Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Zabezpečení databázově koncipovaných informačních systémů

Tomáš Tintěra

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Tomáš Tintěra

Informatika

Název práce

Zabezpečení databázově koncipovaných IS

Název anglicky

Security of database designed Information systems

Cíle práce

Bakalářská práce je zaměřena na problematiku zabezpečení databázově koncipovaných informačních systémů. Hlavním cílem této práce je:
a) objasnit teoretické principy relačně databázové technologie v kontextu s problematikou zabezpečení databázových evidencí,
b) zmapovat momentální stav této problematiky a vymezit její relevantnost včetně požadavků na ni kladených,
c) navrhnout možnosti přijatelných řešení těchto požadavků,
d) ověřit funkčnost navržených záležitostí,
e) ověřené záležitosti zobecnit pro další možná uplatnění.

Metodika

Použitá metodika zadané bakalářské práce bude založena na studiu a analýze dostupných informačních zdrojů, zkušeností jejího autora a existujících řešení v dané oblasti. Stěžejní pro vypracování této závěrečné práce budou metody a techniky relačně databázové technologie v kontextu s touto problematikou. Navrhované řešení bude zohledňovat identifikované požadavky a očekávání spojená s řešenou záležitostí. Na podkladě syntézy teoretických poznatků a dosažených výsledků budou formulovány závěry této bakalářské práce a následně zobecněny pro další možná použití.

Závazný harmonogram:

Teoretické principy řešené problematiky, literární rešerše – předmět 1. zápočtu z BP: do 5.9.2016

Zmapování momentální situace řešené problematiky, identifikace požadavků s tím spojených: do 15.11.2016

Navržení možného řešení a jeho následné ověření formou praktického řešení – předmět 2. zápočtu z BP: do 28.1.2017

Zobecnění navržených záležitostí pro další možná použití – předmět 3. zápočtu z BP: do 15.3.2017

Doporučený rozsah práce

45-55

Klíčová slova

Relačně db technologie, datová integrita, db pohledy, SQL, práva přístupu

---

Doporučené zdroje informací

AFYOUNI, H. A. Database Security and Auditing: Protecting Data Integrity and Accessibility. Boston, Massachusetts: Thomson Course Technology, 2006. ISBN 978-0619215590.

BASTA, A., ZGOLA, M. Database Security. Boston, Massachusetts: Course Technology, 2011. ISBN 978-1435453906.

NATAN, Ron Ben. Implementing Database Security and Auditing. Newton, Massachusetts: Digital Press, 2005. ISBN 978-1-55558-334-7.

POKORNÝ, J., VALENTA, M. Databázové systémy. Praha: ČVUT, 2013. ISBN 978-80-213-1191-6.

SHARMA N., PERNIU L., CHONG R. F., IYER A., NANDAN Ch., MITEA A. a NONVINKERE M. a DANUBIANU M. Database fundamentals. IBM, 2010. Dostupné z: http://public.dhe.ibm.com/software/dw/db2/express-c/wiki/Database_fundamentals.pdf

VOSTROVSKÝ, V. Vytváření databází v Oracle. Praha: Česká zemědělská univerzita v Praze, 2009. ISBN 978-80-213-1191-6.

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

doc. Dr. Ing. Václav Vostrovský

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 21. 2. 2017

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 21. 2. 2017

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 28. 02. 2017

Čestné prohlášení

Prohlašuji, že jsem svou bakalářskou práci „Zabezpečení databázově koncipovaných informačních systémů“ vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na jejím konci. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2017 _____

Poděkování

Rád bych poděkoval doc. Ing. Václavu Vostrovskému, Ph.D. za vedení mé bakalářské práce a za veškeré odborné rady, které mi velmi pomohly. Také bych rád poděkoval Mgr. Tomáši Wehlemu za jazykovou korekturu práce a věcné připomínky týkající se její formy. Poděkování patří i mé matce Romaně Tintěrové za finální kontrolu práce.

Zabezpečení databázově koncipovaných informačních systémů

Souhrn

Tato bakalářská práce se zabývá problematikou zabezpečení databázově koncipovaných systémů, tj. základu databáze, nad kterou se implementuje softwarové řešení. Bakalářská práce se skládá ze dvou částí – teoretické a praktické. Hlavním cílem teoretické části je objasnit teoretické principy relační databázové technologie v kontextu s problematikou zabezpečení. Na zabezpečení jako takové je v poslední době kladen velký důraz. Z toho důvodu si práce klade za cíl zmapovat aktuální stav této oblasti a na příkladu centrálního registru vozidel demonstrovat její relevantnost.

Hlavním cílem praktické části je navrhnout modelovou databázi v souladu s vytyčenými teoretickými východisky a následné ověření její funkčnosti pomocí databázového prostředku Oracle Database 11g. Přínosem této práce by pak mělo být dokázání nezbytnosti bezpečných databází, na kterých jsou založeny dnešní informační systémy. Zároveň je z práce možné čerpat postupy, jak takové databáze vytvářet.

Klíčová slova: SQL, datová integrita, db pohledy, transakční zpracování, práva přístupu, databázové triggery, bezpečnost databáze, šifrování, auditabilita, centrální registr vozidel, datová normalizace.

Security of database designed information systems

Summary

This bachelor thesis deals with security of database designed information systems. That basically means a raw database without software interface, which is usually implemented over raw database. Bachelor thesis is divided into two main sections. The main goal of the first part is to clarify the theoretical principles of relational database technology in the context of database security. Recently, a security is a very important theme. Therefore, another goal of this thesis is to map the current condition of this issue and to prove its relevance. This goal could be presented on Central Register of Vehicles.

The main goal of the second part is to create model database in compliance with set up theoretical principles and to verify functionality of created database on Oracle Database 11g. Contribution of this bachelor thesis should be proving the necessity of secure databases, which the current information systems are based on. At the same time, it is possible to draw inspiration from this bachelor thesis to create such databases.

Keywords: SQL, data integrity, database views, transactions, access rights, database triggers, database security, Central Register of Vehicles, audiability, database normalization, encryption.

Obsah

Seznam obrázků

1 Úvod

Technologický vývoj je stále k nezastavení. V 19. století vynalezl Charles Babbage první předchůdce dnešních počítačů. Ve 20. století se počítače opět zdokonalovaly. Od velkých sálových počítačů se pokročilo k osobním počítačům, na kterých se lidé stali do jisté míry závislými. Dříve se data uchovávala ve velkých skladech, mnozí jistě pamatují kartotéky, které byly v nemocnicích. Tyto kartotéky ještě úplně nevymizely, ovšem lze předpokládat, že v průběhu několika let k tomu dojde. S již avizovaným příchodem osobních počítačů a hlavně internetu bylo nutné data uchovávat elektronicky. S tím však souvisí hlavní problém celé věci. Je totiž třeba řešit zabezpečení elektronických dat. Jelikož jsou data relativně volně k dispozici on-line, je třeba dávat velký pozor, kdo k nim přistupuje a za jakých podmínek. Tato bakalářská práce se zabývá právě touto aktuální problematikou.

V současnosti existuje jen minimum firem, které by se neorientovaly na internet jako na místo relativně výhodného obchodu. Data, která firma střeží, jsou volně dostupná na síti, ať už pro práci zaměstnanců, či práci z domova. Je však stěžejní úlohou firmy tato data co nejlépe zabezpečit před případnými útoky. Rozvoj internetu totiž nepřináší pouze výhody, ale i obrovská rizika. Hlavním rizikem, které v této bakalářské práci bude rozebíráno, je zabezpečení databáze, aby nedocházelo ke krádežím dat. Této problematice se dostane pozornosti hlavně v teoretické části, kde bude nejdříve rozebrána datová integrita jakožto důležitá vlastnost při zakládání nové databáze. Dále bude věnována pozornost nastavení přístupových práv. Je třeba si uvědomit, že k útokům na databázi nemusí docházet nutně zvenku, ale také zevnitř. A to díky neopatrnosti, nebo i možnému úmyslu. Je tedy nutné být ostražitý, jaký člověk může nakládat s jakými daty. Neméně důležitým pojmem jsou databázové pohledy, které můžeme charakterizovat jakožto další způsob zabezpečení. Díky pohledům je možné záměrně zakrýt některá data z originálních tabulek.

Nebezpečným faktorem pro databázové systémy se můžou stát databázové triggery, které fungují sami jako automatické spouště. Nikdo není neomylný, a pokud se hovoří o obrovské firmě, je poměrně jednoduché některé údaje splést nebo zapomenout. Rovněž tomu tak je s databázovými triggery. Je dosti jednoduché zapomenout na to, že v systému

existují. Takové ústřižky kódu představují nebezpečí pro celou databázi. Poslední kapitoly teoretické části jsou pak věnovány transakčnímu zpracování a bezpečnosti databáze. Přičemž na problematiku bezpečnosti databáze by měl být ve firmách kladen velký důraz. Kapitola se věnuje popisu několika možných scénářů narušení databázové bezpečnosti. Popisuje také, jak by se měla data v databázi šifrovat.

Bakalářská práce se rovněž věnuje aktuální a velmi důležité problematice centrálního registru vozidel. Cílem práce není tuto oblast kritizovat, ale pokusit se objektivně a konstruktivně nalézt důvody, které se za problémy s registrem skrývají. V tomto případě bylo využito papírových formulářů jakožto zdrojů k čerpání informací. Po provedení jejich analýzy bylo možno odhadnout, jakou podobu by mohl případný centrální registr (velice zjednodušený) mít.

Samotná praktická část se věnuje problematice vlastního řešení a s tím souvisejícího vytvoření modelové databáze, na které jsou vysvětleny postupy, které byly popsány v teoretickém východisku. Praktická část zároveň často obsahuje kusy SQL kódu, díky kterým je předvedeno, jak by vše fungovalo, pokud by se databáze skutečně implementovala.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce se zabývá problematikou zabezpečení databází a má několik cílů. Hlavním cílem předkládané bakalářské práce je vymezení relevantnosti problematiky zabezpečení databázově koncipovaných informačních systémů v praxi, včetně identifikace možných řešení těchto záležitostí. Dílčím cílem této práce je kriticky a objektivně zhodnotit stav nového centrálního registru vozidel, se kterým jsou velké problémy už od jeho spuštění. Cílem praktické části je pak vytvoření vlastní modelové databáze, na níž budou znázorněna bezpečnostní rizika, která byla popsána v teoretické části.

2.2 Metodika

Metodika této bakalářské práce je založena na shromáždění relevantních informačních zdrojů, které jsou v dané oblasti k dispozici, jejich následné podrobné studium či analýza a jejich kritické zhodnocení ve smyslu relevantnosti k danému tématu. Bakalářská práce je založena hlavně na informačních zdrojích publikací, které v této problematice již vyšly. Zároveň je však čerpáno z elektronických dokumentací firmy Oracle Corporation a z českých zpravodajských portálů. Poslední zdrojem jsou vyplňovací formuláře, které jsou volně k dostání na úřadech s možností registrace nového vozidla. Na základě analýzy vyplňovacích formulářů a studia zpravodajských portálů je vytvořena kapitola zabývající se problematikou centrálního registru vozidel. Praktická část je vytvořena na základě zjištěných poznatků, které jsou popsány v teoretické části. Dále bude využito několika relačně databázových metod a technik, jako jsou datová integrita, datová normalizace, modelování datové základny. Na základě teoretické a praktické části je stanoven závěr, který lze zobecnit pro další možná využití.

3 Teoretická východiska

3.1 Elementární pojmy

Nejprve je nutné vysvětlit několik základních pojmů, které budou těžištěm vlastního řešení. Jedná se o následující pojmy:

- Primární klíč (primary key) – jednoznačně a nezaměnitelně identifikuje každý záznam v tabulce. V každé tabulce by měl být obsažen.[13]
- Cizí klíč (foreign key) – určuje jednoznačné identifikace řádků logicky nadřazené tabulky do logicky podřízené tabulky.[13]
- Atribut – jednotlivý sloupec v tabulce (jméno, ročník, číslo ID).[1]
- Záznam – jednotlivý řádek v tabulce (např. určití uživatelé).[1]
- Relační model – způsob uložení dat v databázi.
- Relační databáze – databáze založená na relačním modelu, často je tak označovaná samotná tabulka.[4]
- NULL – neurčená hodnota, prázdná.
- PL/SQL – procedurální jazyk nad SQL, připomíná PASCAL, vytváření a ukládání bloků kódu.[2]

3.2 Datová integrita

Pojem integrita databáze definuje, že data v ní uložená jsou korektní vůči zavedeným pravidlům. Jsou to mechanismy, které omezují uživatele zadávat nekorektní data. Díky integritě lze zadávat pouze data, která jsou vůči navrženým pravidlům korektní. Pravidla, která toto vymezují, se nazývají integritní omezení a de facto slouží k zajištění integrity databáze, což je velice důležité pro její korektní chod. Rozlišují se tři typy integritních omezení.[3]

3.2.1 Entitní integrita

Tento typ integrity zajišťuje jednoznačnou identifikaci ve všech řádcích. Z toho důvodu by měla mít každá tabulka primární klíč, není to však nezbytná nutnost. Například u spojovacích tabulek není primární klíč žádán. Díky tomu nedochází k nechtěné duplicitě dat. Primárním klíčem se často označuje např. číslo ID nebo číslo občana, které mu bylo

přiděleno. Pokud by entitní integrita neexistovala, docházelo by k tomu, že by mohli mít dva lidé stejné číslo, což je neakceptovatelné.[3] Entitní integrita se vytváří při zakládání tabulky, a to následujícím příkazem:

CREATE TABLE jmeno_tabulky (ID_nazev **CHAR(5) NOT NULL PRIMARY KEY**, první_atribut **VARCHAR (20)**,…);

Primární klíč bude nastaven na nenulovou hodnotu. Díky tomu nelze vkládat NULL hodnotu. Dokonce primární klíč sám o sobě vyžaduje nenulové hodnoty.[4]

3.2.2 Doménová integrita

Jedná se o množinu všech přístupných hodnot v rámci daného atributu. Z důvodu nutnosti vznikly tzv. domény. Pro úplnost by zde měly být zmíněny základní domény v Oracle databázi.

- **CHAR** – povoluje ukládání do atributu textové řetězce až do délky 2000 znaků. Problém nastává, kdy není zřejmé, jak dlouhý řetězec bude. CHAR si alokuje v paměti všechna místa, i pro nevyužité znaky. Z tohoto důvodu je lepší pro delší slova používat VARCHAR.[1]
- **VARCHAR** – jedná se o stejný typ jako CHAR, s tím rozdílem, že je možné ho alokovat až do délky 4000 znaků. Výhodou oproti CHARU je, že pracuje s dynamickým alokováním (je-li definován VARCHAR(20), ale naplní se pouze 10 míst, ostatní místa se „zahodí“ a nezabírají paměť).[1]
- **NUMBER** - povoluje vkládání čísel až do délky 38 míst, včetně desetinných míst.[1]
- **DATE** – specifický typ, slouží k ukládání data, tedy časových hodnot.

Domény lze měnit pomocí příkazů ALTER, který umožňuje upravit strukturu relační tabulky. Tomu odpovídající příkaz by byl následující:[1]

ALTER COLUMN ciselny_atribut **NUMBER(6)**;

Doménová integrita se vytváří rovněž při zakládání tabulky. Zde by se měl vymezit interval hodnot, který je žádaný. Doménová integrita je definována následujícím způsobem:[4]

CREATE TABLE jmeno_tabulky (…,ciselny_atribut **NUMBER(5) CHECK** (ciselny_atribut **BETWEEN** 15000 **AND** 75000));

Takto ošetřená tabulka nepovoluje naplnění atributu mzda méně než 15000 a více než 75000. Pokud toto omezení není zadáno při vytváření tabulky, je nemožné ho doplnit příkazem ALTER.

