Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Návrh a realizace databázové aplikace pro evidenci studentů

Michael Mikuš

Vedoucí práce: Ing. Tomáš Haubert

16. února 2016

Poděkování

Na tomto místě bych rád poděkoval všem, kteří mě podporovali při psaní této práce. Především panu Ing. Tomáši Haubertovi za odborné, vstřícné a trpělivé vedení mé bakalářské práce. Také mé rodině, přítelkyni a přátelům za podporu a zázemí během celé doby mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 16. února 2016 .....

Odkaz na tuto práci

Mikuš, Michael. *Návrh a realizace databázové aplikace pro evidenci studentů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem, realizací a především konfigurací databázové aplikace pro evidenci návštěv kurzů taneční školy skrze čtečku čárových kódů s využitím .NET technologií. Čtenář je seznámen s hlavními databázovými technologiemi, návrhy, postupy, správou a zabezpečením databáze tak, že je schopen vybrat nejen vhodnou databázi společně se systémem řízení báze dat, ale také ji korektně navrhnout či vědět o nejrůznějších možnostech kolem databázových technologií. Hlavním cílem práce je také návrh a realizace databázového úložiště, klientské a serverové aplikace se zaměřením na konfiguraci webového serveru. Výsledkem praktické části práce je docházková aplikace pro evidenci studentů.

Klíčová slova databázová aplikace, docházková aplikace, databáze, SŘBD, SQL Server, .NET Framework, C#, konfigurace IIS, konfigurace WCF, čtečka čárových kódů

Abstract

This bachelor thesis studies the analysis, design, implementation and specifically the configuration of the database application for recording visits of dance school courses via a barcode scanner using .NET technology. The reader is familiarized with the main database technology, design, process, management and security of the database in a sense that he is able to not only choose the appropriate database management system but also design and know of various options in database technology. The main aim of the work is the design and implementation of the database storage, client and server applications with a focus on web server configuration. The result of the practical part is an attendance application to record students.

Keywords database application, attendance application, database, DBMS, SQL Server, .NET Framework, C#, IIS configuration, WCF configuration, barcode reader

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Dnešní moderní IT svět můžeme pojmout jako tržnici, kde je zákazníkům, jakožto klientům, nabízeno pestré množství služeb prostřednictvím webových serverů, které jako skladiště můžou využívat databázovou základnu. V tomto ekosystému se orientujeme prostřednictvím aplikací, aniž bychom registrovali procesy odehrávající se na pozadí.

Z pohledu této bakalářské práce se zabývám ekosystémem od společnosti Microsoft, kde aplikace vyvíjené pod jejími technologiemi jsou uživatelsky přívětivé pro většinovou populaci. Z důvodu nasazení aplikace na operační systém Windows, který budou obsluhovat uživatelé neznalí informačních technologií, jsem zvolil tento přístup.

V průběhu studia se mi naskytla příležitost pracovat pro dánskou firmu vyvíjející aplikace v prostředí .NET. Při seznamování s reálnými projekty jsem brzy přišel do styku s konfigurací webového serveru a webových služeb, které jsou podstatnou součástí pro mnoho dnešních aplikací vyžadující sdílení dat a interakci mezi systémy. Během praxe jsem pochytil mnoho cenných informací a rad od zkušených senior vývojářů v oblasti konfigurace webového serveru Internet Information Services (IIS) a služeb Windows Communication Foundation (WCF) v něm hostovaných. Databázová aplikace tedy získá určitý profesionální přístup a metodiky z praktického hlediska.

Cílem mé bakalářské práce je analyzovat, navrhnout, realizovat a především konfigurovat databázovou aplikaci pro evidenci návštěv kurzů taneční školy skrze čtečku čárových kódů. Teoretickým cílem práce je seznámit čtenáře s hlavními databázovými technologiemi, návrhy, postupy, správou a zabezpečením databáze tak, že je schopen vybrat nejen vhodnou databázi společně se systémem řízení báze dat, ale i ji korektně navrhnout či vědět o nejrůznějších možnostech kolem databázových technologií. Hlavním cílem praktické části je také návrh a realizace databázového úložiště, klientské a serverové aplikace se zaměřením na konfiguraci webového serveru. Výstupem je systém, využívající čtečku čárových kódů a podporující všechny potřebné vlastnosti evidenčního systému těchto kurzů, jenž může být využit v praxi.

Evidenční systém implementuje RESTful webové služby, které zprostředkovávají přístup k její funkcionalitě pro klienty, jenž k ní můžou přistupovat vzdáleně. Jednotlivé datové složky, posílány mezi klientem a službou, jsou posílány obalené v zabezpečeném komunikačním HTTPS protokolu. V současné době jsem se v praxi setkal s nejnovější implementací webových služeb a to pomocí Windows Communication Foundation.

Praktická část je věnována konfiguraci, návrhu a realizaci aplikace evidenčního systému, která se skládá ze tří vrstev - databázové úložiště, serverová a klientská aplikace. Pro datové úložiště byl rozebrán návrh, realizace a testování. Důraz je kladen na realizaci autentizace, autorizace a přístupových práv administrátora a hosta, kteří mají striktně stanovený prostor privilegií. Důležitou praktickou částí je také konfigurace konfiguračního souboru webové služby hostované na webovém serveru, který byl také patřičně nastaven tak, aby byla zajištěna bezpečnost komunikace mezi klientem a serverem. Dále ve stručnosti rozvádím klientskou část využívající již zmíněnou čtečku čárových kódů a popis vytvořené technické dokumentace. Jednotlivé části databázové aplikace byly otestovány jednotkovými testy. Tato bakalářská práce obsahuje větší množství zkratek, které jsou blíže popsány v separátní kapitole na konci této práce.

Mojí motivací k práci na tomto projektu je především možnost zdokonalit se v oblasti mého pracovního zaměření, což jsou programovací jazyk C#. NET, konfigurace webového serveru IIS a webové servisy. Pochopení této problematiky do hlubších detailů mi může dopomoci stát se expertem v dané oblasti. Za další motivační prvek považuji využitelnost bakalářské práce na reálných tanečních kurzech, případná spokojenost uživatelů, možnost rozšíření o nové klienty a podstatný přínos pro čtenáře bádající po korektním výběru databáze.

Kapitola 1

Cíle

V této kapitole budou v návaznosti na zadání podrobněji popsány cíle práce.

1.1 Cíle teoretické části

Osobně jsem postrádal koncepčně vhodný zdroj informací poskytující průřez hlavními databázovými technologiemi, návrhy, postupy a bezpečností, po kterých je čtenář schopen vybrat vhodnou databázi. Hlavní cíl, výběr vhodné „databáze“, je rozčleněn do následující podcílů:

- Seznámit se s historií a terminologií databázových technologií.
- Analyzovat souborový a databázový přístup.
- Seznámit se s jednotlivými fázemi životního cyklu vývoje databázového systému.
- Analyzovat výhody a nevýhody hlavních databázových modelů.
- Analyzovat výhody a nevýhody architektur databázového systému.
- Seznámit se s rolemi administrátor dat a administrátor databáze, zabezpečením a protiopatřením proti potenciálním druhům nebezpečí.
- Analyzovat popularitu systémů řízení báze dat z hlediska různých členění. Zhodnotit jejich výhody a nevýhody.
- Analyzovat nové trendy v databázích.
- Pomocí nabytých znalostí zvolit vhodnou databázi pro evidenci studentů.

1.2 Cíle praktické části

Hlavní cíl, návrh a realizace databázové aplikace pro evidenci studentů se zaměřením na konfiguraci webového serveru, je rozčleněn do následující podcílů:

- Návrh, realizace a testování datového úložiště žáků evidující návštěvy odpoledních a večerních kurzů taneční školy s využitím vybrané databáze z rešeršní části.
- Analýza a návrh databázové aplikace.
- Konfigurace webového serveru IIS, který hostuje WCF REST služby (Web.config).
- Zabezpečení komunikace mezi klientem a webovým serverem. Vytvoření testovacích certifikátů.
- Autentizace a autorizace databázové aplikace.
- Windows firewall konfigurace.
- Správa instancí služby - podpora více klientů.
- Popis důležitých částí realizace a testování klientské a serverové aplikace.
- Popis vytvořené dokumentace.

Kapitola 2

Analýza databázových technologií, návrhu, správy, architektur a realizace

Databázové technologie a jejich vývoj mají nemalý vliv na dnešní IT svět a můžeme říci, že podstatně ovlivňují dění kolem nás od počátků éry informačních technologií a zasahují do mnoha odvětví (státní správa, ekonomika, školství, zemědělství, stavebnictví a další), aniž by si to (do značné míry) běžný uživatel uvědomoval. S technologií přichází i specifická množina pojmů, která napomáhá lepšímu porozumění. Například na pojem jako je *Databázová technologie* můžeme pohlížet z více stran. Jedna z nich pojednává o realizaci hierarchie jistých abstrakcí, které umožňují jasně oddělit pojmy a odpovídající problémy (tříúrovňová a čtyřúrovňová hierarchie) [36, st. 9]. Druhá zase říká, že se databázová technologie zabývá řízením velkého množství perzistentních, spolehlivých a sdílených dat [35, st. 15].

Málokterá oblast informatiky se může pyšnit tak bohatou historií a praxí jako databáze. Právě praxe je považována za hlavní hnací sílu všech paradigmat či koncepcí, které napomohly vývoji od přístupů pomocí ukazatelů a tabulek až po objektové pojetí databází [35, st. 15]. Databáze jsou většinou součástí rozsáhlých IS a i jejich návrh musí být obdobný návrhu informačních systémů. A jako takové mají i databáze svůj životní cyklus.

Svět dat prochází neustálými změnami, které mají vliv na databázové potřeby, v jejich důsledku vznikají nové technologie, trendy, ale i hrozby. IT svět a samozřejmě také databázový systém čelí množství hrozeb, na které je potřeba reagovat a adekvátně chránit DBS. Velkou hybnou silou jsou velké databázové oblasti (internet a jednotlivé vědy, jako jsou fyzika, biologie, medicína a inženýrství), které vyvolávají potřebu vylepšení starých a vývoje nových technologií [36, st. 175] a také různých metodik a technik, jenž pomůžou vytvořit požadovaný systém.

Postrádal jsem koncepčně vhodný zdroj informací poskytující průřez hlavními databázovými technologiemi, návrhy, postupy a bezpečností, po kterých je čtenář schopen vybrat nejen vhodnou databázi společně se systémem řízení báze dat (SŘBD), ale i ji korektně navrhnout. Do tohoto vytvořeného konceptu jsou vloženy také současné databázové trendy. Naklonil jsem se technologiím Microsoftu, což neznamená, že by text ztrácel na své obecnosti. Závěrem této teoretické části bude zvolen vhodný SŘBD společně s databází pro evidenci studentů.

2.1 Stručný vývoj a historie databázových technologií

Technologie kolem databází dnes zahrnuje více než 40 let bohaté praxe a adekvátního výzkumu, v kterém se odrážely požadavky reálného světa. Počátky databázové technologie představují síťové databáze. V rámci konference CODASYL (1965) byl vytvořen výbor známý jako DBTG, který byl pověřen standardizačním postupem vytvořit koncepci DBS [35, st. 17]. Hierarchické databáze postrádají standard a jeho historie je nedílně spjata s SŘDB IMS (viz sekce 2.5.1).

Za odrazový můstek můžeme považovat 70. léta, kdy se konstituovaly první algoritmy, pojmy, metody, které vytvořili separátní vrstvy (aplikační a datovou) vzájemně se neovlivňující. Za hlavní hnací sílu tohoto vývoje můžeme považovat hromadné zpracování dat v podnicích, kdy bylo snahou vytvořit tzv. ASŘP (automatizovaný systém řízení dat podniku). V dalších 20-ti letech databázový software zaznamenal neobvyklý rozkvět, čehož se zasloužily světoznámé firmy zaměřené na databáze (ORACLE, INFORMIX, SYBASE, IBM a Microsoft), jenž také určovaly databázové trendy [37, st. 3] (viz také sekce 2.9).

V roce 1970 přichází návrh relačního modelu s článkem E. F. Coddda (viz sekce 2.5.3). S rokem 1974 přichází na scénu první verze dotazovacího jazyka SQL, který se stal dominantním prostředkem pro práci s relační databází. Trvalo téměř 10 let, než se relační databáze vyrovnaly z výkonostního hlediska DBS síťovým a hierarchickým protějškům [35, st. 17]. V 90. letech se začaly vyvíjet první objektově orientované databáze (viz sekce 2.5.4), založené na objektovém programování. Výhodou tohoto přístupu k datům je možnost uchování různorodých dat (např. znaková, obrazová a zvuková data). Předpokládalo se, že objekty dokonce měly ve třetím tisíciletí nahradit tabulky, ale tato myšlenka se nepotvrdila [45]. V 90. letech se objevil kompromis v podobě objektově relačních SŘBD, které kombinují vlastnosti relačních SŘBD s přínosem OOSŘBD [35, st. 18]. S rokem 1998 se vyvíjí nativní XML databáze. Souhrn historického vývoje databázových systémů nabízí tabulka D.1.

Za nové trendy v databázích můžeme považovat zjednoduššování správy, vyšší bezpečnost, podporu pro cloud a big data. Analytické aplikace nad da-

tabázemi se musí umět v dnešní době vypořádat s nestrukturovanými daty (NoSQL databáze). Důležité přitom je, aby uživatelé bez potřebných technických znalostí (např. manažeři) mohli zadávat jednodušším způsobem komplexní dotazy. Dnešní moderní společnost vyžaduje maximální dostupnost dat a možnost získat potřebnou rychlou zpětnou vazbu z velkého množství informací. V souvislosti s cloud computingem jsou nyní uplatňovány nové cenové modely, kdy jsou databáze jako služba (DaaS, database as a service) zpoplatněny odpovídajícím způsobem dnešní doby. Příkladem je metoda Pay as you go. Kolik toho uživatel spotřebuje, tolik i zaplatí. Požadavky na bezpečnost stále rostou, díky stále přísnějším regulím, proto se mimo jiné vyvinulo Maskování dat, kdy se data poskytovaná vývojářům a testerům maskují [34]. Tedy přímý vliv jednotlivých odvětví je nevyvratitelný.

2.2 Hlavní terminologie databázových technologií

Tato sekce uvádí hlavní pojmy spojené s touto bakalářskou prací.

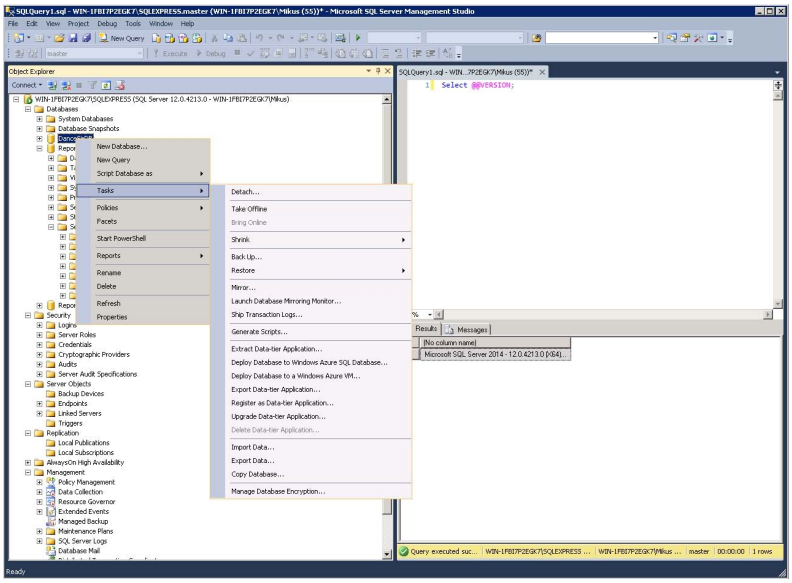
2.2.1 Co je to databáze (DB)?

Databázi můžeme definovat jako množinu záznamů a souborů, které jsou organizovány za určitým účelem. Tento soubor informací se může skládat z různých znaků, čísel, diagramů a dalších objektů. Systematická struktura umožňuje, aby tyto informace mohly být vyhledávány pomocí počítače [44].

2.2.2 Systém řízení báze dat (SŘBD/DBMS)

Systém řízení báze dat (zkráceno na SŘBD nebo podle anglického názvu výše na DBMS). Tento systém, můžeme také říci speciální programové vybavení (souhrn programů), je zodpovědný za centrální správu databáze. První systémy řízení báze dat se objevují v 2. polovině 60. let [37, st. 3].

- Rozdělení vlastností SŘBD
  - **Definice databáze** – specifikace datových typů, struktur a podmínek omezujících data (integritních omezení)
  - **Konstrukce databáze** – proces ukládání dat samotných na vhodné nosiče dat, který je řízen SŘBD
  - **Správa databáze, přístup, manipulace s daty** - dotazovací a vyhledávací funkce, aktualizační operace, generace výstupů
- DBMS podle přístupu k datům
  - RDBMS – Relational DBMS
  - ODBMS – Object DBMS



Obrázek 2.1: Microsoft SQL Server 2014

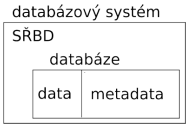
  - ORDBMS – Object-Relational DBMS

- Seznam obsahující příklady některých systémů řízení báze dat
  - Oracle
  - Firebird
  - Microsoft Access
  - Microsoft SQL Server (obr. 2.1)
  - MySQL
  - PostgreSQL
  - SQLite

Občas se pojem SŘBD zaměňuje s pojmem Databázový systém (DBS), který můžeme vyjádřit následujícím vztahem DB + SŘBD = DBS a obrázkem 2.2 [37].

2.2.3 Databázová technologie

Databázová technologie je soubor pojmů, prostředků a technik sloužící pro vytváření informačních systémů (IS). Z obecného pohledu si můžeme představit architekturu IS s databází tak, že data jsou organizována v databázi (DB) a řízena balíkem programů, který se nazývá systém řízení bází dat (SŘBD). Informační systém získává data z DBS přímým přístupem nebo za pomocí



Obrázek 2.2: Struktura DBS

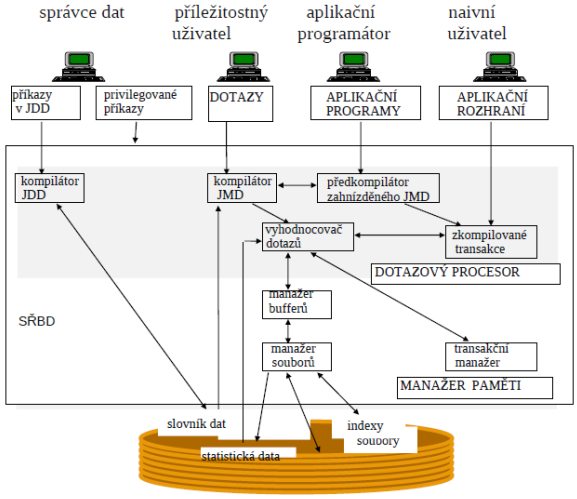
aplikačních programů. K databázové technologii patří techniky transakčního zpracování, řízení souběžného přístupu více uživatelů, řízení zotavení z chyb, řízení utajení dat, zahrnutí jazykových prostředků, odolnost proti chybám, řízení katalogu dat a paměti. Tyto služby poskytuje SŘBD. Do databázové technologie můžeme zahrnout i techniky návrhu databáze [37, st. 5].

- Hlavní přínosy databázové technologie [37]
  - nezávislost dat na (aplikačních) programech
  - efektivní přístup k datům (optimalizace)
  - redukce času potřebného k vývoji aplikace
  - integrita a ochrana dat
  - řízená správa dat a zálohování
  - transakce
  - paralelní přístup více uživatelů
  - zotavení po chybě

2.2.4 Uživatelé DBS

Databázový systém nás vede k tomu, že je zapotřebí rozlišovat jednotlivé uživatelé tohoto systému.

- Uživatelé DBS [37, st. 7]
  - **Správce dat** - často odborný útvar zahrnující osoby odpovědné za autorizovaný přístup do databáze, udělující práva přístupu pro jednotlivé uživatele, sledují využívání databáze, modifikující přístupové cesty a podobně.



Obrázek 2.3: Schéma SŘBD, převzato z [44]

– **Aplikační programátoři** (tvůrce aplikací) - vytvářející uživatelské aplikace.
– **Příležitostní uživatelé** - vyžadují data z databáze v různých, předem nepredikovaných souvislostech (pomocí silnějšího dotazovacího jazyka).
– **Naivní uživatelé** (parametričtí) - v poslední době nejvíce rozšiřující se skupina, která se spokojí s předdefinovanými dotazy, ke kterým přistupují pomocí menu aplikace.

2.2.5 Další databázové pojmy

- Základní pojmy DBS [37]
  – **Entita** je prvek reálného světa (např. pokoj, automobil, člověk).
  – **Atribut dat** je charakteristika, nebo-li vlastnost popisující danou entitu (např. max. rychlost, rok výroby, cena automobilu).
  – **Hodnota dat** je skutečnou hodnotou v každém prvku dat (např. rok výroby 1998).

  - **Obor hodnot** popisuje typ dat (např. rok výroby je datum).
  - **Integritní omezení** jsou tvrzení vymezující korektnost DB. Definují se na konceptuální i databázové úrovní (např. zákazník si může rezervovat nejvýše dva automobily).

2.3 Souborový vs. databázový přístup

Z historie zpracování dat odlišujeme databázový a souborový přístup, který byl v 60. a 70. letech znám jako hromadné zpracování dat (HZD). Následující informace jsou čerpány z [32] a [37, st. 4]. U souborového přístupu jsou data uložena do jednoho nebo více datových souborů (součástí byl i popis), dříve převážně uložených na magnetických páskách. Data byla neodstíněnou součástí aplikace. S tímto přístupem je svázáno mnoho problematik:

- *Redundance a nekonzistence dat.* S rostoucím počtem programátorů pracujících na aplikaci roste pravděpodobnost duplicitních dat a s rostoucí duplicitou se data snadněji stávají nekonzistentními.
- *Obtížný přístup k datům.* Co nový uživatelský požadavek, to nový speciální program. Vznikaly tedy časové prodlevy a uživatel byl značně omezen.
- *Izolovanost dat.* Neexistuje jednotná struktura mezi jednotlivými soubory (formát).
- *Uživatelský přístup.* Aktualizace dat více uživateli může přivést data do nekonzistentního stavu. Další nevýhodou je nemožnost sdílení dat mezi nimi (např. mezi jednotlivými subjekty společnosti).
- *Problém s integritou dat.* Omezená korektnost a celistvost dat. V praxi to znamená kontrolovat vstupní data v aplikačním programu (narůstá složitost programu).
- *Neodstínění programu a dat.* Jednotlivé změny v datech se musí promítnout i do programu.
- *Problém s ochrannou dat.* Každý uživatel by měl mít vhodná příslušná uživatelská práva, a jelikož programy jsou přímo svázány s definicemi souborů, je obtížné data patřičně utajit.
- *Problém s vytvářením vazeb mezi záznamy.* Vše řeší programátor s příslušným programovacím jazykem (např. COBOL, PL/1), tím navyšuje pracnost, rozpočet projektu a čas na něm strávený.

Hlavním cílem vývoje 60. let bylo vyvinout technologii, jenž by odstranila tyto nedostatky. Touto technologií se stal databázový přístup k datům, který

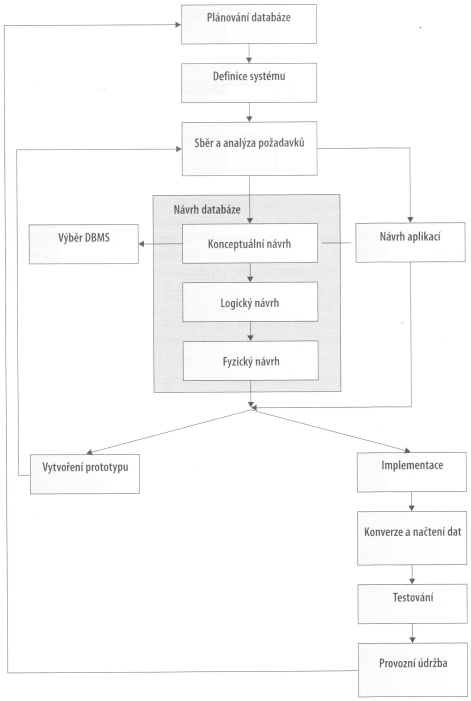
měl odstínit data od uživatelských programů. Vzniká tedy komplikovanější struktura (DBS), která je řízena Systémem řízení báze dat. Tento system poskytuje následující výhody:

- *Zamezení redundance dat.* Správný návrh databáze redukuje redundanci, ale pravděpodobnost dosažení minimální redundance je malá. Některé návrhy například využívají redundanci ke zvýšení výkonu DBS (kontrolovaná redundance).
- *Konsistentnost dat.* Zamezením vzniku redundance klesá i riziko vzniku nekonzistence.
- *Integrita dat.* SŘBD umožňuje zavést integritní omezení, která zabrání vkládání nekorektních a necelistvých dat do DB.
- *Paralelní přístup k datům.* Více uživatelů má možnost paralelního přístupu k datům díky SŘBD. Pokud se nejméně 2 uživatelé současně pokusí aktualizovat stejnou datovou informaci, nemůže tato činnost probíhala náhodně a nekontrolovaně.
- *Ochrana dat.* Data jsou chráněna prostřednictvím ochranného systému SŘBD (autentizace, autorizace, role, práce s právy a další). Jiný přístup k datům není možný.
- *Odstínění aplikace a dat.* Data a aplikační část jsou rozdílnými vrstvami.
- *Jazyk SQL* je standardizovaný strukturovaný dotazovací jazyk.
- *Uživatelsky přívětivější prostředí a lepší práce s větším objemem dat.*

2.4 Životní cyklus vývoje databázového systému

Databáze jsou součástí často komplexních IS a jejich návrh musí být obdobný návrhu pro samotný informační systém. Tato sekce popisuje životní cyklus vývoje databázového systému (DSDLC) pro relační SŘBD, jelikož se stále jedná o nejaktuálnější model dnešní doby. *Jedná se o uspořádaný seznam fází, který popisuje odpovídající techniky a nástroje, jež je třeba použít při vývoji databázového systému* [3, st. 109]. Představa, že fáze DSDLC (2.4) jsou přímočaré, je mylná. Jedná se o logický průchod mezi fázemi s možností opakování předchozích fází, i pro případ problému během cyklu.

Existuje několik technik nazývaných *Techniky nalézání faktů.* Nalézání faktů je formální postup, který zahrnuje například techniky rozhovorů a dotazníků, aby byly nalezeny fakta o systémech, procesech, požadavcích na výkon, preferencích a zabezpečeních [3, st. 125]. Tyto techniky jsou především důležité během počáteční fáze životního cyklu vývoje databázového systému (tzv. před samotným návrhem databáze). Pomáhají zjistit problémy, příležitosti,



Obrázek 2.4: Fáze DSDLC, převzato z [3]

omezení, požadavky a také priority samotné organizace a jejich uživatelů. Nalézání faktů je sice důležitou součástí vývoje, ale při přehnaném zjišťování může být projekt paralyzován analýzou [3, st. 126].

- **Pět obecně rozšířených technik zjišťování faktů [3, st. 127]:**
  - *Prozkoumání dokumentace* poskytuje širší informace o společnosti a umožňuje lépe stanovit sběr požadavků. Nevýhodu je, že přístup k firemním dokumentům nemusí být z různých důvodů možný, jejich množství může být příliš velké nebo můžou obsahovat neaktuální informace. Může se jednat například o e-maily, zápisy z porad, stížnosti zaměstnanců, vyplněné formuláře a další.
  - *Rozhovor* je jednou z nejčastějších technik nalézání faktů. Vyžaduje dobré komunikační vlastnosti osoby vedoucí rozhovor. Technika je velmi časově náročná a nákladná, což nemusí být pravdou v případě videokonference.
  - *Pozorování organizace za provozu* je jednou z nejefektivnějších technik nalézání faktů. Umožňuje zkontrolovat validitu faktů a dat. Vyžaduje přípravu a souhlas pozorovaných lidí. Většinou organizace a pozorovaní lidé začnou vědomě či nevědomě měnit chování, což nemusí odpovídat běžnému provozu.

