Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Logování událostí v databázových evidencí

Bc. Tomáš Jukl

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Tomáš Jukl

Systémové inženýrství a informatika
Informatika

Název práce

Logování událostí v databázových evidencí

Název anglicky

Logging of events in database structures

Cíle práce

Diplomová práce je zaměřena na problematiku logování událostí v relačně databázově koncipovaných evidencí pomocí databázových triggerů. Náplní a smyslem této práce je:
a) objasnit teoretické principy relačně databázové technologie v kontextu s problematikou logování událostí,
b) zmapovat momentální stav této problematiky a vymezit její relevantnost včetně požadavků na ni kladených,
c) navrhnout možnosti přijatelných řešení těchto požadavků spojených s logováním událostí,
d) ověřit funkčnost navržených záležitostí,
e) ověřené záležitosti zobecnit pro další možná uplatnění.

Metodika

Použitá metodika zadané diplomové práce bude založena na studiu a analýze dostupných informačních zdrojů a případných existujících řešení v dané oblasti. Stěžejními metodami této práce budou metody a techniky relačně databázové technologie a SQL. Navrhované řešení bude zohledňovat identifikované požadavky a očekávání spojená s řešenou záležitostí. Na podkladě syntézy teoretických poznatků a dosažených výsledků budou formulovány závěry této diplomové práce a následně zobecněny pro další možná použití.

Závazný harmonogram:

Teoretické principy, literární rešerše – předmět 1. zápočtu z DP: do 5.9.2019

Zmapování momentální situace řešené problematiky, identifikace požadavků na ni kladených: do 31.11.2019

Navržení možného řešení se zřetelem na identifikované požadavky: do 31.1.2020

Ověření a zobecnění navrhovaných záležitostí – předmět 2. zápočtu z DP: do 25.3.2020

Doporučený rozsah práce

55-65 stran

Klíčová slova

Relačně databázová technologie, SQL, logování událostí, databázový trigger, referenční integrita

---

Doporučené zdroje informací

AUER, D J. – KROENKE, D. *Databáze.* Brno: Computer Press, 2015. ISBN 978-80-251-4352-0.

LACKO, Ľ. *Oracle : správa, programování a použití databázového systému.* Brno: Computer Press, 2002. ISBN 80-7226-699-3.

LACKO, Ľ. *SQL : hotová řešení : pro SQL Server, Oracle a MySQL.* Brno: Computer Press, 2003. ISBN 80-7226-975-5.

LACKO, L.: 1001 tipů a triků pro SQL. 1. vydání. Computer press, 2011. 416 stran.

PROCHÁZKA, D.: Oracle. 1vydání. Grada, 2009. 168stran. ISBN: 978-80-247-2762-2.

VALENTA, M. – POKORNÝ, J. *Databázové systémy.* Praha: České vysoké učení technické v Praze, 2013. ISBN 978-80-01-05212-9.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

doc. Dr. Ing. Václav Vostrovský

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 9. 3. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 9. 3. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 02. 04. 2020

Čestné prohlášení

Prohlašuji, že jsem svou diplomovou práci "Logování událostí v databázových evidencí" vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v závěru práce, v seznamu použitých zdrojů. Jakož autor také dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 03. 04. 2020 _____

Poděkování

Tímto bych rád poděkoval panu doc. Ing. Václavu Vostrovskému, Ph.D. za veškeré poskytnuté rady a připomínky, stejně tak jako za čas a trpělivost při konzultacích a za celkové vedení při vypracování této diplomové práce. Taktéž bych rád věnoval toto poděkování rodině a mým nejbližším, kteří mi při vypracování této práce poskytli cenné rady a oporu.

Logování událostí v databázových evidencí

Abstrakt

Diplomová práce na téma "Logování událostí v databázových evidencí" se zabývá problematikou využití databázových spouští neboli triggerů, k zajištění záznamu a uchování důležitých informací. Tyto informace se týkají zejména přístupu uživatelů do databáze – tedy operací LOGON a LOGOFF, jejich aktivity v rámci schéma, a to jak z hlediska zadaných příkazů DDL – CREATE; ALTER; DROP, tak i DML – INSERT; UPDATE; DELETE.

V rámci literární rešerše jsou zmíněny postupně nejdůležitější základní informace relačních databázových systémů, jež souvisí s navrhovaným praktickým řešením. Tyto informace následně přecházejí ze základní a obecné roviny do roviny specializovanější. Pojednává se velmi stručně o základních principech a pravidlech zvoleného databázového systému, tedy relačního databázového systému. Jsou uvedena hlavní negativa a pozitiva tohoto systému a následně je uveden způsob uspořádání dat v relačních db systémech. Dále se práce zaměřuje na teoretické principy normalizace, včetně stručného uvedení jednotlivých normálních forem, integrity dat, kdy jsou uvedeny konkrétní typy integritních omezení a způsoby jejich zajištění. Následně se práce zabývá problematikou databázových spouští. Jsou uvedeny významné oblasti jejich užití, rizika spjatá s jejich aplikací a základní postup jejich sestavení. Nakonec je v literární rešerši popsán formát XML a způsob, jakým ho lze v relační db, konkrétně na platformě společnosti Oracle, vygenerovat na základě informací uložených v příslušné db.

Praktická část této diplomové práce nabízí řešení logování událostí za pomoci PL/SQL databázových spouští a následný export zaznamenaných informací ve formátu XML. Je navržen a prostřednictvím ERD vyobrazen výchozí model. Jsou uvedeny syntaxe jednotlivých tabulek, včetně vzorového vložení hodnot. Tabulky jsou následně zobrazeny včetně celého obsahu. Následně jsou navrženy syntaxe LOGON a LOGOFF spouští pro zaznamenání přístupu do databáze, dále DDL spoušť, pro záznam příslušných operací nad daným modelem, a nakonec typová syntaxe tří spouští, které jsou nad každou jednotlivou tabulkou výchozího modelu, pro potřeby logování DML operací. Navržené řešení je poté opět graficky znázorněno pomocí ERD. Na závěr je navržena syntaxe balíčku

DBMS_XMLGEN pro export požadovaných údajů ve formátu XML. Po celkovém navržení řešení následuje část práce zaměřující se již na praktické ověření požadované funkcionality. Pro potřeby tohoto ověření jsou provedeny modelové DDL a DML operace, stejně tak jako operace připojení a odpojení uživatele od db.

Klíčová slova: Relačně databázová technologie, SQL, logování událostí, databázový trigger, referenční integrita

Logging of events in database structures

Abstract

Diploma thesis "Logging events in database structures" deals with the issue of using database triggers, to ensure the recording and storage of important information. This information mainly concerns users' access to the database - ie operations LOGON and LOGOFF, their activities within the schema, both in terms of the DDL commands - CREATE; ALTER; DROP and DML commands - INSERT; UPDATE; DELETE.

In the theoretical part, the most important basic information of relational database systems related to the proposed practical solution is mentioned. This information then moves from the basic and general level to the more specialized level. The basic principles and rules of the chosen database system, ie relational database system, are discussed very briefly. The main negatives and positives of this system are presented and the way of data arrangement in relational db systems is presented. Furthermore, the thesis focuses on the theoretical principles of normalization, including a brief introduction of individual normal forms, data integrity, where specific types of integrity constraints and ways of securing them are listed. Subsequently, the thesis deals with database triggers. Significant areas of their use, risks associated with their application and basic procedure of their compilation are given. Finally, the theoretical basis describes the XML format and how it can be generated in the relational db, specifically on the Oracle platform, based on the information stored in the appropriate db.

The practical part of this thesis offers a solution of event logging using PL / SQL database triggers and subsequent export of recorded information in XML format. The default model is designed and then illustrated by the ERD. The syntax of each table is shown, including sample insertion of values. The tables are then displayed including the entire contents. Further, the LOGON and LOGOFF triggers are designed to record access to the database, the DDL trigger to record the corresponding operations on the schema, and finally the type syntax of the three triggers which are above each individual table of the default model for logging DML operations is shown. The proposed solution is then again graphically represented by ERD. Finally, the syntax of the DBMS_XMLGEN package is proposed to export the required data in XML format. The overall design of the solution is

followed by a section of the thesis focusing on practical verification of the required functionality. For this purpose, model DDL and DML operations are performed, as well as connect and disconnect operations of a user from the db.

Keywords: Relational database technology, SQL, logging of events, database trigger, referential integrity

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V rámci této práce je probírána problematika automatizace procesu záznamu událostí a operací, jež v databázových evidencích nastávají. Konkrétně se tato práce zabývá těmi událostmi, které nastávají v důsledku činnosti uživatelů. Zejména se jedná o jejich přístup a aktivitu v rámci užívání databáze. Vzhledem k povaze tohoto záznamu není možné, aby byl prováděn manuálně, nýbrž je nutné vytvořit automatizované řešení. Pro tento problém je proto zcela zásadní téma databázových spouští neboli triggerů, s jejichž pomocí lze navrhnout a konstruovat řešení, jež zabezpečí logování událostí.

Celé řešení je nejprve sestaveno a následně testováno na platformě společnosti Oracle. Zaznamenávány jsou operace manipulační, operace definiční a události týkající se přístupu uživatelů k databázi. Tedy DML, DDL a LOGON a LOGOFF příkazy.

Obecně lze stanovit, že databázové triggery jsou velmi účelným a vhodným nástrojem pro automatizaci podnikových procesů. Především tedy těch procesů, které jsou repetitivní a snadné na realizaci, ale vzhledem k počtu potřebných iterací, jsou velmi časově náročné. Navíc díky skutečnosti, že nevyžadují příliš velkou pozornost pro svou realizaci, mnohdy může nastat situace, kdy jsou vkládány hodnoty v nežádoucí podobě, či jiným způsobem nekonzistentně, díky čemuž lze potenciálně narazit na riziko narušení integrity. Proto je dobré zvážit implementaci databázových spouští na ty procesy, které tímto způsobem neefektivně vytěžují lidské zdroje, či zvyšují riziko narušení konzistence a integrity. Neefektivně vytížené lidské zdroje mohou taktéž dále ve svém finálním důsledku zvyšovat kapitálové náklady, a tudíž i finanční zatížení pro daný podnik. Tyto finance a lidské zdroje mohou být v případě užití databázových spouští a automatizace využity jiným, účelnějším způsobem.

V současné době lze říci, že automatizace procesů a následně tedy i automatizace logování u velkých podniků je již prakticky až na výjimky standardem a potenciál automatizace je využit téměř na plno. U středních, a především menších podniků lze pozorovat trend, kdy jsou tyto praktiky sice zaváděny, nicméně stále ne na takové úrovni, jakožto právě u velkých podniků a například právě automatizované logování je zcela opomíjeno, díky čemuž je případný audit či dohledávání příčin konkrétních problémů značně komplikovanou záležitostí. Z tohoto důvodu nastává u složitých a vzájemně propojených podnikových aplikací, ke kterým intenzivně přistupuje velké množství uživatelů, situace, kdy je nutné nalézt řešení otázek, jak zaznamenávat přístup a aktivitu těchto uživatelů v rámci databáze, nad níž jsou tyto aplikace postaveny.

Zároveň lze v této době hovořit o exponenciálním růstu zaznamenávaných dat a díky tomu přímo úměrně i o rostoucích nákladech na správu a zpracování těchto dat. Z tohoto hlediska je implementace automatizace do budoucna nejen velmi důležitá, ale z praktického pohledu téměř nezbytná, pokud podnik chce rozšiřovat své pole působnosti a svou klientelu.

Od řešení, jež je v práci navrženo, lze očekávat, že bude vytvářet v reálném provozu velký objem dat a tabulky příslušící k danému řešení budou zatíženy redundancí dat. Tyto nevýhody jsou však u logování událostí běžným, očekávaným, a dokonce i do jisté míry požadovaným jevem, jelikož je požadováno, aby byl zaznamenán každý vklad hodnot, každé smazání atd. Vždy je vhodno po dosažení určitého objemu záznamů vytvořit zálohu a prostor v databázi uvolnit. Zálohovat lze data například po vytvoření exportu ve vhodném formátu.

Z těchto důvodů se snaží tato práce navrhnout a uvést řešení, jež je zároveň dostatečně obecné a snadné na implementaci, ale zároveň i natolik účelné, aby bylo prakticky použitelné a aby představilo možnosti, jež databázové spouště, a tudíž i automatizace procesů nabízí.

2 Cíl práce a metodika

2.1 Cíl práce

Tato diplomová práce si klade za hlavní cíl navrhnout účelné, přínosné, a především funkční řešení logování událostí prostřednictvím databázových spouští. Zvolené řešení musí být koncipováno s ohledem na požadavek minimalizace náročnosti, a to z důvodu značného množství vyvolání těchto spouští. Jednotlivé spouště by tedy z výpočetního hlediska měli být co možná nejméně náročné.

V rámci tohoto hlavního cíle nedefinovanými dílčími cíli jsou:

- nalezení možných způsobů realizace automatizovaného logování událostí
- identifikace potenciálních komplikací a bariér z hlediska aplikace logování událostí
- zmapování současného stavu implementace automatizovaného záznamu událostí u podnikatelských subjektů
- stanovit aktuální stav zvolené problematiky a vymezit relevantní požadavky na zlepšení
- na základě nalezených způsobů realizace zvolit vhodný postup a navrhnout řešení odpovídající stanoveným požadavkům a respektující identifikované překážky a limity

2.2 Metodika

V rámci zpracování této diplomové práce je užita metodika, jenž vyplývá ze studia a následné analýzy jak tištěné odborné literatury, včetně příslušné dokumentace jazyka PL/SQL firmy Oracle, tak i dalších důvěryhodných a relevantních zdrojů informací, jakož například i již realizovaných řešení. Literatura a další odborné informační zdroje jsou vybírány s ohledem na téma a cíle této diplomové práce, stěžejní jsou proto ty zdroje, které vhodně a dostatečně popisují a vysvětlují problematiku databázových triggerů, exportu dat a souvisejících východisek.

Pro tuto práci budou stěžejní zejména ty principy, východiska, techniky a metody, které se týkají relačního databázového řešení, v kontextu zajištění automatizovaného záznamu událostí a jeho následného exportu. Především budou proto uvedeny informace zaměřené na databázovou normalitu, datovou integritu, typy událostí v DB, databázové triggery a formát XML.

Dle studia a analýzy odborné literatury a dalších informačních zdrojů, budou vybrány informace, jež jsou pro zvolené téma zásadní a přínosné, následně na jejich základě bude sepsána literární rešerše v rozsahu odpovídajícímu řešené problematice.