3.2.3 Referenční integrita

Referenční integrita zajišťuje korektnost vazeb mezi více tabulkami, které spolu logicky souvisí. Tohoto dosahuje pomocí cizího klíče, který se rovněž definuje při zakládání tabulky a není možné ho později doplnit či upravit pomocí příkazu ALTER TABLE. Je tedy nutné, aby hodnota cizího klíče, jestliže v podřízené tabulce nějaký existuje, byla naplněna existujícím záznamem nadřazené tabulky nebo zůstal nevyplněný.[3] Referenční integrita je definována při vytváření tabulky, a to následujícím způsobem:

CREATE TABLE vedlejsi_tabulka (ID_primarni **CHAR(5) NOT NULL PRIMARY KEY**, první_atribut **VARCHAR(20)**, ID_druheTabulky **CHAR(5)**, **FOREIGN KEY** (ID_druheTabulky) **REFERENCES** hlavni_tabulka(ID_ druheTabulky));

Takto propojené tabulky se označují jako podřízená (slave) a nadřízená (master) tabulka. Podřízená tabulka je ta, ve které je použito pravidlo cizího klíče, nadřazená pak ta, na kterou se odkazuje. V tomto případě je podřízená tabulka vedlejsi_tabulka.[1]

Co to je referenční integrita, by mělo být nyní zřejmé, ale je třeba uvést, ve kterých případech může vyvolat problémy. Například při vkládání pomocí příkazu INSERT či změně údajů. Pokud se hodnota klíče nebude shodovat v podřízené tabulce s hodnotou v tabulce nadřízené, bude vyvolána chyba.

Druhá situace, která může nastat, je při mazání či úpravě v nadřazené tabulce. Kontroluje se, zda v podřízené tabulce není klíč se stejnou hodnotou jako v tabulce

nadřízené, aby nedošlo při mazání v nadřízené tabulce ke ztrátě údajů. Jako příklad lze uvést databázi mobilních předvoleb. Z tabulky předvoleb nelze odstranit záznam s konkrétní předvolbou, jestliže je používána alespoň jedním člověkem. Tedy referenční integrita nepovolí smazání záznamů, na které existuje odkaz. Referenční integritu lze udržet několika následujícími způsoby:

- **Set NULL** – hodnoty cizích klíčů se nahrazují za hodnotu NULL.[4] Tato možnost je volitelná pouze pokud se cizímu klíči při zakládání povolí hodnoty NULL. Pokud definujeme atributem NOT NULL, nebude tato možnost proveditelná.
- **Restriktivně** – odstranění záznamu v nadřazené tabulce není možné, pokud na něj existuje odkaz v tabulce podřízené.[4]
- **Kaskádovitě** – poměrně složitý způsob, při kterém je nutné dovést změny z nadřazené tabulky do všech tabulek podřízených. V případě zrušení záznamu je to potom smazání všech dalších záznamů, kde figuruje stejná hodnota cizího klíče.[4]

3.3 Nastavení práv a přístupu, vytváření rolí

Další mechanismus, pomocí nějž se zabezpečuje databáze, souvisí s nastavením práv a s určením, kdo bude přistupovat k čemu a jak. V této kapitole bude věnována pozornost také auditabilitě, jelikož se dotýká dané problematiky.

V dnešní době je nastavování práv zcela běžné. Je samozřejmé, že ten, kdo tabulku či informační systém vytvořil, má automaticky veškerá práva. Může tedy zcela logicky práva delegovat, a to podle toho, jak uzná za vhodné. Zde je možné se setkat s případným nechtěným útokem na informační systém. Může se stát, že administrátor nastaví nechtěně práva špatně, a tak zcela nepovolaná osoba může nabýt práv neakceptovatelných vůči jeho postavení. Tento uživatel je poté potenciálním rizikem. Nevědomky může způsobit nekonzistenci dat nebo skrz jeho účet může být veden útok záměrný. Je tedy zásadní dávat si pozor, jaká práva se delegují a komu.

V praxi se střetávají dva přístupy přidělování práv, přičemž oba mají svá pro i proti. Jeden přístup doporučuje přidělovat práva uživatelům či pracovníkům podle toho, jak jsou kompetentní. Je tedy zřejmé, že nováček ve firmě bude mít práva nulová. Potíže zde

nastávají, když je nucen řešit problém, na který nemá dostatečné oprávnění. Druhý přístup je opačný, a to že se uživateli přidělí velké množství práv a odebírají se mu podle prohřešků. Tento přístup je však dosti nebezpečný a může vést k chtěnému, ale i nechtěnému zničení informačního systému, jehož náprava může být dosti drahá jak časově, tak materiálně. Práva se delegují pomocí příkazu GRANT a odebírají se pomocí REVOKE, a to následujícím způsobem.[1]

```
GRANT {ALL | oprávnění} ON tabulka TO {PUBLIC | uživatel}
```

Klíčová slova jsou zde ALL a PUBLIC. Co se týče klausule ALL, delegují se všechna práva. Co se týče klausule PUBLIC, delegují se práva všem uživatelům v informačním systému.[1]

3.3.1 Vytvoření uživatelských účtů

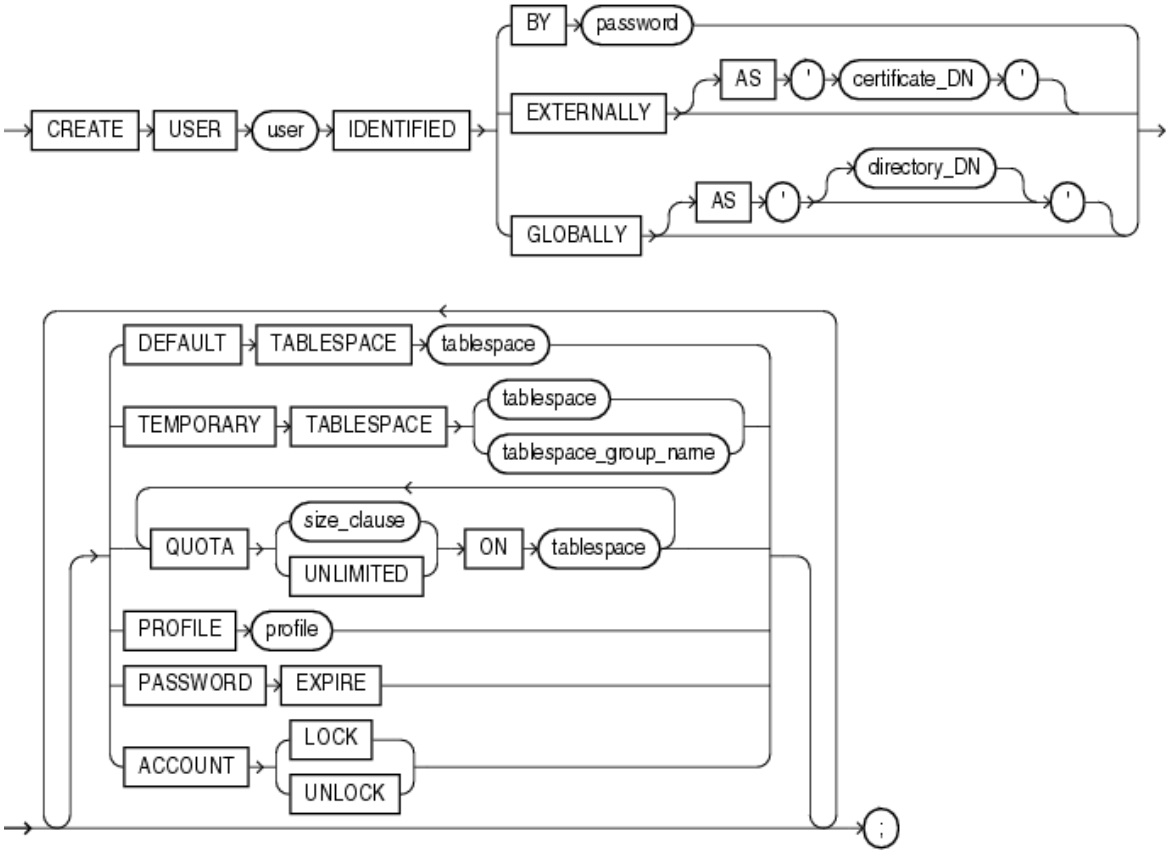
Než je možné delegovat práva, musí se vytvořit uživatelské účty, na které se potom můžou práva delegovat. Vytvářet účty je možné pomocí příkazu CREATE USER, a to různými způsoby (Obr. 1). Jeden z možných je tento:[1]

```
CREATE USER jmeno_uživatele IDENTIFIED BY heslo_uživatele;
```

Klausule IDENTIFIED určuje, jakým způsobem bude uživatel ověřován databázovým strojem.[1] Takto zavedený uživatel by však neměl žádné pravomoci, a tak je na místě mu přidělit nějaká práva, systémová nebo objektová. Rozhodnutí záleží na firmě, a to dle politiky, která je popsána výše. Takto zavedený uživatel by neměl právo se přihlásit, respektive systém ho odmítne. Proto je nutné umožnit mu to pomocí příkazu CREATE SESSION, a to takto:[2]

```
GRANT CREATE SESSION TO jmeno_uživatele;
```

Takto zavedený uživatel je nyní schopný se přihlásit. Dále je třeba řešit, aby mohl vytvářet tabulky, které mu budou pomáhat při práci. Toho je možné dosáhnout pomocí takto zvoleného příkazu:[1]



Obrázek 1 - Cesta vytváření uživatele [1]

GRANT CREATE TABLE TO jmeno_uživatele;

Nyní bude uživatel schopen díky systémovému oprávnění vytvářet tabulky. Tento přístup přidělování práv je však dosti nepraktický. Pro každého uživatele by se musela založit samostatně každá jedna pravomoc. V případě velkých korporací, které čítají tisíce zaměstnanců, by tento přístup byl naprosto neprofesionální. V praxi se tedy používají tzv. role (ty jsou popsané v podkapitole 3.3.4 Zavedení rolí). Následně se přiřadí určité skupině zaměstnanců určité role podle oprávnění či zásluh, záleží na zaměstnavateli.

Toto je jeden způsob vytvoření uživatele, v prostředí Oraclu je možné vytvářet uživatele více způsoby. Logická cesta příkazů je zobrazena na Obr. 1. Syntaxe vždy začíná

klausulí **CREATE USER** user_name **IDENTIFIED**, dále se však mohou dosti měnit. Pro úplnost je třeba vysvětlit některé pojmy, které schéma zachycuje.[1]

- **EXTERNALLY** – Databázovému prostředku definuje, že je třeba vytvořit tzv. externího uživatele. Takto vytvořený uživatel je pak identifikovaný v externí službě (myšleno operačním systému, softwaru třetích stran atd.). Databázový prostředek se poté táže těchto externích služeb, jaká má mít uživatel práva či jaký postoj má vůči takto vytvořenému uživateli zaujmout. Klausule certificate_DN je povinná pouze pro SSL ověřování.[1]
- **GLOBALLY** – Pomocí tohoto přídavku je možné vytvořit globálního uživatele. Takto vytvořený uživatel musí být autorizován službou Oracle Internet Directory. Klausule directory_DN určuje, zda bude mít uživatel privátní globální schéma, nebo sdílené globální schéma.[1]

3.3.2 Objektová oprávnění

Pokud není žádoucí přidělovat veškeré práva, což je nutné zřídkakdy, používají se specifická oprávnění. Pro úplnost je zde uveden seznam možných oprávnění a jejich význam.

- **EXECUTE** – uživateli je povoleno spouštět PL/SQL funkce či procedury.[1]
- **SELECT** – uživateli je povoleno pouze čtení.
- **UPDATE** – uživateli je povolena oprava dat v systému.
- **INSERT** – uživateli je povoleno vkládání dat do systému.
- **DELETE** – uživateli je povoleno mazání veškerého obsahu.[1]

Příkazu pro přidělení čtení z tabulky pro všechny uživatele by potom odpovídalo toto:

GRANT SELECT ON jmeno_tabulky **TO PUBLIC**;

3.3.3 Systémová oprávnění

Systémová oprávnění pomáhají měnit strukturu databáze. Jsou to oprávnění, která na rozdíl od objektového oprávnění pracují čistě se strukturou databáze. Systémová oprávnění jsou následující:[1]

- **CREATE** – uživateli je povoleno vytvářet specifické předprogramované typy.
- **ALTER** – uživateli je povoleno měnit strukturu těchto typů.
- **DROP** – uživateli je umožněno zahazovat či smazat tyto specifické typy.

Jak již bylo zmíněno, práva jdou také odstranit, respektive je možné někoho degradovat, pomocí již zmíněného REVOKE. Jelikož je syntaxe provedení téměř totožná se syntaxí GRANT, není potřeba ji zde uvádět. Místo klausule TO se použije klausule FROM. Odstranění práv, které je zde nadefinováno v minulém odstavci, by bylo uskutečněno pomocí následujícího příkazu:[4]

```
REVOKE SELECT ON jmeno_tabulky FROM PUBLIC;
```

3.3.4 Zavedení rolí

Pro větší skupinu uživatelů lze vytvořit tzv. role. Jedná se o šablony pravomocí pro specifické či běžné skupiny. Například lze založit roli s názvem administrátor a této roli přidělit veškerá práva. Poté stačí určitému uživateli delegovat roli administrátor, která již obsahuje veškeré potřebné pravomoci. Role je výhodné vytvářet v systémech, v nichž jsou časté poměrně velké skupiny uživatelů, kterým budou přidělovány stejné pravomoci. Tomu by odpovídaly následující příkazy:[4]

```
CREATE ROLE nova_role;
GRANT SELECT, INSERT ON jmeno_tabulky TO nova_role;
GRANT nova_role TO nazev_uzivatele;
```

K jednotlivým rolím lze také přidat heslo, a to s použitím příkazu ALTER ROLE v klausuli IDENTIFIED, rovněž jako u uživatelských účtů. Roli je samozřejmě možné opět odheslovat, a to s pomocí klausule NOT IDENTIFIED. V praxi se používání hesel u rolí příliš nepoužívá, ze zřejmých důvodů by to bylo zdlouhavé a zpomalující práci. V praxi se hesla používají

jen u důležitých rolí (většinou jde o administrátorské role), které jsou nesmírně důležité pro chod databáze či systému.[4]

3.3.5 Auditabilita

Ve větších databázích vzniká nutnost vést si záznamy o tom, kdo a jak přistupoval k záznamům. To jest sledovat, které záznamy měnil nebo mazal. Důležitost auditability je zřejmá. Co zřejmé být nemusí, je to, kde se auditabilita hojně používá. Často je tento pojem spojovaný hlavně s internetovým bankovnictvím. Bankovní společnosti chtějí mít logicky přehledný rejstřík toho, kdo a jakým způsobem do databáze přistupoval. Při zpětné kontrole tímto způsobem mohou odhalit nežádoucí vstup do systému nepovolanou osobou. Nicméně to neznamená, že v jiných odvětvích se auditabilitě nedostává kýžené pozornosti, ba naopak. Dnes je zcela zásadní vědět, kdo s databází pracoval.[5]

3.4 Problematika pohledů

V databázovém prostředí existují tzv. pohledy (views). Jedno pravidlo (pravidlo pohledů) z 12 pravidel E. F. Codda pro relační model, definuje, že systém řízení báze dat musí poskytovat možnost práce s pohledy do databáze, včetně aktualizace obsažených dat.[16]

Pohledy umožňují nahlížet na konkrétní tabulky. Výhodou je, že z pohledu správce databáze je možné určit, co pohled bude z dané tabulky zobrazovat a co nikoliv. Pokud je to bráno z hlediska ochrany dat na databázi v nemocnici, díky pohledům není umožněno zobrazování rodného čísla v tabulce lékařů. Uživatel, kterému bude předán pohled z tabulky lékařů, neuvidí rodná čísla, což je bezesporu kýžený výsledek. Nebo je také umožněno modifikovat pouze vybrané atributy. Zároveň pohled není přímý přístup k tabulce, je možné si ji představit jako virtuální tabulku, která si bere údaje z fyzické. Výhodou také je, že díky pohledům je práce s daty přehlednější, často přehlednější je také sestavování dotazů. Rovněž pohledy na rozdíl od fyzické tabulky nezabírají tak velké množství paměti, protože logicky přímo neobsahují data, pouze odkazy, jak již bylo uvedeno.[2]

Podle toho, nad kolika tabulkami se vytváří pohledy, jsou rozděleny do dvou skupin:[13]

- **Jednoduché pohledy** – jsou vytvářeny nad jednou tabulkou.
- **Složité či komplexní pohledy** – jsou vytvářeny z údajů více tabulek.

