VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

VYTVOŘENÍ SQL DATABÁZE PRO PODPORU EVIDENCE HARDWARU

SQL DATABASE CREATION TO SUPPORT EVIDENCE OF HARDWARE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN KONEČNÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ KŘÍŽ, Ph.D.
SUPERVISOR

BRNO 2012

Vysoké učení technické v Brně
Fakulta podnikatelská

Akademický rok: 2011/2012
Ústav informatiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Konečný Martin

Manažerská informatika (6209R021)

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách, Studijním a zkušebním řádem VUT v Brně a Směrnicí děkana pro realizaci bakalářských a magisterských studijních programů zadává bakalářskou práci s názvem:

Vytvoření SQL databáze pro podporu evidence hardwaru

v anglickém jazyce:

SQL Database Creation to Support Evidence of Hardware

Pokyny pro vypracování:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrhy řešení, přínos návrhů řešení
Závěr
Seznam použité literatury
Přílohy

Seznam odborné literatury:

CONOLLY, T., BEGG, C., HOLOWCZAK, R. Mistrovství – Databáze : Profesionální pruvodce tvorbou efektivních databází. Brno : Computer Press, 2009. 584 s. ISBN 978-80-251-2328-7.
HOTEK, M. Microsoft SQL Server 2008: Krok za krokem. Brno: Computer Press, 2009. 488 s. ISBN 978-80-251-2466-6.
LACKO, L. Jak vyzrát na SQL Server 2008. Brno: Computer Press, 2009. 469 s. ISBN 978-80-251-2101-6.
OPEL, A. SQL bez předchozích znalostí. 1. vyd. Brno : Computer press, 2008. 239 s. ISBN 978-80-251-1707-1.

Vedoucí bakalářské práce: Ing. Jiří Kříž, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2011/2012.

L.S.

Ing. Jiří Kříž, Ph.D.	doc. RNDr. Anna Putnová, Ph.D., MBA
Ředitel ústavu	Děkan fakulty

V Brně, dne 24.05.2012

Abstrakt

Tato bakalářská práce se věnuje návrhu a implementaci databáze pro evidenci hardwaru ve firmě IBM Česká republika s.r.o. Databázi implementuji v prostředí MS SQL Server.

Abstract

This bachelor's thesis is dedicate to implement database for evidence of Hardwre in company IBM Czech republic s.r.o. Database is implement in MS SQL Server.

Klíčová slova

SQL, databáze, MS SQL Server, vývojový diagram, normalizace, systém řízení báze dat, hardware, software, proces

Key words

SQL, database, MS SQL Server, flow chart, normalization, database management system, hardware, software, process

Bibliografická citace:

KONEČNÝ, M. *Vytvoření SQL databáze pro podporu evidence hardwaru.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2011. 75 s. Vedoucí práce Ing. Jiří Kříž Ph.D.

Čestné prohlášení:

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenu jsou úplné, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona c. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne …….…… Konečný Martin

Poděkování:

Rád bych poděkoval Ing. Jiřímu Křížovi, Ph.D za vedení při vypracování bakalářské práce.

Dále bych velice rád poděkoval týmu IGA ve firmě IBM Česká republika, ne jen za poskytnutí podkladů pro vypracování bakalářské práce, ale i za obohacení mých znalostí.

Obsah

Úvod

Má bakalářská práce se bude zabývat teoretickým pozadím databází, analýzou problematiky evidence hardwaru ve firmě IBM a tou nejdůležitější části je samotná implementace databáze evidence hardwaru. Navrhnu a implementuji databázi evidence hardwaru, která bude odpovídat všem požadavkům, které analýza stanoví.

IBM v Brně zaměstnává přes 3000 zaměstnanců a každý z nich vlastní počítač a používá další společný hardware jako je např. tiskárna. Jak nejlépe evidovat hardware, informace o něm a hlavně o jeho vlastnících? V tomto počtu je velice nepraktické evidovat vlastníky papírovou formou, a proto v této technologické době vytvoříme databázi. Tato databáze také bude základem pro budoucí informační portál, kde se naprogramují vhodné statistické funkce, abychom získali větší přehled o hardwaru ve firmě. Pracuji v oddělení IGA CZ, které je správcem veškerého IBM hardwaru v ČR, a proto se snažíme tuto agendu, co nejvíce zjednodušit. Toto je jen jedna oblast za kterou je IGA oddělení odpovědné. Dalšími oblastmi jsou např. administrace interních serveru, síťová infrastruktura, lokální doména a to nejdůležitější IT podpora všech zaměstnanců IBM GS Delivery Center Brno . Pro tvorbu databáze použiji strukturovaného dotazovacího jazyka SQL. Jako nejvhodnější platformu pro tvorbu databází jsem z možných vybral MS SQL Server 2008.

Vymezení problémů a cíle práce

Vymezení problémů

Největší problém není úplná absence informačního systému pro evidenci hardwaru, ale jeho nedokonalost pro použití v našem oddělení. Tato nedokonalost je způsobena jeho roztroušeností do jednotlivých databází. Pro evidenci hardwaru používáme čtyři databáze. Nejdůležitější z nich je webový portál eAMT. Tento systém vznikl za účelem evidovat vlastníky konkrétního hardwaru hlavně jako důkazní materiál. Zaměstnanec je hmotně odpovědný za veškerý hardware, který má v této databázi na sebe přidělen. Z tohoto účelu za jakým systém vznikl je jasné, že se neukládají důležité informace, které jako administrátoři tohoto hardwaru potřebujeme. Např. kdo hardware vydal, datum jeho vydání atd. Dalším negativem je, že tento systém funguje ve všech pobočkách IBM po celém světě. Kvůli tomu je jeho výstup na intranetových stránkách firmy někdy pomalý a někdy portál nefunguje vůbec. Druhou důležitou databází je databáze v programu Lotus Notes. V této databázi se již uchovávají všechny důležité informace o hardwaru, ale tato databáze nepokrývá celou agendu evidence hardwaru. Nezahrnuje opravy hardwaru a trpí určitými problémy, které přiblížím v analýze současné situace. Z těchto důvodů jsme se rozhodli, že vytvořím SQL databázi evidence hardwaru přímo na míru pro naše oddělení.

Cíle práce

Z vymezení problému vyplývá, že cílem této práce je návrh a implementace SQL databáze pro podporu evidence hardwaru. Proces tvorby databáze začíná zjištěním požadavků na databázi. Dalším krokem je tvorba procesů spojených s evidencí hardwaru pomocí vývojových diagramů např. výdej hardwaru. Dle této analýzy požadavků a procesů zpracuji návrh databáze. Pomoci jazyka SQL implementuji databázi na základě konceptuálního a logického návrhu, který vytvořím.

1 Teoretická východiska práce

1.1 Datové modelování

Cílem datového modelu je co nejpřesněji uložit data do výpočetní techniky a hlavně co nejpřesněji zachytit skutečnost reálného světa. Snaží se být co nejpřehlednější, odstraňuje duplicitu a také definuje relace mezi daty.

1.1.1 Základní pojmy datového modelování

Informace

„*Na informaci lze nahlížet z různých hledisek. Informaci můžeme chápat jako zprávu, vjem, který splňuje tři požadavky. Prvním je syntaktická relevance. Subjekt, který zprávu přijímá, musí byt schopen ji detekovat a rozumět ji. Druhým požadavkem je sémantická relevance. Subjekt musí vědět, co zpráva znamená, co vypovídá o něm a jeho okolí. Třetím požadavkem je pragmatická relevance. Zpráva musí mít pro přijímající subjekt nějaký význam."* (Koch, Neuwirth, 2008, s. 4).

Data

„V praxi je datům běžně přisuzován význam zpráv." (Koch, Neuwirth, 2008, s. 5). Data jsou jakousi zašifrovanou verzí informací, tedy, aby se z dat stala informace, musíme je dekódovat, ať už nějakým programem nebo lidským mozkem.

Znalosti

Znalosti můžeme chápat jako informace o využití a způsobu využití jiných informací či dat. Jsou podobné metadatum - data o datech. Lidský mozek využívá znalosti při reakcích na informace.

Entita

*„Jedním z nejdůležitějších pojmů je Entita**.** Entita je prvek reálného světa (např. televize, počítač, člověk nebo jakýkoliv předmět atd.), který lze popsat určitými charakteristikami (vlastnostmi). Tyto charakteristiky se v databázovém světě nazývají atributy (např. název, objem, označení, hmotnost, hodnota)."* (Skřivan, 2000)

1.1.2 Datové modely

K vystižení reality nám nestačí pouze atributy jednotlivých entit. Musíme je nějakým způsobem propojit, a to řeší problematika datových modelů.

Známe pět typů datových modelů:

- Lineární
- Hierarchický
- Síťový
- Relační
- Objektový

Lineární datový model

„V lineárních datových modelech není žádná vazba mezi jednotlivými skupinami objektů - tabulkami" (Koch, Neuwirth, 2008, s. 20).

Klasickým příkladem, se kterým se setkáme i později, je kartotéka u lékaře. Jednotlivé karty mezi sebou nemají žádný vztah, tedy když pomineme vztah následující a předcházející karty v zásuvce.

Hierarchický datový model

Jak již z názvu vyplývá, jedná se o model, který je tvořen rodičovskými a podřízenými segmenty. Rodičovské segmenty mají vazbu s dceřinými segmenty, tato vazba je vedena tzv. pointery. Na podřízené segmenty se dá dostat pouze přes rodičovské segmenty. Velkými výhodami tohoto modelu jsou jeho přehlednost a rychlost vyhledávání v něm. Hlavní nevýhodou je doba nutná na reorganizaci jednotlivých segmentů.

Síťový datový model

Síťový model je podobný hierarchickému. Využívá techniku pointerů, nicméně tady pointery neukazují pouze z rodičovského segmentu do podřízeného segmentu, ale můžou ukazovat mezi segmenty v různých směrech. Na rozdíl od hierarchického modelu má síťový datový model výhodu libovolného propojení mezi jednotlivými segmenty.

Objektový datový model

„Objektové datové modely jsou vystavěny na základním prvku - objektu (odpovídá přibližně pojmu věta), kde tento objekt má kromě svých atributů i definované metody, které určují chování objektu." (Koch, Neuwirth, 2008, s. 23).

Relační datový model

Jedná se o nejpoužívanější datový model a i já využiji tento typ. Vzniká spojením lineárních modelu pomocí relačního klíče. Toto spojení není trvalé, ale zaniká, když ukončíme práci s modelem.

1.1.3 Integrita relačního modelu

Modelování dat z reálného světa přináší omezení teoretického modelu. Integritu modelu chápeme jako stav modelu, když data v modelu odpovídají reálnému světu.

Tato omezení dělíme na
- Integritní omezení pro entity (relace)
- Integritní omezení pro vztahy mezi entitami (relační vazby)

Integritní omezení pro entity

Kandidátní klíč

Je množina atributu, která má následující dvě vlastnosti. První z vlastností je jeho jednoznačnost, tedy pro žádné dva řádky v tabulce nemůže nastat situace, že by hodnota kandidátního klíče byla stejná. A druhá vlastnost relace je ta, že se v ní může nacházet více kandidátních klíčů.

Primární klíč

Je podmnožinou kandidátního. V jeho případě nesmí být u žádného z atributu hodnota NULL. Každý řádek relace musí být identifikovatelný dle primárního klíče. Jeden z kandidátních klíčů je vybrán jako primární.

Cizí klíč

Cizí klíč vytváří vazbu mezi tabulkami. Vytvoří spojení jednoho nebo více sloupců tabulky se sloupcem nebo sloupci "cizí" tabulky. Tento sloupec v "cizí" tabulce je vždy primárním klíčem. Tímto způsobem cizí klíč vytvoří relace mezi jednotlivými tabulka-

mi a rozvíjí informace, které byly uvedeny v "cizí" tabulce. Je buď plně zadán, nebo nezadán.

Integritní omezení pro vztahy

Integritní omezení pro vztahy omezuje kardinalitu, tedy max. a min. počet entit v určitém vztahu. Máme různé poměry vztahů: 1:1, 1:N, N:1, N:M

Vazba 1:1

Vždy jedna n-tice relace odpovídá jedné n-tici jiné relace. Např. vztah mezi člověkem a občanským průkazem. Jeden člověk může mít jen jeden občanský průkaz.

Vazba 1:N

Vždy jedna n-tice relace odpovídá jedné nebo více n-ticím jiné relace. Např. vztah bankovního účtů s jeho majitelem. Majitel může vlastnit více bankovních účtu a bankovní účet má jen jednoho vlastníka.

Vazba N:M

Několika n-ticím relace odpovídá jedna nebo více n-tic jiné relace. Např. vztah mezi majitelem a autem. Jedno auto může vlastnit jeden nebo více majitelů a zároveň jeden majitel může vlastnit více aut.

1.1.4 Normalizace

Normalizace je činnost, která upravuje strukturu entit a relací na strukturu fyzického uspořádání tabulek a relací v databázi. Abychom mohli říct, že je databáze normalizována musí splňovat určité podmínky či pravidla. Cílem normalizace je přehlednější, rozšiřitelnější a výkonnější databáze. Aby databáze splňovala nejvyšší úroveň, musí splňovat všechny nižší úrovně.

1. normální forma

Všechny atributy věty nesmí být složené, či vícehodnotové, tedy musí být atomické. Např. adresa je údajem složeným z čísla popisného, města a ulice. Databáze splňuje první normální formu v případě, že neobsahuje žádny takovýto atribut.

2. normální forma

Databáze se nachází v druhé normální formě v případě, že je v první normální formě a všechny atributy jsou plně závislé na celém kandidátním klíči (primárním).

3. normální forma

Relace je v 3. normální formě, pokud je v druhé normální formě a neobsahuje žádné tranzitivní závislosti, tedy závislosti mezi dvěma atributy a klíčem, přičemž jeden z atributů je závislý na druhém atributu, a až ten je závislý na kandidátním klíči (primárním klíči).

Boyce - Coddova normální forma

Je variací třetí normální formy. Má stejná pravidla, ale musí platit mezi hodnotami uvnitř složeného primárního klíče. *„Relace je v Boyce - Coddově normální formě, pokud mezi kandidátními klíči není žádná funkční závislost to za těchto podmínek :*

- *relace musí mít dva nebo více kandidátních klíčů.*
- *nejméně dva z kandidátních klíčů musí být složené*
- *kandidátní klíče se v některých atributech musí překrývat"* (Koch, Neuwirth, 2008, s. 61).

1.1.5 Databáze

Dle mého názoru je nejlepší definice databáze tato: *„Databáze je kolekce vzájemně souvisejících datových položek, které jsou spravovány jako jediná jednotka."* (Oppel, 2008, s. 9). Ať už jsou data představována jednotlivými slovy na papíře nebo uložené na paměťovém mediu.

Historie databází

Již v časech starého Egypta lidé cítili potřebu uchovat informace v databázích. Nebyly to databáze, jak je známe dnes, ale používali metodu "kreslení" hieroglyfů na papyrus. O mnoho let později lidé vynalezli papír, čímž se v uchovávání informací posunuli o hodně dopředu. Papír je jeden z nejpoužívanějších způsobů uchování informací. V dnešní době je dobrý příklad kartotéka u lékaře. V každé kartě naleznete údaje o pacientovi, nemocích atd. Knihy, časopisy a noviny jsou také příkladem databáze. Všechny

tyto databáze jsou uchované na papíře, a tedy sdílí stejný problém. Velice těžko se z nich analyzují, tzv. „dolují" data a také je k nim složitý přístup. Databáze papírového typu jsou vždy na jednom místě, tedy může k nim přistupovat pouze jeden člověk. Kopírování tento problém nevyřeší, jen trochu zvýší rychlost, s jakou se informace k člověku dostane. Všechny tyto problémy vyřešil příchod informačních technologií. V roce 1974 vznikla první forma jazyka pro tvorbu databází, která se jmenovala Sequel. Standardizační skupina ANSI ho v roce 1986 přijala jako standard jazyk, který používala firma IBM. V roce 1989 skupina rozšířila standard o možnost integritního omezení.

Systém řízení báze dat

Jde o software, který dodává výrobce. Je spousta produktů od různých firem např. Microsoft Access, Microsoft SQL Server, Oracle database, DB2, PostgreSQL, MySQL, INGRES.

„SŘBD poskytují všechny základní služby vyžadované k uspořádání a údržbě databáze, a to včetně následujících

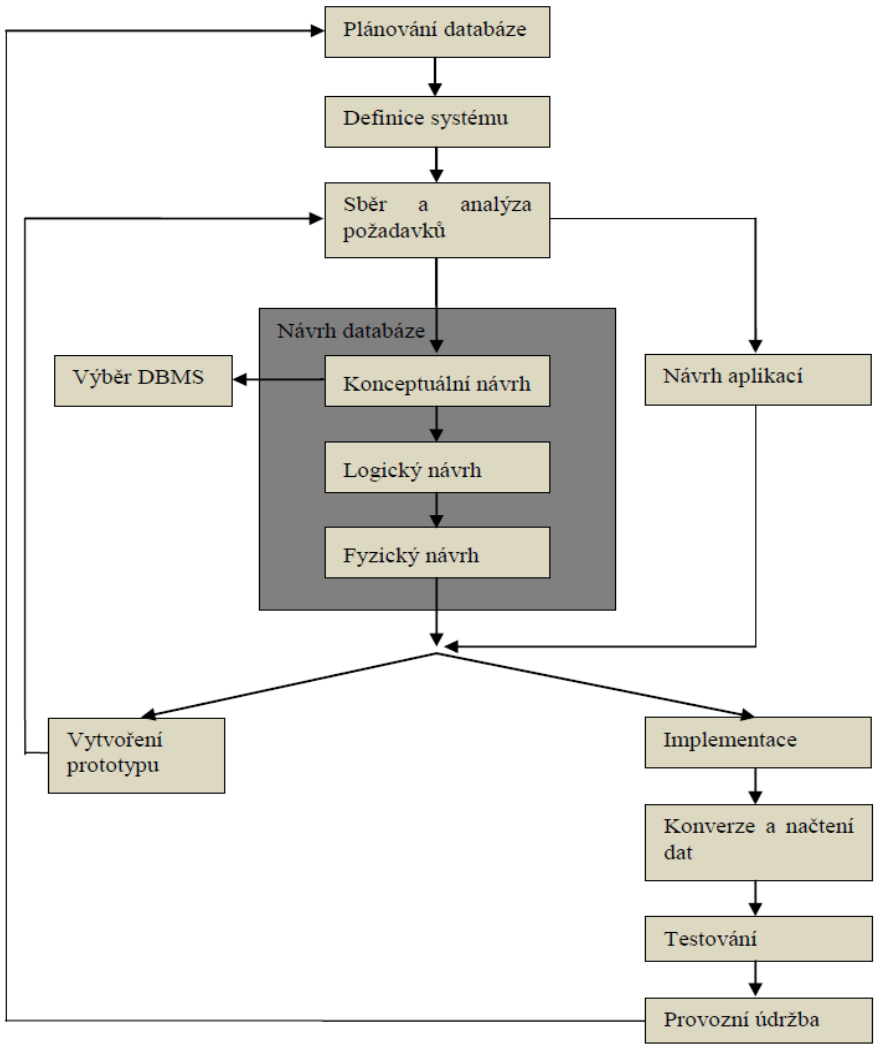
- *Přesouvání dat z a do fyzických datových souborů podle potřeby*
- *Řízení souběžného přístupu k datům více uživateli včetně opatření k zamezí kolizí souběžných aktualizací*
- *Řízení transakcí tak, aby se všechny změny databáze způsobené transakcí provedly zcela nebo vůbec. Jinak řečeno, pokud transakce proběhne úspěšně, zapíší se všechny změny databáze provedené transakcí, pokud se provedení transakce nezdaří, žádné provedené změny se do databáze nezapíší. Všimněte si, že některé relační SŘBD transakce nepodporují*

- *Podpory dotazovacího jazyka, což je systém příkazů, s jejichž pomocí uživatel databáze načítá data z databáze. SQL je primárním dotazovacím jazykem používaným relačním SŘBD a hlavním tématem této knihy.*
- *Opatření pro zálohování databáze a obnovení databáze po selhání*
- *Bezpečnostních mechanizmů k zamezení neoprávněného přístupu a změně dat."* (Oppel, 2008, s. 10-11)

1.1.6 Životní cyklus vývoje databázového systému

Obrázek 1 zachycuje všechny činnosti, kterými musíme projít při tvorbě databáze. Nejdůležitější částí je návrh databáze, který se skládá ze tří kroků: konceptuální, logický a fyzický návrh. V konceptuálním návrhu se vytváří ERD diagram, který je základ pro tvorbu databáze. V logickém návrhu ERD diagram předěláme, aby vyhovoval relačním databázím. Odstraníme vazby N:M, specifikujeme cizí klíče atd. Posledním krokem je fyzický návrh, jehož výsledkem je kód SQL, který implementuje databázi.