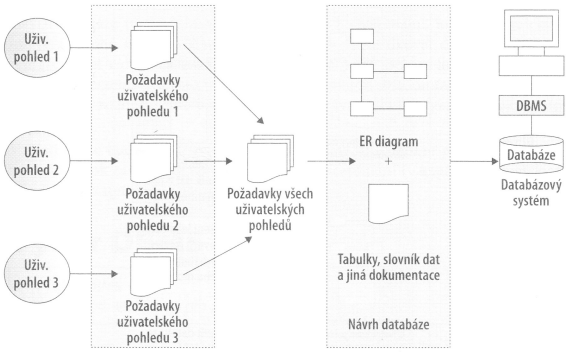
- *Sekundární výzkum* je průzkum aplikací a problémů, které již byly na podobný typ aplikací použity. Existující řešení může podstatně ušetřit čas.
- *Dotazníky* většinou slouží pro sběr informací od většího počtu osob. Je to relativně levný a účinný způsob sběru dat. Odpovědi lze lehce vyhodnocovat pokud jsou dotazníky elektronicky zautomatizovány. Nedoporučuje se papírová forma.

Fáze ***Plánování databáze*** je činnost managementu, které umožní v mezích možností hladký a efektivní průběh jednotlivých fází životního cyklu vývoje databázového systému [3, st. 109]. U této fáze je zapotřebí stanovit hlavní cíl a jednotlivé dílčí cíle databázového projektu (např. vyhledávat údaje o zaměstnancích), které bude databáze podporovat. Součástí by také mělo být odhadnutí rozsahu celkové práce a jednotlivých dílčích kroků. Stanovený rozpočet je důležitou součástí plánování databáze, zásadně ovlivňuje některé fáze DSDLC.

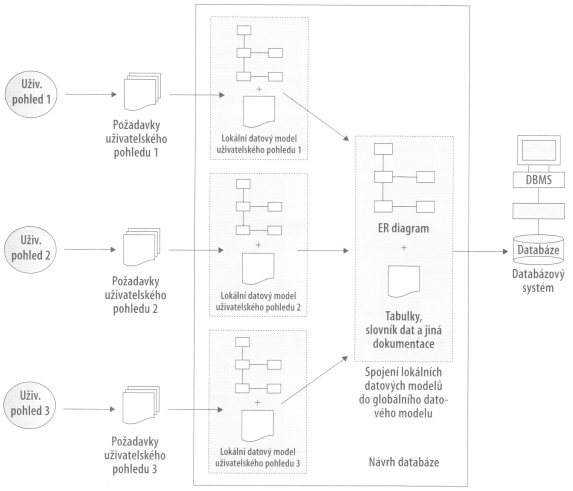
Fáze ***Definice systému*** je definicí hranic databázového systému včetně jeho hlavních uživatelských pohledů [3, st. 111] (zahrnuje pracovní pozice nebo provozní aplikační oblasti), které definují požadavky pro DBS z hlediska konkrétních uživatelů DBS. Jedná se o důležitou část vývoje DBS, protože pohledy chrání před opomenutím či úplným vynechání některé oblasti požadavků na uchovávaná data a transakcí nad těmito daty. Některé uživatelské pohledy můžou být zcela ojedinělé, jiné se zase můžou krýt.

Fáze ***Sběr a analýza požadavků*** je proces sbírání a analýzy informací o organizaci nebo její části, kterou má databázový systém podporovat, a použití těchto informací k určení požadavků na nový databázový systém [3, st. 112]. Informace (např. popis produkovaných dat, včetně toho, jak jsou vytvářena) se sbírají pro každý významný uživatelský pohled. Tyto informace prochází analýzou, aby se mohly určit další funkční požadavky DBS. Určení těchto funkčních požadavků je kritickým místem celé realizace DBS, jelikož systém s neúplnou či neadekvátní funkčností bude uživatele obtěžovat, což může vést k nepoužívání systému a ztrátě peněz. Při zpracovávání vícenásobných uživatelských pohledů existují tři přístupy [3, st. 113-114]:

- *Centralizovaný přístup* je vidět na obrázku 2.5. Požadavky jednotlivých uživatelských pohledů 1 až 3 jsou spojeny do jednoho seznamu požadavků na nový DBS. Datový model reprezentující všechny uživatelské pohledy se realizuje až během fáze návrhu databáze. K tomuto přístupu se uchyluje tehdy, pokud se vícero uživatelských pohledů překrývá a DBS není příliš složitý.
- *Přístup integrace pohledů* vyobrazuje obrázek 2.6. Pro požadavky jednotlivých uživatelských pohledů jsou vytvořeny separátní lokální datové modely, které jsou během návrhu databáze spojeny v jeden globální



Obrázek 2.5: Vícenásobné už. pohledy - centralizovaný přístup, převzato z [3]



Obrázek 2.6: Vícenásobné už. pohledy - přístup integrace pohledů, převzato z [3]

datový model reprezentující všechny uživatelské pohledy v organizaci. K tomuto přístupu se uchyluje v přídě nehomogenních uživatelských pohledů a složitých DBS. Výhodou je separace velkého problému na menší části.

- *Kombinace obou přístupů* je vhodná při návrhu složitějších DBS, kde se dva nebo více uživatelských pohledů může spojit v jeden seznam požadavků a potom použít k vytvoření lokálního datového modelu.

Fáze ***Návrh databáze*** je realizována různými modely systému na základě dříve definovaných požadavků. Jsou vytvářeny modely nejen na úrovni *konceptuální* a *logické*, ale také *fyzický návrh* jako příprava pro implementaci. Během konceptuálního návrhu databáze se snažíme nalézt objekty a relace

mezi nimi, jenž jsou potřeba reprezentovat v databázi. Logický návrh databáze „promítne“ objekty a jejich relace do množiny tabulek. Během fyzického návrhu databáze se rozhoduje o tom, jak je třeba tyto tabulky fyzicky implementovat v cílovém SŘBD. Tyto tři fáze zaslouží pozornost čtenáře, ale tato bakalářská práce je nebude podrobněji rozebírat, jelikož podrobnější zkoumání této problematiky přesahuje teoretický rozsah této práce. V případě zájmu doporučuji následující literaturu [38].

Fáze ***Výběr SŘBD***, jejíž cílem je výběr vhodného systému řízení báze dat, který splňuje současné/budoucí požadavky, má například dostatečně velkou uživatelskou komunitu (lépe se hledají řešení problémů), zohledňuje zkušenosti zaměstnanců (nová školení stojí peníze) a další. Musí být také přihlédnuto k finančním možnostem organizace. Jedním z cílů této práce byla pomoc s výběrem SŘBD. Tuto problematiku analyzuje sekce 2.8.

Fáze ***Návrh aplikací*** zahrnuje návrh uživatelského rozhraní a databázových aplikací, které budou používat a zpracovávat databázi. Na obrázku 2.4 jde vidět, že návrh databáze a aplikací probíhá v DSDLC paralelně. Veškeré funkční požadavky by měly být pokryty návrhem pro databázovou aplikaci.

Fáze ***Vytvoření prototypu databáze*** aneb vytvoření fungujícího modelu databázového systému. Prototyp je fungující model, který nedosahuje takové funkcionality a vlastností jako finální verze systému. Umožňuje uživateli vidět a zhodnotit „demo systém“. Pomocí poskytnutých informací se můžou vyjasnit nebo vzniknout nové požadavky na systém. Prototyp by neměl výrazně zatížit rozpočet celého projektu. Existují dvě strategie prototypů – *Prototyp pro požadavky* pro určení požadavků, který je následně odstraněn nebo není použit jako finální DBS, což by byl hlavní aspekt *Vývojového prototypu.*

Fáze ***Implementace*** je fyzická realizace databáze a návrhu aplikací. Jedná se o první krok po dokončení návrhu. Implementace databáze se provede pomocí DDL nebo GUI (2.5.3) zvoleného SŘBD. Databázové aplikace se doporučuje implementovat pomocí jazyků třetí nebo čtvrté generace (3GL nebo 4GL) a jejich databázové transakce pomocí DML (2.5.3) cílového SŘBD. V této fázi se také implementuje zabezpečení a kontrola integrity pro aplikace [38, st. 119].

Fáze ***Konverze a načtení dat*** jsou nutné pouze tehdy pokud se požaduje zachovat databázová data ze starého systému. V dnešní době můžeme najít SŘBD standardně s utilitou pro konverzi dat. Fáze ***Testování*** zkouší výkonnost, zabezpečení a integritu dat. Po fázi testování je DBS připraven na „ostrý provoz“.

- **Kroky fáze Testování** [32, st. 14]:
  - *Testování výkonnosti* se testuje za různých podmínek, aby bylo možné zjistit chování databáze při stresových situacích (velké množství operací, více souběžných spojení).

- *Testování zabezpečení* testuje, zda uživatelé mají přístup pouze k datům či mají omezená práva nad daty.
- *Testování integrity dat* testuje logické nedostatky, které může databáze obsahovat a jejichž důsledkem jsou ztráty dat nebo jejich nepřesnosti.
- *Vyladění parametrů nebo změna logického či fyzického návrhu v závislosti na výsledcích testů* .

Fáze ***Provozní údržba*** je proces monitorování a údržby po instalaci systému. Jedná se například o optimalizaci tabulek, správa uživatelů a jejich hesel, zálohování a obnovování dat nebo také implementace nových požadavků.

2.5 Analýza hlavních databázových (logických) modelů

Z hlediska způsobu ukládání dat do databáze, jejich organizace, vazeb mezi nimi a způsobu jejich manipulace rozlišujeme dvě velké skupiny. Historicky starší skupinou jsou souborově orientované modely hierarchický, síťový, relační, objektový model a mnoho dalších, z nichž relační a objektový model jsou nejčastěji používanými, jelikož odstraňují většinu nedostatků ostatních modelů. Datovým modelem rozumíme vlastnosti vazeb mezi údaji v databázi a způsob organizace jejich struktury, která prezentuje logickou úroveň SŘBD. *Databázový model* není spjat s běžným pojmem model. Jde o nástroj pro modelování. Takovým nástojem je například *relační datový model*, který umožňuje popisovat a konstruovat relace. Výsledkem není žádný model, nýbrž schéma relační databáze [35, st. 43].

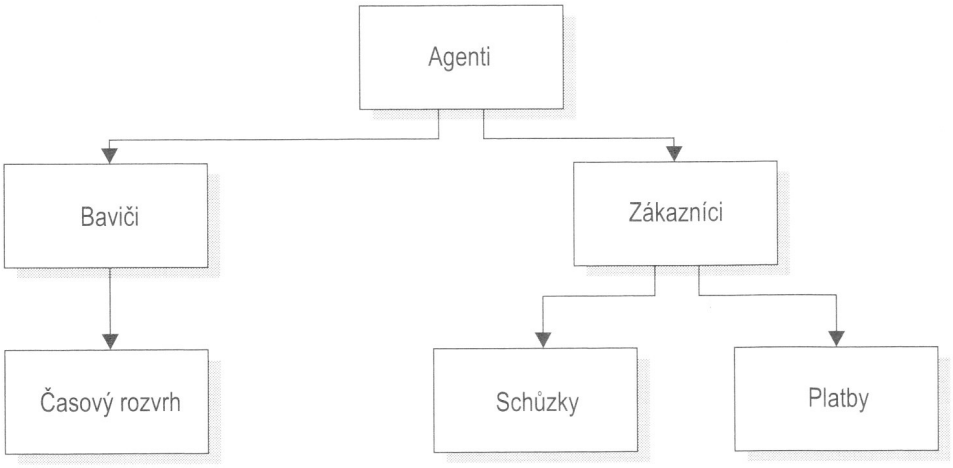
Ještě je potřeba zmínit, že v používané terminologie nerozlišujeme mezi pojmy *Databázový model* a *Datový model*, i když pojem „datový“ je chápán více obecněji. Je však vhodnější hovořit o databázových modelech [35, st. 44].

Tato sekce analyzuje informace z následujících zdrojů a to především knižních [37], [36], [38], [35], [8], [3], [9], [7], [12] a [40].

2.5.1 Hierarchický databázový model

Hierarchický databázový model (obr. 2.7) vzniká koncem 60. let. Data jsou setříděna hierarchicky za použití stromové datové struktury a vztahu „parent-child“ pro ukládání dat. Tento model umožňuje vyjádřit jednosměrné vztahy typu 1:N ve směru shora dolů. Každý záznam představuje uzel ve stromové struktuře se vztahem typu *Rodič/Potomek*. Za vlastníka je považován rodič a za člena jeho potomek, který může mít jen jednoho rodiče (problém duplikace prvků).

V praxi se tento model využívá tam, kde i samotná realita má hierarchickou strukturu (organizační nebo skladové systémy). Nejznámější implementací



Obrázek 2.7: Diagram hierarchického databázového modelu, převzato z [8]

toho modelu byl Information Management System (IMS) pro skladové hospodářství projektu Apollo (60. léta), kde figurovaly společnosti IBM a NAA. Vyhledávání dat v této struktuře probíhá přes záznamy směrem dolů (potomek), nahoru (rodič) a nebo do stran (sourozenec), což přináší i mnoho nevýhod:

- *Redundantní data a cyklické vztahy* jsou způsobeny virtuálními záznamy, které řeší vztahy typu M:N.
- *Složité operace vkládání a rušení záznamů.* Procedurální manipulace s daty (sekvenční průchod stromu nebo jeho větví) nutí v případě změny (přidání nebo odebrání položky) přepracování celé struktury databáze.
- *Složitá implementace* oproti návrhu.
- *Potřebná znalost struktury databáze* pro přístup k datům.

Za případné výhody můžeme považovat:

- *Jednoduchost* návrhu (stromová datová struktura).
- *Datová integrita* díky vztahu *Rodič/Potomek*. V tomto typu vztahu může být tabulka rodiče přidružena k jedné, nebo více tabulkám potomků, ale tabulka potomka může být přidružena pouze k jedné tabulce rodiče. Vymazání rodiče způsobí vymazání všech propojených záznamů v tabulkách potomků.

Tento typ databází byl hojně využíván pro ukládání na kazetopáskové jednotky, které byly používány sálovými počítači (70. léta). Ani výhody rychlosti přímého přístupu k datům nezabránily vzniku nového modelu, který řeší redundanci dat a potlačuje komplexnost vztahů mezi daty.

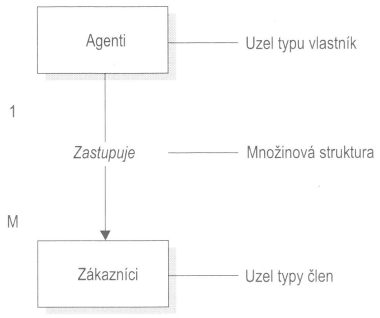
2.5.2 Síťový databázový model

Síťový databázový model (obr. 2.9) vzniká v 60. letech 20. století. Data jsou v modelu logicky i fyzicky uspořádána jako uzly rovinného grafu, kde každý záznam může mít vztahovou vazbu s libovolným počtem dalších záznamů. V souvislosti s tímto modelem jsou spjaty pojmy a vlastnosti jako:

- *Záznam/Uzel (Record).* Jedná se o název entitního typu (po transformaci E-R schématu do síťového).
- *Množina/Množinová struktura (Set).* Množinové typy. Jeden set reprezentuje vztah 1:N (po transformaci E-R schématu do síťového). Je definován pomocí svého vlastníka a členů.
- *Vlatník (OWNER).* Typ záznamu R se nazývá OWNER setu S.
- *Člen (MEMBER).* T je MEMBER setu S.
- Jeden záznam v uzlu člen je ve vztahu pouze k jednomu záznamu v uzlu typu vlastník.
- Záznam v uzlu typu člen navíc nemůže existovat, aniž by byl ve vztahu k nějakému záznamu v odpovídajícím uzlu typu vlastník. (např. klient musí být přiřazen k agentovi, ale agent bez klientů může v databázi být)
- Mezi dvěma uzly může být definována jedna, nebo více množin (spojení), a libovolný uzel může být součástí dalších množin s jinými uzly v databázi.

V podstatě se jedná o zobecněný hierarchický model s mnohonásobnými vztahy (sety), které mohou být lineární i cyklické, tedy lépe popisující objekty a vztahy reálného světa (vztahy M:N). Uzel reprezentuje soubor záznamů a množinová struktura reprezentuje a zřizuje vztah v síťové databázi (viz diagram jednoduché množinové struktury 2.8). Vyhledávání se podstatně ulehčilo oproti hierarchického modelu (začínalo se z kořenové tabulky), kde záznam je možno nalézt pomocí klíče a posunu (např. stranou na dalšího potomka v setu či nahoru z potomka na jeho rodiče v jiném setu). Síťový databázový model přináší i nevýhody:

- *Omezení ve změnách struktury databáze.* Po změně struktury je potřeba složitě přepracovávat celý databázový model a promítnout změny do aplikace.
- *Složitost* struktury databáze kvůli systému ukazatelů, které vyžadují velké množství úprav při mazání, vložení nebo aktualizaci jakéhokoliv záznamu.



Obrázek 2.8: Základní množinová struktura, převzato z [8]

- *Potřebná znalost struktury databáze* pro práci s množinovými strukturami.

Kvůli úspěchu relačního databázového modelu v polovině 70. let neměl síťový model mnoho příležitostí se prosadit významně na trhu, i když může nabídnout následující výhody (především oproti předešlému modelu):

- *Jednoduchost* návrhu jako u hierarchického modelu.
- *Více vztahů mezi záznamy* vyjadřující lépe reálný svět (1:1, 1:N, M:N a rekurzivní). Síťový model zavádí pouze vztahy typů 1:1 a 1:N a to mezi dvěma typy záznamů R (OWNER) a T (MEMBER). Ostatní typy vztahů jsou definovány na konceptuální úrovni.
- *Lepší přístup k datům* je podstatně jednodušší a flexibilnější než u hierarchického modelu.
- *Datová integrita.* Síťový databázový model nedovoluje existenci členům bez vlastníka.
- *Lepší datová nezávislost.* Oproti hierarchického modelu jsou fyzická data lépe odstíněna od samotných aplikací.
- *Rychlý přístup k datům.* Umožňuje uživatelům vytvářet dotazy, které jsou mnohem komplexnější než dotazy v hierarchickém modelu.

Používání hierarchického a síťového modelu je stále méně obvyklé a uvádí se spíše z historických důvodů. Přestože síťový model byl posunem kupředu, tak se stále věřilo, že musí existovat lepší způsob, jak zpracovávat a udržovat velké množství dat. Tím novým způsobem byl *Relační databázový model.*

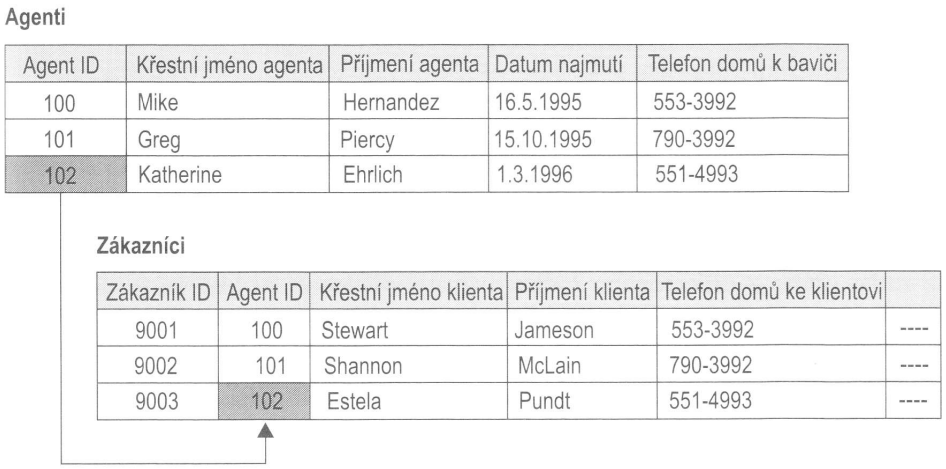
Obrázek 2.9: Diagram síťového databázového modelu, převzato z [8]

2.5.3 Relační databázový model

Relační databázový model vzniká začátkem 70. let 20. století a byl představen pracovníkem firmy IBM - Dr. E. F. Codd. Definoval také jazyk pro vyhledávání informací a manipulaci s daty. Jedná se o databázi založené na pevných matematických základech relačního kalkulu a relační algebry. Relační databázový model je nejrozšířenějším způsobem uložení dat v databázi a to i díky svým výhodám:

- *Jednoduchá struktura*, kdy jsou data organizována v tabulkách, které se skládají z řádků a sloupců.
- *Flexibilita*, kdy informace z rozdílných tabulek můžou být lehce spojeny či extrahovány do vyžadované formy.
- *Transparentnost při manipulaci s daty* do tohoto modelu přináší relační kalkul a relační algebra. Nezajímáme se o přístupové mechanizmy k datům obsažených v relacích.
- *Tabulky lépe popisují část reálného světa* (lépe na tom jsou objektové modely).
- *Omezení redundance dat v relační databázi* díky normalizacím, minimalizující strukturu uloženou na disku.
- *Lepší manipulace s daty* prostřednictvím SQL jazyka.

Základem relačních databází jsou databázové tabulky, které jsou vzájemně propojeny logickými vazbami (tzv. *relacemi* viz obr. 2.10). *Entitou* je nazývána



Obrázek 2.10: Relační databázový model - vztah mezi tabulkami, převzato z [8]

konkretní tabulka, které je chápana jako reálný objekt (např. oddělení, fakulta, škola, auto, ...). Tabulku tvoří řádky (označovány jako n-tice, které sdružují n hodnot tabulky) a sloupce, které popisují vlastnosti (*atributy*) této entity (např. výška, věk, datum splatnosti, ...). Sloupec (atribut) obsahuje hodnoty určitého datového typu, kde obor těchto hodnot se nazývá *doména*. Hodnoty v jednom nebo více sloupcích, které jednoznačně identifikují řádky mezi sebou, se nazývají *primární klíče*. *Cizí klíč* definuje vztah mezi dvěma tabulkami, kde jeho hodnota je obsažena v druhé tabulce v primárním klíči.

S časem se u tohoto modelu a celého relačního přístupu objevil zásadní nedostatek - není plně kompatibilní s objekty a při převodu je potřeba provádět transformace (programátoři stráví 25% času kódování mapováním objektového programu do databáze), což vytvořilo prostor pro objektové databáze. Mezi další nevýhody patří:

- *Scházející podpora struktur* jako jsou pole nebo kolekce.
- *Náročná implementace dědičnosti* pomocí různých mapování na jednu či více tabulek, buď to v celku nebo po částech.

Především nedostačující výkon tehdejších počítačů „bránit“ většímu rozšíření relačního datového modelu, který je náročnější na provádění dotazů než síťový datový model. Z tohoto důvodu může být použití síťového modelu pro některý typ aplikací vhodnější volbou. Velký podíl na rozšíření tohoto modelu můžeme připsat především firmě Oracle.

SQL je dotazovacím jazykem pro relační databáze a skládá se z několika typů jazyků, které jsou potřebné ke správnému fungování databáze. Zde uvádím hlavní z nich:

- *Jazyk pro definici dat (JDD/DDL)* vytváří definici všech potřebných uživatelských dat v aplikaci (logické schéma databáze). Příkazy (např. CREATE, ALTER, DROP) vytvářejí, upravují, doplňují a mažou struktury databáze (tabulky, indexy, pohledy atd.).
- *Jazyk pro manipulaci s daty (JMD/DML)* se používá jak k aktualizaci dat (příkazy INSERT, UPDATE, DELETE), tak i k výběru dat (SELECT) podle daných požadavků.
- *Jazyky pro ovládání dat (DCL) a transakcí (TCL)* nastavují přístupová práva a řídí transakce.
- *Jazyk pro ukládání tabulek (SDL)* a *jazyk pro definici pohledu (VLD)*, určující vytváření pohledů.

Dnešní relační systémy používají jako základní jazyk SQL, který zahrnuje výše zmíněné jazyky [37, st. 6]. Pro objektový model může být jako příklad OQL a pro XML model Xpath nebo XQuery.

2.5.4 Objektový databázový model

Přestože jsou RDBMS v praxi rozšířeny pro běžné komerční aplikace (např. inventáře, správa hostů, zpracování objednávek, ...), tak nedisponují dostatečným aparátem pro takové aplikace jako CAD (Computer-aided design), geografické systémy (GIS) a systémy pro ukládání multimediálních dat. Nástup objektově orientovaných a objektově-relačních databází byla jen otázka času.

Prvně zmíněný objektově orientovaný model (OODM) zachycuje sémantiku objektů podporovaných v objektově orientovaném programování. Neexistuje všeobecně přijímaný OODM. Na rozdíl od teoreticky podloženého RDM, nemá ODM žádný teoretický základ, což zapříčinilo, že neexistuje jednotná definice pro tento model. Existuje však Mezinárodní sdružení firem OMG, které vydalo dokument „Request for Technology for Object Query Services“ obsahující specifikaci abstraktního jazyka OQL. Je třeba zdůraznit, že OMG není standardizující orgán jako ANSI, ale spíše poradní a certifikační skupina. S tímto modelem se pojí i některé mýty [12, st. 77-82]:

- *Objekty = grafika nebo zvuková stopa.* Datový model databáze totiž s typem uložených dat teoreticky vůbec nesouvisí, můžeme si však vybrat vhodnější model pro konkrétní typy dat. Existují relační databáze s obrázky a objektově orientované databáze jen s textovými a číselnými typy.
- *BLOBs a Triggers = objekty.* Mylná představa spočívá v některých rozšířeních relačních databází. BLOBs nejsou rovnocenné objektům, jelikož nemůže například vyselektovat zvukové záznamy (BLOBs) delší než 1 minutu. Triggery spravují chování dat, ale nemají takovou těsnou vazbu

s daty jako objekty a jejich metody, u kterých může být využita dědičnost nebo polymorfismus.

- *Podpora SQL = správná databáze.* Samotný SQL jazyk mnohdy nestačí k vybudování celé programové aplikace a musí se kombinovat s jinými programovacími jazyky (C nebo 4GL), čímž vznikají impedanční problémy mezi programovacím jazykem a dotazovacím jazykem. Pro objektové databáze existují SQL rozšíření (SQL3, Object SQL, OQL, ...), ale mnohem lépe vyhovují standardizované objektové programovací jazyky jako třeba Smalltalk.

- *Objekty = nemožnost transakce a obnovy obsahu DB.* Dnes již tento mýtus neplatí a dokonce můžete uprostřed transakce počítač vypnout a pokračovat druhý den. Rollback u některých databází dokonce umožňuje práci s několika časovými verzemi jednoho a téhož objektu.

- *Objektové databáze neumožňují dotazování.* Toto tvrzení vyvrací možnost navigačního přístupu k objektům (tzv. „*browsing*“).