Pokud se jedná o pohledy, které si berou data z více tabulek, je nutné tyto tabulky pomocí příkazu JOIN logicky provázat klíči. Složité či komplexní pohledy jsou stěžejním tématem této podkapitoly. Většinou nemá až tak velký smysl dělat pohled nad jednou tabulkou, daleko více se v praxi používá pohled nad více tabulkami. Nemusí se řešit stejné připojování tabulek příkazem JOIN, stačí čerpat údaje z pohledu, který již všechny tabulky spojil. Přesto všechno je umožněno stále používat standardní operace jako SELECT, UPDATE atd. Důvodem je databázový prostředek, který zajišťuje správnou manipulaci mezi fyzickými (skutečnými) tabulkami. Pohledy se vytváří pomocí příkazu CREATE VIEW. Pokud by se tedy vytvářel pohled tabulky se jménem atributu a ID_číslem, bez rodných čísel, odpovídal by tomu následující příkaz:[1]

```
CREATE VIEW jmeno_pohledu AS SELECT ID_cislo, jmeno_atributu FROM jmeno_tabulky;
```

Z hlediska práv přístupu je to stejné jako u tabulek. Uživatelský účet musí mít objektové oprávnění na čtení (SELECT), aktualizování (UPDATE) či případné vkládání (INSERT) záznamu. Pokud je možné pohled modifikovat, jedná se o pohled s možností aktualizace, který promítá změny do fyzické tabulky. Stejně jako u tabulek je možné pohledy upravovat pomocí ALTER VIEW či mazat pomocí DROP VIEW, a to následovně.[4]

```
DROP VIEW jmeno_pohledu;
```

Skrz pohledy je možné se na složitou databázi dívat jako na zdánlivě jednoduchou, pohledy toto totiž skryjí. Rovněž tak lze docílit přehlednějšího přístupu k některým datům. Pohledy mají bohužel také nevýhody. Získávání výsledků z fyzické tabulky může být někdy časově zdlouhavé, a to z důvodu komplikovanějších pohledů, kde jsou výpočty daleko náročnější. Tento problém se řeší v databázích pomocí tzv. materializovaných pohledů a vytváří se přidáním klausule MATERIALIZED za příkaz CREATE. Materializované pohledy jsou něco mezi tabulkou a pohledem a je možné je povahou přirovnat

k vyrovnávací paměti. Nespornou výhodou je, že jsou rychlejší než klasické pohledy; nevýhodou je, že dochází k duplicitě dat.[4]

3.5 Databázové triggery

Pokud se jedná o procedurální realizaci v SQL, jedná se o databázové triggery (database triggers). Triggery jsou také mnohdy nazývané automatické spouště. Jelikož se český ekvivalent málokdy používá, bude se v této práci používat anglický název. Triggery se vyvolávají povětšinou při každém DML příkazu INSERT, UPDATE či DELETE.[8] Jedná se o soubor příkazů, který je uložen na serveru společně s databází. Tyto příkazy jsou pak volány a mění dosavadní informace v tabulkách, podle specifické akce. Jedná se o takový zákon akce a reakce. Jde o velice užitečný nástroj a většinou se jedná o bloky PL/SQL.[1]

Podle toho, v jaké chvíli se triggery volají, se rozlišují tyto tři případy využití. První je volání triggeru před operací neboli BEFORE. Trigger, který se volá po operaci, je AFTER trigger. Poslední je trigger, který je volán místo dané operace, INSTEAD OF. Tato varianta je však dostupná pouze pro databázové pohledy, kdy doplňuje dodatečné operace nad pohledem.[1] Jedná se tedy většinou o triggery BEFORE a AFTER. Jako příklad je možné uvést trigger, který se spustí poté, co se stane řadová sestra sestrou hlavní. Logicky se jí zvýší plat. Jestliže je tedy identifikační číslo řadové sestry zakomponováno do tabulky hlavních sester, spustí se automaticky trigger, který jí zvýší plat o určité procento. Syntaxe vytvoření triggeru je potom následující:[8]

CREATE TRIGGER jmeno_triggeru { BEFORE | AFTER } { INSERT, UPDATE, DELETE } **ON** jmeno_tabulky **BEGIN** kod_triggeru_ktery_se_ma_provest **END**;

Pro úplnost je nutné se zmínit o klíčových slovech INSERT, UPDATE a DELETE ve spojení s klausulemi BEFORE a AFTER.[1]

- **BEFORE**
  - INSERT – před vložením nového řádku
  - UPDATE – před úpravou existujícího řádku
  - DELETE – před smazáním řádku

- **AFTER**
    - INSERT – po vložení nového řádku
    - UPDATE – po úpravě existujícího řádku
    - DELETE – po smazání řádku

Je-li nutné trigger smazat, používá se tento příkaz:[4]

DROP TRIGGER nazev_triggeru;

Rovněž je možné trigger přejmenovat, a to pomocí příkazu REPLACE. Některé databázové prostředky ovšem toto nemusí podporovat, proto bývá lepší trigger smazat a znovu ho vytvořit, jak je popsáno výše.[6]

Co se týče bezpečnosti, výhody triggerů jsou zároveň jejich nevýhodou. Výhodou je, že díky nim je možné databázi ošetřit různými způsoby, aby se aktualizovala, tedy chovala se trochu samostatně. To je však i nevýhodou, triggery jakožto krátké ústřižky kódu nejsou vlastně vyvolány ničím přímo, a tak si žijí v databázi vlastním životem. Jejich práce je prováděna jako vedlejší efekt nějaké akce, a tak lidé často zapomínají, že triggery v databázi vůbec jsou. Další problém nastává, když jsou triggery používány špatným způsobem, respektive administrátor či skupina lidí, kteří triggery zavádí, nedokáží dobře odhadnout dopad triggeru či nedokážou předvídat, co vše se v databázi může stát. Tyto ústřižky špatně odhadnutých kódů potom často generují chyby. To může být potenciální riziko pro celou databázi. Může se také stát, že trigger B, který je spuštěn triggerem A, opět spustí trigger A, v tomto případě se generuje chyba dokola a dojde k nekonečnému zacyklení. Toto je třeba mít ošetřené a nedopustit, aby se něco takového mohlo stát. Tohoto problému se také týká, jakým způsobem je řešeno vzájemné volání triggerů, a jestli vůbec.[7]

Je tedy zřejmé, že triggery mohou způsobovat vážné chyby uvnitř databáze a narušit její bezpečnost a korektnost. Není však na místě je jednoznačně odsoudit jako něco špatného. Záleží na lidském úsudku a na odhadnutí situace. Pokud se bude s triggery nakládat rozvážně a důsledně, mohou být ku pomoci, nikoli na obtíž.

3.6 Transakční zpracování

Prostřednictvím transakcí databázový prostředek zajišťuje konzistenci dat. Transakce je akce či posloupnost činností, které se chovají jako celek nebo se nesmí provádět vůbec. Tento krok je nesmírně důležitý, transakce se nesmí chovat jako jednotlivé dílčí kusy operací či příkazů. Narušilo by to její konzistenci.[8] Existují-li 3 tabulky – tabulka zdravotních sester, tabulka čipů na oběd a tabulka kartičky do knihovny. Pokud je zdravotní sestře potřeba z nějakého důvodu změnit ID číslo, je nezbytně nutné změnu provést ve všech tabulkách. Pokud se změna provede pouze ve dvou a na třetí tabulku se zapomene, potom není databáze v konzistentním stavu, což narušuje pravidlo, že databáze musí být v konzistentním stavu před transakcí i po ní. Logicky tedy vyplývá, že databáze může být a mnohdy je v nekonzistentním stavu při právě probíhající transakci. V souvislosti s transakcemi se na ně kladou jistá pravidla označená akronymem **ACID**.[8]

- **(A) Atomicity** – bod, který zde byl nastíněn, transakce musí proběhnout jako celek, nebo vůbec.
- **(C) Consistency** – databáze musí být před transakcí i po ní v konzistentním stavu.
- **(I) Isolation** – zde se jedná o tzv. nezávislost transakcí, dílčí operace transakce nejsou viditelné ostatním transakcím. Viditelné se stávají až po ukončení transakce.
- **(D) Durability** – úspěšně dokončené transakce jsou implementovány do databázového serveru.[8]

V souvislosti s transakčním zpracováním se hovoří o těchto třech příkazech, z toho dva jsou stěžejní: ROLLBACK, COMMIT a SAVEPOINT.[2] ROLLBACK odvolává všechny nepotvrzené změny a vrací databázi do stavu před posledním COMMITEM nebo do stavu při posledním regulérním ukončením databáze. COMMIT potvrzuje transakce. Je tedy zřejmé, že s příkazem COMMIT by se mělo pracovat opatrně, jedná se o velice agresivní nástroj, a pokud se použije špatně, může dojít k nenapravitelným škodám. Pokud se totiž potvrdí špatná transakce, nelze se příkazem ROLLBACK vrátit zpět, což může být velký problém.[8] Příkaz SAVEPOINT funguje jako jakýsi záchytný bod pro uživatele či administrátory. Problémem, ale i výhodou je, že SAVEPOINT není tak agresivní jako

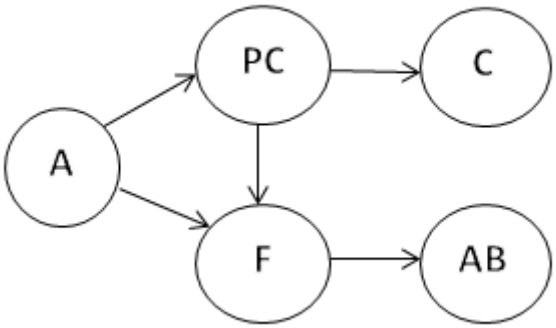
COMMIT. Výhodou je, že SAVEPOINTŮ lze vyrobit, kolik je potřeba. Pokud však dojde k výpadku proudu, všechny SAVEPOINTY jsou ztraceny a rovněž tak se ztratí důležitá data, která nebyla potvrzena. Toto je velká nevýhoda, a proto je třeba být s používáním těchto příkazů velice opatrný. Odpovídající příkaz pro vytvoření by mohl být následující:[8]

COMMIT; | **ROLLBACK**; | **SAVEPOINT** jmeno_savepointu; | **ROLLBACK TO** jmeno_savepointu;

Je na místě zmínit, že pokud je potřeba vrátit se na místo po posledním COMMITU, použije se ROLLBACK samotný. Pokud je potřeba vracet se na určitý SAVEPOINT, musí se přidat klausule TO a jméno daného SAVEPOINTU.[2]

Systém řízení báze dat (SŘBD) potom klade požadavky, které by transakční zpracování měla splňovat. Jedná se o to, aby se dokázala databáze zotavit z nečekaných chyb, například zmiňovaného výpadku. V tomto případě je třeba ihned po nastartování zadat příkaz ROLLBACK a vrátit se k poslední potvrzené transakci. SŘBD také klade požadavek na to, aby byl přístup k datům korektní a rychlý pro více uživatelů. Pokud v jedné databázi pracuje více uživatelů, jedná se o paralelní zpracování transakcí. Logicky potom případ, kdy jsou transakce prováděny za sebou, se nazývá sériové zpracování transakcí. Cílem paralelního zpracování je efektivní rozdělení prostředků (zdrojů) více uživatelům, jako jsou procesor či paměti.[9]

V souvislosti s transakcemi se hovoří také o určitých stavech, ve kterých se transakce mohou objevit a objevují. Schéma je nastíněno v Obr. 2.[12]



Obrázek 2 - Stav transakcí [12]

- **Active** (Aktivní) – proces, který je od počátku provádění transakce.[12]
- **Partially committed** (Částečně schválený) – stav, ve kterém se proces nachází po poslední operaci.
- **Failed** (Chybný) – z nějakého důvodu není možné pokračovat dále v transakci.
- **Aborted** (Zrušený) - stav nedokončený, zrušený, je třeba se vrátit pomocí příkazu ROLLBACK.
- **Commited** (Potvrzený) – stav, který nastane po úspěšném zadání příkazu COMMIT.

Z hrubého popisu stavů a schématu vyplývá, že transakce končí v bodech C a AB. Začíná v bodě A. Ze stavu PC do stavu C se dostává po úspěšném zadání příkazu COMMIT. Ze stavu F do stavu AB se pak dostává pomocí zadání ROLLBACKU. Zajímavý je přechod mezi stavy PC a F, k tomuto může dojít například při již zmiňovaném výpadku proudu, kdy se obsah vnitřní paměti neodeslal na disk. Pokud se proces nachází ve stavu AB, je možné zkusit transakci opakovat nebo nechat běžet databázi dál; je ve stavu před posledním COMMITEM.[12]

Také zde je prostor pro chyby. Chyby na lokální úrovni ovlivňují transakci, ale hovoří se i o chybách globálních, které mohou mít vliv na více než jednu transakci. Pokud se hovoří o globálních chybách, zmiňují se často tyto tři typy:

- **Chyby úložných médií** – může dojít k poškození serveru, kde je databáze uložena, poškození disku, požáru apod.
- **Chyby systémové** – zde se jedná o tzv. uváznutí, jde o chyby na úrovni transakcí, neovlivňují celou databázi.[8]
- **Neočekávané chyby** – spadnutí systému, při nedokončené transakci dochází ke ztrátě dat. Vnitřní paměti po výpadku proudu ztratí veškerá data.