Tímto jsme dokončili návrh databáze, ale k provozu databáze je toho potřeba více. Musí se do ní uložit všechna data a musí se otestovat, jestli opravdu funguje, tak jak má.



Obr. 1: Fáze životního cyklu vývoje databázových systémů (Zdroj: Doseděl, 2004, s. 110)

1.2 Funkční modelování

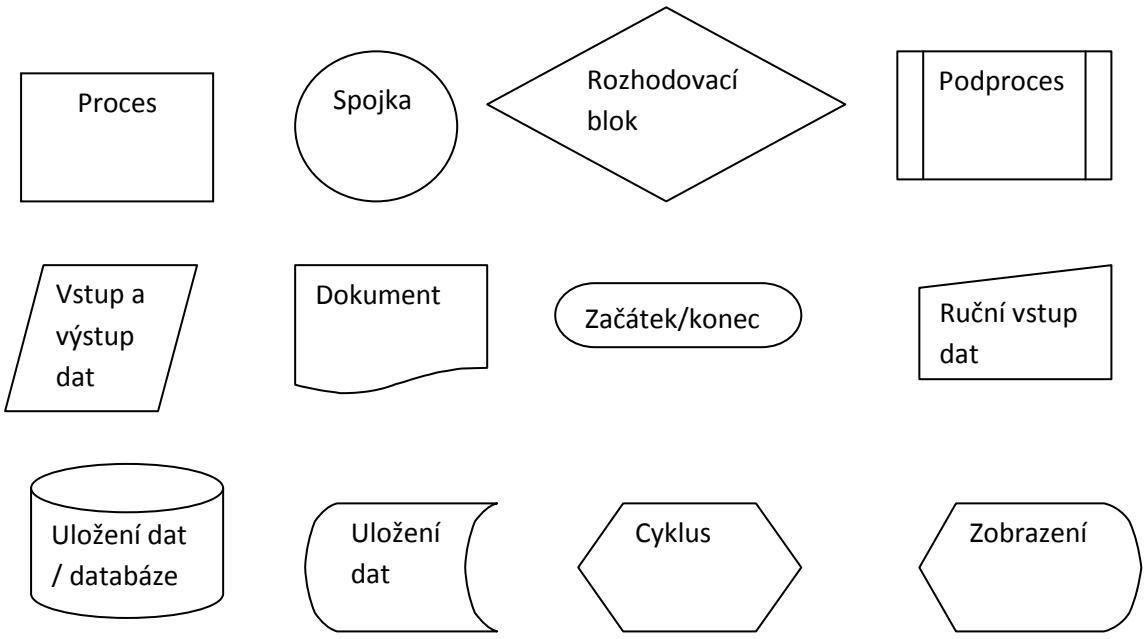
Funkční modelování analyzuje a algoritmizuje procesy a činnosti, které probíhají v informačním systému. Pro naše účely nám stačí zmínit ER a vývojový diagram.

1.2.1 ER diagram

ER diagram je nejdůležitějším diagramem v tvorbě databází. Z tohoto diagramu vychází kód SQL, který implementuje danou databázi. Znázorňuje jednotlivé vazby mezi entitami, včetně typů těchto vazeb. Způsobů zápisu je mnoho, ale každý z nich zachycuje definované entity a vztahy mezi nimi.

1.2.2 Vývojový diagram

Je to jeden z nejpoužívanějších diagramů. Jeho hlavní výhodou je, že velice jednoduše zachycuje větvení procesu díky rozhodovacímu bloku. Také je velmi oblíbený díky velmi rozmanitým značkám.



Obr. 2: Značky používané ve vývojovém diagramu (Zdroj: Koch, Neuwirth, 2008, s. 61)

1.3 Jazyk SQL

Jazyk SQL se prosadil jako univerzální jazyk pro relační databáze. Je implementován skoro ve všech SŘBD. Postupem času prošel určitými změnami a standardizací. Vyznačuje se vysokou přenositelností hlavně kvůli standardizaci. SQL je zkratka anglického názvu Structered Query Langugage, tedy strukturovaný dotazovací jazyk.

1.3.1 Datové typy jazyka SQL

Největší rozdíl mezi tabulkovým kalkulátorem a databází je právě v datových typech. V tabulkovém kalkulátoru můžu v jednom sloupci míchat jednotlivé datové typy, jak se mi zlíbí. Toto v databázi nelze. Pro každý sloupec/atribut je jasně definován typ a maximální délka. Toto je základní prvek omezení databáze. Maximální délka delší než je ve skutečnosti potřeba, může v databázi o milionech řádků nadělat velké problémy, třeba i z jinak fungující databáze udělat nefungující. Datové typy můžeme rozdělit na několik kategorií.

1.3.2 Základní dotazy jazyka SQL

Zde uvedu nejdůležitější dotazy jazyka SQL. Dotazy se děli do několika kategorií: Jazyk DDL (Data Definition Language), Jazyk DQL (Data Query Language), Jazyk DML (Data Manipulation Language), Jazyk DCL (Data Control Language). Každá z těchto kategorií se zabývá určitými typy dotazů, které už vyplývají z názvu kategorie.

1.3.3 Dotazy jazyka DDL

Dotazy tohoto typu definují databázové objekty, ale nepovolují jakoukoliv manipulaci s daty uloženými v databázi. Patří sem příkazy obsahující klíčová slova CREATE, ALTER, DROP

Dotaz Create Database

Definice databáze se odlišuje podle dodavatele softwaru. Obecná syntaxe definuje databázi takto:

CREATE DATABASE název_databáze [specifické požadavky dodavatele];

Dotaz Create scheme

Seskupuje databázové objekty (tabulky) za účelem snadnější správy databáze. Dodavatelé mají jiné databázové architektury, a právě proto se tento příkaz implementuje různými způsoby.

CREATE SCHEME název_schematu AUTORIZATION [nový_majitel_schématu];

Dotaz Create Table

Jedná se o nejdůležitější příkaz jazyka SQL. Než uložíme data do databáze, vždy musíme nejprve vytvořit tabulku, do které budeme data ukládat. Syntaxe příkazu:

CREATE TABLE název_tabulky (

<definice sloupce>

[,<definice sloupce> …]

[,<omezení sloupce> …]);

Dotaz Create Index

Indexy zvyšují výkon dotazů na data. V databázi musíme najít optimální počet indexu, protože údržba indexu zatěžuje hostitelský počítač. Při každé aktualizaci, vkládání a odstraňování dat se musí indexy znova přepočítat.

CREATE [Unique] INDEX název_indexu ON název_tabulky

(název_sloupce [ASC | DESC] [, název_sloupce [ASC | DESC] …]);

Dotaz Create View

Pohled je vlastně uložený SQL dotaz. Vytvoří fiktivní tabulku z již existujících tabulek a na tuto fiktivní tabulku se můžeme odkazovat v jiných příkazech. Někde se dočteme o tzv. virtuální tabulce.

CREATE [OR REPLACE] VIEW název_pohledu AS dotaz_sql;

Dotaz Alter Table

Tento příkaz upravuje již vytvořenou tabulku. Může upravovat sloupce i omezení. Někteří programátoři tento příkaz využívají k definici omezení, protože mají radši přehlednější příkaz create table.

Dotaz Drop

Tímto příkazem odstraňujeme libovolný objekt (tabulku, pohled, index atd.)

DROP <typ_objektu> název_objektu [<možnosti odstranění>]

1.3.4 Dotazy jazyka DQL

Obsahuje pouze jediný příkaz, a tím je SELECT. Je to nejpoužívanější příkaz jazyka SQL. Tímto příkazem vybíráme data, která chceme zobrazit a určujeme, jak je chceme zobrazit. Pomocí tohoto příkazu můžeme propojovat tabulky. Výsledkem příkazu Select je sada výsledků. Obecnou formu příkazu psát nebudu, protože obsahuje spoustu volitelných parametrů, funkcí atd.

Výsledky můžeme řadit pomocí klauzule ORDER BY. Pomocí klauzule WHERE můžeme výsledky filtrovat. V této klauzuli velice často využijeme matematické operátory.

Nejdůležitější funkcí příkazu SELECT je spojování více tabulek, tedy kombinování sloupců různých tabulek do výsledku jediného dotazu. Můžeme využít vice možností, jak tohoto cíle dosáhnout. Prvním je klauzule WHERE, pomocí které vyloučíme výsledky, které nám nevyhovují. Další je klauzule JOIN, která je dle mého názoru lepší variantou. Má kratší syntax a také je v některých případech rychlejší. Rozlišujeme také více druhů klauzule JOIN např. Inner, outer, atd. Vývoj klauzule JOIN je reakcí dodavatelů na zvyšující se požadavky zákazníků.

1.3.5 Dotazy jazyka DML

Tato část SQL umožňuje přidávání, úpravu a mazání záznamů z tabulek databáze. Obsahuje tři základní příkazy DELETE, UPDATE a INSERT. Těmito příkazy můžeme manipulovat pouze s údaji v jedné tabulce.

Dotaz Insert

INSERT INTO nazev_tabulky

(<seznam_sloupců>)

VALUES (<hodnota>, <hodnota>, … n);

Dotaz Delete

DELETE FROM název_tabulky_nebo_pohledu

[WHERE Klauzule]

Dotaz Update

UPDATE název_tabulky_nebo_pohledu

SET název_sloupce = výraz

[, název_sloupce = výraz …]

[WHERE Klauzule]

Klauzule where je v obou případech, protože musíme např. specifikovat konkrétní řádek v tabulce.

1.3.6 Uložené procedury a funkce

Procedury a funkce mají jedno společné, a tím je využití proměnných. V jazyku SQL rozlišujeme dva druhy proměnných: místní a globální. Globální proměnné nemůžeme deklarovat ani měnit. Místní proměnné můžeme vytvořit a ukládat do nich. Proměnnou vytvoříme pomocí klauzule DECLARE, do které napíšeme název a datový typ proměnné. Před místní proměnnou se uvádí jeden znak "@" a před globální se uvádí dva. Příklad deklarace proměnné:

```
DECLARE     @datum date,
            @pomocna int
```

Procedura je jakousi dávkou příkazu T-SQL , která je označena názvem a uložena v databázi. Funkce umožňují provádět výpočty a vracejí výsledek aplikaci. Procedura na rozdíl od funkce manipuluje s tabulkami. Může do nich ukládat a vypisovat data, měnit samotné tabulky atd. Toto ve funkci nelze.

Uložené procedury umožňují použít konstrukci řízení toku, kterými jsou: RETURN, GOTO, IF ELSE**,** BEGIN END**,** WHILE, WAITFOR, BREAK/CONTINUE

Konstrukce IF, ELSE podmínečně povoluje provedení části kódu. Pokud je podmínka splněna, kód se provede. Pokud splněna není a je uveden parametr ELSE, provede se část kódu, která je až za klíčovým slovem ELSE.

Příkaz WHILE spouští kód tak dlouho dokud je podmínka pravdivá. Se smyčkou WHILE souvisí samozřejmě i BREAK A CONTINUE. Jak i v jiných programovacích jazycích, tak i tady příkaz BREAK přeruší provádění kódu uvnitř smyčky WHILE a přeskočí až za smyčku. Příkaz CONTINUE zapříčiní pokračovaní kódu uvnitř smyčky.

WAITFOR pozastavuje provádění kódu. Může čekat na určitý čas nebo má nastavenou dobu prodlevy.

Příkaz RETURN ukončuje provádění procedury. Všechny příkazy za příkazem RETURN se nespustí.

Zjednodušená obecná forma deklarace procedury:

CREATE PROCEDURE název_procedury

[@nazev_parametru datový_typ,

@nazev_parametru2 datový_typ, ... n]

AS

<příkazy SQL>

Spuštění uložené procedury:

EXEC nazev_procedury @parametr1=<hodnota1>, @parametr2=<hodnota2>, ... n

nebo

EXEC nazev_procedury <hodnota1>, <hodnota2>, ... n

Obecnou formu funkce zde nebudu psát, protože existuje více druhů a jejich deklarace je poněkud rozsáhlá.

1.4 SWOT Analýza

SWOT je zkratka čtyř anglických slov – strengths (síla) , weaknesses (slabost), opportunities (příležitosti), threats (hrozby). SWOT analýza se snaží zjistit silné, slabé stránky, příležitosti a hrozby projektu, firmy, části firmy, podnikatelského záměru atd. Abychom zjistili tyto vlastnosti, používáme další analýzy – vnějšího okolí, vnitřního okolí. Interní analýza využívá metod 7S. Externí analýza využívá Porterův model 5 hybných sil k analýze konkurence a SLEPT analýzu ke zkoumání obecného okolí firmy. Z těchto dílčích analýz můžeme jednoduše vyvodit SWOT analýzu.

2 Analýza problémů a současné situace

2.1 Analýza současné situace

V této analýze představím firmu IBM a provedu SWOT analýzu jedné její pobočky, konkrétně GS Delivery Center v Brně. Představím databáze, které se používají pro evidenci hardwaru. Hlavně se budu věnovat rodině Lotus Notes, protože je to nejdůležitější aplikace, kterou používáme nejen pro evidenci hardwaru. Dále vysvětlím procesy, které jsou spjaté s evidencí hardwaru např. výdej počítačů.

2.1.1 Historie firmy a její vývoj do dnešní podoby

Společnost byla založena v roce 1911 jako Computing Tabulating Recording Corporation (CTR) a ze začátku vyráběla první kalkulačky, hodiny na počítání pracovní doby, stroje na řezání masa a sýru atd. Společnost vznikla spojením tří firem: the Tabulating Machine company, The International Time Recording Company a Computing Scale Corporation. Zaměstnávali okolo 1300 lidí a měli pobočky po celých USA. Tržby firmy byly $ 9 milionů a díky tomu se začala rozšiřovat i do Evropy. V roce 1924 společnost přijala nové jméno International Business Machines (IBM). Během 2. Světové války IBM vyráběla malé zbraně.

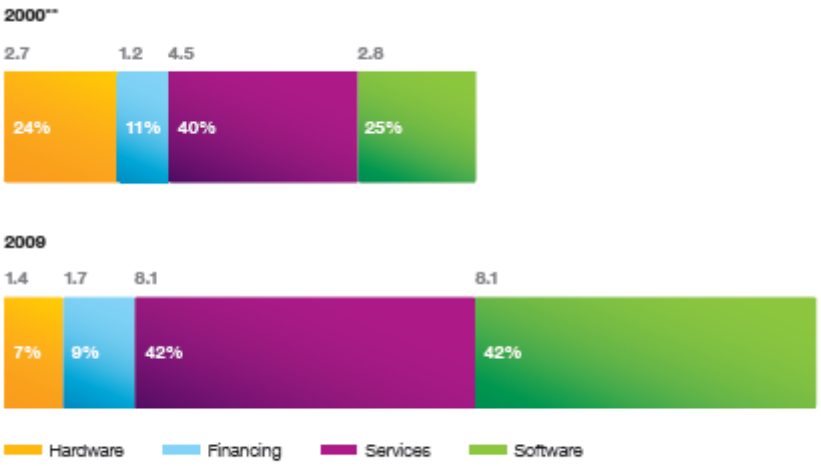
V roce 1956 Arthur L. Samuel naprogramoval první IBM 704, které dokázalo hrát dámu a „učit se“ z vlastních zkušeností. Šlo tedy o první samoučící program v historii. Demonstroval využití inteligence vytvořené člověkem.

V roce 1957 IBM vynalezlo FORTRAN (FORmula TRANslation) vědecký programovací jazyk. IBM vynalezlo také SABRE - rezervovací systém pro letecké společnosti. V roce 1963 společnost IBM pomohla NASA, najít orbitální let astronautu Mercury. Rok na to společnost přesunula ústředí firmy do Armonk, NY, kde zůstalo dodnes. V dalších letech IBM asistovalo NASA při prozkoumávání vesmíru.

V roce 1964 IBM představilo svůj první počítačový systém IBM Systém/360. První osobní počítač byl představen v roce 1981 a okamžitě se stal standardem na trhu. V roce 1991 IBM prodalo Lexmark, tedy divizi zabývající se tiskárnami. V naší pobočce máme ještě i staré IBM tiskárny, ale většina jich je již od firmy Lexmark. V roce 2005 IBM prodalo divizi osobních počítačů firmě Lenovo. Dnešní Lenovo vyrábí značky vytvoře-

né IBM jako thinkcenter nebo thinkpad. Tímto IBM dalo najevo, že její strategický plán se odklonil od výroby hardwaru.

Dnešní nejdůležitější činností je outsourcování služeb v IT. Poskytování těchto služeb vytváří větší přidanou hodnotu pro zákazníka tedy i pro IBM. Druhou nejdůležitější částí, kterou se IBM zabývá, je vývoj software a následný prodej licencí. Vlajkovou lodí v poli softwaru je určitě groupware Lotus Notes. Tento groupware využívají všichni zaměstnanci IBM a samozřejmě spousty firem po celém světě.



Obr. 3: Velikost a rozdělení zisku v jednotlivých letech podle segmentu trhu (Zdroj: Complete 2009 IBM Annual report, 2009)

Na tomto obrázku vidíme, jak se změnil poměr zisku před zdaněním z různých oborů.

Dnešním CEO společnosti je Virginia M (Ginni) Rometty, Chairman of the board je bývalý CEO Samuel J. Palmisano a generálním ředitelem firmy IBM Česká republika s.r.o se stal v roce 2010 Vladimír Šlezingr. Má další práce se bude zabývat pouze pobočkou IBM GS Delivery Center Brno.

2.1.2 SWOT analýza firmy

SWOT analýzu jsem vypracoval na základě těchto dílčích analýz: Porteruv model pěti sil, SLEPT a analýza interních faktorů.

SWOT analýza má za účel vyhledat silné, slabé stránky, příležitosti a hrozby pro firmu. Tyto vlastnosti jsme schopni určit až po provedení výše uvedených analýz. Tato analýza je provedena na pobočce IBM GS Delivery Center v Brně (IDC).

Tab. 1: SWOT analýza IBM GS Delivery center Brno (Zdroj: Konečný, 2011, 9 s.)