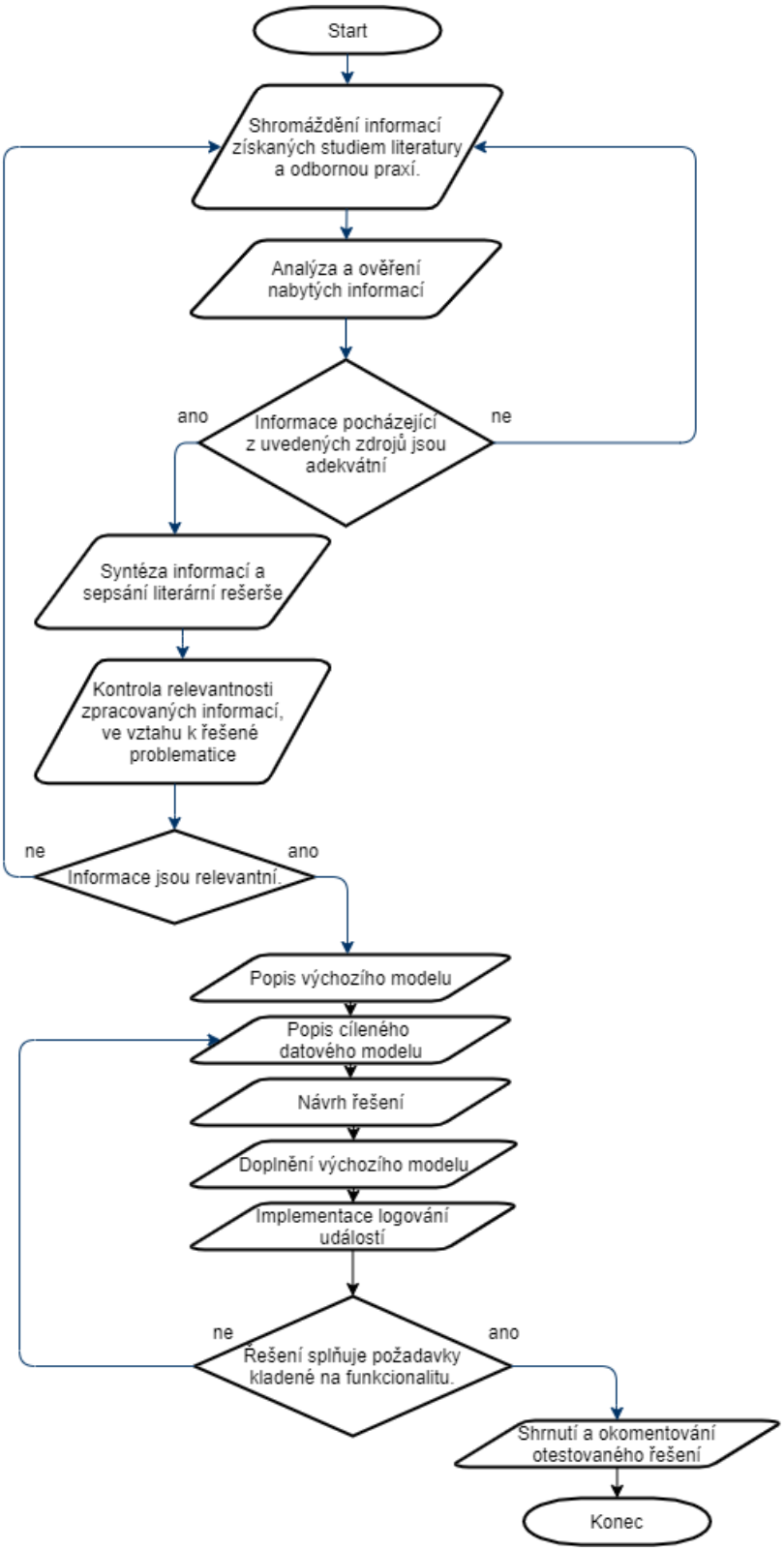
Praktická část práce bude zaměřena na prezentaci navrhovaného řešení a následné otestování adekvátnosti a správnosti jeho funkcionality. Řešení bude kompletně koncipováno na platformě PL/SQL, konkrétně od společnosti Oracle. Realizované řešení bude vycházet z identifikovaných požadavků a předpokladů, které jsou na toto řešení kladeny. Výsledný přínos je demonstrován v rámci ověření navrhovaného řešení.

Závěry této diplomové práce budou formulovány na základě syntézy teoretických podkladů a výsledků dosažených v rámci praktické části. Dosažené výsledky budou společně s učiněnými závěry následně zobecněny pro další případná použití.

Zvolená metodika vedoucí ke splnění výše vytyčených cílů se skládá z posloupnosti následujících kroků a je následně schematicky znázorněna diagramem na obrázku č.1.

- Na základě studia dostupných odborných zdrojů různých forem, budou shromážděny potřebné informace.
- Získané informace budou analyzovány a ověřeny, zda jsou aktuální, správné a v současné době stále přínosné.
- Pokud nebudou tyto informace shledány z těchto hledisek pro zpracovávané téma adekvátní, poté probíhá proces shromáždění informací a následná analýza znovu, dokud nebudou získány vhodné informace.
- Jestliže tyto informace budou shledány z těchto hledisek pro zpracovávané téma adekvátní, probíhá následně syntéza informací, jejímž výstupem je literární rešerše.
- Tato literární rešerše dále prochází kontrolou, zda je v dané formě a struktuře, s danými informacemi relevantní ve vztahu k řešené problematice.
- Pakliže literární rešerše není v těchto bodech shledána jakožto relevantní, poté se postup zpracování vrací do výchozího bodu – jsou tedy opětovně shromažďovány informace na základě studia odborných zdrojů, dle kterých musí být v rámci následujících kroků opět literární rešerše přepracována.
- Jeli literární rešerše v uvedených bodech relevantní, poté je zahájena realizace vlastního řešení, které prochází postupně 5 dílčími kroky:
- Popis výchozího modelu;
- Popis cíleného datového modelu
- Návrh řešení
- Doplnění výchozího modelu
- Implementace logování událostí
- Po těchto krocích je vlastní řešení otestováno a vyhodnoceno, zda splňuje požadavky, jež jsou kladeny na dané řešení a jeho funkcionalitu v rámci cílů této práce.
- Pokud požadavky nejsou splněny, vrací se postup řešení do kroku popisu cíleného datového modelu, kde dochází k re-specifikaci zvoleného řešení.
- Jsou-li požadavky splněny, poté je dané vlastní otestované řešení shrnuto, okomentováno a zjednodušeno pro případná další užití.

Obrázek č. 1 – Postup řešení



Zdroj: Autor

3 Teoretická východiska

Prakticky každé podnikové prostředí využívá k uchovávání dat databázové systémy. Tyto systémy slouží k uchovávání a distribuování dat, která jsou pro dané subjekty klíčová. Vzhledem k povaze těchto dat je nutné zvolit správný typ databázového systému, a to jak z hlediska hardwarové náročnosti, tak z hlediska poskytovaných funkcionalit. Ať už se jedná o databáze těch největších subjektů, či těch nejmenších, vždy je základním požadavkem rychlost zpracování dat a rychlost realizace zadaných příkazů.

Z tohoto hlediska je v současné době stále nejužívanějším a nejoblíbenějším typem datového uložiště relační databáze, což dělá z relačního modelu zdaleka nejdůležitější standard databázového oboru. Z tohoto důvodu je následně uvedena pouze teorie týkající se relačního datového modelu, a to konkrétně společnosti ORACLE. Veškeré zdrojové kódy jsou proto uvedeny v dotazovacím jazyce PL/SQL. [1]

3.1 Zvolený typ databázového systému

Pro relační databáze je zcela neopominutelná studie od doktora E. F. Codda s označením: „A Relational Model of Data for Large Shared Data Banks“, neboli česky: „Relační model dat pro obsáhlé datové banky“. Tato publikace pochází již z roku 1970, ale principy, jež jsou v ní popsány, jsou využívány dodnes a stále z nich lze vycházet. Jazyky SQL a následně tedy i PL/SQL jsou koncipovány právě s ohledem na požadavky a principy relačního databázového modelu. Tyto klíčové principy jsou uvedeny ve zmiňované studii. Zejména stojí za zmínku 12 pravidel E.F.Codda. [2]

3.2 Způsob uspořádání dat v relačním databázovém systému

Databázový systém dr. Codda je založen na matematické koncepci relační algebry, kterou aplikuje k „rozdrobení“ informací a dat na skupinu množin a jejich souvisejících podmnožin. Tyto množiny zde lze označit za tabulkovou strukturu, kde se každá tabulka skládá ze skupiny sloupců, které jsou atributy jednotlivých záznamů a skupiny řádků, jež obsahují konkrétní hodnoty jednotlivých atributů pro daný záznam. Skupinu sloupců označujeme jakožto pole a sadu polí v řádku za záznam. Názorná ukázka takové tabulky je uvedena níže v tabulce č.1:

Tabulka č. 1 – Příklad relační tabulky

ID	NAME	SURNAME	WAGE	GROUP
1	Anthony	Travis	1500	A
2	Peter	Southbridge	1400	B
3	Julia	Curtis	1300	C

Zdroj: SKŘIVAN, J.: Databáze a jazyk SQL

Každý řádek je tedy záznamem a pro každý záznam platí určitá pravidla. Tato pravidla se souhrnně označují jako Normální formy**.** [18]

3.3 Normalizace

Pod tímto pojmem rozumíme způsob nebo proces uspořádání dat za účelem odstranění redundance dat a případné nekonzistence. Normalizace je proces, který zajistí, že každá relace či tabulka má pouze jedno téma. Tohoto stavu je dosaženo po rozkladu relace či tabulky s více tématy na sadu vícero relací či tabulek, kde každá má pouze jedno téma. [1]

Za význačné přednosti a přínosy normalizace se pokládá:

a) Snížení nároků na úložiště

   Bez normalizace se v databázích mohou vyskytovat duplicitní data, díky kterým se ukládá větší množství dat, než je nutné. V důsledku je vyžadováno více místa na discích a více paměti. Odstranění či omezení redundance tedy vede k efektivnějšímu využití prostředků, které jsou databázovému serveru k dispozici. [8] [18]

b) Integrita dat

   Pokud se určitá hodnota opakovaně vyskytuje ve vícero tabulkách, může se snadno stát, že v některém z těchto výskytů dojde při zadávání či aktualizaci k překlepu nebo že hodnota bude aktualizována pouze v jedné z tabulek, kde se tato hodnota vyskytuje. Tomuto jevu se poté říká nekonzistence dat. [6]

c) Udržitelnost

Veškerá pravidla normalizace se soustředí na údržbu relací – vztahů a díky tomu mají tabulky jasný, logický a definovaný smysl. Do jedné tabulky jsou například ukládány informace o zaměstnancích a do druhé informace o pracovištích, kde každý jednotlivý zaměstnanec pracuje. Takto uspořádaná data se udržují daleko snáze, než kdyby byla všechna nahromaděna a smíchána do jediné obrovské tabulky. [8] [18]

Důležité je dodat, že i normalizovaná databáze, jež bude označována za dobře navrženou, může mít negativní vliv na výkon. Databázový server totiž potřebuje na spojování tabulek určitý čas a výkon. Tento negativní vliv normalizace lze omezit správným indexováním, správným použitím klíčů, a také filtrováním takovým způsobem, aby databázový server prováděl ty nejsložitější úkoly s co nejmenším počtem záznamů. Případně lze opět denormalizovat, což však musí být velice striktně odůvodněno. Obecně je taktéž doporučeno normalizovat pouze do třetí či do BC normální formy. [6] [7]

3.3.1 Normální formy

V praxi existuje 7 používaných a respektovaných normálních forem, nejčastěji je užíváno prvních čtyř. Vždy lze o databázové tabulce tvrdit, že je v určité normální formě, pokud splňuje dané požadavky na ni kladené. Každá následující normální forma musí navíc splňovat požadavky všech předcházejících normálních forem. [3] [18] [19]

Od roku 1972 jsou E. F. Coddem definované tři úrovně normalizace:

I. 1NF až 3NF – založené na funkčních závislostech mezi atributy relace.
II. Boyce-Coddova NF (BCNF) – důslednější 3NF.
III. 4NF a 5NF – založené na vícehodnotových závislostech a join závislostech.

3.4 Atribut & Doména

Jak vyplývá z předchozích kapitol, data a informace jsou v databázi ukládány v naprosté většině případů do několika tabulek. Mezi těmito tabulkami jsou relační vazby.
Z hlediska těchto vazeb jsou nejdůležitější pojmy:

- **Atribut**
  Pro každý záznam definovaný hodnotou daného sloupce – atributu.
- **Doména**
  Určuje množinu hodnot, které daný atribut může pro každý jeden záznam nabývat. Z tohoto hlediska jej lze označit za definiční obor daného atributu. Dále můžeme definovat dva druhy domén – doménu prostou a doménu kompozitní.
  Prostá doména je vázána pouze k atributu, jež je atributem jednoduchým – není složen z více částí.
  Naopak doména kompozitní je vázána k atributu, jenž je kartézským součinem hodnot vícero jednoduchých atributů. Ideálním příkladem této domény je datum, jež se skládá z hodnot dne, měsíce a roku.
- **Relace**
  Hovoříme o skupině vazeb a vztahů mezi členy různých domén, které jsou vyjádřeny pomocí tabulek, které jsou provázány prostřednictvím klíčů.
- **Schéma relace**
  Vyjádřeno jakož: název relace + názvy atributů.
  Schéma je v čase invariantní.
- **Primární a cizí klíč**
  Může být tvořen jedním či více atributy, jehož hodnota (či kombinace hodnot v případě klíčů složených) jednoznačně identifikuje n-tici relace.

[6]

3.4.1 Význam domén z hlediska relačních operací

Domény jsou zcela zásadní zejména při definování vztahů mezi tabulkami. Pokud budeme sestavovat závislost mezi tabulkami, u kterých budeme dále předpokládat opakovanou realizaci operací spojování a porovnávání, poté je nezbytné stanovit stejné domény pro příslušné atributy v dotčených tabulkách. [6]

3.5 Integrita dat

Je velmi důležité, aby data dodržovala a splňovala předem definovanou sadu pravidel, která jsou definována databázovým administrátorem, či developerem aplikace. Tímto způsobem lze efektivně kontrolovat, jaký typ dat a v jakém formátu bude vložen do databáze.

Tato pravidla souhrnně nazýváme omezení a to proto, jelikož určitým způsobem vymezují, jaké hodnoty jsou akceptovatelné a které naopak nejsou. Díky tomu lze garantovat přesnost a konzistenci dat uložených v databázi. Bez těchto omezení by totiž mohl kdokoliv vkládat do tabulek takové hodnoty, které by mohly právě konzistenci a přesnost dat ohrozit.

Například lze uvést případ, kdy jedna osoba bude vkládat telefonní číslo ve tvaru: „741 852 963“ a druhá ve tvaru: „741852963“. Již tato situace má za následek narušení konzistence, a právě užití integritních omezení umožňuje této a podobným situacím předejít. Nekonzistentní data mohou mít například za následek obtížné získávání požadovaných dat a jejich následné porovnávání.