Pokud systém selže, z nějakého důvodu přestane fungovat, dostává se do stavu, který se značí jako nedefinovaný. Z tohoto stavu je poměrně lehká cesta, a to zadat ROLLBACK, který systémový stav opět definuje. Z tohoto důvodu existuje v databázi nástroj, který se nazývá **žurnál** (log file). Žurnál je de facto deník transakcí, databázový prostředek si v něm dokáže najít poslední bod záchrany a vrátit se k němu. Žurnál tedy

obsahuje historii všech změn transakcí nad danou databází. Takovýto nástroj je pro bezpečnost nesmírně důležitý.[8]

Ve druhém bodě byly nastíněny chyby systémové, kdy dochází k tzv. uváznutí. Obecně se tento problém týká dvou stavů A a B, přičemž stav A může být dokončen až po úspěšném dokončení stavu B; ten však potřebuje ke svému dokončení stav A. Situace se dostává do paradoxu a nekonečné smyčky. V databázi se toto může stát při používání databáze více uživateli. Zde je možné se setkat s termínem eskalace, kdy databázový prostředek má schopnost uzavírat či uzamykat větší celky. Existují-li dva uživatelé A a B, kteří pracují se stejnou tabulkou, která má 10 záznamů, tak pokud si uživatel přeje potvrdit polovinu z nich, stačí to ke splnění podmínky. Problém nastává, kdy i uživatel B si přeje uzavřít polovinu záznamů. Dochází k uváznutí. Databázový prostředek by chtěl uzavřít tabulku jak uživateli A, tak B. To však není možné. Z tohoto důvodu některé databázové prostředky nepodporují eskalaci zámků (např. Oracle Corporation).[1][8]

Může ovšem dojít k daleko závažnějším chybám, například k chybě na samotném serveru. Zde se hovoří o chybách fyzických. Disky se mohou poškodit, může dojít k nevratné ztrátě dat, což může být pro mnoho společností fatální či finančně nákladné. Z hlediska času se jedná o dny vracení databáze do konzistentního stavu. Z tohoto důvodu se všechny moderní databázové servery několikrát zálohují na jiné servery, dochází také k záloze žurnálu. V praxi je potom obnovování takového problému, volání databáze ze zálohových zdrojů a pomocí žurnálu uvedení databáze do konzistentního stavu.[8]

3.7 Bezpečnost databází a následky nebezpečných situací

Jak již bylo nastíněno, v dnešní době mají firmy nesmírné množství dat. Zcela logickou věcí se pak stává, že takto velké množství dat musí být někde skladováno, zálohováno a hlídáno. Dříve k tomuto účelu sloužily velké místnosti, kde bylo vše uloženo v papírové podobě. Rovněž se pak používaly magnetické pásky, na kterých data dlouho vydržela. V moderní době, by sotva stačilo pár budov a stohy papírů. Z těchto důvodů vznikají velké místnosti tzv. „serverovny“, kde jsou uloženy servery, na kterých jsou databáze uloženy. Firma je pak povinna zabezpečit serverům bezproblémový a hlavně bezpečný chod. Mnohdy se totiž na serverech nacházejí důležitá data, která by server

neměla za žádnou cenu opustit. K tomu však může za určitých okolností dojít, a to hned z několika důvodů.

3.7.1 Šifrování dat na serveru

Jak již bylo naznačeno, data se uskladňují na serverech. Servery jsou nejčastěji odlehlé místnosti (z důvodu většího hluku), kde je plno disků, které uchovávají data. Data, která se nacházejí na serveru, je třeba chránit softwarovým způsobem, například šifrováním. Dnes se používají asymetrické a symetrické šifry k šifrování dat. V praxi se tyto šifry používají spolu. Asymetrické šifrování je pomalé, avšak velice bezpečné. Symetrické šifrování je velice rychlé, ale není tolik bezpečné. V praxi se soubor (data) zašifrují pomocí symetrického šifrování a pomocí asymetrického šifrování se zašifrují klíče.[15]

3.7.2 Ztráta integrity

V kapitole 3.2 Datová integrita jsou nastíněny teoretické principy této problematiky. Je zcela zřejmé, že pro společnost může být nerespektování pravidel fatální a mnohdy se může databáze dostat do nekonzistentního stavu. Následné hledání dat, která se ztratila z důvodu nedodržení integrity, může být zdlouhavé, finančně náročné a mnohdy to končí nenalezením dat, tedy jejich ztrátou.[13]

3.7.3 Krádež databáze a její zneužití

Ke krádežím dochází v běžném životě poměrně často. Málokdy si však lidé uvědomují, že v dnešní době modernizace a sdílení veškerých dat je hlavní nebezpečí může potkat hlavně na internetu. Data, která se nacházejí v informačních systémech, jsou napadnutelná a může dojít i k jejich zneužití, respektive zpronevěře. Firmy či stát, který databáze udržuje v chodu, pak musí vynaložit nemalé náklady, aby k tomu nedocházelo. Jedná se zde například o fyzické zabezpečení databází, dále o hlídání specifického objektu kamerovými systémy, lidmi, případně dveřmi, které ověřují identitu jedince. Může se jednat o různé typy útoků, avšak vždy jde o lidskou činnost, která to má na svědomí. Může se jednat o nevědomé či nechtěné zneužití nebo o zneužití chtěné, za cílem vlastního zisku.[13]

3.7.4 Ztráta utajení

Toto se týká hlavně firem na trhu. Snad v každé takovéto databázi jsou tajné informace, které mají pro konkurenceschopnost firmy nesmírnou důležitost. Ztráta utajení souvisí s problematikou v kapitole 3.3 Nastavení práv a přístupu, vytváření rolí. Řešením ztráty utajení je předejít mu, a to správným nastavením práv lidem, kteří ve společnosti pracují. Pokud však ke ztrátě utajení dojde, nemusí to znamenat konec firmy, často to však znamená její oslabení vůči ostatním, konkurenčním firmám. Je tedy důležité na toto dávat pozor.[13]

3.7.5 Ztráta soukromí

Ztráta soukromí se logicky týká každého člověka. V databázích, a to nejen těch od ministerstev (centrální registr vozidel, databáze občanů atd.), se mnohdy objevují informace o každém občanovi. Například školní informační systém je také databází, ve které se nachází citlivá data – rodná čísla, bydliště atd. Z hlediska bezpečnosti se musí předejít úniku těchto dat. Pokud se nejedná o fyzické zajištění, je opět potřeba, aby příslušná práva ke čtení těchto dat měli pouze ti nejpovolanější. Je třeba si zároveň uvědomit, že firmy jsou ze zákona povinny předejít tomu, aby ke ztrátě soukromí došlo.[13]

3.7.6 Ztráta dostupnosti

Ztráta dostupnosti se týká hlavně společností, které nabízejí virtuální služby dostupné 24 hodin denně. V posledních letech dochází ke vzestupu tzv. cloudu, což je virtuální úložiště. Ztráta dostupnosti je pak neschopnost navázat spojení s takovýmto typem serveru. Pro firmy to může znamenat ztrátu financí, ale hlavně prestiže a přechod potenciálních i aktuálních zákazníků ke konkurenci.[13]

V podkapitole 3.7.3 Krádež databáze a její zneužití bylo nastíněno, jak se má chovat společnost po stránce fyzické bezpečnosti databáze. Naproti tomu ostatní problémy popsané v ostatních podkapitolách se týkají zabezpečení systémů síťovým typem. Jako příklad je vhodné uvést firewally, specifické softwary, které jsou mnohdy vytvářeny ve firmách na zakázky. Nebo také proxy servery, filtrování packetů, filtrování IP adres a jejich následné zakazování. Problematiku této věci řeší firmy, které svou databázi mají volně dostupnou na internetu. Jedná se tedy především o banky, e-shopy apod.[14]

Z této podkapitoly je zcela zřejmé, proč je bezpečnost tak důležitá. Následky narušení bezpečnosti jsou pak mnohdy daleko větší pro společnost jako takovou než pro samotnou databázi. Při navrhování databáze je tedy nutné, předvídat, co se může stát a jaké útoky hrozí, potažmo odkud. Je dobré si nejprve uvědomit, v jakém prostředí systém bude implementován, a až poté ho začít vytvářet. Mohlo by se stát, že by se vytvořil příliš bezpečný, velký a nákladný systém do prostředí, které vyžaduje systém malý a rychlý. Při zabezpečení databází pak zpravidla jde o síťové zabezpečení a dodržení teoretických principů, na kterých databáze pracuje – pohledy, triggery, integrita, autentizace uživatelů. Jedná se však také o vnitřní funkce databázového prostředku, díky kterým se dokáže databáze vrátit do konzistentního a správného chodu. A to hlavně díky transakcím, které jsou popsány v kapitole 3.6 Transakční zpracování. Pokud se pak hovoří o spouštěčích nebezpečných situací, myslí se tím zaměstnance firmy, ale i osoby mimo společnost (např. bývalé zaměstnance, nebo cizí osoby), chybným hardwaru nebo softwaru. Spouštěčem však může být i nedostatečně dobře odhadnutá situace před vznikem databáze, tedy špatně navržená bezpečnostní politika. Bezpečnost systému je tedy tak dobrá, jak dobří byli lidé, kteří jí navrhovali a implementovali.

4 Centrální registr vozidel

4.1 Uvedení do problematiky nového CRV

Mnozí v roce 2012 zaznamenali kauzu okolo nového centrálního registru vozidel. Ať už jako běžný provozovatel, který se o této problematice dozvěděl z médií, nebo jako zúčastněný člověk. Na straně problému, nebo na té druhé, která ho kritizovala. Centrální registr vozidel (dále jen CRV) se podařilo spustit v červenci roku 2012. Jelikož je to problematika relativně současná, je v této práci detailněji popsána. Následně je v další kapitole navrženo řešení, které je samozřejmě v rámci možností této práce značně zjednodušeno. Je třeba dodat, že tato práce nemá za cíl pranýřovat řešitele problému, ale podívat se na problematiku z hlediska technického. Proto se zde nevyskytují názvy firem atd.

Nový CRV byl vytvořen ze dvou hlavních důvodů. Prvním byl požadavek přejít na novější informační systém, který bude nově spravovat ministerstvo dopravy, a nikoliv ministerstvo vnitra, jak tomu bylo doposud. Za druhé, nový informační systém CRV umožňuje propojení databáze se zahraničím, což je velká výhoda. Zadávací dokumentace pro nový CRV je bohužel držena v tajnosti, respektive je nedostupná na internetu. Rovněž tak zde chybí konkrétní vyjádření chyb. Ministerstvo dopravy to obhajuje tím, že kdyby byla vydána dokumentace, jak nynější systém funguje, mohlo by dojít k bezpečnostnímu riziku.

Jelikož jsou informace ohledně nového CRV nedostatečné, čerpalo se z článků v mediích z let 2011 a 2012, které jsou přiloženy jako zdroj, a také z formulářů, které lidé musí vyplnit při zapsání nového vozidla do CRV, při jeho vyřazení a při převodu mezi majiteli.

Ihned po spuštění CRV se začaly na úřadech tvořit fronty. Například nový systém běžel hodinu a poté „spadnul“, mnohdy se ho nepodařilo rozchodit po zbytek dne.[10] Občas se stávalo, že dokázal pracovat celý den, ale jeho struktura byla značně poškozena. Toto poškození nejspíše nastalo při přechodu mezi informačními systémy, kdy se nedodržela databázová integrita a docházelo k hrubým porušením struktury databáze. Mnohdy se stávalo, že majitel podle nového CRV vlastnil například tahač letadel, přitom doma v garáži měl starou felicii. Tento problém se týká nedodržení referenční integrity,

kdy neexistovalo logické spojení dvou tabulek – majitele a tabulky vozidel.[11] Nedodržení referenční integrity se považuje za velice hrubou chybu. Stávaly se i případy, kdy dva různí lidé vlastnili stejné vozidlo. V tomto případě docházelo k chybě primárního klíče, který by neměl v tabulce vozidel povolovat vklad stejného vozidla. V praxi existují tyto dvě tabulky. Přičemž tabulka majitel má primární klíč IČO a tabulka vozidlo primární klíč SPZ a cizí klíč IČO (poukazující na tabulku majitelů). Tedy nelze vytvořit další záznam v tabulce vozidel, který by propojoval tabulku majitel a vozidel (primární klíč by byl duplicitní, což databázový prostředek nesmí umožnit).

Portál iDnes.cz dokonce informoval o tom, že některá vozidla vlastnili zesnulí lidé.[11] V tomto případě nejspíše docházelo k pomíchání dat ze záložních serverů a dat aktuálních. Data, která jsou vymazána ze současného CRV, se pro případ „backupu“ ukládají na servery. I v případě, kdy by například kriminalisté chtěli dohledat, kdo vlastnil konkrétní vůz před desítkami let. Rovněž se na iDnes.cz uvádí, že státní poznávací značka nesouhlasila v mnoha případech; zde je opět na vině nedodržení referenční integrity, tedy nesprávné propojení tabulek, a tím pomíchání dat s daty nekorektními.

Mnoho zaměstnanců příslušných úřadů a hlavně občané si stěžovali, že je nový CRV nesmírně pomalý. Server iDnes.cz ve svých článcích uvádí, že je o něco pomalejší, protože se musí prokousávat k základním registrům. Neuvádí však, že k již zmíněnému prokousávání dochází, protože je nová databáze špatně logicky propojena. Pokud bychom dosáhli správného logického propojení, byla by databáze mnohem rychlejší.[11]

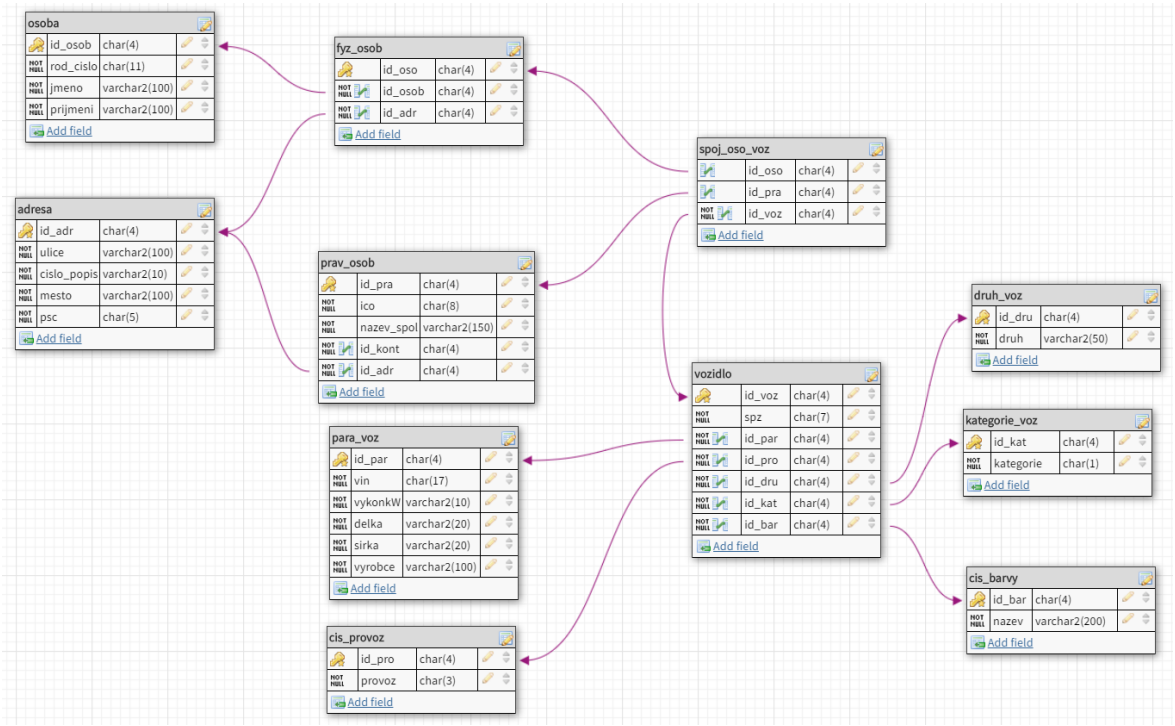
Otázku, jaký je rozdíl mezi provozovatelem a majitelem, si nejspíše nepoložili lidé, kteří nový CRV vymýšleli. Někteří majitelé, kteří vozidlo řádně splatili, byli stále vedeni pod leasingovou společností. Pro vysvětlení, majitelem může být leasingová společnost a provozovatelem člověk, který vůz teprve společnosti splácí. Zde mohlo dojít opět k problému s referenční integritou mezi tabulkami majitel a provozovatel. Nebo k situaci, o které je zmínka ve věci zesnulého občana o dva odstavce výše.