Silné stránky:	**Slabé stránky:**
- dobrý produkt	- vysoké nároky na reorganizaci a strategii
- transparentnost	- těžká nahraditelnost některých dodavatelů IT
- pevný vztah se stavajícími zákazníky	- vysoké náklady na některé dodavatele
- dobrý marketing	- špatná synchronizace personální politiky
- flexibilita	- vysoká fluktuace kvalifikovaných zaměstnanců
- adekvátní náklady	- tlak na snižování nákladů vs. kvalita
-inovace a výzkum v praxi	
- know-how	
- vzdělávání zaměstnanců – elearning, školení	
- automatizace (Pokud produkuje IDC substituty samo, jsou pod kontrolou a dochází ke snižování nákladů)	
Příležitosti:	**Hrozby:**
- příznivé vnější faktory ČR	-vysoká závislost na vnějším okolí
- velikost IDC ovlivňuje úspory z rozsahu	- ekonomické faktory ČR
- kapitálová náročnost vstupu do odvětví	- krize = nižší ziskovost zákazníků
- atraktivita ČR pro pracovní sílu ze zahraničí	- automatizace (Pokud automatizuje služby jiná konkurence, není to pod kontrolou a dochází k tlaku na využití stávající kapacity IDC)
- flexibilita (7 IDC firmy po celém světě)	
- přesun další složitější práce do IDC1	

Z této analýzy lze vyvodit spoustu doporučení pro budoucí vývoj centra v Brně. Já se v této prácí budu zabývat pouze těmi nejdůležitějšími.

Z mého pohledu je největší problém v kvalifikovaných lidských zdrojích. Pokud bude chtít IDC vykonávat lepší práci, musí poskytovat služby v určité kvalitě. Čím kompli-kovanější je požadovaná služba, tím více se projevuje vliv zkušenosti zaměstnanců na kvalitu těchto služeb. Firma tedy musí změnit politiku vůči zkušeným zaměstnancům, kteří pracují v IDC1 dlouhodobě (dnes to znamená zhruba 3 roky a více), nebo které Firma již vyškolila a poskytla jim certifikace. Vysoká fluktuace zkušených a certifiko-vaných zaměstnanců nutí IDC přibírat nové lidské zdroje za méně výhodných podmínek

pro firmu. Proto si myslím, že ze strategického hlediska bude výhodnější více ohodnotit stávající kvalifikované zaměstnance, kteří už mají potřebné certifikace, a kteří se navíc dobře orientují v procesech a kultuře firmy, než znovu zaučovat nového zaměstnance a platit jim drahé školení a certifikace. Navíc ti pak nebudou mít důvod z firmy odcházet a posilovat konkurenci.

Dalším mým návrhem je standardizace rolí. Ve všech centrech poskytujících služby existuje jasná organizační struktura. V té se pořád objevuje množství rolí, které vznikají společně se změnami strategie poskytování služeb zákazníkům. Tyto pracovní místa je nutné konkrétně popsat, aby se staly oficiálními. Někteří zákazníci vyžadují natolik nestandardní přístup, že není možné reagovat jinak, než vytvořením nestandardních pozic. Zaměstnanci na takových postech drží znalostní monopoly, a stávají se tím nenahraditelnými nebo alespoň velmi složitě nahraditelnými. IBM často přesouvá různé aktivity v rámci různých IDC center, a proto IDC Brno potřebuje standardní role. Jediným řešením je tlak vyššího managementu a vznik optimalizačního projektu.

2.1.3 portál eAMT

Tento portál je používán IBM v celé České republice. Je dostupný pouze z intranetu IBM, tedy přímo z pobočky IBM nebo použitím připojení VPN. Jeho účelem je evidovat hardware, ale jen některého druhu např. laptopy, desktopy, mobily. Nejdůležitější vlastností této databáze je, že zaměstnanci jsou hmotně odpovědní za hardware přidělený ke své osobě. Každý by si tedy měl dávat pozor, jestli nemá přidělený hardware, který fakticky nevlastní. Každý zaměstnanec se na tomto portálu může podívat na majetek, který je u něj evidován. Může jej přidělit jinému člověku nebo na sklad IGA. My jako tým máme jen základní práva k této databázi. V případě, že nám někdo pošle požadavek na přidělení hardwaru, který nám faktický nevrátil, máme právo tento požadavek odmítnout. Dále můžeme přeposílat hardware, který máme na skladě, například když vydáme počítač nováčkovi. Toto je vše k čemu my jako tým, starající se o veškerý hardware v Brněnském Delivery Centru, máme právo. Je jasné, že tento portál je pro naše účely nedostačující. Jeho konkrétním nedostatkům se budu věnovat v analýze problémů.

2.1.4 Rodina programů Lotus Notes/Domino

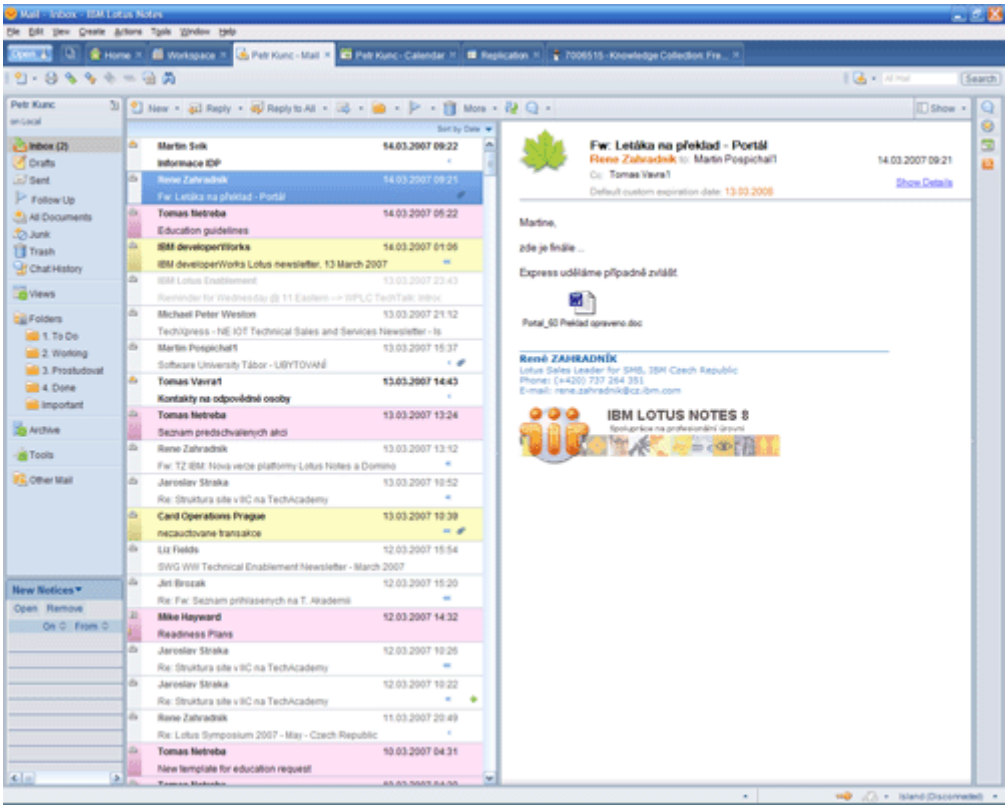
Lotus Notes

Je velmi složité definovat, co je vlastně Lotus Notes, protože LN znamená pro každého člověka něco jiného, jelikož každý člověk využívá jiné nástroje a LN jich poskytuje nepřeberné množství na jednom místě – emailový klient, kalendář, to do list, address book, online komunikace (Lotus Sametime) atd.

LN pomáhá lidem sdílet informace a nápady, organizovat čas atd. Je to jediný „pravý" groupware na světě, tedy jediný software, který poskytuje nástroje pro spolupráci více lidí na jednom projektu. Jediným dalším konkurentem je MS Exchange, ale tento software začínal pouze jako emailový klient MS Outlook, tedy není považován za „pravý" groupware. LN je od začátku svého vzniku vyvíjen jako groupware.

Velká výhoda LN je ve funkci replikace. Replikace vytvoří/zkopíruje .NSF soubor databáze ze serveru na pevný disk uživatelova počítače a uživatel muže díky tomu upravovat databáze i v offline módu. Jakmile se připojí do sítě, data se přenesou na server. V LN lze nastavit kriteria, kdy se má databáze replikovat např. každých 5 min, v pondělí v 5 ráno apod.

Což znamená menší zatížení pro servery. Uživatel nepřistupuje na server každým otevřením emailu, úpravou atributu databáze atd., ale pouze v průběhu replikace. LN je pouze část softwaru, který je používán konečným uživatelem. LN využívá aplikační/databázové servery programu Lotus Domino.

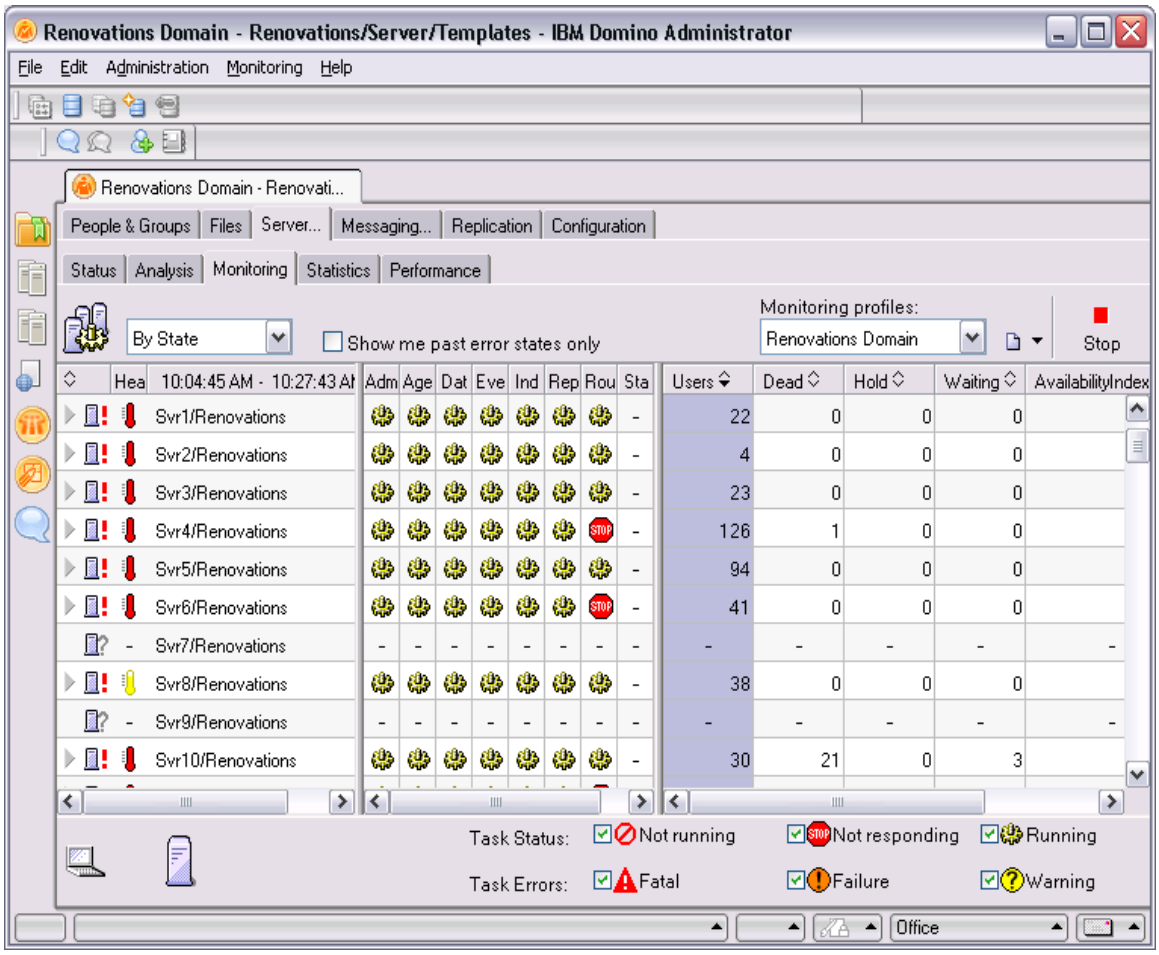


Obr. 4: Vzhled Lotus Notes – emailová schránka (Zdroj: Kunc, 2007)

Lotus Domino

Technologie Lotus Notes/Domino je postavena na typu klient-server, kde aplikace je uložena na serveru Lotus Domino a je využívána klientem používající Lotus Notes. Aplikace Lotus Domino můžou být umístěny na více serverech na jednou. LD má vestavěný databázový systém ve formátu .NSF. Tato databáze má mnoho rozdílu s SQL databázi, kterou tvořím. Nejvýraznější z nich je, že v databázovém souboru .NSF je uložen i vzhled databáze. Dalším důležitým rozdílem je, že se databáze vytváří pomocí grafického klienta Lotus Domino Administrátor, tedy nelze vytvořit pouze v textovém editoru jako SQL databáze. V tomto klientu je součástí i konzole pro zkušenější uživatele.

Skoro všechny problémy, které jsou spojeny s evidencí hardwaru, řeší naše HW evidence v programu Lotus Notes. LD Administrátor se používá pro hodně činnost spojených s LD, jako jsou např. správa uživatelů nebo skupin, jednotlivých serveru nebo clusteru více serverů, vytváření, editace a oprava databázi, přidávaní nebo odebírání práva atd.



Obr. 5 Vzhled Domino Administrátora (Zdroj: Lotus Domino documentation, 2010)

Bezpečnost

Hodně společností si LN koupili právě kvůli velmi dobré bezpečnosti. Velkými zákazníky, které velmi hodně zajímá bezpečnost, jsou státy. Mnohé České úřady používají LN. V čem spočívá tato bezpečnost, vysvětlím níže. Otázka kompletní bezpečnosti se skládá z více součástí: autentizace, autorizace, komunikace.

Autentizace

Lotus Notes přišel s revoluční novinkou – Lotus ID. Uživatel se autentizuje, nejen pomocí toho co zná (login a heslo), ale i pomocí toho co má (Lotus ID). Lotus ID je reprezentováno malým souborem uloženým na nějakém úložišti např. pevný disk. V tomto souboru jsou uloženy: certifikáty, doba jejich platnosti, soukromý klíč, identifikace uživatele atd. Pokaždé když chce uživatel pracovat se systémem, musí zadat Lotus ID.

Tento soubor lze otevřít jen pomocí hesla, které si vybere uživatel. Toto vše lze přirovnat k platební kartě a pinu.

Autorizace

Po autentizaci server ví, s jakou osobou komunikuje. Dále server musí rozhodnout k jakým operacím, je uživatel autorizován. Lotus Domino tento problém řeší v několika vrstvách.

V první vrstvě server rozhodne, zda uživatel může na server přistupovat. Pokud uživatel není v seznamu povolených lidí, nedostane se dále. V další úrovní LN využívá ACL (access control list), ve kterém nastavíme, kdo s ní může pracovat a jaká má práva. V ACL jsou různé úrovně přístupů např. bez přístupu, autor, editor, návrhář a manažer. Za třetí lze nastavit práva k jednotlivým dokumentům a i jednotlivým částem dokumentu.

Komunikace

LN dodržují standard S/MIME, takže komunikace mimo firemní sítě je důkladně zabezpečena. Server LD komunikuje s klientem LN pouze pomocí vnitřního protokolu. Tento protokol se připojuje pouze pomocí portu 1352. Tímto je vyřešena vnitřní komunikace. Lze zašifrovat i lokální repliky databází, tedy při odcizení laptopu, či zkopírování databázi, nedojde k odcizení dat uložených v databázích. U manažera i jiných zaměstnanců, kteří mají na disku uloženou lokální repliku např. svého emailu, kde mají spoustu citlivých informací, je toto šifrování nezbytné. Šifrování probíhá pomocí již zmíněného ID souboru. Lotus Domino může sloužit i jako certifikační jednotka, takže může vydávat certifikáty splňující normu X.509. Pomocí těchto certifikátů lze šifrovaně komunikovat.

Databáze Hardwarové evidence

Tato databáze splňuje hlavní nároky, které na ní klademe. Eviduje skoro veškerý hardware např. laptopy, desktopy, mobily, tiskárny atd. Jedinou věcí, která není v evidenci, jsou tonery.

Lze přiřadit/změnit vlastníka k jednotlivému hardwaru pomoci IBM address book. Lze vytvořit půjčku hardwaru pro určitého zaměstnance na dobu určitou. Toto trochu zne-

příjemňuje fakt, že pokud zaměstnanec nedostal ještě své LN ID, nemůžeme na něj hardware převést, protože není v IBM address booku. Vzhledem k tomu, že velmi často vydáváme počítače pro nováčky, je tento problém nepříjemný, protože nepřevedení počítače znamená zbytečné hledání při inventuře. Databáze disponuje několika pohledy např. veškeré počítače, všechny půjčky, mobily atd. Jak jsem již zmínil, databáze v LN mají danou i grafickou stránku. Databáze neřeší objednávky hardware a vzhledem k tomu, že velkou část hardwaru objednává jiný tým, je důležité tento problém odstranit.