Integrita dat je tedy pojmem zastřešujícím garanci přesnosti a konzistence dat nacházejících se v db. [3] [7] [17]

3.5.1 Typy datové integrity

- **Nulové pravidlo**
  Definováno samostatně na sloupec, kde umožňuje či naopak zakazuje vkládání či aktualizaci na nulovou hodnotu.
- **Unikátní sloupcové hodnoty**
  Specifikovány na sloupci (či skupině sloupců), kde umožňuje vložení či aktualizaci na striktně unikátní hodnotu pro daný sloupec.
- **Hodnota primárního klíče**
  Stanovena pro jeden či více (u složeného klíče) sloupců, kde umožňuje unikátní identifikaci každého řádku tabulky.
- **Pravidlo referenční integrity**
  Definováno na klíči (sloupec či sada sloupců) jedné tabulky, kde zaručuje, že hodnota cizího klíče této tabulky odpovídá hodnotě primárního klíče tabulky, se kterou souvisí, respektive na jejíž hodnotu se odkazuje.

Referenční integrita dále obsahuje sadu pravidel, která při aplikaci stanovují, jaký typ manipulace s daty je přípustný pro odkazované hodnoty, a jak tyto modifikace ovlivňují závisející hodnoty:

- **Restrict**
  Znemožnění aktualizace či smazání odkazovaných dat
- **Set to Null**
  Jakmile jsou odkazovaná data smazána či aktualizována, všechna související závislá data jsou nastavena na nulovou hodnotu neboli „NULL“.
- **Set to Default**
  Pokud jsou odkazovaná data smazána či aktualizována, všechna související závislá data jsou nastavena na výchozí hodnotu.
- **Cascade**
  Pokud jsou odkazovaná data smazána či aktualizována, všechna související závislá data jsou korespondujícím způsobem aktualizována. Tedy změny závisející tabulky jsou kaskádovitě dovedeny do odpovídající výchozí tabulky. Pokud bude smazán odkazovaný řádek, poté všechny navázané řádky závislé tabulky jsou také smazány.
- **No Action**
  Znemožnění dané aktualizace či smazání. Zadaný příkaz se neprovede a uživatel bude upozorněn, že se danou akcí dopouští narušení referenční integrity. Od pravidla „Restrict“ se toto pravidlo liší zejména v tom, že kontrola probíhá na konci příkazu, nebo na konci transakce, pokud je omezení porušeno.
  ORACLE užívá toto pravidlo jakožto výchozí ze všech výše uvedených.

- **Komplexní integritní kontrola**
  Souhrnné označení pro uživatelem definované sloupcové pravidlo.
  Může se týkat jednoho sloupce, ale i skupiny sloupců.

Umožňuje či naopak zakazuje vložení/ aktualizaci/ mazání řádku na základě hodnoty v něm obsažené pro daný sloupec či skupinu sloupců.
[3] [7] [13] [17]

3.5.2 Metody zajištění datové integrity

Všechny typy datové integrity uvedené v předchozí kapitole lze implementovat pomocí integritních omezení v rámci příkazu CREATE TABLE či pomocí databázových spouští.

Integritní omezení jsou deklarativní metodou definice pravidla pro sloupec dané konkrétní tabulky. ORACLE podporuje následující omezení (vypsána v pořadí odpovídajícím předchozí kapitole):

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

[6]

3.5.2.1 Implementace „NOT NULL“

Všechny sloupce tabulky jsou vždy obecně přednastaveny takovým způsobem, aby byl umožněn vklad nulových hodnot. „NOT NULL“ omezení poté vyžaduje, aby tato situace v daném sloupci nenastala. Názorně v tabulce zaměstnanců neboli tabulce č. 2 níže je tohoto omezení užito na sloupec „JMZAM“ – tedy jméno zaměstnance v tabulce „ZAM“. [20]

Tabulka č. 2 – Tabulka ZAM – „NOT NULL“ [12]

Tabulka ZAM

CZAM	JMZAM	POZ	MAN	NAST	PLAT	CODD
7435	FIALA	CEO		10-DEC-90	55 000	10
7436	URBAN	V_PRODEJE	7435	12-JUN-92	45 000	20
7437	SVOBODA	MANAGER	7436	20-MAY-95	40 000	20
7438	MAREK	PRODEJCE	7437	18-FEB-93	28 000	20

Aplikováno integritní omezení NOT NULL

Absence integritního omezení NOT NULL

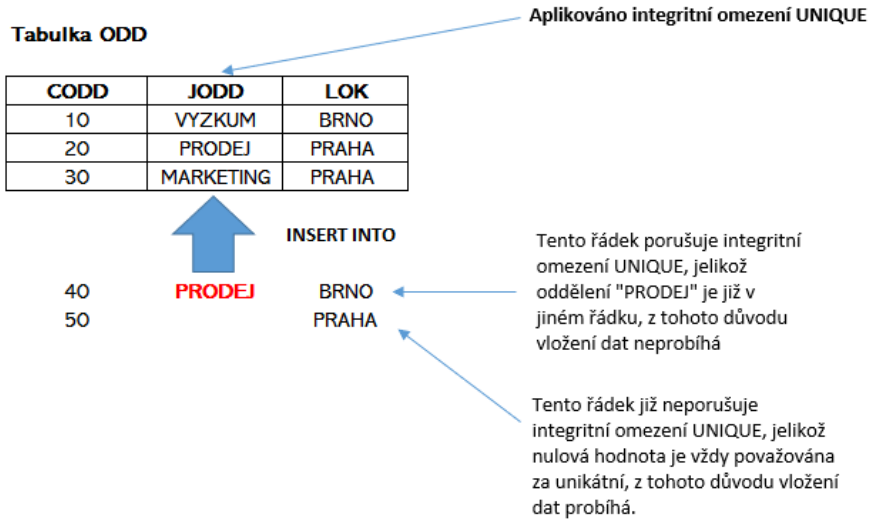
Aplikováno integritní omezení NOT NULL

Zdroj: Data Integrity. ORACLE, 2019.

3.5.2.2 Implementace „UNIQUE“

Pokud je vyžadováno, aby žádné dva řádky v tabulce nebyly shodné, poté je nutno užít omezení „UNIQUE“. Názorně v tabulce oddělení fiktivní společnosti – tabulka č. 3, je toto omezení užito na sloupec „JODD“, neboli název oddělení. Následně jsou znázorněna dvě vložení dat do tabulky, kdy první bude odmítnuto z důvodu narušení integritního omezení „UNIQUE“ nad sloupcem „JODD“, jelikož oddělení s názvem „PRODEJ“ je již v tabulce obsaženo. Naopak sloupec „LOK“ neobsahuje toto omezení, a proto druhé vložení již proběhne bez problému. Zároveň je vhodno upozornit, že unikátnost záznamu není žádným způsobem porušena vložením nulové hodnoty.
Pokud by náš záměr byl takový, aby v daném sloupci byly hodnoty zároveň unikátní a nenulové, bylo by nutno aplikovat obě výše uvedená omezení současně. Tato situace v praxi taktéž může nastat, jelikož nulová hodnota je vždy považována za unikátní, a to i tehdy, pokud je zadána pro vícero řádků, tudíž záznamů. [12]

Tabulka č. 3 – Tabulka ODD – „UNIQUE“ [20]



Zdroj: VALENTA, M. Integritní omezení (IO). Praha: FIT ČVUT, 2011.

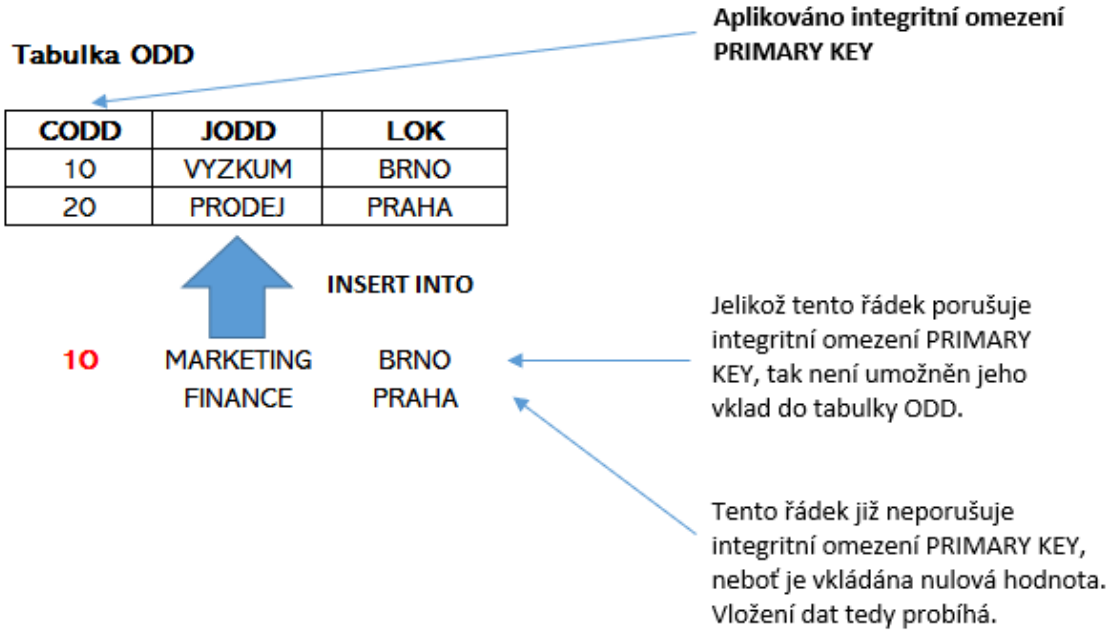
3.5.2.3 Implementace „PRIMARY KEY“

Každá tabulka v databázi může mít nanejvýš jednou aplikované omezení „PRIMARY KEY“. To však neznamená, že se primární klíč vždy skládá pouze z jednoho sloupce. Hodnoty v množině jednoho a více sloupců, které jsou subjekty tohoto omezení jsou unikátním identifikátorem daného řádku.

Toto omezení garantuje, že žádné dva řádky tabulky nemají stejnou hodnotu ve specifikovaných sloupcích. Zároveň na rozdíl od omezení „UNIQUE“ nejsou akceptovány nulové hodnoty, tedy každý specifikovaný sloupec musí být obsazen hodnotami.

V tabulce oddělení č.4 je názorně uvedena aplikace primárního klíče včetně vložení dvou záznamů, z nichž první porušuje podmínku unikátnosti a druhý porušuje podmínku nenulovosti atributu, jež je primárním klíčem. Tedy ani jeden z uvedených vkladů není proveden. [1]

Tabulka č. 4 – Tabulka ODD – „PRIMARY KEY“ [20]



Zdroj: VALENTA, M. Integritní omezení (IO). Praha: FIT ČVUT, 2011.

3.5.2.4 Implementace referenční integrity s využitím „FOREIGN KEY“

Relační databáze běžně obsahují odlišné tabulky, jejichž data jsou však logicky závislá a tuto skutečnost je proto třeba i funkčně implementovat. K tomu lze využít integritních omezení, či databázových spouští.

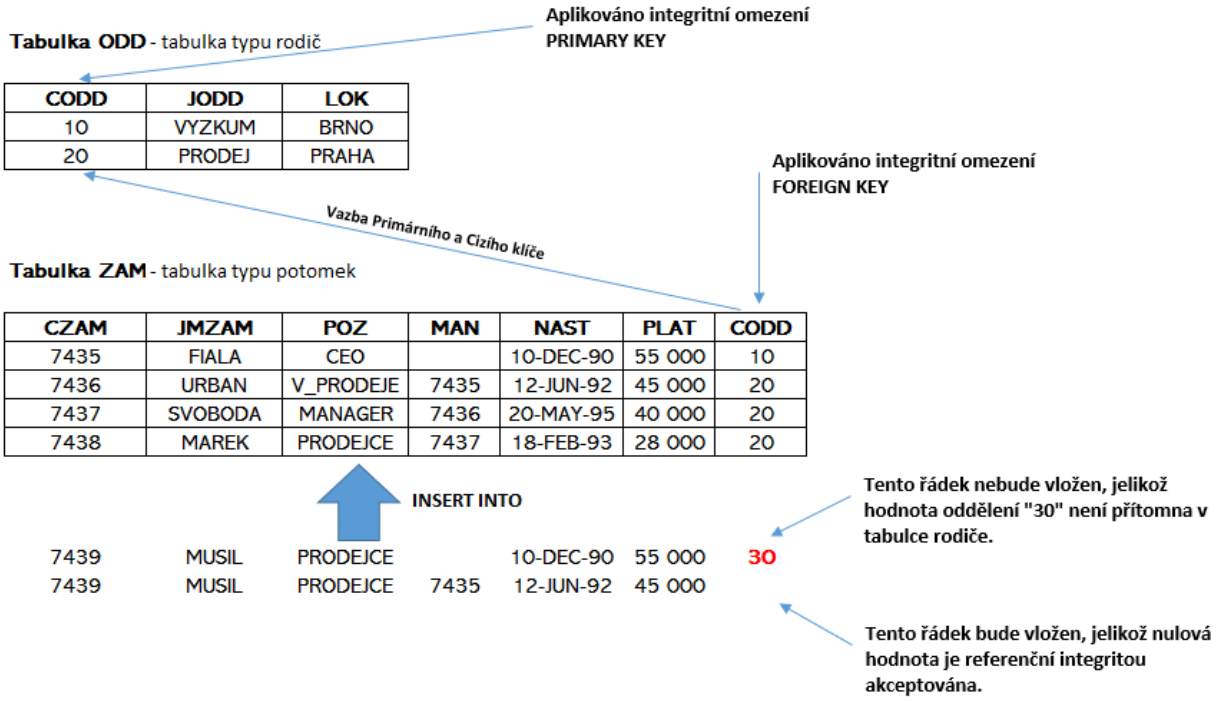
V rámci užití cizího a primárního klíče pro zajištění referenční integrity tabulek relační databáze je klíčová existence shodných sloupců v těchto tabulkách. Cizí klíč je sloupcem nebo sadou sloupců začleněných v definici omezení nad tabulkou potomka, které odkazuje na primární klíč neboli odkazovaný klíč tabulky rodiče neboli předka.

Takto stanovené omezení striktně vyžaduje, aby každý řádek tabulky potomka měl pro hodnotu obsaženou v cizím klíči odpovídající hodnotu primárního klíče v tabulce předka. Pokud je tomuto požadavku zcela vyhověno, poté není porušena referenční integrita.

Názorně demonstrováno na schéma tabulek č. 5 – Referenční integrita, kde jsou vytvořeny tabulky oddělení – „ODD“ a zaměstnanců – „ZAM“. Tabulka oddělení je zde rodičovskou tabulkou a tabulka zaměstnanců je tabulkou typu potomek. Primární i cizí klíč je zde definován nad atributem čísla oddělení – „CODD“, jež je tedy v tabulce „ODD“ primárním klíčem a v tabulce „ZAM“ klíčem cizím.

