Univerzita Karlova
Pedagogická fakulta

Katedra informačních technologií a technické výchovy

BAKALÁŘSKÁ PRÁCE

Porovnání SQL a noSQL přístupu při vývoji aplikace

Comparison of SQL and noSQL application development

Filip Hanel

Vedoucí práce: PhDr. Josef Procházka, Ph.D.

Studijní program: Specializace v pedagogice

Studijní obor: Informační technologie se zaměřením na vzdělávání

2020

Prohlašuji, že jsem bakalářskou práci na téma Porovnání SQL a noSQL přístupu při vývoji aplikace vypracoval pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Dále prohlašuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

V Praze dne 23. 4. 2020

.....

Podpis

Rád bych poděkoval vedoucímu mé bakalářské práce, PhDr. Josefu Procházkovi, Ph.D., za pomoc, rady a trpělivost, neboť vznikala za stížených podmínek, způsobených pandemií COVID-19.

ANOTACE

Bakalářská práce přináší porovnání SQL a noSQL databázových přístupů používaných při vývoji aplikace. Cílem porovnání je zjištění, kdy a za jakých podmínek vybrat vhodnou databázi. Porovnání nejprve proběhne na teoretické úrovni vycházející z teoretických aspektů daných systémů a poté na praktické úrovni v podobě konkrétní webové aplikace. Práce mimo jiné mapuje historii zpracování dat a vznik databází se zaměřením na SQL a noSQL databáze, které charakterizuje a představí jejich konkrétní zástupce. Práce sleduje aktuální databázové a vývojové trendy vztahující se ke zmíněným databázím.

KLÍČOVÁ SLOVA

databáze, databázové trendy, historie databází, návrh databáze, MariaDB, MongoDB, noSQL, SQL, webová aplikace

ANNOTATION

This bachelor thesis compares SQL and noSQL database application used in development. The aim of the comparison is to find out when and under what conditions to choose a suitable database. The comparison will first take place at the theoretical level based on the theoretical aspects of the systems and then at the practical level in the form of a specific web application. The work also maps the history of data processing and the emergence of databases with a focus on SQL and noSQL databases that characterize and introduce their specific representatives. The thesis follows current database and development trends related to the mentioned databases.

KEYWORDS

database, database trends, history of databases, database design, MariaDB, MongoDB, noSQL, SQL, web application

Obsah

1 Úvod

V současné době se na trhu objevuje velké množství databází. Mezi nimi se nacházejí dlouhodobě zaběhnuté a stále populární databáze založené na principu SQL, ale vzniká zde i prostor pro nové databáze založené na principu noSQL, které nabývají popularity a nabízejí různé kompromisy. Vybrat vhodnou databázi je často mnohem náročnější, než se zdá. Tato práce přináší souhrn těchto dvou databázových přístupů v závislosti na současných databázových trendech. Databázovými přístupy se rozumí kategorie databází, kterými jsou SQL a noSQL databáze. Práce zkoumá jejich rozdíly, využití a možnosti.

Práce je rozdělena do osmi kapitol obsahující teoretickou a praktickou část. V teoretické části jsou vysvětleny základní termíny používané v databázích, které se v práci často vyskytují. Následně je věnovaná pozornost historii zpracovávání dat až ke vzniku samotných databází. Historie je rozdělena do časových období od 40. let 20. století až do počátku 21. století. Větší pozornost je věnována 70. letům, ve kterých se poprvé objevují relační databáze neboli SQL databáze. Následuje počátek 21. století se vznikem noSQL databází a jejich charakteristika. Tato kapitola je zakončena krátkým zamyšlením nad budoucností daných databázových přístupů. Poté následují jednotlivé kapitoly věnované SQL a noSQL s konkrétními databázovými distribucemi postavenými na zmíněných databázových přístupech. Dále je věnovaná pozornost různým konceptům daných systémů. Závěr teoretické části je zakončen kapitolou Shrnutí teorie, ve které jsou SQL a noSQL databáze společně porovnány na teoretické úrovni.

Po teoretické části přichází praktická část, ve které dojde k aplikování SQL a noSQL databázového přístupu na konkrétní webové aplikaci. Byla vytvořena aplikace pro chov agam vousatých. Velká část je věnovaná samotné implementaci a návrhu daných databází s ukázkami a s následným porovnáním obou variant, včetně subjektivního názoru. Kromě samotného porovnání databází je tato část věnovaná i návrhu samotné aplikace.

2 K čemu slouží databáze

V této kapitole bude vysvětlena samotná definice databáze, včetně konkrétního využití databází a seznámení se se základními termíny, jako je perzistence, redundance, konzistence dat, security a integrita databáze. Následně bude rozebrán databázový a souborový přístup k databázím, systém řízení báze dat, databázová aplikace, databázový systém a samotná báze dat. V poslední části bude věnována pozornost skupinám uživatelů databáze.

2.1 Definice databáze

Databáze lze definovat jako souhrn navzájem vázaných dat, uložených bez zbytečné redundance, aby mohla sloužit více databázovým aplikacím. Data jsou uložena v paměti tak, že jsou nezávislá na programech, které ji používají. Přidávání nových dat, modifikace a výběr již existujících dat v rámci databáze jsou centrálně řízeny. Data je možné využívat a aktualizovat v různém rozsahu a z různých hledisek uživatelem nebo současně více uživateli a s využitím různých přístupových metod. (Bíla, 1999, s. 5)

2.2 Výhody a využití databází

Velkou výhodou databází je jejich univerzalita, škálovatelnost a nezávislost. Se současným trendem webových aplikací a komunikace po Internetu jsou databáze stále více a více žádané. Data jsou uložena na jednom místě nezávisle na aplikacích, které je využívají. Tyto aplikace mohou být různorodé, například webová, mobilní, desktopová. Jedná se o tzv. perzistenci dat.

Klasickým příkladem využití perzistence dat může být denní přehled výkyvu teplot za určité období. Lze jej realizovat pomocí různých nezávislých aplikací. Zvolená aplikace každý den zaznamená do databáze hodnoty, do kterých lze přistoupit skrze webovou aplikaci. Ta zapíše data například do tabulky. Další možností je použít desktopovou aplikaci, která zobrazí data grafem. (Pokorný, 2013) (Bíla, 1999)

2.3 Databázový a souborový přístup

Databáze řeší problémy souborového přístupu, jako je vysoká míra redundance a nekonzistence dat. Redundancí dat se rozumí nadbytečnost či duplicita dat. Souborový přístup je typický tím, že ukládá data do jednoho nebo několika souborů. Z počátku musela každá aplikace mít svá vlastní izolovaná data. Nekonzistence dat pak vznikala, pokud se změnila hodnota informací u nějaké z duplicitních hodnot a u ostatních nikoliv. Integrita databáze znamená, že se data zadávají podle předem definovaných pravidel se snahou vyhnout se ztrátě konzistence dat

(Misha, 2010). Další termín security řeší ochranu před neoprávněným přístupem, kterou provádí pomocí tzv. autorizace. Tímto problémem trpěl právě souborový přístup. Jak již bylo zmíněno, databázový přístup ukládá data do centralizovaně zpracované struktury dat zvané databáze neboli báze dat. Fyzicky se data nejčastěji ukládají na disk. (Bíla, 1999)

2.4 Systém řízení báze dat

SŘBD (Systém řízení báze dat), dále jen DBMS (z anglického Database Management System), je softwarový systém, který uživateli umožnuje řízený přístup a manipulaci s databází (Conolly, 2009, s. 38). Tvoří pomyslnou vrstvu nacházející se mezi uživatelem a databází. S DBMS je spojena existence dvou typů jazyka. Jazyk pro definici dat DDL (Data Definition Language) a jazyk pro manipulaci dat DML (Data Manipulation Language).

DDL slouží k vytváření všech definic dat v databázi. Popis dat v jedné databázi se nazývá logické schéma databáze. DML se používá pro manipulaci dat v databázi. Tento jazyk se používá pro operace jako je vkládání, editace a mazání dat. Jazyk pro manipulaci s daty se nazývá dotazovací jazyk. Příkladem může být jazyk SQL (Structured Query Language), který zahrnuje DDL i DML a je dostatečně univerzální, avšak postrádá programovací struktury, jako jsou podmínky, cykly a další. Proto ho lze kombinovat s jiným programovacím jazykem. Podrobnější informace budou uvedeny v kapitole 4.3 SQL. Dalším příkladem dotazovacího jazyka je QBE (Query By Example), který k dotazování využívá grafické znázornění, nebo dotazovací jazyk pro XML (Extensible Markup Language) nazvaný XQuery od skupiny W3C (World Wide Web Consortium). (Connoly, 2009) (Pokorný, 2013, s. 8) (Bíla, 1999)

2.5 Databázová aplikace, databázový systém, báze dat

Další důležitý termín je databázová aplikace. Jedná se o aplikaci napsanou v některém programovacím jazyku, která slouží pro výběr, prezentaci, zpracování a tisk dat uložených v databázi. Databázový systém je ve své podstatě spojení databáze a systému řízení báze dat. Báze dat je kolekce dat vztahující se k určitému řešenému úkolu. (Havlíček, 1992, s. 11) (Bíla, 1999, s. 4)

2.6 Uživatelské skupiny

Uživatelé databázových systémů se dělí na 4 skupiny. Správci dat (administrátoři) jsou osoby odpovědné za autorizovaný přístup do databáze. Udělují práva přístupu jednotlivým uživatelům. Aplikační programátoři vyvíjejí uživatelskou aplikaci. Příležitostní uživatelé umějí

používat dotazovací jazyk a formulovat vlastní specifické dotazy pro manipulaci s daty. Uživatelé naivní bývají typicky neprogramátoři, kteří pracují s databází prostřednictvím aplikačního programu. (Pokorný, 2013) (Hronek, 2007)

3 Historie

Tato kapitola se bude věnovat historii zpracování dat, ale i samostatné definici pojmu dat a informací. Představí postupný vývoj automatizace, počátku počítačů až k modelu jedné z nejpoužívanějších relační databází a neopomene ani objektovou databázi. Zmíní příchod noSQL databází a v závěru nastíní budoucnost databázových systémů.

3.1 40., 50., 60., léta – data, zpracování dat

Od počátku 20. století se množství dat, které bylo potřeba zpracovávat, výrazně zvyšovalo a přestávalo být postupně v lidských silách je zpracovávat. Nicméně, spolu s tím se zlepšovala i technika, a už během druhé světové války byla k dispozici základní strojová automatizace (mechanické počítače). Poté se postupně přešlo k polovodičovým počítačům (60. léta), což byl další posun vedoucí ke zrychlení automatizace.

3.1.1 Data a zpracování dat

Pojmy data a informace bývají často zaměňovány. Definice dat podle Bíla (1999) „Data jsou údaje o okamžitém stavu skutečnosti, zachycená člověkem nebo strojem a uložená v našem vědomí nebo na jiném paměťovém médiu. Zároveň jsou data nositeli skutečné nebo potencionální informace.“ V informatice tvoří informace kódovaná data, která lze přijímat, odesílat či uchovávat, a jejich nosičem je signál. Informace snižuje míru neurčenosti, tzv. entropie, mezi odesílateli a příjemci. Druhem dat popisujeme jednotlivé typy informací zpracovávaných v databázi. (Merunka, 2002) (Bradwen, 2017)

Z počátku se převážně jednalo o tzv. ruční přístup k datům, kdy to byly především papírové kartotéky. Každá kartotéka obsahovala záznamy (kartotéční lístky) a samotná práce s kartotékou nebyla příliš složitá (Pokorný, 2013, s. 5). Ve své podstatě se evidovaly dané záznamy podle určitého kritéria např. abeceda, rok, lokalita. Problém nastal, když bylo potřeba záznamy, kterých bylo velké množství, aktualizovat. Vše se muselo ručně opravit, což bylo jednak časově náročné, a zároveň velmi nepraktické. Často pak docházelo k nekonzistenci dat. Použití kartoték bylo využíváno zejména v knihovnách, u lékaře a v dalších odvětvích. Se stejným systémem se lze setkat i v diářích, telefonních seznamech, můžeme ho brát jako předchůdce databází, které využíváme dnes. (Havlíček, 1992, s. 9)

V dnešní době je typické, že s databázemi pracuje počítač, který používá software fungující na principu klient-server. Server poskytuje databázi klientům. Klientský počítač využívá služeb serveru, který se nachází na Internetu.

3.1.2 Zpracování dat na medium, CODASYL

Z hlediska ukládání dat na medium se nejprve používaly děrné štítky, později magnetické pásky, a ještě později magnetické disky. Také je důležité si uvědomit, že se databáze vyvíjely společně s počítači a také byly omezené tehdejšími technologiemi. V 50. letech vznikaly primárně počítače pro hromadné dávkové zpracování dat. To přispělo ke vzniku prvních programů šitých tzv. na tělo. Jedná se o souborový přístup, který je vysvětlen v předešlé kapitole 2.3 Databázový a souborový přístup. Programy byly napsané některými z prvních programovacích jazyků, jako například FORTRAN nebo ALGOL. Vstupní data byla závislá na pořadí a požadovala samostatný datový soubor, nejčastěji uložený na magnetické nebo děrné pásce. Vznikala opět velká míra redundance a nekonzistence dat. De facto se jednalo o všechny nevýhody souborového přístupu. Každá změna ve struktuře dat měla za následek úpravu programu. V 60. letech vzniklo seskupení CODASYL (Conference/Committee on Data Systems Languages) spolu s vývojem programovacího jazyka COBOL. (Bíla, 1999) (Hronek, 2007)

3.2 Model databáze, 70. a 80. léta, relační databáze

Model databáze, též označovány jako datový model, se dělí podle způsobu ukládání dat a vazeb, a tím vytváří danou strukturu databáze. Tu reprezentuje na logické úrovni, a to na rozhraní systému řízení báze dat DBMS. Podle definice Merunky (2002, s. 16) „Datový model představuje definici formalizovaných přístupů k uložení a práci informací v paměti počítače.“

3.2.1 Hierarchický a síťový data model

Historicky nejstarším počítačem zpracovávaným modelem je hierarchický model HDM (Hierarchical Database Model). Ten vyvinuly firmy IBM a North American Aviation na konci 60. let pro vesmírný program Apollo. Jedná se o stromovou strukturu, kdy záznamy tvoří uzly uspořádané podle úrovní. Na nejvyšší úrovni se nachází kořen, který je provázán s dalšími záznamy podle jednoho ze vztahu rodič/potomek tzv. rodičovský vztah. Každý uzel má právě jeden vztah směrem k rodiči a žádný, jeden nebo více směrem k potomkům.

Síťový model NDM (Network Data Model), jehož příkladem je systém IDMS (Integrated Database Management System) od firmy IBM, se využíval v době sálových počítačů a byl standardizován v roce 1971 na konferenci CODASYL. Odstranil problémy hierarchického modelu a rozšířil jej tak, že vazby mezi záznamy (uzly) nejsou závislé na dané úrovni a mohou být propojeny mezi více záznamy tzv. sety (mnohonásobné vztahy).

Síťové modely NDM, stejně tak jako hierarchické modely HDM, jsou velice rychlé a zpočátku poráží relační model, který přišel později. To bylo dáno především tehdejším výkonem počítačů. Oba tyto modely trpěly však problémy, které byly pro programátora nepraktické. Každá změna programu je totiž nutila změnit celou strukturu. (Bíla, 1999) (Merunka, 2002) (Misha, 2010)

3.2.2 Relační databáze a relační model (70. léta)

V 70. letech přichází zaměstnanec firmy IBM, informatik a matematik Edgar Frank „Ted" Codd, který publikoval v roce 1970 článek „A Relational Model of Data for Large Shared Data Banks", ve kterém představil návrh na implementaci nového datového modelu, který nazval relační model RDM (Relational Data Model). Tím se „Ted" Codd považuje za zakladatele budoucích relačních databází. Hlavní důvod, který vedl „Teda" Codda k novému návrhu, byla celková nespokojenost s dříve zmíněnými data modely HDM, NMD, které využívaly CODASYL databáze, a i samotné IBM ve svém produktu IMS (Information Management System). To byl právě jeden z důvodů, proč se od tohoto návrhu zpočátku IBM distancovala, protože by musela zavrhnout svůj produkt IMS, a tak tento návrh byl brán spíše jako kuriozita. (Žák, 2001)

Relační databáze RDBMS (Relational Database Management System) jsou založené na relačním modelu RDM. „Ted" Codd při návrhu RDM využíval matematický aparát v podobě relačního kalkulu a algebry. Zbavil tak strukturu myšlenky hierarchických vztahů. Základní strukturu pro ukládání dat uvažoval tabulku, kde každý řádek odpovídá jednomu záznamu s jedinečným ID klíčem a každý sloupec odpovídá jednomu atributu. Další zjednodušení bylo v podobě příkazů zapsaných ve srozumitelném anglickém jazyce, nikoliv tedy ve specifickém jazyce se specifickými znaky. Dále tento návrh uvažoval o nezávislosti na použitém hardwaru a způsobu fyzického uložení. Uživatel manipuluje s množinou dat, a ne pouze s jedním záznamem. Tyto operace vychází z relační teorie, jako je sjednocení, kartézský součin, rozdíl, selekce, projekce a spojení. Databáze funguje na principu klient-server. (Žák, 2001) (Kokeš, 2005)

3.2.3 Postupný vývoj relační databáze (70. - 80. léta)

Počátek relační databáze RDBMS tedy vznikl v 70. letech, ale trvalo 10 let, než tuto myšlenku bylo možné uplatnit v praxi. Takže až 80. léta lze nazvat počátkem zlatého věku relačních databází. Vývoj probíhal ve dvou větvích. Projekt System-R od firmy IBM probíhal mezi lety 1974 až 1979 a měl celkem dvě fáze. V první fázi šlo o testování a ověřování samotného

relačního modelu (RDM). Původním dotazovacím jazykem v projektu System-R byl jazyk SEQUEL. V druhé fázi vznikl jazyk SEQUEL 2, který je později přejmenován na jazyk SQL. Ve druhé větvi přichází projekt Ingres, který byl vyvíjen na univerzitě UC-Berkeley společně s podporou armády. Snahou bylo vytvořit systém určený pro geografická data tehdejší Berkeleyské ekonomické skupiny. Ingres používal svůj vlastní jazyk QUEL. (Žák, 2001) (Kokeš, 2005)

Na konci 70. let navrhl americký informatik Jim Gray transakční model pro SQL. Na něj v roce 1983 navázali Andreas Reuter a Theo Härder a zavedli zkratku ACID. Jedná se o model konzistence dat. Pojem transakční model a ACID bude rozebrán v kapitole 4.2. Různé koncepty relačních databází a SQL. (Carvelli, 2015)

V roce 1980 přichází na trh první SQL databáze od firmy Oracle. Tuto firmu založil americký podnikatel Larry Ellison, který se inspiroval Systemem-R. Firma IBM v té době přišla na trh se svým produktem DB2. V roce 1982 byl projekt Ingres ukončen a transformován do projektu Postgres. Jednalo se o snahu vytvořit relačně-objektovou databázi. Tato databáze umožňuje uživatelům definovat vlastní metody pro přístupy a manipulaci s daty. Projekt Postgres trval do roku 1994. V následujícím roce byl přejmenován na Postgres95 pod vedením dvou studentů UC-Berkeley. V 1996 přechází do open source podoby a byl přejmenován na dnes známé PostgreSQL databáze. (Žák, 2001) (Oracle 3, 2019)

3.3 90. léta – objektové databáze, objektově relační databáze

Se zvyšující se popularitou programování v objektově orientovaných jazycích OOP (Objected Oriented Programming) přichází tento trend i do databází. Vznikají objektově orientované databáze OODBMS (Object-oriented Database Management System) založené na vlastním objektovém datovém modelu ODM (Object Data Model). Tento datový model využívá typické aspekty objektového programování, jako je tvorba objektů a tříd, zapouzdření, dědičnost, polymorfismus, jednoznačná identifikace objektu OID (Object Identifier) a reference mezi objekty. Princip OOP vychází z modelování reálného světa tak, jak je pro člověka přirozené pomocí objektů popisující reálný svět. Objekt vzniká instancí třídy. Třída má atributy a metody.