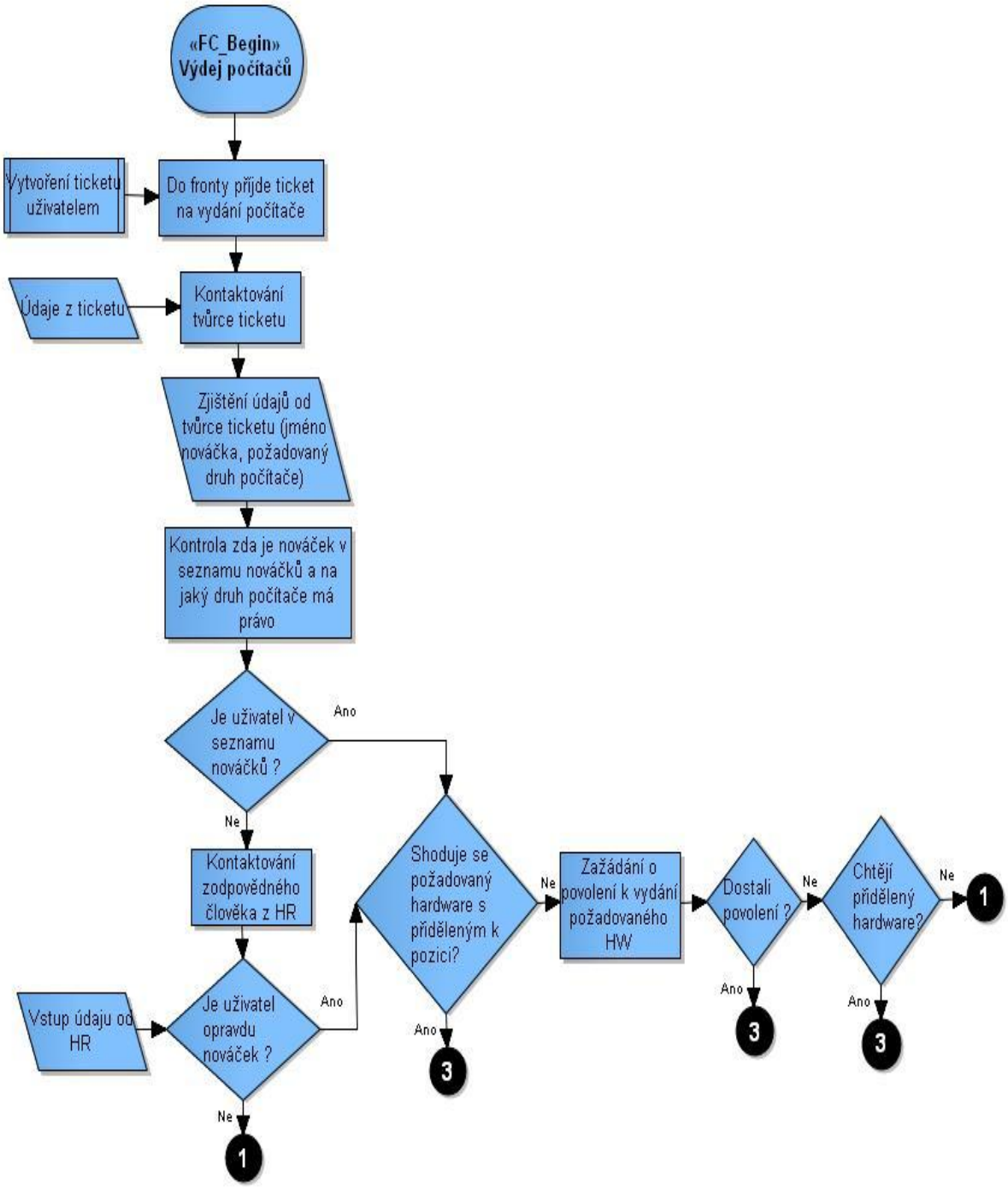
2.1.5 Interní procesy

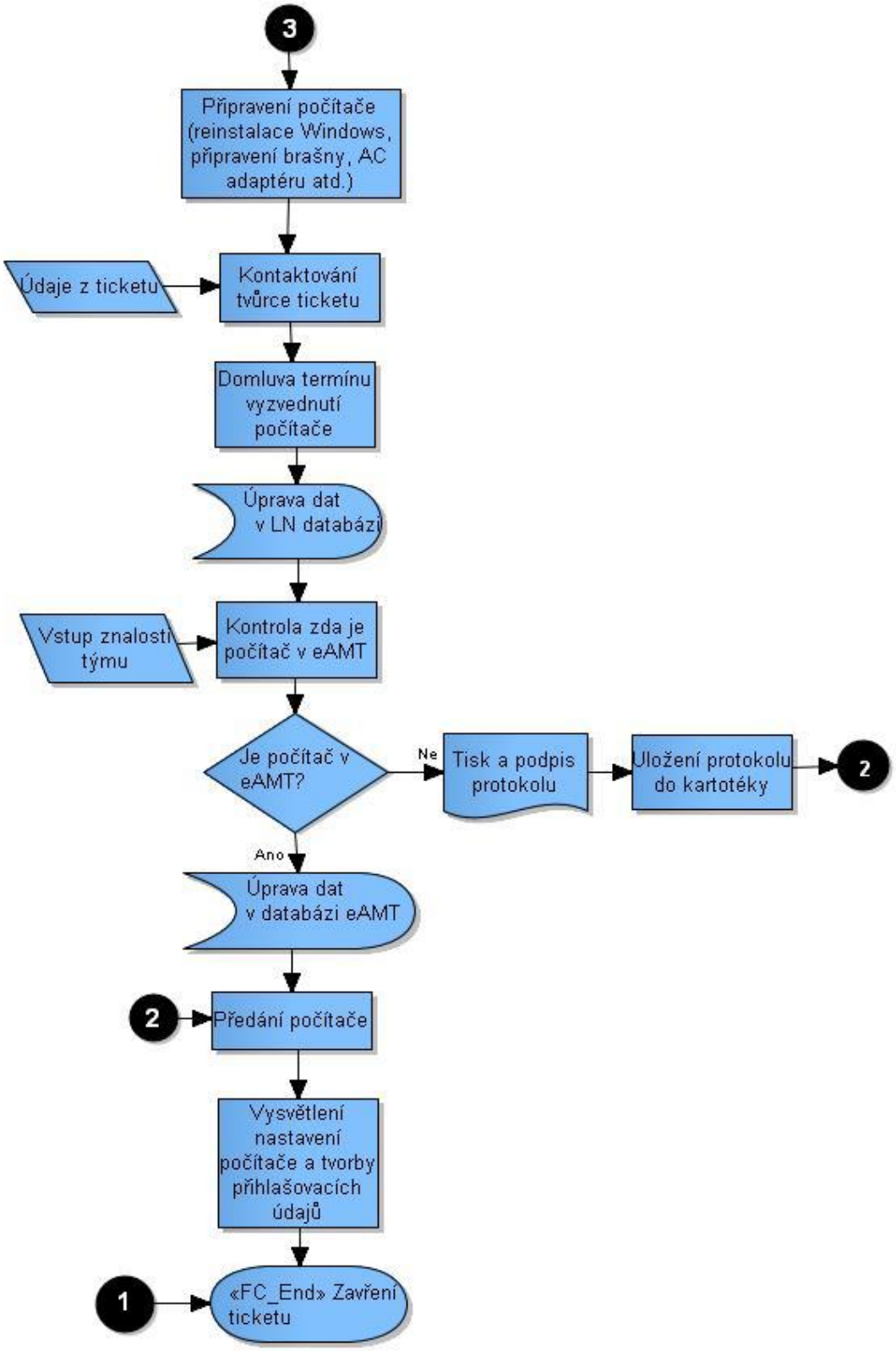
Interní procesy jsem umístil do analýzy současného stavu, protože díky mému návrhu řešení se změní minimálně. V podstatě jen vyměníme databázi, do které se budou zapisovat údaje. Nejdůležitější procesy z hlediska evidence hardwaru jsou: vracení počítačů, vydání počítačů a půjčovaní laptopů. Každý proces popíší slovně a vývojovým diagramem, který jej znázorňuje. Má databáze musí zohlednit všechny tyto procesy a umožnit jejich vykonání.

Výdej počítačů

Prvním krokem manažera, když chce získat počítač pro nováčka, je zavolání na helpdesk a vytvoření ticketu, kde specifikuje, pro jakého nováčka počítač chce. Tento ticket helpdesk přiřadí do naší fronty a jeden z nás rozhodne komu konkrétně ticket přidělí. Člověk, který ho dostane, zkontroluje seznam nováčků, který nám poskytuje HR, a zjistí, zda je uživatel opravdu nováček a zda má právo na laptop nebo desktop. Když tvůrce ticketu chce pro nováčka laptop a přitom má novaček právo pouze na desktop, může se obrátit na vedení týmu Brno Site Operations, kteří můžou rozhodnout o dodatečném vydání laptopu. Jestli má nováček právo na laptop, záleží na pozici, na kterou nastupuje. Dále už jen agent připraví počítač, přepíše údaje v hardwarové evidenci v programu LN a přepíše vlastníka v databázi eAMT.

Když počítač ještě není v databázi eAMT, tak musíme vytisknout předávací protokoly a zaměstnanec je musí podepsat. Tento problém nastává velmi často, protože objednáváme spousty počítačů a finanční oddělení, které má tuto databázi na starost, je v přidávání velmi pomalé. Dále se stačí domluvit se zaměstnancem, ať dojde k nám do kanceláře, a počítač předáme. Podepsané protokoly skladujeme jako důkazní materiál. Jako poslední krok následuje zavření ticketu.

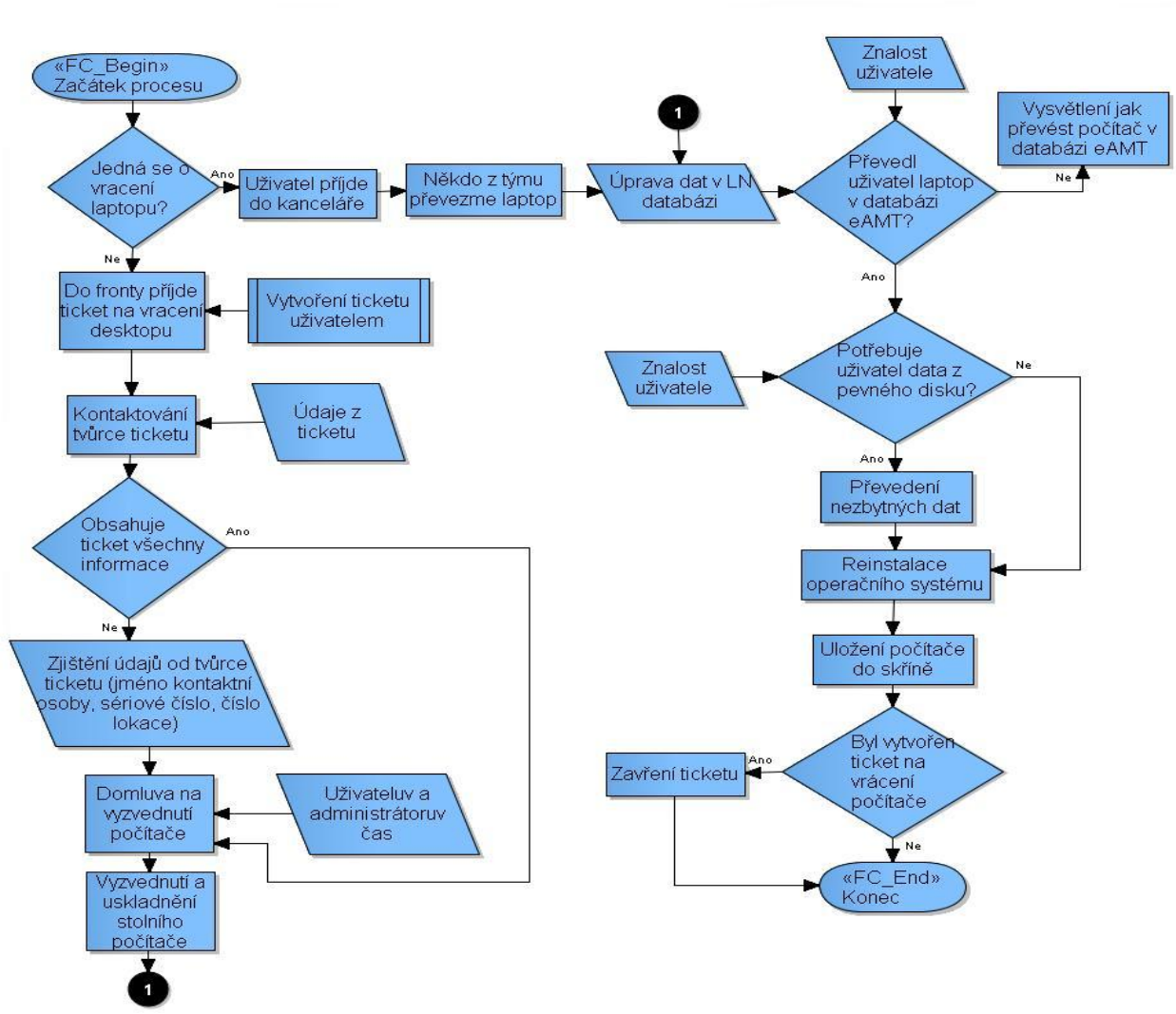

«FC_Begin»
Výdej počítačů
Vytvoření ticketu uživatelem
Do fronty přijde ticket na vydání počítače
Údaje z ticketu
Kontaktování tvůrce ticketu
Zjištění údajů od tvůrce ticketu (jméno nováčka, požadovaný druh počítače)
Kontrola zda je nováček v seznamu nováčků a na jaký druh počítače má právo
Je uživatel v seznamu nováčků ?
Ano
Ne
Kontaktování zodpovědného člověka z HR
Vstup údaju od HR
Je uživatel opravdu nováček ?
Ano
Ne
1
Shoduje se požadovaný hardware s přiděleným k pozici?
Ne
Ano
3
Zažádání o povolení k vydání požadovaného HW
Dostali povolení ?
Ne
Ano
3
Chtějí přidělený hardware?
Ne
1
Ano
3



Obr. 6: Vývojový diagram - Výdej počítačů (Zdroj: vlastní)

Vracení počítačů

Vracení počítačů probíhá obdobně. Když se jedná o stolní počítač zaměstnanec, vytvoří ticket, kde specifikuje sériové číslo počítače, kontaktní osobu a kde ho můžeme najít. Někdo z nás se s ním domluví, kde a kdy si počítač vyzvedne. Stolní počítače dáváme do skladu. Když se jedná o laptop, zaměstnanec přijde k nám do kanceláře a vrátí nám ho. Zeptáme se, zda má všechna data z laptopu. Když ne, data zkopírujeme na externí pevný disk. Dále reinstalujeme operační systém a dáme ho do příslušné skříně. Administrátor musí zaměstnanci říct, ať nezapomene počítač přepsat na sklad v databázi eAMT. My můžeme přepisovat pouze počítače, které jsou přiřazeny na sklad. Dále jen přepíše údaje v hardwarové evidenci v programu LN. Po ukončení činnosti musí zavřít ticket.

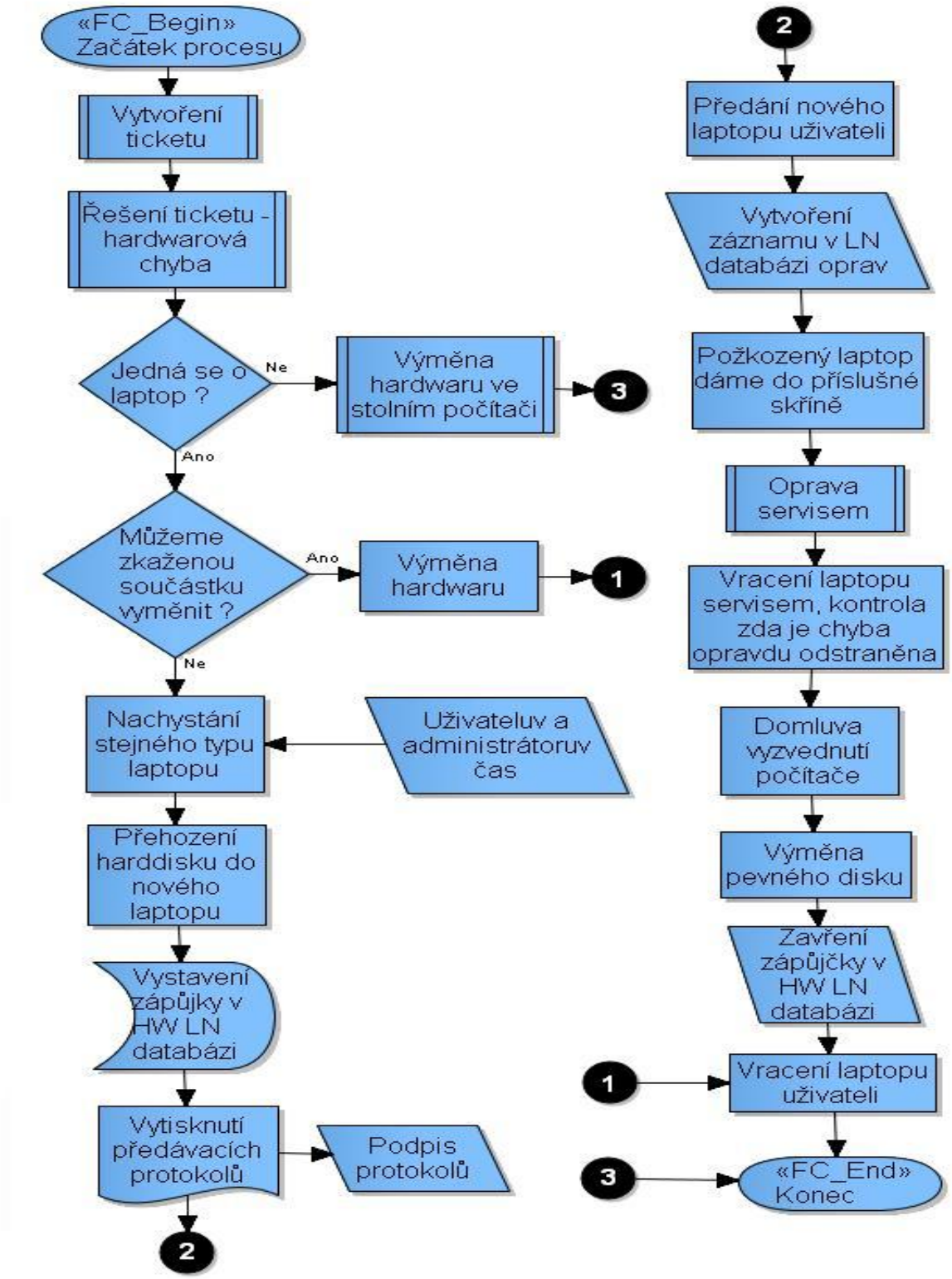


Obr. 7: Vývojový diagram - Vracení počítačů (Zdroj: vlastní)

Půjčování laptopů za laptopy poslané do opravy

Laptop, který má vážnou hardwarovou poruchu, například má zničenou základní desku, posíláme externí firmě na opravu. Tento proces začíná zase vytvořením ticketu zaměstnancem. Se zaměstnancem se domluvíme, ať přijde k nám do kanceláře. Identifikuji, jaká součástka je zničená. Pokud se jedná o něco, co můžu vyměnit bez pomoci např. pevný disk, větrák, udělám to.