Je potřeba zmínit dvě „hnutí“. První z nich jsou konzervativní zastánci relačního modelu, kteří jsou přesvědčení, že jeho rozšíření o objektový přístup je postačující. Druhá strana je naopak přesvědčena, že základní relační model je neadekvátním řešením pro aplikace, které mají zcela jiné požadavky než tradiční obchodní aplikace (viz [3, st. 430-432]):

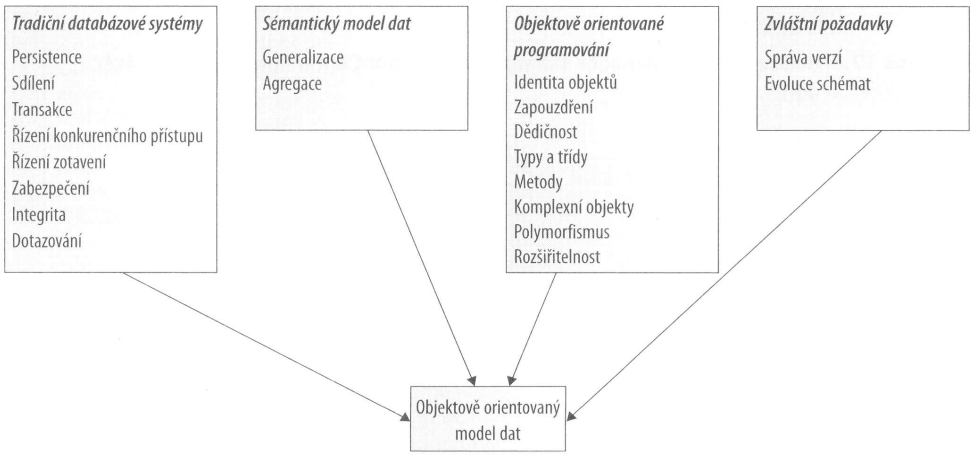
- *Interaktivní a dynamické webové stránky* (např. 3D zobrazení, doprovodný audio komentář)

- *Kancelářské informační systémy a multimediální systémy* (např. diagramy, náčrty, videosekvence).

- *Geografické informační systémy* (např. rozsáhlé satelitní fotografie).

- *Počítačově podporovaný návrh softwaru* (např. správa verzí).

V určité fázi integrace OODBMS (spravuje OODB) někteří analytici byli toho názoru, že trh s touto technologií poroste tempem 50 % ročně, tedy rychleji než samotný trh databází, ale těchto prognostik nebylo dosaženo. Situace, že by OODBMS překonaly prodej relačních systémů je málo pravděpodobná, jelikož mnoho významných oborů podnikání nadále považuje relační DBMS za přijatelné [3, st. 429]. Dovolím si přirovnat relační databáze a její vývoj k ropě a vývoji automobilového průmyslu, kdy do vývoje bylo vloženo tolik prostředků a peněz, že nalezení lepší technologie, která by byla „protlačena“ na trh, je noční můrou pro mnoho korporací a raději by vynaložily další miliony dolarů, jen aby zachovaly stávající „mega-systémy“. Ve své podstatě peníze a moc v určitém smyslu drží vědu zkrátka, což je obrovská škoda.



Obrázek 2.11: Oblasti inspirací OODM, převzato z [3]

Koncepty objektově orientovaných modelů čerpají z několika různých oblastí, jak poukazuje obrázek 2.11.

Několik významných dodavatelů DBMS dalo vzniknout skupině Object Data Management Group (ODMG), aby definovala standardy pro OODBMS. Tato skupina také definovala objektový dotazovací model (OQL), který poskytuje deklarativní přístup k objektům databáze pomocí syntaxe podobné SQL. Ve stručnosti za výhody a nevýhody OODBMS můžeme považovat [3, st. 455]:

- **Výhody:**
  - *Rozšířené modelovací schopnosti.*
  - *Rozšiřitelnost* (je tady možné vytvářet nové datové typy).
  - *Odstranění impedančního nesouladu.*
  - *Možnost použití v pokročilých databázových aplikacích.*
  - *Podpora evoluce schémat.*
  - *Podpora pro dlouho trvající transakce.*
  - *Výrazně bohatší dotazovací jazyk.*
  - *Lepší výkonnost.*
- **Nevýhody:**
  - *Neexistence univerzálního modelu dat.*
  - *Nedostatek standardů.*
  - *Nedostatek zkušeností.*

- *Konkurence ze strany RDBMS/ORDBMS.*
- *Složitost.*
- *Slabá podpora pohledů a zabezpečení.*

2.5.5 Objektově-relační databázový model

Koncem 90. let se zdálo, že vývoj DBMS se může uchylovat jen dvěma cestami, RDBMS a OODBMS, ale objevila se střední cesta hybridního vývoje. Tlak na dodavatele RDBMS ze strany OODBMS byl značný. Uvědomovali si výhody OODBMS a věděli, že pokud chtějí nějakým způsobem udržet RDBMS v praxi, bude potřeba adekvátních „záplat". Při pohledu na trendy (obr. 2.11) objevujících se u pokročilých databází je zřejmý značných vliv objektově orientovaných vlastností (třídy, zapouzdření, dědičnost, polymorfismus, ...). Tedy hlavně o tyto vlastnosti bylo potřeba rozšířit RDBMS. Vzniklo tedy mnoho rozšíření, ale každé s jinou kombinací vlastností, což zapříčinilo, že neexistuje jediný univerzální vzor pro rozšíření RDBMS. ORDBMS tří hlavních databázových společností - Oracle, Microsoft a IBM - se také poněkud liší. Odpovídající datové modely mají mnoho zcela protichůdných rysů.

Standard SQL se samozřejmě zachoval, jen se od roku 1991 začalo pracovat na objektovém rozšíření standardu, kde se tato rozšíření stala součástí verze SQL:1999. Výhody a nevýhody ORDBMS jsou následující [3, st. 447]:

- **Výhody:**
  - *Opakovaná použitelnost a sdílení.* Opakovaná použitelnost je výsledkem rozšíření DBMS serveru, takže je standardní funkčnost poskytována centrálně, namísto toho, aby museli být implementovány v každé aplikaci.
  - *Zvýšení produktivity* jak pro vývojáře, tak také pro koncové uživatele.
  - *Využití zkušeností s vývojem RDBMS.* Další výhodou plynoucí z rozšíření relačního přístupu je ochrana významných oblastí znalostí, zkušeností a vývoje RDBMS. Je to podstatná výhoda pro organizace, kterým tento způsob umožňuje využít nová rozšíření, aniž by ztratily výhody plynoucí z aktuální databáze.
- **Nevýhody:**
  - *Složitost a související zvýšené náklady.* Zastánci relačního přístupu mají pochybnosti ohledně jeho jednoduchosti a „čistoty" po rozšíření. Nevěří, že by zásady relačního modelu byly poté zachovány.
  - *Klamná prezentace DBMS.* Dodavatelé ORDBMS se však snaží prezentovat objektové modely jako rozšíření relačního modelu, ale objektové aplikace nejsou tak zaměřeny na data jako aplikace relační a lépe vystihují „reálný svět".

2.5.6 Další databázové modely

Nejen aplikace, ale i samotná data se mění a jejich množství v posledních letech výrazně roste. Stávající modely jsou z různých důvodu méně vhodné a musí se nacházet nové alternativy (viz také sekce 2.9). Zajímavou alternativou dnešní doby je oblast kolem termínu NoSQL (viz 2.9.1). Jedna kategorie NoSQL databází jsou tzv. grafové databáze, které se v posledních letech těší obrovské popularitě [38].

Model založený na XML dokumentech se zase například hodí pro integraci databází napříč různými technologiemi a platformami. Podobá se hierarchickému modelu, protože XML dokument je chápán jako strom (DOM). XML data představují semistrukturovanou formu dat. Pro dotazování se používají silné a standardizované jazyky XPath nebo XQuery [3, st. 381].

2.6 Architektura DBS

Tato sekce analyzuje možnosti architektur databázových systémů. Při návrhu a realizaci databáze musíme brát v úvahu jednotlivé typy architektur DBS a jejich technickou a finanční náročnost, typ navrhované databáze, počet uživatelů a typ zpracovávaných informací.

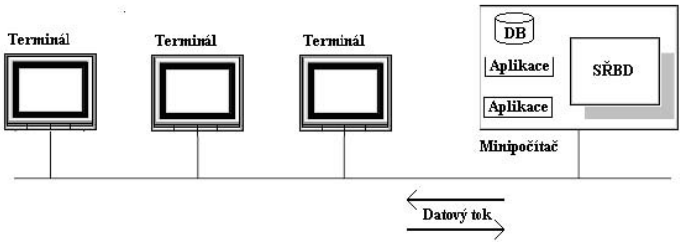
2.6.1 Jednovrstvá architektura

Jednovrstvou, nebo-li centralizovanou architekturu (obr. 2.12), můžeme spíše označit za zastaralou. Databáze (např. desktopové DB - Microsoft Access, Filemaker Pro, Alpha Five, Paradox, Lotus Approach) a DBMS jsou společně na centrálním počítači, kde také dochází ke zpracování požadavků a vstupních dat. Komunikaci (uživatele a centrálního počítače) a zobrazení výsledků požadavků zajišťuje pouze terminál. Tato architektura je vhodná pro databázové aplikace bez sdílení dat mezi více uživateli [32, st. 113].

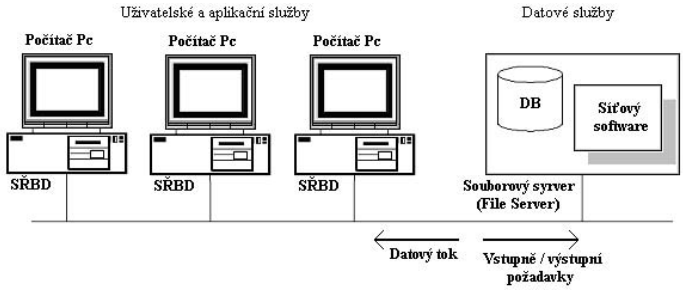
- **Výhody:**
  - *Rychlost.* Lokálně uložená data se nepřenášejí po síti mimo centrální počítač, tedy nevyžadují separátní databázový server.
- **Nevýhody:**
  - *Omezené množství dat.*
  - *Omezený počet uživatelů.*

2.6.2 Dvouvrstvá architektura

Do této architektury spadají dvě podskupiny *Architektura File-Server* a *Architektura Klient-server*, které se liší podle umístění výkonu spojeným s aplikačními službami klienta.



Obrázek 2.12: Centralizovaná architektura DBS, převzato z [32]



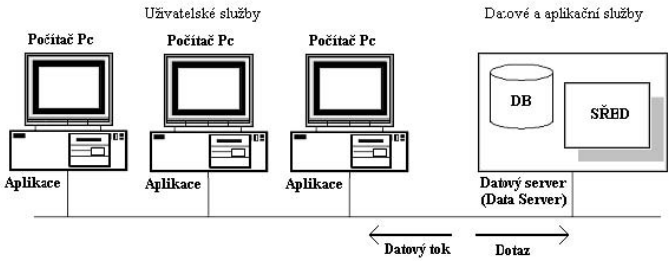
Obrázek 2.13: Architektura File-Server, převzato z [32]

2.6.2.1 Architektura File-Server

U *architektury File-Server* (obr. 2.13) je DB umístěna na souborovém serveru. Prostřednictvím síťového software a jednotlivých SŘBD (DBMS), umístěných na uživatelských počítačích, probíhá datová komunikace. Je potřeba zajistit ochranu záznamů, jelikož architektura umožňuje přistupovat více uživatelským aplikacím (SŘBD) najednou.

- Popis komunikace mezi souborovým serverem a uživatelským PC [32, st. 114]:
  - Uživatel vytvoří dotaz.
  - SŘBD zpracuje dotaz a odešle konkrétní datový požadavek na DB.
  - File-Server odešle bloky dat na lokální uživatelský počítač, kde SŘBD data dále zpracuje.
  - Výsledky se uloží na PC, zobrazí na monitoru nebo se vytisknou jako sestava.

V polovině 90. let začaly vznikat sofistikovanější aplikace, které mohly být potenciálně rozmístěny až k tisícům konečných uživatelů, ale objevil se problém ze strany tzv. *Tlustného klienta*, který vyžadoval podstatné zdroje v klientském počítači pro efektivní běh (zdroje včetně diskového prostoru, RAM, výkonu CPU) a mnohdy odbornější administrativní činnost [3, st. 42].



Obrázek 2.14: Architektura Klient-Server, převzato z [32]

2.6.2.2 Architektura Klient-Server

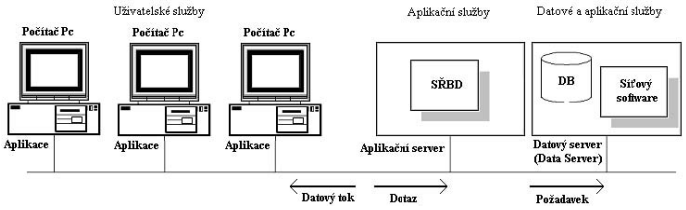
U *architektury Klient-Server* (obr. 2.14) běží na počítačích aplikace, které mají na starost uživatelské rozhraní a hlavní logiku provozu a zpracování dat. Datový server je zodpovědný za validaci dat a přístupu k databázi, což z něho dělá nejvytíženější vrstvu architektury.

- Popis komunikace mezi databázovým/datovým serverem a aplikací na počítači [32, st. 115]:
  - Aplikace na počítači formuluje dotaz nebo požadavek na data pomocí strukturovaného jazyka (většinou v podobě SQL dotazu) a odešle jej na server.
  - Server zpracuje dotaz.
  - Výsledek dotazu posléze odešle do počítače, kde jej aplikace převede do výstupní podoby.

Uživatel disponuje pouze uživatelskými službami a přijímá pouze požadované informace od datového serveru. Aplikační a datové služby probíhají na serveru. Klient se podstatně „odlehčil“, uživatel je tzv. *Tenký klient*. Tato architektura snížila tok dat pohybující se sítí, protože dotazy jsou prováděny přímo na databázovém serveru a na počítač jsou posílány pouze výsledky. Zvýšila se také výkonnost klienta, díky lepšímu rozložení zpracování záznamů.

2.6.3 Třívrstvá (vícevrstvá) architektura

U *třívrstvé (vícevrstvé) architektury* (obr. 2.15), stejně jako u *architektury Klient-Server*, výkon spojený s aplikačními službami je soustředěn na databázovém serveru společně s validací dat a přístupem do databáze. Klient pouze poskytuje uživatelské rozhraní, přičemž logika zpracování dat a provozu (aplikační služby) jsou umístěné na separátním aplikačním serveru nebo na stejném serveru. Rozložení provozní zátěže na vícero serverů získává architektura vyšší úroveň stability a případnou modularitu, což usnadňuje změnu nebo nahrazení jedné vrstvy bez ovlivnění ostatních vrstev. Další výhodu je, že třívrstvá



Obrázek 2.15: Vícevrstvá architektura, převzato z [32]

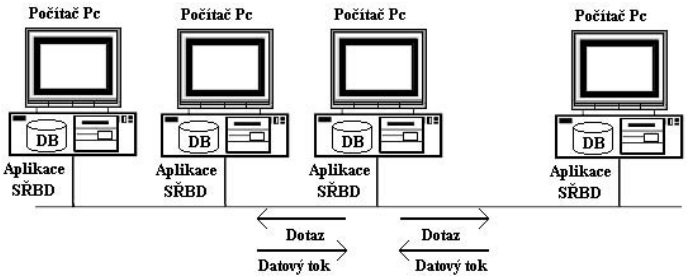
architektura celkem přirozeně odpovídá prostředí webu, kdy tenký klient odpovídá webovému prohlížeči a aplikační server odpovídá webovému serveru [3, st. 43].

2.6.4 Architektura distribuovaných databázových systémů

Při použití této architektury jsou data z DB rozložena v několika počítačích (obr. 2.16), což uživatel neregistruje, jelikož se navenek jeví jako jediná celistvá databáze. Každý počítač hostuje také lokální SŘBD společně s kopií distribuované SŘBD z globálního SŘBD. Distribuovaný/globální SŘBD eviduje rozmístění dat, zajišťuje převod požadavků, referenční integritu a řízení přístupu k datům. Lokální SŘBD pracuje s daty umístěnými v konkrétní části databáze a vytváří exportní schéma, které definuje data sdílená s jinými uživateli [32, st. 117].

- Tři základní vlastnosti distribuovaných DBS [32, st. 116]:
  - *Transparentnost.* Klient nezaznamená, že DB je rozdistribuována a uživatelské příkazy nemusí být přizpůsobeny zvlášť pro lokální a vzdálená data - o vše se stará SŘBD.
  - *Autonomnost.* S každou lokální DB zapojenou do distribuovaného DBS je možno pracovat nezávisle na ostatních DB (odolnost proti výpadkům).
  - *Nezávislost na typu sítě.* Architektura podporuje různé typy sítí, lokálních i globálních sítí (LAN, WAN) a jejich vzájemné propojení.

- **Výhody:**
  - *Zvýšená spolehlivost a míra dostupnosti dat.*
  - *Místní řízení báze dat a snazší růst systému.*
  - *Snazší implementaci dalších lokálních databází.*
  - *Menší nároky a náklady na komunikaci.*
  - *Rychlejší odezvy.*



Obrázek 2.16: Architektura distribuovaných DBS, převzato z [32]

- **Nevýhody:**
  - *Obtížnější kontrola referenční integrity dat.*

2.7 Správa dat a zabezpečení databáze

Tato sekce se zabývá rolemi *administrátor dat* (DA) a *administrátor databáze* (DBA), zabezpečením a protiopatřením proti potenciálním druhům nebezpečí, kterým tyto role musí umět čelit.

2.7.1 Správa dat a správa databáze

Výše zmíněné role mají zodpovědnost za správu a všechny ostatní činnosti spojené s daty a databází. Každá role se specializuje na určité datové oblasti a podle složitosti jsou vykonávány určitým počtem pracovníků. Správce dat je více zapojen do dění v počáteční fázi DSDLC, počínaje s logickým návrhem databáze, kdežto správce databáze (DBA) nastupuje do dění v závěrečných fázích DSDLC (od návrhu databáze či aplikací až po údržbu DBS).

Správa dat je správa a kontrola dat organizace, včetně plánování databáze, vývoje a údržby databáze, údržby standardů, všeobecných postupů, procedur a logického návrhu databáze [3, st. 290]. Můžeme říci, že DA je více teoreticky či „manažersky“ zaměřen než DBS, který je spíše prakticky nebo-li technicky orientován. Musí být specializovaní na určitý SŘBD, pomocí něhož a operačního systému je prováděna *Správa databáze*. Jedná se o správu a kontrolování fyzické realizace DBS, včetně fyzického návrhu databáze a implementace, nastavení zabezpečení a kontroly integrity, monitorování výkonnosti systému a v případě potřeby reorganizace databáze [3, st. 291].

2.7.2 Zabezpečení databáze

Zabezpečení databáze jsou mechanismy, které chrání databáze proti záměrným nebo náhodným ohrožením [3, st. 292]. Je mylnou představou, že zabezpečení se týká jen databázových dat. Databázová bezpečnost zasahuje do ob-

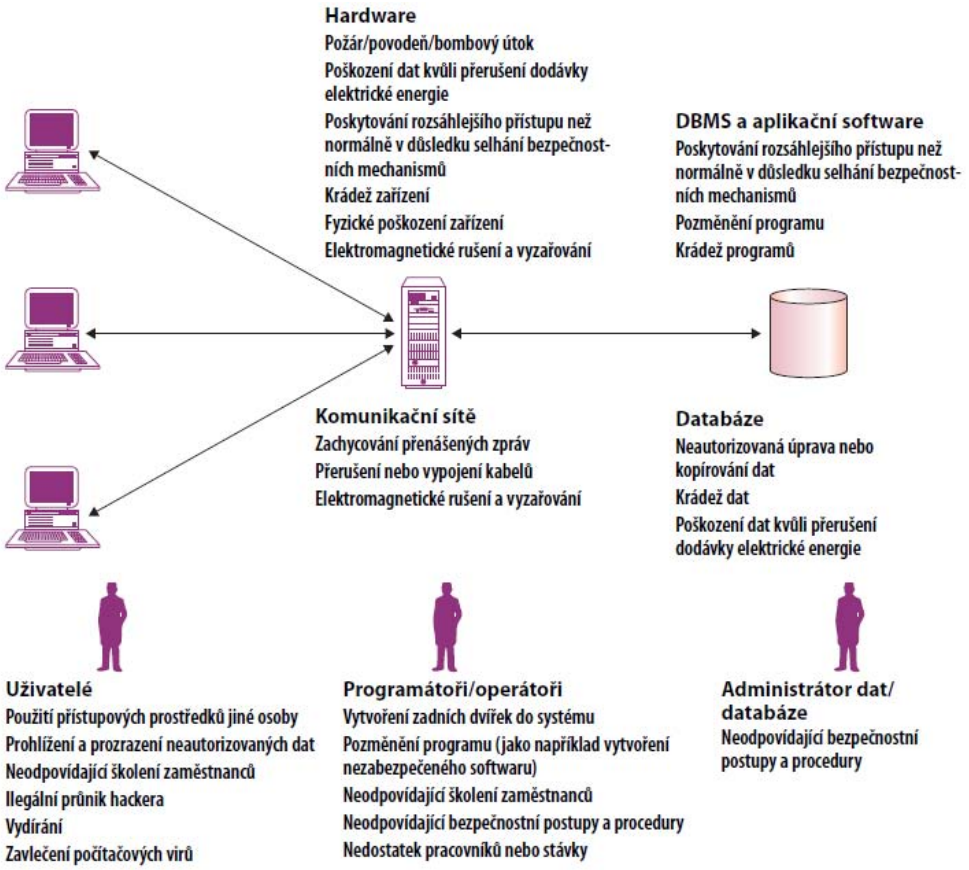
Tabulka 2.1: Hlavní rozdíly úkolů DA a DBA, převzato z [3, 292]

Správa dat	**Správa databáze**
Zapojení do strategického plánování IS	Vyhodnocuje nové DBMS
Určuje dlouhodobé cíle	Vykonává plány, aby se splnily cíle
Určuje standardy, všeobecné postupy a procedury	Kontroluje dodržování standardů, všeobecných popstupů a procedur
Určuje požadavky na data	Implementuje požadavky na data
Vyvíjí logický návrh databáze	Vyvíjí fyzický návrh databáze
Vyvíjí a udržuje datový model organizace	Implementuje fyzický návrh databáze
Koordinuje vývoj databáze	Monitoruje a kontroluje užívání databáze
Manažerská orientace	Technická orientace
Činnost nezávisí na DBMS	Činnost závisí na DBMS

lastí hardware, software, ale i do lidských zdrojů. Databáze a především data v ní jsou velmi citlivou součástí. Každý objekt spravující citlivá data by měl minimalizovat rizika [3, st. 292]:

- *Krádež a zpronevěra* nemusí změnit data, ale může podstatně poškodit celou firmu. Musí se především redukovat příležitosti vzniku těchto činů, jelikož příčinou je lidský faktor.
- *Ztráta utajení* je zapříčiněna poškozením nebo nedostatečným zabezpečením tajných dat a informací, ke kterým se dostala neautorizovaná osoba. Následkem může být ztráta konkurenceschopnosti.
- *Ztráta soukromí* může vzniknout nedostatečnou ochrannou dat nebo informací osob, které nechtějí sdílet své soukromé informace s jinými osobami. Únik soukromých dat může zapříčinit právní akci proti jednotlivci nebo celé skupině.
- *Ztráta integrity* způsobuje neplatnost dat uložených v databázi, což může značně ovlivnit chod dané organizace či firmy.
- *Ztráta dostupnosti* dat či informací, ať již nepatrná, má vliv na finanční zisky organizace a na data samotné, které mohou být znehodnoceny.

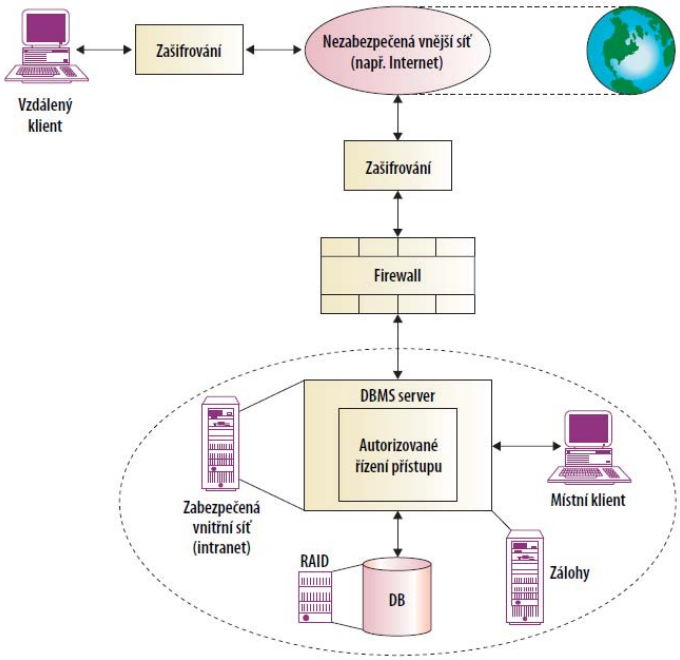
Cílem zabezpečení databáze je minimalizovat újmy či ztráty způsobené předvídatelnými událostmi nebo situacemi, které můžou být záměrné i nezáměrné. V takovém případě se jedná se o ohrožení, která jsou různě nebezpečná a můžou mít dopad na organizaci, proto by měly investovat značné úsilí na identifikaci nejzávažnějších druhů ohrožení (obr. 2.17) a inicializovat vypracování příslušných opatření, při zohlednění finanční stránky a stupně ohrožení. Například ohrožení způsobené počítačovými viry může mít za následek ztrátu integrity nebo dostupnosti.



Obrázek 2.17: Shrnutí potenciálních ohrožení počítačových systémů, převzato z [3, st. 296]

Spektrum protiopatření začíná na úrovní hardware a končí administrátorskými činnostmi. SŘBD je úzce spjata s OS, tady zabezpečení systému řízení báze dat dosahuje maximálně zabezpečení operačního systému. Víceuživatelská prostředí (obr. 2.18) by měla být podrobena bezpečnostním opatřením následujících typů [3, st. 296]:

- Autorizace
- Pohledy
- Zálohování a zotavení
- Integrita
- Zašifrování
- Redundance pole nezávislých disků (RAID)



Obrázek 2.18: Typické uživatelské prostředí výpočetního systému, převzato z [3, st. 297]

- Zabezpečení sítě

Autorizace je proces ověření přístupových oprávnění uživatele vstupující do systému. S autorizací je úzce spjata *autentizace*, která slouží k jednoznačnému určení uživatele, který přistupuje k systému. Prvně by měla být zabezpečena autentizace do samotného výpočetního systému pomocí uživatelských účtů, která mají nastavená oprávnění podle potřeb a zásad zabezpečení. Jedná se o nutný základ zabezpečení. Nyní se DBA postará o autentizaci a autorizaci na straně SŘBD a databázových aplikací. Uživatelská práva dovolují plnit uživatelům úkoly spojené s jejich pracovní činností. Pro DBA je velmi důležité, aby znal dostupné autorizační a kontrolní mechanismy příslušného SŘBD.

Pohled si lze představit jako virtuální (logické) tabulky, které nemusí vždy existovat v databázi, ale například v době vznešení požadavku uživatele. Pohledy mají velké uplatnění z hlediska ochrany dat, jelikož můžeme vytvořit pohled, který bude mít stejný obsah jako konkretní tabulka nebo specifické spojení tabulek, ale nebude obsahovat citlivé data (např. rodné číslo). DBA posléze nemusí nastavovat právo čtení konkrétní tabulky, ale umožní čtení vytvořeného pohledu bez citlivých dat. Z hlediska práce s tabulkami se pohled (anglicky view) tváří jako klasická databázová tabulka.