Byla zde snaha ulehčit a urychlit práci s databází. U relačních databází RDBMS lze vyjádřit všechny vztahy pouze tabulkami, což v určitých případech vede k vytvoření velkého množství tabulek.

Neexistuje žádný oficiální standard. Respektive standardem se stala kniha Morgana Kaufmana „The Object Database Standard: ODMG-V2.0“. Dotazovací jazyky jsou zde objektově orientované jazyky jako jsou C++, Java a další. Objektové databáze nabyly popularity v oblastech inženýrství, telekomunikace, molekulární biologii, ale nenahradily relační databáze. (Thakur)

Existuje i řada problémů. Zmíněný datový model není oficiálně uznávaný, a s tím je spojený nedostatek standardů. Ve většině těchto modelů chybí teoretický základ. Největším problémem je konkurence v podobě vznikajících produktů založených na objektově-relačně datovém modelu ORDBMS (Object Relational Database Management System). (Masarykova univerzita)

3.3.1 Problémy objektového programování

Další nedostatky jsou spojené s obecnými problémy objektového programování. Jedna s z námitek, kterou popisuje Račanský (2017), je například míchání datové struktury s funkcemi. Podle něj se jedná o absolutně odlišné věci. Funkce jsou tvořeny pomocí imperativů „proveď toto a toto“. Její vstupy a výstupy jsou právě ony datové struktury, které jsou v podstatě deklarativní a nedělají nic víc. Podle dalších tvrzení je funkce obvykle vnímána jako část systému, jejíž schopnost je transformace jednoho typu datové struktury do druhé. Další námitka může být, že všechno musí být objekt. Proč se stalo objektové programování populární. Opět lze vyjít z tvrzení Račanského (2017) „Jednak kolem toho vznikl velký rozruch a vzniklo nové odvětví softwarového průmyslu a dále zde byla domněnka snadného naučení programování v OOP, což se ukázalo, že tomu tak není.“

3.3.2 Objektová a objektově relační databáze

Pro shrnutí platí, že vztahy v OODBMS jsou reprezentovány pomocí odkazů identifikátoru objektu OID a záznamy ukládá do objektů. Používá se indexování na disk pro ukládání objektů, a proto jsou schopné poskytnout trvalé úložiště pro strukturované objekty. OODBMS dokáže zpracovávat větší a složitější data než RDBMS. Dotazovacím jazykem je některý z vyšších programovacích jazyků C++ a Java.

Objektově relační databáze ORDBMS je hybrid objektové OODBMS a relační databáze RDBMS. Vztahy jsou reprezentovány pomocí klíčů, stejně jako v RDBMS, a tudíž data ukládá do tabulek. Ve své podstatě rozšiřuje relační model o objekty. Dotazovacím jazykem je rozšířená verze SQL, v praxi se také uplatňuje ORM (Object-relational mapping), které

zajišťuje automatickou konverzi mezi relační databází a objektovým jazykem. Její snaha je odstranit nejvyšší míru psaní SQL. (Parahar, 2019) (Masarykova univerzita)

Specifika	RDBMS	OODBMS	ORDBMS
Celý název	Relational Database Management System	Object Oriented Database Management System	Object Relational Database Management System
Způsob ukládání dat	Data ukládá do tabulek	Data ukládá jako objekty	Data ukládá do tabulek
Dotazovací jazyk	SQL	C++, Java	Rozšířená verze SQL
Složitost dat	Zpracovává jednoduchá data	Zpracovává velká a komplexní data	Zpracovává jednoduchá a komplexní data
Klíče	Primární klíč ID	OID identifikátor	Primární klíč ID
Distribuce	MySQL	Object Store	PostgreSQL
Popularita	Vysoká	Nízká	Střední

Tabulka 1 - Přehled RDBMS, OODBMS, ORDBMS

3.4 21. století – distribuované systémy, vznik noSQL databází

Nástupem 21. století vzrůstal zájem o distribuované úložné systémy. Cílem těchto systémů je rovnoměrně rozložit data do více vzájemně propojených uzlů (počítačů, serverů) v síti, nikoliv tedy na jeden počítač. To umožňuje využívat méně výkonné stroje za účelem snížení nákladů. Poskytuje snadné škálování, omezení latence a celkové selhání. Problém ovšem nastává ve chvíli, kdy selže daný uzel nebo nelze navázat spojení.

3.4.1 CAP teorém a vznik distribuovaných systémů

V roce 1999 americký informatik Eric Brewer definoval CAP teorém, nebo také Brewerův teorém. Tvrdí, že distribuovaný datový sklad nemůže současně nabídnout více než dvě ze tří

uvedených záruk. Relační databáze tedy splňují pravidla CA a noSQL databáze splňují CP nebo AP:

- C – Konzistence: Data v databázi zůstávají konzistentní. Například po aktualizaci systémů uživatel uvidí stejná data.
- A – Dostupnost: Systém je neustále k dispozici.
- P – Tolerance oddílů: Systém bude nadále fungovat, pokud komunikace mezi servery není spolehlivá. Díky tomu, že lze servery rozdělit do několika skupin viz princip distribuovaných systémů, dojde k rozložení zátěže mezi více uzlů. V roce 2002 byla potvrzena pravost CAP teorému. (Foote, 2018) (Wang, 2017) (Bártík, 2011)

Jedním z průkopníků distribuovaných systémů byla společnost Google. Ta v roce 2004 publikovala dokument o popisu distribuovaného systému GFS (Google File System) a vybudovala na něm úložný systém, který pojmenovala Bigtable. (Carvelli, 2015)

3.4.2 BASE model

Po roce 2008 se stává populární model BASE, založený na CAP teorému, jakožto model konzistence dat, a zároveň je alternativou k modelu ACID, který je oblíbený u RDBMS. Model BASE nabízí mnohem flexibilnější řešení vhodnější pro nestrukturovaná data, se kterými pracují noSQL databáze. Celkově model BASE poskytuje méně záruk než model ACID:

- BA – Basic Availability: Zaměřuje se na dostupnost dat i v případě více selhání díky využívání distribučních systému.
- S – Soft State: Opouští požadavky na konzistenci dat. Ta by měla být řešena vývojářem, nikoliv na úrovni databáze.
- E – Eventual Consistency: Jediný požadavek na konzistenci dat je, aby data v budoucnu konvergovala do konzistentního stavu. (Foote, 2018) (Chapple, 2020)

3.4.3 Vznik noSQL databází

Termín noSQL byl poprvé použit v roce 1998 vývojářem Carlem Strozzim, který tak pojmenoval svou „relační“ databázi, která ovšem nepoužívala jazyk SQL.

V červnu 2009 v San Franciscu uspořádal softwarový inženýr Johan Oskarsson setkání, kde představil nové technologie pro zpracování a ukládání dat. Předvedené produkty Bigtable nebo Dynano nefungovaly na relačním principu. Po schůzce bylo nutné najít nový termín, který by zastřešoval určené produkty pro hashtag na Twitteru. Jeden z účastníků, Eric Evancs, navrhl

termín noSQL, který neměl mít žádný hluboký význam, ale později se celosvětově rozšířil. Zkratka noSQL znamená „žádné SQL“ nebo také „nejen SQL“. Z anglicky vycházejících zkratek (Not Only SQL, Non SQL, Non Relational). Nevychází tedy z žádné uznatelné definice nebo akceptované vědeckou institucí. Obecně zastřešuje takové databáze, které buď nejsou relační, nebo nepoužívají strukturovaný dotazovací jazyk SQL. Nejčastěji fungují na principu Key-Value. Dnes je více než přes 150 různých noSQL databází. (Vasiliev, 2013) (Foote, 2018)

3.5 Krátké zamyšlení nad budoucností

Podle prezentace databázového analytika Craiga Mullinse z Data Summitu v květnu roku 2019, který vycházel ze statistik IDC (International Data Corporation) platí, že od roku 2005 do roku 2020 se data ze 130 exabytů rozrostla na 40 000 exabytu ($1\ 024^6$). Pro lepší představu se jedná o 40 bilionů gigabytu ($1\ 024^3$), kde zhruba 5200 gigabytu připadá na každého člověka v roce 2020. IDC odhaduje pro rok 2025 až 163 zettabytu ($1\ 024^7$). Je nutné podotknout, že se zvětšujícím objemem dat je potřeba brát větší ohled na samotnou bezpečnost dat. IDC uvádí, že v období 2010 až 2020 nastává až padesátinásobný nárůst dat. Pro rok 2020 IDC uvádí 40 zettabytu, nicméně IDC dodává, že skutečně chráněných dat je jen polovina. (Wells, 2019)

3.5.1 Budoucnost noSQL, SQL databází

Z hlediska databází se dá předpokládat nárůst popularity v odvětví noSQL databází, díky dobré podpoře a moderním trendům, jako jsou Cloudy, IoT, AI, Big data. Tyto trendy jsou vysvětleny v kapitole 5.2 Související koncepty – Big data, moderní trendy.

Nelze říci, že by SQL databáze byly nahrazeny noSQL databázemi, jelikož mezi nimi jsou rozdíly, o kterých tato práce ještě bude pojednávat v kapitole 6 Shrnutí Teorie. Těžko předpokládat, zda se tradiční SQL databáze technologicky dají zdokonalit, jelikož jsou podstatně starší než noSQL databáze, a tudíž dostatečně etablované na trhu. Jeden z hlavních limitů SQL databází je teoretické omezení výpočetního výkon, díky absenci horizontálního škálování, což by teoreticky při stálém zvyšování objemu dat mohl být problém. Nicméně existuje řešení tohoto limitu. Mohl by to být budoucí rozvoj tzv. NewSQL databází. Jedná se o klasické relační databáze RDBMS s podporou OLPT, které jsou obohaceny o zmíněné horizontální škálování při zachování transakčního modelu ACID. (Simsek, 2019)

Další možností je, že v budoucnosti přijde nový druh databáze, který není SQL, NewSQL, ani noSQL. Lze se ale pozastavit nad myšlenkou, zda by vůbec nový druh databází byl potřeba.

4 Relační databáze a SQL

Cílem této kapitoly je představit relační databáze z hlediska využití a porovnání konkrétních databázových distribucí a vysvětlit klíčové koncepty pro obecné pochopení relačních databází.

4.1 Využití relačních databázových systémů

Relační databáze se staly populární díky své jednoduchosti a snadné pochopitelnosti. Mezi nejznámější zástupce relačních databázových systémů DBS patří Oracle, PostgreSQL, MS SQL, MySQL, MariaDB, SQLite a DB2. Dané DBS se liší podle použití, licencemi a historií. Populárním řešením pro tvorbu webových aplikací menšího rozsahu se stává typicky MySQL, popřípadě MariaDB, spolu s použitím jazyka PHP nebo Python. Hlavním důvodem popularity MySQL je projekt WordPress, který dnes ovládá přibližně 60 % systémů CMS nebo 34 % celého webu. (Jankov, 2020)

MySQL a MariaDB

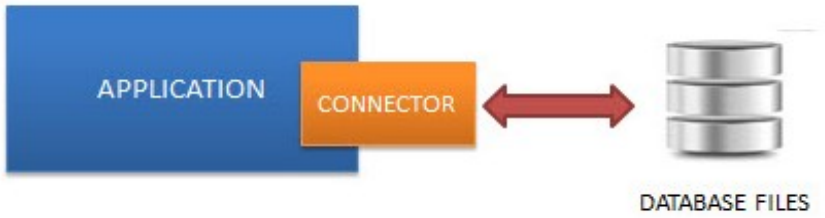
MySQL vyvinuli v roce 1995 vývojáři Michael „Monty“ Wideius a David Axmark. V té době konkurovala drahým řešením od společností Microsoft a Oracle (Jankov, 2020). V současné době vlastní MySQL společnost Oracle. Kvůli přijetí duální licence bezplatné GPL a komerční placené licence vznikla MariaDB, která si ponechala GPL licenci. MariaDB vytvořili v říjnu 2009 již zmiňovaní vývojáři z verze MySQL. Databáze dostala jméno podle dcery Michaela Wideiuse. Cílem MariaDB je udržování kompatibility s MySQL s cílem zachovat si bezplatnou licenci. (Jankov, 2020). Jak bylo zmíněno, oba tyto databázové systémy jsou populární ve webovém průmyslu, ale nejsou zcela totožné. Při vývoji webových aplikací bývají například součástí tzv. „LAMP stacku“. Jedná se o využití následujících technologií: Linux + Apache + MySQL + PHP. Nelze říci, která z těchto DBS je lepší. Každá z nich nachází své uplatnění. WordPress je svobodný redakční systém CMS, patřící spolu s Joomlou a Drupalem mezi tři nejrozšířenější na trhu. Jedná se o komplexní balíček služeb se vším potřebným pro provoz webových stránek. Často převyšuje i placené produkty. Konkrétně WordPress využívá služeb MySQL a jazyka PHP. (Oskow, 2017) (Sarig, 2019) (WordPress)

MariaDB	MySQL
Google	NASA
Wikipedia	Apple Inc.
Tumblr	eBay
Nimbuzz	Yamaha
Paybox	YouTube
Teleplan	Hewlett Packard
OLX	Facebook

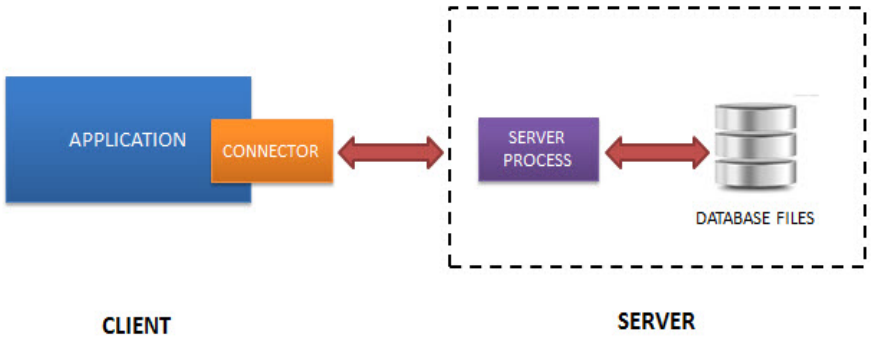
Tabulka 2 - Přehled využití MariaDB a MySQL

SQLite

Jejím autorem je D. Richard Hipp. Nejedná se o klasickou relační databází založenou na modelu klient-server, jako je tomu u MySQL. Jedná se spíše o malou knihovnu. Databáze je uložena v souboru na disku a předpokládá se, že k datům přistupuje software běžící na stejném počítači nebo serveru. Hlavní výhodou je jednoduchost, celkově malá velikost, minimální konfigurace a jedná se o „public domain licence“ (volné dílo). Jelikož databáze je uložena v jednom souboru, zálohování je v tomto případě snadné. Mezi hlavní nevýhody patří poměrně nízký výkon a nemožnost realizace paralelního zápisu do databáze. Také nelze z databáze číst, pokud se do ní zapisuje. Její uplatnění může být například jako cachování dat nad nějakým větším databázovým systémem, jako je třeba PostgreSQL. Popřípadě ji lze použít při tvorbě nějaké malé jednoduché desktopové, mobilní aplikace, která umožňuje využít vlastní souborový systém. SQLite se může pochlubit přes 1 trillionem aktivně používaných SQLite databází. (SQLite) (Kumst, 2016)



Obrázek 1 - Ukázka architektury využívající SQLite



Obrázek 2 - Ukázka architektury Klient-Server

Access

Je proprietární program pro správu relační databáze od společnosti Microsoft nabízený jako součást balíku Microsoft Office. Access poskytuje grafické uživatelské rozhraní pro správu databáze. Nativní databází je „Microsoft Jet Database Engine“, ale dokáže pracovat i s MS SQL, či Oracle databází. K přístupu ke zmíněným databázím využívá ODBC (Open Database Connectivity), což je specifické API, které je nezávislé na jakémkoliv DBMS nebo operačním systému. Access se hodí pro malé až středně velké organizace, nebo tam, kde nevystačí tabulkový procesor, ale není potřeba nasazovat velkou databázi. Poskytuje výhody v podobě snadné práce díky grafickému rozhraní a další funkcionality, jako například tvorbu formulářů nebo sestav. Je kompatibilní s vývojem v. NETu, a tak popularitu získal zejména jako desktopová databáze. Jedním z problémů může být přílišná orientace na Windows, a tak není kompatibilní s jinými operačními systémy. Nabízená manipulace pomocí SQL není tak robustní a je značně omezená oproti v MS SQL nebo Oracle serveru. Výrazným problémem je, že Access není vhodný pro webové aplikace, které v dnešní době dominují, co se týče vývoje aplikací. Dalším problémem je například vzdálený přístup pomocí VPN, který výrazně zpomaluje výkon. (Roth, 2017) (Besthosting, 2020) (Sqlprogrammers, 2014) (Learnitanytime, 2013)

4.2 Různé koncepty relačních databází a SQL

Relace

Základním principem relačních databází RDBMS a samotného relačního modelu RDM jsou relace vycházející z relační algebry. Mluví-li se o relaci, mluví se o „vztazích“ mezi prvky. Z hlediska relačního modelu RDM se místo o souboru dat, mluví o relaci a místo o záznamech, se mluví jako o prvcích relace nebo také o n-ticích (Bíla, 1999, s. 18). N-tice se skládají z jednotlivých složek entit, které vyjadřují vlastnost n-tice a kterým se říká atribut. Každý atribut by měl být nedělitelnou (atomickou) hodnotou. Relaci můžeme zobrazit ve formě dvourozměrné tabulky, ve které řádky tvoří jednotlivé n-tice a sloupce atributy. (Bíla, 1999)

Podle definice Kokeše (2005, s. 142) „Relace popisuje vztahy mezi entitními typy tvořící binární vztah (tzn. vstupující do nich dvě entity), u kterých lze rozlišit směr.“ Jedna entita je řídící (rodičovská nebo hlavní) a druhá podřízená (dceřinná). Pro popis vztahů mezi sebou se používá termín kardinalita vztahů. Podle Skřivana (2000) „Entita je prvek reálného světa, který je popsán charakteristickými vlastnostmi, zvané atributy.“ (Kokeš, 2005)

Kardinalita vztahů

Kardinalita popisuje vazbu mezi entitami vyjádřenou v poměru. S kardinalitou se také pojí tzv. parcialita vztahů neboli povinnost, či volitelnost existence. Například „Musí mít každý muž manželku?“ (informační technologie, 2009). Uvedený příklad kardinality vztahů vychází podle Lasáka (2010).

- 1:1 - Jednomu záznamu v tabulce odpovídá právě jeden záznam z druhé tabulky.
- 1:N - Jeden záznam v první tabulce odpovídá několika záznamům ve druhé tabulce.
- N:M – Více záznamu v první tabulce odpovídá více záznamů v druhé tabulce.
- Žádný vztah – mezi záznamy v tabulkách není žádný vztah.

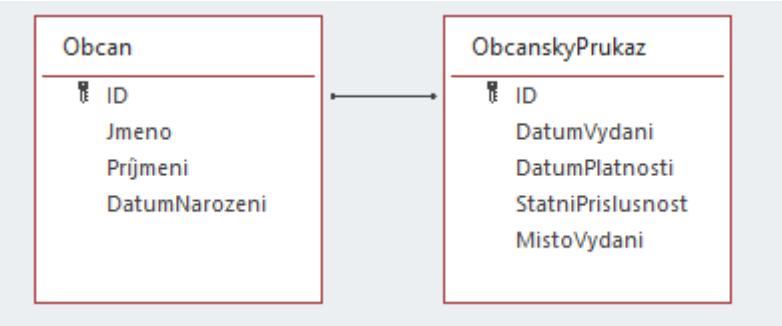
Kokeš (2005) nehovoří o záznamech tabulky, ale o instanci jednotlivých rodičovských a dceřiných entit. Příklad vztahu 1:1 definuje následovně. Každá instance z rodičovské entity odpovídá maximálně 1 instanci z dceřiné entity. Nicméně význam je stejný, jde pouze o jiné formulování.