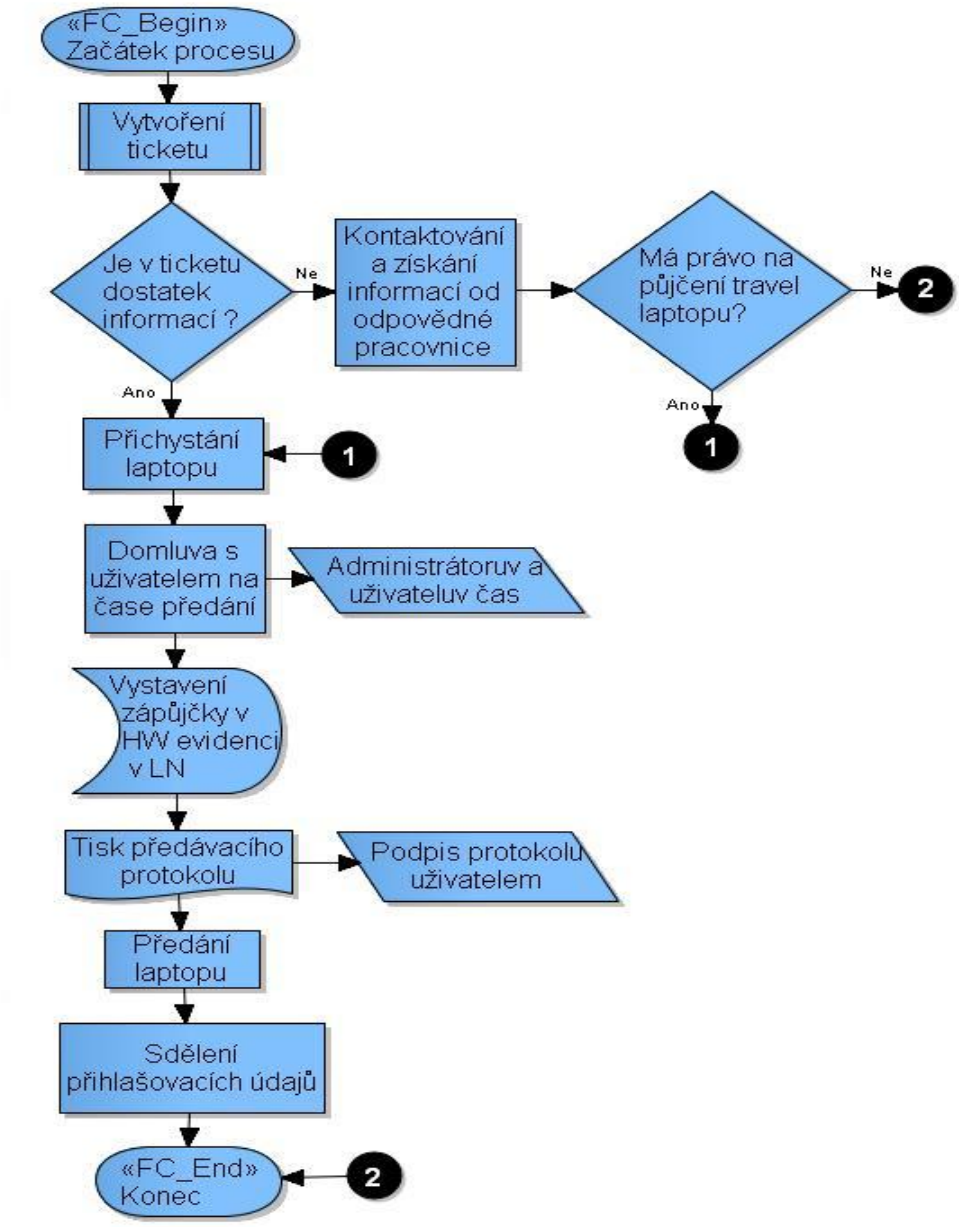
Když je to složitější, zaměstnanci půjčím laptop stejného typu. Tuto půjčku zapíšu do hardwarové evidence. Vyměním pevné disky v laptopech, aby zaměstnanec mohl pokračovat v práci. Zaměstnanec odejde s půjčeným laptopem. Já vytvořím záznam v databázi oprav a odložím počítač do příslušné skříně. Následně přijede technik z externí firmy a vyzvedne si veškerý poškozený hardware a přiveze ten opravený. Jakmile technik doveze zpátky opravený počítač, kontaktuji zaměstnance. Zase prohodíme pevné disky a zaměstnanec si vezme svůj počítač. Naposledy zavřeme výpůjčku v hardwarové evidenci a následně i ticket.



Obr. 8: Vývojový diagram - Půjčky laptopu za pokažené (Zdroj: vlastní)

Půjčování laptopů na služební cesty

Tickety na půjčení těchto laptopů vytváří pouze jedna osoba a je v něm uvedeno konkrétně, který laptop se má půjčit a na jaký časový interval. Vytvoříme zápůjčku a vytiskneme předávací protokoly. Předáme laptop, sdělíme přístupové údaje a zaměstnanec nám podepíše předávací protokol. Dále následuje proces vracení počítače.



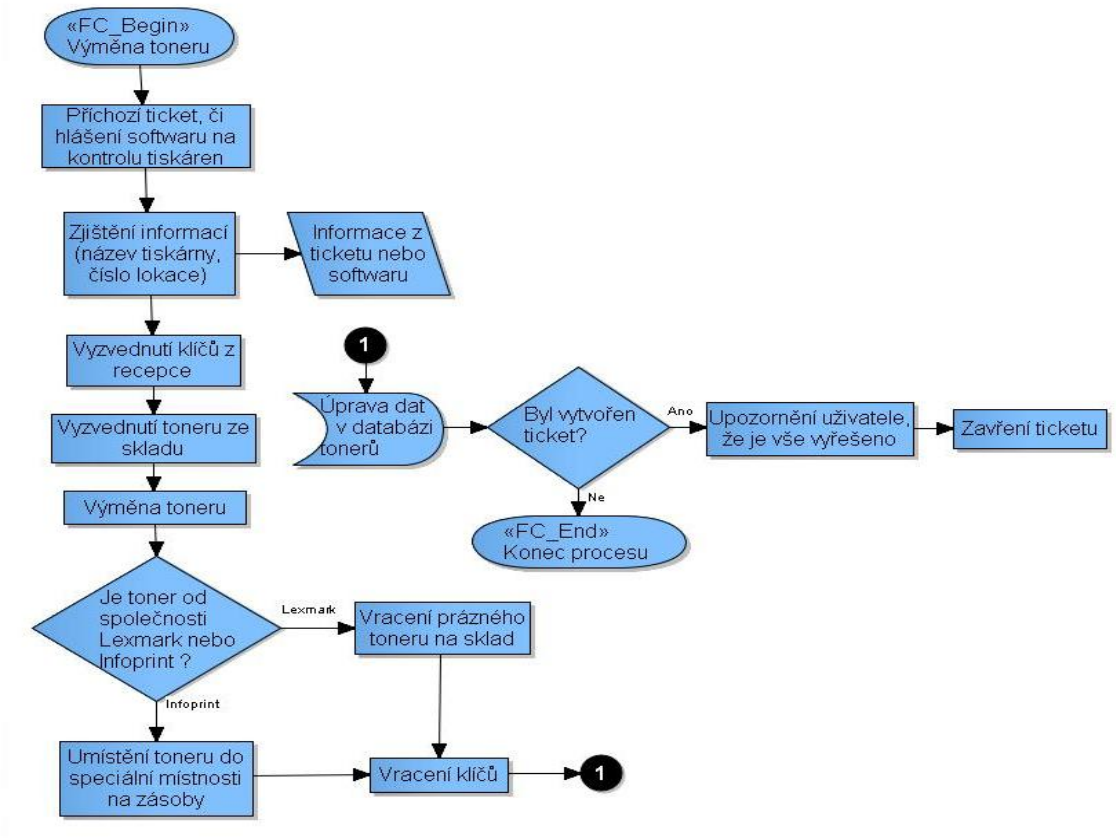
Obr. 9: Vývojový diagram - Půjčky laptopů na služební cesty (Zdroj: vlastní)

Objednávky hardwaru

Většinu hardwaru objednává jiný tým, ale přesto tento hardware přebíráme od dodavatelských firem my. Proto se vymyslel proces, který zabezpečuje, abychom nepřebrali neobjednaný hardware. Je to velmi jednoduchý proces, kdy člen týmu, který hardware objednává, musí upozornit emailem vedoucí pracovníky našeho týmu, a ti upozorní ostatní členy týmu nebo sami tento hardware přeberou od dodavatele.

Evidence tonerů

Tuto evidenci neřeší ani jedna z databází. Je vedena pouze jako tabulka v programu Lotus Syphony/Microsoft Excel, uložená jako soubor v LN databázi. Je to velmi nepraktické, protože musíme editovat LN databázi i samotný .ODF soubor. Navíc se zvyšuje pravděpodobnost, že to někdo udělá špatně, a pak může nastat situace, že musíme odstavit tiskárnu, protože fakticky nemáme toner, pouze jen číslo v databázi.



Obr. 10: Vývojový diagram - Výměna tonerů (Zdroj: vlastní)

2.2 Analýza problémů

Předchozí analýza poukázala na několik problémů, jejichž řešení bude hlavním kritériem při návrhu databáze. Nejdůležitější problémy jsou evidence tonerů a objednávek hardwaru. Tyto nedostatky odstraníme pomocí nového návrhu databáze. Hlavním důvodem tvorby databáze je, že chceme vytvořit webový portál, který by na základě této databáze počítal různé statistické údaje, a proto tomu chceme přizpůsobit i návrh databáze.

2.2.1 Interní procesy

Výdej počítačů, Vracení počítačů, Půjčování laptopů na služební cesty a Půjčování laptopů za laptopy poslané do opravy

Tyto procesy se radikálně nezmění. Bude to vypadat, jako kdyby neexistovala žádná databáze evidence hardwaru a všechno se evidovalo papírovou cestou. Nastanou spíše kosmetické úpravy. Například, když z nějakého důvodu nepůjde použít hardwarová evidence v programu LN, použije se tato SQL databáze.

Evidence tonerů, Objednávky hardwaru

Tyto procesy se změní nejvíce. Pro evidenci tonerů se přestane používat tabulkový kalkulátor a přejde se na SQL databázi. Toto řešení přinese úsporu času a hlavně přesnější evidenci. Budeme přesně vědět, kolik tonerů máme na skladě a vytvoříme funkci, která nás bude upozorňovat, když nemáme dostatek tonerů do každého typu tiskárny. Objednávky hardwaru se budou zapisovat do této databáze pro větší přehlednost. Každý člen týmu si bude moci zkontrolovat, zda jsme příchozí hardware opravdu objednali, a proto bez problému objednávku převzít.

3 Vlastní návrhy řešení

Na základě analýzy současného stavu a teoretických poznatků můžu přistoupit ke tvorbě návrhu databáze. Jak jsme se již dozvěděli z životního cyklu vývoje databáze, začnu s konceptuálním návrhem databáze. Identifikuji všechny entity, které chci do databáze uložit a dále vytvořím ERD diagram.

Dalším krokem je tvorba logického návrhu databáze. Zde jsou znázorněny všechny tabulky, které se následně vytvoří i jejich cizí klíče , které je propojují. Posledním krokem ve tvorbě databáze je fyzický návrh. Jedná se o faktické vytvoření tabulek, procedur, triggerů atd. pomocí SQL kódu.

3.1 Konceptuální návrh databáze

Hlavním cílem konceptuálního návrhu je vytvoření ER diagramu, který bude splňovat všechny požadavky na databázi. Aby se diagram lépe vytvářel, nejprve identifikuji všechny entity a jejich kardinalitu. Důležitou součástí je také identifikace atributů entit, které potřebuje uložit.

3.1.1 Identifikace entit

Identifikuji základní entity, které bude databáze obsahovat. Základní, protože toto nejsou všechny entity, které bude databáze obsahovat, jelikož konceptuální návrh databáze nezohledňuje dekompozici vztahů, normalizaci atd.

Tab. 2: Identifikace entit (Zdroj: vlastní)

Název entity	Popis entity
PC	osobní počítače
server	servery ve společnosti
printer	tikárny ve společnosti
mobile	mobily ve společnosti
sim_card	sim karty
manufacturer	Výrobce hardwaru
administrator	administrátoři systému (hlavně pro rozpoznání, kdo inicioval změnu v datábázi. Např. přepsání vlastníka laptopu Dále pro evidenci půjček a objednávek hw)
user	zaměstnanci firmy (evidence vlastníku hardwaru)
loan	Výpůjčky mobilu/pc

order	objednávky hardwaru
family	rodina hardwaru např. T410, M52
history	zde se bude ukládat historii změn u jednotlivého hardwaru
status	číselník možností statusů např. warehouse, in-use, closed
toner	tonery na skladu

3.1.2 Identifikace kardinality vztahů mezi entitami

Určíme všechny kardinality vztahů mezi entitami. Tato tabulka pomůže v pozdější fázi, provést dekompozici tabulek.

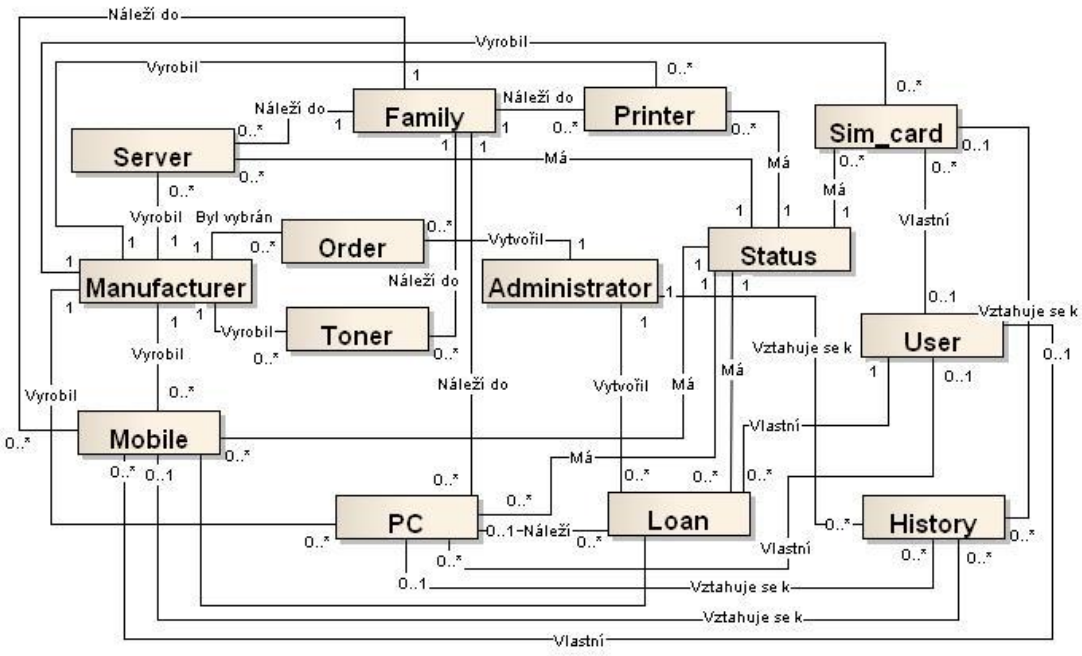
Tab. 3: Identifikace kardinality vztahů mezi entitami (Zdroj: vlastní)

entity	kardinalita vztahů mezi entitami	popis vztahu
PC - manufacturer	N:1	počítač může mít pouze jednoho výrobce a výrobce může vyrobit N počítačů
server - manufacturer	N:1	server může mít pouze jednoho výrobce a výrobce může vyrobit N serverů
mobile - manufacturer	N:1	mobil může mít pouze jednoho výrobce a výrobce může vyrobit N mobilů
sim_card - manufacturer	N:1	sim karta může mít pouze jednoho výrobce a výrobce může vyrobit N sim karet
user - PC	N:1	počítač může mít pouze jednoho vlástníka, ale vlastník může mít N počítačů
user - mobile	N:1	mobil může mít pouze jednoho vlástníka, ale vlastník může mít N mobilů
user- sim_card	N:1	sim karta může mít pouze jednoho vlástníka, ale vlastník může mít N sim karet
loan - administrator	N:1	jedna půjčka má jednoho administrátora a jeden administrátor může vytvořit N půjček
loan - user	N:1	půjčka má jednoho vlastníka a jeden vlastník může vytvořit N půjček
loan - mobile	N:1	půjčká může být pouze na jeden mobil a mobil může být půjčen N krát
loan - pc	N:1	půjčka je pro jeden pc a pc může být pujčeno N krát
loan -status	N:1	půjčka má vždy jeden status a jeden status může mít N půjček
order - manufacturer	N:1	objednávka má vždy jednoho výrobce a výrobce může být přiřazen k N objednávkám
order - administrator	N:1	objednávku vytvořil vždy jeden administrátor a jeden administrátor může vytvořit N objednávek
family - pc	N:1	počítač může mít jen jednu rodinu a do jedné rodiny může náležet N počítačů
family - mobile	N:1	mobil může mít jen jednu rodinu a do jedné rodiny může náležet N mobilů
family - server	N:1	server může mít jen jednu rodinu a do jedné rodiny může náležet N serverů

family - printer	N:1	tiskárna může mít jen jednu rodinu a do jedné rodiny může náležet N tiskáren
history - pc	N:1	počítač má N záznamů v historii a historie se váže jen na jeden počítač
history - mobile	N:1	mobil má N záznamů v historii a historie se vztahuje na jeden mobil
history - sim_card	N:1	sim karta má N záznamů v historii a historie se vztahuje na jednu sim kartu
history - administrator	N:1	historii vytváří pouze jeden administrátor a jeden administrátor může může vytvořit N historií
status - pc	N:1	počítač má pouze jeden status a jeden status může být přiřazen N počítačům
status - mobile	N:1	mobil má pouze jeden status a jeden status může být přiřazen N mobilům
status - printer	N:1	tiskárna má pouze jeden status a jeden status může být přiřazen N tiskárnám
status - sim_card	N:1	sim karta má pouze jeden status a jeden status může být přiřazen N sim kartám

3.1.3 ER Diagram

ER diagram je základ pro tvorbu databáze. Můžeme si povšimnout, že shodou okolností neobsahuje žádné vazby N:M, tedy v logickém návrhu nemusíme dekomponovat tabulky. Pro ulehčení tvorby diagramu jsem použil tabulky zmíněné výše. Tento ERD diagram popisuje pouze entity a vztahy mezi nimi.



Obr. 11: ERD diagram - Pouze entity (Zdroj: vlastní)

3.2 Logický návrh databáze

3.2.1 Datový slovník

Datový slovník je velmi důležitý dokument pro tvorbu databáze. Obsahuje všechny entity a můžeme z něj vyčíst, do jakých schémat jsme je rozdělili. V mém případě např. schéma hardware. Každá entita je dopodrobna popsaná. Jsou zde vidět všechny atributy, primární klíče, cizí klíče, datové typy atributů, délka jednotlivých atributů atd. Při tvorbě tabulek pomocí jazyka SQL jsem postupoval podle této tabulky.