Následně jsou učiněny dva pokusy o vložení dat do tabulky EMP. První vklad porušuje referenční omezení, jelikož číslo oddělení „10“ se nevyskytuje v tabulce předka – neexistuje odpovídající primární klíč, proto není takový vklad umožněn a nedochází k jeho provedení. Naopak druhý vklad, ačkoliv neobsahuje žádné číslo oddělení, tak je umožněn, jelikož cizí klíč může zůstat nevyplněný, pokud bychom chtěli, aby bylo striktně vyžadováno vyplnění tohoto atributu při vkládání do tabulky potomka, poté by muselo být při definici tabulky taktéž užito omezení „NOT NULL“. [1] [9]

Tabulka č. 5 – Referenční integrita [9]

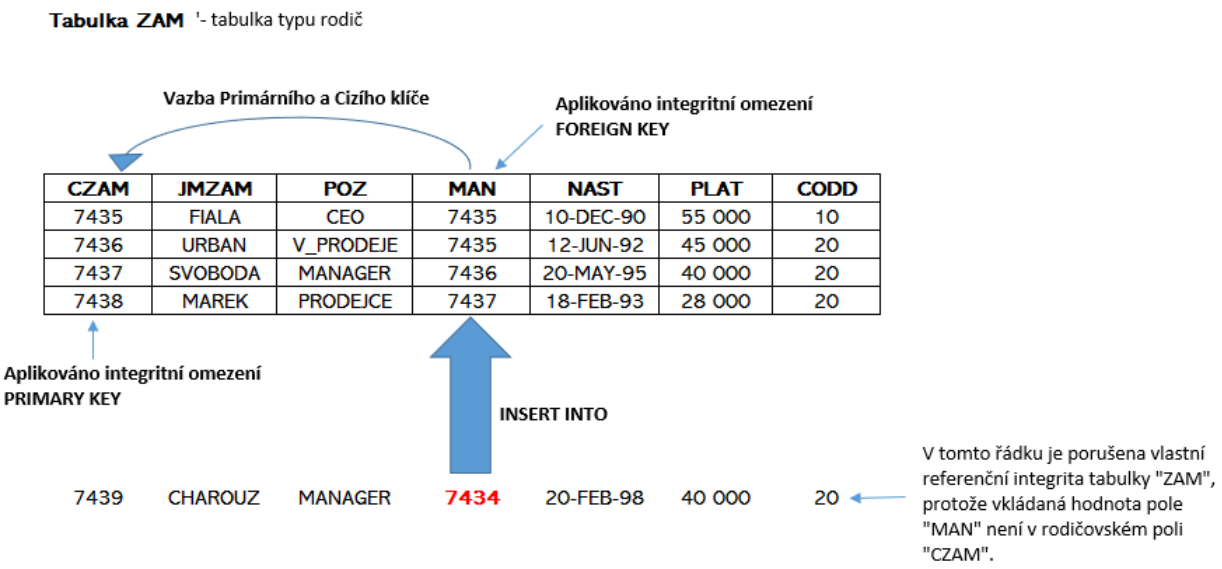


Zdroj: BROOK, C.: Data protection 101, 2019.

3.5.2.5 Vlastní referenční integrita

Referenční integrita je taktéž aplikovatelná nad jedinou tabulkou, tabulka předka a potomka je zde následně jednou a tou samou tabulkou. Omezení, jež je takto definováno, se nazývá vlastní referenční integrita a je jí dosaženou pomocí cizího klíče odkazujícího se na vlastní primární klíč v té samé tabulce. Díky tomuto postupu je zajištěno, že nově vložený záznam může v poli pro atribut cizího klíče obsahovat pouze hodnotu, která se již v tabulce vyskytuje v atributu primárního klíče. Navíc však může obsahovat taktéž nulovou hodnotu, proto je vhodno doplnit omezením „NOT NULL“. Názorně uvedena praktická ukázka nad tabulkou č. 6 – Vlastní referenční integrita.

Tabulka č. 6 – Vlastní referenční integrita [12]



Zdroj: Data Integrity. ORACLE, 2019.

Ve výše uvedené tabulce zaměstnanců č. 6 tedy vidíme atribut číslo zaměstnance - „CZAM“, jež je primárním klíčem, který je odkazován atributem čísla manažera – „MAN“, což je právě cizí klíč. V této praktické ukázce je proto dosaženo stavu, kdy může být vložen pouze takový zaměstnanec, jenž bude mít takového nadřízeného, který již v dané tabulce je zadán, ale nemusí být bezpodmínečně ve stejném řádku. Díky tomu nemůže být vložena chybné číslo nadřízeného. Opět by bylo vhodno doplnit o integritní omezení „NOT NULL“. [12]

3.5.2.6 Implementace „CHECK“

Omezení tohoto typu je opět aplikováno v rámci příkazu „CREATE TABLE“. Je využíváno pro vytváření velmi specifických integritních pravidel, které vyžadují, aby byla splněna určitá podmínka pro každý řádek tabulky. Pokud DML příkaz zapříčiní negativní vyhodnocení této podmínky, poté se takový DML příkaz neprovede.

Předem je nutno zmínit, že toto omezení lze užít jen za určitých podmínek. Zejména musí být toto omezení koncipováno jakožto booleovský výraz, který je vyhodnocován za využití hodnot v řádku, který je vkládán či aktualizován. Dále není podporováno užívání pod dotazů, sekvencí, SQL funkcí: „SYSDATE“; „UID“; „USER“; „USERENV“ a ani pseudo sloupců „LEVEL“ nebo „ROWNUM“.

Dalším důležitým bodem je možnost násobného užití tohoto omezení nad jediným sloupcem. ORACLE žádným způsobem neomezuje množství omezení „CHECK“, která jsou aplikována nad jedním sloupcem. Pokud však budeme vytvářet takovéto složité násobné omezení, poté je nutné věnovat dostatečné množství pozornosti jejich účelu, jelikož se mohou snadno dostat do konfliktu. [12] [20]

3.5.3 Postup vyhodnocení integritních omezení

Postup vyhodnocování integritních omezení by měl být znám jak administrátorům či samotným programátorům, tak samozřejmě i uživatelům. Bez tohoto povědomí by bylo náročné usuzovat, jaké typy operací jsou při přítomnosti omezení povoleny.

Nelze předpokládat pořadí, v kterém budou jednotlivá omezení vyhodnocována. Zcela zásadní je však moment, kdy jsou vyhodnocována, a to až po vykonání celého příkazu.

Pro názornost bude využito praktické ukázky – konkrétně z tabulky č. 6 – Vlastní referenční integrita, jež bude zjednodušena pouze na sloupce čísla zaměstnance – „CZAM“ a čísla manažera – „MAN“.

Předpokládejme stav, kdy je již integritní omezení aplikováno a tabulka vytvořena, není však ještě naplněna. Jedná se tudíž o situaci, kdy je žádáno vložení řádků – zaměstnanců. Je zde však omezení, že nemůže být vložen takový záznam, jež by obsahoval atribut s číslem manažera, které neodpovídá žádnému číslu zaměstnance. Pokud budeme vycházet ze znalosti uvedených informací, nabízí se tři řešení:

1. Pokud není nad sloupcem „MAN“ aplikováno též omezení „NOT NULL“, poté lze vložit řádek zaměstnance s prázdným atributem manažera, je tedy vyhověno omezení, jelikož cizí klíč umožňuje nulové hodnoty.
2. Stejná hodnota může být v řádku vložena jak do primárního klíče, tak i do cizího klíče, tato možnost názorně dokazuje, že ORACLE provádí kontrolu integritních omezení až po provedení celého příkazu.
3. Poslední možností je užití příkazu INSERT pro víceřádkové vložení záznamů. Například se může jednat o vnořený příkaz SELECT uvnitř příkazu INSERT, díky kterému mohou být vloženy dva záznamy, které jsou vzájemně propojeny. Prakticky takto mohou být vloženy řádky, kdy první bude obsahovat číslo zaměstnance 1 a manažera 2 a druhý řádek číslo zaměstnance 2 a manažera 1. Touto možností je opět dokázáno, že vyhodnocení omezení je odloženo až po vykonání celého příkazu – tedy po vložení obou řádků. [12]

Nyní je tedy tabulka ve stavu znázorněném tabulkou č. 7. – Naplněná tabulka zaměstnanců.

Tabulka č. 7 – Průběh kontroly vlastní referenční integrity [12]

CZAM	MAN
110	
111	110
112	111

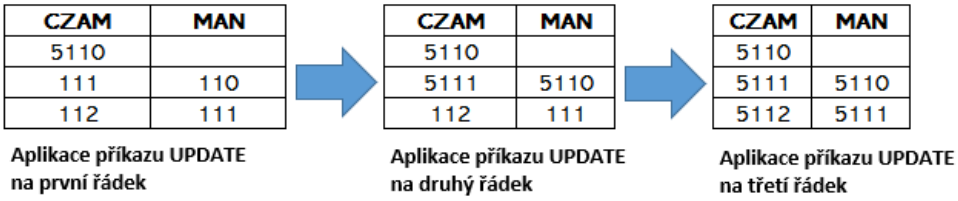
Zdroj: Data Integrity. ORACLE, 2019.

Představme si zde situaci, kdy například dojde k organizační změně uvnitř společnosti a následkem této změny je potřeba upravit všechna čísla zaměstnanců a manažerů takovým způsobem, aby začínala číslem 5. Pro tuto úpravu je užito následujícího příkazu:

```
UPDATE   zam
    SET  czam = czam + 5000,
         man = man + 5000;
```

Tento příkaz žádným způsobem nenarušuje vlastní referenční integritní omezení, jelikož ORACLE nejdříve aktualizuje první řádek, následně druhý řádek, poté třetí a až nakonec po dokončení celého příkazu UPDATE vyhodnocuje toto omezení – viz tabulka č. 8 – Průběh kontroly vlastní referenční integrity.

Tabulka č. 8 – Průběh kontroly vlastní referenční integrity [12]



Zdroj: Data Integrity. ORACLE, 2019.

Nakonec je nutno doplnit, že stejný mechanismus ověření integritního omezení je použit nejen u DML operací „INSERT“ a „UPDATE“, ale i u operace – „DELETE“. Zároveň je stejný mechanismus ověření aplikován taktéž pro všechny typy integritních omezení. [12]

3.6 Typy událostí v databázích a jejich členění

Předtím, nežli bude uveden obecný popis triggerů neboli spouští, jejich struktury, implementace a následně užití pro logování databázových událostí, je vhodné představit, co se pod pojmem databázové události skutečně nachází.

Pro pochopení jednotlivých typů událostí, je nejprve vhodné zmínit, jakým způsobem lze členit samotný jazyk SQL, jelikož od tohoto členění se následně odvíjí i typy databázových událostí.

Jazyk SQL lze rozdělit do čtyř hlavních částí, mezi kterými jsou:

- **„Data Definition Language“ - DDL**
  Jazyk pro definici dat slouží především k sestavení databázového schéma, tedy k vytvoření a upravení struktury databázových objektů.
  Příkazy tohoto jazyka jsou:
  „CREATE“; „DROP“; „ALTER“; „TRUNCATE“; „COMMENT“; „RENAME“.

- **„Data Manipulation Language“ – DML**
  Jazyk užívaný pro manipulaci s daty. Spadají do něho ty příkazy, které umožňují operace vkládání, výběru a zobrazení, aktualizaci a mazání dat.
  Těmito příkazy jsou:
  „SELECT“; „INSERT“; „UPDATE“; „DELETE“.

- **„Data Control Language“ – DCL**
  Tento jazyk umožňuje správu a řízení oprávnění k přístupu a manipulaci s databází.
  DCL příkazy:
  „GRANT“; „REVOKE“.

- **„Transaction Control Language“ – TCL**
  Pomocí TCL je možno realizovat a obsluhovat transakce.
  Mezi příkazy tohoto jazyka řadíme:
  „COMMIT“; „ROLLBACK“; „SAVEPOINT“; „SET TRANSACTION“.

Samotné události lze potom členit na základní dvě velké skupiny, které je rozdělují na základě skutečnosti, zda se daná událost vztahuje na individuální tabulky, případně řádky, či zda se vztahuje na celou instanci nebo celé databázové schéma.

Události, které se vztahují na celou instanci označujeme jako databázové a ty, které se vztahují na specifickou tabulku jako uživatelské. [4] [11]

3.6.1 Databázové události

Databázové události jsou charakteristické již zmiňovanou vazbou na celou instanci, konkrétně se jedná o: „STARTUP“; „SHUTDOWN“; „DB_ROLE_CHANGE“; „SERVERERROR“. Je taktéž nutno stanovit určitá omezení, která jsou kladena na triggery, jež jsou s těmito událostmi provázané.

Spouště, jež jsou navázány na události „STARTUP“ a „SHUTDOWN“, tedy zapnutí a vypnutí instance, musí být definované nad databázovou instancí. Jak je již na první pohled zřejmé, „STARTUP“ spouště nelze kombinovat s atributem „BEFORE“, stejně jako „SHUTDOWN“ spouště nelze kombinovat s atributem „AFTER“. Pokus o vytvoření takových spouští končí návratem chybových hlášek „ORA-30500“ a „ORA-30501“ které upozorňují o neplatné kombinaci typu spouště a atributu.

Spouště, jež využívají událostí typu „SERVERERROR“, které lze navázat na určité číslo chyby, mohou být definovány jak nad instancí, tak i nad databázovým schématem. Podobně jako u výše zmíněných spouští, i zde je omezení na kombinaci s atributem „BEFORE“. Pokus o vytvoření takové spouště navrací chybu s kódem „ORA-30500“. [14]

3.6.2 Uživatelské události

Tento druh je specifický propojením se specifickou tabulkou, z toho vyplývá, že se jedná zejména o události jazyků DML – „DELETE“ / „INSERT“ / „UPDATE“, DDL – „CREATE“ / „ALTER“ / „DROP“ a DCL – „GRANT“ / „REVOKE“. Posledním typem událostí, které spadají pod uživatelské jsou „LOGON“ a „LOGOFF“. [14]

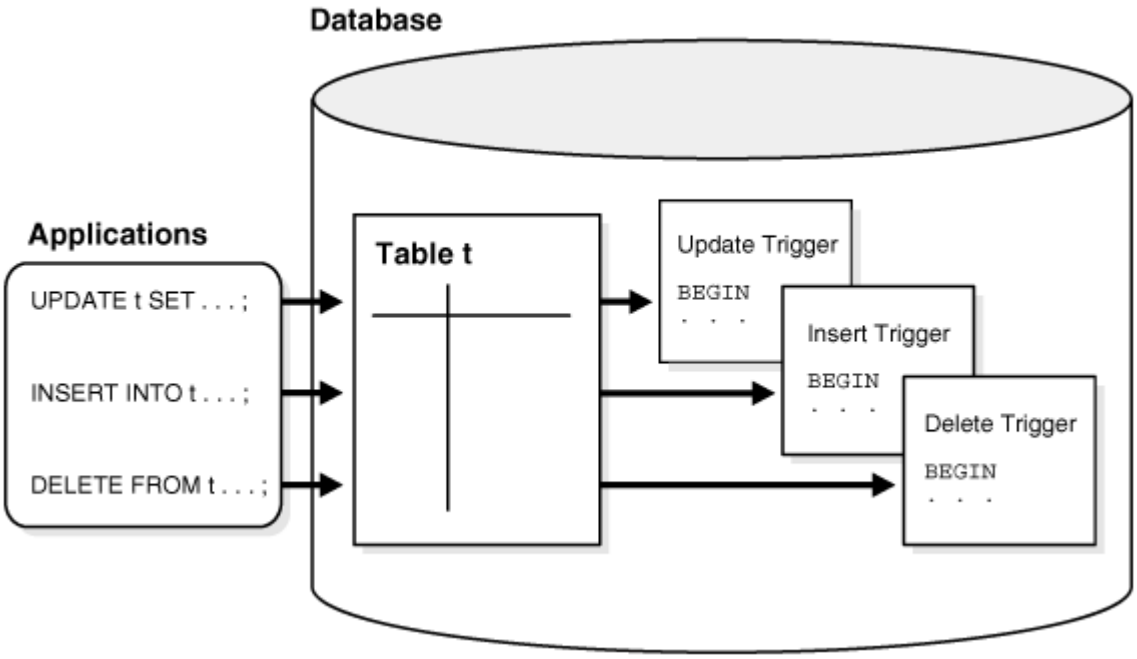
3.7 Databázové spouště

Spoušť neboli anglicky „Trigger“, je v pojetí relačních databází pojmem zaštiťujícím speciální procedury, jež jsou sepsány pomocí jazyka PL/SQL.