Využití jednotlivých relací

Podle Lasáka (2010) relace typu 1:N jsou v projektech nejčastější. Kdežto vazby typu N:M se v praxi převádějí na vztahy 1:N. Zbývající vztah 1:1 je nejjednodušší vazbou, ale ne zcela použitelnou, až na výjimky. Následují konkrétní příklady daných relací.

Relace 1:1

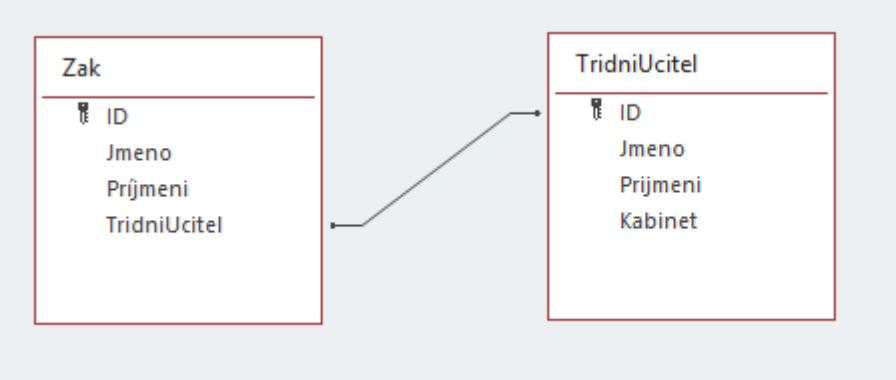
V následujícím příkladě jsou entity občan a občanský průkaz. Pro upřesnění platí, že entita v zadaném příkladě je název tabulky (občan) a atributy jsou vlastnosti dané entity (jméno). Záznam pak vzniká vyplněním atributu (jméno Pavel příjmení Novák). Pro zjištění parciality by pak platila následující otázka. Může občan existovat bez občanského průkazu a může občanský průkaz existovat bez občana? V tomto příkladě občanský průkaz nemůže být přidělen neexistujícímu občanovi a občan musí mít přidělen nejvýše jeden občanský průkaz, ale nemusí mít žádný (Mendelu 1). Relace 1:1 říká, že každému občanovi je přidělen právě jeden občanský průkaz, a zároveň platí, že každý občanský průkaz je přidělen právě jednomu občanovi.



Obrázek 3 - Ukázka relace 1:1 v MS-Access

Relace 1:N

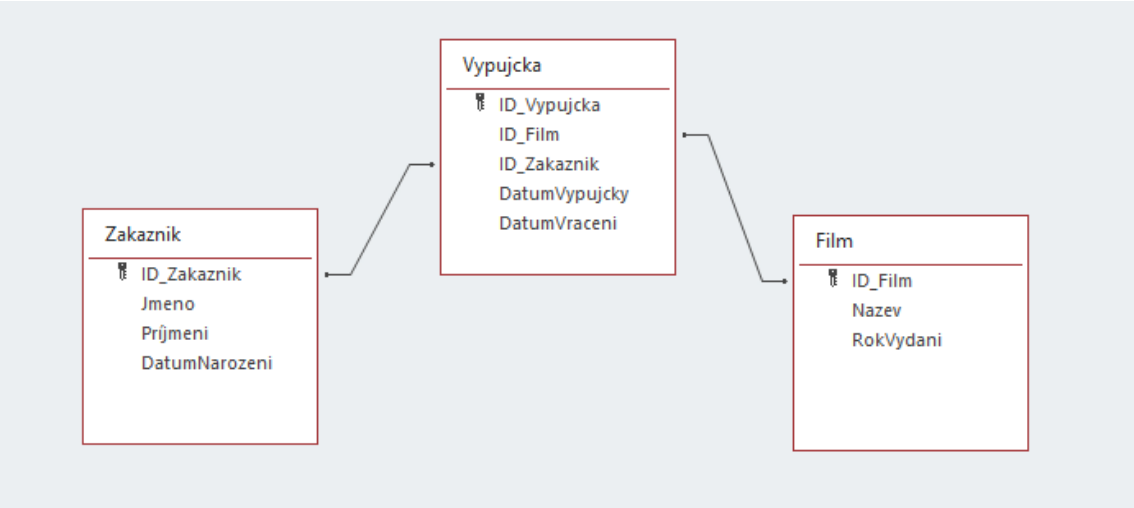
V tomto příkladě jsou entity žák a třídní učitel. Relace 1:N říká, že každý žák má právě jednoho třídního učitele, ale třídní učitel může mít více žáků.



Obrázek 4 - Ukázka relace 1:N v MS-Access

Relace N:M

Entity zákazník, film a propojovací entitou je výpůjčka. Relace N:M říká, že daný zákazník si může vypůjčit více filmů a více filmů může být vypůjčeno více zákazníky v závislosti na entitě výpůjčky, a to podle data vypůjčení a vrácení.



Obrázek 5 - Ukázka relace N:M v MS-Access

Relace a matematika

Z matematického hlediska relace umožňují dávat dohromady prvky množin, které jsou spolu v nějakém vztahu tvořící uspořádané n-tice vycházející z kartézského součinu. Kartézský součin je množina všech uspořádaných dvojic prvků množiny A B. Formálně zapsáno $A \times B = \{(a,b) \mid a \in A, b \in B\}$. (Mendelu 2)

Klíče

Klíče neboli klíčové atributy, se dělí podle dvou hledisek. První dělení rozděluje na primární a sekundární klíče, kde primární klíč slouží k jednoznačné unikátní identifikaci řádku a nesmí být prázdný, tzv. nulové hodnoty, a neměl by se příliš často měnit. Zpravidla bývá jen jeden v dané tabulce například rodné číslo. Dále existuje tzv. umělý primární klíč, nejčastěji „ID“, ten se používá, když žádný z údajů v tabulce nelze použít jako primární klíč. Funkce primárního klíče umožňuje vzájemně propojit dvě tabulky a vytvořit mezi nimi relaci. Kdežto sekundární klíč není jedinečný, například „jméno“, a tak má spíše doplňkový význam. (Kokeš, 2005)

Druhé dělení klíčů je na vlastní a cizí klíče. Toto dělení se pojí s relacemi mezi entitami. Pokud je klíčový atribut součástí entity vždy, bez ohledu na její existenci, pak se jedná o vlastní klíč (Kokeš, 2005, s. 140). Cizí klíč je pak klíčový atribut v jiné entitě za účelem vytvoření vazby

mezi entitami. Kokeš (2005) jinými slovy říká: „Cizí klíč definuje vazbu z podřízené do nadřízené entity.“

Indexy

Databázové indexy slouží pro zrychlený přístup k datům, podle kterých se dotazuje. Dotazování je vznášení požadavků na data z databáze. Mezi tyto požadavky patří operace spojené s manipulací dat, jako je vyhledávání, třídění, spojování tabulek. Index je ve své podstatě databázový soubor, který řadí tabulku podle jiných atributů, než je defaultně seřazená. Patří sem primární index PRIMARY, který vytváří automatické indexy u primárních klíčů, a vytváří tak jedinečnost údajů. Unikátní index UNIQUE také zajišťuje jedinečnost údajů, ale neváže se pouze na klíče. Vedlejší index INDEX je pak klasický index vytvořený nad sloupcem. Nicméně, není vhodné vytvářet indexy všude, protože každý index stojí určitý výkon u operace měnící obsah sloupců, jako je vkládání, mazání (SQL operace update, insert). Indexy se implementují pomocí své vlastní datové struktury zvanou B-Strom. (Vrána, 2003)

Transakce a ACID

Transakce je svazek jedné nebo více operací, která převede databázi z jednoho konzistentního stavu do druhého. Přesnější definice je od Bíla (1999) „Transakce je posloupnost operací, které realizují jednu nebo více operací s databází a s kterou se zachází jako s jedním celkem.“ Typické pro transakce je, že buď se provedou všechny operace, nebo se neprovedou vůbec. Transakce řeší spolehlivost zpracovávaných a uchovávaných dat, tak i současný přístup více uživatelů do databáze (Merunka, 2002, s. 34). Transakce rovněž řeší bezpečnost systému v případě havárie. To se stalo hlavními požadavky pro DBMS, pro které zajišťují transakce určité záruky, a vznikla tak transakční pravidla zvaná ACID. Smyslem těchto pravidel a samotných transakcí se stal fakt, že dnešní doba je odkázaná na komunikaci pomocí počítačových sítí a zmíněné závislosti na rychlosti a spolehlivosti zpracovávaných dat. Může tak docházet ke ztrátě dat, což může představovat velkou finanční ztrátu, popřípadě samotný zánik společnosti využívající databázové systémy.

ACID

- A – Atomic: Všechny operace a transakce jsou nedělitelné (atomické) a provádějí se jako celek. Pokud selže nějaká část transakce, stav databáze se nezmění a transakce se neprovedou vůbec.

- C – Consistent: Transakce musí dodržovat definovaná pravidla a neporušovat tak integritu databáze.
- I – Isolated: Zajišťuje souběžné provedení transakcí vedoucí ke změně stavu systému. Dále zajišťuje, že neúplná transakce nemůže ovlivnit další transakci, a není tak pro ni viditelná.
- D – Durable: Všechny úspěšně provedené transakce tzv. „commit“ budou stálé a permanentně uložené na disku, a tak nemohou tak být ztraceny. (Hugg, 2015)

Normalizace databáze

Normalizace databáze se pojí s relačními databázemi RDBMS. Utváří určitá pravidla, tzv. normální formy, dále jen NF. Normalizace je tedy technika, která se používá při návrhu optimální relační databáze. Základní myšlenkou normalizace je rozdělit tabulky do menších dílčích tabulek a zamezit zbytečné redundanci a ztráty konzistence dat. Podle jiné definice od Merunky (2002, s. 41) „Normalizace dat představuje takový způsob seskupení takových prvků do struktury záznamů, který zabraňuje problémům s jejich aktualizací.“ Autorem níže uvedených normalizací NF je zakladatel relačního modelu RDM Edgar Frank „Ted“ Codd.

- 1NF – Relace obsahuje pouze nedělitelné (atomické) atributy. Když nesplňuje je potřeba relaci rozložit do několika menších relací, tak, aby splňoval 1NF.
- 2NF – V jedné relaci mezi atributy existuje tzv. funkční závislost. Každý neklíčový atribut je závislý na primárním klíči. 2NF se týká jen tabulek, kde primární klíč je složený.
- 3NF – Všechny neklíčové atributy musejí být závislé jen na primárním klíči, a ne mezi sebou.
- BCNF (Boyce Coddova normální) - Podle Merunky (2002) někdy zvaná jako 3.5NF. Všechny atributy musejí být závislé jen na primárním klíči, a ne mezi sebou. Oproti 3NF musí platit i uvnitř složeného klíče.

Existují však ještě další NF. Nejčastěji se udává 5NF s tím, že každá vyšší NF musí splňovat pravidla té předchozí (Skřivan, 2000) (Kokeš, 2005) (Merunka, 2002, s. 41) (Goel, 2020)

4.3 SQL

Jazyk SQL (Structured Query Language) je dotazovací a zároveň deklarativní jazyk, používaný v RDBMS. Dotazy jsou příkazy sloužící k ovládání databáze. Tyto příkazy dělíme do čtyř kategorii: DDL, DQL, DML, DCL.

Deklarativní programování je programovací paradigma, kde kód vyjadřuje logiku výpočtu, nikoliv jeho průběh (Pekař, 2019). Zjednodušeně se řekne „co se má udělat“ než „jak se to má udělat“, což je příklad imperativního programování. Příkladem imperativního jazyka je například jazyk C.

DDL (Data Definition Language) slouží k definování samotné databáze. Patří sem příkazy:

- CREATE pro vytvoření databáze nebo jejich objektů, jako jsou tabulky, indexy, funkce, procedury a triggery, česky spouštěče.
- ALTER slouží ke změně databázové struktury. Například ALTER TABLE dovoluje přidávat, mazat, upravovat strukturu tabulky.
- DROP slouží k mazání objektů databáze.
- RENAME pro přejmenování existujícího objektu.
- TRUNCATE pro odstranění všech záznamů z tabulky.
- COMMENT pro přejmenování existujícího objektu.

DQL (Data Query Language) se používá k provádění dotazů:

- SELECT pro načtení dat z databáze.

DML (Data Manipulation Language) jsou takové příkazy, které slouží pro manipulaci s daty v databázi. Patří sem následující příkazy:

- INSERT slouží ke vkládání dat do tabulky.
- UPDATE slouží k aktualizaci existujících dat v tabulce.
- DELETE slouží k mazání záznamů v tabulce.

DCL (Data Control Language) je kategorie, do kterých patří takové příkazy, které se zabývají právy:

- GRANT uděluje uživatelům přístupová práva.
- REVOKE ruší uživatelům přidělená práva.

TCL (Transaction Control Language) je kategorie příkazů zabývající se transakcí v databázi:

- COMMIT zavazuje, „commituje“, transakci.
- ROLLBACK vrátí transakci v případě jakékoliv chyby.

- SAVEPOINT umožňuje nastavit místo uložení v rámci transakce. (Geeksforgeeks)

SQL cursor

Podle definice je to kurzor, který ukazuje na konkrétní řádek v rámci výsledného dotazu. V RDBMS jsou operace prováděny na řádcích. Ukazatel lze přesunout z řádku na řádek. Kurzory se používají tam, kde nelze použít operace založené na sadě například příkaz SELECT, který vrací sadu řádků. Problém velkého množství kurzorů je značné zabírání systémové paměti. (Wenzel, 2020) (Ashraf, 2019) (Onet, 2016)

5 noSQL databáze

Cílem této kapitoly je představit noSQL databáze. Seznámit se s jejich využitím, jednotlivou kategorizací, včetně uvedení určitých zástupců. Dále bude pozornost věnovaná klíčovým konceptům databází a moderním trendům.

Využití noSQL databází

S navyšující se přenosovou kapacitou dat, díky stále se zlepšujících technologii přenosu po sítích, se zvyšuje i množství dat. noSQL databáze vznikly z limitů, které mají SQL RDBMS databáze. noSQL databáze jsou zaměřené na rychlé zpracování velkých dat všech struktur nejčastěji s tzv. Big daty. To zvládají, protože mohou být postavené na distribuovaných systémech, které pomáhají s rozložením zátěže dat v sítích, a z toho vyplývá, že jsou snadno škálovatelné a lze snadno zvyšovat výpočetní výkon, a také, že fungují na principu Key-Value struktury. noSQL jsou založené na BASE modelu dat. (Ncache, 2020) (Pivert, 2018)

5.1 Typy noSQL databází a zástupci

Key-Value Stores

Jedná se o nejjednodušší druh noSQL databází. Tato databáze implementuje hashovací tabulku, do které se ukládají jedinečné klíče (atributy) spolu s ukazateli tzv. pointery na odpovídající hodnoty dat (Value), například Key: City, Value: Prague. Tyto hodnoty mohou být skalární (číselné), binární, textové, ale i složitější struktury, jako je formát JSON. Poskytuje vysoký výkon a snadnou škálovatelnost a nízké náklady na implementaci. Pro manipulaci s databází se využívá příslušné API. Tento druh databáze se hodí pro aplikace mající jednoduchý datový model a nevyžadující časté aktualizace. Příklady Key-Value databází jsou Redis, Riak, Amazon DynamoDB.

Redis

Je open source s licencí BSD založený na Key-Value databázích. Díky své jednoduchosti a rychlosti vycházející z povahy Key-Value databází, viz výše, se nejčastěji hodí a používá jako cache (vyrovnávací paměť), nebo jako message broker používající v IoT pro přeposílání messages (asynchronní data). Podporuje celou řadu datových struktur, jako jsou řetězce, hashe, seznamy a další. Podporuje celou řadu programovacích nástrojů a je multiplatformní, i když přímo Redis doporučuje využívat platformu Linux (Redis 3). Je vhodný a snadno spustitelný na nízkonákladovém hardwaru. Uživatelé používají rozhraní API pro práci s databází, jako je vkládání (příkaz put KEY), mazání (delete KEY), zápis a čtení dat (execute, get KEY), se

kterým je na jednu stranu snadná práce, ale na druhou stranu značně omezena, co se týče možností. Společnosti využívající Redis jsou Twitter, GitHub, Snapchat, StackOverflow, Flickr (Redis 2). Na trhu je, mimo jiné, i komerční distribuce Redis Enterprise s uzavřeným kódem, který rozšiřuje Redis o mnoho funkcí a je například plně ACID kompatibilní, nasaditelný v cloudu. (Redislabs, 2020)

Document Stores

Ve své podstatě tento typ noSQL databází je podobný Key-Value databázím. Opět je zde klíč a k němu odpovídající hodnota, ale s rozdílem, že tyto klíče a hodnoty jsou strukturovány do dokumentu a uloženy v nějakém formátu (XML, JSON, BSON, YAML, PDF). Dokument je tedy objekt obsahující metadata atributů (Key) a zadanou hodnotou (Value), jako je řetězec, pole, popřípadě vnořený dokument. Implementuje transakce ACID, protože atomicita je zde podmíněna. Dále umožňuje indexovat dokument na základě primárního klíče a následně dotazovat. Tyto dokumenty mohou být seskupeny do kontejnerů, nebo též zvaných kolekcí. Opět mezi výhody patří snadná škálovatelnost. Dotazovací jazyky jsou dostatečně flexibilní. Toto řešení není vhodné, pokud aplikace vyžadují složité transakce s více operacemi. Kvůli dokumentové struktuře můžou vznikat duplicity ve větší míře než například u RDBMS. Patří sem XML databáze, MongoDB, CouchDB.