Tab. 4: Datový slovník (zdroj: vlastní)

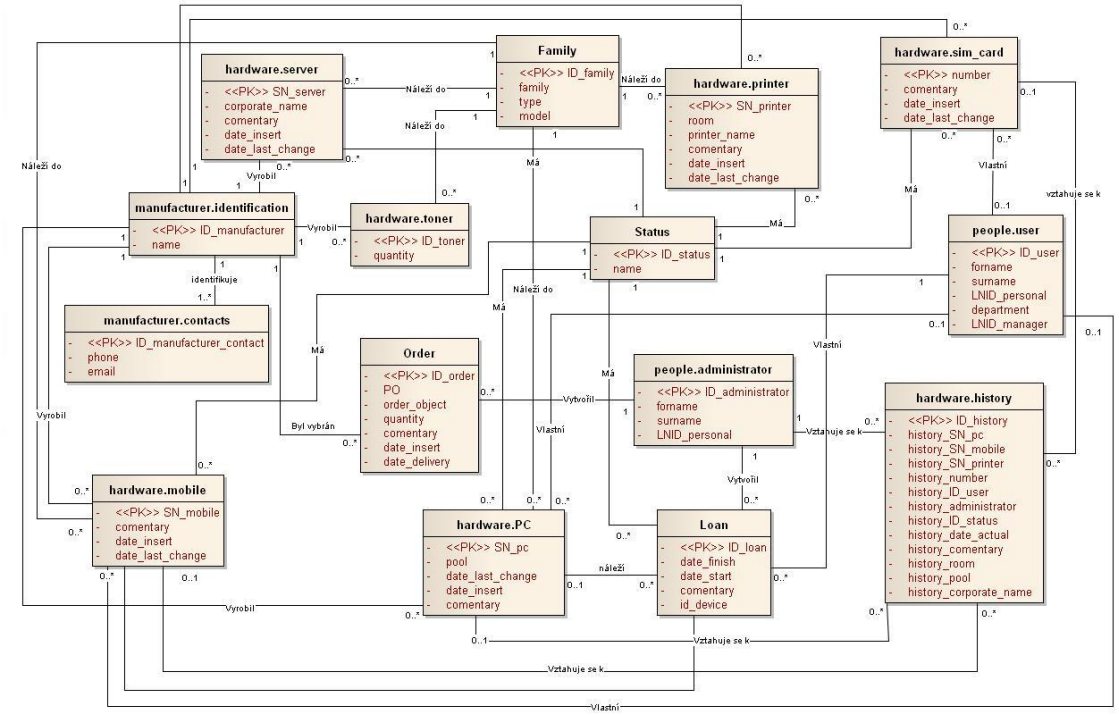
Tabulka	Položka	Typ	Délka	PK, FK	Další omezení
Hardware.pc	SN_pc	varchar	10	PK	
	ID_user	int		FK	
	ID_manufacturer	int		FK	
	ID_family	int		FK	
	ID_status	int		FK	
	pool	int	6		
	date_last_change	date			Not null, akt. datum

	date_insert	date			Not null, akt. datum
	comentary	varchar	75		
Hardware.server	SN_server	varchar	10	PK	
	ID_manufacturer	int		FK	
	ID_family	int		FK	
	ID_status	int		FK	
	corporate_name	varchar	20		Not null
	comentary	varchar	75		
	date_insert	date			Not null, akt. datum
	date_last_change	date			Not null, akt. datum
Hardware.printer	SN_printer	varchar	10	PK	
	ID_manufacturer	int		FK	
	ID_family	int		FK	
	ID_status	int		FK	
	room	varchar	5		Not null
	printer_name	varchar	10		Not null
	comentary	varchar	75		
	date_insert	date			Not null, akt. datum
	date_last_change	date			Not null, akt. datum
Hardware.mobile	SN_mobile	varchar	15	PK	
	ID_manufacturer	int		FK	
	ID_user	int		FK	
	ID_family	int		FK	
	ID_status	int		FK	
	comentary	varchar	50		
	date_insert	date			Not null, akt. datum
	date_last_change	date			Not null, akt. datum
Hardware.sim_card	number	varchar	9	PK	
	ID_user	int		FK	
	ID_manufacturer	int		FK	
	ID_status	int		FK	
	comentary	varchar	50		
	date_insert	date			Not null, akt. datum
	date_last_change	date			Not null, akt. datum
hardware.history	ID_history	int	Identity(1,1)	PK	
	history_SN_pc	varchar	10		
	history_SN_mobile	varchar	15		
	history_SN_printer	varchar	10		
	history_number	varchar	9		
	history_ID_user	int			
	history_administrator	varchar			DEFAULT SU-SER_SNAME()
	history_ID_status	int			
	history_Date_actual	date			

	history_comentary	varchar	75		
	history_room	varchar	5		
	history_pool	varchar	6		
	history_corporate_name	varchar	20		
hardware.toners	ID_toner	int	Identity(1,1)	PK	
	ID_manufacturer	int		FK	
	ID_family	int		FK	
	quantity	int			Not null
Manufactur-er.identification	ID_manufacturer	int	Identity(1,1)	PK	
	name	varchar	20		Not null
Manufactur-er.contact	ID_manufacturer_contact	int	Identity(1,1)		
	ID_manufacturer	int			
	phone	varchar	20		
	email	varchar	35		
Peo-ple.administrator	ID_administartor	int	Identity(1,1)	PK	
	forename	varchar	15		
	surname	varchar	20		
	LNID_personal	varchar	7		Not null
People.users	ID_user	int	Identity(1,1)	PK	
	forename	varchar	20		
	surname	varchar	20		
	LNID_personal	varchar	7		Not null
	department	varchar	6		
	LNID_manager	varchar	7		
Loan	ID_loan	int	Identity(1,1)	PK	
	ID_user	int		FK	
	ID_administartor	int		FK	
	SN_mobile	varchar	10	FK	
	SN_pc	varchar	10	FK	
	ID_status	int		FK	
	date_finish	date			not null, Akt. datum
	date_start	date			not null, Akt. datum
	comentary	varchar	50		
	ID_device	tinyint			Not null
Order	ID_order	int	Identity(1,1)	PK	
	ID_administrator	int		FK	
	ID_manufacturer	int		FK	
	PO	varchar	15		
	order_object	varchar	20		Not null
	quantity	int			
	comentary	varchar	50		
	date_insert	date			not null, act. Date
	date_delivery	date			not null, act. Date
Family	ID_family	int	Identity(1,1)	PK	
	family	varchar	10		
	type	varchar	4		
	model	varchar	4		
Status	ID_status	int	Identity(1,1)	PK	
	name	varchar	9		in-use,scrap, atd

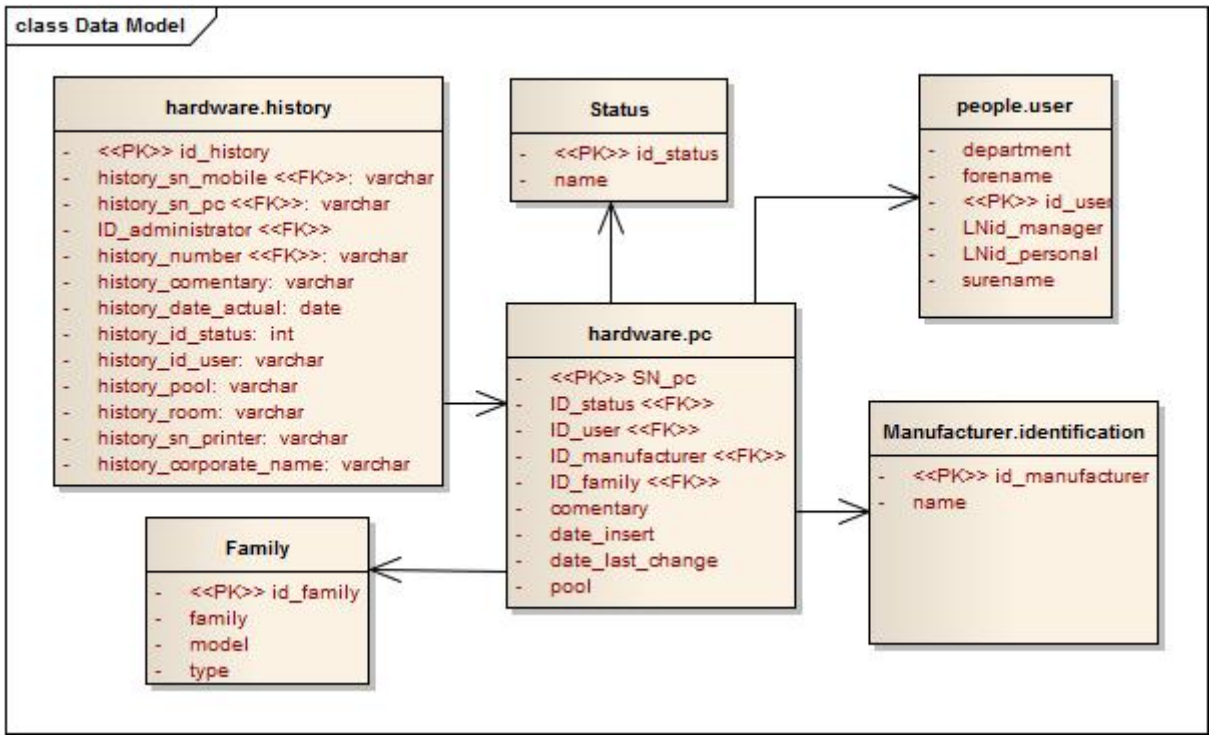
3.2.2 Logický návrh databáze

Tento návrh je velmi podobný ER diagramu, protože jak jsem již zmínil, nemusíme dekomponovat tabulky se vztahem N:M. Jedinou zásadní změnu jsem provedl v tabulce Manafacturer, kdy jsem ji rozdělil na dvě tabulky, abych mohl k jednomu výrobci ukládat více kontaktů. V tomto diagramu v podstatě propojíme datový slovník a ERD diagram, které jsou znázorněné výše. Následně diagram rozdělím do jednotlivých okruhů podle funkce, kterou provádí. Jednotlivé okruhy vysvětlím a díky tomu bude dobře pochopitelný celkový obsah logického návrhu.



Obr. 12: Logický návrh databáze (Zdroj: vlastní)

3.2.3 Evidence počítačů



Obr. 13: Logický návrh - evidence počítačů (Zdroj: vlastní)

Primárním klíčem tabulky pc jsem zvolil sériové číslo počítače. V tabulce je zastoupen atributem SN_pc. Následují cizí klíče do tabulek status, family, manufacturer a user. Atribut pool určuje, do jaké skupiny laptopů počítač patří. Nejdůležitějšími skupinami jsou travel, repair a loan laptopy, tedy na cesty, půjčky a místo porouchaných laptopů. Poslední tři atributy jsou v každé tabulce mimo tonerů. Jedná se o datum vložení položky do tabulky, datum poslední změny a komentář k položce

Zbývající tabulky popíši pouze tady, protože se budou často opakovat.

Tabulka status má primární klíč ID_status. Je to číselník, který obsahuje pouze jméno statusu např. expire, closed.

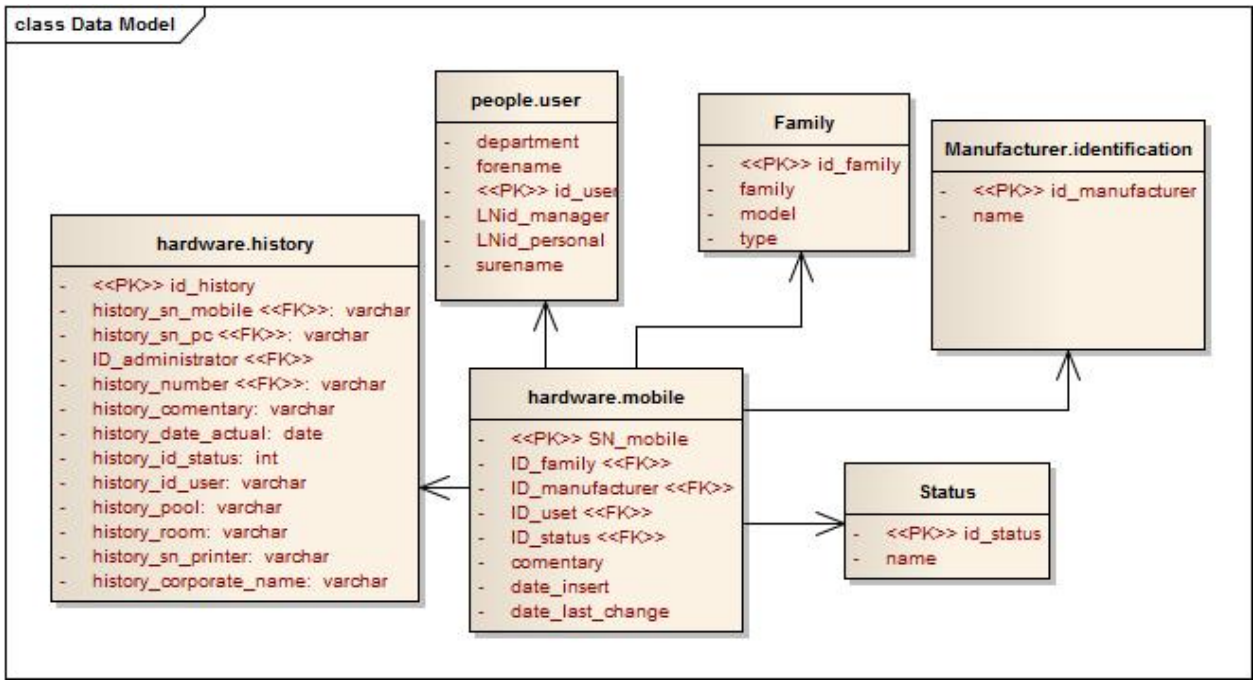
Schéma people v sobě zahrnuje dvě tabulky. V tomto diagramu se vyskytuje pouze tabulka user. Tabulka user má primární klíč ID_user. Samozřejmě obsahuje atributy jméno a příjmení. Důležitými údaji jsou LNID_personal, které ukládá Lotus Notes ID uživatele, a LNID_manager, které ukládá Lotus Notes ID manažera od uživatele. Posledním atributem je department, kde se uloží oddělení, ve kterém uživatel pracuje.

Tabulka manufacturer.identification ukládá jednotlivé výrobce a skládá se pouze z dvou atributů. ID_manufactuerer je primární klíč tabulky. Atribut name ukládá jméno výrobce hardwaru.

Tabulka family podrobněji popisuje hardware, protože jen název výrobce nestačí. Nachází se zde atributy family, model a type. Family ukládá rodinu hardwaru např. T410 od výrobce Lenovo. Type ukládá typ jednotlivé rodiny např. 2500. Laptop se stejnou rodinou a jiným typem je vzhledově úplně stejný, ale liší se v některých parametrech např. má jiné množství RAM paměti nebo jiný procesor. Atribut model ještě upřesňuje jednotlivý typ.

Zbývající tabulka zaznamenává historii. Do této tabulky nebude nikdo nic zapisovat. O zápis se postará trigger, který se spustí vždy, když někdo upraví tabulku pc. Zapíše údaje, které se smažou touto úpravou, abychom zachovali staré údaje např. bývalého vlastníka počítače.

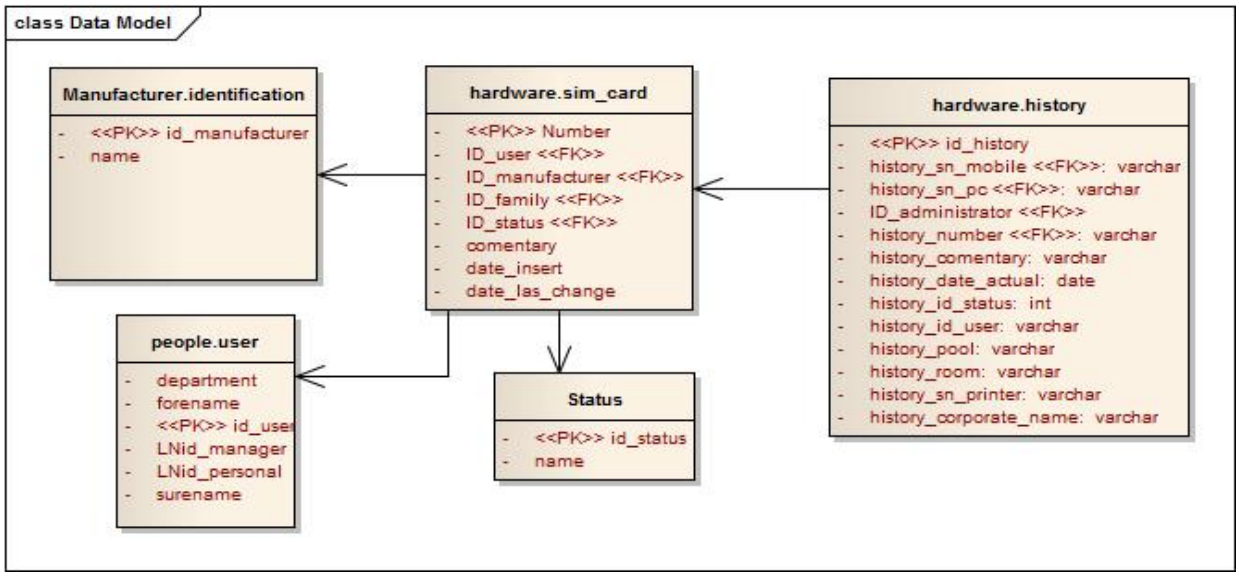
3.2.4 Evidence mobilních telefonů



Obr. 14: Logický návrh - evidence mobilů (Zdroj: vlastní)

V tabulce mobile jsem určil primárním klíčem sériové číslo mobilu, uložené jako SN_mobile. V této tabulce jsou stejné cizí klíče jako u tabulky pc.

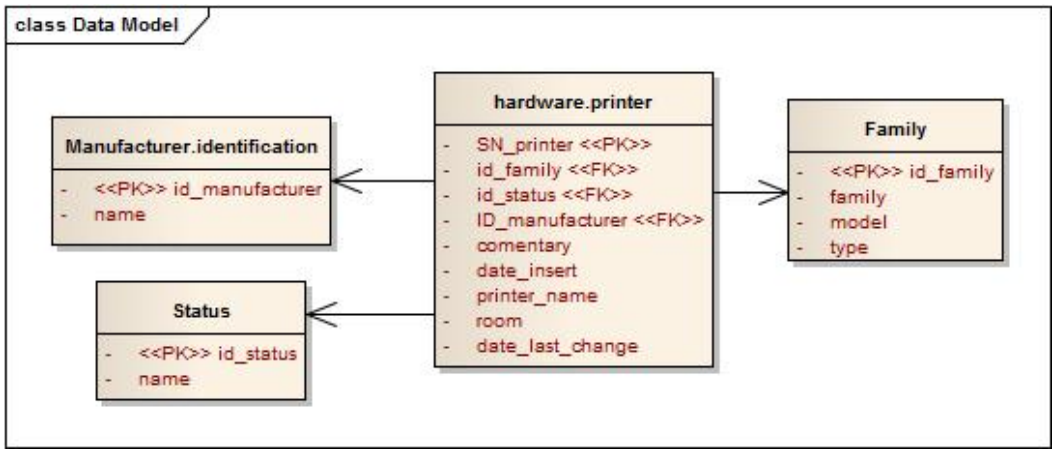
3.2.5 Evidence sim karet



Obr. 15: Logický návrh - evidence sim karet (Zdroj: vlastní)

V této tabulce je primárním klíčem atribut number. Jedná se o číslo sim karty. Atribut bude omezen pouze na 9 znaků, aby se co nejvíce zredukovala možnost různých druhů zápisu. Také zde jsou stejné cizí klíče a zbytek atributů např. comentary atd.

3.2.6 Evidence tiskáren

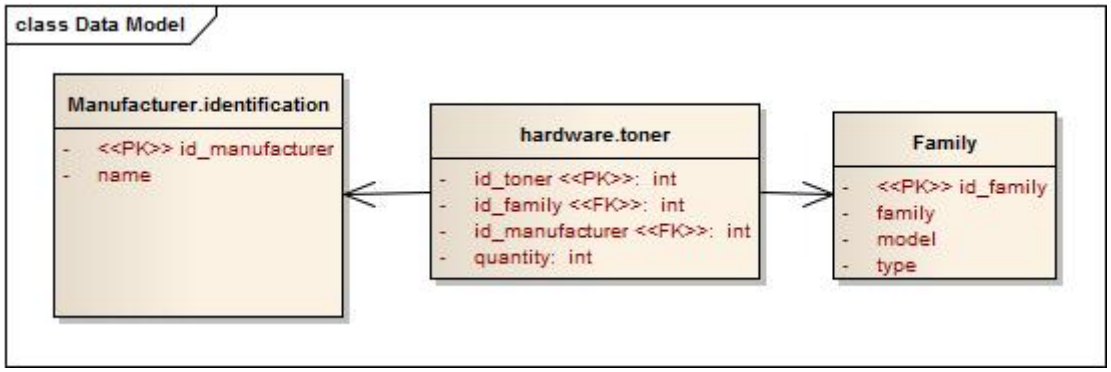


Obr. 16: Logický návrh - evidence tiskáren (Zdroj: vlastní)

Tabulka printer ukládá údaje o tiskárnách. Jejím primárním klíčem je sériové číslo tiskárny uložené jako SN_printer. Zde jsou jen tři primární klíče, a to id_family, id_status a id_manufacturer. Cizí klič id_user chybí, protože nemá smysl ukládat vlastníky tiskárny, protože je nikdo nevlastní. Atribut printer_name ukládá jméno tiskárny. Atribut

room místnost, kde je tiskárna umístěna. Posledními atributy jsou opět data vložení a poslední změny.

