE INDEX pk_RezervacePokoje ON RezervacePokoju (IDrezervace,CisloPokoje);
CREATE INDEX fk_IDrezervace ON RezervacePokoju (IDrezervace);
CREATE INDEX fk_CisloPokoje ON RezervacePokoju (CisloPokoje);
```

Druhá část SQL kodu vytváří stejnou strukturu pro VoltDB databázi

```
CREATE TABLE Mesta (
IDmesta INTEGER UNIQUE NOT NULL,
Mesto VARCHAR(50) NOT NULL,
Ulice VARCHAR(50),
PRIMARY KEY(IDmesta)
);
CREATE TABLE Zamestnanci (
IDzamestnance INTEGER UNIQUE NOT NULL,
Jmeno VARCHAR(15) NOT NULL,
Prijmeni VARCHAR(25) NOT NULL,
IDmesta INTEGER NOT NULL,
Telefon VARCHAR(25),
PRIMARY KEY(IDzamestnance)
);
CREATE TABLE Zakaznici (
IDzakaznika INTEGER UNIQUE NOT NULL,
Jmeno VARCHAR(15) NOT NULL,
Prijmeni VARCHAR(25) NOT NULL,
Telefon VARCHAR(25) NOT NULL,
IDmesta INTEGER NOT NULL,
PRIMARY KEY(IDzakaznika)
);
CREATE TABLE Pokoje (
CisloPokoje SMALLINT UNIQUE NOT NULL,
```

```
TypPokoje Varchar(20) NOT NULL,
CenaPokoje INTEGER NOT NULL,
PRIMARY KEY(CisloPokoje)
);
CREATE TABLE RezervacePokoju (
IDrezervace INTEGER NOT NULL,
DatumPrijezdu TIMESTAMP NOT NULL,
DatumOdjezdu TIMESTAMP NOT NULL,
CisloPokoje SMALLINT NOT NULL,
PRIMARY KEY(IDrezervace, CisloPokoje)
);
CREATE TABLE Rezervace (
IDrezervace INTEGER NOT NULL,
DatumRezervace TIMESTAMP NOT NULL,
PredbeznaCena FLOAT NOT NULL,
IDzakaznika INTEGER NOT NULL,
IDzamestnance INTEGER NOT NULL,
PRIMARY KEY(IDrezervace)
);
PARTITION TABLE RezervacePokoju ON COLUMN CisloPokoje;
PARTITION TABLE Zakaznici ON COLUMN IDzakaznika;
PARTITION TABLE Rezervace ON COLUMN IDrezervace;
```

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Chaloupka Martin	Milady Horákové 1073/15, Hradec Králové - Nový Hradec Králové	I1301318

TÉMA ČESKY:

Big data a jejich využití

TÉMA ANGLICKY:

Big data and its usage

VEDOUCÍ PRÁCE:

Ing. Barbora Tesařová, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je představit NoSQL a NewSQL jako nový přístup k managementu dat.

1) Úvod tématu
2) Relační databáze
3) Big data představení
4) NoSQL databáze
5) NewSQL
6) CAP teorém
7) Představení funkcionality VoltDB
8) Vytvoření databázové struktury v programu VoltDB
9) Dotazování pomocí jazyka SQL
10) Shrnutí výsledků a závěr

SEZNAM DOPORUČENÉ LITERATURY:

Grolinger, Katarina, a další. Data management in cloud environments: NoSQL and NewSQL data stores. 2013, Sv. Journal of cloud computing: advances, systems and applications, stránky 1-24.

Sadalage, Paramod. NoSQL databases: an overview. [Online] 2. Říjen 2014. [Citace: 3. Listopad 2016.] https://www.thoughtworks.com/insights/blog/nosql-databases-overview.

Musil, Jan. Relační nebo NoSQL databáze? Nebo obojí? IBM Corporation, 2016. IBM Analytics.

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....