Nutno upozornit, že spoušť a uložená procedura není jedno a to samé, ačkoli jsou si velmi podobné. Hlavním rozdílem je způsob, kterým jsou vyvolány.

Spoušť je implicitně aktivována, pokud nastává uživatelem definovaná událost, a to bez jakéhokoliv ohledu na to, jaký uživatel je připojen nebo jaká aplikace je užívána. Na obrázku č. 2 vidíme znázorněné uložení tabulky a spouští, jež je odděleno od aplikace, která pokusem o manipulaci s daty uloženými v dané tabulce aktivuje tyto spouště. [4] [5]

Obrázek č. 2 – Implicitní aktivace spouště [13]



Zdroj: Database Concepts. ORACLE, 2019.

Procedura je na rozdíl od spouště explicitně spuštěna uživatelem, případně aplikací či právě spouští. Název „Spoušť“ se odvíjí od skutečnosti, že takto uložená procedura je provedena pouze tehdy, když nastane předem uživatelem specifikovaná spouštěcí událost. Případně spouště taktéž umožňují spuštění procedur napsaných v programovacím jazyce C.

Na základě typu těchto událostí jsou spouště dále členěny:

a) **DML spouště**
   Jsou aktivovány po, či před užitím příkazů „INSERT“/ „UPDATE“/ „DELETE“ jakýmkoliv uživatelem.

b) **DDL spouště**
   Aktivují se nejčastěji po, či před příkazy „CREATE“/ „ALTER“ užité buď specifikovaným uživatelem nebo jakýmkoli uživatelem.

c) **Spouště vázané na databázové události**
   Aktivace probíhá po událostech „logon“/ „logoff“, po výskytu chyb nebo před vypnutím db, případně po zapnutí db.

Jazyk PL/SQL umožňuje stanovit logiku, s kterou je blok příkazů v těle spouště proveden. K dispozici jsou 3 možné varianty– „BEFORE“; „AFTER“; „INSTEAD OF“. [14]

3.7.1 Oblasti užití DB spouští

Ačkoliv lze většinu z možných způsobů užití spouští nahradit i běžnějším řešením, většinou neposkytují ani zdaleka takové možnosti, jako právě řešení koncipovaná na bázi triggerů. Spouště doplňují standardní funkcionality systému řízení báze dat o uživatelem definovaná omezení, podmínky aj.

Spouště lze užít zejména k:

- Automatickému generování derivovaných sloupcových hodnot
- Zamezení provedení neplatných transakcí
- Vynucení komplexních bezpečnostních autorizací
- Zabezpečení referenční integrity napříč uzly v distribuované databázi
- Zajištění komplexních byznysových pravidel
- Transparentnímu logování událostí
- Poskytování auditů
- Udržování synchronních tabulkových kopií
- Sběru statistik o přístupu k databázi
- Úpravě tabulkových dat, když jsou aplikovány DML příkazy nad pohledy
- Publikování informací o databázových událostech, uživatelských událostech a SQL příkazech do konkrétních aplikací

[13] [14]

3.7.2 Rizika spjatá s užitím DB spouští

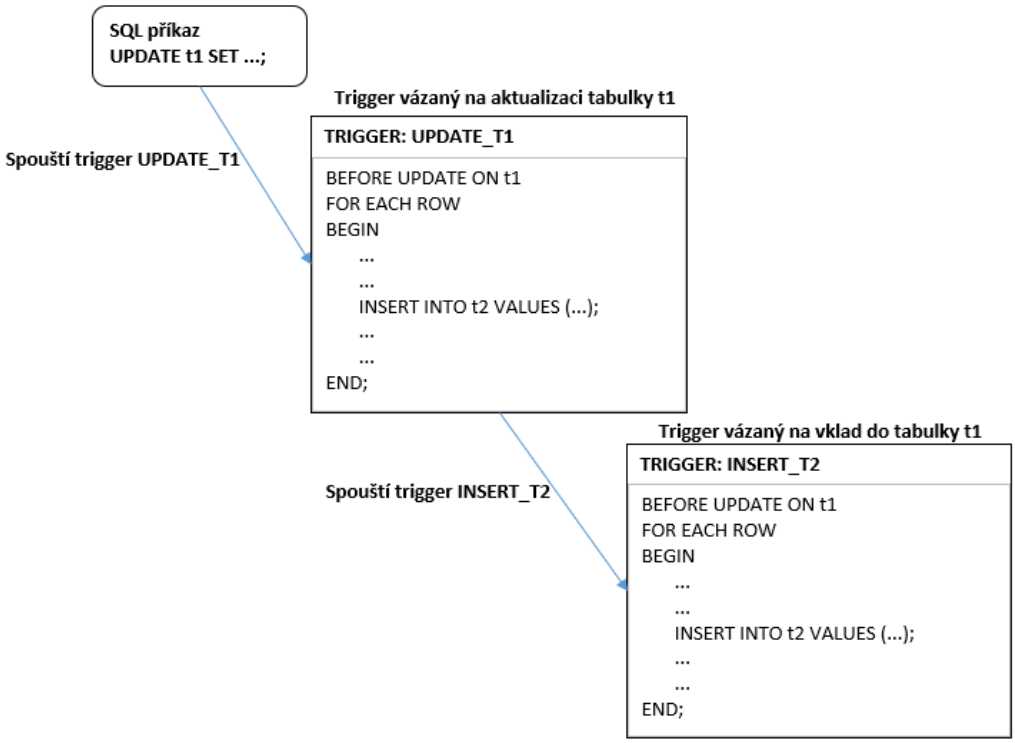
Stejně tak, jako mohou spouště automatizací procesů a přizpůsobením databáze ušetřit spoustu práce i času a tím i případných financí, tak mohou taktéž veliké množství práce přidělat. Obecně je silně doporučováno, aby byly spouště využívány pouze v určitých, úzce specifikovaných situacích, ve kterých je jejich užití téměř nezbytné.

Nadměrné užívání spouští může vyústit v komplikované vzájemné souvislosti mezi jednotlivými spouštěmi a způsobit tak nepřehledné a neočekávané chování databáze. Taková databáze je poté neefektivní a náročná na údržbu.

Situace, kdy SQL příkazy uvnitř vyvolané spouště způsobí aktivaci další spouště označujeme jako kaskádové spouště.

Na obrázku č. 3 je znázorněn případ takových spouští.

Obrázek č. 3 – Kaskádové spouště [13]



Zdroj: Database Concepts. ORACLE, 2019.

Jak je možno vidět na obrázku č. 3, uživatel zadává příkaz na aktualizaci dat v tabulce t1, což má za následek aktivaci první spouště „UPDATE_T1“, jež následně vkládá hodnoty do tabulky t2, což má však za následek aktivaci další spouště „INSERT_T2“, která provede další vložení dat. Takovéto řetězení spouští se může velmi rychle stát nepřehledným a zmatečným. [13]

3.7.3 Integritní omezení a spouště

Ačkoliv je možné zajistit integritní pravidla, jak pomocí spouští, tak i pomocí samotných integritních omezení, tak Oracle výrazně doporučuje užívání spouští k tomuto účelu pouze ve třech případech.

Prvním případem je situace, kdy se snažíme zajistit referenční integritu, ale tabulka typu potomek je na jiném uzlu distribuované databáze nežli tabulka předka.

Druhou možností je vynucení komplexních byznysových pravidel, která však nelze definovat a zajistit pomocí samotných integritních omezení.

Třetí případ nastává, pokud není možné zajistit referenční integritu s využitím integritních omezení: „NOT NULL“; „UNIQUE“ / „PRIMARY KEY“ / „FOREIGN KEY“/ „CHECK“ / „DELETE CASCADE“ / „DELETE SET NULL“. [11]

3.7.4 Konstrukce spouští

Každá spoušť se musí skládat minimálně ze dvou částí. V praxi se lze však setkat se spouštěmi, jež se skládají i ze tří částí, které na sebe logicky navazují.

1. Spouštěcí událost
   Základem je vždy deklarace spouštěcí události, jež určuje, kdy se spoušť aktivuje. Spouštěcí událostí jsou zpravidla DML příkazy.

2. Omezení spouště
   Následně může, ale nemusí být stanoveno omezení, které dále specifikuje podmínky pro aktivaci spouště.

3. Tělo spouště
   Nakonec je deklarována akce, jež se skládá z SQL příkazů, které budou po aktivaci spouště provedeny.

[4] [13]

Základní syntaxe spouště je poté následující:

```
CREATE [OR REPLACE] TRIGGER označení_spouště
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
ON {označení_tabulky | označení_pohledu}
FOR EACH {ROW | STATEMENT} [WHEN (podmínka)]
BEGIN
...
PL/SQL blok neboli tělo spouště
...
END;
```

Pro konkrétní znázornění konstrukce a jednotlivých částí spouště je dále uveden příklad spouště, která vytváří před vymazáním záznamu studenta z tabulky studentů zálohu do tabulky bývalých studentů. Vzhledem k demonstrativní povaze obou uvedených tabulek jsou značně zjednodušeny. Před sestavením samotné spouště, je nejdříve nutné vytvořit původní tabulku studentů, jež obsahuje aktuální studenty. Pro správnou funkcionalitu spouště je

naprosto nezbytné, aby následně vytvořená tabulka bývalých studentů, byla zcela identická s touto původní tabulkou. Tabulka bývalých studentů bude poté obsahovat studenty vymazané z první tabulky.
[4] [13]

Syntaxe tabulky studentů:

```
CREATE TABLE student
(
  id_stud    NUMBER(5) PRIMARY KEY,
  jmeno      VARCHAR2(25),
  rok_nar    DATE,
);
```

Syntaxe pro tabulku bývalých studentů:

```
CREATE TABLE old_student
(
  id_stud    NUMBER(5) PRIMARY KEY,
  jmeno      VARCHAR2(25),
  rok_nar    DATE,
);
```

Syntaxe pro spoušť „zaloha_studentu“:

```
CREATE OR REPLACE TRIGGER old_stud_trigger
  BEFORE DELETE ON student
  FOR EACH ROW
    BEGIN
    INSERT INTO old_student VALUES (:OLD.id_stud, :OLD.jmeno,
    :OLD.rok_nar);
END old_stud_trigger;
```

Jak je vidno u prvních tří řádků syntaxe uvedené výše, nejdříve byla stanovena spouštěcí podmínka, která určuje, že spoušť je aktivována před každým smazáním řádku z tabulky student.

Na čtvrtém až sedmém řádku je tělo spouště, které obsahuje příkaz INSERT, jenž slouží k vložení starých – odmazávaných hodnot do zálohovací tabulky. K tomu je využito prefixu :OLD, který lze u triggerů používat právě k dosažení původních hodnot ovlivňovaných řádků, nových hodnot by bylo možno dosáhnout prefixem :NEW.

Dále si můžeme povšimnout, že tato syntaxe neobsahuje žádné dodatečné omezení spouště pomocí klauzule „WHEN“. I přes tento fakt je však trigger validní, jelikož tato část není mandatorní. [4] [13]

3.7.5 Příkazy pro manipulaci se spouštěmi

Základním příkazem při práci se spouštěmi je samozřejmě klauzule „CREATE TRIGGER“, ta již byla uvedena a popsána v předcházejících kapitolách.

Pro manipulaci se spouštěmi lze však užít dalších dvou příkazů, které umožňují smazání spouští a jejich úpravu.

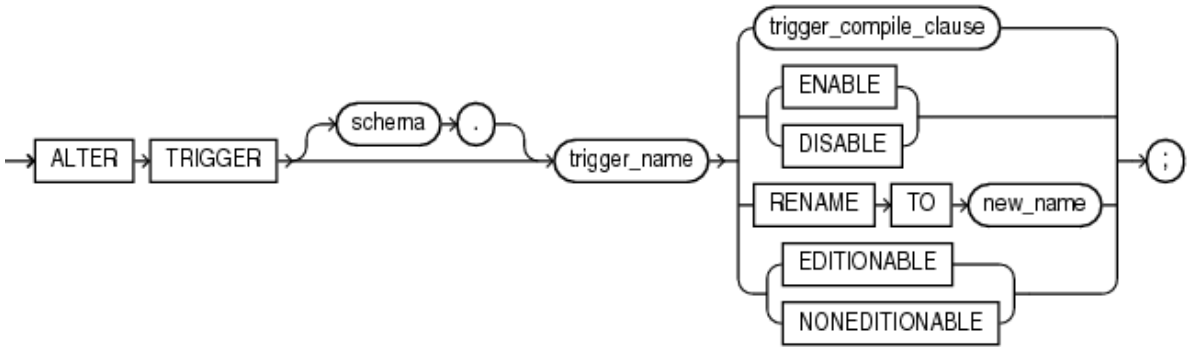
Pro úpravu spouští slouží příkaz „ALTER TRIGGER“, který zpřístupňuje následující operace:

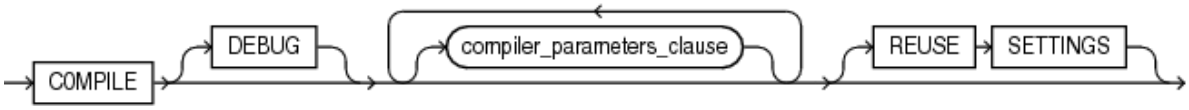
- Povolení spouště
- Zakázání spouště (Defaultně jsou spouště povoleny.)
- Kompilaci
- Přejmenování

Aby mohl uživatel úspěšně použít tento příkaz, musí být připojen jakožto SYSDBA, nebo pokud se daný trigger nachází mimo SYS schéma, poté musí uživatel být vlastníkem schéma, ve kterém se trigger nachází, případně musí mít „ALTER ANY TRIGGER“ systémové oprávnění. Pokud je upravován trigger přímo v databázi, poté musí mít uživatel oprávnění „ADMINISTER DATABASE TRIGGER“.