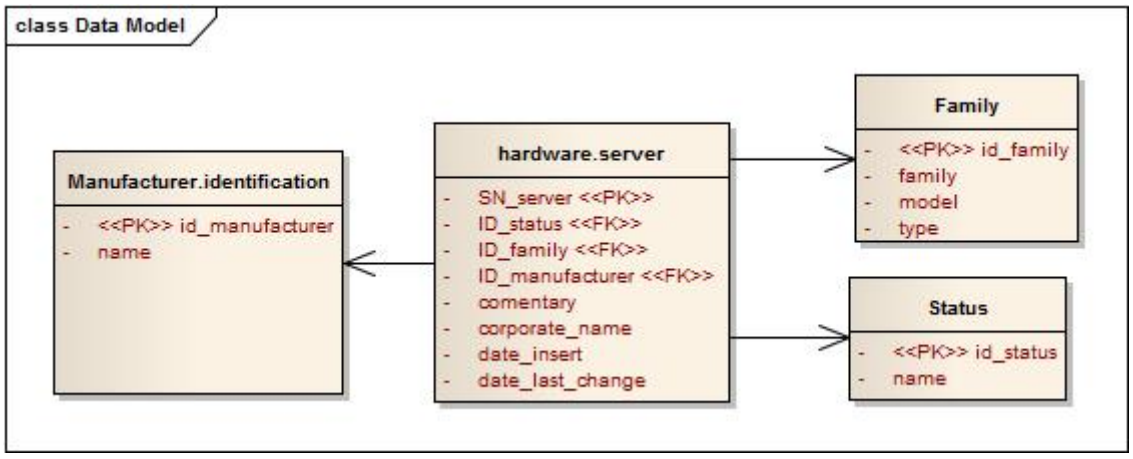
3.2.7 Evidence tonerů



Obr. 17: Logický návrh - evidence tonerů (Zdroj: vlastní)

Primárním klíčem v tabulce toner je id_toner. Hodnota tohoto atributu se bude automaticky zvětšovat o jedna s každým přibývajícím záznamem. Tato tabulka má dva cizí klíče id_family, id_manufacturer. Nejdůležitějším atributem je quantity. Tady se bude ukládat počet tonerů na skladu. Také vytvořím funkci na odečítání množství, aby se tabulka mohla jednoduše editovat, když se nějaký toner použije.

3.2.8 Evidence serverů

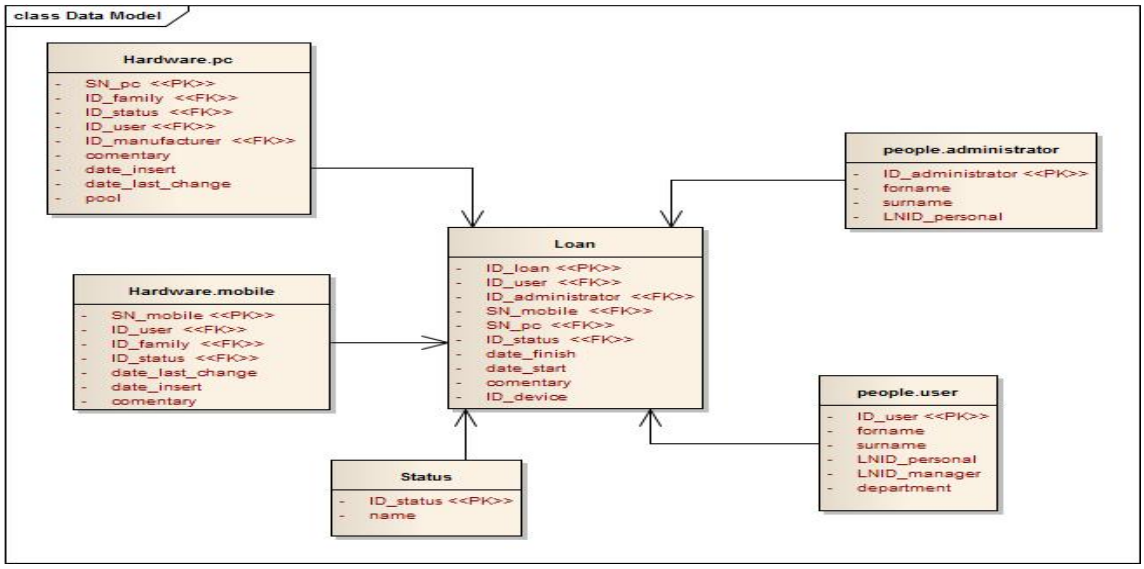


Obr. 18: Logický návrh - evidence serverů (Zdroj: vlastní)

Primárním klíčem tabulky server je sériové číslo serveru, tedy atribut SN_server. Tabulka obsahuje tři cizí klíče. Tyto klíče propojují tabulku s tabulkami family, status, manufacturer.identification. Specialní atribut je corporate_name, který ukládá jméno

serveru, které se používá po celé IBM. Ani zde nechybí komentář, datum vložení a datum poslední změny.

3.2.9 Půjčování hardwaru



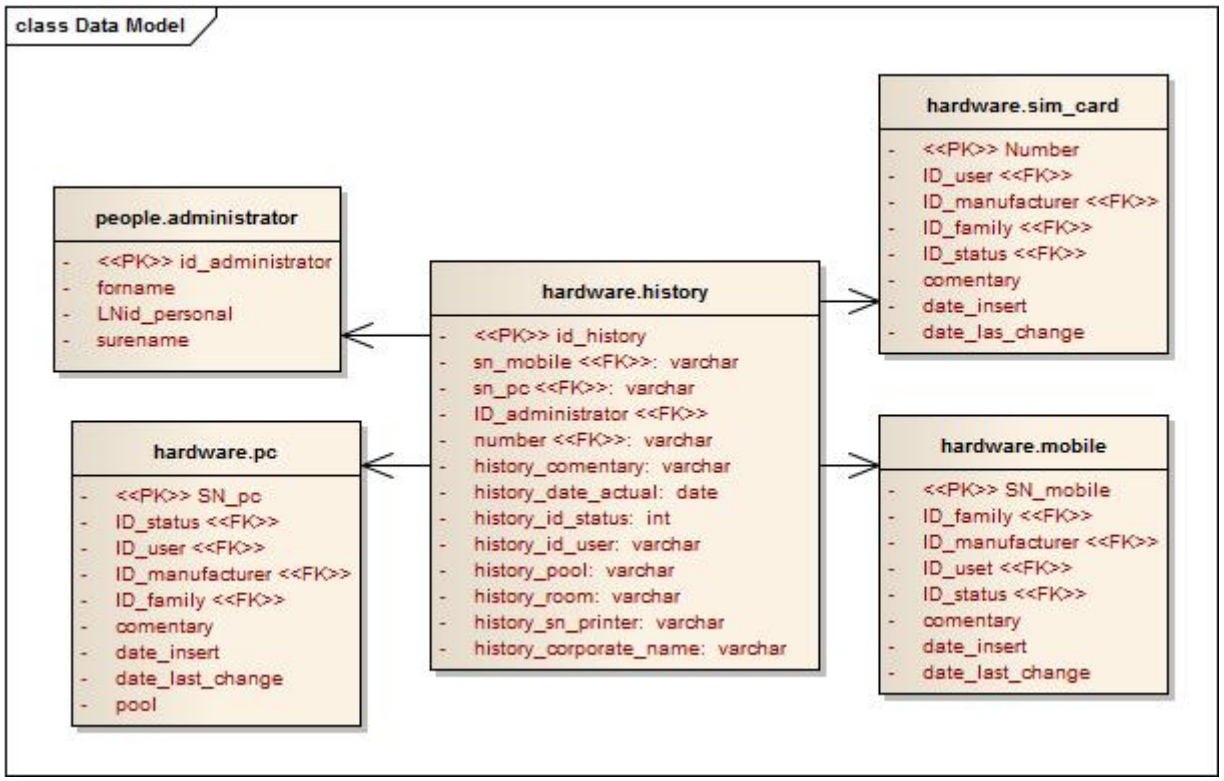
Obr. 19: Logický návrh – půjčování hardwaru (Zdroj: vlastní)

Nejdůležitější entitou z tohoto schématu je entita Loan. Tato entita je určena primárním klíčem ID_loan a obsahuje základní informace o půjčce. Atributy date_finish, date_start a ID_device musí být vyplněny. Zbylé atributy jsou cizí klíče, které propojují tabulku s ostatními tabulkami. Atribut ID_device bude usnadňovat budoucí programování i tvorbu dotazů na databázi. V tomto případě stačí dvě hodnoty. Když se bude jednat o laptop, bude roven nule, když o desktop, bude roven jedné. Když bude aktuální datum větší než atribut date_finish, tak se přepíše ID_status na hodnotu expire a odešle se email uživateli a jeho manažerovi, že musí příslušný hardware vrátit. Přepis statusu realizujeme pomocí kurzoru.

Schéma people v sobě zahrnuje dvě tabulky. Tabulka people.user má primární klíč ID_user a ukládá základní údaje o uživateli. Tabulka people.administrator je velmi podobná, akorát uchovává údaje o administrátorech a její primarní klíč je ID_administrator.

Schéma hardware ve vztahu k půjčkám má dvě tabulky. Tyto tabulky a tabulku status jsem již představil v předchozí kapitole.

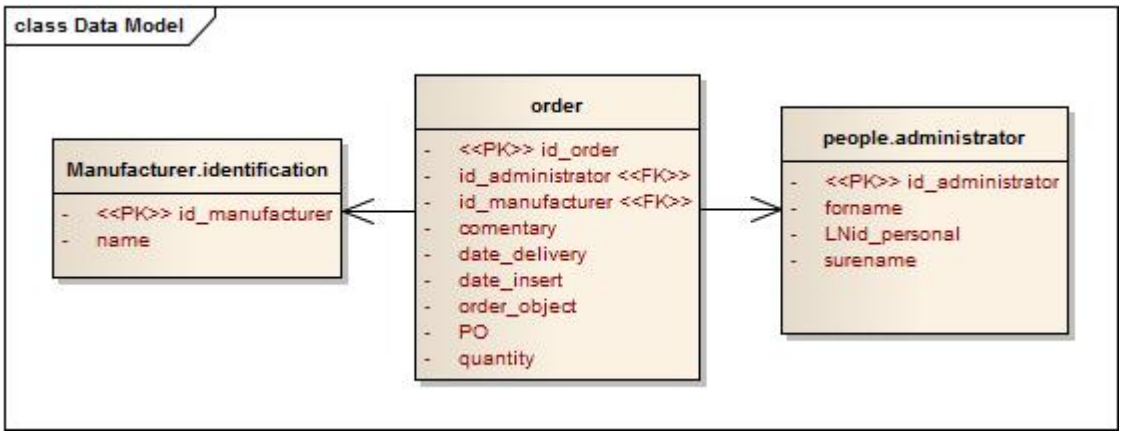
3.2.10 Ukládání historie



Obr. 20: Logický návrh - Historie (Zdroj: vlastní)

Primárním klíčem tabulky hardware.history je automaticky generované číslo zastoupené atributem id_history. Cizími klíči jsem zvolil id_administrátora, sn_pc, sn_mobile a number. Ostatní tabulky schématu hardware jsem vynechal, protože pro ně nemá smysl ukládat historii. Databáze by šla jednoduše předělat, aby ukládala historii pro všechny tabulky, ale je to zbytečné. Všechny atributy začínající "history_" jsou hodnoty, které byly přemazány úpravou jednotlivých řádků tabulek. Pro každý sledovaný atribut jsem musel vytvořit atribut v tabulce history. O tom jak se budou data ukládat, napíšu až ve fyzickém návrhu databáze. Zbylé tabulky jsem popsal výše

3.2.11 Objednávky hardwaru



Obr. 21: Logický návrh - objednávky hardwaru (Zdroj: vlastní)

Hlavní tabulkou je tabulka order. Budeme do ní zapisovat veškeré údaje týkající se objednávek hardwaru. Tabulka obsahuje primární klíč, který se automaticky generuje. Dále obsahuje dva cizí klíče id_administrator a id_manufacturer, které propojují tabulku order s tabulkami people.administrator a manufacturer.identification. Tato tabulka má také atributy comentary a date_insert. Zvláštními atributy jsou date_delivery, order_object, PO a quantity. Atribut date_delivery bude obsahovat předpokládané datum doručení hardwaru. Do atributu order_object budeme ukládat objekt objednávky např. laptopy Lenovo t410. Atribut PO je číslo objednávky, které je zde uvedeno, protože musíme kontrolovat hardware při přijímání od firmy zabývající se logistikou. Atribut quantity určuje množství hardwaru, které bylo objednáno.

3.3 Fyzický návrh

V této části jsem vytvořil kód v jazyce SQL. Můžete jej najít v příloze číslo 1. Datový slovník a logický návrh, z kterých jsem vycházel, najdete výše v textu. Zde se budu zabývat jen vybranými procedurami, kurzory, pohledy a triggery.

3.3.1 Pohledy

První pohled zobrazuje vlastníka počítače a vybrané údaje o počítači. Pohled se opakuje pro každou tabulku, kde má smysl zobrazovat vlastníka hardwaru. Jde v podstatě o dotaz select, kterým propojím čtyři tabulky, a to hardware.pc, status, family a people.user.

Tabulku manufacturer jsem záměrně vynechal, protože nemáme žádné počítače s jinou značkou než Lenovo. V návrhu jsem možnost propojení s výrobcem zanechal, protože se firma může rozhodnout změnit dodavatele. Left outer join u tabulky user jsem použil, aby se mi zobrazily i počítače, které nemají vlastníka, tedy jsou na skladu.

```
create view owner_workstation as
select pc.sn_pc, pc.pool, s.name as status, f.family, pe.forname,
pe.surname, pc.date_last_change, pc.date_insert, f.model,
f.type,ma.name
from hardware.pc pc
      join status s
on pc.id_status = s.id_status
      join family f
on    pc.id_family = f.id_family
      left outer join people.users pe
on pc.Id_user=pe.Id_user
      join manufacturer.identification ma
on ma.Id_manufacturer = pc.id_manufacturer

go
```

Další pohled se týká tabulky Loan a zobrazí všechny půjčky a důležité údaje. Obdobné pohledy jsem vytvořil i pro tabulky hardware.toner a order. Pomocí dotazu select jsem propojil tabulky loan, people.user, people.administrator, hardware.mobile, hardware.pc a status. Některé atributy jsem musel přejmenovat, protože jejich názvy byly duplicitní. Left outer join jsem použil ze stejného důvodu jako v minulém pohledu. Půjčka se vždy vztahuje pouze k jedné věci, tudíž je jasné, že vždy bude atribut sn_pc nebo sn_mobile prázdné. Kdybych nepoužil outer join, pohled mi nezobrazí všechny půjčky.

```
create view view_loan as
select s.name as Status,pe.forname as Users_Forname, pe.surname as
Users_Surename,pe.lnid_personal,pa.forname as Admnistrator_Forname,
pa.surname as Administrator_Surname, pc.sn_pc, mo.sn_mobile,
lo.date_start, lo.date_finish,lo.comentary
from loan lo
      join people.users pe
on lo.id_user = pe.Id_user
      join people.administrator pa
on pa.Id_administrator = lo.id_administrator
      left outer join hardware.pc pc
on pc.sn_pc = lo.Sn_pc
      left outer join hardware.mobile mo
on mo.sn_mobile = lo.SN_mobile
      join status s
on s.id_status = lo.id_status

go
```

Poslední druh pohledu, který zmíním, je náhled na historii jednotlivého druhu hardware. Pro každý druh hardwaru, u kterého to má smysl, zobrazím všechnu historii. Tento pohled využiji v proceduře, ve které hledám historii pro konkrétní kus hardwaru. Zase používám outer join pro propojení tabulek. Atribut name z tabulky status jsem přejmenoval pro lepší orientaci v pohledu.

```
create view history_pc
as
select
hi.sn_pc,pe.forname,hi.history_pool,pe.surname,hi.history_administrato
r, s.name as status, hi.history_comentary
from hardware.history hi
      right outer join people.users pe
on (pe.id_user = hi.history_id_user)
      right outer join hardware.pc pc
on (pc.sn_pc = hi.sn_pc)
      join status s
on (s.id_status = hi.history_id_status)
go
```

3.3.2 Procedury

Pro každou tabulku jsem vytvořil proceduru, která vkládá údaje do tabulek. Je to velmi jednoduchá procedůra, která využívá dotaz insert into. V případě tabulky people.user musíme zadat povinné údaje jméno, příjmení, LNID uživatel, LNID manažera uživatele a číslo oddělení. Tyto údaje, které zadávame se samozřejmě liší pro každou tabulku.

```
create procedure insert_people_users
@forname varchar(20),
@surename varchar(20),
@LNid_personal varchar(7),
@LNid_manager varchar(7),
@department varchar(6)

as

insert                          into                          people.users
      (forname,surname,LNID_personal,LNID_manager,Department)
values (@forname,@surename,@lnid_personal, @LNid_manager,@department)
go
```

Dotaz pro vyvolání procedury

```
execute insert_people_users

'Martin','Valek','y952365','z23565','01f2s3'
```

Abych mohl tabulky jednoduše upravovat, vytvořil jsem procedury, využívající dotaz update. Každé tabulce odpovídá jedna procedura. Musíme zase zadat povinné údaje

jméno, příjmení, LNID uživatel, LNID manažera uživatele a číslo oddělení, ale zde přibýva ještě ID_user, protože musíme rozeznat jaký řádek tabulky chceme upravovat.

```
create procedure update_people_user
@forname varchar(20),
@surename varchar(20),
@LNid_personal varchar(7),
@LNid_manager varchar(7),
@department varchar(6),
@id_user int

as

update people.users

set                 forname=@forname,                surname=@surename,
LNID_manager=@LNid_manager,LNID_personal=@LNid_personal,
Department=@department
where Id_user = @id_user

go
```

Dotaz pro vyvolání procedury

```
execute update_people_user
'Martin','Valek','y81411',' z23565','015f36',8
```

Tato procedura se bude používat na vyhledání historie pro konkretní kus hardwaru, ať už se jedna o počítač, mobil či sim kartu. Využiji atributu ID_device, podle kterého rozeznám o jaký druh hardwaru se jedná. Když se bude rovnat jedné, jde o mobil atd. V procedůře využívam pohledy history, které jsem vytvořil dříve.

```
create procedure find_history
@sn varchar (15),
@id_device int

as

if @id_device = 1
select * from history_mobile
where sn_mobile = @sn

if @id_device = 2
select * from history_pc
where sn_pc = @sn

if @id_device = 3
select * from history_simcard
where number = @sn

go
```

Dotaz pro vyvolání procedury

```
execute find_history

'l3f125g',2
```

Poslední proceduru, kterou zmíním je reduce_quantity_toner. Jak již z názvu vyplývá jedná se o proceduru, která snižuje množství toneru na skladě zadané hodnoty. Můžeme si všimnou, že proměnná @quantity není povinná a je předdefinována na hodnotu jedna, protože se tato hodnota bude používat nejčastěji.

```
create procedure reduce_quantity_toner
@id_toner int,
@quantity int = 1
as

update hardware.toner
set quantity = quantity-@quantity
where id_toner=@id_toner

go
```