MongoDB

Je populární multiplatformní databáze postavená na Document Store databázích. Vznikla v roce 2009 s aktuální verzí 4.2.0 (9. 8. 2019) a je zdarma k použití. Ukládá do dokumentů typu JSON, BSON. MongoDB je distribuovaná ve svém jádru, nabízí vysokou dostupnost, rychlost, horizontální škálování, podporuje dotazování skrze JavaScript. I MongoDB má limity, mezi které patří například vysoké využívání paměti, omezení velikosti dokumentu dat a omezení vnořených dokumentů. MongoDB Enterprise advanced je komerční verze. Organizace využívající tuto databázi jsou například BOSCH, SEGA, Adobe, Cisco, Google, Facebook. (MongoDB 1, 2020) (Dataflair, 2018)

XML

XML databáze jsou dva základní typy. XML enabled, kde se jedná o takovou databázi, která pro ukládání může používat souborový systém, relační nebo objektovou databázi. Další typ je NXD (Native XML), která využívá nativní velice univerzální XML formát souborů. Typický jazyk pro dotazování v XML je XQuery. Hlavní výhoda je univerzální formát XML. Nejčastěji se tyto databáze používají, pokud získáváme data přímo ve formátu XML. Bohužel nedosahují

robustnosti, jako jsou transakce, škálovatelnost relačních systémů. Chybí plná řada standardů. Proto XML databáze se nestanou následníkem relačních databází, ale tvoří jakousi alternativu vhodnou pouze pro určité typy dat a aplikací. (Tutorialspoint 1, 2020)

Wide Column Store/Column Families

Jedná se o sloupcové databáze, které ukládají tabulky jako oddíly sloupců, nikoliv na řádky, jako tomu je u RDBMS. Každý sloupec může, ale nemusí, obsahovat hodnotu pro každý řádek. Tyto sloupce jsou uloženy v tzv. keyspace. Obsahuje všechny column families (rodiny sloupců). Ty obsahují Row key (řádkový klíč), který slouží jako jedinečný identifikátor pro daný řádek. Sloupec obsahuje název, hodnotu a časové razítko. Časová známka slouží k určení nejnovější verze dat. Za výhody se považuje efektivní komprese dat. Díky sloupcové struktuře fungují dobře agregační funkce. Jsou dobře škálovatelné. Hodí se k velkému paralelnímu zpracování dat do velké skupiny strojů. Díky rychlému čtení jsou vhodné pro organizace zabývající se Big daty. Nevýhody jsou, že obecně neexistují transakce a mají řadu omezení pro vývojáře, kteří jsou zvyklí na RDBMS. Zápis je oproti čtení o dost pomalejší než čtení. Patří sem HBase, Cassandra. (Olivera, 2019)

Apache HBase

Je distribuovaná sloupcová databáze postavená na systému Hadoop, konkrétně na HDFS. Opět je to open source napsaná v jazyce Java a vznikla v roce 2008. Ukládá Big data do HDFS souborů. HBase podporuje rychlé vyhledávání se zaměřením na nízkou latenci a automatickou podporu při selhání nebo konzistentní dobu zápisu a čtení. Klient využívá Java API pro práci s ní. Mezi nevýhody patří vysoká náročnost na CPU a paměť. Organizace využívající HBase jsou například Yahoo, Adobe, Twitter. (Hbase, 2020) (Tutorialspoint 2, 2020)

Apache Cassandra

Je opět, jako HBase, open source sloupcová databáze napsána v jazyce Java. Projekt vytvořili inženýři ze společnosti Facebook v roce 2008 a byl původně navržen, tak aby uživatelům Facebooku poskytoval rychlé vyhledávání obsahu v konverzacích. V roce 2010 se Cassandra stala součástí projektu Apache. Dotazovacím jazykem je CQL (Cassandra Query Language), který je velmi podobný jazyku SQL, ale není s ním kompatibilní. Využíváním projektu Cassandra se mohou pyšnit organizace jako CERN, eBay, GitHub, Instagram, Netflix. Cassandra je mimo jiné populární jako cloudové řešení. Za zmínku stojí i distribuce DataStax Enterprise postavené na Apache Cassandra, která je pro tyto cloudové účely navržena. Za nevýhody se dají považovat opět vysoké nároky na CPU a paměť. Cassandra dokáže, stejně

jako HBase, intergrovat Hadoop, ale dokáže existovat i bez něho jako samostatná databáze, což je velkým rozdílem oproti HBase databází. (Bártík, 2013) (Intellipaat, 2016)

Graph Database

Fungují na základě matematické teorie grafů. Jedná se o kolekci vztahů, kde každý uzel (node) je entitou obsahující atributy, které jsou reprezentovány jako Key-Value. Tyto uzly jsou propojeny vztahy, které jsou označovány jako hrany (edge). Každý tato hrana má vždy počáteční a koncový uzel za pomocí jedinečných identifikátorů. Uzle nelze odstranit, aniž by se odstranily přidružené uzle. Tyto databáze poskytují tzv. bezindexovou sousednost. To znamená, že každý uzel obsahuje přímý ukazatel na sousední uzel, aniž by se nutně muselo vyhledávat přes indexy, ale poskytuje možnost indexace. Vztahy mohou být obousměrné a může jich být více mezi danými uzly. Grafové databáze se používají nejčastěji u analytických úloh nebo obecně pro grafická data. Olivera uvádí příklady, jako jsou konstrukce kosmických raket, dopravní systém, detekce podvodů, síťové IT operace. Poskytuje opět snadnou škálovatelnost, ale tyto databáze bývají náročné na vysoký výkon. Další problém přichází z hlediska nedostatku standardních jazyků, jako je například jazyk Cypher. Patří sem databáze Neo4J. (Pore, 2018) (Foote, 2018) (Olivera, 2019) (Vasilies, 2013)

Neo4J

Je open source noSQL grafová databáze, která vznikla v roce 2003, ale veřejně dostupná je od roku 2007. Byla napsána v jazyce Java a Scala. Je k dispozici ve dvou edicích Community nebo Enterprise. Enterprise je určena především pro podnikové činnosti. V zásadě obsahuje vše, co Community, ale navíc je obohacena o zálohování nebo clustering. Neo4J používá nativní grafovou databázi a podporuje ACID transakce. Dotazovacím jazykem je Cypher. Nejčastěji je používaná v průmyslových a finančních odvětví. (Neo4j, 2020)

5.2 Související koncepty – Big data, moderní trendy

Škálovatelnost

Je schopností systému navyšovat neboli „škálovat“ výpočetní výkon. Slouží k efektivnějšímu a rychlejšímu zpracovávání rostoucích dat. Protože množství dat roste, je potřeba dané databázové systémy vylepšovat, a škálovatelnost se tak stává zároveň důležitým parametrem databázových systémů na trhu. Existují dva typy škálovatelnosti: vertikální a horizontální, popřípadě je lze kombinovat. Každý z nich má své výhody a nevýhody, které budou následně rozebrány spolu s jejich principy.

Vertikální škálovatelnost je typická především pro RDBMS a zároveň je nejstarším typem. Ve své podstatě jde o zvyšování výkonu daného serveru pomocí obměny nebo přidáním hardwaru, jako je například výkonnější CPU, větší paměť atd. Toto řešení má však své problémy. Největším z nich je technologické omezení, protože nelze neustále přidávat hardware navyšující výkon serverů.

Horizontální škálovatelnost spočívá v přidáváním dalších serveru (uzlů, výpočetních kapacit) do výpočetního systému a pomáhá tak rozložit zátěž. Toto řešení je oproti vertikálnímu tedy efektivnější z hlediska zmíněného omezení výkonu, protože teoreticky lze přidávat servery do nekonečna. Problémem je zvýšená náročnost pro vývojáře aplikací, kteří musí zajistit paralelní provoz a zpracovávání úloh, což je z hlediska znalostí a doby testování obtížnější a finančně nákladnější. Dalším problémem je celková správa propojených serverů a často vzniklé problémy například latencí (stabilitou připojení k síti). Toto škálování je na trhu velice populární a je typickým škálováním pro noSQL databázové systémy. (Mullins, 2018) (Špoljarič, 2019)

JSON a BSON

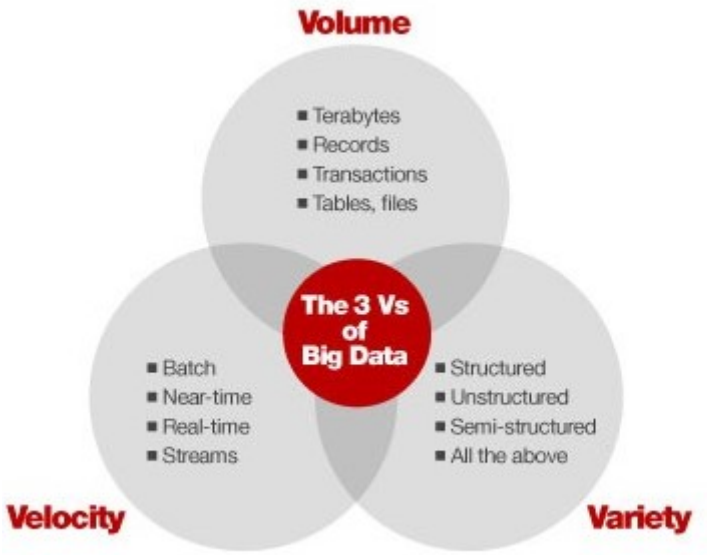
JSON (JavaScript Object Notation) je člověkem dobře čitelný i strojově generovatelný nezávislý datový formát určený pro přenos dat. JSON spolu s XML formátem patří mezi nejpoužívanější v dnešním webovém průmyslu. Nejčastěji je strukturován jako asociativní pole[1], neboli též kolekce popisující objekt. Tento objekt je typicky uvozen párovými složenými závorkami skládající se klíčových páru Key-Value oddělené mezi sebou dvojtečkou. Jednotlivé záznamy jsou odděleny čárkou. BSON (Binary JSON) je binárně kódovaný JSON formát. Každému atributu připadá vhodný datový typ. (MongoDB 2, 2020) (Json)

Big data

Big data, česky „veledata“, jsou velká data v řádech terabytu, či petabytu. Zahrnují, jak strukturovaná, tak především ve větší míře nestrukturovaná data. Neexistuje žádná přesná definice, proto se nejčastěji charakterizuje třemi vlastnostmi tzv. 3V: Volume (objem dat), Velocity (rychlost, kterou data přicházejí), Variety (různorodost dat). Obecně tedy platí, čím více rostou tyto 3V, tím více je Big dat. Čím rychleji data přicházejí a čím více různorodější jsou, tím mají větší celkový objem. Cílem je nasbírat co nejvíce dat za krátkou dobu a uložit je. Právě noSQL databáze jsou vhodné pro sběr Big dat z důvodu rychlého ukládání

[1] asociativní pole – datová struktura skládající se z dvojic

nestrukturovaných dat, kdežto RDBMS jsou kvůli zmíněné povaze dat nevhodné. Alternativou je Warehousing, který bude zmíněn a porovnán s Big daty v následujíc kapitole 6. Shrnutí teorie.



Obrázek 6 - Souhrn 3V

Big data jsou všechna data, která zanechávají jakoukoliv digitální stopu (sociální sítě: tweety, příspěvky, komentáře, finanční transakce, maily, zvukové a video soubory, digitalizované dokumenty, obecně cokoliv online).

Big data jsou nejčastěji využívána k analýze dat na trhu nebo se sbírají pro pozdější využití. Díky dobré analýze pak firma může vydávat různé produkty na míru podle zjištěné poptávky a snižovat náklady a zvyšovat efektivitu, popřípadě zaměřit marketing určitým směrem. (Černý, 2013) (Krčmář, 2016)

Apache Hadoop

Je open source projekt (framework) pro práci s Big daty, ale nejedná se o databázi. Ústřední technologie jsou HDFS (Hadoop Distributed File System) a YARN (Yet Another Resource Negotiator). HDFS je tedy distribuovaný souborový systém (běží na clusteru), který dovoluje ukládat velké množství dat. Dále je zaměřen na vysokou odolnost vůči chybám a vhodný pro nízkonákladový hardware. YARN je technologie správy zdrojů a plánování úloh. Ta se pak stará o přidělování systémových prostředků různým aplikacím běžícím v clusteru. (Hadoop, 2019) (Vitfo, 2018)

IoT

IoT (Internet of Things) česky internet věcí, je trend z oblasti kontroly a komunikace předmětů běžného využití mezi sebou nebo s člověkem, a to zejména prostřednictvím bezdrátových technologii přenosů dat a Internetem (Iot, 2020). Vzniklé projekty mohou být například dálkově ovládané spotřebiče nebo komponenty, jako jsou zásuvky, osvětlení, měřící sensory, kamery atd. Kromě domácího využívání se IoT používá v logistice, zdravotnictví, dopravě, meteorologii atd. I IoT projekty musí ukládat data. Ty se ukládají na souborový systém, do cloudu, nebo dokonce do databáze. IoT spolehá na rychlost a pokud to jde, tak i na to, aby dané provedení bylo co nejlevnější.

Cloudová databáze

Je zde zmíněna především jako moderní trend dnešní doby. V podstatě se jedná většinou o SQL RDBMS a noSQL databáze běžící v cloudu. Definice cloudu opět není příliš jednotná a zřetelná. Dá se říct, že „cloud je rozsáhlá síť vzájemně propojených vzdálených serverů po celém světě tvořící jeden ekosystém“. Databáze může v cloudu běžet jako tzv. tradiční databáze, kde si společnost pronajme prostor (virtuální server běžící v cloudu), nebo databáze jako služba DBaaS (Database-as-a-Service), kterou cloudový poskytovatel nabízí za úplatu, a tu uživatelé spravují pomocí API nebo administrátorského rozhraní. DBaaS řešení poskytuje celou řadu služeb včetně optimalizace v podobě automatizace softwaru, zálohování atd.

Hlavními výhodami cloudového řešení je především dobrá dostupnost, rychlost, a zároveň i bezpečnost, a to, jak z hlediska výpadku, tak i z hlediska bezpečnosti dat. O ty se uživatel nemusí starat a zajišťuje je přímo cloudový poskytovatel. Cloud je na jednu stranu za úplatu snadno škálovatelný, ale na druhou stranu může být poskytovateli záměrně výkonnostně omezen, což se v konečné fázi může projevit na vyšších nákladech. Nevýhodou může být i samotný fakt, že data nejsou u společnosti, ale u poskytovatele cloudových služeb v datacentru. Mezi příklady cloudu patří Azure od Microsoftu, AWS od Amazonu, GCP od Googlu, nebo Oracle cloud. (Oracle 1, 2020) (Azure 1, 2020)

6 Shrnutí teorie

Cílem této kapitoly je shrnutí SQL a noSQL databází na teoretické úrovni a vzájemně je porovnat z hlediska vhodnosti, limitů a principů. Závěr kapitoly bude věnován Warehousingu.

6.1 SQL databáze

Za SQL databázemi stojí pevný teoretický základ vybudovaný na relační algebře a plná standardizace. Používají se více než 40 let, a tak jsou dobře kompatibilní se staršími systémy. Data jsou strukturována do tabulek tvořící entity, sloupce tvořící atributy a řádky pak záznamy. Fungují centralizovaně na modelu klient-server a jsou pouze vertikálně škálovatelné. To znamená, že lze navyšovat výpočetní výkon stroje, což může být dost nákladné. Dotazovacím jazykem je SQL, který je flexibilní a jelikož existuje stejně dlouhou dobu, jako samotné databáze, tak se dostal dostatečně do povědomí vývojářů.

Hodí se pro takové systémy, které vyžadují logické nebo diskrétní zpracování dat oproti noSQL. Používají se například pro účetní, bankovní systémy, či obecně pro takové systémy, které vyžadují složitou manipulaci s daty, či nutnosti relací. Samotné navržení databázového schématu bývá náročnější kvůli určitým zásadám, jako je normalizace sloužící k zamezení redundance dat, správné nastavení klíčů, znalost toku dat v daném modelu. Data jsou tedy závislá na schématu, a to je potřeba neustále udržovat a aktualizovat mezi aplikací a databází. Transakčním modelem je ACID. Ukázka porovnání viz Příloha 1 – Porovnání MongoDB a MySQL.

Limity a nevýhody

- Možnost pouze vertikálního škálování, což má důsledek omezení výkonu a finanční náročnost.
- Složitý návrh databázového schématu.
- Nevhodné pro Big data.

Výhody

- Plně standardizovaný.
- Vhodný pro systémy, vyžadující složitější strukturu dat.
- Na trhu zaběhnutý řadu let.

6.2 noSQL databáze a jejich rozdíly

Je souhrnem databázových systémů, které nejsou relační, nebo nepoužívají dotazovací jazyk SQL. Jsou rozdělené do několika kategorií: Key-Value, Document, Column a Graph based. Chybí velké množství standardů, a tak každý produkt dané databázové kategorie může fungovat odlišným způsobem, což může být náročné, jak na naučení s daným systémem, tak na přecházení více noSQL databázovými systémy. To samé platí i u dotazovacích jazyků, které bývají různé v závislosti na kategorii, ale nejčastěji se jedná o API, což na jednu stranu poskytuje třeba snazší práci než s SQL, ale na druhou stranu může být práce omezena. Vesměs se jedná o open source projekty. Oproti SQL databázím jsou mladší, využívají se zhruba posledních deset let, a tak povědomí o nich je menší. Zároveň nejsou kompatibilní se staršími systémy.

Data jsou strukturovaná nejčastěji ve stylu Key-Value, u kterých odpovídá danému klíči hodnota, či pole hodnot. Nejčastěji noSQL databáze využívají formát JSON, který je populární při práci s webovými technologiemi a tzv. microservisovou architekturou.

Poskytují vertikální i horizontální škálování, což teoreticky umožňuje neomezeně zvyšovat výpočetní výkon. Horizontální škálování je možné díky povaze distribučních systémů, které noSQL databáze poskytují. Benefity přinášející zmíněné distribuční systémy jsou například vysoká dostupnost a dobrá latence. Z důvodu využívání distribučních systémů není zcela vhodné využívat transakční model ACID, a tak tyto databáze využívají model BASE založený na poznatcích ohledně dostupnosti dat z CAP teorému. Hlavním rozdílem je konzistentnost dat, která je zde řešena na úrovní aplikace, nikoliv databáze.

noSQL databáze jsou určeny pro rychlé zpracování velkého objemu dat, kterých dosahuje díky dvěma poznatkům. První poznatek je struktura Key-Value, která funguje rychleji než relační, jelikož neobsahuje relace, ale zároveň není vhodná pro složitější systémy, které je vyžadují z logického a funkčního hlediska. Druhý poznatek je vysoká dostupnost a škálování díky distribučním systémům. Tak jsou noSQL oblíbená v IoT, analýzy dat na trhu v reálném čase, kde se víceméně rychle zpracovávají data a nijak se s nimi nemanipuluje. Data jsou nezávislá na schématu aplikace, což činí návrh databázového schématu snadnějším než u SQL databází.

Limity a nevýhody

- Nevhodný pro systémy požadující složitou manipulaci s daty.
- Chybí podpora standardů.
- Problémy vzniklé důsledkem distribučních systémů.
- Oproti SQL není moc zaběhnutý.

Výhody

- Možnost vertikálního i horizontálního škálování.
- Vysoká dostupnost, latence.
- Vhodný pro Big data.
- Open source projekty.
- Snadný návrh databáze.

  (Azure 2, 2020) (Perry, 2018) (Krisciunas, 2014)

Warehousing a Big data

Data Warehouse též WMS (Warehouse Management System) česky „datový sklad“ je systém správy dat. Cílem Warehousingu je shromáždění velkého množství historických dat, které se pohybuje v řádech terabytu určené především pro analýzu dat na trhu. Je to obdobný pojem, jako jsou Big data, která jsou vysvětlena v kapitole 5.2 Související koncepty – Big data, moderní trendy. Hlavním rozdílem je, že Warehousing využívá RDBMS databáze. Na vstupu je opět velké množství různorodých dat, která se následně transformují do jednotného schématu tak, že se vyfiltrují pouze podstatná data, a ta se následně uloží do tabulek. Celá tato operace, která se provádí, je výkonnostně náročná, a tak je rozdělena do jednotlivých částí, konkrétně úrovní tzv. tieru, které jsou jednotlivě standardizovány. Warehousing mimo jiné využívá technologie OLAP (Online Analytical Processing) a OLTP (Online Transaction Processing), což jsou technologie, které ukládají tato velká data (podléhají NF) a analyzují, či vyhodnocují. (Lange, 2019) (Blackbaund) (Oracle 2, 2020) (Mendelu 3)

7 Praktická část

Cílem praktické části je vytvořit aplikaci za účelem srovnání SQL a noSQL databázových přístupů a zároveň ověřit jejich teoretické aspekty, o kterých hovoří teoretická část práce. Nejprve se představí cíl aplikace spolu s výběrem konkrétních technologii včetně představení aplikace. Pozornost bude věnována zejména implementaci s následným srovnáním obou variant zakončené subjektivním zhodnocením vyvíjené aplikace.

7.1 Představení cíle

Bude se jednat o webovou aplikaci, jejíž tzv. frontendová část bude totožná, a tzv. backendová část bude odlišná v závislosti na použité technologii pro obě varianty. Pod pojmem frontend se rozumí ta část aplikace, která je viditelná pro klienta a obsahuje veškeré uživatelské rozhraní. Klientem může být například uživatelem používaný webový prohlížeč. Z toho lze vyvodit, že klient je uživatelem aplikace a uživatelské rozhraní slouží pro manipulaci a orientaci v aplikaci. Backend je ta část aplikace, která není viditelná pro klienta a běží na serveru. Obsahuje programovatelnou logiku aplikace, přístup a manipulaci s datovou částí například databází v dané aplikaci. V tomto případě se bude jednat o SQL a noSQL databázi. Stručně řečeno, backend obsahuje vše programovatelné na straně serveru, které se následně po vykonání dané činnosti pošle na frontend. Nicméně lze podotknout, že programovat lze i na straně klienta (frontendu) pomocí JavaScriptu. (Dostalová, 2014)

7.2 Výběr konkrétních technologii pro realizaci

Frontendová část bude využívat technologie HTML5, CSS styly, Bootstrap, JavaScript. Backendová část bude napsána pro oba případy databází v jazyce PHP, ačkoliv se v praxi více používají JavaScriptové frameworky pro práci s noSQL databázemi, které jsou často společně realizovány s tzv. mikroservisovou architekturou webu. Důvodem je vesměs trend vývoje webových aplikací v současné době, jako je snaha co nejvíce obsahu generovat na straně klienta pomocí JavaScriptu. Zmíněné mikroservisy jsou ve své podstatě autonomně běžící nezávislé služby. Backendem jsou samotné mikroservisy a jejich princip spočívá v tom, že například na určité adrese a pod určitým portem je k dispozici JSON string. Ten je následně zpracován na straně klienta JavaScriptem, bez nutnosti přímého přístupu k serveru, na kterém jsou data. Nicméně i PHP s patřičným driverem dokáže pracovat s noSQL databází, a tak pro tento případ byl vybrán, jak z důvodu osobní preference, tak z důvodu názorného porovnání, jelikož jej budou využívat obě varianty.