Zálohování a zotavení databáze po selhání poskytuje většina moderních nástrojů SŘBD. Pro usnadnění zotavení je možné použít log soubor, který

obsahuje záznamy o transakcích potřebné k efektivnímu zotavení po selhání (např. id transakce, typ log záznamu, obraz dat před a po operaci), čili informace o všech aktualizacích databáze. Výhodou žurnálování je především obnova databáze do posledního známého konsistentního stavu prostřednictvím záložní kopie databáze a informacích obsažených v žurnálu [3, st. 300]. Není-li v SŘBD zapnuto žurnálování a databáze potřebuje zotavit, jedinou možností je poslední záložní kopie databáze, což způsobí, že ztratíme všechna data vzniklá po této záloze. Typy záloh se můžou kategorizovat podle toho, zda mají klienti přístup k databázi během procesu zálohování [3, st. 300]:

- *Online zálohování* (hot backup) umožňuje klientům připojení k databázi i během procesu zálohování, ale některé operace s databází mohou být omezené, aby nebyla narušena integrita zálohovaných dat.
- *Offline zálohování* (cold backup) neumožní klientům přístup k databázovému systému v průběhu zálohování. Jedná se o jednodušší způsob, jelikož není riziko změny uložených dat během zálohování. V praxi je situace většinou jiná a vypnutí databáze nepřipadá v úvahu.

Zálohování může být také kategorizováno podle množství dat pro jedinou zálohu [3, st. 301]:

- *Úplná záloha* zálohuje všechna data v databázi.
- *Inkrementální záloha* zálohuje pouze data od poslední zálohy.
- *Diferenciální záloha* zálohuje pouze data od poslední úplné zálohy.

Integrita je v oblasti databází velmi důležitý pojem. Jde především o zachování následující vlastností [1]:

- *Integrita databáze* - celková správnost a ochrana před technickými závadami
- *Elementární integrita* - změny a záznamy mohou provádět pouze autorizované entity
- *Elementární správnost* - přijata jen korektní data s odpovídající typem a hodnotou

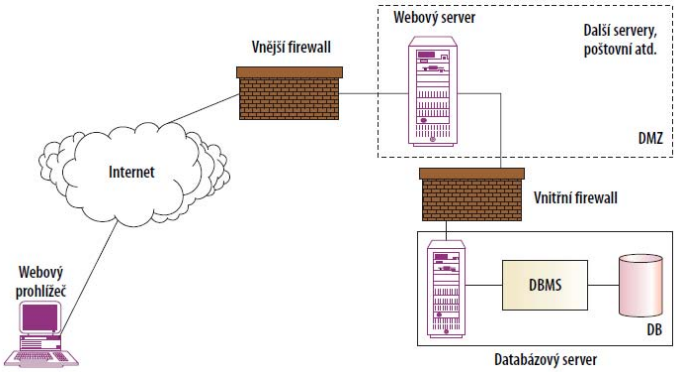
Zašifrování senzitivních dat (např. čísla bankovních účtů nebo řidičských průkazů) chrání před náhodným vyzrazením. Taková data můžou být šifrována funkcemi SŘBD. Systémy řízení báze dat můžou také k datům přistupovat, ale samozřejmě až po jejich dešifraci, což může vést k degradaci výkonnosti systému kvůli času zmíněnému dešifrování. Zašifrování by mělo být použito také pro data přenášená po soukromých linkách či nezabezpečených sítích.

Šifrování se kategorizuje na dvě oblasti, z nichž první je nazývána symetrické šifrování a druhá asymetrické. Liší se mezi sebou způsobem, kde a jak vzít klíč k dešifrování zpráv. První oblast používá stejný klíč pro šifrování i dešifrování zprávy a druhá se vyznačuje používáním sady dvou klíčů, přičemž veřejný klíč může znát kdokoli a používá se k zašifrování zprávy, zatímco soukromý klíč si uchovává příjemce a je tajný.

V roce 1977 [29] zavedla americká vláda blokovou šifru DES (Data Encryption Standard) založenou na Luciferově algoritmu jako standard pro symetrické šifrování v civilních státních organizacích v USA a následně se rozšířila i do soukromého sektoru. DES kóduje 64 bitové bloky pomocí 56 bitového klíče. V dnešní době je šifra považována za nepříliš bezpečnou, jelikož v roce 1998 superpočítač „Deep Crack“ jako první hrubou silou prolomil DES klíč za pouhých 56h [29]. To dalo vzniknout Triple DES, který používá tři 56 bitové klíče. Triple DES se dnes hojně používá v on-line bankovnictví nebo SSL spojení. Ačkoliv je dnes 3DES považován za bezpečný, díky aktualizovanému klíči a upravenému algoritmu, je značně pomalý [29], což byl důvod, proč AES (Advanced Encryption Standard) starší DES šifru v roce 2002 nahradil. AES je založen na Rijndaelově algoritmu a kóduje 128 bitové bloky dat pomocí klíčů s délkou 128, 192 a 256 bitů [29]. Nevýhodou symetrických postupů je, že nejsou vhodné například pro přenos dat. Bezpečného přenosu symetrických šifrovaných dat jde docílit pomocí asymetrického šifrování. Nejznámějším asymetrickým šifrovacím systémem je RSA, jehož bezpečnost je založena na faktu, že není znám výpočetní algoritmus, který by jednoduše dokázal velká čísla rozložit na prvočísla. Obecně také platí, že symetrické algoritmy jsou při zpracování na PC mnohem rychlejší něž asymetrické [3, st. 303]. V praxi se často používají „hybridní“ postupy (např. pro přenos dat na webu, v e-mailovém styku či během on-line bankovnictví), kdy se využívají výhody obou typu šifer.

Redundance pole nezávislých disků (RAID). RAID je v podstatě pole fyzických nezávislých diskových jednotek, které jsou organizovány tak, že se jeví jako jedna velká logická paměťová jednotka a zároveň tak, aby se zvýšila výkonnost a spolehlivost [3, st. 303]. Existuje několik různých druhů konfigurací RAID polí s rozdílnou výkonností a spolehlivostí. Toto rozsáhlé téma přesahuje rámec této práce a zaslouží si samostatnou kapitolu, proto zde uvedu jen nejčastěji používané úrovně RAID pro databázové aplikace – RAID 1, RAID 1+0 a RAID 5 [3, st. 304].

Zabezpečení sítě je opatřením, které se od předchozích opatření liší tím, že se nesoustředí na interní kontrolu v rámci SŘBD a databáze, ale na ochranu serverů před vniknutím pomocí bezpečnostních architektur pro sítě. Základních architektura zabezpečení sítě pro třívrstvého DBS je znázorněna na obrázku 2.19. Jedná se pouze o jednu z mnoha architektur zabezpečení sítě. Bezpečnost sítí je velmi rozsáhlé téma, které má svůj specializovaný obor. Pro účely této práce postačí obecné pojmy. Mezi důležité pojmy je zařazen Firewall – síťové zařízení (server nebo router s dvěma nebo více síťovými rozhraními



Obrázek 2.19: Základní architektura zabezpečení sítě v případě třívrstvého databázového systému, převzato z [3, st. 305]

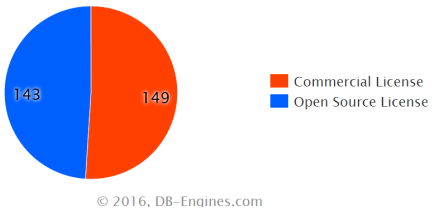
a speciálním software), které má za cíl chránit (blokovat/filtrovat zprávy) síťový provoz. Na již zmíněném obrázku je možné na vnějším firewallu blokovat všechny zprávy, které se netýkají požadavků na webové stránky a e-maily. Toto opatření sníží pravděpodobnost zneužití dalších nedostatků služeb za vnějším firewallem. Přísnější vnitřní firewall povoluje jen komunikaci mezi webovým serverem a databází. Oblast s omezenou sítí mezi těmito dvěma firewally se nazývá demilitarizovaná zóna (DMZ). Většinou obsahuje služby, které jsou k dispozici celému internetu. Účelem DMZ je zajistit, aby případný útočník, který získá přístup pouze k zařízení, které je v DMZ, nějakým způsobem nepoškodil zbytek lokální sítě.

2.8 Analýza popularity a členění SŘBD

Systém řízení báze dat byl blíže popsán v sekci 2.2.2, tato sekce by měla být přínosem při rozhodování o výběru SŘBD. Popularita je z určité části obrazem vhodné použitelnosti SŘBD. Srovnáme-li popularitu SŘBD z vícero pohledů, tak nabyté informace nám mohou mnohé prozradit, či navést čtenáře k vhodnější volbě SŘBD. Informace byly čerpány pomocí *DB-Engines* (http://db-engines.com/). Jedná se o iniciativu, která sbírá a prezentuje informace o systémech řízení báze dat.

2.8.1 Obecné členění a popularita SŘBD

Prvním obecnějším pohledem na celé spektrum SŘBD můžeme zaznamenat dvě množiny, které rozděluje pojetí licencování. První množina tzv. *Open source SŘBD*, která má volně přístupný zdrojový kód pro modifikaci a užití podle licence. Druhá - komerčně pojatá - množina, jejichž prvky sjednocuje název *Komerční SŘBD*. Členění podle databázového modelu, kde některé z těchto modelů analyzuje sekce 2.5, přináší druhý obecnější pohled na SŘBD.



Obrázek 2.20: Počet počet open source a komerčních databází, leden 2016, převzato z [6]

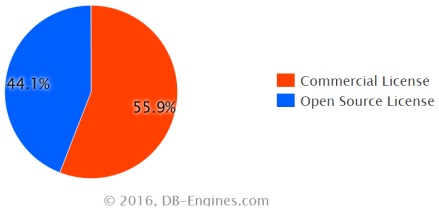
2.8.1.1 Členění a popularita podle licencí

Na koláčovém grafu 2.20 je znázorněn poměr *open source* a *komerčních* SŘBD k lednu 2016. K porovnání využiji také data, která jsem uchovával od března roku 2015, kdy *DB-Engines* zpracovával data o 257 různých systémech řízení báze dat, v lednu 2016 jejich počet narostl na 292. V prvně zmíněném období byl poměr 127:130 $(0,976)$ pro komerční SŘBD. Poměr pro druhé období poskytuje již zmíněný koláčový graf 143:149 $(0,959)$. V tomto období se počet zpracovávaných open source zmenšil ku počtu komerčních SŘBD.

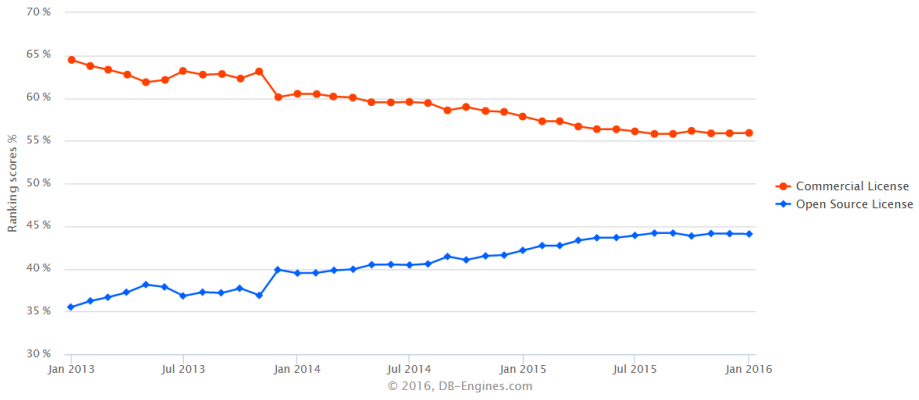
Komerční systémy řízení báze dat vítězí v žebříčku popularity nad open source (obr. 2.21), ale z hlediska dlouhodobého trendu (obr. 2.22) můžeme očekávat zvrat, jelikož v lednu 2013 byla oblíbenost SŘBD s volně přístupným zdrojovým kódem 35.53 % a v lednu roku 2016 již 44.08 %. K tomuto nárůstu značně přispěly SŘBD na vrcholu žebříčku (obr. 2.24), kde klíčová specifika *MySQL*, *PostrgeSQL* a *MongoDB* jsou blíže popsány v podsekci 2.8.2. Tato sekce taktéž popisuje i SŘBD *Oracle* a *Microsoft SQL Server*, které patří do žebříčku (obr. 2.23) nejpopulárnějších komerčních SŘBD. Popularita těchto dvou stran licencí je také rozdílná pro jednotlivé datové modely (obr. 2.25). Zajímavým faktem je, že téměř všechny (99.7 %) grafové SŘBD spadají pod open source licence a objektové se svými 94 % pod komerční licence. Následující podsekce se blíže zabývá analýzou a členěním podle databázových modelů.

2.8.1.2 Členění a popularita podle databázových modelů

Dalším pohledem jsou množiny, kde každá z nich je definována charakteristickým datovým modelem. Jak již bylo zmíněno, DB-Engines zpracovává (k lednu 2016) 292 rozdílných SŘBD, nejpočetnější skupinou jsou relační modely s po-



Obrázek 2.21: Popularita open source a komerčních databází, březen 2015, převzato z [6]



Obrázek 2.22: Trend popularity open source a komerčních databází, leden 2013-2016, převzato z [6]

Rank	System	Score	Overall Rank
1.	MySQL	1299	2.
2.	MongoDB	306	4.
3.	PostgreSQL	282	5.
4.	Cassandra	131	8.
5.	SQLite	104	9.

Obrázek 2.23: Pět nejpopulárnějších open source systémů k lednu 2016, převzato z [6]

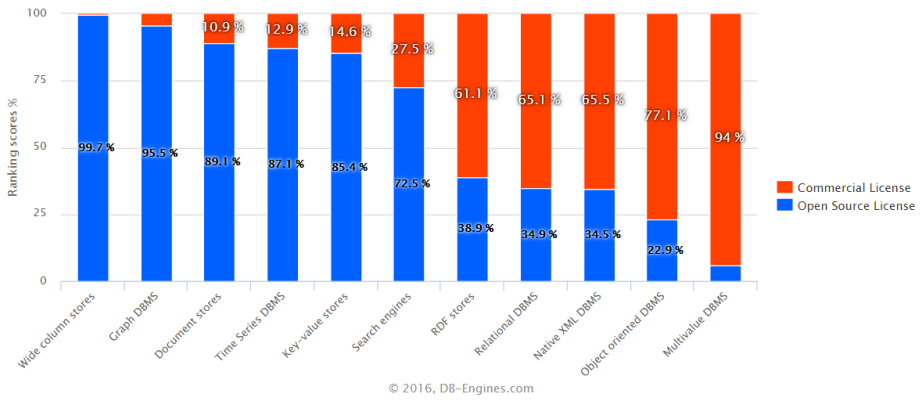
Rank	System	Score	Overall Rank
1.	Oracle	1496	1.
2.	Microsoft SQL Server	1144	3.
3.	DB2	196	6.
4.	Microsoft Access	134	7.
5.	SAP Adaptive Server	83	11.

Obrázek 2.24: Pět nejpopulárnějších komerčních systémů k lednu 2016, převzato z [6]

čtem 113 SŘBD, což můžeme vidět na koláčovém grafu vlevo nahoře na obrázku 2.26. Některé ze systémů spadají do více něž jedné kategorie. S popularitou u tohoto nejpočetnějšího datového modelů tomu není jinak (obr. 2.26 - vpravo nahoře). S 81.5 procenty „vládne“ nad všemi databázemi, které jsou spíše minoritou. Součet všech hodnocených modelů je 100 %. Graf změn popularity (obr. 2.26 - dole) však vykazuje zajímavé hodnoty trendu pro *Grafové SŘBD*, jejichž růst se výrazně odlišuje od všech ostatních. Z pragmatického pohledu však nelze říct, že by během několika málo let předběhly v hodnocení popularity relační modely, neboť dosahují jen 0.8 %.

2.8.2 Klíčová specifika vybraných SŘBD

Iniciativa *DB-Engines* vytvořila vlastní metodu [5] pro vyhodnocování trendu popularity. Metodika zahrnuje použití systémů jako je Google, Bing, Stack

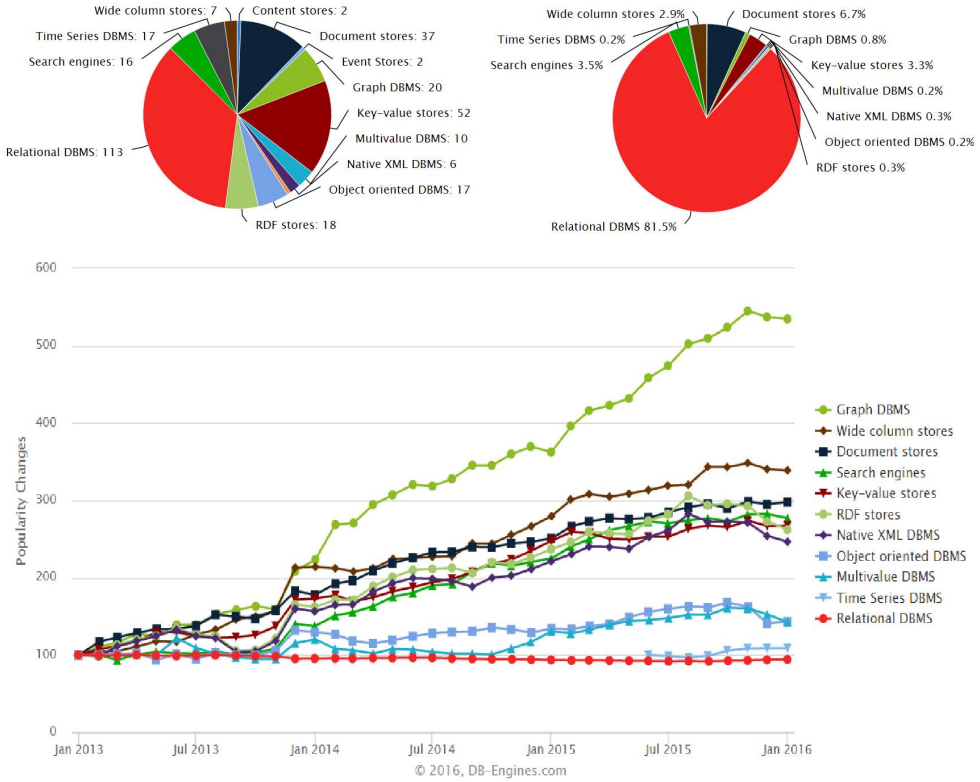


Obrázek 2.25: Členění popularity podle databázového modelu k lednu 2016, převzato z [6]

Overflow, DBA Stack Exchange, Indeed, Simply Hired, LinkedIn a Twitter. Výsledkům této metodiky za posledních několik měsíců dominují dvě komerční databáze (Oracle, Microsoft SQL Server) a tři open source (MySQL, MongoDB, PostgreSQL). Následující podkapitoly shrnují hlavní specifika zmíněných SŘBD. Nejdříve bude porovnána popularita výše zmíněných systémů řízení báze dat. Z hlediska dlouhodobého trendu popularity, viditelném na obrázku 2.27, je zřejmé, že „světu SŘBD“ vládnou Oracle, MySQL a Microsoft SQL Server. O pár řádů níže následují zbylé dva PostrgreSQL a MongoDB, které momentálně svádí souboj o 4. příčku. Přihlédneme-li však k jejich specializaci, tak MongoDB se řadí mezi NoSQL databáze (Document store), kdežto ostatní jsou tradiční relační databáze. Zajímavý je také pohled na data prvního vydání, které přináší tabulka D.2 společně s ostatními informacemi o SŘBD.

2.9 Současné a nové trendy v databázích

Svět dat prochází neustálými změnami, které mají vliv na databázové potřeby, v jejich důsledku vznikají nové technologie a trendy. Velkou hybnou silou jsou velké databázové oblasti (internet a jednotlivé vědy, jako jsou fyzika,

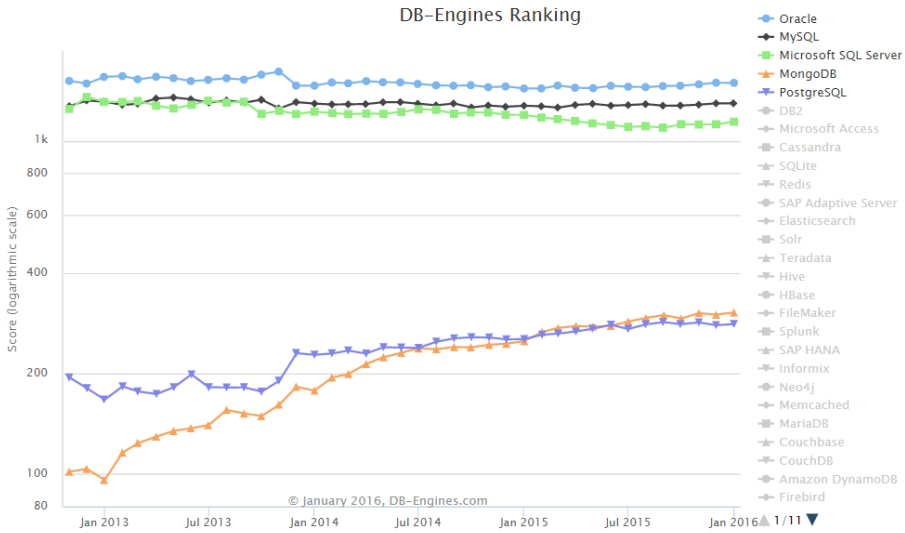


Obrázek 2.26: SŘBD popularita podle databového modelu k lednu 2016, převzato z [4]

biologie, medicína a inženýrství), které vyvolávají potřebu vylepšení starých a vývoje nových technologií [36, st. 175] a také různých metodik a technik, jenž pomůžou vytvořit požadovaný systém. Následující kapitola ve stručnosti sumarizuje hlavní moderní trendy v databázích.

2.9.1 NoSQL, NewSQL

Termín *NoSQL databáze* volně specifikuje oblast nerelačních datových úložišť. Výhodou této oblasti je jednoduchost návrhu a také fakt, že data v NoSQL databázi mohou být rozložena nejen horizontálně, ale i vertikálně. „Not only SQL“ databáze, jak databázová komunita tento typ databází nazývá, jsou v podstatě vysoce optimalizovaná datová úložiště, kde nejjednodušší NoSQL databáze, zvané *úložiště typu klíč-hodnota*, obsahují pouze množinu dvojic (klíč, hodnota). Ukládání dat a s tím spojená složitost vyhledávání je oproti RDBMS odlišná, jedná se například o stromovou či grafovou strukturu. Horizontální škálování brání naplnění ACID vlastností. V praxi se začínají objevovat systémy (např. relační SŘBD MySQL Cluster, VoltDB, Clustrix), kde

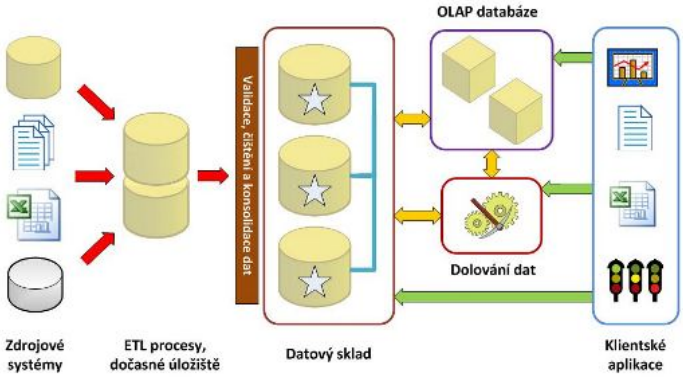


Obrázek 2.27: Graf popularity DBMS podle metodiky iniciativy DB-Engines, převzato z [5]

architektura (pro cloud computing) umožňuje horizontální škálování a zároveň jsou zachovány ACID vlastnosti. Tato kategorie, vysoce škálovatelných databází a pružných transakčních relačních SŘBD, je označována termínem *NewSQL* a charakteristická následujícími vlastnostmi [38, st. 250]:

- Horizontální škálování v architekturách založených na „sdílení ničeho“ (podobně jako u NoSQL).
- Garantují ACID.
- Aplikace interagují s databází primárně pomocí SQL.
- Uživatelé nejsou blokování, díky používání řídících mechanismů bez zamykání pro souběžné zpracování.
- Oproti tradičním systémům poskytují vyšší výkon.

Bez ohledu na různá omezení si NoSQL databáze našly své místo ve světě databází (nestrukturované data, vysoké požadavky na škálování), ale mají zatím daleko k vyspělým databázovým technologiím (spolehlivost) a svou koncepcí nemůžou nahradit tradiční relační SŘBD [38, st. 249].



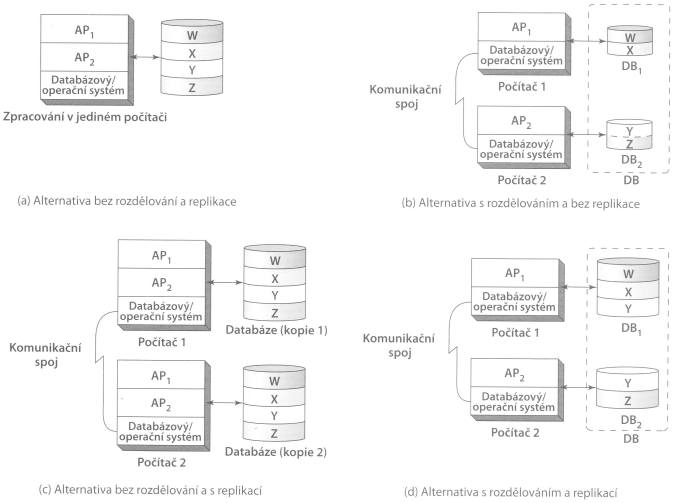
Obrázek 2.28: Architektura BI, převzato z [33]

2.9.2 Big Data, Business Intelligence systémy a datové sklady

V této podkapitole zanalyzuji technologie řešící problémy související s rychle rostoucím objemem dat. Jedná se o tzv. *Big Data*, což je současný termín pro mimořádné rozsáhlé datové množiny generované webovými aplikacemi (např. vyhledávače Google a Bing) a sociálními sítěmi Web 2.0 (např. Facebook, LinkedIn a Twitter). Tyto velké objemy dat potřebují organizace nějakým způsobem zpracovávat, proto také existují systémy Business Intelligence (BI). Je to komplex přístupů, aplikací a technologií, které pomáhají nejen manažerům při analýze aktuálních a minulých dat, ale i předpovídají budoucí události [9, st. 468].

Architektura tradiční BI se skládá z několika základních komponent, jejichž vazby zachycuje následující schéma 2.28. Data ze zdrojových systémů (např. provozní databáze, interní a externí databáze) se plní do *datového skladu* za pomocí systému ETL, jehož procesy data extrahují, čistí a připravují ke zpracování. Nad datovým skladem je postavena databáze OLAP (Online Analytical Processing), přizpůsobená pro analýzu dat. Klientské aplikace jsou jakýmsi prostředníkem mezi BI systémy a uživateli.