Dotaz pro vyvolání procedury

```
execute reduce_quantity_toner
1
```

3.3.3 Triggery

Triggery jsou velmi důležíte pro udržování dat v databázi. Já jsem triggery vyřešil problematiku ukládaní historie. Ukládání nového řádku do tabulky historie po úpravě třeba jednoho sloupce tabulky v pc by bylo neefektivní, a proto jsem vytvořil následujíci trigger. Vytvoři jsem obdobné triggery i pro ostatní tabulky, u kterých chci sledovat historii. Trigger se spusti při úpravě tabulky hardware.pc. Vezme data, která se změní úpravou a uloží je do tabulky hardware.history. Podmínka COLUMNS_UPDATED()) > 0 zajišťuje, že aspoň jeden řádek byl upraven.

```
create trigger History_pc
on hardware.pc
for update as

IF (COLUMNS_UPDATED()) > 0
BEGIN
      INSERT INTO
      hardware.history(sn_pc,history_Id_user,history_id_status,history
_coment     ary,history_pool,history_date_actual)
      select

del.sn_pc,del.Id_user,del.id_status,del.comentary,del.pool,GETDATE()
      from deleted del
END
```

```
go
```

Následujícím triggerem jsem ošetřil změnu statusu, když administrátor prodlouží půjčku. Bez tohoto triggeru by administrátor při prodloužení pujčky musel upravovat i atribut ID_status. Trigger se spustí při úpravě v tabulce loan sloupce date_finish. Jednoduše porovná vložené datum s datem, které bylo v tabulce, a když je vkládané datum pozdější, upraví id_status na hodnotu 7, což odpovída hodnotě "extended"

```
create trigger extended
on loan
for update as

declare @date_finish_old date
declare @date_finish_new date
declare @id_loan int
IF UPDATE(date_finish)
BEGIN
      set @date_finish_old = (select del.date_finish from deleted del)
      set  @date_finish_new  =  (select  ins.date_finish  from  inserted
ins)
      set @id_loan = (select ins.ID_loan from inserted ins)
      if @date_finish_old < @date_finish_new
            begin
            update loan
            set id_status=7
            where ID_loan=@id_loan
            end
end
go
```

3.3.4 Kurzor

Funkcí tohoto kurzoru je, přepsání stavu půjčky na hodnotu „expire“, když půjčka propadla, tedy aktuální datum je novější než datum uložene v atributu date_finish. Kurzor jsem použil, protože dotazem select dostanu sadu výsledků a já potředuji každý výsledek zpracovat zvlašť. Dotaz select vybere všechny řádky z tabulky loan, které mají atribut date_finish mladší než aktuální datum. Výsledky dotazu zapisuji do proměnných @id_loan a @id_status. Podmínkou „if @id_status <> 5“ kontroluji, že půjčka již není zavřená, protože je logické, že zavřene půjčky nemůžou propadnout. Dále již jednoduše přiřadím atributu id_status hodnotu 9, což odpovídá statusu expire. Cyklus while obsahuje stejný kód a je zde zařazen, aby kurzor provedl změny pro každý řádek výsledku dotazu select a ne jen pro první řádek.

```
declare expire cursor  for
select id_loan, id_status from
loan
where date_finish<GETDATE()
declare @id_loan int
declare @id_status int

open expire


fetch next from expire into @id_loan, @id_status
if @id_status <> 5
begin
      update loan
      set id_status = 9
      where ID_loan=@id_loan

      while @@FETCH_STATUS=0
            begin
                  fetch next from expire into @id_loan, @id_status
                  if @id_status <> 5
                  begin
                        update loan
                        set id_status = 9
                        where ID_loan=@id_loan
                  end
            end
END
close expire
deallocate expire
go
```

Závěr

Cílem této práce bylo navrhnout a implementovat databázi, která bude podporovat evidenci hardwaru. Tato databáze měla vycházet z analýzy současného stavu a měla zohlednit všechny interní procesy ve firmě. Důležitým požadavkem bylo vytvoření historie záznamů, aby změny v databázi byly dohledatelné.

V analýze současného stavu jsem popsal všechny interní procesy, které v našem oddělení fungují. Na základě těchto procesů a dalších požadavků jsem ve vlastním návrhu vypracoval návrh databáze, který splňuje všechny tyto požadavky. Dle tohoto návrhu jsem databázi implementoval pomocí jazyka SQL. Dále jsem přidal spoustu procedur, spouští a kurzorů, aby manipulace s databází byla v budoucnu velmi jednoduchá. Tímto jsem splnil všechny vytyčené cíle své bakalářské práce.

Posledním krokem k reálnému používání této databáze je vytvoření informačního portálu, který bude s touto databází pracovat. Administrátoři budou moci upravovat, přidávat a mazat údaje v tabulkách. Uvidí různé statistické údaje např. počet kusů jednotlivého typu hardwaru, počet propadnutých zápůjček atd. Kolega z práce vytvoří program, který bude kontrolovat, jaký software má uživatel na počítači nainstalovaný, kdy byl naposledy v síti a jakou měl IP adresu atd. Tyto všechny údaje budou také obsahem informačního portálu.

Literatura

[1] BURKE, D. *How to use Lotus Notes 6*. 1. Vyd. Indianapolis: Que, 2003. 258 s. ISBN 0-7897-2796-X.

[2] CONOLLY, T., BEGG, C., HOLOWCZAK, R. *Mistrovství – Databáze : Profesionální pruvodce tvorbou efektivních databází*. Brno : Computer Press, 2009. 584 s. ISBN 978-80-251-2328-7.

[3] DOSEDĚL, T. *Počítačová bezpečnost a ochrana dat*. 1. vyd. Brno: Computer Press, 2004. 190s. ISBN 80-251-0106-1.

[4] HOTEK, M. *Microsoft SQL Server 2008: Krok za krokem*. 1.vyd. Brno: Computer Press, 2009. 488 s. ISBN 978-80-251-2466-6.

[5] KOCH, M., NEUWIRTH, B. *Datové a funkční modelování*. 1.vyd. Brno: Akademické nakladatelství CERM, 2008. 121 s. ISBM 978-80-214-373-9

[6] KONEČNÝ, M. *Zpráva z praxe - SWOT analýza IBM GS Delivery Center Brno.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2011. 23 s. Vedoucí práce Ing. Jiří Kříž Ph.D.

[7] LACKO, L. *Jak vyzrát na SQL Server 2008*. 1.vyd. Brno: Computer Press, 2009. 469 s. ISBN 978-80-251-2101-6.

[8] OPEL, A. *SQL bez předchozích znalostí*. 1. vyd. Brno: Computer press, 2008. 239 s. ISBN 978-80-251-1707-1.

Elektronické zdroje

[1] *Complete 2009 IBM Annual report* [online]. c2009, poslední revize 30.4.2012 [cit. 30.4.2012]. Dostupné z: ftp://public.dhe.ibm.com/annualreport/2010/2010_ibm_annual.pdf

[2] *History of IBM* [online]. c2008, poslední revize 30.4.2012 [cit. 30.4.2012]. Dostupné z: http://www-03.ibm.com/ibm/history/history/history_intro.html

[3] *History of Lotus Notes and Domino* [online]. c2011, poslední revize 30.4.2012 [cit. 30.4.2012]. Dostupné z: http://www.ibm.com/developerworks/lotus/library/ls-NDHistory/

[4] KUNC,P. *Bezpečnost v Lotus Notes* [online]. c2007, poslední revize 11.10.2010 [cit. 30.4.2012]. Dostupné z: http://petrkunc.net/lotus/1188342511-clanek-bezpecnost-v-lotus-notes.html

[5] *Lotus Domino documentation* [online]. c2011, poslední revize 30.4.2012 [cit. 30.4.2012]. Dostupné z: https://www.ibm.com/developerworks/lotus/documentation/domino/

[6] *Lotus Notes documentation* [online]. c2011, poslední revize 30.4.2012 [cit. 30.4.2012]. Dostupné z: http://www.ibm.com/developerworks/lotus/documentation/notes/

[7] *Low cost of ownership* [online]. c2010, poslední revize 30.4.2012 [cit. 30.4.2012]. Dostupné z: http://www-01.ibm.com/software/lotus/products/domino/lowtco.html

[8] SKŘIVAN, J. *Databáze a jazyk SQL*. [online]. 2000 [cit. 2011-1-2]. Dostupné z: http://interval.cz/clanky/databaze-a-jazyk-sql/.

Seznam obrázků a tabulek

Seznam obrázků

Seznam tabulek

Seznam příloh

Příloha 1 – Fyzický návrh databáze – zdrojový kód SQL

```
create database bp
go

use BP
go

create schema hardware
go
create schema people
go
create schema manufacturer
go

create table people.users (
Id_user int identity(1,1) primary key,
forname varchar (20),
surname varchar (20),
LNID_personal varchar(7) not null, -- Lotus Notes ID
LNID_manager varchar(7),
Department varchar(6)
)

create table people.administrator (
```

```
Id_administrator int identity (1,1) primary key,
forname varchar (15),
surname varchar (20),
LNId_personal varchar (7) not null
)

create table Manufacturer.identification (
Id_manufacturer int identity(1,1) primary key,
name varchar (20) not null
)
```

```
create table Manufacturer.contact (
ID_manufacturer_contact int identity (1,1) primary key,
id_manufacturer int foreign key references
manufacturer.identification(id_manufacturer),
phone varchar (20),
email varchar (35)
)

create table status (
id_status int identity(1,1) primary key,
name varchar (9)

)



create table orders (
id_order int identity (1,1) primary key,
id_administrator int foreign key references
people.administrator(Id_administrator),
id_manufacturer int foreign key references manufacturer.identification
(id_manufacturer),
po varchar (15),
order_object varchar (20) not null,
quantity int,
comentary varchar (50),
date_insert date not null,
date_delivery date
)

create table family (
id_family int identity(1,1) primary key,
family varchar (10),
type varchar (4),
model varchar (4)
)

create table hardware.server (
sn_servers varchar (10) primary key,
id_manufacturer int foreign key references
manufacturer.identification(id_manufacturer),
id_family int foreign key references family (id_family),
id_status int foreign key references status(id_status),
corporate_name varchar (20),
date_insert date not null,
date_last_change date not null ,
comentary varchar (75)
)

create table hardware.pc (
sn_pc varchar (10) primary key,
Id_user int foreign key references people.users(id_user),
id_manufacturer int foreign key references
manufacturer.identification(id_manufacturer),
id_family int foreign key references family (id_family),
id_status int foreign key references status(id_status),
```

```
pool varchar (6),
date_last_change date not null ,
date_insert date not null,
comentary varchar (75)
)


create table hardware.printer (
sn_printer varchar (10) primary key,
id_manufacturer int foreign key references
manufacturer.identification(id_manufacturer),
id_family int foreign key references family (id_family),
id_status int foreign key references status(id_status),
room varchar (5) not null,
printer_name varchar (10) not null,
date_insert date not null,
comentary varchar (75),
date_last_change date not null
)

create table hardware.mobile (
sn_mobile varchar (15) primary key,
Id_user int foreign key references people.users(id_user),
id_manufacturer int foreign key references
manufacturer.identification(id_manufacturer),
id_family int foreign key references family (id_family),
id_status int foreign key references status(id_status),
date_insert date not null,
date_last_change date not null ,
comentary varchar (75)
)

create table hardware.sim_card (
number varchar (9) primary key,
Id_user int foreign key references people.users(id_user),
id_manufacturer int foreign key references
manufacturer.identification(id_manufacturer),
id_status int foreign key references status(id_status),
date_insert date not null,
date_last_change date not null ,
comentary varchar (75)
)

create table hardware.history (
id_history int identity(1,1) primary key,
sn_pc varchar (10) foreign key references hardware.pc(sn_pc) ,
sn_mobile varchar (15) foreign key references
hardware.mobile(sn_mobile) ,
number varchar(9) foreign key references hardware.sim_card(number),
history_id_user int,
history_Administrator sysname DEFAULT SUSER_SNAME(),
history_id_status int,
history_date_actual date not null,
history_comentary varchar (75),
history_room varchar (5) ,
history_pool varchar (6),
history_corporate_name varchar (20)
)
```

```
create table hardware.toner (
id_toner  int identity(1,1) primary key,
id_manufacturer int foreign key references
manufacturer.identification(id_manufacturer),
id_family int foreign key references family (id_family),
quantity int
)


create table loan (
ID_loan int identity (1,1) primary key,
id_user int foreign key references people.users (id_user),
id_administrator int foreign key references
people.administrator(id_administrator),
SN_mobile varchar (15) foreign key references
hardware.mobile(sn_mobile),
Sn_pc varchar (10) foreign key references hardware.pc(sn_pc),
id_status int foreign key references status(id_status),
id_device tinyint not null,
date_finish date not null,
date_start date not null,
comentary varchar (50)
)

go

create view toner as
select t.quantity, fa.family, fa.model, ma.name
from hardware.toner t
      join family fa
on (fa.id_family=t.id_family)
      join manufacturer.identification ma
on (ma.id_manufacturer = t.id_manufacturer)
go

create view owner_workstation as
select pc.sn_pc, pc.pool, s.name as status, f.family, pe.forname,
pe.surname, pc.date_last_change, pc.date_insert, f.model,
f.type,ma.name
from hardware.pc pc
      join status s
on pc.id_status = s.id_status
      join family f
on    pc.id_family = f.id_family
      left outer join people.users pe
on pc.Id_user=pe.Id_user
      join manufacturer.identification ma
on ma.Id_manufacturer = pc.id_manufacturer

go

create view owner_mobiles as
select mo.sn_mobile, s.name as status, f.family, pe.forname,
pe.surname, mo.date_last_change, mo.date_insert,ma.name
from hardware.mobile mo
      join status s

on mo.id_status = s.id_status
      join family f
```

```
on    mo.id_family = f.id_family
      left outer join people.users pe
on mo.Id_user=pe.Id_user
      join manufacturer.identification ma
on ma.Id_manufacturer = mo.id_manufacturer
go

create view owner_servers as
select se.sn_servers , se.corporate_name, s.name as status,
se.date_last_change, se.date_insert, f.family, f.type,f.model,ma.name
from  hardware.server se
      join status s
on se.id_status = s.id_status
      join family f
on    se.id_family = f.id_family
      join manufacturer.identification ma
on ma.Id_manufacturer = se.id_manufacturer
go

create view owner_printers as
select pr.sn_printer,pr.printer_name,pr.room,ma.name as manufacturer,
s.name as status, f.family,pr.date_last_change, pr.date_insert,ma.name
from hardware.printer pr
      join status s
on pr.id_status = s.id_status
      join family f
on    pr.id_family = f.id_family
      join manufacturer.identification ma
on ma.Id_manufacturer = pr.id_manufacturer
go

create view view_loan as
select s.name as Status,pe.forname as Users_Forname, pe.surname as
Users_Surename,pe.lnid_personal,pa.forname as Admnistrator_Forname,
pa.surname as
Administrator_Surname,pc.sn_pc,mo.sn_mobile,lo.date_start,lo.date_fini
sh,lo.comentary
from loan lo
      join people.users pe
on lo.id_user = pe.Id_user
      join people.administrator pa
on pa.Id_administrator = lo.id_administrator
      left outer join hardware.pc pc
on pc.sn_pc = lo.Sn_pc
      left outer join hardware.mobile mo
on mo.sn_mobile = lo.SN_mobile
      join status s
on s.id_status = lo.id_status

go

create view owner_sim_card as
select si.number, s.name as status, pe.forname, pe.surname,
si.date_last_change, si.date_insert,ma.name
from hardware.sim_card si
      join status s
on si.id_status = s.id_status
      left outer join people.users pe
on si.Id_user=pe.Id_user
```

```
      join manufacturer.identification ma
on ma.Id_manufacturer = si.id_manufacturer
go

create view view_orders as
select o.PO, o.order_object, o.quantity, o.comentary, o.date_delivery,
o.date_insert, ma.name, pe.forname,pe.surname
from  orders o
      join manufacturer.identification ma
on o.id_manufacturer = ma.Id_manufacturer
      join people.administrator pe
on pe.Id_administrator = o.id_administrator
go


create trigger History_mobile
on hardware.mobile
for update as

IF (COLUMNS_UPDATED()) > 0
BEGIN
      INSERT INTO
hardware.history(sn_mobile,history_Id_user,history_id_status,history_c
omentary,history_date_actual)
      select
      del.sn_mobile,del.Id_user,del.id_status,del.comentary,GETDATE()
      from deleted del
end
go

create trigger History_pc
on hardware.pc
for update as

IF (COLUMNS_UPDATED()) > 0
BEGIN
      INSERT INTO
hardware.history(sn_pc,history_Id_user,history_id_status,history_comen
tary,history_pool,history_date_actual)
      select

del.sn_pc,del.Id_user,del.id_status,del.comentary,del.pool,GETDATE()
      from deleted del
end
go

create trigger History_simcard
on hardware.sim_card
for update as

IF (COLUMNS_UPDATED()) > 0
BEGIN
      INSERT INTO
hardware.history(number,history_Id_user,history_id_status,history_come
ntary,history_date_actual)
      select
      del.number,del.Id_user,del.id_status,del.comentary,GETDATE()
      from deleted del
end
```

```
go

create view history_pc
as
select hi.sn_pc,pe.forname,hi.history_pool,
pe.surname,hi.history_administrator, s.name as status,
hi.history_comentary
from hardware.history hi
      right outer join people.users pe
on (pe.id_user = hi.history_id_user)
      right outer join hardware.pc pc
on (pc.sn_pc = hi.sn_pc)
      join status s
on (s.id_status = hi.history_id_status)
go

create view history_mobile
as
select hi.sn_mobile,pe.forname, pe.surname,hi.history_administrator,
s.name as status, hi.history_comentary
from hardware.history hi
      right outer join people.users pe
on (pe.id_user = hi.history_id_user)
      right outer join hardware.mobile mo
on (mo.sn_mobile = hi.sn_mobile)
      join status s
on (s.id_status = hi.history_id_status)
go

create view history_simcard
as
select hi.number,pe.forname, pe.surname,hi.history_administrator,
s.name as status, hi.history_comentary
from hardware.history hi
      right outer join people.users pe
on (pe.id_user = hi.history_id_user)
      right outer join hardware.sim_card si
on (si.number = hi.number)
      join status s
on (s.id_status = hi.history_id_status)
go

create procedure find_history
@sn varchar (15),
@id_device int

as

if @id_device = 1
select * from history_mobile
where sn_mobile = @sn

if @id_device = 2
select * from history_pc
where sn_pc = @sn

if @id_device = 3
select * from history_simcard
where number = @sn
```

```
go

declare expire cursor for
select id_loan, id_status from
loan
where date_finish<GETDATE()
declare @id_loan int
declare @id_status int

open expire


fetch next from expire into @id_loan, @id_status
if @id_status <> 5
begin
      update loan
      set id_status = 9
      where ID_loan=@id_loan

      while @@FETCH_STATUS=0
            begin
                  fetch next from expire into @id_loan, @id_status
                  if @id_status <> 5
                  begin
                        update loan
                        set id_status = 9
                        where ID_loan=@id_loan
                  end
            end
END
close expire
deallocate expire
go


create trigger extended
on loan
for update as

declare @date_finish_old date
declare @date_finish_new date
declare @id_loan int
IF UPDATE(date_finish)
BEGIN
      set @date_finish_old = (select del.date_finish from deleted del)
      set @date_finish_new = (select ins.date_finish from inserted
ins)
      set @id_loan = (select ins.ID_loan from inserted ins)
      if @date_finish_old < @date_finish_new
            begin
            update loan
            set id_status=7
            where ID_loan=@id_loan
            end
end
go

create procedure reduce_quantity_toner
```

```
@id_toner int,
@quantity int = 1
as

update hardware.toner
set quantity = quantity-@quantity
where id_toner=@id_toner

go
```