Při spuštění CRV docházelo také, sice ke správnému spojení tabulek majitel a vozidlo, ale informace o vozidlech zcela chyběly. Jiní zase měli problémy s jiným obsahem motoru svého vozu nebo se špatnými daty mezi prodaným vozem a novým majitelem.

Lze předpokládat, že všechny zde uvedené problémy jsou dostačujícím dokladem špatného fungování centrálního registru. Docházelo k velkému pomíchání dat, nesprávnému spojení, zpomalení databáze oproti navrhovanému řešení a k nekorektním datům. Obecně lze říct, že nový CRV měl špatně stanovenou datovou integritu. V mnoha případech docházelo k selhání referenční integrity, ale bohužel také k selhání primárních klíčů, tedy integrity entitní. Docházelo ke špatné synchronizaci starého a nového informačního systému.

4.2 Simulace a schéma CRV v praxi

Jak bylo již v minulé podkapitole zmíněno, tato část se bude věnovat vlastnímu řešení CRV. Výsledná databáze je pro tento příklad značně zjednodušena a vytvořena (simulována) v prostředí Oracle. Bude tedy využita syntaxe, kterou využívá databázový prostředek Oracle. Kvůli nedostatku dat se čerpalo z formulářů, které jsou k dispozici na městském úřadě Černošice. Formuláře umožňují přidat nové vozidlo. Na jejich základě byla vytvořena provizorní databáze, respektive napodobenina CRV, na které bude demonstrována problematika popsaná v předešlé podkapitole. Pro zjednodušení se vynechali právničtí a fyzičtí provozovatelé. Jednalo by se pouze o přidání dvou nových tabulek a spojení s ostatními. Pro tyto účely lépe poslouží model databáze bez nich.



Obrázek 3 - Modelový centrální registr vozidel

Ze schématu je zřejmé, že se jedná o model s 11 tabulkami. Pokud bychom zahrnuli i provozovatele, jednalo by se o tabulek 13. Tabulky provozovatele by bylo vhodné propojit referenční integritou s tabulkou osoba a adresa. V modelu se rozlišují dvě osoby, právnická a fyzická. Právnická obsahuje pouze název firmy a identifikační číslo osoby, které musí být unikátní. Dále je pomocí referenční integrity připojena adresa. Fyzická osoba si pomocí cizího klíče přebírá data z tabulky osoba, kde byly jako atributy zvoleny jméno, příjmení a unikátní rodné číslo. Důvod, proč dát tyto údaje do samostatné tabulky, je jednoduchý – zamezení redundanci dat.

Další větší tabulka je tabulka vozidel. Ta obsahuje pouze SPZ atribut, který musí být unikátní. Další údaje o vozidlech se ukládají do specifických tabulek. Existují zde dva číselníky. Jeden je pro provozní hodnotu (zda je vůz provozuschopný, či nikoliv), druhý je číselník barev. Pokud by se barvy a provozní hodnota dali přímo do tabulky vozidel, došlo by opět velké redundanci. Například by byla hodnota bílá v databázi statisíckrát. Pak zde jsou tabulky, které definují, jaké je vozidlo druh či kategorie, a nakonec také tabulka, která definuje jeho technické parametry. Všechny tabulky jsou propojeny pomocí cizích klíčů tzv. referenční integritou.

Neméně důležitá a zároveň poslední, o které nebyla řeč, je spojovací tabulka vozidel a osob. Je zřejmé, že by se zde mohlo narazit na potenciální problém. Například situace, kdy jeden člověk vlastní více než jedno vozidlo. Pokud by se spojila referenční integritou tabulka osoba a tabulka vozidlo bez spojovací tabulky, mohla by mít každá osoba pouze jedno vozidlo. Nebo by bylo třeba každou osobu pro nové vozidlo zadat znovu, tedy dosáhlo by se nechtěného výsledku, a to duplicity dat. Proto v rámci vazby 1:M je nutné založit spojovací tabulku, která se v modelu nazývá spoj_oso_voz. Pro úplnost jsou zde CREATE příkazy (pro vytvoření modelové databáze).

```
CREATE TABLE cis_provoz (id_pro CHAR(4) NOT NULL PRIMARY KEY, provoz CHAR(3) NOT NULL);
CREATE TABLE druh_voz (id_dru CHAR(4) NOT NULL PRIMARY KEY, druh VARCHAR(50) NOT NULL);
CREATE TABLE kategorie_voz (id_kat CHAR(4) NOT NULL PRIMARY KEY, kategorie CHAR(1) NOT NULL);
```

CREATE TABLE cis_barvy (id_bar **CHAR(4) NOT NULL PRIMARY KEY**, nazev **VARCHAR(200) NOT NULL**);

CREATE TABLE para_voz (id_par **CHAR(4) NOT NULL PRIMARY KEY**, vin **CHAR(17) NOT NULL UNIQUE**, vykonkW **VARCHAR(10) NOT NULL**, delka **VARCHAR(20) NOT NULL**, sirka **VARCHAR(20) NOT NULL**, vyrobce **VARCHAR(100) NOT NULL**);

CREATE TABLE vozidlo (id_voz **CHAR(4) NOT NULL PRIMARY KEY**, spz **CHAR(7) NOT NULL UNIQUE**, id_par **CHAR(4) NOT NULL, FOREIGN KEY** (id_par) **REFERENCES** para_voz(id_par), id_pro **CHAR(4) NOT NULL, FOREIGN KEY** (id_pro) **REFERENCES** cis_provoz(id_pro), id_dru **CHAR(4) NOT NULL, FOREIGN KEY** (id_dru) **REFERENCES** druh_voz(id_dru), id_kat **CHAR(4) NOT NULL, FOREIGN KEY** (id_kat) **REFERENCES** kategorie_voz(id_kat), id_bar **CHAR(4) NOT NULL, FOREIGN KEY** (id_bar) **REFERENCES** cis_barvy(id_bar));

CREATE TABLE adresa (id_adr **CHAR(4) NOT NULL PRIMARY KEY**, ulice **VARCHAR(100) NOT NULL**, cislo_popis **VARCHAR(10) NOT NULL**, mesto **VARCHAR(100) NOT NULL**, psc **CHAR(5) NOT NULL**);

CREATE TABLE osoba (id_osob **CHAR(4) NOT NULL PRIMARY KEY**, rod_cislo **CHAR(11) NOT NULL UNIQUE**, jmeno **VARCHAR(100) NOT NULL**, prijmeni **VARCHAR(100) NOT NULL**);

CREATE TABLE fyz_osob (id_oso **CHAR(4) NOT NULL PRIMARY KEY**, id_osob **CHAR(4) NOT NULL, FOREIGN KEY** (id_osob) **REFERENCES** osoba(id_osob), id_adr **CHAR(4) NOT NULL**, **FOREIGN KEY** (id_adr) **REFERENCES** adresa(id_adr));

CREATE TABLE prav_osob (id_pra **CHAR(4) NOT NULL PRIMARY KEY**, ico **CHAR(8) NOT NULL UNIQUE**, nazev_spol **VARCHAR(150) NOT NULL**, id_adr **CHAR(4) NOT NULL, FOREIGN KEY** (id_adr) **REFERENCES** adresa(id_adr));

CREATE TABLE spoj_oso_voz (id_oso **CHAR(4), FOREIGN KEY** (id_oso) **REFERENCES** fyz_osob(id_oso), id_pra **CHAR(4) , FOREIGN KEY** (id_pra) **REFERENCES** prav_osob(id_pra), id_voz **CHAR(4) NOT NULL, FOREIGN KEY** (id_voz) **REFERENCES** vozidlo(id_voz));

Pokud by byla takto navržena cílová databáze, nemělo by docházet k problémům s referenční integritou. Zabránilo by se duplicitě dat v rámci dvou majitelů jednoho vozidla

(primární klíč to nepovoluje). Pokud by bylo třeba odstranit například tabulku osoba, nebude to umožněno, protože zde existuje logická návaznost na ostatní tabulky. Je tedy třeba použít například kaskádovitý styl zachování referenční integrity. Tento styl je popsán v kapitole 3.2.3. Referenční integrita. Otázkou zůstává, zda při takto popsaných CREATE příkazech je možné získat informace napříč tabulkami. Jako příklad je zde doloženo jak získat údaje o vozidlech se specifickými parametry. Parametry jsou tyto: spz, vin, provoz, druh, barva. Příkaz SELECT by byl následující:

```
SELECT vozidlo.spz AS SPZ, para_voz.vin AS VIN, cis_provoz.provoz AS Provoz, druh_voz.druh AS Druh, cis_barvy.nazev AS Nazev_barvy FROM vozidlo, para_voz, cis_provoz, druh_voz, cis_barvy WHERE vozidlo.id_par = para_voz.id_par AND vozidlo.id_pro = cis_provoz.id_pro AND vozidlo.id_dru = druh_voz.id_dru AND vozidlo.id_bar = cis_barvy.id_bar;
```

Pokud by ve všech tabulkách byly údaje, které tam mají být, databázový prostředek nalezne všechna takto vybraná vozidla. Pokud ne, databázový prostředek odpoví **no rows selected**. Tento příkaz spojuje více tabulek pomocí tzv. tečkové konvence a příkazu WHERE; ten se překládá stejně jako INNER JOIN. Tečková konvence se používá, aby nedocházelo k syntaktickým problémům. Pokud by se nepoužila tečková konvence, mohlo by dojít k tomu, že databázový prostředek bere dva stejnojmenné atributy ze dvou různých tabulek. Databázový prostředek následně neví, ze které tabulky má daný atribut čerpat. Některé databázové prostředky odpoví hláškou: **Column name is too ambigous** (například v Teradatě).

Databázový prostředek umožňuje ještě jeden typ propojení tabulek - pomocí příkazu JOIN. Výše uvedený SELECT by mohl být přepracován pomocí příkazů JOIN následovně:

```
SELECT vozidlo.spz, para_voz.vin, cis_provoz.provoz, druh_voz.druh, cis_barvy.nazev FROM vozidlo INNER JOIN para_voz ON vozidlo.id_par = para_voz.id_par INNER JOIN cis_provoz ON vozidlo.id_pro = cis_provoz.id_pro INNER JOIN druh_voz ON vozidlo.id_dru = druh_voz.id_dru INNER JOIN cis_barvy ON vozidlo.id_bar = cis_barvy.id_bar;
```

Bylo by vhodné poukázat na ještě jeden důležitý SELECT, tentokrát pouze pomocí JOINŮ. Jedná se o SELECT, který získá jméno a příjmení z tabulky osoba, rodné číslo a z tabulky vozidlo spz. V této situaci je nutné využít spojovací tabulku. Příkaz, který by data získal, by byl následující:

SELECT osoba.jmeno, osoba.prijmeni, osoba.rod_cislo, vozidlo.spz **FROM** osoba **INNER JOIN** fyz_osob **ON** osoba.id_osob = fyz_osob.id_osob **INNER JOIN** spoj_oso_voz **ON** fyz_osob.id_oso = spoj_oso_voz.id_oso **INNER JOIN** vozidlo **ON** vozidlo.id_voz = spoj_oso_voz.id_voz;

Poslední otázka, kterou se sluší zodpovědět, je, proč jsou zde uvedeny dva příklady spojení tabulek – WHERE a JOIN, když oba de facto udělaly stejně dobrou práci. Dříve se mělo za to, že příkaz JOIN zpomaluje databázi, že je klasický příkaz pomocí klausule WHERE rychlejší. To však není pravda, obě syntaxe jsou stejně rychlé. Rozdíl je v myšlení administrátora. Uvádí se také, že díky formulaci JOIN se lépe hledají chyby související se syntaxí. Obě možnosti jsou stejně přijatelné, s tím, že JOIN se rozhodně hodí na některé operace více.

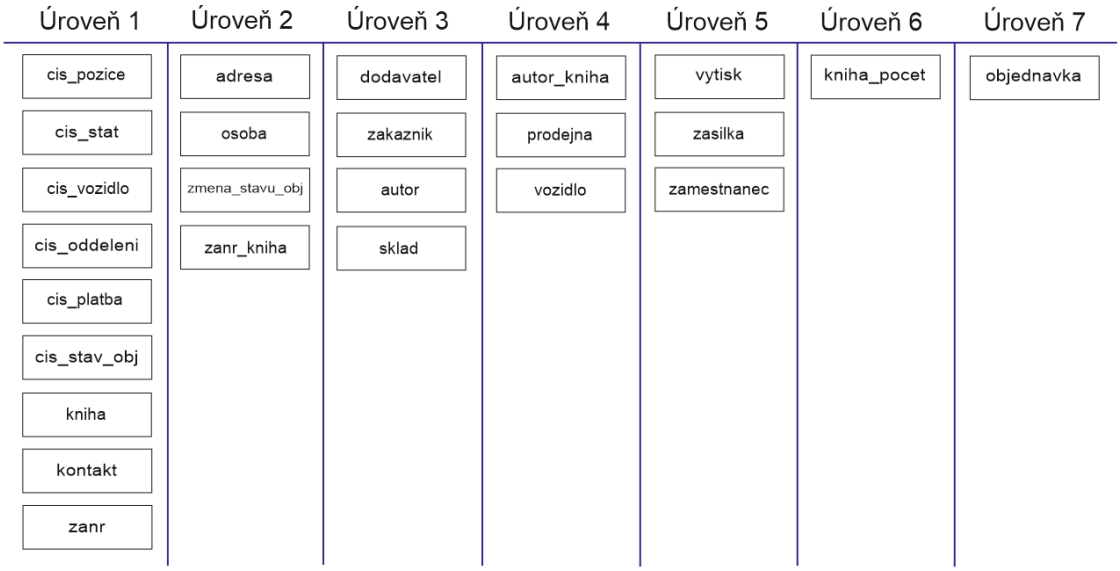
Neexistuje totiž pouze INNER JOIN. Databázový prostředek nabízí modifikace jako LEFT JOIN, RIGHT JOIN, OUTTER JOIN. Každý příkaz má svůj důvod. Kde INNER JOIN je vlastně průnik ve Vennových diagramech, potom LEFT JOIN je celá tabulka A + společné atributy s tabulkou B. RIGHT JOIN je potom logicky celá tabulka B + společné atributy s tabulkou B. FULL OUTTER JOIN je naprostý opak k INNER JOIN, kdy je vybráno vše, co není společné. Je tedy zřejmé, že použití klauzule WHERE nebo JOIN záleží na situaci.

5 Vlastní řešení

5.1 Popis modelované databáze a schéma úrovní databáze

Praktická část se věnuje ukázce postupů, které byly popsány v teoretické části. Postupy budou demonstrovány na modelové databázi. Je třeba podotknout, že by bylo daleko vhodnější použít příklad z praxe, popsat databázi již fungující. To však není možné, jelikož si firmy své databáze velice obezřetně chrání. I když jste zaměstnance takovéto firmy, nelze nic použít kvůli dodržení mlčenlivosti.