Syntaxi lze znázornit pomocí diagramu, jež je uveden na obrázku č.4 níže:

Obrázek č. 4 – ALTER TRIGGER & COMPILE [15]





Zdroj: Database SQL Reference. ORACLE, 2019.

Příklad modifikace:

- Příkaz na zakázání spouště „test_trigger“:
  `ALTER TRIGGER test_trigger DISABLE;`

Pro odstranění spouště z databáze slouží příkaz DROP TRIGGER, který opět vyžaduje určité prerekvizity. Aby uživatel mohl užít tento příkaz, musí se daný trigger nacházet v jeho vlastním schéma, nebo musí mít systémové oprávnění „DROP ANY TRIGGER“. Pokud by se daná spoušť nacházela v databázi, jež by byla v schématu jiného uživatele, poté je nutnou podmínkou systémové oprávnění „ADMINISTER DATABASE TRIGGER“.

Syntaxe je znázorněna pomocí diagramu uvedeného níže, na obrázku č.5:

Obrázek č. 5 – DROP TRIGGER [15]

Zdroj: Database SQL Reference. ORACLE, 2019.

Příklad odstranění:

- Příkaz na odstranění spouště „test_trigger“:
  `DROP TRIGGER test_trigger;`

[15]

3.8 XML

Z důvodu velmi časté rozdílnosti platforem a technologií, které různé podnikové aplikace a informační systémy používají, je vhodné využít možností, které nabízí jazyk XML. Tento jazyk totiž umožňuje komunikaci mezi těmito systémy a aplikacemi bez závislosti na platformě. Vždy obsahuje definici formátu údajů a operace s těmito údaji. O XML lze s nadsázkou stanovit, že je stejně dobře pochopitelný jak pro stroj, tak pro člověka.

Formát XML je naprostým standardem jak u ERP a CRM aplikací, tak i u webových služeb a aplikací. [4]

3.8.1 Struktura XML kódu

Kód tohoto jazyka je vždy sestaven v hierarchické stromové struktuře a je složen z elementů, kdy je vyžadováno, aby alespoň jeden element byl kořenový – tedy tzv. “ROOT“ element.
Vždy je taktéž silně doporučováno uvádět na začátku dokumentu verzi užitého XML, ačkoliv to není nezbytné. [3]

Příklad krátkého XML kódu:

```
<?xml version="1.0"?>
<zamestnanci>
      <zamestnanec>
            <jmeno>Petr</jmeno>
            <prijmeni>Koudela</prijmeni>
            <pozice>Analytik</pozice>
            <plat>35000</plat>
      </zamestnanec>
      <zamestnanec>
            <jmeno>Jan</jmeno>
            <prijmeni>Kysela</prijmeni>
            <pozice>CEO</pozice>
            <plat>55000</plat>
      </zamestnanec>
</zamestnanci>
```

3.8.2 Omezení kladená na prvky XML

Název každého jednoho prvku musí na začátku obsahovat písmeno, dále není povoleno užívat mezery a určité speciální znaky – z tohoto důvodu je doporučováno neužívat diakritiky. Taktéž je nutno stanovit, zda jsou rozlišována velká a malá písmena.

V rámci deklarace značky lze uvést atributy daného prvku s již přiřazenou hodnotou.

Obdobně, jak tomu je u prvků, se stahují stejná omezení i na tvorbu názvu atributu. Znak "=" slouží pro přiřazení hodnoty atributu, která se nachází buď v jednoduchých nebo dvojitých uvozovkách. [3] [4]

Názorný příklad:

```
<Mark atribute1="value1" atribute2="value2">obsah prvku</Mark>

<Zamestnanec cislo="1234" pozice="analytik">Petr Koudela</EMP>
```

3.8.3 Podpora XML v rámci ORACLE

Jazyk XML je platformou Oracle akceptován, pokud jsou dodrženy standardy stanovené normotvorným konsorciem W3C. Na úrovni db serveru jsou implementovány technologie sloužící pro podporu dokumentů formátu XML. Tyto technologie slouží k vzájemné dualitě, kdy jsou umožněny operace ukládání, vyhledávání atd. jak v rámci XML operací nad údaji uloženými v relační db, tak i v rámci SQL operací nad údaji uloženými v XML dokumentech.

Díky implementaci této technologie je tedy umožněno užívat jak výhod relačních databází, tak i XML. [16]

3.8.4 Vytvoření XML z údajů uložených v db tabulce

Pro vytvoření XML dokumentu je potřeba využít několika funkcí, jež Oracle k tomuto účelu poskytuje. Zejména se bude jednat o funkce:

- XMLElement()
  Jak lze již z názvu usoudit, jedná se o funkci, jež umožní vložit hodnotu z db tabulky do elementu XML.
- XMLAttributes()
  Tato funkce slouží k výpisu zvolených atributů, přičemž lze zvolit název atributu.
- XMLForest()
  Této funkce užijeme, pokud požadujeme výpis vícero atributů organizovaných do samostatných elementů.

[16]

Kombinací těchto funkcí lze vytvořit například následující XML dokument:

Příkaz:

```
SELECT XMLElement("Zamestnanec",
          XMLAttributes(czam as "cislo", poz as "pozice"),
          XMLElement("jmeno",jzam||' '||pzam), XMLForest(tel as
          "telefon", plat as "mzda", man as "manazer"))
FROM zam WHERE czam = 1234;
```

XML:

```
<Zamestnanec cislo="1234" pozice="analytik">
     <jmeno>Petr Koudela</jmeno>
     <telefon>721005582</telefon>
     <mzda>35000</mzda>
     <manazer>1230</manazer>
</Zamestnanec>
```

3.9 Package DBMS_XMLGEN

Pro vygenerování XML lze taktéž užít balíčku, jež je součástí Oracle XML DB a který se nachází ve všech nových verzích Oracle DB – verze 9i a vyšší. Konkrétně se jedná o balíček DBMS_XMLGEN.

Tento balíček vytváří automaticky XML dokument na základě SQL dotazu, kdy lze definovat počet řádků, které chceme do XML vložit, či počet řádků, které požadujeme přeskočit. Tato možnost je užitečná zejména u webových aplikací ve spojení s požadavky paginace. [3] [4]

3.9.1 Vytvoření a užití balíčku

Při vytváření a užití balíčku je nutné postupovat dle následujících kroků:

1. Získání kontextu z balíčku prostřednictvím SQL dotazu a voláním funkce "newContext".
2. Předání kontextu všem procedurám a funkcím uvnitř balíčku za účelem nastavení jeho možností.
3. Získání výsledného XML zavoláním funkce "getXML", či "getXMLType". Nastavením maximálního počtu řádků, které budou načteny pro každé zavolání užitím PL/SQL procedury "setMaxRows", lze zavolat každou z výše uvedených funkcí opakovaně, kdy je vždy vracen až maximální nastavený počet řádků. Chceme-li zjistit přesný počet řádků, které jsou načteny, lze použít funkce "getNumRowsProcessed".
4. Lze resetovat dotaz a opakovat postup kroku 3.
5. Pro uvolnění všech použitých prostředků je nutné zavolat PL/SQL proceduru closeContext. [16]

3.9.2 Příklad balíčku DBMS_XMLGEN

Syntaxe DBMS_XMLGEN:

```
/* Vytvoření dočasné tabulky pro vložení výsledku*/
CREATE TABLE tab_vys (result clob);
DECLARE
  qryCtx DBMS_XMLGEN.ctxHandle;
  result CLOB;
BEGIN
  /* Získání kontextu z dotazu */
  qryCtx := DBMS_XMLGEN.newContext('SELECT * FROM zam');
  /* Nastavíme maximální počet řádků na 3 */
  DBMS_XMLGEN.setMaxRows(qryCtx, 3);
  LOOP
    /* Získání výsledku */
    result := DBMS_XMLGEN.getXML(qryCtx);
    /* Pokud nenajdeme žádné řádky, poté dojde k ukončení */
    EXIT WHEN DBMS_XMLGEN.getNumRowsProcessed(qryCtx) = 0;

    /* Vložení výsledků do dočasné tabulky tab_vys*/
    INSERT INTO tab_vys VALUES(result);
  END LOOP;
  /* uzavření kontextu a uvolnění prostředků */
  DBMS_XMLGEN.closeContext(qryCtx);
END;
/
```

Výsledný výstup:

```
SELECT * FROM tab_vys WHERE rownum < 4;

RESULT
------------------------------------------------------------
<?xml version="1.0"?>
<ROWSET>
 <ROW>
  <CZAM>101</CZAM>
  <JZAM>Martin</JZAM>
  <PZAM>Koch</PZAM>
  <TEL>720314156</TEL>
  <POZ>CEO</POZ>
  <PLAT>65000</PLAT>
 </ROW>
 <ROW>
  <CZAM>102</CZAM>
  <JZAM>Jan</JZAM>
  <PZAM>Kysela</PZAM>
  <TEL>745654789</TEL>
  <POZ>manazer</POZ>
  <PLAT>55000</PLAT>
  <MAN>101</MAN>
 </ROW>
 <ROW>
  <CZAM>103</CZAM>
  <JZAM>Petr</JZAM>
  <PZAM>Koudela</PZAM>
  <TEL>721030245</TEL>
  <POZ>analytik</POZ>
  <PLAT>35000</PLAT>
  <MAN>102</MAN>
 </ROW>
</ROWSET>
3 rows selected. [10]
```

4 Praktická část

Tato část práce je zaměřena na praktickou ukázku konkrétního způsobu aplikace databázových spouští za účelem logování událostí, na zjednodušeném modelu autopůjčovny a konkrétního způsobu exportování zaznamenaných informací ve formátu XML. Tyto spouště slouží k záznamu neboli logování uživatelských aktivit, tedy operací a událostí, které se v uvedeném modelu uskuteční. Mezi tyto události patří – „LOGON“ a „LOGOFF“; „DELETE“ / „INSERT“ / „UPDATE“ a „CREATE“ / „ALTER“ / „DROP“. Pro záznam událostí LOGON a LOGOFF jsou sestaveny dvě spouště, jež plní tabulku ACCESS_LOG, pro záznam DML událostí jsou sestaveny 3 spouště pro každou výchozí tabulku a pro záznam DDL událostí je sestavena jedna spoušť. Spouště sestavované pro DML a DDL události vkládají patřičné informace do tabulky EVENT_LOG. Pro každou tabulku je uveden způsob realizace, včetně plnění a každá spoušť má uvedenou svou syntaxi. Následně je provedeno ověření požadované funkcionality.

4.1 Výchozí model

Uvedený model slouží k uchování informací souvisejících s činností autopůjčovny. Je zde tedy požadavek na evidenci údajů výpůjček a všech v tomto procesu zúčastněných osob, tedy zákazníků a personálu autopůjčovny. U těchto osob je kladen důraz zejména na kontaktní údaje.

Mezi výchozí tabulky patří tabulka osob, jež uchovává obecné údaje o osobách a která je rodičovskou tabulkou pro tabulky zákazníků a personálu. Vztah těchto tabulek je řešen skrze referenční integritu. Dále tabulka vozidel, která udržuje údaje o modelu, značce, datu servisu a datu výroby. Nakonec je obsažena tabulka výpůjček, která obsahuje informace o datu půjčení, plánovaném datu vrácení a potvrzení, zda již bylo vozidlo vráceno, dále jsou skrze referenční integritu s tabulkou výpůjček propojeny tabulky personálu, zákazníků a vozidel.

Také je vhodné uvést, že vzhledem k modelové povaze tohoto schématu jsou připuštěna určitá zjednodušení, která nejsou v souladu s určitými předpoklady, které jsou kladeny na návrh databázových tabulek. Jedná se však o zjednodušení, která nemají dopad na navrhovanou funkcionalitu. Konkrétně se jedná například o nedodržení atomičnosti atributu adresy, která je pro zjednodušení ukládána jakožto řetězec v celku a nikoli po jednotlivých částech.

4.1.1 Syntaxe tabulek výchozího modelu

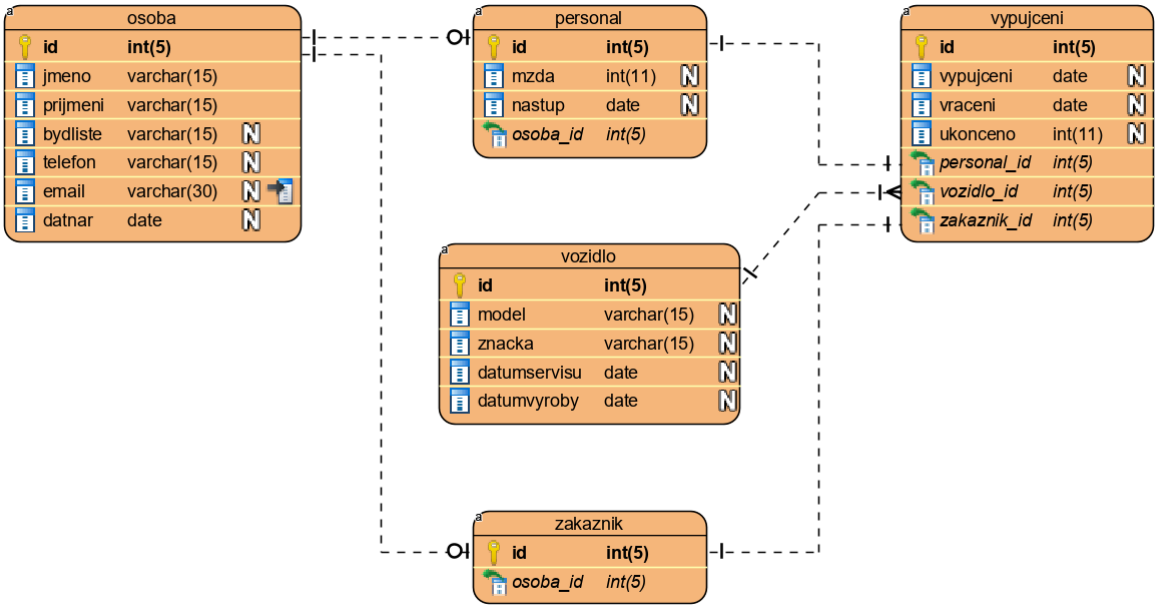
Syntaxe tabulek:

```
CREATE TABLE vozidlo (
    id number constraint vozidlo_pk primary key,
    model varchar2(15),
    znacka varchar2(15),
    datumservisu date,
    datumvyroby date
);


CREATE TABLE osoba (
    id number constraint osoba_pk primary key,
    jmeno varchar2(15) not null,
    prijmeni varchar2(15) not null,
    bydliste varchar2(15),
    telefon varchar2(15),
    email varchar2(30),
    datnar date
);
alter table osoba add constraint osoba_email_uq unique (email);

CREATE TABLE zakaznik (
    id number constraint zakaznik_pk primary key,
    osoba_id number,
    FOREIGN KEY (osoba_id) REFERENCES Osoba(id)
);

CREATE TABLE personal (
    id number constraint personal_pk primary key,
    mzda number,
    nastup date,
    osoba_id number not null,
    FOREIGN KEY (osoba_id) REFERENCES Osoba(id)
);

CREATE TABLE vypujceni (
    id number constraint vypujceni_pk primary key,
    vypujceni date,
    vraceni date,
    ukonceno number,
    personal_id number not null,
    vozidlo_id number not null,
    zakaznik_id number not null,
    FOREIGN KEY (personal_id) REFERENCES Personal(id),
    FOREIGN KEY (vozidlo_id) REFERENCES Vozidlo(id),
    FOREIGN KEY (zakaznik_id) REFERENCES Zakaznik(id)
);
```

4.1.2 ERD výchozího modelu

Na obrázku č. 6 níže vidíme zmiňovaný výchozí model a jeho příslušné vnitřní vazby, tedy vztahy, které vychází ze syntaxe tabulek, jež jsou výše uvedeny. Diagram dále taktéž zobrazuje kardinality vazeb, integritní omezení a datové typy.