Přímé dotazování na provozní databáze můžeme v praxi využít pouze u nejmenších a nejjednodušších aplikací a databází BI, což má několik důvodů [9, st. 468]. Vytěžování provozní databáze výkonnostně náročnými dotazy může vést k jejich zpomalení nebo dokonce dočasnému výpadku. Provozní data vykazují problémy omezující užitečnost pro aplikace BI, které pro svou činnost potřebují speciální programy a nástroje, které provozní databáze obvykle postrádají. Vznikají tedy datové sklady a datová tržiště, do kterých jsou data z heterogenních datových zdrojů extrahována, transformována a načtena pomocí systémů ETL (extract, transform, and load), jejichž hlavní úlohou je zaručit, že datový sklad bude obsahovat pouze správná a důvěryhodná data. Datové tržiště je kolekce vzájemně příbuzných dat menšího rozsahu než da-



Obrázek 2.29: Typy distribuovaných databází, převzato z [9, st. 449]

tový sklad. Datový sklad například distribuuje data webového protokolu menšímu datovému tržišti webových prodejů za pomocí nástrojů BI pro analýzu webových klepnutí, z čehož můžou webový specialisté čerpat vlastnosti návrhu webových stránek. Databáze OLAP (On-Line Analytical Processing) je přizpůsobená pro analytické dotazování. Obsahuje datové kostky, které modelují data z tabulek faktů a dimenzí (pro více informací [9, st. 438] a obrázek 8.8) do logického multidimenzionálního prostoru, z kterého je možné intuitivně nahlížet na data z nejrůznějších perspektiv za pomocí klientských aplikací [33]. Nasazení BI řešení představuje návratovou investici především pro společnosti, které chtějí využít rozsáhlé firemní datové množiny, ve svůj prospěch.

2.9.3 Distribuované databáze

Databázový systém typu NoSQL pomáhající lépe zpracovávat velké datové množiny, je obvykle založen na *distribuované replikované databázi*. Slovo distribuované značí, že databáze je sdílená mezi servery z clusteru. Databázi lze distribuovat pomocí rozdělování (partitioning), kdy části databáze jsou rozmístěny do více počítačů. Replikace poskytuje další možnost distribuce, kdy více počítačových jednotek vlastní kopii celé databáze. Poslední možností je kombinace předešlých [9, st. 449]. Tyto typy distribuovaných databází popisuje obrázek 2.29.

(a) Nedistribuovaná databáze se čtyřmi částmi (W, X, Y, Z).

(b) Rozdělená databáze bez replikace. Části databáze W a X jsou uloženy a zpracovávají se v PC č. 1, zatímco části Y a Z PC č. 2.

(c) Replikovaná databáze bez rozdělení. PC č. 1 a 2 zpracovávají celou databázi.

(d) Rozdělená replikovaná databáze. Část databáze Y je uložena a zpracovávána PC č. 1 a 2.

- **Výhody distribuovaných databází podle [9]:**
  - *Výkon.* Rozdělením databáze do více počítačů lze snížit zatížení (např. sdílení pracovní zátěže).
  - *Kontrola.* Každá část databáze může mít vlastní sadu autorizovaných uživatelů se specifickými oprávněními.
  - *Transparentnost.* Uživatel má ponětí, že všechna data jsou zpracovávána na jednom serveru v lokální databázi.
  - *Autonomnost.* Každé lokální databázi zapojené do distribuované databáze je umožněno autonomní uložení a zpracování dat nezávisle na ostatních databázích.
  - *Lepší a trvalejší dostupnost* díky redundanci (replikace dat na vzdálených počítačích). Systém není závislý na jednotlivých počítačích, může fungovat i v případě havárie jednoho z nich.
- **Nevýhody:**
  - *Návrh distribuované databáze je složitější* než v centralizovaném případě (viz podsekce 2.6.1) (problémy s rozdělením dat).
  - *Složitost koordinace dat ukládaných na různých počítačích* má i své důsledky v podobě nákladů na vývoj software a větší pravděpodobnost chyb.

2.9.4 Cloud computing

Cloud computing je mladý rozvíjející se obor, který není zcela striktně definován. Často uváděnou definicí na internetu je ta z roku 2009 od NIST (National Institute of Standards and Technology): *Cloud computing je model umožňující pohodlný síťový přístup na vyžádání ke sdílenému zásobníku konfigurovatelných výpočetních zdrojů (např. sítí, serverů, pamětí, aplikací a služeb), které mohou být rychle dodány a uvolněny s minimálním úsilím investovaným pro řídící činnost či interakce s poskytovatelem služby [11].* Hlavní myšlenkou cloud computingu je poskytování služeb pomocí internetu, což s sebou přináší mnoho výhod, ale i nevýhod.

- **Časté charakteristiky cloud computingu:** ([38, st. 229] a [39])
  - *Shromažďování zdrojů.* Jejich velikost, umístění a struktura jsou uživatelům utajeny.

  - *Široký přístup k síti.* Uživatel se může připojit kdekoliv, nezávisle na platformě (PC, tablet, mobilní telefon, ...).
  - *Rychlost pružnosti služby* (rychlá škálovatelnost, upravování prostředků).
  - *Měření služby a jejich optimalizace* (dynamické přidělování výkonu podle potřeb).
  - *Šetří pořizovací náklady* (hardware, síťová konektivita a softwarové licence).
  - *Šetří provozní náklady* (energie na provoz serverů a chlazení).

- **Časté obavy a nevýhody cloud computingu:** [39]

  - *Možná nedostupnost služeb* (výpadek služby nebo zprostředkovatele připojení k internetu (ISP)).
  - *Obavy o ochranu dat a osobních údajů.* Citlivé informace jsou předány poskytovateli cloud služby (poskytovaná data šifrovat).
  - *Obavy z kybernetických útoků.* Data na internetu jsou náchylnější k vnějším útokům a hrozbám (např. DDoS útoky - přetížení cílového serveru požadavky).
  - *Méně funkcí.* Deskopové řešení má většinou rozmanitější nabídku funkcí něž SaaS.

Distribuční modely reprezentují pohled na cloud computing podle poskytovaných služeb, které řadíme do tří základních modelů [11]:

- Infrastructure as a service (IaaS) - kompletní virtuální stroj ve vlastní správě dle sjednané konfigurace (např. Amazon EC2, Windows Azure, Rackspace, Google Compute Engine)
- Platfrom as a service (PaaS) - kompletní platforma pro hostování aplikace (např. AWS Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, Apache Stratos)
- Software as a service (SaaS) - poskytován pouze konkrétní software (např. Google Apps, Microsoft Office 365)

Z hlediska databází lze uvažovat službu PaaS, kde poskytnutí databáze je jednou z možností, nebo pouze databázovou službu DaaS (Databse as a service), která poskytuje přístup k databázi na požádání s cílem uložit data [2].

2.10 Výběr vhodné databáze pro evidenci studentů

Slovní spojení „výběr vhodné databáze“ je poněkud velmi obecný a vybírat jen vhodnou databázi bez přihlédnutí na strukturu DBS ($DB + S\check{R}BD = DBS$) je nepřínosné, jelikož databáze je úzce spojena se systémem řízení báze dat, jak je vidět na obrázku 2.2. Vzhledem k množstvím problémů, které jsou spojeny se souborovým přístupem, jsem se rozhodl pro databázový přístup, jenž odstraňuje nedostatky prvně zmíněného přístupu. Jedná se také o vyspělejší technologii pro přístup k datům.

Zprvu jsem zúžil výběr kandidátů pro databázový model a vyloučil jsem použití historických modelů. Jednosměrné vztahy typu 1:N hierarchického modelu a fakt, že evidence studentů není jen hierarchickou strukturou, vyloučili použití tohoto modelu. Síťový model sice lépe popisuje reálný svět, ale zejména nepružnost a obtížná změna jeho struktury tento model také vylučují. Přestože síťový model byl posunem kupředu, tak stále není tak vyspělý jako relační model, který je hojně využíván pro běžné komerční aplikace (např. inventáře, správa nebo evidence hostů či také studentů, zpracování objednávek a další). Základní relační model je však méně vhodným řešením pro aplikace s více objektovou strukturou dat jako například 3D zobrazení, doprovodný audio komentář nebo rozsáhlé satelitní fotografie. Taková data se však tohoto evidenčního projektu netýkají.

Když jsou porovnávány jen výhody nerelačních databází, často také označované NoSQL, občas je možné mít klamnou představu o tom, kdy je vhodné tento populární databázový model použít a zcela zapomenout, kdy je lepší použít tradiční relační model a jeho výhod. Relační databáze a NoSQL databáze mají své silné a slabé stránky, které je potřeba zvážit pro konkrétní projekt a strukturu dat v něm uložených. Pro projekt *Evidence studentů tanečních škol* jsou data v jednoduché struktuře tabulek, pro které se nejvíce hodí relační databázový model, jenž poskytuje konzistenci, spolehlivost a referenční integritu. Pokud by se jednalo o projekt, který by využil výhod NoSQL databází a zároveň by vyžadoval silnější konzistenci, doporučil bych technologii NewSQL.

Ze současných nových trendů má mnoho využití cloud computing. Například bezpečné a spolehlivé zálohování databáze na „cloud“, kdy jsou data záloh zabezpečená při přenosu i po uložení. Zálohy jsou navíc uložené v geograficky replikovaném úložišti. Při návrhu a realizace databáze musíme brát v úvahu také jednotlivé typy architektur. Pro tento typ projektu jsem zvolil vícevrstvou architekturu s tenkým klientem, od čehož očekávám vyšší modularitu, což především usnadňuje změnu nebo nahrazení jiné vrstvy bez ovlivnění ostatních vrstev.

Administrátor databáze by měl být specializován na určitý SŘBD, pomocí něhož je prováděna správa databáze. Mou specializací je Microsoft SQL Server, který se řadí mezi nejpopulárnější komerční produkty a zároveň je úzce spjat

s technologiemi Microsoft, které patří mezi požadavky pro tuto databázovou aplikaci (.NET, WCF a IIS). Bližší informace poskytne diagram nasazení v teoretické části bakalářské práce.

Kapitola 3

Datové úložiště - návrh, realizace a testování

Následující sekce se zabývá návrhem, realizací a testováním datového úložiště žáků evidující návštěvy odpoledních a večerních kurzů taneční školy.

3.1 Popis domény

Taneční společnost vyžaduje evidenci návštěv odpoledních a večerních kurzů a žáků taneční školy. Jednotlivé kurzy (eviduje se povinný název a nepovinné položky popis, typ, kapacita, místo konání, datum začátku a konce kurzu) můžou spadat pod určitou sezónu, která má svůj povinný název a nepovinný popis a informaci o roku či letech konání. Každá lekce (evidujeme povinný údaj číslo lekce) má svůj kurz.

Taneční společnost provozuje dva typy karet. Taneční karty (evidují povinně unikátní čárový kód a nepovinně textový dodatek a informaci o tom, zda je karta aktivní a zda je na černé listině) a gardenky (evidují stejné informace jako taneční karty a navíc nesou informaci o číslu sedadla), které slouží jako vstupní karta hostům/doprovodu/rodičům na konkrétní lekci. Za osoby využívající gardenky zodpovídá žák, na kterého je karta vedena. Karty můžou existovat samostatně nebo můžou být přiřazeny určitému kurzu a/nebo žákovi.

Taneční společnost eviduje žáky (povinné udaje jsou jméno, příjmení, pohlaví a nepovinné údaje e-mailová adresa a telefonní číslo) a jejich platby (povinné údaje jsou částka a stav platby - nepovinné údaje jsou datum platby a textový dodatek k platbě), které můžou spadat pod konkrétní kurz.

Je potřeba také evidovat podvodné chování žáků - tedy i karet tanečního kurzu - na černé listině (povinný údaj s odůvodněním přidání na černou listinu a datum přidání záznamu). Záznam v černé listině se vždy týká konkrétního žáka a případně i jeho karet. Žák může mít více záznamů na černé listině.

Taneční společnost eviduje návštěvy registrovaných žáků a karet na něho registrovaných pro jednotlivé lekce kurzu. Docházka nese povinné informace o tom, zda a kdy se karta vedená na žáka zúčastnila dané lekce.

Je požadována také správa uživatelů klientské aplikace. Uživatel musí povinně uvést své uživatelské jméno a heslo. Uživatelská role (povinně název role) může spravovat více uživatelů.

Platí ještě tato omezení: Záznam v černé listině a docházce může obsahovat informace jen k jedné existující kartě spravované daným žákem. Pokud karta není přiřazená konkrétnímu žákovi, nemůže být aktivní a naopak. Pokud černá listina nese záznam o kartě, záznam o této události se promítne na danou kartu a naopak. Karta nemůže být blokována pokud nemá záznam na černé listině a naopak. Počet tanečních karet přiřazených k danému kurzu nemůže být větší než jeho kapacita. Datum začátku kurzu nesmí být pozdější než datum jeho konce. Taneční karta nepřiřazená nějakému žákovi nemůže být přiřazena do kurzu.

3.2 Konceptuální schéma

Konceptuální schéma popisuje strukturu a sémantiku dat a je implementačně nezávislé na databázovém systému. V navrhované databázi konceptuální schéma (viz obr. 3.1) obsahuje 11 entit (Season, Course, Lesson, DanceCard, SeatReservationCard, Payment, Pupil, Blacklist, Attendance, Role a User). Jednotlivé entity obsahují položku časového razítka, jedná se řešení pro možné využití do budoucna (například k vybudování vlastní replikace dat nebo k řešení synchronizací).

3.2.1 Diskuze smyček

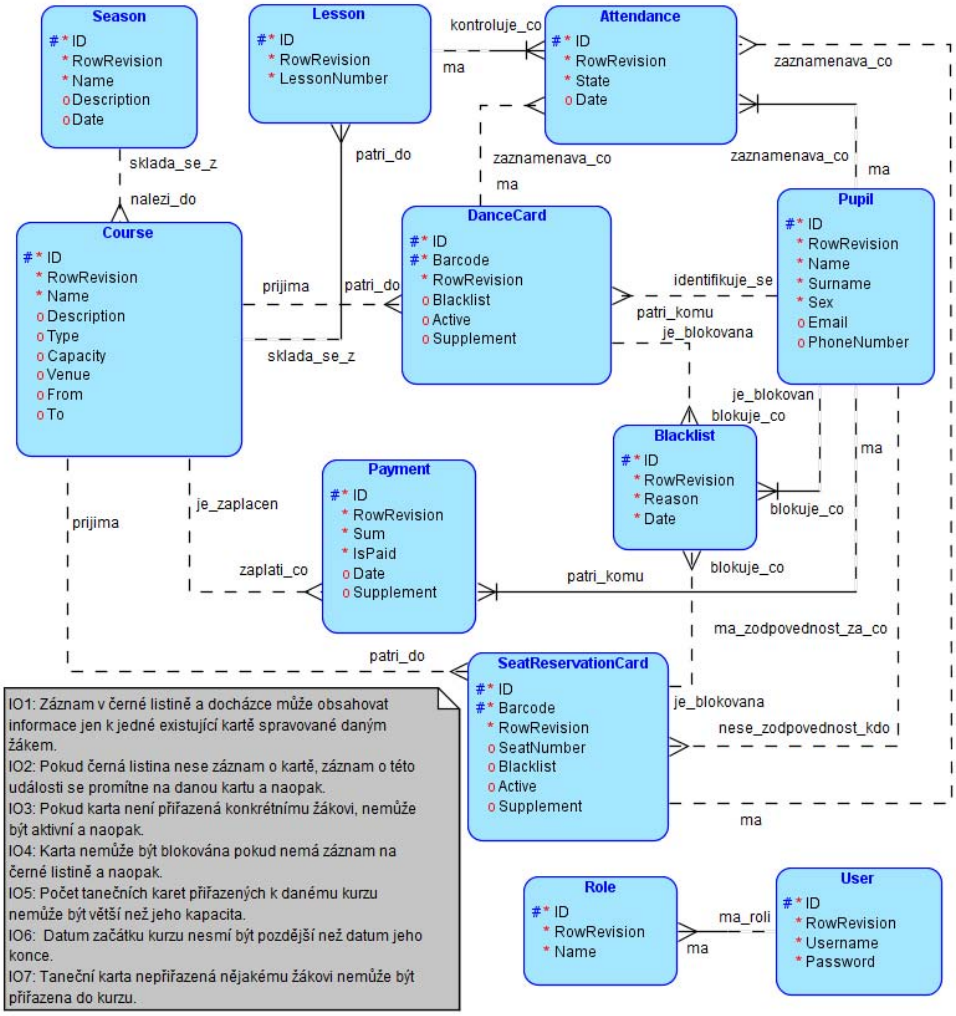
Schéma obsahuje smyčky:

- DanceCard - Attendance - Pupil - DanceCard
- SeatReservationCard - Attendance - Pupil - SeatReservationCard
- DanceCard - Blacklist - Pupil - DanceCard
- SeatReservationCard - Blacklist - Pupil - SeatReservationCard

Jejich nebezpečí řeší přidané integritní omezení IO1.

3.3 Relační schéma

Relační schéma vygenerované z konceptuálního modelu (viz obr. 3.2) obsahuje více implementačních detailů (např. cizí klíče). Po kontrole relačního schématu jsem doplnil následující integritní omezení:



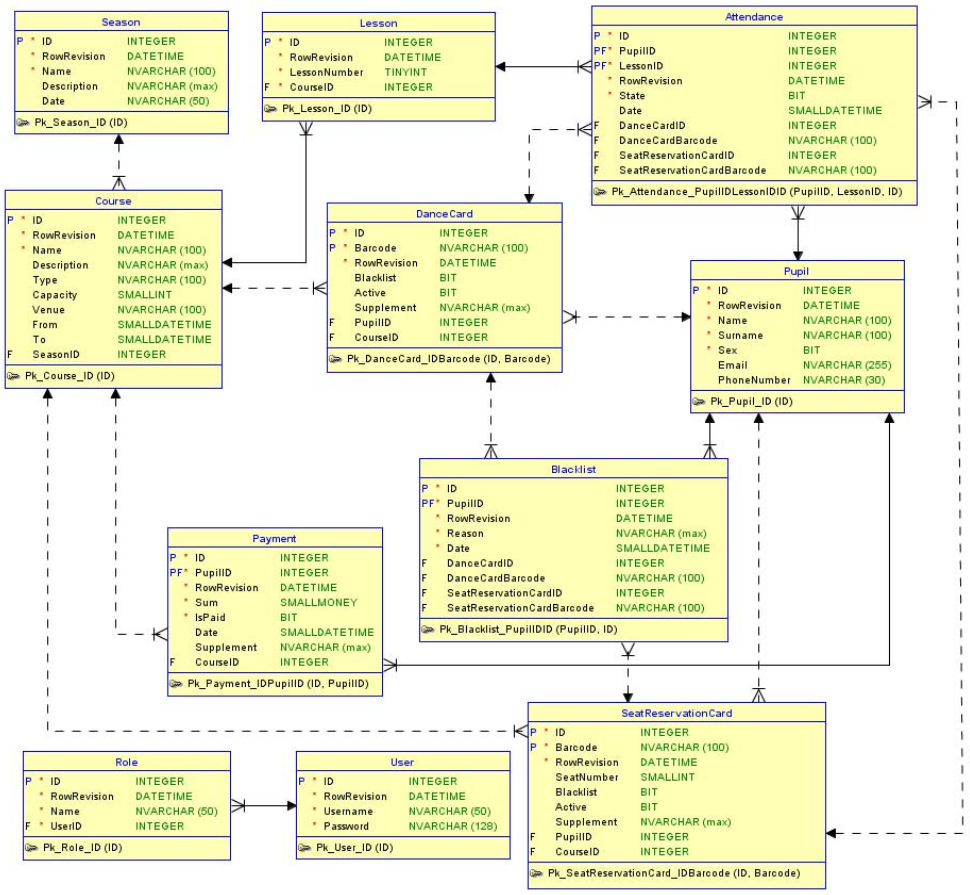
Obrázek 3.1: Konceptuální schéma datového úložiště

- Černá listina a docházka obsahují cizí klíče na složený primární klíč (ID, Barcode) karet. Tyto cizí klíče musí existovat v páru (tabulka DanceCard a SeatReservationCard) jako složené cizí klíže.
- Karta (taneční karta a gardenka) je specifická svým čárovým číslem. S touto kartou může konkrétní osoba navštěvovat více kurzů.
- Karta a lekce vedená v docházce musí patřit do stejného kurzu.
- Pokud je žákova karta (taneční karta a gardenka) zařazena na černou listinu, tak je nemožné s kartou navštívit přidružené kurzy k této kartě.
- Karta je nepřenosná.
- Číslo sedadla u gardenky a informace o kapacitě kurzu nemůžou být záporné.

Dále byl vytvořen SQL script, který po spuštění vytvoří relační schéma datového úložiště. Byly také vytvořeny různé varianty skriptů, které se liší tím, jestli daný script naplní databází testovacími daty či ji smaže před vytvořením nové. Vytvořenou databázi je možné naplnit testovacími daty, aby se nad ní mohly provádět dotazy či spouštět testy. Dávka je připravena tak, aby ji bylo možné spouštět bez chyb opakovaně.

3.4 Testování

Vytvořené schéma a jeho integritní omezení byla otestována unit testy. Byl vytvořen *SQL Server Database Project*, do kterého bylo naimportováno databázové schéma. Pro potřeby testování byla vytvořená databáze *DanceSirDB_CI*, která je kopií produkční databáze. Tato databáze byla naplněna testovacími daty a byl vytvořen uživatel *DbTester*, který do ní může přistupovat jako její vlastník prostřednictím SQL autentizace. Následně byly vytvořeny SQL Server unit testy, které jsou spouštěny proti testovací databázi *DanceSirDB_CI*. Unit test session a strukturu databázového projektu ve Visual Studiu je možné vidět na obrázku C.3 v přílohách práce. Na závěr byly vyexportovány výsledky unit testů do HTML formátu.



Obrázek 3.2: Relační schéma datového úložiště

Kapitola 4

Databázová aplikace - analýza, návrh a realizace

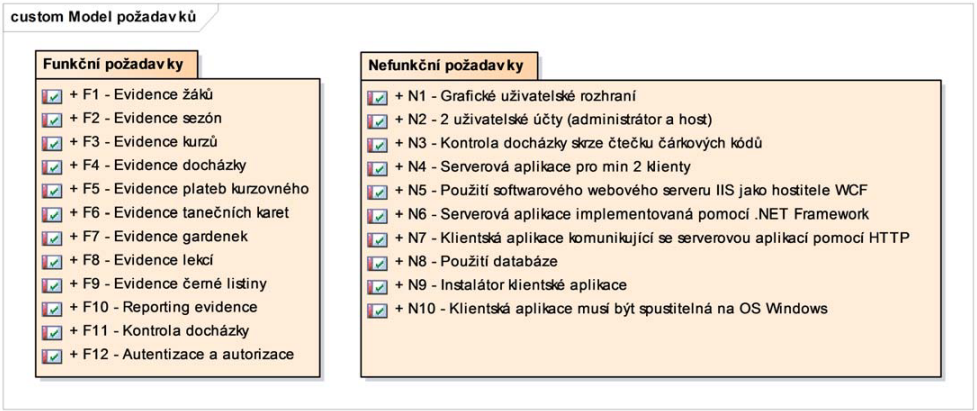
Tato kapitola se bude zabývat vývojem aplikace pro evidenci studentů.

4.1 Analýza

Tato sekce se zabývá analýzou požadavků a případy užití databázové aplikace.

4.1.1 Požadavky

Kapitola obsahuje popis všech požadavků, které jsou na nově vznikající aplikaci kladeny. Tyto požadavky jsou rozděleny na dvě základní části a to požadavky funkční a nefunkční. Požadavky jsou modelovány UML diagramem na obr. 4.1.



Obrázek 4.1: Model požadavků

4.1.1.1 Funkční

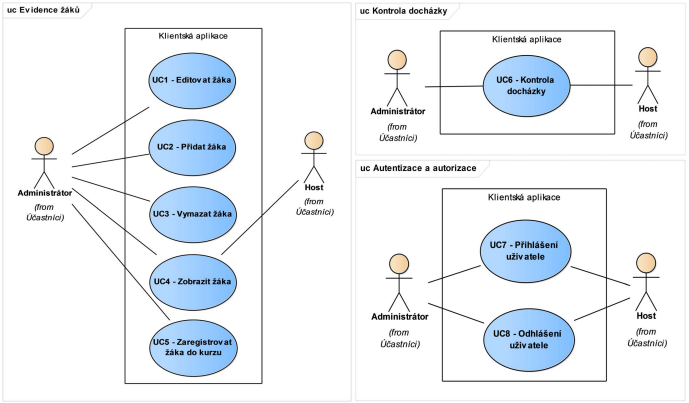
Sekce obsahuje popis funkčních požadavků, které jsou kladeny na aplikaci.

- **F1 - Evidence žáků**. Aplikace bude umožňovat evidovat žáky a také je registrovat do jednotlivých kurzů. Registrace je vícekroková (zařazení žáka, žák, taneční karty, gardenky a souhrn). Při prvním kroku registrace bude možné zobrazit stav kurzu s informativními údaji o počtu dívek a chlapců a zbývající kapacita v kurzu.
- **F2 - Evidence sezón**. Aplikace bude umožňovat evidovat sezóny.
- **F3 - Evidence kurzů**. Aplikace bude umožňovat evidovat kurzy.
- **F4 - Evidence docházky**. Aplikace bude umožňovat evidovat docházky.
- **F5 - Evidence plateb kurzovného**. Aplikace bude umožňovat evidovat platby kurzovného.
- **F6 - Evidence tanečních karet**. Aplikace bude umožňovat evidovat taneční karty.
- **F7 - Evidence gardenek**. Aplikace bude umožňovat evidovat gardenky.
- **F8 - Evidence lekcí**. Aplikace bude umožňovat evidovat lekce kurzů.
- **F9 - Evidence černé listiny**. Aplikace bude umožňovat evidovat černou listinu pro karty a žáky.
- **F10 - Reporting evidence**. Aplikace bude umožňovat vytvářet reporty pro jednotlivé evidence. Zobrazený report bude možné vytisknout či exportovat do souboru různých formátů (Excel, PDF, Word). Bude také možné zobrazit rozložení tisku, vyhledat text, nastavit stránku a měnit velikost zobrazení stránky.
- **F11 - Kontrola docházky**. Aplikace bude umožňovat kontrolovat docházku pro konkrétní lekci. Pokud taneční karta/gardenka konkrétního žáka patří do lekce, na semaforu se rozsvítí zelená a ozve se „souhlasná“ zvuková informace. Pokud taneční karta/gardenka nepatří do lekce, prošla již docházkou nebo je na černé listině - na semaforu se rozsvítí červená a ozve se „nesouhlasná“ zvuková informace.
- **F12 - Autentizace a autorizace**. Aplikace bude umožňovat přihlášení/odhlášení administrátora a hosta a pracovat s aplikací v rámci jejich oprávnění.

4.1.1.2 Nefunkční

Sekce obsahuje popis všech požadavků, které nesouvisejí přímo s funkčností systému, ale přesto jsou pro správný provoz systému důležité.

- **N1 - Grafické uživatelské rozhraní**. Aplikace bude nabízet jednoduché grafické desktopové rozhraní, které umožní ovládání myší a klávesnicí, tak jak jsou uživatelé zvyklí pracovat na operačním systému Windows. Pro možné rozšíření o nové typy klientů v budoucnu by serverová aplikace měla umět komunikovat i s jinými webovými GUI (formát JSON).
- **N2 - 2 uživatelské účty (administrátor a host)**. Administrátor má práva na zápis a čtení všech tabulek databáze. Host má právo na čtení a zápis do tabulky Attendance a čtení všech ostatních tabulek databáze.
- **N3 - Kontrola docházky skrze čtečku čárkových kódů**. Pro urychlení kontroly docházky žáků na jednotlivé lekce bude použita čtečka čárových kódů.
- **N4 - Serverová aplikace pro min 2 klienty**. Aplikace bude umět obsluhovat minimálně 2 klienty (Administrátor a Host).
- **N5 - Použití softwarového webového serveru IIS jako hostitele WCF REST**. Servisní orientace aplikace umožňuje komunikaci .NET komponent, ale i komponent jiných platforem, unifikovaným způsobem [10].
- **N6 - Serverová aplikace implementovaná pomocí .NET Framework**. Aplikace bude naprogramována v jazyce C# s využitím .NET frameworku, jehož součástí je i WCF [10].
- **N7 - Klientská aplikace komunikující se serverovou aplikací pomocí HTTP**. Bude použita RESTful architektura, která používá HTTP pro CRUD operace (Create/Read/Update/Delete) a dotazovací metody (např. GET, POST, PUT, DELETE) [10]. RESTful servisy budou vracet krátká strukturovaná data ve formátu JSON.
- **N8 - Použití databáze**. Relační databáze bude pod správou relačního SŘBD (Microsoft SQL Server) (viz závěr rešeršní části 2.10)
- **N9 - Instalátor klientské desktopové aplikace**. Jednoduchá instalace desktopového klienta zajistí větší komfort uživatelům.
- **N10 - Klientská aplikace musí být spustitelná na OS Windows**. Z důvodů programovacího jazyka .NET bude aplikace přizpůsobena výchozímu operačnímu systému - Windows - pro tento framework.