Z toho důvodu zde bude popsána a vytvořena modelová databáze pro knihkupectví, které prodává i přes internet. Jedná se o středně velké až velké knihkupectví, které má zájem dovážet zboží i do zahraničí (pro začátek na Slovensko a do Německa). Model, který byl vytvořen, je nástinem reálné databáze, která by mohla v praxi existovat. Je zřejmé, že některé tabulky by bylo dobré, pro úplnost systému přidat. Avšak pro popis teoretických východisek je toto řešení více než dostačující.



Obrázek 4 - Schéma úrovní

Databáze byla vytvořena v prostředku Oracle Database 11g. Celý systém se skládá z jedné databáze a 25 tabulek, které jsou propojené pomocí referenční integrity. Je tedy třeba zakládat tabulky od těch nadřazených, respektive od těch, které nejsou provázané uvnitř referenční integritou. Kdyby se zakládaly tabulky, které mají uvnitř sebe cizí klíč (aniž by byla založena tabulka, na kterou se klíč odkazuje), tak by databázový prostředek vrátil chybovou hlášku, jelikož nemůže najít tabulku, na kterou se odkazovat.

Jako první je dobré znázornit schéma tabulek, z něhož je naprosto zřejmé, které tabulky se budou zakládat první a které poslední (obr. 4). Na schématu je zřejmé, že pokud se tato databáze bude takto zakládat, je nutné kvůli cizím klíčům postupovat zleva doprava, tedy začínat první úrovní a končit úrovní sedmou. Naopak pokud je potřeba mazat tabulky, musí se postupovat zprava doleva. Pokud je potřeba údaje měnit, musí se rovněž postupovat zprava doleva. Kdyby bylo potřeba změnit primární klíč dodavatelů s tím, že reference je v podřazené tabulce zásilka, databázový prostředek zahlásí chybu: **integrity constraint (SYSTEM.SYS_C007478) violated - child record found**.

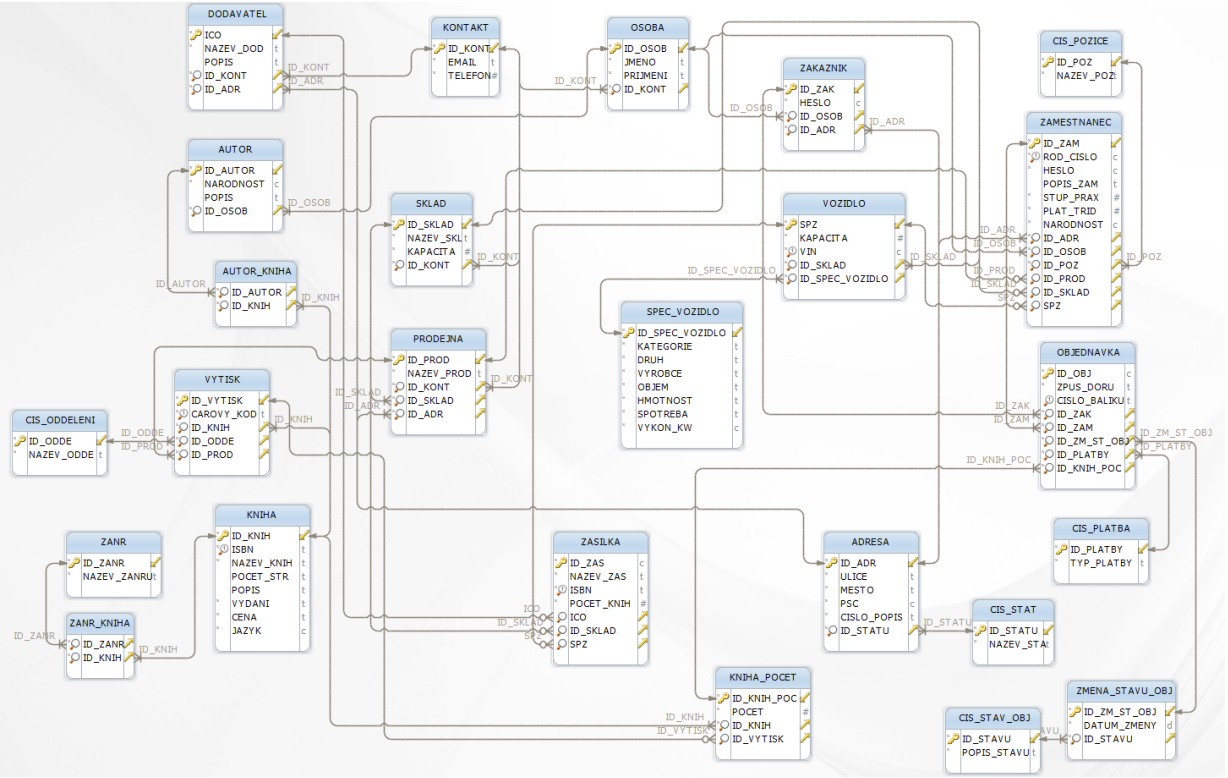
Toto schéma je velice důležité si uvědomit, při změně údajů referenční integrity je to klíčové. Hodilo by se ji měnit restriktivním způsobem a to dovést změny od nejpodřízenější tabulky do tabulky nadřízené. Na druhou stranu je třeba podotknout, že výše popsaný postup je obecný. Pokud je známo, které tabulky jsou provázané s kterými, je možné měnit údaje i třeba ve třetí úrovni, aniž by se zasahovalo do úrovní nižších. Jako příklad lze uvést tabulky vozidlo a osoba. Tabulky jsou na první pohled v jiné úrovni, ale pokud se odstraní tabulka osoba, neovlivní se tím nijak fungování tabulky vozidlo. Z toho důvodu je dobré demonstrovat ještě jedno schéma, které ukáže, jak jsou tabulky přesně propojené. Toto schéma se nazývá relační model.

5.2 Relační model řešené databáze

Relační model se vytváří za účelem zpřehlednění celé databáze. Je poměrně obtížné hledat chyby, pokud je databáze vytvořená pouze SQL příkazy. Z toho důvodu existuje relační model, jenž graficky zobrazí, které tabulky jsou logicky provázány cizími klíči odkazujícími na primární klíče jiných tabulek. Díky relačnímu modelu je možné vygenerovat vytvářecí (CREATE) SQL příkazy. Tuto funkci zpravidla obstarávají softwary určené k vytváření relačních modelů.

Na obrázku 5 je vidno 25 tabulek, které jsou mezi sebou propojeny cizími klíči. Žlutý klíč (zpravidla u prvního záznamu) značí primární klíč. Primární klíč má přívlastek not null, ten je značen pomocí hvězdičky u atributu (v levé části). Lupa značí hodnoty unikátní. Cizí klíče mají nastavené defaultně unikátnost. V některých případech je potřeba unikátnost doplnit (hovoří-li se o normálních atributech). Například ISBN v tabulce kniha – jako primární klíč je potřeba ID_KNIH, ale není možné povolit, aby ISBN bylo neunikátní, docházelo by tak

k duplicitě dat. Některé obyčejné atributy na druhou stranu není nutné mít ani vyplněné (např. popis u dodavatelů). Pokud je u cizích klíčů prázdné kolečko, nemá atribut přídavek not null.



Obrázek 5 - Relační model

5.3 Bezpečnostní rizika

Než bude možné databázi vytvořit, je třeba si uvědomit, která bezpečnostní rizika mohou hrozit.

- Je třeba ohlídat, aby všichni uživatelé, kteří se systémem budou pracovat, měli taková práva, aby nemohli narušit běh databáze. Toto se bude řešit v podkapitole 5.8.2 Vytvoření rolí.
- Je třeba si uvědomit, jak zabezpečit reálně vyskytující se servery. Serverové místnosti hlídat pomocí kamer a ochranky, která pustí k serverům jen povolané lidi. Pokud by se serverové místnosti nehlídaly, mohlo by dojít ke krádeži dat konkurencí nebo ke zničení systému.
- V rámci obchodování na internetu je třeba zajistit bezpečnost zvenčí. Takováto bezpečnost se zajistí šifrováním dat, která server odesílá. Toto je potřeba vyřešit na softwarové úrovni.

- Nesmí docházet ke ztrátě dostupnosti, to by znamenalo pro on-line prodej finanční ztráty. Je třeba vyřešit záložní zdroje napájení v případě výpadku proudu, aby servery běžely dál.

5.4 Realizace datové normalizace

Snad nejdůležitějším pojmem při vytváření databází je datová normalizace. V databázích není možné připustit redundanci dat. Rovněž musí být databáze schopna umožnit vyhledat jakákoliv smysluplná data. Řešení těchto problémů je dosažitelné pomocí tzv. normálních forem. Stručně řečeno, normální formy umožňují ověřit správnost návrhu databáze. Díky normálním formám je možné razantně omezit duplicitu dat a dosáhnout úspory místa, které databáze zabírá.[13]

5.4.1 Aplikace první normální formy

První normální forma definuje, že databáze nesmí obsahovat násobná data, respektive nesmí docházet k tzv. multizávislostem. Výsledkem toho je neplochost záznamů, to znamená, že jedna tabulka obsahuje více záznamů v rámci jednoho atributu. Jako příklad lze uvést tabulku sklad, která by v sobě měla uveden atribut vozidlo, v němž by bylo více vozidel. Toto se nesmí stát, proto je třeba vytvořit tabulku sklad a tabulku vozidlo, přičemž tabulky budou provázány cizími klíči.[13]

5.4.2 Aplikace druhé normální formy

Druhá normální forma v podstatě popisuje, že neklíčová data relace musí funkčně záviset na primárním klíči. Pokud tato forma nebude dodržena, objeví se v databázi velká redundance dat. Jako příklad je možné uvést číselníky. Například tabulka adresa, která má atribut stát. Pokud by se stát definoval jako VARCHAR(100) a vždy by se plnil při vkládání nového zaměstnance, vznikala by opět velká redundance. Z tohoto důvodu existuje tabulka číselník států, tento číselník je propojen s tabulkou adresy cizím klíčem.[13]

5.4.3 Aplikace třetí normální formy

Třetí normální forma je poslední formu, která bude v této práci popsána, ostatní nejsou pro záměr této práce tak důležité. Tato forma definuje, že všechna neklíčová data nesmí záviset na hodnotách mezi sebou, ale pouze na hodnotách klíčových. Je třeba to demonstrovat opět na příkladu navrhované databáze. Nesprávné by bylo, kdyby tabulce objednávka byly přidány atributy zákazníka, tedy nevytvářela by se tabulka zákazník, ale pouze by se atributy

tabulky zákazník přidaly do objednávky, tedy vznikly by atributy – id_objednávky, název, id_zákazníka, jméno. Pro splnění třetí normální formy je potřeba rozdělit problém na tabulku objednávky a tabulku zákazníků a opět je propojit pomocí cizích klíčů.[13]

5.5 Datová integrita modelu

Před založením samotné databáze je potřeba uvědomit si, kde a jak bude použita datová integrita. Teoretická východiska jsou popsána v kapitole 3.2. Datová integrita. Každá tabulka by měla mít primární klíč. Pokud by nebyl některé tabulce přiřazen primární klíč, mohlo by dojít k duplicitě dat. Primární klíč pro své naplnění zpravidla používá funkci auto increment. Jedná se o velice užitečnou funkci, která vyplňuje automaticky hodnotu primárního klíče. Většinou se začíná od nuly a přičítá se pro každý nový záznam jednička. Je třeba si uvědomit, kde auto increment použít a kde nikoliv. Tato funkce bude hojně využívaná u tabulek, kde primární klíč není tolik důležitý, respektive kde slouží pouze jako cizí klíč jiné tabulky, nebo pro unikátnost záznamů. Tyto tabulky jsou například objednávky a výtisk. Na druhou stranu někde je potřeba vložit primární klíč individuálně, respektive nemůže být automaticky vygenerován. Například primární klíč tabulky vozidlo, kde je primárním klíčem SPZ číslo vozidla. Zde by bylo naprosto nevhodné použít auto increment.

V modelové databázi nebyla nikde využita doménová integrita. Nikde nebylo potřeba omezit vkládané údaje na určitý interval. Na druhou stranu třetí typ integrity, a to referenční integritu byl používán hojně. Je třeba logicky související tabulky propojit, případně je potřeba vytvořit pomocnou (spojovou) tabulku, jelikož může existovat vazba M:N.

5.6 Vytvoření databáze

Aby vše fungovalo, je potřeba ke schématům, která jsou popsána výše přiřadit SQL příkazy, jež modelovou databázi vytvoří. V této podkapitole bude postupně vytvořena celá databáze, přičemž u každé tabulky bude popsáno to důležité. Nejlehčí bude použít schéma úrovní (obr. 4). Jako první bude nejvýhodnější vytvořit všechny číselníky. Číselníky se v databázi používají jako permanentní tabulky, které se naplní při zakládání databáze a poté už se jen využívají. V tomto případě se to týká tabulek s příponou CIS_. Do číselníků jsou například zahrnuty: číselník států (seznam všech států), číselník platby atd. Číselníky se vytváří, aby se zamezilo redundanci dat. Číselníky není třeba komentovat, jelikož jsou jednoduché. Většinou obsahují pouze to, co je v názvu.

```
CREATE TABLE cis_pozice (id_poz CHAR(4) NOT NULL PRIMARY KEY, nazev_poz VARCHAR(100) NOT NULL);
CREATE TABLE cis_stat (id_statu CHAR(4) NOT NULL PRIMARY KEY, nazev_sta VARCHAR(150) NOT NULL);
CREATE TABLE cis_oddeleni (id_odde CHAR(4) NOT NULL PRIMARY KEY, nazev_odde VARCHAR(100) NOT NULL);
CREATE TABLE cis_zpu_doru (id_zpu_doru CHAR(4) NOT NULL PRIMARY KEY, zpus_doru VARCHAR(100) NOT NULL);
CREATE TABLE cis_platba (id_platby CHAR(4) NOT NULL PRIMARY KEY, typ_platby VARCHAR(50) NOT NULL);
CREATE TABLE cis_stav_obj (id_stavu CHAR(4) NOT NULL PRIMARY KEY, popis_stavu VARCHAR(50) NOT NULL);
```

Poslední dva číselníky se týkají on-line objednávek: jakým způsobem zákazník platí a dále v jakém stavu se objednávka nachází. Pro úplnost je vhodné uvést data z některých číselníků (např. cis_stav_obj bude plněn daty – Nová, Ve zpracování, Zaplacená, Doručená). Pokud bude objednávka doručena, není třeba, aby v systému zůstávala, a je třeba ji odstranit. Toto je možné provést například databázovými triggery.

Dále je vhodné vytvořit obecnou tabulku kniha. V ní se každá kniha vyskytuje pouze jednou. Jednotlivé výtisky jsou k tabulce kniha přiřazovány cizím klíčem.

CREATE TABLE kniha (id_knih **CHAR(4) NOT NULL PRIMARY KEY**, isbn **VARCHAR(30) NOT NULL UNIQUE**, nazev_knih **VARCHAR(100) NOT NULL**, pocet_str **VARCHAR(10),** popis **VARCHAR(400)**, vydani **VARCHAR(30) NOT NULL**, cena **VARCHAR(10) NOT NULL**, jazyk **CHAR(2) NOT NULL**);

Každá osoba zaznamenaná v systému musí mít kontaktní údaje, stejně jako je musí mít i pobočky, sklady a dodavatelé. Kontaktní údaje lze dělit na dvě části – první část na tabulku kontakt, druhá část na tabulku osoba. Jelikož u skladů nejsou zaznamenávány kontaktní údaje jako jméno či příjmení, používá se zde pouze tabulka kontakt; u zaměstnanců se používá tabulka osoba, která bude popsána níže.