Obrázek č. 6 – ERD autopůjčovna



Zdroj: Autor

4.2 Vložení hodnot

Pro názornost jsou uvedeny vždy pouze 2 vložení do každé tabulky, celková podoba všech tabulek je poté v kapitole 4.3, kde jsou zobrazeny všechny tabulky pomocí příkazu SELECT.

Tabulka OSOBA:

```
insert into Osoba values(1, 'Jakub', 'Novotny', 'Neplatna 15',
'701465798', 'jakub.novotny@eschranka.cz',DATE '1990-03-04');
1 row(s) inserted.

insert into Osoba values(2, 'Petr', 'Flinta', 'Mastna 20',
'703795712', 'petr.flinta@eschranka.cz',DATE '1992-05-05');
1 row(s) inserted.
```

Tabulka PERSONAL:

```
insert into Personal values(1, 25000, DATE '2013-01-03', 6);
1 row(s) inserted.

insert into Personal values(2, 27000, DATE '2010-05-07', 7);
1 row(s) inserted.
```

Tabulka VOZIDLO:

```
insert into Vozidlo values(1, 'Octavia', 'Skoda',DATE '2017-01-
05',DATE '2001-01-02');
1 row(s) inserted.

insert into Vozidlo values(2, 'Fabia', 'Skoda',DATE '2016-01-
02',DATE '2003-02-03');
1 row(s) inserted.
```

Tabulka ZAKAZNIK:

```
insert into Zakaznik values(1, 1);
1 row(s) inserted.

insert into Zakaznik values(2, 2);
1 row(s) inserted.
```

Tabulka VYPUJCENI:

```
insert into Vypujceni values(1,DATE '2018-01-02',DATE '2019-01-02',
1, 1, 1, 1);
1 row(s) inserted.

insert into Vypujceni values(2,DATE '2018-07-05',DATE '2019-07-05',
0, 1, 2, 2);
1 row(s) inserted.
```

4.3 Přehled tabulek

Za účelem přehlednosti a kontroly obsahu je zobrazen celkový obsah všech tabulek, jež jsou součástí výchozího modelu – viz. tabulky č.9 až č.13.

Tabulka č. 9 – Tabulka VOZIDLO

```
SELECT * FROM vozidlo
```

ID	MODEL	ZNACKA	DATUMSERVISU	DATUMVYROBY
1	Octavia	Skoda	05-JAN-17	02-JAN-01
2	Fabia	Skoda	02-JAN-16	03-FEB-03
3	Focus	Ford	03-JAN-18	05-APR-05
4	Fiesta	Ford	05-JAN-18	07-JUN-07
5	Astra	Opel	09-JAN-18	09-JAN-18

```
5 rows selected.
```

Zdroj: Autor

Tabulka č. 10 – Tabulka OSOBA

```
SELECT * FROM osoba
```

ID	JMENO	PRIJMENI	BYDLISTE	TELEFON	EMAIL	DATNAR
1	Jakub	Novotny	Neplatna 15	701465798	jakub.novotny@eschranka.cz	04-MAR-90
2	Petr	Flinta	Mastna 20	703795712	petr.flinta@eschranka.cz	05-MAY-92
3	Zaneta	Fialova	Potocna 30	705235168	zaneta.fialova@eschranka.cz	01-MAR-94
4	Jan	Zito	Konopna 7	708796458	jan.zito@eschranka.cz	03-FEB-80
5	Frantisek	Buben	Mlecna 5	721764751	frantisek.buben@echrankal.cz	03-APR-85
6	Jana	Novakova	Fojtikova 13	720753159	jana.novakova@eschranka.cz	05-JUL-89
7	Ludvik	Pokorny	Petrzelova 8	720951753	ludvik.pokorny@eschranka.cz	08-MAY-90

```
7 rows selected.
```

Zdroj: Autor

Tabulka č. 11 – Tabulka ZAKAZNIK

```
SELECT * FROM zakaznik
```

ID	OSOBA_ID
1	1
2	2
3	3
4	4
5	5

```
5 rows selected.
```

Zdroj: Autor

Tabulka č. 12 – Tabulka PERSONAL

```
SELECT * FROM personal
```

ID	MZDA	NASTUP	OSOBA_ID
1	25000	03-JAN-13	6
2	27000	07-MAY-10	7

```
2 rows selected.
```

Zdroj: Autor

Tabulka č. 13 – Tabulka VYPUJCENI

```
SELECT * FROM vypujceni
```

ID	VYPUJCENI	VRACENI	UKONCENO	PERSONAL_ID	VOZIDLO_ID	ZAKAZNIK_ID
1	02-JAN-20	31-JAN-20	1	1	1	1
2	05-JUL-18	05-JUL-19	0	1	2	2
3	08-OCT-18	08-OCT-19	0	1	3	3
4	12-DEC-18	12-DEC-19	0	2	4	4
5	26-JAN-19	29-JAN-20	1	2	5	5

```
5 rows selected.
```

Zdroj: Autor

4.4 Specifikace logování

K uvedeným výchozím tabulkám je nutné pro potřeby logování sestavit dvě dodatečné tabulky, ve kterých budou uchovávány údaje o výše zmíněných aktivitách – událostech. Pro uchování DML a DDL operací je navržena tabulka EVENT_LOG, která uchovává informace o uživateli, který operaci provedl, typu a pojmenování objektu, nad nímž byla provedena, operaci, jež byla provedena, datumu a času, kdy byla provedena. Pro uchování informací o přístupu do databáze je sestavena tabulka ACCESS_LOG, která uchovává informace o uživateli, který se připojil či odpojil, datu a času připojení a odpojení od DB.

Syntaxe tabulek EVENT_LOG & ACCESS_LOG:

```
CREATE TABLE access_log (
    uzivatel varchar2(20),
    udalost varchar2(30),
    logon_dat date,
    logon_cas varchar2(15),
    logoff_dat date,
    logoff_cas varchar2(15)
);


CREATE TABLE event_log (
    uzivatel varchar2(30),
    objekt varchar2(30),
    oznaceni varchar2(30),
    udalost varchar2(30),
    dat_udalosti date,
    cas_udalosti varchar2(15)
);
```

Jak z popisu tabulek vyplývá, EVENT_LOG bude naplňován spouštěmi, které se řadí mezi tzv. „Schema level“ spouště, kdežto ACCESS_LOG bude naplňován spouštěmi, jež se řadí mezi tzv. „Database level“ spouště. Tato informace je důležitá především z hlediska požadovaných oprávnění pro vytvoření daných spouští. Pro první zmiňovaný typ stačí být vlastníkem schématu, ale pro druhý je již nutné mít oprávnění umožňující vytvářet objekty v celé databázi – viz. teoretická východiska této práce.

4.4.1 Spouště plnící ACCESS_LOG

Syntaxe spouští ACCESS_LOGON_TRIG & ACCESS_LOGOFF_TRIG:

```
CREATE OR REPLACE TRIGGER access_logoff_trig
BEFORE LOGOFF ON DATABASE
BEGIN
    INSERT INTO access_log VALUES(
        user,
        ora_sysevent,
        NULL,
        NULL,
        SYSDATE,
        TO_CHAR(sysdate, 'hh24:mi:ss')
        );
COMMIT;
END;
```

```
CREATE OR REPLACE TRIGGER access_logon_trig
AFTER LOGON ON DATABASE
BEGIN
    INSERT INTO access_log VALUES(
        user,
        ora_sysevent,
        SYSDATE,
        TO_CHAR(sysdate, 'hh24:mi:ss'),
        NULL,
        NULL
        );
COMMIT;
END;
```

4.4.2 Spouště plnící EVENT_LOG

Syntaxe spouště DDL_EVENT_TRIG:

```
CREATE OR REPLACE TRIGGER ddl_event_trig
AFTER CREATE OR DROP OR ALTER ON SCHEMA
BEGIN
    INSERT INTO EVENT_LOG VALUES (
        user,
        ORA_DICT_OBJ_TYPE,
        ORA_DICT_OBJ_NAME,
        ORA_SYSEVENT,
        sysdate,
        TO_CHAR(sysdate, 'hh24:mi:ss')
        );
END;
```

Syntaxe DML spouští:

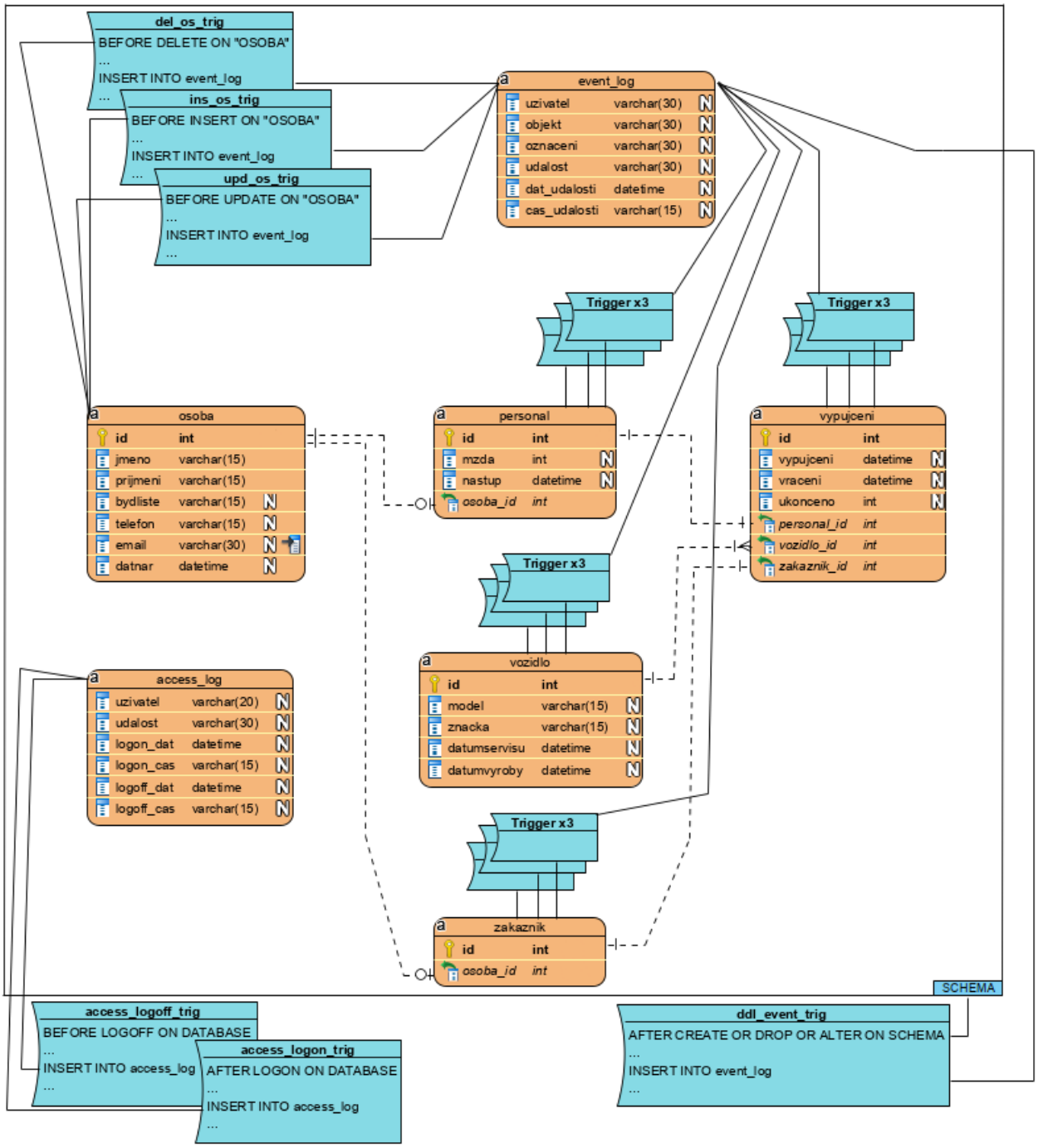
- Je uvedena typová syntaxe 3 spouští nad jednou tabulkou, pro dosažení plné funkcionality řešení logování událostí je nutné obdobné spouště sestavit i pro zbývající 4 tabulky. Pro tabulku OSOBA se tedy jedná o spouště: DEL_OS_TRIG; INS_OS_TRIG; UPD_OS_TRIG.

```
CREATE OR REPLACE TRIGGER DEL_OS_TRIG
BEFORE DELETE ON "OSOBA"
BEGIN
    INSERT INTO EVENT_LOG VALUES (
        user,
        'TABLE',
        'OSOBA',
        'DELETE',
        sysdate,
        TO_CHAR(sysdate, 'hh24:mi:ss')
        );
END;


CREATE OR REPLACE TRIGGER INS_OS_TRIG
BEFORE INSERT ON "OSOBA"
BEGIN
    INSERT INTO EVENT_LOG VALUES (
        user,
        'TABLE',
        'OSOBA',
        'INSERT',
        sysdate,
        TO_CHAR(sysdate, 'hh24:mi:ss')
        );
END;


CREATE OR REPLACE TRIGGER UPD_OS_TRIG
BEFORE UPDATE ON "OSOBA"
BEGIN
    INSERT INTO EVENT_LOG VALUES (
        user,
        'TABLE',
        'OSOBA',
        'UPDATE',
        sysdate,
        TO_CHAR(sysdate, 'hh24:mi:ss')
        );
END;
```

4.4.3 ERD modelu doplněného o navržené řešení záznamu

Na obrázku č. 7 je uveden výchozí model doplněný o tabulky pro logování událostí a přístupu, tedy o tabulky EVENT_LOG a ACCESS_LOG. Dále jsou zobrazeny v předchozích kapitolách popsané triggery, které jsou pro správnou funkcionalitu zamýšleného řešení nezbytné.