Obrázek 4.2: Důležité případy užití

4.1.2 Případy užití

Sekce obsahuje popis případů užití databázové aplikace pro evidenci studentů. Jedná se o funkcionality, které bude nově navrhovaná aplikace poskytovat svým uživatelům. Dále obsahuje popis všech uživatelů systému. Nejedná se o kompletní analýzu případů užití celé databázové aplikace, ale pouze důležité případy užití. Případy užití týkají se evidence jsou totožné. Ostatní části analýzy případu užití zde nejsou z důvodu rozsahu uvedeny.

4.1.2.1 Účastníci

Sekce obsahuje popis účastníků, kteří budou aplikaci využívat.

- **Administrátor** má práva na zápis a čtení všech tabulek databáze.
- **Host** má právo na čtení a zápis do tabulky Attendance a čtení všech ostatních tabulek databáze. Host je primárně určen pro kontrolu docházky.

4.1.3 Evidence

Jelikož jsou funkcionality pro jednotlivé evidence téměř totožné (mimo UC5), tak pro pochopení všech ostatních postačí uvést jen jeden případ evidence. Obrázek 4.2 část Evidence žáků popisuje funkčnosti aplikace související s evidencí žáků.

4.1.3.1 UC1 - Editovat žáka

1. Případ užití začíná při nahlášení změny v údajích žáka. Administrátor si od žáka vyžádá nové údaje.

2. V sekci Evidence vybere administrátor záložku Žák, kde při kliknutí myší na položku žáka v tabulce se vyplní aktuální údaje žáka do formuláře.

3. Tyto údaje jsou patřičně změněny a nahrány do systému po kliknutí na ikonu tužky.

4.1.3.2 UC2 - Přidat žáka

1. Případ začíná při potřebě přidání žáka mimo registrační formulář. Administrátor si od žáka vyžádá potřebné údaje.

2. Tyto údaje zadá v sekci Evidence v záložce Žák do formuláře.

3. Získané informace jsou patřičně nahrány do systému po kliknutí na ikonu plus.

4.1.3.3 UC3 - Vymazat žáka

1. Případ začíná při potřebě odstranit žáka. Odstranit žáka je možné v sekci Evidence v záložce Žák.

2. Při prvním kliknutí na ikonu s křížkem jsou v tabulce zobrazeny checkboxy u jednotlivých žáků a také checkbox pro výběr všech žáků v tabulce.

3. Administrátor vybere konkrétního žáka pro vymazání dvojitým kliknutím na checkbox.

4. Po druhém kliknutí na ikonu s křížkem se zobrazí upozornění a třetí klik žáka maže.

4.1.3.4 UC4 - Zobrazit žáka

1. Případ začíná při potřebě zobrazit informace o žákovi.

2. Administrátor/Host se přemístí do sekce Evidence a do záložky Žák.

3. Po kliknutí v tabulce na příslušného žáka jsou údaje žáka zobrazeny ve formuláři.

4.1.3.5 UC5 - Zaregistrovat žáka do kurzu

1. Případ užití začíná po příchodu neregistrovaného žáka, který má zájem o registraci do konkrétního kurzu. Administrátor spustí proces registrace ze své aplikace v sekce *Registrace* po kliknutí na ikonu panáčka s modrým plus. Otevře se registrační okno.

2. Prvním krokem je zařazení žáka. Administrátor vybere sezónu a kurz, do kterého bude žák zařazen.

3. V kroku *Žák* administrátor vyplní do formuláře údaje o žákovi nebo vybere existujícího žáka ze systému.

4. V kroku *Taneční karty* je žákovi přiřazena taneční karta.

5. V kroku *Gardenky* jsou žákovi přiřazeny potřebné gardenky.

6. Posledním krokem je souhrn a potřebné potvrzení registrace, čímž je registrace dokončena.

4.1.4 Kontrola docházky

Obrázek 4.2 část Kontrola docházky popisuje funkčnosti aplikace související s kontrolou docházky před taneční lekcí.

4.1.4.1 UC6 - Kontrola docházky

1. Případ užití začíná v případě kontroly docházky žáků na lekci daného kurzu. Administrátor/Host se přemístí do sekce *Docházka* v aplikaci.

2. Vybere patřičnou sezónu, kurz a lekci pro kontrolu karet zařazených do dané lekce.

3. Do velkého pole (textBox) zadá číslo karty.

4. Aplikace vydá patřičný zvukový tón a zobrazí výsledek kontroly na obrazovce, čím je kontrola docházky dokončena.

4.1.5 Autentizace a autorizace

Obrázek 4.2 část Autentizace a autorizace popisuje funkčnosti aplikace související s přístupem uživatele do aplikace a určení jeho práv v aplikaci.

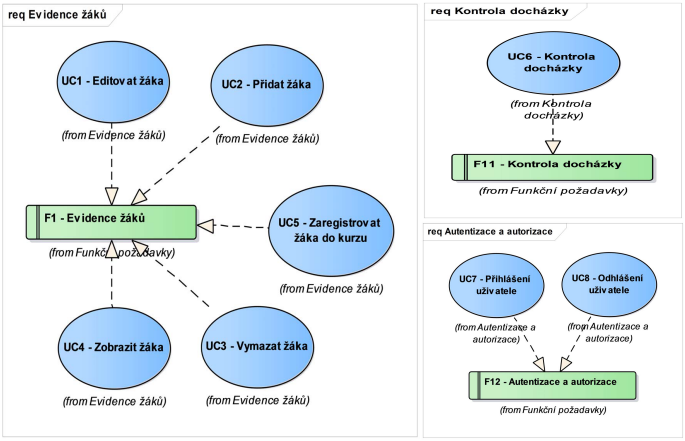
4.1.5.1 UC7 - Přihlášení uživatele

1. Případ užití začíná v případě potřeby použití aplikace. Uživatel (administrátor/host) klikne na ikonu aplikace, která byla vytvořena na jeho ploše při instalaci klientské aplikace.

2. Po spuštění aplikace uživatel klikne na ikonu dveří se šipkou směřující dovnitř.

3. Otevře se přihlašovací formulář, kde je potřeba vyplnit přihlašovací údaje uživatele.

4. Pokud jsou údaje korektní, formulář je uzavřen a uživatel je vpuštěn do systému pod konkrétními právy.



Obrázek 4.3: Vybrané mapování případů užití na požadavky

4.1.5.2 UC8 - Odhlášení uživatele

1. Případ užití začíná v případě potřeby opuštění aplikace.
2. Uživatel je odhlášen po kliknutí na ikonu dveří se šipkou směřující ven.

4.1.6 Mapování případů užití na požadavky

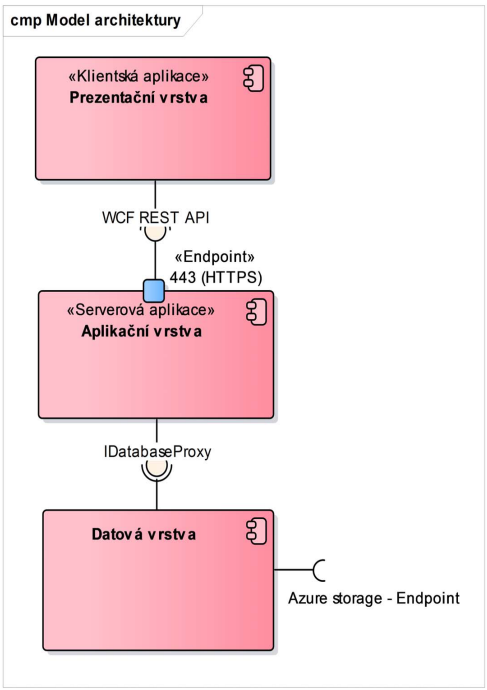
Zachycení plnění některých požadavků znázorňují diagramy na obrázku 4.3. Krok 4. (Žák) v případu užití u UC5, kdy administrátor vyplní do formuláře údaje o žákovi nebo vybere existujícího žáka ze systému, není stejným případem užití jako UC2. UC5 tady neodkazuje na UC2 pomocí relace «Include». UC5 lze realizovat a je úplný i bez UC2.

4.2 Návrh

S následující sekcí je čtenář obeznámen s návrhem architektury databázové aplikace pro evidenci studentů s pomocí nabytých znalostí ze sekce 2.6. Hlavním účelem následujícího obecného modelu nasazení je znázornit rozložení jednotlivých softwarových komponent na hardwarových zdrojích a jejich vzájemnou spolupráci. Poslední důležitou částí této sekce je výběr technologií a nástrojů podílejících se na vývoji jednotlivých částí těchto dvou modelů.

4.2.1 Model architektury

Jedná se o *třívrstvou architekturu* s nahraditelným tenkým klientem. Aplikační vrstva nabízí koncový bod WCF REST na zabezpečeném komunikačním kanálu pomocí protokolu SSL (Secure Sockets Layer). Toto RESTové API přístupné pomocí URL navýšilo modularitu tohoto systému. V budoucnu je

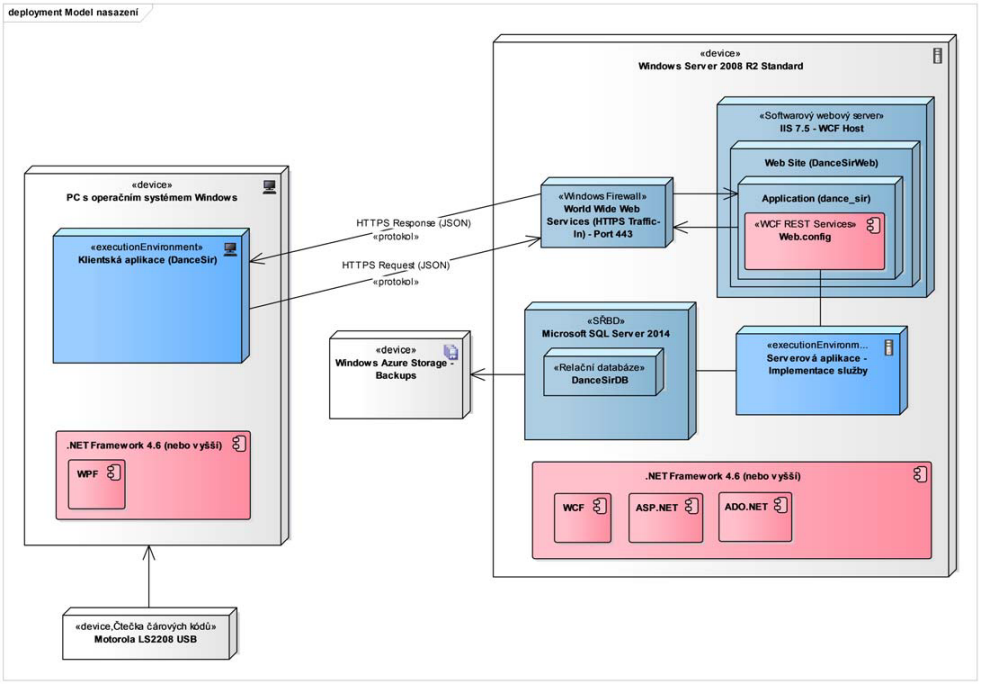


Obrázek 4.4: Model architektury

například možné vytvořit klientskou mobilní aplikaci, která bude disponovat svým vlastním mechanismem pro čtení čárových kódů, čímž by odpadl jeden hardwarový zdroj (čtečka čárových kódů) a celá aplikace by se podstatně zmenšila a byla by jednoduše přenosná. Tenký klient pouze poskytuje uživatelské rozhraní, přičemž logika zpracování dat a provozu (Aplikační služby) jsou umístěné na stejném serveru. Rozhraní `IDatabaseProxy` poskytuje spojení s datovou vrstvou. Zvolená architektura umožňuje vytvořit jednotkové testy pro logiku aplikační vrstvy „namockováním“ tohoto rozhraní. Tedy není potřeba reálné spojení s databází pro jednotkové testování prostřední vrstvy architektury.

4.2.2 Model nasazení

Sekce popisuje umístění jednotlivých částí na fyzická zařízení (viz obr. 4.5) a jejich stručný popis. Z důvodu rozsahu této práce a sofistikovanějšímu postupu nasazení serverové části aplikace nebude detailněji popsáno nasazení celé aplikace. Popis důležitých části konfigurací je možné prozkoumat v následujících sekcích. Klientská aplikace je vytvořena jako desktopová aplikace, takže je možné ji nainstalovat a spouštět přímo na osobním počítači. Následující sekce popisuje některé části Modelu nasazení a nástroje pro jejich vývoj.



Obrázek 4.5: Model nasazení

4.2.3 Zvolené technologie a nástroje

Při tvorbě různých částí databázové aplikace bylo použito množství nástrojů a technologií. Do této kategorie patří jak grafické, návrhové a vývojové nástroje tak i operační systém, kde je systém vyvíjen a testován. Některé systémy jsou open-source a další například nabízejí studentské licence. Následuje souhrn nejdůležitějších technologií a nástrojů použitých pro tuto bakalářskou práci.

4.2.3.1 Windows [26]

Všechny části aplikace byly vyvíjeny pod systémem Windows 10 a Windows Server 2008 R2.

4.2.3.2 Microsoft SQL Server [23]

SQL Server je relační databázový systém (RDBMS) od společnosti Microsoft. Microsoft SQL Server 2014 byl využit pro správu datového úložiště databázové aplikace.

4.2.3.3 Windows Azure [20]

Windows Azure Platform je cloudová platforma společnosti Microsoft. Využívá se k vytváření, hostování a škálování aplikací skrze Microsoft datacentra, která

jsou rozmístěná po celém světě. Součástí je samostatný operační systém Windows Azure. Tato bakalářská práce použila SQL Server zálohovaní pomocí URL (Windows Azure storage). Z důvodu rozsahu této práce nebude SQL Server zálohování prostřednictvím URL (Windows Azure storage) blíže specifikováno. Hlavním důvodem použití této služby byla ochrana a geografická redundance, což v praxi znamená, že data jsou replikovaná alespoň ve dvou datacentrech, které jsou na jiném místě. Nehrozí tedy ztráta dat v případě výpadku jednoho z center. Je potřeba zdůraznit, že tato služba není poskytována zdarma. Konfigurace byla provedena v rámci limitované zkušební verze.

4.2.3.4 IIS [18]

Microsoft IIS je druhý nejpopulárnější web server software. Podporuje Hypertext Transfer Protocol (HTTP) a další služby pro správu webů. Verze IIS 7.5 byla použita pro hostování WCF REST služby.

4.2.3.5 C# [16]

C# je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft. Při vývoji a testování byla použita verze C# 6.0.

4.2.3.6 .NET Framework [21]

Microsoft .NET Framework je nejrozšířenější platforma pro osobní počítače s operačním systémem Microsoft Windows. Klientská a serverová aplikace využívá k implementaci verzi 4.6.

4.2.3.7 WCF [25]

Windows Communication Foundation je sada knihoven, API a běhového prostředí, dohromady tvořící framework v rámci frameworku .NET, zajišťující komunikaci mezi aplikacemi a umožňující vytvářet servisně orientované aplikace.

4.2.3.8 WPF [27]

Windows Presentation Foundation (podmnožinou .NET Frameworku od verze 3.0), poskytuje vývojářům jednotný programovací model pro vytváření moderních desktopových aplikací, jak je možné vidět i u klientské aplikace.

4.2.3.9 ADO.NET [14]

ADO.NET je nedílnou součástí rozhraní .NET Framework a představuje množinu tříd nabízejících služby pro přístup k datům. Tyto služby byly využity při implementaci této databázové aplikace.

4.2.3.10 ASP.NET [15]

ASP.NET je součást .NET Frameworku pro tvorbu webových aplikací a služeb. WCF této aplikace používá ASP.NET mód kompatibility.

4.2.3.11 Visual Studio [24]

Visual Studio je komplexní soubor vývojářských nástrojů a služeb, které pomáhají vytvářet aplikace nejen pro platformu Microsoft. K vývoji některých částí této práce byla použita verze Microsoft Visual Studio 2015 Enterprise.

4.2.3.12 Motorola LS2208 [28]

Dle zadání bakalářské práce bude použita USB čtečka čárového kódu a to již zmíněného typu LS2208 značky Motorola. Čtečka čárových kódů Motorola LS2208 je ruční laserový snímač do obchodů i jednodušších aplikací.

4.2.3.13 Enterprise Architect [42]

Enterprise Architect (EA) od společnosti Sparx Systems je kompletní nástroj pro systémovou analýzu a návrh. Pomocí tohoto nástroje byly analyzovány a navrhnuty následující modely: Model požadavků, Use Case Model, Model architektury a Model nasazení.

4.2.3.14 Oracle Data Modeler [31]

Oracle SQL Developer Data Modeler obsahuje nástroje pro datové modelování na logické i fyzické úrovni. Využití těchto modelů na této práci je znázorněno v kapitole 3.

4.2.3.15 InkScape [43]

InkScape je editor vektorové grafiky, pomocí něhož byla vytvořena potřebná grafika pro klientskou aplikaci.

4.2.3.16 Git, SourceTree a Bitbucket [30]

SourceTree je Git Client pro Windows poskytující grafické rozhraní. Bitbucket je webová služba podporující vývoj softwaru při používání verzovacích nástrojů Git. Všechny důležité části této bakalářské práce (včetně samotné aplikace) byly verzovány pomocí těchto nástrojů, čímž se například zamezilo případným problémům se ztrátou dat.

4.2.3.17 MakeCert [19]

Jedná se o nástroj společnosti Microsoft pro vytváření testovacích certifikátů. Databázová aplikace je bude používat jako plnohodnotné certifikáty pro vývoj a testování (viz sekce 4.3).

4.2.3.18 Pvk2Pfx [22]

Pvk2Pfx (Pvk2Pfx.exe) je nástroj příkazového řádku, který kopíruje informace veřejného a soukromého klíče z *.spc*, *.cer* a *.pvk* souborů do Personal Information Exchange souboru. Použití toho nástroje lze nalézt v sekci 4.3.

4.2.3.19 Certmgr [17]

Nástroj Správce certifikátů (Certmgr.exe) spravuje certifikáty, seznamy důvěryhodných certifikátů (CTL) a seznamy odvolaných certifikátů (CRL). Jeho použití lze nalézt v sekci 4.3.

Další sekce se zaměřují na obecný návrh a realizaci nejdůležitějších částí databázové aplikace. Pro účely této práce a specializace mého oboru je podstatná část sekcí věnovaná konfiguracím. Jsou zde odkazy na zajímavé úseky zdrojového kódu v příloze práce. Kódy jsou většinou přizpůsobeny pro účely tohoto textu. Nejdříve jsou popsány společné části aplikace, poté následují sekce zaměřené na funkcionalitu konkrétní aplikace.

4.3 Vytvoření a import Self-signed certifikátů pro vývoj

Následující sekce popisuje vytvoření Self-signed certifikátů X.509 (Root, server a client) za pomocí programu *makecert.exe*. V případě nasazení aplikace do reálného prostředí by měl být použit SSL certifikát podepsaný privátním klíčem důvěrných certifikačních autorit, ale pro vývoj a testování zcela splní svůj účel Self-signed certifikát.

Certifikát je možno obdržet od velkých certifikačních autorit. Tyto certifikáty jsou většinou přidány mezi důvěryhodné certifikáty daného zařízení, takže již není potřeba distribuovat veřejné klíče k jednotlivým uživatelům aplikace. Určitou překážkou jsou poplatky za použití těchto certifikátů v komerční sféře. Tyto certifikáty je možno obstarat například pomocí certifikačních autorit GlobalSign a StartCom. Poslední zmíněná certifikační autorita nabízí zdarma nelimitované StartSSL certifikáty, ale pouze pro nekomerční použití (více informací na [41]).

4.3.1 Vytvoření klíčů a self-signed certifikátu certifikační autority

Existuje několik nástrojů (např. Makecert a OpenSSL), které umožňují vytvoření vlastního certifikátu. Těmto certifikátům klienti (např. webový prohlížeč nebo emailový klient) nedůvěřují a je zapotřebí klientům dodat veřejný klíč tohoto certifikátu.

```
1 Microsoft Windows [Version 10.0.10586]
2 (c) 2015 Microsoft Corporation. All rights reserved.
3
4 c:\> makecert.exe -r -n "CN=DanceSirCARoot" -pe -a sha512 -len 4096 -cy authority -sv
      CARoot.pvk CARoot.cer
```

Kód 4.1: Příkazový řádek s příkazy pro vytvoření DanceSirCARoot

Kvůli rozsahu této práce neuvádím význam parametrů jednotlivých nástrojů (viz jednolivé manuálové stránky [19], [22] a [17]). Prvním z příkazů je vytvoření certifikátu certifikační autority. Po spuštění tohoto příkazu vzniknou soubory *CARoot.pvk* a *CARoot.cer*. První soubor je tzv. soukromý klíč a je zapotřebí ho utajit. Kořenový certifikát představuje druhý soubor, který je zapotřebí importovat na zařízení všech klientů (Trusted Root Certification

Authorities Store skrze Microsoft Management konzoli nebo *certmgr.exe* nástroj) a získat si tím důvěru k vytvořené CA.

Soubory s příponami *.pvk* a *.pfx* je potřeba chránit, jelikož první ze souborů obsahuje soukromý klíč pro soubor s příponou *.cer* a druhý soubor obsahuje jak *.cer*, tak i *.pvk* soubory, což v případě odcizení znamená, že ostatní by si mohli podepsat své veřejné klíče bez svolení certifikační autority. Sdílet se může jen soubor s příponou *.cer*, který obsahuje jen veřejný klíč.

4.3.2 Důvěra klienta certifikační autoritě DanceSirCARoot

Následující příkaz spuštěný na klientském počítači pod právy administrátora naimportuje *CARoot.cer* (obsahuje veřejný klíč) do *Trusted Root Certificate* úložiště klienta. Klient následně důvěřuje certifikační autoritě DanceSirCARoot.

```
1 c:\> certmgr.exe -add CARoot.cer -s -r localMachine Root
```

Kód 4.2: Import *CARoot.cer*

4.3.2.1 Vytvoření klíčů a certifikátu pro IIS

Následující příkaz použije výše zmíněnou CA pro vytvoření serverového certifikátu, typicky pro web server. V případě této aplikace se jedná o IIS Web server.

```
1 c:\> makecert -iv CARoot.pvk -ic CARoot.cer -n "CN=147.32.200.115" -pe -sv
      WebServerSSL.pvk -a sha512 -len 4096 -sky exchange -eku 1.3.6.1.5.5.7.3.1
      WebServerSSL.cer
2 c:\> certmgr.exe -add CARoot.cer -s -r localMachine Root
3 c:\> pvk2pfx.exe -pvk WebServerSSL.pvk -spc WebServerSSL.cer -pfx WebServerSSL.pfx -po
       PfxWebPfx2016
```

Kód 4.3: Příkazový řádek s příkazy pro vytvoření web server certifikátu

Druhý příkaz spuštěný na serveru pod právy administrátora naimportuje *CARoot.cer* (obsahuje veřejný klíč) do *Trusted Root Certificate* úložiště serveru. Server následně důvěřuje certifikační autoritě DanceSirCARoot.

Poslední z příkazů používá nástroj *pvk2pfx.exe* (je součástí Windows SDK) pro kopírování informací veřejného klíče a soukromého klíče z *CARoot.pvk* a *CARoot.cer* do *.pfx* (Personal information exchange) souboru. Tento soubor je následně importován do certifikátů web serveru IIS (viz také sekce 4.5.1.2).

4.4 Autentizace a autorizace

Tato sekce se zabývá funkčním požadavkem F12 (Autentizace a autorizace) a nefunkčním požadavkem N2 (2 uživatelské účty - administrátor a host). Následuje popis důležitých implementačních částí těchto dvou požadavků začínaje od klientské části.

Po přihlášení uživatele jsou vyplněné přihlašovací údaje posílány prostřednictvím POST *login* dotazu (viz také sekce 4.7.1) po zabezpečeném komunikačním kanálu (HTTPS) na server, bez možnosti použití HTTP. Server poskytuje certifikát pro prokázání své identity, aby bylo zajištěno, že komunikace probíhá se správným serverem (ochrana proti útoku *Man in the middle*). Ten který je podepsaný certifikační autoritou (DanceSirCaRoot), které důvěřuje klient (viz sekce 4.3). Třída `CredentialsValidator` ověří přihlašovací údaje tak, že porovná zakódovaný tvar hesla uživatele z databáze s jinou zakódovanou hodnotou, která byla vytvořena pomocí přijatého uživatelského hesla a *kryptografické soli*. Pokud autentizace proběhla v pořádku, následuje autorizace. Při autorizaci se zjišťují role, do kterých přihlášený uživatel patří. Tato aplikace poskytuje dva uživatelské účty Administrator a Host. První účet spadá do role Administrators a druhý do role AttendanceUsers. Pro přihlášeného uživatele je vytvořen dočasný (životnost 20 minut) *autentizační token*, který je pod správou třídy `GlobalCachingProvider`. Autentizační token je použít pro ověření identity uživatele. Autentizační token se používá namísto hesla, které by muselo být přenášeno při každé autentizaci. Odpovědí pro uživatele je struktura datového kontraktu `LoginResponse`.

```
1  [DataContract]
2  public sealed class LoginResponse
3  {
4    [DataMember(Name = "username")]
5    public string UserName { get; set; }
6
7      [DataMember(Name = "token")]
8      public string Token { get; set; }
9
10     [DataMember(Name = "roles")]
11     public string[] Roles { get; set; }
12 }
```

Kód 4.4: Datový kontrakt LoginResponse

Klientská třída `GlobalCachingProvider` spravuje přijatý autentizační token ze serveru. Tento token použije klientská aplikace při každém dalším dotazu jako „autentizaci“. Po zpracování odpovědi ze serveru se GUI klientské aplikace přizpůsobí přihlášenému uživateli podle jeho práv. Uživateli Host nejsou například zobrazeny tlačítka přidat, vymazat a editovat. Někoho by možná mohlo napadnout, že by tuhle funkcionalitu mohl obejít přímým dotazem na server prostřednictvím nějakého REST klienta. Na takovou situaci je serverová aplikace připravená. Pomocí metody `Application_BeginRequest` serverové třídy `Global.asax` je při každém požadavku prováděna autentizace a autorizace uživatele. Autentizace prostřednictvím autentizačního tokenu a autorizace pomocí *Role-Based Security*, kdy jednotlivé operační kontrakty požadují účast uživatele v patřičné roli. V následující ukázce má uživatel Administrator a Host přístup k operačnímu kontraktu `GetAttendances`, protože spadají do rolí Administrators a AttendanceUsers. Dále platí, že Host má

právo na operační kontrakty, které čtou a zapisují do tabulky Attendance a na operační kontrakty, které čtou ze všech ostatních tabulek databáze. Uživatel Administrator má práva na všechny operační kontrakty. Tímto je pokryt nefunkční požadavek N2 4.1.1.2.

```
...
protected void Application_BeginRequest(object sender, EventArgs e)
{
    _customPrincipal = Thread.CurrentPrincipal as CustomPrincipal ?? new
        CustomPrincipal();
    var token = HttpContext.Current.Request.Headers["Token"];
    var recognizeUserViaToken = GlobalCachingProvider.Instance.
        RecognizeUserViaToken(token);
    var userTokenInfo = recognizeUserViaToken.Split(':');
    if (userTokenInfo.Length != 2 || userTokenInfo[0] == null || userTokenInfo
        [1] == null)
    {
        _customPrincipal.Identity = new AnonymousIdentity();
    }
    else
    {
        var user = new User(userTokenInfo[0], new[] { userTokenInfo[1] });
        _customPrincipal.Identity = new CustomIdentity(user.Name, user.Roles);
    }
}
...
[PrincipalPermission(SecurityAction.Demand, Role = "Administrators")]
[PrincipalPermission(SecurityAction.Demand, Role = "AttendanceUsers")]
[Description("Get all attendances")]
[WebInvoke(Method = "GET", UriTemplate = "attendances", ResponseFormat =
    WebMessageFormat.Json)]
[OperationContract]
public AttendanceResponse GetAttendances()
{
    return new AttendanceResponse
    {
        Attendances = _databaseProxy.GetAttendances()
    };
}
...
```

Kód 4.5: Metoda Application_BeginRequest a operační kontrakt GetAttendances

Najednou lze používat minimálně 2 klienty. Jeden je přihlášen s uživatelským účtem Administrator a druhý s uživatelským účtem Host. Použití vícero klientů najednou se stejným uživatelským účtem není možné. Omezení použití dvou „hostů“ lze vyřešit například přidáním dalšího uživatelského účtu Host2, který má přidanou roli AttendanceUsers.