Za zástupce SQL databáze byla vybrána distribuce MariaDB z důvodu nativní dostupnosti v každé linuxové distribuci. Co se týče noSQL databází, byla vybrána distribuce MongoDB z několika důvodů, jako je názorně a viditelně odlišná dokumentová struktura oproti relačním systémům nebo přívětivý dotazovací jazyk.

Node-RED je open source programovací nástroj napsaný v JavaScriptu a běžící na platformě NodeJS. Používá se k programování jednotlivých vstupních a výstupních asynchronních komunikací. Uživatel má k dispozici grafické rozhraní, ve kterém programuje jednotlivé nody. Node-RED je často používán v IoT odvětví.

7.3 Představení hotové aplikace

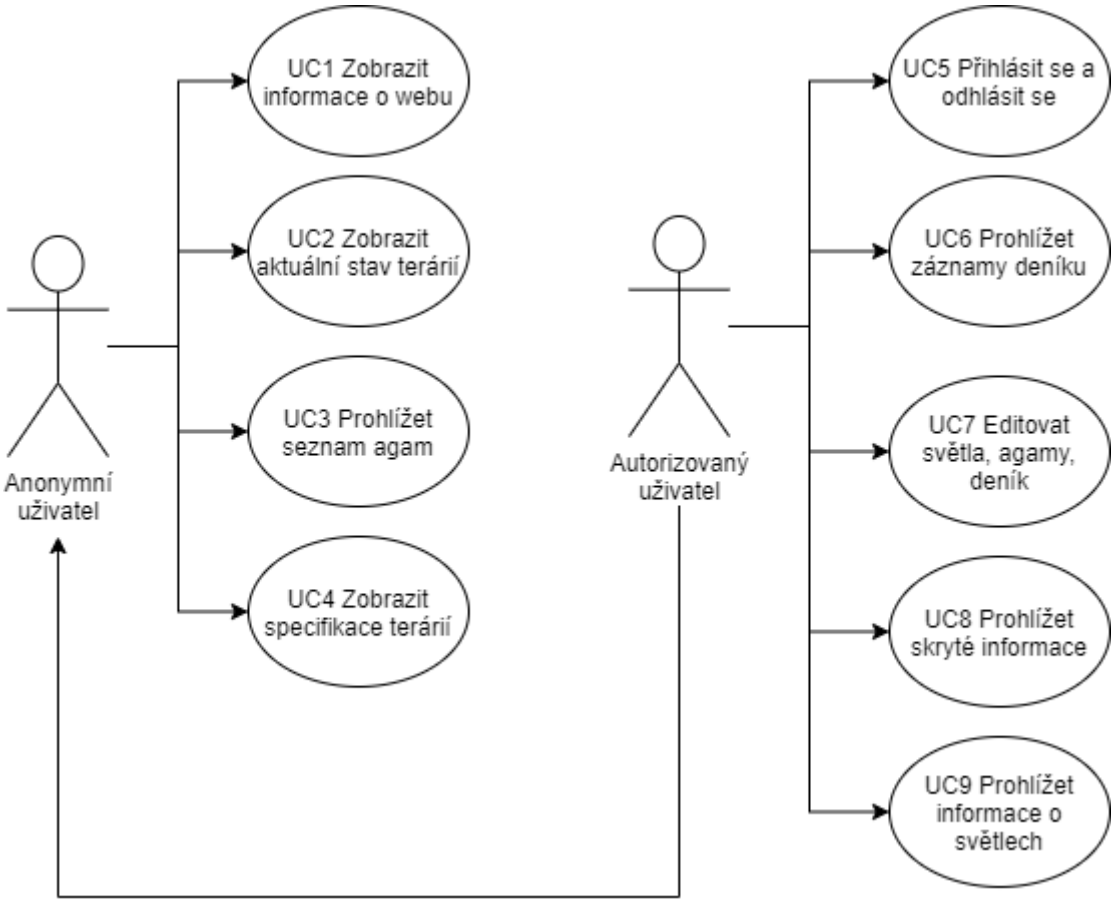
Aplikace je rozdělena do dvou webových projektů. Pro variantu s MongoDB je přístupná na adrese: http://debbie.westwinder.eu/mongodb a pro variantu s MariaDB na adrese: http://debbie.westwinder.eu/mariadb. Ukázky z výsledné aplikace viz Přílohy 2, 3, 4, 5.

7.3.1 Popis aplikace

Webová aplikace slouží jako informační web chovatele agam. Má veřejnou část obsahující informace o webu, jednotlivých terárií, včetně technických parametrů, a seznamu agam s jejich popisem. Dále je k dispozici tabulka „Aktuální stav terária“ s aktuálně naměřenými hodnotami v jednotlivých teráriích (teplota, vlhkost, tlak). Soukromá část webu obsahuje funkcionalitu deníku, kde si autorizovaný uživatel může za pomoci formuláře evidovat každodenní záznamy týkající se jednotlivých agam a jejich denních potřeb. Jedná se například o krmení, vylučování a následně celkový výpis těchto záznamů. Autorizovaný uživatel má nadále informace o světlech v teráriích, které je nutné pravidelně měnit. Tyto informace může editovat, včetně informací o aktuálním stavu agam, jako je jejich aktuální váha a délka.

Use Case diagram

V češtině diagram využití je druh UML diagramu (Unified Modeling Language), používající se při návrhu aplikace. Principem je zobrazit chování systému, jak jej vidí uživatelé, a popsat danou funkcionalitu systému. Diagram říká pouze, co systém má umět, ale neříká, jak to bude dělat. K tomu slouží technický popis. Diagram se skládá z aktéra popisující uživatelé a jednotlivých use case neboli funkcionalit, a vztahy mezi nimi. Ve své podstatě se jedná o grafické zobrazení popisu aplikace. (Čápka, 2020)



Obrázek 7 - UC diagram

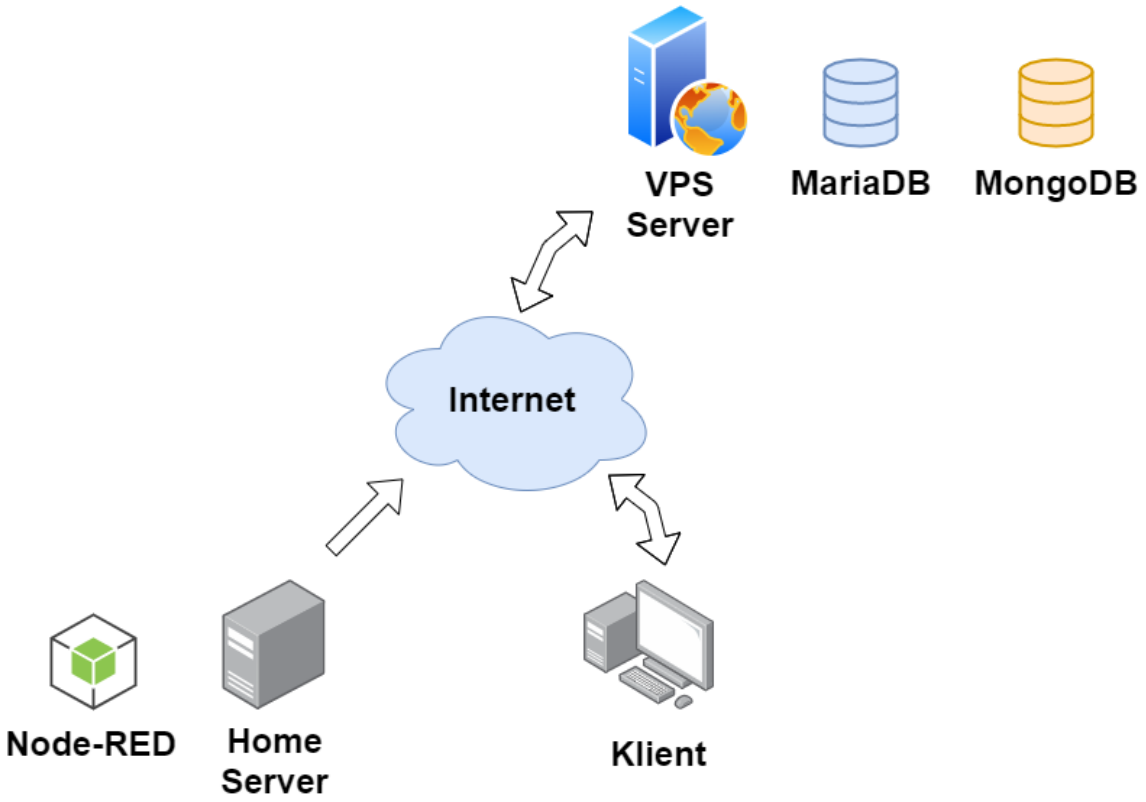
7.3.2 Technický popis

Webové projekty se liší v backendové části, kde se jednou manipuluje s SQL databází MariaDB a podruhé s noSQL databází MongoDB, a dále se liší v samotné implementaci, respektive návrhu a realizaci konkrétní databáze. Všechen informační obsah je načítán z databáze kromě informací o webu a autorizace, které jsou statické. Tabulka aktuální stav terária je získávána z časově pevně pořízených snapshotů s omezenou životností v databázi. Snapshoty jsou tvořeny

naměřenými daty získaných ze senzorů vložených v jednotlivých terárií. Naměřená data jsou zpracována a následně uložena do jednotlivých databází v prostředí Node-RED. Autorizovaný uživatel může mít k dispozici více informací k zobrazení, jako je například stav baterie v senzorech.

7.4 Návrh a implementace

Následující schéma zobrazuje, jak je daná aplikace zapojena z hlediska rozložení v počítačové sítí. Skládá se ze dvou serverů. První je Home server, na kterém běží aplikace Node-RED, který není dostupný z Internetu, a ten následné posílá data na druhý server VPS (Virtual Private Server). Jedná se o plnohodnotné servery, které jsou poskytovateli pronajímány za úplatu, a jsou dostupné z Internetu. Na této VPS běží webový server a dvě databáze. Klient je jakýkoliv uživatel přistupující na webovou stránku.

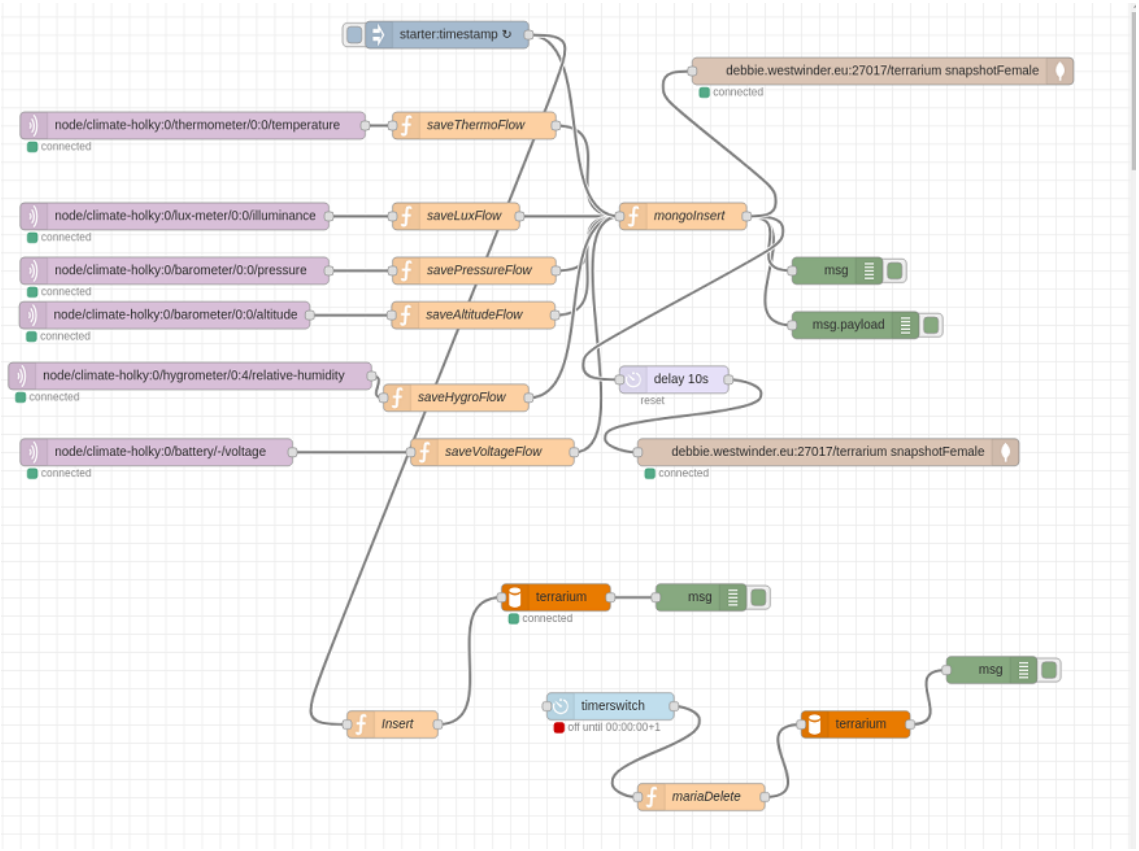


Obrázek 8 - Schéma zapojení aplikace v síti Internet

Node-RED

Následující schéma demonstruje logiku zapojení senzorů a zpracování naměřených dat. Schéma se skládá z fialových nodů symbolizující konkrétní senzory. Béžové nody jsou funkce, které ukládají zachycená data senzorem. Ty jsou následně hromadně vkládány do jednotlivých

databází. Databázové nody obsahují konfiguraci pro připojení ke konkrétní databáze. Hnědé nody symbolizují databázi Mongo a oranžové Marii.



Obrázek 9 - Schéma zapojení v aplikaci Node-RED

7.4.1 Varianta s relační databází

Relační databáze neboli SQL databáze vyžadují při návrhu pečlivé rozvržení tabulek, tak aby alespoň splňovala 3NF. Pro návrh databázového schématu se nejčastěji používá konceptuální, logické a fyzické modelování.

Konceptuální modelování

Do konceptuálního modelování spadá ER (Entity Relationship) diagram. Jeho principem je popsání reality za pomocí množin objektů zvaných entity a vztahů mezi nimi, které se nazývají relationship. Velký důraz je dán na objektovou analýzu a schopnost aplikovat danou úlohu do reálného světa. V následujícím ER diagramu se vyskytují čtyři entity (Deník, Agama, Terárium, Snapshot, Světlo) popisující reálný objekt. Každá entita obsahuje atributy popisující vlastnosti entit včetně klíčového atributu. ER diagram se může lišit od fyzického modelování.

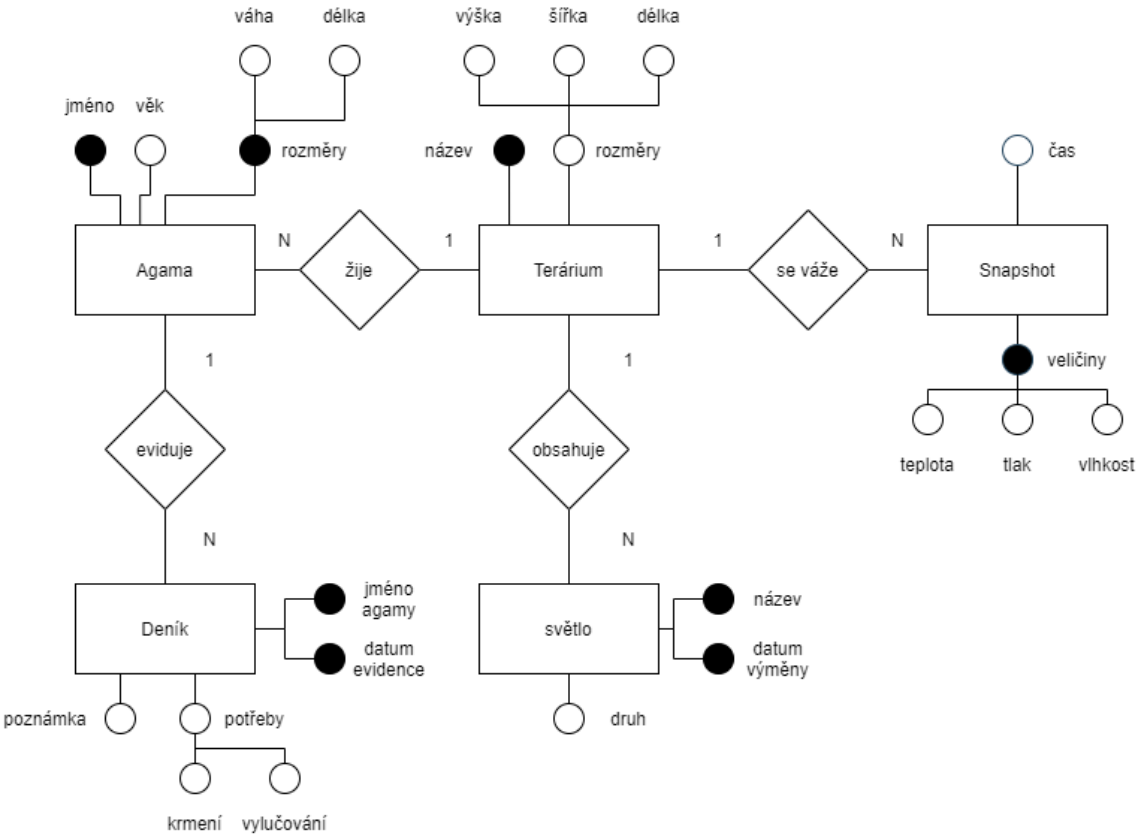
Objektová analýza

Entita agama obsahuje atributy jméno, věk a složený atribut rozměry. Unikátním identifikátorem je složený primární klíč skládající se z atributu jméno a rozměry identifikující konkrétní agamu. Zvolit například pouze jméno jako identifikátor není vhodné v případě, že by existovaly dvě agamy se stejným jménem, pak by nebylo možné jednoznačně oddělit stejnojmenné agamy. Zato kombinace jména a rozměrů agam jsou lepším identifikátorem, protože je velice malá pravděpodobnost shody stejnojmenné agamy se stejnými rozměry. Ještě se nabízí možnost zvolit klíčový atribut rozměry, jako unikátní identifikátor, což dle mého názoru není špatná úvaha. Je ale příliš konkrétní, a tak se osobně klaním identifikovat agamu jménem, jakožto první identifikátor, a v případě shody jména dále atributem rozměry. Na rozdíl od entity Terárium, kde je jako unikátní identifikátor zvolen pouze název. Terária jsou pojmenována podle místností. Je to zvoleno z důvodu konvence pojmenování terária, neboť se v aktuální dispozici domu nepočítá s tím, že by se nacházela dvě stejnojmenná terária v jedné místnosti. Je vždy potřeba aplikovat tuto abstrakci na konkrétní příklad. Kardinalita vztahu bude vysvětlena u konkrétního modelu.