CREATE TABLE kontakt (id_kont **CHAR(4) NOT NULL PRIMARY KEY**, email **VARCHAR(50) NOT NULL**, telefon **NUMBER(9) NOT NULL**);

Každá kniha musí obsahovat informace o tom, jaký je to žánr. Avšak nesmí se přidat atribut žánr do tabulky kniha. Zaprvé může mít kniha více žánrů a za druhé by mohla vzniknout redundance dat.

CREATE TABLE zanr (id_zanr **CHAR(4) NOT NULL PRIMARY KEY**, nazev_zanru **VARCHAR(100) NOT NULL**);

V rámci této úrovně je ještě vhodné založit tabulku specifikace vozidla. Taková tabulka obsahuje důležité informace o vozidlech, která jsou zajímavé (např. výkon, objem a spotřeba).

CREATE TABLE spec_vozidlo (id_spec_vozidlo **CHAR(4) NOT NULL PRIMARY KEY**, kategorie **VARCHAR(10) NOT NULL**, druh **VARCHAR(20) NOT NULL**, vyrobce **VARCHAR(100) NOT NULL**, objem **VARCHAR(10) NOT NULL**, hmotnost **VARCHAR(10) NOT NULL**, spotreba **VARCHAR(10) NOT NULL**, vykon_kw **CHAR(4) NOT NULL**);

Ve druhé úrovni je potřeba založit adresu. Adresa je zakládána až poté, co existuje číselník států, do kterého si tabulka adresa sahá.

CREATE TABLE adresa (id_adr **CHAR(4) NOT NULL PRIMARY KEY**, ulice **VARCHAR(100) NOT NULL**, mesto **VARCHAR(100) NOT NULL**, psc **CHAR(5) NOT NULL**, cislo_popis **VARCHAR(20) NOT NULL**, id_statu **CHAR(4) NOT NULL, FOREIGN KEY** (id_statu) **REFERENCES** cis_stat(id_statu));

O pár řádků výše byla v první úrovni založena tabulka kontakt, nyní je možné založit tabulku osoba. Z důvodu toho, že si tabulka osoba sahá pro kontaktní informace do tabulky kontakt. Tyto dvě tabulky nejsou v jedné, protože například sklad nemá kontaktní informace jako jméno a příjmení. A přidat atributy do cílových tabulek by znamenalo povolit redundanci dat.

CREATE TABLE osoba (id_osob **CHAR(4) NOT NULL PRIMARY KEY**, jmeno **VARCHAR(50) NOT NULL**, prijmeni **VARCHAR(50) NOT NULL**, id_kont **CHAR(4) NOT NULL, FOREIGN KEY** (id_kont) **REFERENCES** kontakt(id_kont));

Následně je potřeba založit spojovací tabulku žánrů a knih. Jelikož zde existuje vazba M:N, je třeba situaci elegantně vyřešit. Toho je docíleno vytvořením spojovací tabulky, která bude pouze obsahovat cizí klíče na tabulky kniha a žánr. Tím je povoleno více žánrů jedné knize, a přitom je zamezeno redundanci a narušení integrity.

CREATE TABLE zanr_kniha (id_zanr **CHAR(4) NOT NULL, FOREIGN KEY** (id_zanr) **REFERENCES** zanr(id_zanr), id_knih **CHAR(4) NOT NULL, FOREIGN KEY** (id_knih) **REFERENCES** kniha(id_knih));

Jako poslední ve druhé úrovni je třeba vytvořit tabulku změna stavu objednávky. Jelikož je třeba zaznamenávat i historii změn, obsahuje tato tabulka datum změn. Zároveň si sahá do číselníku stavu objednávky (aby nedocházelo k redundanci).

CREATE TABLE zmena_stavu_obj (id_zm_st_obj **CHAR(10) NOT NULL PRIMARY KEY**, datum_zmeny **DATE**, id_stavu **CHAR(4) NOT NULL, FOREIGN** KEY (id_stavu) **REFERENCES** cis_stav_obj(id_stavu));

Dále je nutné vytvořit tabulku dodavatelů. V tabulce je nutné zaznamenávat popis dodavatele, jeho adresu (kterou si bude přebírat z tabulky adresa) a jeho kontaktní informace (z tabulky kontakt).

CREATE TABLE dodavatel (ico **CHAR(8) NOT NULL PRIMARY KEY**, nazev_dod **VARCHAR(100) NOT NULL**, popis **VARCHAR(400)**, id_adr **CHAR(4) NOT NULL, FOREIGN KEY** (id_adr) **REFERENCES** adresa(id_adr), id_kont **CHAR(4) NOT NULL, FOREIGN KEY** (id_kont) **REFERENCES** kontakt(id_kont));

Při prodeji on-line je umožněn prodej zákazníkům, je tedy potřeba vytvořit tabulku zákazník. Na ní je dobře vidět, proč byly vytvořeny dvě tabulky – kontakt a osoba. Zákazníkovi se přiřadí cizím klíčem pouze tabulka osoba – jelikož si tabulka bere data z tabulky kontakt a neumožňuje nulovou hodnotu záznamů, kontaktní informace budou k dispozici pomocí jedné vazby. Kdyby se přiřadily tabulka kontakt (email a telefon) a tabulka osoba (jméno a příjmení) zvlášť, tak by bylo třeba použít vazby dvě. Zákazník má také heslo, které se volí. Heslo je obsaženo v tabulce a má doménu CHAR(32). Hesla budou šifrována pomocí MD5 hashovací funkce, proto byla použita takto definovaná doména.

CREATE TABLE zakaznik (id_zak **CHAR(4) NOT NULL PRIMARY KEY**, heslo **CHAR(32) NOT NULL**, id_osob **CHAR(4) NOT NULL, FOREIGN KE**Y (id_osob) **REFERENCES** osoba(id_osob), id_adr **CHAR(4) NOT NULL, FOREIGN KEY** (id_adr) **REFERENCES** adresa(id_adr));

Dále je potřeba vytvořit tabulku autorů. Každá kniha musí mít svého autora, je třeba vyřešit problém s více autory. Například kniha, kterou nenapsal jeden autor, ale dva. Pokud by toto bylo špatně vyřešené, došlo by k neplochosti záznamů. To není možné dopustit. V dalším kroku se vytvoří spojovací tabulka kniha_autor, která bude obsahovat pouze cizí klíče, stejně jako byl tento problém vyřešen u tabulky kniha a její žánr.

CREATE TABLE autor (id_autor **CHAR(4) NOT NULL PRIMARY KEY**, narodnost **CHAR(2) NOT NULL**, popis **VARCHAR(400)**, id_osob **CHAR(4) NOT NULL, FOREIGN KEY** (id_osob) **REFERENCES** osoba(id_osob));

CREATE TABLE autor_kniha (id_autor **CHAR(4) NOT NULL, FOREIGN KEY** (id_autor) **REFERENCES** autor(id_autor), id_knih **CHAR(4) NOT NULL, FOREIGN KEY** (id_knih) **REFERENCES** kniha(id_knih));

V rámci třetí úrovně je ještě potřeba vytvořit sklad, ze kterého bude rozváženo zboží do jednotlivých prodejen a dostávat zboží od dodavatelů – nakladatelů. U skladu je zajímavá jeho kapacita – kapacita je uvedena v počtu zásilek (pro zjednodušení).

CREATE TABLE sklad (id_sklad **CHAR(4) NOT NULL PRIMARY KEY**, nazev_skl **VARCHAR(100) NOT NULL**, kapacita **NUMBER(10) NOT NULL**, id_kont **CHAR(4) NOT NULL, FOREIGN KEY** (id_kont) **REFERENCES** kontakt(id_kont), id_adr **CHAR(4) NOT NULL, FOREIGN KEY** (id_adr) **REFERENCES** adresa(id_adr));

Ze skladu se bude pomocí vozidel přepravovat zboží do prodejen. Je tedy vhodné si definovat tabulky vozidla a prodejna. Tabulka vozidel si bude sahat pro informace do tabulky specifikace vozidel, která byla již založena. Důležitým atributem je kapacita vozidla, tedy kolik zásilek dokáží rozvést jednotlivá vozidla.

CREATE TABLE vozidlo (spz **CHAR(7) NOT NULL PRIMARY KEY**, kapacita **NUMBER(10) NOT NULL**, vin **CHAR(17) NOT NULL UNIQUE**, id_sklad **CHAR(4) NOT NULL, FOREIGN KEY** (id_sklad) **REFERENCES** sklad(id_sklad), id_spec_vozidlo **CHAR(4) NOT NULL**, **FOREIGN KEY** (id_spec_vozidlo) **REFERENCES** spec_vozidlo(id_spec_vozidlo));

CREATE TABLE prodejna (id_prod **CHAR(4) NOT NULL PRIMARY KEY**, nazev_prod **VARCHAR(100) NOT NULL**, id_kont **CHAR(4) NOT NULL, FOREIGN KEY** (id_kont) **REFERENCES** kontakt(id_kont), id_sklad **CHAR(4) NOT NULL, FOREIGN KEY** (id_sklad) **REFERENCES** sklad(id_sklad), id_adr **CHAR(4) NOT NULL, FOREIGN KEY** (id_adr) **REFERENCES** adresa(id_adr));

Na začátku byla vytvořena tabulka kniha, ale není to tabulka kniha, která je důležitá při prodeji. Tabulka kniha existuje pouze proto, aby zabránila redundanci dat, a v tomto případě redundanci značné. Například by se každý výtisk zaznamenával do tabulky kniha, kde by byl stejný název, popis, rok vydání, a to pro každý výtisk. Jediné v čem by se záznamy lišily, by byl čárový kód. Toto není možné dopustit, proto je nutné vytvořit tabulku výtisk. Tabulka výtisk bude obsahovat čárový kód a bude logicky propojena s tabulkou kniha. Dále bude výtisk využíván při on-line obchodu.

CREATE TABLE vytisk (id_vytisk **CHAR(10) NOT NULL PRIMARY KEY**, carovy_kod **VARCHAR(30) NOT NULL UNIQUE**, id_knih **CHAR(4) NOT NULL, FOREIGN KEY** (id_knih) **REFERENCES** kniha(id_knih), id_odde **CHAR(4) NOT NULL, FOREIGN KEY** (id_odde) **REFERENCES** cis_oddeleni(id_odde), id_prod **CHAR(4) NOT NULL, FOREIGN KEY** (id_prod) **REFERENCES** prodejna(id_prod));

Prozatím se zde hovořilo o převozu zboží mezi sklady teoreticky, nyní je však potřeba definovat tabulku zásilka. Tato tabulka eviduje všechny zásilky zboží, proto je možné určit, kde se zásilka nachází. To díky referenční integritě, díky níž je logicky propojeno více tabulek. Je třeba si uvědomit, že pomocí SQL není možní zaručit správnost takto definované tabulky. Takto definovaná tabulka umožňuje naplnit všechny tři cizí klíče, tedy nastalo by to,

že zásilka se nachází u dodavatelů a zároveň ve vozidle a také ve skladě. To je samozřejmě nesmysl a je potřeba na to myslet při vytváření softwaru.

CREATE TABLE zasilka (id_zas **CHAR(4) NOT NULL PRIMARY KEY**, nazev_zas **VARCHAR(100) NOT NULL**, isbn **VARCHAR(30) NOT NULL UNIQUE**, pocet_knih **NUMBER(10) NOT NULL**, ico **CHAR(8), FOREIGN KEY** (ico) **REFERENCES** dodavatel(ico), id_sklad **CHAR(4), FOREIGN KEY** (id_sklad) **REFERENCES** sklad(id_sklad), spz **CHAR(7), FOREIGN KEY** (spz) **REFERENCES** vozidlo(spz));

Jako jednu z posledních je potřeba vytvořit tabulku zaměstnanců. Zaměstnanci budou mít své ID, pomocí kterého se budou přihlašovat do systému, zároveň budou mít heslo, které bude stejně jako u zákazníků šifrované pomocí hashovací funkce MD5. Je třeba si opět uvědomit, že jsou-li povoleny nulové hodnoty pro logické provázání s prodejnou, vozidlem a skladem, je to opět ze stejného důvodu. Ne každý má vozidlo, ale někdo ho může mít. Rovněž tak zaměstnanec může pracovat buď pouze na skladě, nebo na prodejně. Tento problém není řešen v SQL příkazech, ale opět by byl vyřešen lehce pomocí softwarové stránky.

CREATE TABLE zamestnanec (id_zam **CHAR(4) NOT NULL PRIMARY KEY**, rod_cislo **CHAR(11) NOT NULL UNIQUE**, heslo **CHAR(32) NOT NULL**, popis_zam **VARCHAR(400)**, stup_prax **NUMBER(2) NOT NULL**, plat_trid **NUMBER(2) NOT NULL**, narodnost **CHAR(2) NOT NULL**, id_adr **CHAR(4) NOT NULL, FOREIGN KEY** (id_adr) **REFERENCES** adresa(id_adr), id_osob **CHAR(4) NOT NULL, FOREIGN KEY** (id_osob) **REFERENCES** osoba(id_osob), id_poz **CHAR(4) NOT NULL, FOREIGN KEY** (id_poz) **REFERENCES** cis_pozice(id_poz), id_prod **CHAR(4), FOREIGN KEY** (id_prod) **REFERENCES** prodejna(id_prod), id_sklad **CHAR(4), FOREIGN KEY** (id_sklad) **REFERENCES** sklad(id_sklad), spz **CHAR(7), FOREIGN KEY** (spz) **REFERENCES** vozidlo(spz));

V rámci objednávek nastává ještě jeden problém. Pokud by se teď vytvořila jako poslední tabulka objednávek, bylo by možné ke každé objednávce přiřadit pouze jeden výtisk každé knihy. Je tedy třeba vytvořit pomocnou tabulku, která bude zaznamenávat počet knih, jež si zákazník objednal. Tabulka obsahuje dva cizí klíče, a to klíč referující na primární klíč tabulky kniha a druhý na tabulku výtisk. Takto je to řešeno z jednoduchého důvodu. Pokud by se odkazovalo pouze na výtisk, mohlo by se stát, že by si zákazník objednával výtisk, který není na skladě ani na prodejně, čili výtisk, který by neexistoval. Proto je zde ještě druhý cizí

klíč, který specifikuje přímo knihu, již by si uživatel přál koupit. Zbytek by se opět řešil softwarově. Pokud by nebyl naplněný cizí klíč výtisku, bylo by potřeba danou knihu objednat od nakladatelů.

CREATE TABLE kniha_pocet (id_knih_poc **CHAR(4) NOT NULL PRIMARY KEY**, pocet **NUMBER(10) NOT NULL**, id_knih **CHAR(4), FOREIGN KEY** (id_knih) **REFERENCES** kniha(id_knih), id_vytisk **CHAR(10), FOREIGN KEY** (id_vytisk) **REFERENCES** vytisk(id_vytisk));

Jako poslední je třeba vytvořit tabulku objednávek, která zobrazuje jednotlivé objednávky.