4.5 Windows Server 2008 R2

Následující sekce popisuje nejdůležitější části konfigurace a nastavení pro WCF REST službu na Windows Server 2008 R2.

4.5.1 Konfigurace webového serveru IIS

Pomocí nástroje *Server Manager* byla na server nainstalována role *Web Server (IIS)* a *.NET Framework 3.5.1 Feature*. Pro roli IIS byly nainstalovány služby rolí IIS Common HTTP Features (povolující například HTTP Errors). Pro roli *.NET Framework 3.5.1 Feature* je potřeba také aktivovat samotné WCF a WCF HTTP.

4.5.1.1 Hostování WCF REST v IIS

Na obrázku 4.6 je znázorněna konfigurace vytvořené *Web Site*. Povolený protokol HTTP (zahrnuje jak HTTP, tak i HTTPS) specifikuje protokol přístupu k aplikaci. Tato *Web Site* je zařazena do *Application Pool DanceSirWeb*. Tento pool má nastaven *Integrated Mode*, protože ho vyžaduje URL Routing Modul (viz 4.5.2). Pool také používá .NET Framework Version v4.0.30319, což umožňuje použít .NET 4.6 aplikaci. Aby identita (ApplicationPoolIdentity) pool aplikace *DanceSirWeb* měla přístup k *Web Site* složkám a k souborům, byl k nim povolen přístup pro skupinu *IIS_IUSRS* (členem skupiny je „IIS APPPOOL \DanceSirWeb“) s patřičnými oprávněními (Read and execute, Read, List folder contents). Nastavení oprávnění Full Control není žádoucí. V případě kompromitace aplikační identity by mohl mít vymazán celý obsah *Web Site*. *Web Site Bindings* je nastaven na HTTPS s SSL certifikátem (viz 4.5.1.2).

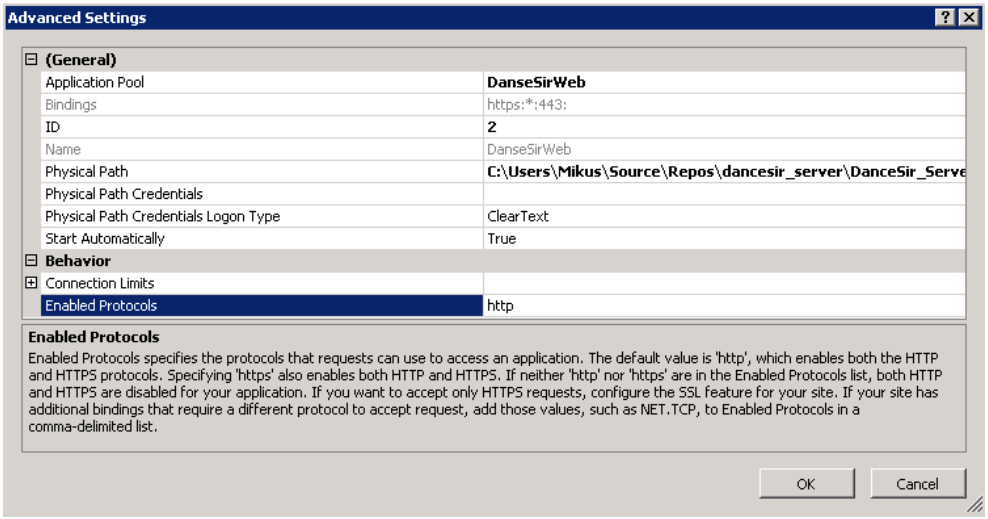
Web Site obsahuje aplikaci *dance_sir*, jejíž fyzická cesta směřuje do složky (`src/DanceSir_Server/Host`) s konfiguračním souborem `Web.config` (viz příloha B.1 a sekce 4.5.2).

4.5.1.2 Přidání vazby protokolu SSL

Před samotným přidáním vazby protokolu SLL je zapotřebí importovat soubor *WebServerSSL.pfx* do certifikátů web serveru IIS. Tyto certifikáty může následně webový server využít pro *Web Site* (*DanceSirWeb*) a její konfiguraci SSL. Dalším krokem je přidání vazby webu pomocí dialogového okna. V tomto okně je zvoleno HTTPS pro typ a popisný název certifikátu, který byl přidán do certifikátů web serveru IIS.

4.5.1.3 Konfigurace virtuálního adresáře pro protokol SSL

Pro virtuální adresář (*dance_sir*) je zapotřebí nastavení protokolu SSL. V prostředí okna virtuálního adresáře v části IIS se nachází *SSL Settings* část, v níž



Obrázek 4.6: Nastavení Web Site v IIS

je požadován protokol SSL a klientské certifikáty jsou ignorovány. Konfigurací služby WCF REST pro přenos HTTP zabezpečení se zabývá následující sekce.

4.5.2 Konfigurace WCF hostované IIS

Windows Communication Foundation služba je konfigurována pomocí .NET Framework technologie. XML prvky jsou přidány do souboru *Web.config* pro IIS, který je hostitelem služby Windows Communication Foundation (WCF). Administrativní nastavení WCF pomocí konfiguračního souboru má své výhody oproti imperativnímu přístupu (konfigurace v kódu programu). Tento textový XML soubor umožňuje měnit konfiguraci bez nutnosti kompilace kódu. Ukázka konfiguračního viz příloha B.1.

Hlavní výhodou hostování služby v IIS je automatické spuštění hostovaného procesu při první žádosti klienta. Tento proces využívá funkce služby IIS, jako je například proces recyklace, vypnutí při nečinnosti a monitoring běhu. Za hlavní nevýhodu je považována podpora pouze protokolu HTTP [10], což neodporuje návrhu této aplikace. Tedy nefunkční požadavek N7 (viz model požadavků 4.1) je stále pokryt. Následuje popis důležitých částí konfigurace WCF REST.

Endpoint služby je vždy tvořen třemi základními prvky, které se uvádí pod názvem ABC (Address, Binding a Contract). První prvek `Address` uvádí kde se služba nachází. V případě této RESTové služby je adresa využita k verzování RESTful API (`https://147.32.200.115/dance_sir/v1`), aby bylo možno zajistit kompatibilitu s různými verzemi klientů. Jedná se o jakousi pojistku do budoucna celé aplikace.

```
...
<service name="DanceSir.Backend.Service" behaviorConfiguration="
    ServiceBehaviour" >
   <endpoint address="v1" binding="webHttpBinding" bindingConfiguration="
       webHttpTransportSecurity" behaviorConfiguration="webHttpBehavior"
       contract="DanceSir.Contracts.IService" />
</service>
...
```

Kód 4.6: Konfigurační soubor Web.config - nastavení servisy

Prvek `Binding` určuje jak služba komunikuje. Binding `webHttpBinding` povoluje RESTové webové služby aplikace přijímat jednoduché volání přes webový protokol HTTP. Pokud je nastaven bezpečnostní mód `Transport` pro tento binding, WCF používá bezpečnostní komunikační protokol [10]. V případě naší služby se jedná o HTTPS, přičemž SSL musí být nakonfigurován na IIS, aby přenášená data byla šifrována, například pro předávání zašifrovaných informací mezi klientem a serverem.

```
...
<bindings>
  <webHttpBinding>
     <binding name="webHttpTransportSecurity">
      <security mode="Transport" />
     </binding>
  </webHttpBinding>
</bindings>
...
```

Kód 4.7: Konfigurační soubor Web.config - bindingConfiguration

Prvek `Contract` určuje, co služba dělá a co od ní můžeme očekávat. V případě webové služby aplikace se jedná o rozhraní `IService`, který je označen atributem `ServiceContract`. Toto rozhraní implementuje jiná rozhraní (např. `ILogin`, `ISeason`, `ICourse` a `IPupil`) se stejným atributem poskytující metody, které jsou zahrnuty do celkového kontraktu služby. Takové metody jsou označeny atributem `OperationContract`. Vlastní kód služby pak implementuje toto rozhraní `IService`.

```
[ServiceContract]
public interface ISeason
{
    [Description("Get all seasons")]
    [WebInvoke(Method = "GET", UriTemplate = "seasons", ResponseFormat =
        WebMessageFormat.Json)]
    [OperationContract]
    SeasonResponse GetSeasons();
}
```

Kód 4.8: Ukázka Service Contract - ISeason

Vlastní objekty používané jednotlivými metodami služby nesou atribut `DataContract` a jejich datové členy atribut `DataMember`.

```
[DataContract]
public sealed class SeasonResponse
{
    [DataMember(Name = "seasons")]
     public List<Season> Seasons { get; set; }
     ...
}
```

Kód 4.9: Ukázka Data Contract - část SeasonResponse

Vlastnosti chování služby a jejího endpointu jsou nastaveny pomocí sad pravidel, které obsahuje element `behavior`. ServiceDebug obsahuje následující pravidlo `includeExceptionDetailInFaults`, které by mělo být povoleno jen v době vývoje aplikace, kdy je třeba získat více informací o chybách přicházejících ze služby. Výchozí nastavení obsahuje hodnotu `false`. Pravidlo webHttp konfiguruje `WebHttp` chování na koncovém bodě, což znamená, že služba používá WCF Web HTTP Programming Model umožňující konfiguraci jiných než SOAP koncových bodů (REST). V případě služby aplikace se jedná o koncový bod vracející nezabalenou odpověď v JSON formátu nevybraného automaticky WCF s možností *Help page*.

```
...
    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceBehaviour">
          <serviceDebug includeExceptionDetailInFaults="false"/>
          ...
        </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="webHttpBehavior">
          <webHttp defaultBodyStyle="Bare" helpEnabled="true"
              defaultOutgoingResponseFormat="Json"
              automaticFormatSelectionEnabled="false"/>
        </behavior>
      </endpointBehaviors>
    </behaviors>
...
```

Kód 4.10: Konfigurační soubor Web.config - behaviors

S pomocí ASP.NET byly využity některé sofistikované směrovací schopnosti (ASP.NET Routing Integration). Třída `ServiceHostFactory` umožňuje, stejně jako třída `WebServiceHostFactory`, používat WCF REST služby.

Když přijde HTTP požadavek, třída `ServiceHostFactory` pomocí třídy `Service` namapuje požadavek na konkrétní operaci v závislosti na použitém URI a HTTP požadavku. Prázdný řetězec v konstruktoru `ServiceRoute` může být například nahrazen řetězce pro rozlišení verze REST API „api/v1“ (`https://147.32.200.115/dance_sir/api/v1/attendances`).

```
protected void Application_Start(object sender, EventArgs e)
{
    RouteTable.Routes.Add(new ServiceRoute(string.Empty, new
        ServiceHostFactory(), typeof(Service)));
}
```

Kód 4.11: Global.asax

Toto směrování například umožnilo odstranit příponu *svc* k dané službě. Pro použití ASP.NET Routing Integration funkce je potřeba zapnout ASP.NET kompatibilitu.

```
...
  <system.serviceModel>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
    ...
  </system.serviceModel>
...
```

Kód 4.12: ASP.NET kompatibilita zapnuta

Toto nastavení také umožňuje přístup k `HttpContext.Current` objektu, pomocí kterého je možné přistupovat k většině standardním funkcím ASP.NET pro dotaz a odpověď. Pokud ASP.NET je povolena pro tuto aplikaci 4.12. Je potřeba přidat atribut AspNetCompatibilityRequirements pro službu s nastavením RequirementsMode jako „Allowed“ nebo „Required“.

```
...
[AspNetCompatibilityRequirements(RequirementsMode =
    AspNetCompatibilityRequirementsMode.Allowed)]
public partial class Service : IService
{
  ...
}
...
```

Kód 4.13: AspNetCompatibilityRequirementsMode

Dotazovací metody PUT a DELETE nejsou ve výchozím nastavení podporovány „handlerem“ *ExtensionlessUrl-Integrated-4.0* a je zapotřebí jejich povolení [13].

```
...
  <system.webServer>
    <handlers>
      <remove name="ExtensionlessUrl-Integrated-4.0"/>
      <add name="ExtensionlessUrl-Integrated-4.0" path="*." verb="GET,HEAD,
          POST,DELETE,PUT" type="System.Web.Handlers.TransferRequestHandler"
          preCondition="integratedMode,runtimeVersionv4.0"/>
    </handlers>
  </system.webServer>
...
```

Kód 4.14: Povolení metod PUT a DELETE

4.5.3 Povolení portu ve Windows firewall

Během instalace role *Web Server (IIS)* na Windows Server 2008 R2 instalační proces přidal *Inbound Windows Firewall* pravidla, které povolují komunikaci pro vybrané služby rolí IIS. Díky nainstalovaným službám rolí, které souvisí s HTTP a HTTPS, byly nainstalovány pravidla *World Wide Web Services HTTP Traffic In* a *World Wide Web Services HTTPS Traffic In*. Tyto automaticky povolené pravidla povolují komunikaci pro HTTP na portu 80 a HTTPS na portu 443.

4.6 Serverová aplikace

Serverová aplikace je součástí modelu architektury 4.2.1 aplikační vrstvou. Serverová aplikace je v určitém smyslu „middleware“, který zajišťuje operace prováděné vstupními požadavky (WCF REST) a daty (třída `IDatabaseProxy`). Implementuje WCF REST služby, které realizují požadované chování aplikace z pohledu aplikační logiky. Obsahuje také WCF REST konfigurační soubor `Web.config`, jehož konfigurace je detailněji rozepsána v sekci 4.5.2. Důležitou součástí serverové aplikace je také implementace autentizace a autorizace.

4.6.1 Správa instancí - podpora více klientů

Tato sekce popisuje nastavení vytváření instancí WCF REST služby. Instance je spravována s využitím vlastnosti `InstanceContextMode`, která nabývá hodnoty `PerCall`. Tento mód zajistí, že vždy při zavolání jakékoliv operace je vytvořena nová instance služby s vlastní instancí třídy `DatabaseProxy`, která zajišťuje komunikaci serverové aplikace s databází.

```
1  ...
2  [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
3  public partial class Service : IService
4  {
5      private readonly IDatabaseProxy _databaseProxy;
6      private readonly ILogManager _logManager = new ServiceLogManager();
7      public Service() : this(new DatabaseProxy()){}
8      public Service(IDatabaseProxy databaseProxy) { _databaseProxy =
           databaseProxy; }
9  }
10 ...
```

Kód 4.15: Ukázka použití atributu ServiceBehavior

4.6.2 Testování

Za pomocí mockovacího frameworku NSubstitute a MSTest frameworku pro tvorbu jednotkových testů byly napsány jednotkové testy na nejdůležitější metody služby, pro kterou byla mockována třída `DatabaseProxy` komunikující

s databází. Toto odstínění třídy komunikující s databází od implementační logiky služby umožnilo její testovatelnost. Pro toto testování byl použit Unit test vzor AAA (Arrange, Act, Assert). Ukázkový test viz B.3.

4.7 Klientská aplikace

Klientská aplikace je vytvořena pomocí WPF (Windows Presentation Foundation) frameworku. Jelikož detailnější popis této problematiky přesahuje rámec této práce, byly vybrány jen podstatné konfigurační a implementační části, které se vážou na předchozí sekce. Příloze C.1 obsahuje náhled na aplikaci a příloha C.2 obsahuje náhled na okno Registrace uživatele.

4.7.1 Komunikace se serverovou aplikací

Klient komunikuje se serverovou aplikací za prostřednictvím abstraktní třídy `WebRequest` (System.Net), kde statická metoda `Create(string)` inicializuje novou WebRequest instanci pro zadané schéma identifikátoru URI (viz následující sekce 4.7.1.2).

Je provedena konverze na odvozenou třídu `HttpWebRequest`, která poskytuje implementaci zaměřenou na HTTP. Déle je specifikována HTTP metoda, specifikace formátu dat zasílaných v požadavku a hlavička nesoucí informaci o autentizaci a autorizaci. Pokud HTTP dotaz proběhne v pořádku, je odpověď ze serveru deserializována za pomocí třídy `DataContractJsonSerializer` do patřičného datového kontraktu (`LessonResponse`).

```
1  public LessonResponse GetLessons(int courseId)
2  {
3   string url = $"{_backendScheme}://{_backendAddress}:{_backendPort}//{_appName
       }//{_version}//lessons?courseId=" + courseId;
4  HttpWebRequest request = WebRequest.Create(url) as HttpWebRequest;
5  request.Method = "GET";
6  request.ContentType = "application/json";
7  request.Headers["Token"] = TokenManager.GetToken();
8
9  using (WebResponse response = request.GetResponse())
10 {
11     DataContractJsonSerializer jsonSerializer = new DataContractJsonSerializer
           (typeof(LessonResponse));
12     using (Stream dataStream = response.GetResponseStream())
13     {
14         return (LessonResponse)jsonSerializer.ReadObject(dataStream);
15     }
16 }
17 }
```

Kód 4.16: Ukázka metody komunikující se serverem - GetLessons

4.7.1.1 Čtení čárových kódů

V dnešní moderní době nemusíme číst čárové kódy klasickými čtečkami, ale můžeme využít chytré mobilní telefony nebo zařízení podobného typu, které umožňují instalaci aplikací nahrazující čtečku. Tato možnost je sice pohodlnější a snižuje z určité části provozní náklady, ale musíme brát v potaz prostředí nasazení. Pokud je vyžadovaná přesnost či spolehlivost spíše využijeme první možnost volby. Dle zadání bakalářské práce bude použita USB čtečka čárového kódu a to následujícího typu LS2208 značky Motorola, která přebírá chování klávesnice a pro korektní čtení kódu je zapotřebí být v režimu anglické klávesnice.

4.7.1.2 Konfigurační soubor klientské aplikace

V konfiguračním souboru klientské aplikace *App.config* je možné nastavit hodnoty pro přístup k REST API. Do budoucna je možné rozšířit klienta o sekci *Nastavení* a dát Administrátorovi klientské aplikace více pravomocí prostřednicím GUI aplikace.

```
...
  <appSettings>
    <add key="BackendScheme" value="https" />
    <add key="BackendAddress" value="147.32.200.115" />
    <add key="BackendPort" value="443" />
    <add key="AppName" value="dance_sir" />
    <add key="Version" value="v1" />
  </appSettings>
...
```

Kód 4.17: Část konfiguračního souboru klientské aplikace App.config

4.7.2 Instalátor klientské aplikace

Klientskou aplikaci je možné nainstalovat na osobní počítač s operačním systém Windows (viz také 4.2.2) pomocí instalačního souboru *DanceSir_v1.msi*, který obsahuje explicitní instrukce pro instalaci a odinstalaci programu. Instalátor byl vytvořen pomocí Visual Studio Setup projektu. Jako požadavek na instalaci byl stanoven .NET Framework 4.6. Pokud ním klientský počítač nedisponuje, instalátor nabídne jeho stažení z webové stránky a vyzve k jeho instalaci, jinak instalace aplikace není možná. Po úspěšné instalaci se na ploše uživatele vytvoří zástupce aplikace. Pomocí ovládacího panelu *Programy a funkce* lze odinstalovat tento program ze systému Windows.

4.7.3 Testování

Za pomocí mockovacího frameworku NSubstitute a MSTest frameworku pro tvorbu jednotkových testů byly napsány jednotkové testy na nejdůležitější

části ViewModel vrstvy, pro kterou byla mockována Model vrstva návrhového vzoru MVVM pro WPF aplikaci. Pro toto testování byl použit Unit test vzor AAA (Arrange, Act, Assert). Ukázkový test viz B.2.

4.8 Dokumentace

Následující sekce popisuje ve zkratce jednotlivé části dokumentace databázové aplikace přiložené k CD bakalářské práce. Kód jednotlivých projektů (DanceSir_Server a DanceSir_Client) je řádně okomentován, což umožnilo vygenerovat sofistikovanou technickou dokumentaci pomocí nástroje *Document! X*. Tento nástroj umožnil zdokumentovat i webové služby jako WCF REST Help Page, čímž vznikla přehledná dokumentace RESTového API aplikace se všemi podstatnými informacemi (uri, method, response, request). Dokumentaci je možné procházet pomocí webového prohlížeče nebo Microsoft Compiled HTML Help.

Kapitola 5

Přínosy, výsledky a možné rozšíření

V této kapitole budou v návaznosti na realizaci bakalářské práce podrobněji popsány její důležité přínosy, výsledky a možnosti rozšíření.

5.1 Hlavní přínosy práce

- Práce poskytuje čtenáři koncepčně vhodný zdroj informací poskytující průřez hlavními databázovými technologiemi, návrhy, postupy, bezpečností a novinkami, po kterých je čtenář schopen vybrat vhodnou databázi společně se systémem řízení báze dat.
- Praktická část je vhodný zdroj informací pro .NET komunitu, jelikož jsem za celou dobu realizace této práce a praxe v této oblasti nenašel detailní jednotný popis konfigurace webového serveru IIS jako hostitele WCF REST služeb, token autentizace společně s Role Based Authorization a vytvoření potřebných certifikátů pro zabezpečenou komunikaci mezi webovým serverem a klientem.

5.2 Možnosti rozšíření

- V budoucnu je možné vytvořit klientskou mobilní aplikaci, která bude disponovat svým vlastním mechanismem pro čtení čárových kódů, čímž by odpadl jeden hardwarový zdroj (čtečka čárových kódů) a celá aplikace by se podstatně zmenšila a byla by jednoduše přenosná.
- Rozšíření databázové aplikace o funkční požadavky tisk čárových kódů, vytváření uživatelských účtů, uživatelská konfigurace aplikace, emailová či SMS notifikace, vložení fotky registrovaného uživatele a mnoho dalších podobných rozšíření.

Závěr

Cílem práce bylo vybrat vhodnou databázi pro evidenci studentů, realizovat tuto databázi, naprogramovat a vhodně nakonfigurovat serverovou aplikaci pro minimálně dva klienty (Administrátor a Host), konfigurovat softwarový webový server IIS hostující WCF REST služby, naprogramovat klientskou aplikaci komunikující se serverovou aplikací pomocí standardního komunikačního protokolu, omezit prostor privilegií uživatelům klientské aplikace (autentizace a autorizace) a využít také čtečku čárových kódů. Všechny body zadání byly splněny a zároveň byly splněny důležité body mimo rozsah zadání.

Pro lepší použitelnost databázové aplikace v praxi jsem stanovil další cíle. Nadstandardním cílem práce bylo zaměřit se na výběr databáze pro strukturu (DB + SŘBD = DBS), provést analýzu a návrh jednotlivých částí aplikace, zabezpečit komunikaci mezi klientem a webovým serverem s pomocí vytvořených testovacích certifikátů, vytvořit modul Reporting evidence, vytvořit uživatelské okno pro registraci uživatelů a kontrolu docházky, vytvoření dokumentace aplikace a REST API. Klientskou aplikaci je možné nainstalovat na osobní počítač s operačním systém Windows.

Databázová aplikace takového rozsahu zasahuje do širokého spektra vývoje a znalostí, čemuž odpovídala i její časová náročnost, spíše typická pro týmovou práci různých specialistů. Konfigurace webového serveru byla vybrána jako hlavní problematika praktické části této práce. Pro nasazení aplikace do produkce je vhodné provést další testování, včetně manuálního, aby se předešlo případným chybám na reálných tanečních kurzech.

Práci je do budoucna možné rozšířit o klientskou mobilní aplikaci, která bude disponovat svým vlastním mechanismem pro čtení čárových kódů, čímž by odpadl jeden hardwarový zdroj (čtečka čárových kódů) a celá aplikace by se podstatně zmenšila a byla by jednoduše přenosná. Dalšími možnými rozšířeními databázové aplikace jsou funkční požadavky jako například tisk čárových kódů, vytváření uživatelských účtů, uživatelská konfigurace aplikace, emailová notifikace a mnoho dalších podobných rozšíření.

Tato bakalářská práce byla pro mě profesně přínosná. Práce obohatila mé

znalosti programovacího jazyka C#.NET, konfigurace webového serveru IIS a webových služeb. Za kolektivní přínos práce považuji, že praktická část je vhodný zdroj informací pro .NET komunitu, jelikož za celou dobu realizace této práce a praxe v této oblasti jsem nenalezl detailní a jednotný popis konfigurace webového serveru IIS jako hostitele WCF REST služeb, token autentizace společně s Role Based Authorization a vytvoření potřebných certifikátů pro zabezpečenou komunikaci mezi webovým serverem a klientem. Zvažuji také zveřejnění této problematiky na webu .NET komunity.

Použité zdroje

[1] BENEŠ, A.: *Bezpečnost v databázích [online]*. [cit. 2016-01-19]. Dostupné z: `http://www.obluda.cz/iprednasky/08_databaze.pdf`

[2] BOBROWSKI, S.: *Database as a Service [online]*. [cit. 2016-01-14]. Dostupné z: `https://dbaas.wordpress.com/2008/05/14/what-exactly-is-database-as-a-service/`

[3] Conollly, T.; Begg, C.; Holowczak, R.: *Databáze*. Brno: Computer Press, první vydání, 2009, ISBN 978-80-251-2328-7.

[4] DB-ENGINES: *DBMS popularity broken down by database model [online]*. 2015, [cit. 2016-01-16]. Dostupné z: `http://db-engines.com/en/ranking_categories`

[5] DB-ENGINES: *Method of calculating the scores of the DB-Engines Ranking [online]*. 2015, [cit. 2015-03-14]. Dostupné z: `http://db-engines.com/en/ranking_definition`

[6] DB-ENGINES: *Popularity of open source DBMS versus commercial DBMS [online]*. 2015, [cit. 2016-01-16]. Dostupné z: `http://db-engines.com/en/ranking_osvsc`

[7] Digitaltrike: *Technology Defined Part 1: Databases [online]*. [cit. 2016-01-01]. Dostupné z: `http://www.digitaltrike.com/technology-defined-part-1-databases/`

[8] HERNANDES, M. J.: *Návrh databází*. Praha: Vydavatelství GRADA, první vydání, 2006, ISBN 80-247-0900-7.

[9] Kroenke, D. M.; Auer, D. J.: *Databáze*. Brno: Computer Press, první vydání, 2015, ISBN 978-80-251-4352-0.