Relace

Agama žije v jednom teráriu a v jednom teráriu může žít více agam. K teráriu se váže více pořízených snapshotů a jeden snapshot se váže k jednomu teráriu. Každé terárium obsahuje více druhů světel. Agama eviduje více záznamů v deníku a jeden záznam je evidován pro jednu agamu. Alternativně by šlo uvažovat o parcialitě. Může agama existovat bez terária a může terárium existovat bez agamy?

Je důležité podotknout, že se jedná pouze o abstrakci a výsledné fyzické schéma se může lišit i z důvodu technického provedení. Například v praxi je pohodlnější využívat umělé identifikační klíče, nicméně v ER diagramech není vhodné uvažovat o umělém klíči kvůli ztrátě abstrakce, jelikož snaha konceptuálního modelování je popsat reálnou věc. Nadále nelze říct, že vždy jedno řešení je správně, vždy záleží na konkrétní aplikaci.



Obrázek 10 - Ukázka chen ER diagramu

Fyzický model

Popisuje ER diagram v praxi. Již se nejedná pouze o abstrakci, ale o konkrétní fyzickou implementaci. Entita je název tabulky a atributy jsou sloupce tabulky, které navíc obsahují datový typ. Zvolení správného datového typu má vliv na velikost daného záznamu. Vztahy ve fyzickém modelu vznikají pomocí primárních klíčů odkazující na cizí klíče. Například vyjádření vztahu 1:N. Pro jeden záznam s jedním primárním klíčem existuje více záznamů s různými cizími klíči v odkazované tabulce.

Model 1:N

Primární klíče jsou umělé identifikátory pojmenované „idNázevTabulky“. Cizí klíče jsou stejnojmenné atributy v tabulce, na které odkazuje konkrétní primární klíč. Na jeden záznam v tabulce Dragon se váže více záznamů v tabulce Diary. Z toho vyplývá vztah 1:N vyjádřený pomocí klíčů následovně. Primární klíč idDragon v tabulce Dragon odkazuje na stejnojmenný cizí klíč v tabulce Diary a DragonSize. Jeden záznam v tabulce Terrarium se váže na více záznamů v tabulkách Dragon, Light a Snapshot pomocí odkazujících klíčů idTerrarium. V

konceptuálním schématu je druh světla uveden jako atribut, nicméně ve fyzickém modelu se nakonec druh světla stal samostatnou tabulkou TypeLight. Je to z důvodu snadného potenciálního přidávání jednotlivých druhů světel v budoucnosti. To samé platí pro tabulku DragonSize, která se také stala samostatnou tabulkou, protože její atributy jsou proměnlivé.

Ke každému atributu náleží datový typ určující povahu dat zaznamenaných ve sloupci. V následujícím schématu se vyskytují následující datové typy: INT (celočíselné číslo), TINYINT(dvojice čísel nabývají hodnot 0 a 1), FLOAT (desetinné číslo), VARCHAR (řetězec), CHAR (jeden znak) LONGTEXT (dlouhý text), TIMESTAMP (formát data, časové známky).

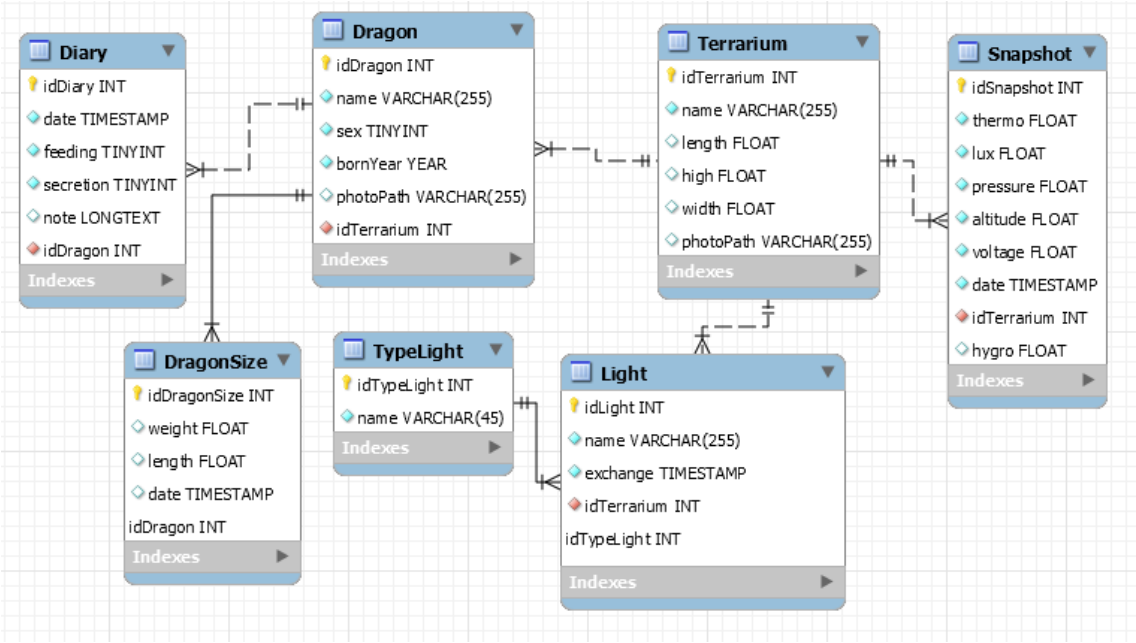
TINYINT a CHAR

U atributů sex, feeding, secretion, name z tabulky TypeLight je možné vybrat datový typ TINYINT nebo VARCHAR, či CHAR. V případě TINYINT by se ukládaly dvě hodnoty 0,1 a na úrovni aplikace by se musel zajistit překlad, který je snadněji pochopitelný pro člověka, například pro 0 - ne a pro 1 - ano. V případě zvolení CHAR (1) lze ukládat jeden znak například „y“. Toto vyjádření je pro člověka přirozenější, nicméně je potřeba ošetřit formát vstupních dat, jako je například počet znaků. Případně přímo vymezit tvar vstupního řetězce, či ošetřit velikost písmen, jestli budou písmena malá nebo velká. Alternativně lze využít VARCHAR, či menší číslicový datový typ pro ukládání více hodnot, což vede potenciálnímu narůstání objemu dat. Pokud je úmyslem navrhovat databázi co nejmenší.

Atribut sex v tabulce Dragon a atributy feeding, secretion v tabulce Diary jou nakonec zvoleny jako datový typ TINYINT. Je to z důvodu zachování dvou možností. Atribut name z tabulky TypeLight byl zvolen jako maximálně čtyřiceti pěti znakový VARCHAR. Zvolit jej, jako CHAR je nedostačující, protože jedním znakem nelze vyjádřit druh a zvolit jej, jako TINYINT také nelze v případě, že je v plánu v budoucnu přidávat další druh záznamu. Alternativně se jeví vhodná varianta menšího číselného datového typu, ale je nepraktické při každém přidání nového druhu záznamu zasahovat do aplikace a přidávat nový název pro nový překlad.

Další problematika přichází v podobě ukládání obrázku do databáze. Do databáze je možné ukládat přímo obrázky. V MariaDB existuje datový typ LONGBLOB pro data obrazového charakteru. Nicméně zde hraje roli velikost obrázku, a pak úvaha, zda je vhodné do databáze ukládat obrázky. Například zabírá paměť v databázi nebo může omezit rychlost databáze. Alternativně lze ukládat cestu k obrázku ležícímu na serveru. Tato varianta také není zcela

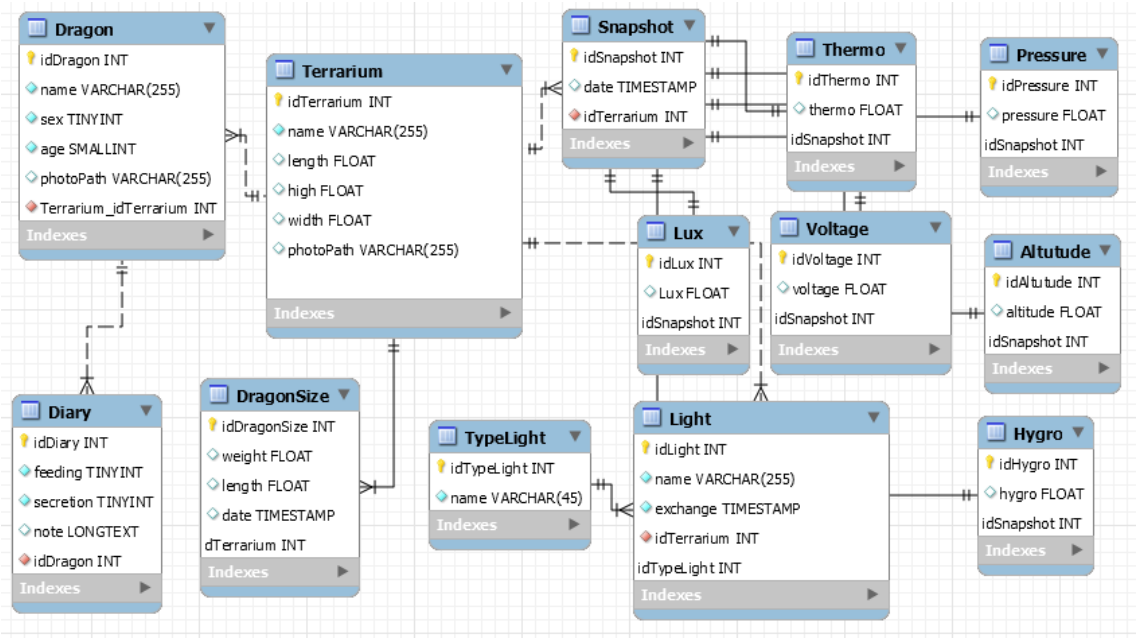
vhodná. Vzniká potenciální bezpečnostní díra a v případě migrace serveru, tzv. změny serveru, je potřeba pak zajistit stejná data na migrovaném serveru, či upravit záznamy.



Obrázek 11 - Fyzický mode 1:N

Verze 1:1

V této variantě tabulka Snapshot má pouze atribut datum a samotné veličiny tvoří jednotlivé tabulky propojené s tabulkou Snapshot ve vztahu 1:1. Výsledkem jsou jednotlivé tabulky obsahující data o konkrétní veličině. Výhodou této varianty je snadná možnost pro budoucí rozšíření systému využívajícího tuto variantu. Oproti předchozí variantě nazvané 1:N by byl viditelný rozdíl v samotném provedení aplikace, která by načítala data z databáze.



Obrázek 12 - Fyzický model 1:1

7.4.2 Varianta s noSQL databází

noSQL databáze nejsou standardizované, a tudíž neexistuje přesný postup, jak při návrhu databáze postupovat. Záleží přímo na tvůrci databáze, popřípadě na komunitách konkrétních databází, či dokumentacích. Lze se inspirovat konceptuálním modelováním a navrhnout si schéma pro noSQL databází. Lze využít například UML diagram. Nicméně není potřeba, jelikož se nepracuje s relacemi.

MongoDB využívá document based databáze. Do kolekce spadá dokument. Ten je ve formátu JSON obsahujícím atributy a jejich záznamy ve tvaru Key-Value. Aplikace se skládá z šesti kolekcí. Kolekce dragon obsahuje informace o jednotlivých agamách. Kolekce diary obsahuje informace o vykonaných potřebách jednotlivých agam. Kolekce terrarium popisuje jednotlivá terária. Kolekce light obsahuje informace o světlech v teráriu. Kolekce snapshotFemale a snapshotMale jsou konkrétní kolekce popisující aktuální stav, tzv. snapshot terárií. Jelikož v projektu jsou dvě terária, pro každé terárium je vytvořen právě jeden specifický snapshot. Bez identifikátoru, by nebylo možné oddělit patřičné záznamy od sebe a přiřadit je ke konkrétním terárií.

Alternativně se nabízí možnost vytvořit jednu kolekci snapshot stejně jako v SQL. Poté evidovat id a vytvořit relaci mezi dokumenty, kterou MongoDB do jisté míry poskytuje. Popřípadě na úrovni aplikace pracovat s id, ale nikoliv přímo v databázi. Nicméně tato varianta

nebyla vybrána pro tento projekt. Je to z důvodu zachování myšlenky noSQL, a tou je absence relací.

Následuje ukázka konkrétního JSON dokumentu kolekce snapshotFemale. Key jsou vlastnosti, jako je teplota, osvětlení, tlak a Value jsou konkrétní řetězcové záznamy. MongoDB defaultně generuje id. V následující ukázce je k dispozici msgid, kde se jedná o id message, čili identifikátor generovaný aplikaci Node-RED.


```
_id: ObjectId("5e6fb3b25d73c51689226fb8")
thermo: "24.12"
lux: "11.7"
pressure: "99497.75"
altitude: "153.19"
hygro: "30.2"
voltage: "2.67"
date: "2020-3-16 18:13:12"
_msgid: "889345d4.b79a78"
```


Obrázek 13 - Ukázka JSON Snapshot

7.5 Srovnání obou variant

Cílem této kapitoly je porovnat oba databázové přístupy z hlediska konkrétního aplikování na konkrétní úlohu. Zejména se bude jednat o implementační rozdíly spolu se subjektivním názorem získaném během praktické části.

7.5.1 Výkonnostní testování

Díky omezeným podmínkách zejména technologického charakteru a povahy aplikované úlohy nelze provést objektivní výkonnostní testování. Proto se tato práce nebude zabývat rychlostí zpracování dat jednotlivých SQL a noSQL databází. Nicméně je známý fakt, že MongoDB je náročná na RAM paměť, ale existují možné konfigurace, jak tento problém omezit. Výkonnostní testování v obecné rovině čili nevztahující se na tuto aplikaci, by bylo možné za následujících podmínek a kroků.

- Vlastnit dostatečně výkonný hardware.
- Provádět testování na konkrétní aplikaci určené pro testování.
- Vybrání vhodných zástupců databází SQL a noSQL.
- Mít dostatečně optimalizované databáze na konkrétní úlohy.
- Změřit rychlost zápisu a čtení daných databází za daných podmínek, jako je změna optimalizace, změna struktury atd., a ty evidovat do výsledku měření.
- Následně porovnat výsledky měření.

7.5.2 Implementační rozdíly

Hlavní rozdíly vycházejí z povahy SQL a noSQL databází, které jsou vysvětleny v kapitole 6. Shrnutí teorie. Například se jedná o strukturu dat, která v případě SQL databází má formu tabulky a v případě noSQL databází má formu dokumentu uloženém v kolekci.

Dalším viditelným rozdílem jsou relace, a to konkrétně absence relací u noSQL. V následující ukázce lze vidět u noSQL varianty JSON řetězec. Je potřeba evidovat jméno agamy atributem „nameDragon“ pro identifikaci záznamu, aby bylo zřejmé, že k dané agamě se pojí zbytek záznamu. Přičemž u SQL varianty (tabulka) se odkazuje na tabulku Dragon, která poskytuje patřičné informace o agamě. To vede k tomu, že u noSQL databáze vznikají duplicity a redundance, protože při každém evidování nového záznamu do deníku je nutné uvádět jméno agamy. Přičemž, jak bylo zmíněno u SQL varianty, se odkáže na daný primární klíč a duplicity nevznikají. Z této situace vyplývá, že v případě dodržení NF poskytuje SQL varianta méně duplicit a redundance dat.

Obrázek 14 Porovnání struktur RDBMS a noSQL

Instalace a příprava

Z hlediska backendu byla u noSQL databáze nutná specifická konfigurace, a to nainstalování Mongo driveru do PHP a vygenerování projektu composerem. Další implementační rozdíly byly spojené s návrhem databáze a manipulací s příslušnou databází pomocí specifických dotazovacích jazyků. Následuje ukázka připojení databáze k MariaDB pomocí PDO a ukázka připojení k MongoDB pomocí PHPLIB.

```php
<?php
$servername = "";
$username = "";
$password = "";

try {
        $conn = new PDO("mysql:host=$servername;dbname=", $username, $password);

    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

            //QUERY
            }
      catch(PDOException $e)
            {
            echo "Connection failed: " . $e->getMessage();
            }
      ?>
```

```php
<?php
require 'vendor/autoload.php'; // include Composer's autoloader
$user = "";
$pwd = "";
$client = new MongoDB\Client('mongodb://'.$user.':'.$pwd.'@localhost/dbName');
try
{
// QUERY
}
      catch (MongoDB\Driver\Exception\ConnectionTimeoutException $e)
      {

      }
      ?>
```

Dotazování a injection

Následují dvě ukázky části kódu. Jedná se o ten samý kód ve dvou provedeních. Cílem kódu je zaznamenat data poslaná metodou POST z formuláře. Tento formulář eviduje následující: datum, krmení, vylučování, poznámku a příslušnou agamu.

V prvním případě se jedná o databázi MariaDB. Proměnné, obsahující data formulářových polí, se nevkládají přímo do dotazu, ale do tzv. prepare statementu společně s využitím placeholderu. V tomto případě je placeholder symbol znaku „?“. Tomu je následně přiřazena daná proměnná a na závěr je celý dotaz proveden příkazem execute a přesměrován na jinou stránku. Jedná se opět o bezpečnostní opatření. Kdyby se vkládaly do dotazu přímo proměnné, tak by bylo možné do dotazu vnořit další dotaz nebo škodlivý kód, se kterým by se útočník mohl dostat do databáze. Jde o tzv. SQL injection.

```
1. <?php
2.
3. require "db.php";
4.
5. if(isset($_POST['submit'])){
6. $DATE=date("Y-m-d H:i:s", strtotime($_POST["datepicker"]));
7. $FEEDING=$_POST["feeding"];
8. $SECRETION=$_POST["secretion"];
9. $NOTE=$_POST["textarea"];
10.       $IDDRAGON=$_POST["agamy"];
11.       $diary= $conn->prepare("
12.          INSERT INTO Diary (date, feeding, secretion, note, idDragon)
13.          VALUES (?,?,?,?,?)
14.
15.          ");
16.
17.       $diary->bindParam(1, $DATE);
18.       $diary->bindParam(2, $FEEDING);
19.       $diary->bindParam(3, $SECRETION);
20.       $diary->bindParam(4, $NOTE);
21.       $diary->bindParam(5, $IDDRAGON);
22.
23.       $diary->execute();
24.       echo "Posláno";
25.       header('location: '."diary.php");
26.
27.       }else{
28.               echo "nefunguje ";
29.       }
```

V druhé ukázce se jedná o databázi MongoDB. Pro zachycení dat jednotlivých formulářových polí se použije metoda addslahes(), která znemožňuje útočníkovi rozbít dotaz a pokusit se tak o tzv. injection. Další ochranou je ujištění, že jsou proměnné vhodného datového typu. Následně se provede dotaz, který provede insert dat do databáze a přesměruje na danou stránku. Povědomí o injection v databázích noSQL není tak velké jako u SQL databází. Injection může docházet, jak u řešení s php, tak s JavaScriptem. (Mittal, 2016) (Idontoplaydarts, 2010)

```php
<?php
require 'db.php';
if(isset($_POST['submit'])){
$DATE=addslashes(date("Y-m-d H:i:s", strtotime($_POST["datepicker"])));
$FEEDING=addslashes($_POST["feeding"]);
$SECRETION=addslashes($_POST["secretion"]);
$NOTE=addslashes($_POST["textarea"]);
$DRAGON=addslashes($_POST["agamy"]);

        //$collDiary = $client->terrarium->diary;
        $result = $collDiary->insertOne( [ 'date' => (date)$DATE, 'feeding' => (int)$FEEDING, 'secretion' => (int)$SECRETION, 'note' =>(string)$NOTE, 'dragon'=>(string)$DRAGON
        echo "Posláno";
        header('location: '."diary.php");

        }else{
                echo "nefunguje Form1";
        }
```

Další ukázka dotazu – porovnání

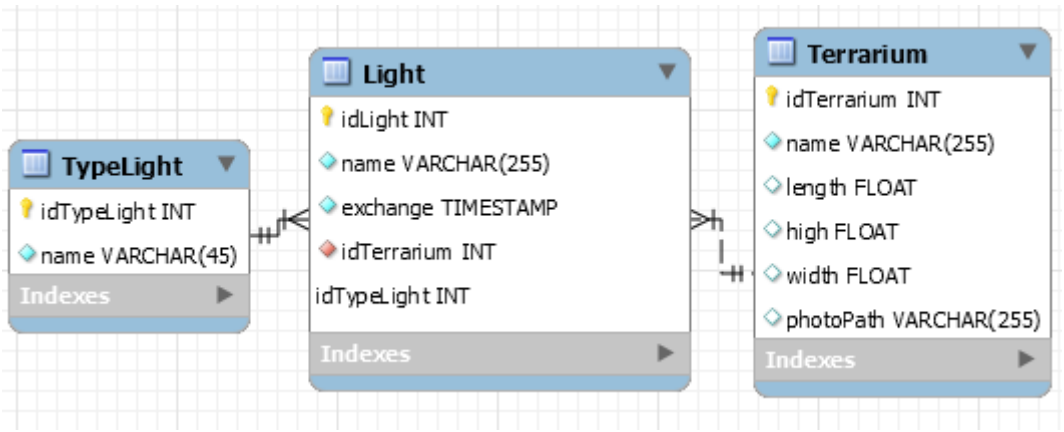
Následující tabulka popisuje seznam světel v daných teráriích. Tabulka se liší v dotazování v daných databázích. Nejprve bude vysvětlena varianta s MariaDB a potom s MongoDB.