CREATE TABLE objednavka (id_obj **CHAR(4) NOT NULL PRIMARY KEY**, popis **VARCHAR(400)**, cislo_baliku **CHAR(30) UNIQUE**, id_zpu_doru **CHAR(4) NOT NULL**, **FOREIGN KEY** (id_zpu_doru) **REFERENCES** cis_zpu_doru(id_zpu_doru), id_zak **CHAR(4) NOT NULL, FOREIGN KEY** (id_zak) **REFERENCES** zakaznik(id_zak), id_zam **CHAR(4) NOT NULL, FOREIGN KEY** (id_zam) **REFERENCES** zamestnanec(id_zam), id_zm_st_obj **CHAR(10) NOT NULL, FOREIGN KEY** (id_zm_st_obj) **REFERENCES** zmena_stavu_obj(id_zm_st_obj), id_platby **CHAR(4) NOT NULL, FOREIGN KEY** (id_platby) **REFERENCES** cis_platba(id_platby), id_knih_poc **CHAR(4) NOT NULL, FOREIGN KEY** (id_knih_poc) **REFERENCES** kniha_pocet(id_knih_poc));

5.7 Řešení auditability

Nad takto vytvořenou modelovou databází je nyní možné zakládat nové podmínky fungování či implementovat různé funkce pro lepší chod databáze. Jak už bylo nastíněno v teoretických východiscích, je důležité, kdo přistupoval k databázi a v jakém čase. Pro tyto účely bude stěžejní ID zaměstnance, popřípadě ID zákazníka, a jaké SQL příkazy zadával. Je třeba uvést, že zaměstnanec i zákazník budou s databází pracovat prostřednictvím softwaru, který by nad databází vznikl jako další krok vývoje informačního systému. Na pozadí se však budou zaznamenávat SQL příkazy. Dále je samozřejmě důležitý datum, aby bylo možné určit, kdy došlo ke ztrátě přístupnosti, ztrátě dat apod. Firmy by auditabilitu neměly podceňovat, ba naopak mělo by to být aktivně řešené téma.

5.8 Založení uživatele, přidělení práv, vytvoření rolí

Ještě než bude možné vůbec někoho sledovat, jak přistupuje k databázi, je třeba nadefinovat mu způsob, jakým může přistupovat k datům. Při vytváření databáze existuje zpravidla několik superuživatelů. Superuživatel má všechna přístupová práva a je přidělen do většiny rolí. Pokud existuje větší databáze, tedy existuje více datových kontejnerů, jsou pro každý kontejner určití superuživatelé. Superuživatelé se používají většinou tehdy, když je potřeba nad nějakou databází provést INSERT či DELETE, ale uživatelé na to nemají práva, ani specifické role. Při prvním vytvoření databáze bude vytvořen uživatel admin, který má administrátorská práva. Skrze něj se v praxi nepracuje, využívá se pouze na začátku, aby se díky němu vytvořil nový uživatel pro administrátora databáze (pro reálného člověka, který se o databázi bude starat).

5.8.1 Vytvoření uživatele

Pro vytvoření nového uživatele se bude opět vycházet z teoretického východiska, kde je přesně popsáno, jak uživatele vytvořit. Pro demonstraci je zde vytvořen uživatel, který se bude identifikovat pomocí hesla. Vytvoří se pomocí příkazu CREATE USER.

```
CREATE USER administrator IDENTIFIED BY ou17859@;
```

Administrátor se uloží do tabulky uživatelů/zaměstnanců. Heslo se uloží pomocí šifrování MD5 hashovací funkce. Takto vytvořený uživatel by však neměl na nic práva, dokonce by se ani nemohl přihlásit. To je potřeba povolit pomocí příkazu:

```
GRANT CREATE SESSION TO administrator;
```

Nyní je možné se jako administrátor přihlásit, avšak takto vytvořený administrátor nebude mít na nic práva, což je dosti nepraktické. V dalších krocích je třeba přidělit účtu administrátor práva, a to všechna. Pomocí klausule ALL a příkazu GRANT mu lze přidělit všechna práva. Po nějaké době, kdy se zaměstnanec rozhodne opustit firmu, je třeba jeho uživatelský účet zrušit, aby k němu neměl neoprávněně přístup. To lze provést následovně:

```
DROP USER administrator;
```

Databázový prostředek zahlásí hlášku: **User dropped**.

5.8.2 Vytvoření rolí

Jak již bylo zmíněno, je nevýhodné každému uživateli přidělovat práva samostatně, z toho důvodu existují role. Určitá práva se přiřadí do specifické role a roli je poté možné přidělit uživatelům. Role se vytváří pomocí příkazu CREATE ROLE. Pro účely této práce bude vytvořena role skladníka a role zákazníka. Následně je rolím možné přidělit práva a poté je možné role přiřadit existujícím uživatelům či zaměstnancům.

```
CREATE ROLE skladnik_role;
CREATE ROLE zakaznik_role;
GRANT SELECT, UPDATE, INSERT ON sklad TO skladnik_role;
GRANT SELECT ON kniha TO zakaznik_role;
GRANT skladnik_role TO s001;
GRANT zakaznik_role TO z001;
```

Pokud byl předtím vytvořen uživatel s001 a z001 a byl jim přidělen GRANT SESSION, měli by být nyní schopni provádět příkazy, které jim role povolují. To jest zákazníkovi nahlížet do tabulky kniha a skladníkovi nahlížet do tabulky sklad, měnit v nich údaje a upravovat je. Z bezpečnostního hlediska je lepší skladníkovi nedávat příkaz DELETE, tento příkaz by měli mít možnost provést pouze jeho nadřízení. Je také možné, že některé role se budou rušit. Například pokud se mění politika podniku a tato role by byla zastaralá. Nehodí-li se již role skladnik_role. Zruší se následujícím příkazem:

```
DROP ROLE skladnik_role;
```

Databázový prostředek zahlásí: **Role dropped**.

5.9 Vytvoření pohledů

Jelikož mnohdy není žádoucí, aby zaměstnanci pracovali přímo s tabulkou (jelikož jsou v ní údaje, které nesmějí být zveřejněny), je nutné vytvořit pohledy. Pohledy je možné vytvářet nad jednou tabulkou, ale i nad více tabulkami. Je potřeba, aby se zaměstnanec mohl podívat na objednávku jiného zaměstnance, respektive aby při vyřizování reklamace bylo vidět, kdo objednávku zpracovával. Popřípadě aby bylo možné zobrazit údaje o daném zaměstnanci. Pokud by se umožnila práce přímo s tabulkou, došlo by k problému. Jednak by bylo možné zobrazit heslo (což by nebyl takový problém, protože to by bylo zašifrováno), ale

také by bylo vidět zaměstnancovo rodné číslo. Proto je nutné vytvořit pohled, který tuto problematiku ošetří:

```
CREATE VIEW zame_view AS SELECT b.jmeno, b.prijmeni, a.popis_zam, a.narodnost, c.nazev_pozice FROM zamestnanec a INNER JOIN osoba b ON b.id_osob = a.id_osob INNER JOIN cis_pozice c ON c.id_poz = a.id_poz;
```

Takto vytvořený pohled zobrazí pouze jméno a příjmení zaměstnance, jeho popis, národnost a pozici. Ostatní údaje z tabulek cis_pozice, zamestnanec, osoba jsou nepřístupné, což je více než vyhovující. Díky pohledům je tedy možné zabránit úniku informací, které jako řídicí jednotka podniku je nutné chránit. Navíc správně stylizovaný pohled zlehčuje práci s více tabulkami. Pohledy se ruší pomocí stejné syntaxe jako tabulky pomocí příkazu DROP VIEW zame_view.

Takto by bylo možné vyřešit situaci, kdy je potřeba na webových stránkách zobrazit pouze stručné informace ohledně knihy, autora a žánru. Toho je docíleno následujícím pohledem:

```
CREATE VIEW struc_kniha AS SELECT osoba.jmeno, osoba.prijmeni, kniha.nazev_knih, kniha.popis, zanr.nazev_zanru FROM osoba INNER JOIN autor ON autor.id_osob = osoba.id_osob INNER JOIN autor_kniha ON autor.id_autor = autor_kniha.id_autor INNER JOIN kniha ON kniha.id_knih = autor_kniha.id_knih INNER JOIN zanr_kniha ON zanr_kniha.id_knih = kniha.id_knih INNER JOIN zanr ON zanr.id_zanr = zanr_kniha.id_zanr;
```

5.10 Použití transakčního zpracování

Zpravidla se stává, že je potřeba provést v určité SESSION změny dat. Občas se však stane, že uživatel změní data špatně nebo smaže nějaký záznam, který smazat nechtěl. Proto je výhodné využívat příkazů COMMIT a ROLLBACK. Případně si je možné pomoci i návratovými body, které jsou popsány v teoretické části. Pro modelovou situaci existuje tento příklad: prodavač ve skladu prodejny špatně načte knihy ze zásilky; místo titulu A zadá do systému titul B. Zpravidla existují dvě varianty, jak tuto situaci vyřešit. Může požádat vedoucího, aby odstranil několik set záznamů, což je v praxi nemožné. Nebo se provede tzv. ROLLBACK, což je daleko přijatelnější varianta. Zde bude uvedena modelová situace pro pár záznamů, pro více záznamů by to bylo stejné. Před zadáním knih do databáze se provede COMMIT. Následně skladník zadá do systému čárové kódy výtisků, ale zařadí je do špatné kategorie nebo špatně zadá některý údaj atp.

```
COMMIT;
INSERT INTO vytisk (carovy_kod, id_knih, id_odde, id_prod) VALUES
('978945126','0145','0014','0001');
INSERT INTO vytisk (carovy_kod, id_knih, id_odde, id_prod) VALUES
(974582563,'0155','0024','0021');
```

Nezadává se primární klíč, jelikož se klíč zadává automaticky, vkládají se tedy pouze ostatní údaje. Prodavač si uvědomil, že zadal špatně oddělení, kde se kniha nachází. Je třeba provést ROLLBACK, který vrátí situaci po zadaném COMMITU. Nyní může prodavač knihy zadat znovu. Pokud by se jednalo o pár knih, je možné to vyřešit příkazem UPDATE, pokud by měl prodavač příslušná práva.

6 Závěr

Cílem práce bylo demonstrovat na modelovém příkladu, jak se současné počítačové databáze zabezpečují. Z analýzy odborných publikací, ze studia informačních zdrojů a při vytváření vlastního řešení byly zjištěny následující poznatky.

V praxi se primární klíč či index většinou plní automaticky. Plnění obstarává sama databáze. Číslo takto vzniklé je většinou pseudonáhodné, často se totiž využívá příkazu, který vybere maximální číslo primárního klíče a přičte k němu jedničku. Do stejné kategorie, tedy do datové integrity, patří i referenční integrita. Referenční integrita je zčásti zabezpečená databázovým prostředkem. Dojde-li k pokusu o smazání tabulky propojené referenční integritou, není to umožněno, rovněž tak si databázový prostředek hlídá, pokud se zadávají nekorektní data. Tato zabezpečení v praxi však nestačí. Proto je důležité při zakládání nové databáze mít na paměti, v jakých místech je třeba ohlídat správnost chodu databáze. Při následné implementaci softwarového rozhraní (nad danou databází) je důležité tato nezabezpečená místa ošetřit a zakázat softwarovým způsobem vkládání dat na místa, kam by to databázový prostředek i přes nesmyslnost povolil.

Bylo zjištěno, že při nastavování práv novému uživateli je naprosto irelevantní přidělovat práva jednotlivě. Nejefektivnějším způsobem je vytvořit si role, které všem uživatelům přidělujeme. Také bylo zjištěno, že lepším způsobem je práva přidělovat postupně, tak jak je uživatel potřebuje (a jak mu je to povoleno). Druhý způsob přidělení všech práv a postupného odmazávání při prohřešku je krajně nebezpečný.

Při vytváření modelové databáze bylo zjištěno, že je daleko výhodnější novou databázi vytvářet pomocí logického modelu. Je zcela neefektivní databázi vytvářet nejdříve pomocí `CREATE` příkazů. Často dochází k úpravě referenční integrity mezi tabulkami, ke změně jmen atributů, ke změně datových typů. Takovýmto způsobem je lehké udělat velké množství chyb. Daleko efektivnější je vytvořit databázi pomocí logického modelu a následně automaticky vygenerovat SQL příkazy, které je potom třeba zkontrolovat.

Dále bylo zjištěno, že vytvoření bezpečné databáze nestačí spoléhat pouze na zakládací SQL příkazy. Je nutné si uvědomit, co je potřeba ošetřit, a následně toho docílit například pomocí PL/SQL.

7 Literatura

[1] Oracle Database Help Center. Database documentation [online]. 21. 11. 2014. [cit. 2017-02-01]. Dostupné z: https://docs.oracle.com/en/database/

[2] BRYLA Bob, LONEY Kevin. Oracle Database 11g DBA Handbook. McGraw-Hill Education, 2007. ISBN: 978-0071496636

[3] SHARMA Neeraj, PERNIU Liviu, CHONG F. Raul, IYER Abhishek, NANDAN Chaitali, MITEA Adi-Cristina, NONVINKERE Mallarswami, DANUBIANU Mirela. Database Fundamentals. IBM Corporation, 2010. ISBN: 9780986628375

[4] LONEY Kevin. Oracle Database 11g: The Complete Reference. McGraw-Hill Education, 2009. ISBN: 978-0-07-159876-7

[5] PFLEEGER P. Charles, PFLEEGER Lawrence Shari, MARGULIES Jonathan. Security in Computing. Prentice Hall, 2015, 5th Edition. ISBN: 978-0134085043

[6] NATAN Ron Ben. Implementing Database Security and Auditing. Digital Press, 2005, 1st Edition. ISBN: 1-55558-334-2

[7] NJOKU David. The Problem With Triggers [online]. 16. 3. 2015. [cit. 2017-02-06]. Dostupné z: http://allthingsoracle.com/the-problem-with-triggers/

[8] KORTH F. Henry, SUDARSHAN S. Chawathe, SILBERSCHATZ Abraham. Database System Concepts. McGraw-Hill Education, 2010, 6th Edition. ISBN: 978-0-07-352332-3

[9] POKORNÝ Jaroslav, VALENTA Michal. Databázové systémy. Česká Technika – Vydavatelství ČVUT, 2013. ISBN: 978-80-01-05212-9

[10] VENTUROVÁ Jitka., HORÁČEK Aleš., JEŽEK Petr., KOLAŘÍK Michal. Nový centrální registr vozidel zkolaboval po pár minutách provozu [online]. 9. 7. 2012. [cit. 2017-02-06]. Dostupné z: http://zpravy.idnes.cz/centralni-registr-vozidel-nefunguje-dsp-/domaci.aspx?c=A120709_085450_domaci_jpl

[11] VENTUROVÁ Jitka. Centrální registr vozidel funguje, ale pomalu a s potížemi [online]. 10. 7. 2012. [cit. 2017-02-06]. Dostupné z: http://zpravy.idnes.cz/centralni-registr-vozidel-funguje-ale-pomalu-fan-/domaci.aspx?c=A120710_091824_domaci_jpl

[12] KORTH F. Henry, SUDARSHAN S. Chawathe, SILBERSCHATZ Abraham. Database System Concepts [online]. 2010, 6th Edition [cit. 2017-02-07]. Dostupné z: http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/

[13] CONNOLLY Thomas, BEGG Carolyn. Database Systems: A Practical Approach to Design Implementation, and Management. Pearson Education Limited, 2005, 4th Edition. ISBN: 978-0-321-21025-8

[14] STALLINGS William. Cryptography and Network Security: Principles and Practices. Prentice Hall, 2005, 4th Edition. ISBN: 0-13-187319-9

[15] VANSTONE Scott, OORSCHOT van Paul, MENEZES Alfred. Handbook of Applied Cryptography. Boca Raton: CRC Press, 1997. ISBN: 0-8493-8523-7

[16] CODD Edgar Frank. “Is Your DBMS Really Relational?” and “Does Your DBMS Run By the Rules?” Computerworld magazine, October 14 and October 21, 1985.