[10] LÖWY, J.: *Programming WCF services*. Sebastopol: O'Reilly, třetí vydání, 2010, ISBN 9780596805487.

[11] Mell, P.; Grance, T.: *The NIST Definition of Cloud Computing [online]*. NIST, [cit. 2016-01-12]. Dostupné z: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[12] MERUNKA, V.: *Objektový přístup v databázích*. Praha: CREDIT, první vydání, 2002, ISBN 80-213-0882-6.

[13] Michelotti, S.: *Resolve 404 in IIS Express for PUT and DELETE Verbs [online]*. [cit. 2016-02-11]. Dostupné z: http://stevemichelotti.com/resolve-404-in-iis-express-for-put-and-delete-verbs/

[14] MICROSOFT: *ADO.NET* [software] [cit. 2016-01-22]. Dostupné z: https://msdn.microsoft.com/en-us/library/e80y5yhx(v=vs.110).aspx

[15] MICROSOFT: *ASP.NET* [software] [cit. 2016-01-22]. Dostupné z: http://www.asp.net/

[16] MICROSOFT: *C sharp* [software] [cit. 2016-01-20]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/67ef8sbd.aspx?f=255&MSPPError=-2147217396

[17] MICROSOFT: *Certmgr.exe (Certificate Manager Tool)* [software] [cit. 2016-01-22]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/e78byta0(v=vs.110).aspx

[18] MICROSOFT: *IIS* [software] [cit. 2016-01-20]. Dostupné z: https://www.iis.net/

[19] MICROSOFT: *MakeCert* [software] [cit. 2016-01-22]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/aa386968(v=vs.85).aspx

[20] MICROSOFT: *Microsoft Azure* [software] [cit. 2016-01-20]. Dostupné z: https://azure.microsoft.com/cs-cz/

[21] MICROSOFT: *Microsoft .NET Framework 4.6* [software] [cit. 2016-01-22]. Dostupné z: https://www.microsoft.com/cs-cz/download/details.aspx?id=48130

[22] MICROSOFT: *Pvk2Pfx* [software] [cit. 2016-01-22]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/hardware/ff550672(v=vs.85).aspx

[23] MICROSOFT: *SQL Server 2014* [software] [cit. 2016-01-20]. Dostupné z: https://www.microsoft.com/cs-cz/server-cloud/products/sql-server/

[24] MICROSOFT: *Visual Studio* [software] [cit. 2016-01-22]. Dostupné z: https://www.visualstudio.com/

[25] MICROSOFT: *WCF* [software] [cit. 2016-01-22]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd456779(v=vs.110).aspx

[26] MICROSOFT: *Windows* [software] [cit. 2016-01-20]. Dostupné z: https://www.microsoft.com/library/errorpages/smarterror.aspx?aspxerrorpath=/en-us/windows/

[27] MICROSOFT: *WPF* [software] [cit. 2016-01-22]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/ms754130(v=vs.110).aspx

[28] MOTOROLA: *Motorola LS2208* [software] [cit. 2016-01-22]. Dostupné z: https://www.zebra.com/gb/en/products/scanners/general-purpose-scanners/handheld/ls2208.html

[29] MÜLLER, C.: *Ochrana soukromých dat [online].* [cit. 2016-01-20]. Dostupné z: http://www.chip.cz/soubory/dokumenty/c7306012b20beab7e79c4dae07608bce.pdf

[30] NOEHR, J.: *Bitbucket* [software] [cit. 2016-01-22]. Dostupné z: https://bitbucket.org/

[31] ORACLE: *Oracle Data Modeler* [software] [cit. 2016-01-22]. Dostupné z: http://www.oracle.com/technetwork/developer-tools/datamodeler/overview/index.html

[32] OTTE, L.: *Studijní opory k píednáškám a cvičení z předmětu Databázové systémy [online].* Technická univerzita Ostrava, 2012, [cit. 2015-12-30]. Dostupné z: http://homel.vsb.cz/~ott007/Predmety/databaze/Doprovodny_text_DB.pdf

[33] PETERKA, M.: *Seznamte se s BI [online].* DAQUAS, [cit. 2016-01-10]. Dostupné z: http://www.daquas.cz/Articles/379-seznamte-se-s-bi.aspx

[34] PLACHÝ, P.: *Trendy moderních databází [online].* Systemonline, 2014, [cit. 2015-12-29]. Dostupné z: http://www.systemonline.cz/business-intelligence/trendy-modernich-databazi.htm

[35] POKORNÝ, J.: *Databázová abeceda.* Brno: SCIENCE, první vydání, 1998, ISBN 80-86083-02-0.

[36] POKORNÝ, J.: *Databázové systémy 2.* Praha: Vydavatelství ČVUT, první vydání, 2007, ISBN 978-80-01-03797-3.

[37] POKORNÝ, J.; HALAŠKA, I.: *Databázové systémy.* Praha: Vydavatelství ČVUT, druhé vydání, 2003, ISBN 80-01-02789-9.

[38] POKORNÝ, J.; Valenta, M.: *Databázové systémy.* Praha: Vydavatelství ČVUT, první vydání, 2013, ISBN 978-80-01-05212-9.

[39] ROKOS, M.: *Výhody a nevýhody Cloud computingu oproti vlastní infrastruktuře [online].* [cit. 2016-01-13]. Dostupné z: http://praha.educanet.cz/uploads/uspechy

[40] SHARMAN, R.: *Data Models in DBMS [online].* 2015, [cit. 2016-01-02]. Dostupné z: http://dbmsnotes-ritu.blogspot.cz/2015/08/data-models-in-dbms.html

[41] STARTCOM: *StartCom Ltd. (Start Commercial Limited)* [software] [cit. 2016-01-22]. Dostupné z: https://www.startssl.com/policy.pdf

[42] SYSTEM, S.: *Enterprise Architect* [software] [cit. 2016-01-22]. Dostupné z: http://www.sparxsystems.com.au/products/ea/

[43] TEAM, T. I.: *Inkscape* [software] [cit. 2016-01-22]. Dostupné z: https://inkscape.org/en/

[44] VALENTA, M.: *Architektura SŘBD [online].* 2010, [cit. 2015-12-29]. Dostupné z: http://users.fit.cvut.cz/valenta/doku/lib/exe/fetch.php/bivs/dbs2_02_architektura_srbd.pdf

[45] ŽÁK, K.: *Historie relačních databází [online].* ROOT.CZ, 2001, [cit. 2015-12-29]. Dostupné z: http://www.root.cz/clanky/historie-relacnich-databazi/

Příloha **A**

Seznam použitých zkratek

.NET NET pochází z network

3DES Triple Data Encryption Standard

AES Advanced Encryption Standard

ANSI American National Standards Institute

ASŘP Automatizovaný systém řízení dat podniku

CAD Computer-aided design

CODASYL Conference on Data Systems Languages

DA Data administrator

DBA Database administrator

DBMS Database management system

DBS Database System

DBTG Database Task Group

DB Database

DCL Data Control Language

DDEX Data Designer Extesibility

DDL Data definition language

DES Data Encryption Standard

DML Data manipulation language

DMZ Demilitarized zone

DSDLC Database systems development life cycle

DaaS Database as a service

ETL Extract, transform and load

GUI Graphical User Interface

HZD Hromadné zpracování dat

IBM International Business Machines Corporation

IIS Internet Information Services

IMS Information Management System

IaaS Infrastructure as a service

Ingres Interactive Graphics and Retrieval System

MVC Microsoft Visual Studio

ODBMS Object DBMS

ODMG Object Data Management Group

ODM Objektový databázový model

OLAP Online Analytical Processing

OMG Object Management Group

OODBMS Systém řízení OODB

OODM Object oriented data model

OOSŘBD Objektově orientovaný systém řízení báze dat

OQL Object Query Language

ORDBMS Object-Relational DBMS

PaaS Platfrom as a service

RDBMS Relational DBMS

RDM Relační datový model

RSA Rivest, Shamir and Adleman

SDL Storage Definition Language

SaaS Software as a service

SŘBD Systém řízení báze dat

TCL Transaction Control language

VDL View Definition Language

WCF Windows Communication Foundation

Příloha B

Zdrojové kódy

B.1 Konfigurační soubor Web.config

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="log4net" type="log4net.Config.
        Log4NetConfigurationSectionHandler, log4net"/>
  </configSections>
  <system.web>
    <customErrors mode="Off"/>
    <compilation debug="true" targetFramework="4.6"/>
    <pages controlRenderingCompatibilityVersion="4.0"/>
  </system.web>
  <system.serviceModel>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
    <services>
      <service name="DanceSir.Backend.Service"
              behaviorConfiguration="ServiceBehaviour" >
        <endpoint address="v1"
                 binding="webHttpBinding"
        bindingConfiguration="webHttpTransportSecurity"
        behaviorConfiguration="webHttpBehavior"
        contract="DanceSir.Contracts.IService" />
      </service>
    </services>
    <bindings>
      <webHttpBinding>
        <binding name="webHttpTransportSecurity">
          <security mode="Transport" />
        </binding>
      </webHttpBinding>
    </bindings>
    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceBehaviour">
          <serviceDebug includeExceptionDetailInFaults="false"/>
```

```
34          <serviceAuthorization principalPermissionMode="Custom">
35            <authorizationPolicies>
36              <add policyType="DanceSir.Backend.ServiceCustomizations.
                    Authorization.AuthorizationPolicy, DanceSir.Backend"/>
37            </authorizationPolicies>
38          </serviceAuthorization>
39        </behavior>
40      </serviceBehaviors>
41      <endpointBehaviors>
42        <behavior name="webHttpBehavior">
43          <webHttp defaultBodyStyle="Bare" helpEnabled="true"
                defaultOutgoingResponseFormat="Json"
                automaticFormatSelectionEnabled="false"/>
44        </behavior>
45      </endpointBehaviors>
46    </behaviors>
47  </system.serviceModel>
48  <system.webServer>
49    <handlers>
50      <remove name="ExtensionlessUrl-Integrated-4.0"/>
51      <add name="ExtensionlessUrl-Integrated-4.0" path="*." verb="GET,HEAD,
            POST,DELETE,PUT" type="System.Web.Handlers.TransferRequestHandler"
            preCondition="integratedMode,runtimeVersionv4.0"/>
52    </handlers>
53  </system.webServer>
54  <log4net>
55    <appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
56      <file value="logs\\evsir.log"/>
57      <appendToFile value="true"/>
58      <maximumFileSize value="10MB"/>
59      <maxSizeRollBackups value="5"/>
60      <layout type="log4net.Layout.PatternLayout">
61        <conversionPattern value="%d [%5t] %-5p &lt;%-12X{User}&gt; - %m%n"/>
62      </layout>
63    </appender>
64    <root>
65      <!-- set log level for root logger which is derived by all other loggers
            -->
66      <level value="ALL"/>
67      <appender-ref ref="RollingFile"/>
68    </root>
69    <!-- overridden configurations for loggers -->
70    <logger name="DanceSir.Backend">
71      <level value="WARN"/>
72    </logger>
73  </log4net>
74 </configuration>
```

Kód B.1: Konfigurační soubor Web.config

B.2 Ukázka jednotkového testu - klient

```
/// <summary>
/// Attendance - GetAttendances - view entries count is zero
/// </summary>
[TestMethod]
public void TestGetAttendancesViewEntriesCountIsZero()
{
    // Arrange (NSubstitute)
    var attendanceModel = Substitute.For<IAttendanceModel>();
    var attendanceResponse = new AttendanceResponse { Attendances = new List<
        AttendanceResponse.Attendance>() };
    attendanceModel.GetAttendances().ReturnsForAnyArgs(attendanceResponse);
    var statusViewModel = new StatusViewModel();
    var attendanceViewModel = new AttendanceViewModel(statusViewModel,
        attendanceModel);
    // Act
    attendanceViewModel.GetAttendances();
    // Assert
    Assert.AreEqual(statusViewModel.Status, "Tabulka docházka je zatím prázdná
        . Můžete ji přidat tlačítkem + !");
}
```

Kód B.2: Ukázka jednotkového testu - klient

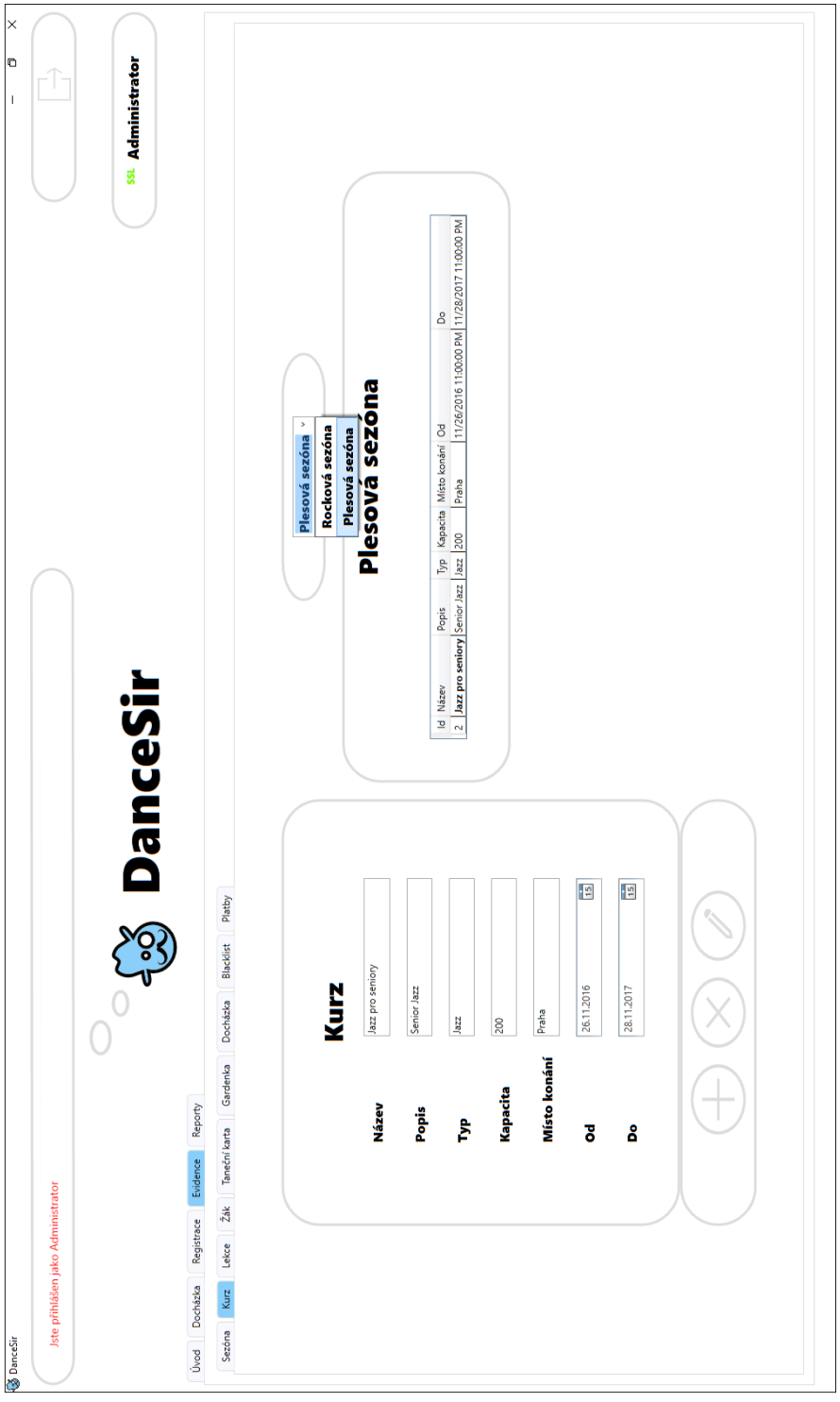
B.3 Ukázka jednotkového testu - server

```
/// <summary>
/// Blacklist - AddBlacklist - check PrincipalPermission role
/// Host user has not permission to add row of the Blacklist table
/// </summary>
[TestMethod]
[ExpectedException(typeof(SecurityException))]
public void TestAddBlacklistCheckPrincipalPermissionRoleHost()
{
    // Arrange (NSubstitute)
    var databaseProxy = Substitute.For<IDatabaseProxy>();
    CustomPrincipal customPrincipal = new CustomPrincipal
    {
        Identity = new CustomIdentity("Host", new[] { "AttendanceUsers" })
    };
    Thread.CurrentPrincipal = customPrincipal;
    var baseService = new Service(databaseProxy);
    // Act
    baseService.AddBlacklist(new BlacklistRequest());
    // Assert is handled by the SecurityException
}
```

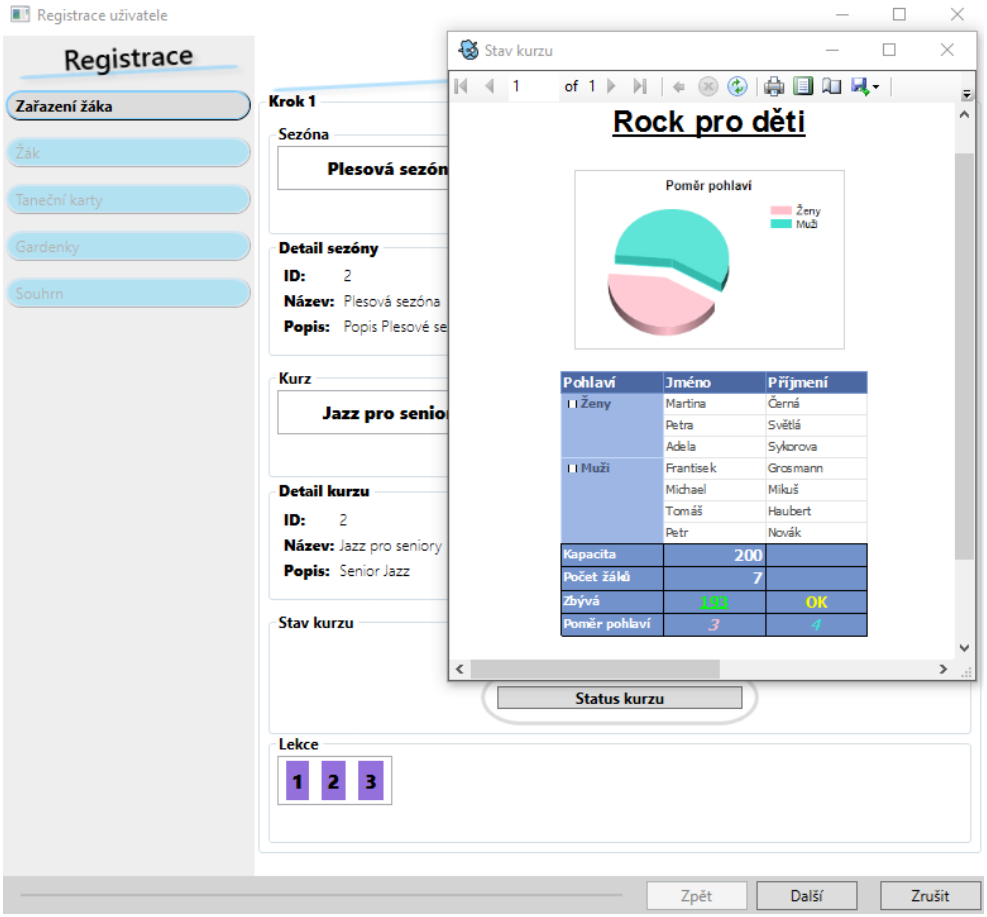
Kód B.3: Ukázka jednotkového testu - server

Příloha C

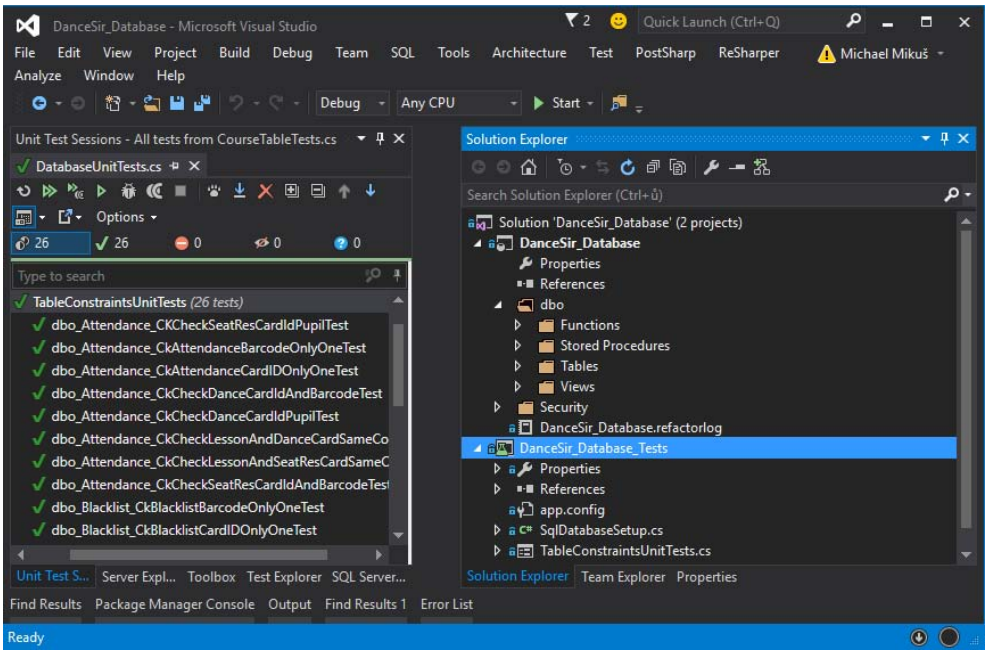
Obrazové přílohy



Obrázek C.1: Náhled klientské aplikace DanceSir



Obrázek C.2: Náhled na okno Registrace uživatele



Obrázek C.3: Unit test session a struktura databázového projektu ve Visual Studiu

Příloha D

Tabulky

D.1 Historický vývoj databázových systémů

D.2 Vybrané SŘBD - základní informace

Tabulka D.1: Historický vývoj databázových systémů, převzato z [3, st. 49]

Časový rámec	Vývoj	Komentář
Od 60. let	Souborově orientované systémy	Předchůdce databázových systémů, decentralizovaný přístup: každé oddělení uchovává a kontroluje vlastní data.
Polovina 60. let	Hierarchický a síťový model	Představují první generaci DBMS. Hlavním hierarchickým systémem je IMS od firmy IBM, hlavním síťovým systémem IDMS/R od firmy Computer Associated. Nedostatek nezávislostí dat a nutnost vyvíjet složité programy pro zpracování dat.
1970	Návrh relačního modelu	Zveřejněno průlomové pojednání E. F. Codda „A relational model of data for large shared data banks“ (Relační model dat pro velké sdílené databanky), které reagovalo na slabiny systémů první generace.
70. léta	Vyvíjejí se prototypy RDBMS	Během tohoto období se objevily dva hlavní prototypy: projekt INGRES na Kalifornské universitě v Berkeley (začal roku 1970) a projekt Systém R v IBM Research Laboratory v San José v Kalifornii (začal roku 1974), který vedl k vývoji SQL.
1976	Návrh ER modelu	Zveřejnění pojednání „The Entity Relationship model - Toward a unified view of data“ (Entitně relační model - k sjednocenému pohledu na data) od Chena. ER modelování se stalo podstatnou součástí metodologie návrhu databází.
1979	Objevují se komerční RDBMS	Objevily se komerční RDBMS jako Oracle, INGRES a DB2. Ty představují druhou generaci DBMS.
1987	Standard ISO SQL	ISO (americká organizace pro standardizaci) standardizovala SQL, existují také následovné verze standardu 1989, 1992 (SQL2), 1999 (SQL: 1999§) a 2003 (SQL:2003).
90. léta	Objevují se OODBMS a ORDBMS	V tomto období vznikl nejprve OODBMS a později ORDBMS (Oracle 8 s objektovými vlastnostmi byl uvolněn v roce 1997).
90. léta	Datové sklady	V tomto období začali také dodavatelé DBMS uvolňovat systémy skladování dat a poté produkty zaměřené na data-mining.
Polovina 90. let	Integrace web-data-báze	Objevily se první aplikace databází na Internetu. Dodavatelé DBMS a třetí strany rozpoznali důležitost Internetu a podpory integrace databází a webu.
1998	XML	W3C schválilo XML 1.0. Došlo k integraci XML s databázovými produkty a vyvíjí se nativní XML databáze.

Tabulka D.2: Vybrané SŘBD - základní informace

DBMS	**Oracle**	**MySQL**	**Microsoft SQL Server**	**PostgreSQL**	**MongoDB**
Developer	Oracle	Oracle	Microsoft	PostgreSQL GDG	MongoDB, Inc
Webové stránky	oracle.com	mysql.com	microsoft.com	postgresql.org	mongodb.org
Databázový model	Relational DBMS	Relational DBMS	Relational DBMS	Relational DBMS	Document store
První vydání	1980	1995	1989	1989	2009
Aktuální vydání	12 Release 1 (12.1.0.2), July 2014	5.7.10, December 2015	SQL Server 2014, April 2014	9.5, January 2016	3.2.0, December 2015
Licence	commercial	Open Source	commercial	Open Source	Open Source
Jazyk implementace	C and C++	C and C++	C++	C	C++
Serverový operační systém	AIX, HP-UX Linux, OS X Solaris, Windows z/OS	FreeBSD, Linux OS X, Solaris Windows	Windows	FreeBSD, HP-UX Linux, NetBSD OpenBSD, OS X Solaris, Unix Windows	Linux, OS X Solaris, Windows
Podporované programovací jazyky	C, C#, C++ Clojure, Cobol Eiffel, Erlang Fortran, Groovy Haskell, Java JavaScript, Lisp Objective C, OCaml Perl, PHP Python, R Ruby, Scala Tcl, Visual Basic	Ada, C, C# C++, D, Eiffel Erlang, Haskell Java, Objective-C OCaml, Perl PHP, Python Ruby, Scheme Tcl	.Net, Java PHP, Python Ruby, Visual Basic	.Net, C, C++ Java, Perl Python, Tcl	Actionscript, C C#, C++, Clojure ColdFusion, D Dart, Delphi Erlang, Go Groovy, Haskell Java, JavaScript Lisp, Lua MatLab, Perl PHP, PowerShell Prolog, Python R, Ruby Scala, Smalltalk
Transakční koncepty	ACID	ACID	ACID	ACID	no
SQL	yes	yes	yes	yes	no
APIs and other access methods	ODP.NET Oracle Call Interface (OCI) JDBC, ODBC	ADO.NET, JDBC, ODBC	OLE DB Tabular Data Stream (TDS) ADO.NET JDBC, ODBC	native C library streaming API for large objects ADO.NET JDBC, ODBC	proprietary protocol using JSON

Příloha **E**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `cert` ..... certifikáty databázové aplikace
  - `DanceSir_Client` ..... implementace klientské aplikace
    - `DanceSir_Client.sln` ..... Visual Studio solution
    - `DanceSir_v1.msi` ..... instalátor klientské aplikace
  - `DanceSir_Database` ..... SQL Server Database Project
    - `DanceSir_Database.sln` ..... Visual Studio solution
  - `DanceSir_Server` ..... implementace serverové aplikace
    - `DanceSir_Server.sln` ..... Visual Studio solution
  - `scripts` ..... SQL skripty pro datové úložiště
  - `thesis` ..... zdrojová forma práce
    - `BP_Mikuš_Michael_2016.tex` ..... text práce ve formátu LaTeX
    - `csn690.bst` ..... citační norma
    - `mybibliographyfile.bib` ..... použité zdroje
    - `figures` ..... obrázky použité v textu
- `doc` ..... dokumentace
- `text` ..... text práce
  - `BP_Mikuš_Michael_2016.pdf` ..... text práce ve formátu PDF