Terárko	Světlo	Kategorie	Datum výměny	Odstranit
Terárko pokoj	SunLux UVB 15.0 25W	UVB	20-10-2019	delete
Terárko pokoj	SunLux UV 50W PAR30	UVA+UVB	01-10-2019	delete
Terárko pokoj	Narva Biovital 30W 90cm T8	Plnospektrální zářivka	12-09-2019	delete
Terárko chodba	Narva Biovital 30W 90cm T8	Plnospektrální zářivka	15-06-2019	delete
Terárko chodba	Reptisun 10.0 UVB 26W	UVB	20-10-2019	delete
Terárko chodba	Repti Basking Spot Lamp 40W	UVA	18-01-2020	delete

Obrázek 15 - Tabulka seznamu světel v aplikaci

Ve variantě MariaDB se provede select (výběr) vybraných sloupců z tabulek Light a TypeLight a Terrarium. Tabulky jsou propojené inner joinem přes klíč idTypeLight a idTerrarium.

```
1. $light = $conn->query("
2.         SELECT Light.idLight, Light.name as lamp, Light.exchange,
   TypeLight.name as category, Terrarium.name
3.         FROM Light
4.                         INNER JOIN TypeLight
5.                         ON Light.idTypeLight=TypeLight.idTypeLight
6.                         INNER JOIN Terrarium
7.                         ON Light.idTerrarium=Terrarium.idTerrarium
8.         ");
```



Obrázek 16 - Schéma tabulek pro seznam světel

V případě varianty v MongoDB se provede find (výběr) z kolekce light. Oproti předchozí variantě je tento zápis znatelně kratší díky absence joinu, neboli spojování tabulek.

```
1. $collLight = $client->terrarium->light; // nastavení kolekce
2. $lightFind = $collLight->find();
```

Další příklad – odlišné řešení

Výpis konkrétních světel k danému teráriu je ve variantě v MariaDB řešen jako select (výběr) sloupců v závislosti na ID terária. Viz následující ukázka. Klauzule where filtruje záznamy v závislosti na parametru. V tomto případě je to pojmenovaný placeholder :ID, který je zde demonstrován jako další varianta placeholderu.

```
1. $lightInTerrarium= $conn->prepare("
2.      SELECT Terrarium.name AS nameTerrarium, Light.name FROM Light
   INNER JOIN Terrarium ON Terrarium.idTerrarium = Light.idTerrarium
   WHERE Terrarium.idTerrarium = :IDT
3.
4.    ");
5. $lightInTerrariumExcution = $lightInTerrarium->execute([
6. 'IDT' => $NUM,
7. ]);
```

Ve variantě s MongoDB je výpis konkrétních světel k danému terárium řešen pomocí dvou separátních dotazů, které jsou filtrované podle názvu terária. Toto řešení není oproti předchozí variantně zcela automatizované. Problém přichází v případě, když by přibylo nové terárium, protože pak by bylo nutné vytvořit další dotaz pro nové terárium. Alternativně by šel problém vyřešit ručním přidáním číslovaného ID na úrovni aplikace a ten editovat jako atribut. Případně využít Mongo ID, který, ale není automatické číslo.

```
1. $lightStairs = $collLight->find(array(
2.         "terrarium" => "Terárko chodba"
3. ));
4.
5.
6. $lightRoom = $collLight->find(array(
7.          "terrarium" => "Terárko pokoj"
8.  ));
```

7.5.3 Subjektivní zhodnocení

V této podkapitole zhodnotím noSQL a SQL databáze dle vlastních poznatků získaných během praktické části práce a znalostí nabytých z teoretické části práce. Formou krátkých odpovědí shrnu svůj názor, který se bude týkat pouze databází, nikoli tvorby celé aplikace.

Návrh databáze

Navrhnout dané modely tak, aby celý systém dával smysl s ohledem na dodržování pravidel, bylo časově náročnější než návrh noSQL databáze. Jelikož osobně mám několik zkušeností s návrhem SQL, tak při tvorbě noSQL databáze mi dělalo problémy nemyslet „relačně“ a oprostit se od myšlenky klíčů a zároveň přijmou fakt, že občas eviduji duplicitní záznamy, místo abych odkázal na jinou entitu, viz příklad Implementační rozdíly. Nicméně s prací v SQL databázi jsem se několikrát musel vracet do návrhu a předělávat ho. Také jsem musel řešit problematiku kolem datových typů, kdy a za jakých podmínek vybrat vhodný datový typ.

Práce s databázemi

Co se týká samotné práce s databázemi, s oběma se mi pracovalo dobře. Při manipulaci s databází bylo potřeba využít i dotazovací jazyky. Syntaxe dotazovacího jazyka v MongoDB byla lehce odvoditelná a svým způsobem mi tento dotazovací jazyk připomínal SQL s jinou syntaxí. Syntaxe měla podrobnou dokumentaci na stránkách MongoDB. Dle mého názoru je i snadnější a přirozenější.

Konfigurace databáze

Konfigurace MongoDB byla složitější a časově komplikovanější z důvodu nutnosti externí konfigurace. U MariaDB stačilo nainstalovat a pouze spustit službu.

Udělal bych příště něco jinak?

Nejspíš bych zvolil noSQL databázi a web s prostým výpisem informací, neboť toto řešení je časově méně náročné.

8 Závěr

Cílem práce bylo porovnat databázové přístupy a vyhodnotit je. Zároveň zjistit, zda SQL databáze, které jsou starší, budou nahrazeny novějšími noSQL databázemi. Z práce vyplývá, že nelze říci jednoznačně, zda budou SQL databáze nahrazeny. Každý databázový přístup je ovlivněn svou historií, ale i uplatněním, které si našly oba databázové přístupy.

Výsledná aplikace byla úspěšně realizována použitím obou popisovaných databázových přístupů. Po stránce výkonnostní se v práci nepodařilo rozhodnout, která z databází je lepší, z důvodu nedostatečné optimalizace a povahy aplikace. Nicméně po stránce návrhu a implementace vychází výsledná aplikace s SQL databází jako časově náročnější na realizaci. Je úspornější po stránce datové, neboť nedochází v takové míře k redundanci dat oproti noSQL variantě. Je nutné podotknout, že tento výsledek je značně ovlivněn specifickými požadavky na výslednou aplikaci, a tak pokud by aplikace měla jiné požadavky, výsledek by mohl být odlišný. Práce totiž ukazuje, že pokud je výsledná aplikace složitějším systémem, například redakční systém, je vhodnější SQL databáze. Pokud by aplikace měla za úkol pouze vypisovat velké množství dat, jeví se noSQL databáze jako vhodnější, a to z důvodu rychlejšího a snazšího návrhu.

9 Seznam použitých informačních zdrojů

ASHRAF, Haroon. Replacing SQL Cursors with Alternatives to Avoid Performance Issues. *Codingsight.com* [online]. 2019 [cit. 2020-04-17]. Dostupné z: https://codingsight.com/replacing-sql-cursors-with-alternatives-to-avoid-performance-issues/

BÁRTÍK, František. Cassandra: Databáze pro skutečně velké projekty. *Linuxexpres* [online]. 2013 [cit. 2020-04-16]. Dostupné z: https://www.linuxexpres.cz/software/cassandra-databaze-pro-skutecne-velke-projekty

BÁRTÍK, František. Cassandra DB - I. *Linuxsoft.cz* [online]. 2011 [cit. 2020-04-17]. Dostupné z: http://archiv.linuxsoft.cz/article.php?id_article=1850

BAWDEN, David a Lyn ROBINSON. *Úvod do informační vědy*. Doubravník: Flow, 2017. ISBN 978-80-88123-10-1.

BÍLA, Jiří a František KRÁL. *Databázové a znalostní systémy*. Praha: České vysoké učení technické, 1999. ISBN 80-01-01925-X.

CARVELLI, Roberto. SQL & NOSQL: A BRIEF HISTORY. *Grio.com* [online]. 2015 [cit. 2020-04-16]. Dostupné z: https://blog.grio.com/2015/11/sql-noSQL-a-brief-history.html

CONOLLY, Thomas, Carolyn E. BEGG a Richard HOLOWCZAK. *Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází*. Brno: Computer press, 2009. ISBN 978-80-251-2328-7.

ČÁPKA, David. Lekce 2 - UML - Use Case Diagram. *Itnetwork.cz* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram

ČERNÝ, Michal. Big data a jejich zpracování. *Root.cz* [online]. 2013 [cit. 2020-04-16]. Dostupné z: https://www.root.cz/clanky/big-data-a-jejich-zpracovani/

DOSTALOVÁ, Zuzana. Frontend vs. Backend. *Czechitas* [online]. 2014 [cit. 2020-04-20]. Dostupné z: https://www.czechitas.cz/cs/blog/zaciname-s-it/frontend-vs-backend

FOOTE, Keith. A Brief History of Non-Relational Databases. *Dataversity.net* [online]. 2018 [cit. 2020-04-16]. Dostupné z: https://www.dataversity.net/a-brief-history-of-non-relational-databases/

GOEL, Aman. Normalization in DBMS: 1NF, 2NF, 3NF and BCNF with Examples. *Hackr.io* [online]. 2020 [cit. 2020-04-16]. Dostupné z: https://hackr.io/blog/dbms-normalization

HAVLÍČEK, Zdeněk. *Databázové systémy: Paradox*. Praha: Grada, 1992. Educa. ISBN 80-85424-97-5.

HRONEK, Jiří. *Databázové systémy* [online]. Olomouc, 2007 [cit. 2020-04-16]. Dostupné z: https://docplayer.cz/1526745-Databazove-systemy-jiri-hronek-katedra-informatiky-prirodovedecka-fakulta-univerzita-palackeho.html

HUGG, John. Disambiguating ACID and CAP. *Voltdb.com* [online]. 2015 [cit. 2020-04-23]. Dostupné z: https://www.voltdb.com/blog/2015/10/22/disambiguating-acid-cap/

CHAPPLE, Mike. Abandoning ACID in Favor of BASE in Database Engineering. *Lifewire.com* [online]. 2020 [cit. 2020-04-23]. Dostupné z: https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674

JANKOV, Tonino. MariaDB vs MySQL, a Database Technologies Rundown. *Kinsta* [online]. 2020 [cit. 2020-04-16]. Dostupné z: https://kinsta.com/blog/mariadb-vs-mysql/

KOKEŠ, Josef. *Programové systémy*. Praha: Vydavatelství ČVUT, 2005. ISBN 80-01-03149-7.

KRČMÁŘ, Michal. Co jsou big data a k čemu jsou dobrá? *Objevit.cz* [online]. 2016 [cit. 2020-04-16]. Dostupné z: https://www.objevit.cz/co-jsou-big-data-a-k-cemu-jsou-dobra-t157688

KRISCIUNAS, Albertas. BlogNoSQL vs. SQL: Database Comparison. *Cloud.netapp.com* [online]. 2014 [cit. 2020-04-17]. Dostupné z: https://www.devbridge.com/articles/benefits-of-noSQL/

KUMST, Martin. Seznámení s SQLite. *Root.cz* [online]. 2016 [cit. 2020-04-16]. Dostupné z: https://blog.root.cz/maertienuv-obcasny-blog/seznameni-s-sqlite/

LANGE, Carly. How to Select the Right Warehouse Management System (WMS). *Establish* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://www.establishinc.com/supply-chain-blog/warehouse-management-system-selection

LASÁK, Pavel. Relace v databázích - teorie. *Lasakovi.com* [online]. 2010 [cit. 2020-04-16]. Dostupné z: https://office.lasakovi.com/access/tabulky/relace-vazby-v-databazich-teorie/

MERUNKA, Vojtěch. *Objektový přístup v databázových systémech*. Praha: Česká zemědělská univerzita, 2002. ISBN 80-213-0882-6.

MISHA. Databáze. *Databaze.chytrak.com* [online]. 2010 [cit. 2020-04-17]. Dostupné z: http://www.databaze.chytrak.cz

MITTAL, Nikhil. MongoDB security – Injection attacks with php. Securelayer7.net [online]. 2016 [cit. 2020-04-17]. Dostupné z: https://blog.securelayer7.net/mongodb-security-injection-attacks-with-php/

MULLINS, Craig. What do we mean by Database Scalability? *Nuodb* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://www.nuodb.com/techblog/what-do-we-mean-database-scalability

OLIVERA, Lucas. Everything you need to know about NoSQL databases. *Dev.to* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://dev.to/lmolivera/everything-you-need-to-know-about-nosql-databases-3o3h

ONET, Sergiu. Using SQL Server cursors – Advantages and disadvantages. *Sqlshack* [online]. 2016 [cit. 2020-04-17]. Dostupné z: https://www.sqlshack.com/using-sql-server-cursors-advantages-and-disadvantages/

OSKOW, Kevin. MariaDB or MySQL – Have you chosen the DBMS that you˙NEED? *Uplers.com* [online]. 2017 [cit. 2020-04-16]. Dostupné z: https://www.uplers.com/blog/mariadb-vs-mysql-an-industry-wise-comparison/

PARAHAR, Mahesh. Difference between RDBMS and OODBMS. *Tutorialspoint.com* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://www.tutorialspoint.com/difference-between-rdbms-and-oodbms

PEKÁŘ, Lukáš. DEKLARATIVNÍ PROGRAMOVÁNÍ. *Bonsai development* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://bonsai-development.cz/clanek/deklarativni-programovani

PERRY, Yifat. BlogNoSQL vs. SQL: Database Comparison. *Cloud.netapp.com* [online]. 2018 [cit. 2020-04-17]. Dostupné z: https://cloud.netapp.com/blog/noSQL-vs-sql-database-what-is-the-difference

PIVERT, Olivier. *NoSQL data models: trends and challenges*. London: ISTE, 2018. Computer engineering series. ISBN 978-1-78630-364-6.

POKORNÝ, Jaroslav a Michal VALENTA. *Databázové systémy*. Praha: Česká technika - nakladatelství ČVUT, 2013. Vysokoškolská učebnice. ISBN 978-80-01-05212-9.

PORE, Akshay. NoSQL Data Architecture & Data Governance: Everything You Need to Know. *Dataversity.net* [online]. 2018 [cit. 2020-04-16]. Dostupné z: https://www.dataversity.net/noSQL-data-architecture-data-governance-everything-need-know/

RAČANSKÝ, Luboš. Proč stojí objektové programování za starou belu. *Zdrojak.cz* [online]. 2017 [cit. 2020-04-23]. Dostupné z: https://www.zdrojak.cz/clanky/proc-stoji-objektove-programovani-za-starou-belu/

ROTH, Jason. What Is ODBC? *Sqlite* [online]. 2017 [cit. 2020-04-17]. Dostupné z: https://docs.microsoft.com/en-us/sql/odbc/reference/what-is-odbc?view=sql-server-ver15

SARIG, Matan. MariaDB Vs MySQL In 2019: Compatibility, Performance, And Syntax. *Panoply* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://blog.panoply.io/a-comparative-vmariadb-vs-mysql

SIMSEK, Gokhan. What Is New About NewSQL? *Softwareengineeringdaily.com* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://softwareengineeringdaily.com/2019/02/24/what-is-new-about-newsql/

SKŘIVAN, Jaromír. Databáze a jazyk SQL. *Interval.cz* [online]. 2000 [cit. 2020-04-16]. Dostupné z: https://www.interval.cz/clanky/databaze-a-jazyk-sql/

ŠPOLJARIČ, Damir. Trend s názvem „horizontální škálování“: kdy má smysl a o co vlastně jde? *Vshosting* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://vshosting.cz/blog/trend-s-nazvem-horizontalni-skalovani-kdy-ma-smysl-a-o-co-vlastne-jde

THAKUR, Dinesh. What is Object Oriented Database (OODB)? *Ecomputernotes.com* [online]. [cit. 2020-04-17]. Dostupné z: http://ecomputernotes.com/database-system/adv-database/object-oriented-database-oodb

VASILIEV, Alexey. World of the NoSQL databases. *Leopard blog* [online]. 2013 [cit. 2020-04-16]. Dostupné z: https://leopard.in.ua/2013/11/08/nosql-world#.XpiZVTfVKUk

VRÁNA, Jakub. Využití databázových indexů. *Root.cz* [online]. 2003 [cit. 2020-04-16]. Dostupné z: https://www.root.cz/clanky/vyuziti-databazovych-indexu/

WANG, Ruihan a Zongyan YANG. SQL vs NoSQL: A Performance Comparison. *Rochester.edu* [online]. University of Rochester, 2017 [cit. 2020-04-17]. Dostupné z: https://www.cs.rochester.edu/courses/261/fall2017/termpaper/submissions/06/Paper.pdf

WELLS, Joyce. Key Database Trends Now and for the Future. *Dbta.com* [online]. 2019 [cit. 2020-04-17]. Dostupné z: http://www.dbta.com/Editorial/News-Flashes/Key-Database-Trends-Now-and-for-the-Future-131888.aspx

WENZEL, Kris. What is a Database Cursor? *Essentialsql.com* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.essentialsql.com/database-cursor/

ŽÁK, Karel. Historie Relačních Databází. *Root.cz* [online]. 2001 [cit. 2020-04-16]. Dostupné z: https://www.root.cz/clanky/historie-relacnich-databazi/

Azure 1. *Co je cloud computing?* [online]. 2020 [cit. 2020-04-23]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-cloud-computing/

Azure 2. *Databáze NoSQL* [online]. 2020 [cit. 2020-04-23]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/nosql-database/

Blackbaund. *Data Warehouse Structures* [online]. [cit. 2020-04-17]. Dostupné z: https://www.blackbaud.com/files/support/guides/infinitydevguide/Subsystems/infrep-developer-help/content/reportcamp/codatawarehousestructures.htm

Besthosting. *Co je to Microsoft Offcie Access?* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://best-hosting.cz/cs/napoveda/co-je-to-microsoft-office-access

Dataflair. *Advantages of MongoDB | Disadvantages of MongoDB* [online]. 2018 [cit. 2020-04-17]. Dostupné z: https://data-flair.training/blogs/advantages-of-mongodb/

Geeksforgeeks. *SQL | DDL, DQL, DML, DCL and TCL Commands* [online]. [cit. 2020-04-17]. Dostupné z: https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/

Hadoop. *Apache Hadoop YARN* [online]. 2019 [cit. 2020-04-17]. Dostupné z: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Hbase. *Powered By Apache HBase* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://hbase.apache.org/poweredbyhbase.html

Idontplaydarts. *Mongodb is vulnerable to SQL injection in PHP at least* [online]. 2010 [cit. 2020-04-17]. Dostupné z: https://www.idontplaydarts.com/2010/07/mongodb-is-vulnerable-to-sql-injection-in-php-at-least/

Informační technologie. *Kardinalita vztahu* [online]. 2009 [cit. 2020-04-17]. Dostupné z: https://informacni-technologie.studentske.cz/2009/02/kardinalita-vztahu.html

Intellipaat. *What is Cassandra?* [online]. 2016 [cit. 2020-04-17]. Dostupné z: https://intellipaat.com/blog/what-is-apache-cassandra/

Iot. *Co je IoT?* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.iot-portal.cz/co-je-iot/

Json. *Úvod do JSON* [online]. [cit. 2020-04-17]. Dostupné z: https://www.json.org/json-cz.html

Learnitanytime. *Know The Advantages And Disadvantages Of Microsoft Access* [online]. 2013 [cit. 2020-04-17]. Dostupné z: http://learnitanytime.com/4031/know-the-advantages-and-disadvantages-of-microsoft-access-2/

Masarykova Univerzita – Fakulta informatiky. *Relační vs. objektově- elační vs. Objektové databáze* [online]. [cit. 2020-04-20]. Dostupné z: https://www.fi.muni.cz/~xbatko/oracle/compare.html

Mendelu 1. *Mendelova univerzita v Brně* [online]. [cit. 2020-04-17]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=9071

Mendelu 2. *Mendelova univerzita v Brně* [online]. [cit. 2020-04-17]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/popupokno.pl?pojem=2775

Mendelu 3. *Mendelova univerzita v Brně* [online]. [cit. 2020-04-17]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=5124

MongoDB 1. *Who use MongoDB* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.mongodb.com/who-uses-mongodb

MongoDB 2. *JSON a BSON* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.mongodb.com/json-and-bson

Ncache. *Why NoSQL?* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.alachisoft.com/nosdb/why-nosql.html

Neo4j. *What is a Graph Database?* [online]. 2020 [cit. 2020-04-23]. Dostupné z: https://neo4j.com/developer/graph-database/

Oracle 1. *Co je to cloudová databáze?* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.oracle.com/cz/database/what-is-a-cloud-database/

Oracle 2. *Co je datový sklad?* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.oracle.com/cz/database/what-is-a-data-warehouse

Oracle 3. *Co je relační databáze* [online]. 2020 [cit. 2020-04-16]. Dostupné z: https://www.oracle.com/cz/database/what-is-a-relational-database/

Redis 1. *Introduction* [online]. [cit. 2020-04-17]. Dostupné z: https://redis.io/topics/introduction

Redis 2. Who's using redis [online]. [cit. 2020-04-17]. Dostupné z: https://redis.io/topics/whos-using-redis

Redislabs.com. *Redis Enterprise Overview* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://redislabs.com/redis-enterprise/

Tutorialspoint 1. *XML - Databases* [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.tutorialspoint.com/xml/xml_databases.htm

Tutorialspoint 2. *Hbase - Overview* [online]. 2020 [cit. 2020-04-23]. Dostupné z: https://www.tutorialspoint.com/hbase/hbase_overview.htm

Sqlite. *What Is SQLite?* [online]. [cit. 2020-04-17]. Dostupné z: https://www.sqlite.org/index.html

Sqlprogrammers. *Disadvantages of Using Access* [online]. Chicago, 2014 [cit. 2020-04-17]. Dostupné z: https://sql-programmers.com/disadvantages-of-access

Vitfo. *Začínáme s Hadoopem: Co je to HDFS* [online]. 2018 [cit. 2020-04-17]. Dostupné z: https://www.vitfo.cz/2018/05/17/zaciname-s-hadoopem-co-je-to-hdfs/

WordPress. *Mujwordpress.cz* [online]. [cit. 2020-04-17]. Dostupné z: http://www.mujwordpress.cz

10 Seznam příloh

Příloha 1 - Porovnání MongoDB a MySQL[2]

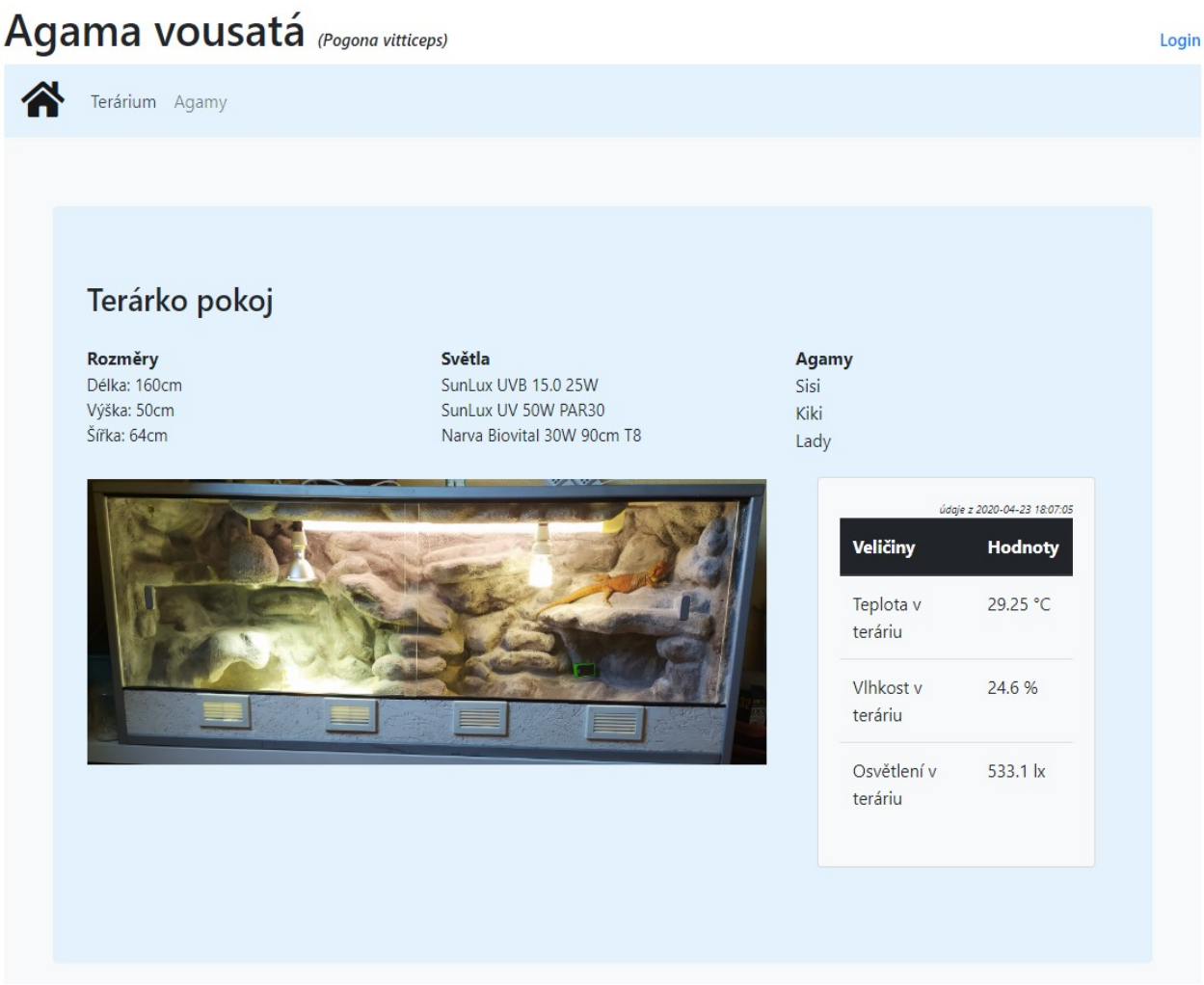
Type	Feature	mongoDB	MySQL
Deployment	Cloud, SaaS, Web	✓	✓
	Developers	MongoDB Inc.	Oracle Corporation
	Schema	Flexible	Rigid
	OS	Multi platform	Multi platform
Design & Features	Data storage	JSON	Column & Rows
	Query lang.	Volatile memory file system	SQL
	MapReduce	✓	✗
	Unicode	✓	✓
	Backup	✓	✓
	Creation/Development	✓	✗
	Database conversion	✓	✗
	Monitoring	✓	✓
	Performance analysis	✓	✗
	Queries	✓	✗
	Relational interface	✓	✗
	Virtualisation	✓	✗
Integrity	Integrity model	BASE	ACID
	Atomicity	Conditional	✓
	Consistency	✓	✓
	Isolation	✗	✓
	Durability (data storage)	✓	✓
	Transactions	✗	✓
	Referential integrity	✗	✓
Indexing	Secondary indexes	✓	✓
	Composite keys	✓	✓
	Full text search	✓	✓
	Geospatial indexes	✓	✗
	Graph support	✗	✗
Distribution	CAP	CP	CA
	Horixontal scalablility	✓	Conditional
	Replication	✓	✓
	Data migration	✓	✓
	Replication mode	Master-Slave	Master-Master/Slave
	Sharding	✓	✓
	Shared nothing architecture	✓	✓
System	Programming language	C, C++, Java	C, C++, C#, Python, Java, NodeJS

SIMFORM

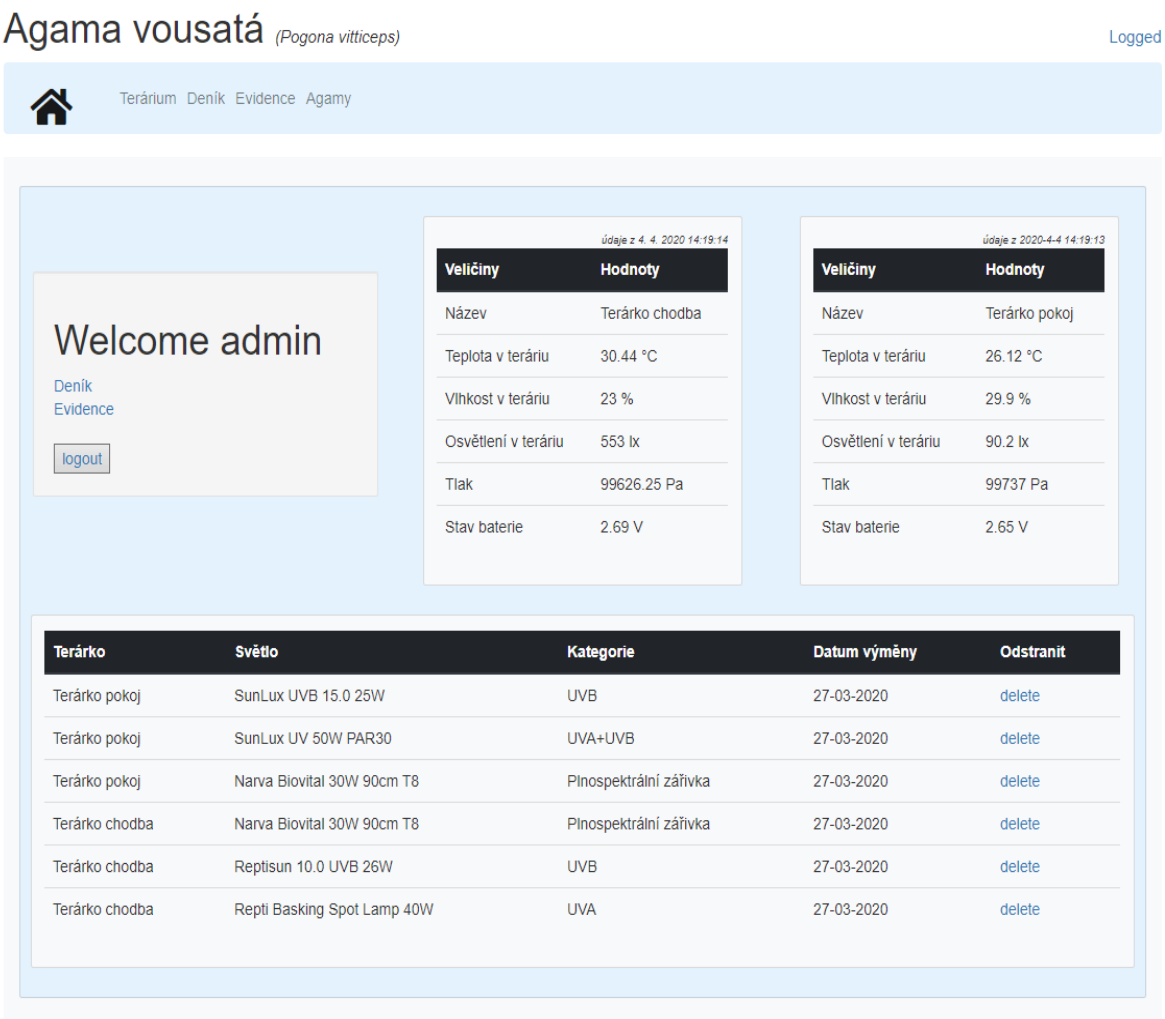
If you use this graphic, please cite and link to www.Simform.com

[2] Software Development Company USA | Simform [online]. Dostupné z: https://www.simform.com/mongodb-vs-mysql-databases/#section3

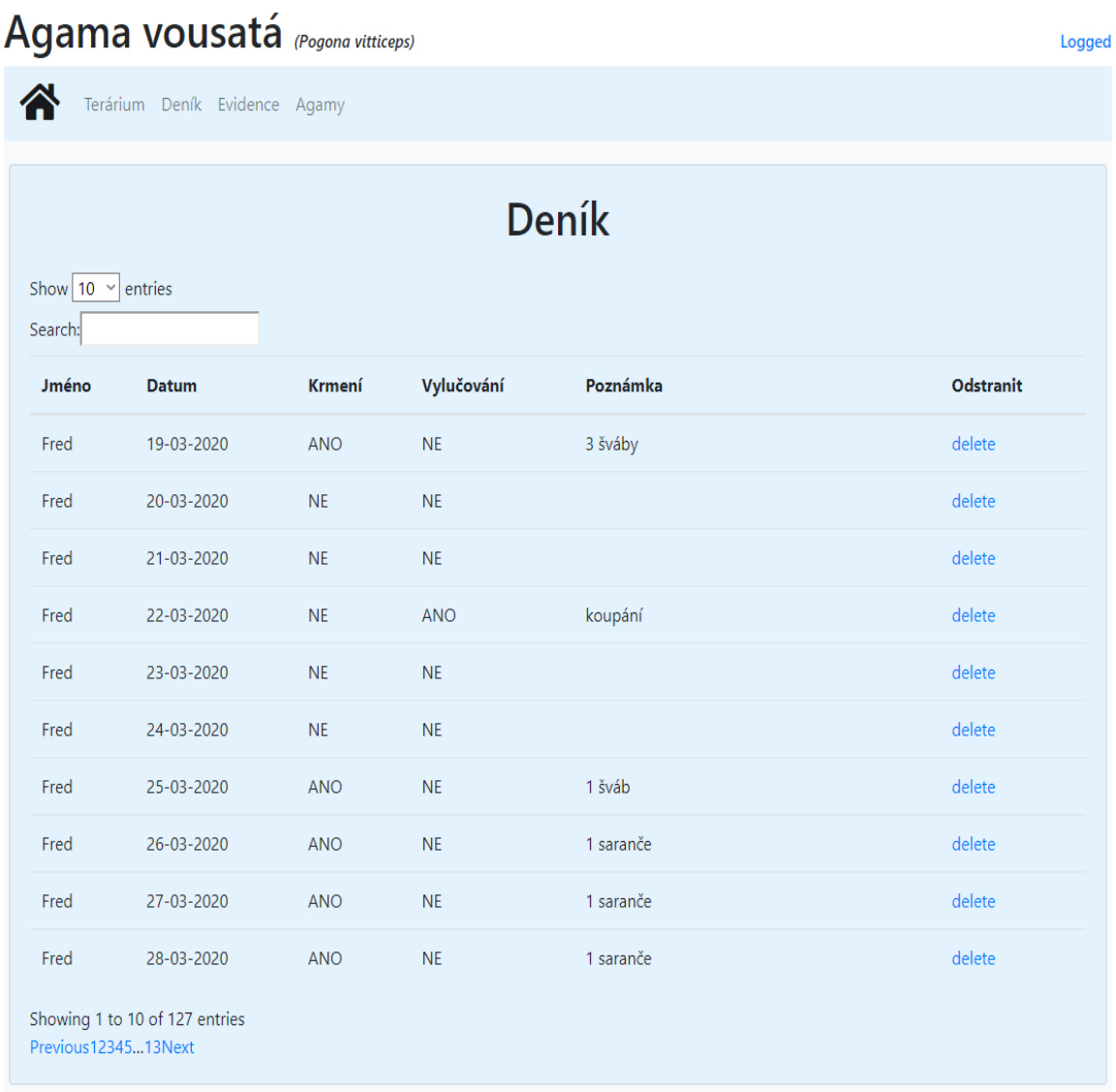
Příloha 2 - Ukázka terária z vytvořené aplikace



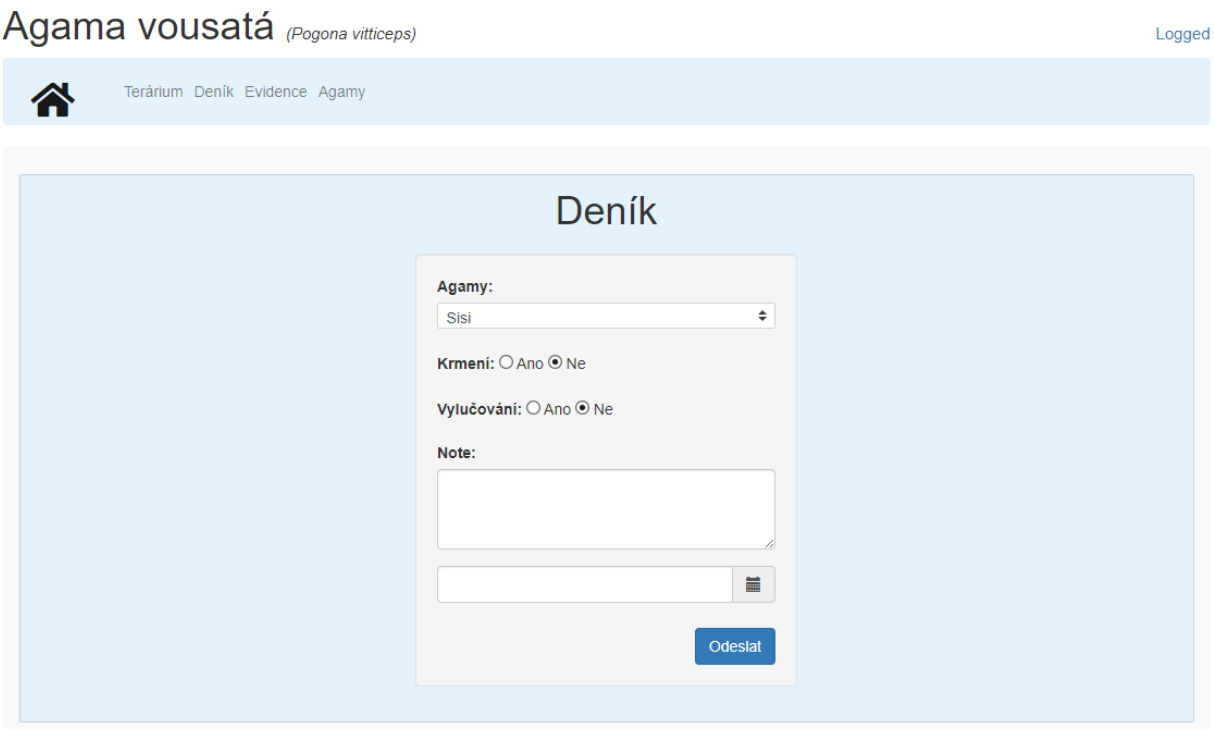
Příloha 3 - Ukázka administrace z vytvořené aplikace



Příloha 4 - Ukázka výpisu deníku z vytvořené aplikace



Příloha 5 - Ukázka vkládajícího formuláře deníku z vytvořené aplikace



Příloha 6 - Ukázka formuláře z vytvořené aplikace