Obrázek č. 7 – ERD navrženého řešení



Zdroj: Autor

4.5 Navržení exportu logu do formátu XML

Export bude proveden pro dvě různé tabulky, proto je potřeba následující DBMS_XMLGEN navrhnout dvakrát. Tento návrh vychází z postupu uvedeného v teoretické části. Nejprve definujeme tabulku pro uchování výstupu ve formátu XML a následně je definováno samotné tělo balíčku:

Pro tabulku EVENT_LOG:

```
CREATE TABLE XML_EVENT_LOG (result clob)
Table created.
```

Následně sestavíme samotný package DBMS_XMLGEN:

```
DECLARE
  qryCtx DBMS_XMLGEN.ctxHandle;
  result CLOB;
BEGIN

  /* Získání kontextu z dotazu */

  qryCtx := DBMS_XMLGEN.newContext('SELECT * FROM event_log ORDER BY
CAS_UDALOSTI');

  /* Nastavíme maximální počet řádků na 5 – pro testovací účely */

  DBMS_XMLGEN.setMaxRows(qryCtx, 5);
  LOOP

    /* Získání výsledku */

    result := DBMS_XMLGEN.getXML(qryCtx);

    /* Pokud nenajdeme žádné řádky, poté dojde k ukončení */

    EXIT WHEN DBMS_XMLGEN.getNumRowsProcessed(qryCtx) = 0;

    /* Vložení výsledků do tabulky XML_EVENT_LOG*/

    INSERT INTO xml_event_log VALUES(result);
  END LOOP;

  /* uzavření kontextu a uvolnění prostředků */

  DBMS_XMLGEN.closeContext(qryCtx);
END;
```

Pro tabulku ACCESS_LOG:

```
CREATE TABLE XML_ACCESS_LOG (result clob)
Table created.
```

Následně sestavíme samotný package DBMS_XMLGEN:

```
DECLARE
  qryCtx DBMS_XMLGEN.ctxHandle;
  result CLOB;
BEGIN

  /* Získání kontextu z dotazu */

  qryCtx := DBMS_XMLGEN.newContext('SELECT * FROM access_log');

  /* Maximální počet řádků nastaven pro testovací účely na 2 */

  DBMS_XMLGEN.setMaxRows(qryCtx, 2);
  LOOP

    /* Zachycení výsledku */

    result := DBMS_XMLGEN.getXML(qryCtx);

    /* Pakliže DBMS_XMLGEN nenachází žádné řádky, dochází k ukončení
*/

    EXIT WHEN DBMS_XMLGEN.getNumRowsProcessed(qryCtx) = 0;

    /* Naplnění tabulky XML_ACCESS_LOG výsledkem */

    INSERT INTO xml_access_log VALUES(result);
  END LOOP;

  /* uzavření kontextu a uvolnění prostředků */

  DBMS_XMLGEN.closeContext(qryCtx);
END;
```

4.6 Otestování navrhovaných řešení

Tato část práce se dále zaměřuje na praktické ověření navržených řešení. Vždy je provedeno několik operací tak, aby byly otestovány všechny možné případy, které mohou při práci s výchozím modelem nastat. Jsou tedy aplikovány příkazy jazyka DML – INSERT; UPDATE; DELETE a příkazy jazyka DDL – CREATE; ALTER; DROP. Následně je ověřeno zaznamenávání přístupu neboli události LOGON a LOGOFF. Poslední testovanou částí je tvorba logu ve formátu XML na základě hodnot uložených prostřednictvím triggerů v příslušných tabulkách – EVENT_LOG a ACCESS_LOG.

4.6.1 Testy DML a DDL logování

Nyní ověříme funkcionalitu logování událostí do tabulky EVENT_LOG, tedy DML a DDL logování. Nejprve bude vložen nový záznam do tabulky osob, následně bude upraven, a nakonec opět z tabulky smazán. Po provedení zmíněných operací bude zobrazen obsah tabulky EVENT_LOG po provedení těchto DML operací – viz. tabulka č.14.

```
INSERT INTO osoba values(8, 'Petr', 'Mikes', 'Domazlicka 8',
'701649798', 'petr.mikes@eschranka.cz',DATE '1991-07-07')

1 row(s) inserted.


UPDATE Osoba
SET Jmeno = 'Ondrej', Prijmeni = 'Hasek', Email =
'ondrej.hasek@eschranka.cz'
WHERE Id = 8

1 row(s) updated.


DELETE FROM osoba WHERE Id = 8

1 row(s) deleted.
```

Tabulka č. 14 – Tabulka EVENT_LOG – DML test

```
SELECT * FROM event_log
```

UZIVATEL	OBJEKT	OZNACENI	UDALOST	DAT_UDALOSTI	CAS_UDALOSTI
APEX_PUBLIC_USER	TABLE	OSOBA	INSERT	26-JAN-20	12:56:51
APEX_PUBLIC_USER	TABLE	OSOBA	UPDATE	26-JAN-20	12:56:51
APEX_PUBLIC_USER	TABLE	OSOBA	DELETE	26-JAN-20	12:56:51

```
3 rows selected.
```

Zdroj: Autor

Jak vidíme v tabulce č. 14 výše, tabulka zaznamenává požadované informace o DML operacích, nyní otestujeme logování událostí CREATE, ALTER a DROP, pomocí vytvoření testovací tabulky SPONZOR. Následně je záznam ověřen opět zobrazením tabulky EVENT_LOG – tabulka č. 15.

Syntaxe tabulky sponzor:

```
CREATE TABLE sponzor (
    Id int,
    Jmeno varchar2(20),
    Prijmeni varchar2(20),
    Adresa varchar2(20),
    Mesto varchar2(20)
);
Table created.
```

Úprava tabulky a její následné smazání:

```
ALTER TABLE Sponzor
ADD Prispevky_celkem number
Table altered.

DROP TABLE Sponzor
Table dropped.
```

Zobrazení obsahu tabulky EVENT_LOG:

Tabulka č. 15 – Tabulka EVENT_LOG – DDL test

```
SELECT * FROM event_log
```

UZIVATEL	OBJEKT	OZNACENI	UDALOST	DAT_UDALOSTI	CAS_UDALOSTI
APEX_PUBLIC_USER	TABLE	OSOBA	INSERT	26-JAN-20	12:56:51
APEX_PUBLIC_USER	TABLE	OSOBA	UPDATE	26-JAN-20	12:56:51
APEX_PUBLIC_USER	TABLE	OSOBA	DELETE	26-JAN-20	12:56:51
APEX_PUBLIC_USER	TABLE	SPONZOR	CREATE	26-JAN-20	12:56:53
APEX_PUBLIC_USER	TABLE	SPONZOR	ALTER	26-JAN-20	12:56:55
APEX_PUBLIC_USER	TABLE	SPONZOR	DROP	26-JAN-20	12:56:56

```
6 rows selected.
```

Zdroj: Autor

Nyní ještě provedeme kontrolu, zda logování probíhá i pro jiné objekty nežli pouze tabulky. Pro názornost je uvedena sekvence a spoušť. Výsledný záznam je zobrazen v tabulce č. 16.

```
CREATE SEQUENCE vozidlo_seq
 START WITH     1
 INCREMENT BY   1
 NOCACHE
 NOCYCLE
;
Sequence created.
DROP SEQUENCE vozidlo_seq;
Sequence dropped.
```

```
CREATE OR REPLACE TRIGGER TEST
  BEFORE
    INSERT OR
    UPDATE OR
    DELETE
  ON OSOBA
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING THEN
      DBMS_OUTPUT.PUT_LINE('Updating');
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Deleting');
  END CASE;
    END;
```

```
Trigger created.
```

```
DROP TRIGGER TEST
```

```
Trigger dropped.
```

Zobrazení obsahu tabulky EVENT_LOG:

Tabulka č. 16 – Tabulka EVENT_LOG – objekt test

```
SELECT * FROM EVENT_LOG ORDER BY CAS_UDALOSTI
```

UZIVATEL	OBJEKT	OZNACENI	UDALOST	DAT_UDALOSTI	CAS_UDALOSTI
APEX_PUBLIC_USER	TABLE	OSOBA	INSERT	26-JAN-20	12:56:51
APEX_PUBLIC_USER	TABLE	OSOBA	UPDATE	26-JAN-20	12:56:51
APEX_PUBLIC_USER	TABLE	OSOBA	DELETE	26-JAN-20	12:56:51
APEX_PUBLIC_USER	TABLE	SPONZOR	CREATE	26-JAN-20	12:56:53
APEX_PUBLIC_USER	TABLE	SPONZOR	ALTER	26-JAN-20	12:56:55
APEX_PUBLIC_USER	TABLE	SPONZOR	DROP	26-JAN-20	12:56:56
APEX_PUBLIC_USER	SEQUENCE	VOZIDLO_SEQ	CREATE	26-JAN-20	12:56:57
APEX_PUBLIC_USER	SEQUENCE	VOZIDLO_SEQ	DROP	26-JAN-20	12:56:58
APEX_PUBLIC_USER	TRIGGER	TEST	CREATE	26-JAN-20	12:56:59
APEX_PUBLIC_USER	TRIGGER	TEST	DROP	26-JAN-20	12:57:00

```
10 rows selected.
```

Zdroj: Autor

4.6.2 Testy LOGON a LOGOFF logování

Nyní ověříme funkcionalitu logování přístupu, tedy záznam náležitých informací do tabulky ACCESS_LOG. Nejprve byl uživatel odhlášen prostřednictvím příkazu DISC; a následně se opět připojil k DB, ve které se nachází schema s naším modelem. Po provedení zmíněných operací bude zobrazen obsah tabulky ACCESS_LOG – viz. tabulka č. 17.

Tabulka č. 17 – Tabulka ACCESS_LOG – test záznamu přístupu

```
SELECT * FROM access_log
```

UZIVATEL	UDALOST	LOGON_DAT	LOGON_CAS	LOGOFF_DAT	LOGOFF_CAS
APEX_PUBLIC_USER	LOGOFF	NULL	NULL	26-JAN-20	13:00:51
APEX_PUBLIC_USER	LOGON	26-JAN-20	13:01:21	NULL	NULL

```
2 rows selected.
```

Zdroj: Autor

4.6.3 Ověření XML výstupů

Výstupem dvou navrhovaných package DBMS_XMLGEN je následující XML dokument, který získáme zobrazením obsahu tabulky XML_EVENT_LOG a tabulky XML_ACCESS_LOG:

Obsah tabulky XML_EVENT_LOG:

```
SELECT * FROM xml_event_log
```

(Pro názornost je uvedeno pouze 5 záznamů, skutečný výstup by byl znatelně rozsáhlejší.)

RESULT

```
<?xml version="1.0"?>
<ROWSET>
<ROW>
<UZIVATEL>APEX_PUBLIC_USER</UZIVATEL>
<OBJEKT>TABLE</OBJEKT>
<OZNACENI>OSOBA</OZNACENI>
<UDALOST>INSERT</UDALOST>
<DAT_UDALOSTI>26-JAN-20</DAT_UDALOSTI>
<CAS_UDALOSTI>11:29:23</CAS_UDALOSTI>
</ROW>
<ROW>
<UZIVATEL>APEX_PUBLIC_USER</UZIVATEL>
<OBJEKT>TABLE</OBJEKT>
<OZNACENI>OSOBA</OZNACENI>
<UDALOST>UPDATE</UDALOST>
<DAT_UDALOSTI>26-JAN-20</DAT_UDALOSTI>
<CAS_UDALOSTI>11:29:23</CAS_UDALOSTI>
</ROW>
<ROW>
<UZIVATEL>APEX_PUBLIC_USER</UZIVATEL>
<OBJEKT>TABLE</OBJEKT>
<OZNACENI>OSOBA</OZNACENI>
<UDALOST>DELETE</UDALOST>
<DAT_UDALOSTI>26-JAN-20</DAT_UDALOSTI>
<CAS_UDALOSTI>11:29:24</CAS_UDALOSTI>
</ROW>
<ROW> <UZIVATEL>APEX_PUBLIC_USER</UZIVATEL>
<OBJEKT>TABLE</OBJEKT>
<OZNACENI>SPONZOR</OZNACENI>
<UDALOST>CREATE</UDALOST>
<DAT_UDALOSTI>26-JAN-20</DAT_UDALOSTI>
<CAS_UDALOSTI>11:29:25</CAS_UDALOSTI>
</ROW>
<ROW>
<UZIVATEL>APEX_PUBLIC_USER</UZIVATEL>
<OBJEKT>TABLE</OBJEKT>
<OZNACENI>SPONZOR</OZNACENI>
<UDALOST>ALTER</UDALOST>
<DAT_UDALOSTI>26-JAN-20</DAT_UDALOSTI>
<CAS_UDALOSTI>11:29:28</CAS_UDALOSTI>
</ROW>
</ROWSET>
```

Obsah tabulky XML_ACCESS_LOG:

```
SELECT * FROM xml_access_log
```

(Pro názornost jsou uvedeny pouze 2 záznamy, skutečný výstup by byl opět znatelně rozsáhlejší.)

RESULT

```
<?xml version="1.0"?>
<ROWSET>
<ROW>
<UZIVATEL>HR</UZIVATEL>
<UDALOST>LOGOFF</UDALOST>
<LOGON_DAT>NULL</LOGON_DAT>
<LOGON_CAS>NULL</LOGON_CAS>
<LOGOFF_DAT>26-JAN-20</LOGOFF_DAT>
<LOGOFF_CAS>13:00:51</LOGOFF_CAS>
</ROW>
<ROW>
<UZIVATEL>HR</UZIVATEL>
<UDALOST>LOGON</UDALOST>
<LOGON_DAT>26-JAN-20</LOGON_DAT>
<LOGON_CAS>13:01:21</LOGON_CAS>
<LOGOFF_DAT>NULL</LOGOFF_DAT>
<LOGOFF_CAS>NULL</LOGOFF_CAS>
</ROW>
</ROWSET>
```

5 Diskuse

Záznam událostí se může na první pohled zdát jako nadbytečná až zbytečná záležitost, nicméně tak tomu není. Historii provedených operací a přístupu je vhodno vést vždy, kdy je požadována naprostá kontrola a úplný přehled nad stavem informací uložených v databázi, v různých časových momentech. Často hovoříme o složitých systémech, které jsou postaveny právě nad databází, jež se vyznačuje značnou mírou dynamiky z hlediska změn uložených dat a z hlediska přístupu k databázi. Proto je nutné udržovat přehled, kdo, kdy a jak data mění, a případně k nim přistupuje. Zcela zásadní se tudíž z tohoto pohledu jeví uchování alespoň základních informací – datum a čas dané operace; název uživatele, jenž danou operaci provedl; druh této operace a název objektu, nad kterým byla operace provedena.

Tak, jak je již u samotné automatizace procesů předmětem řešení zjednodušení správy a administrace, které jsou značně komplikovány exponenciálním růstem shromažďovaných informací, tak i u problematiky logování je nutné počítat s podstatným nárůstem shromažďovaných informací v databázi. V rámci automatizace procesů se však jedná o důvod, proč je vůbec nutné se k ní uchýlit, naopak u téma logování se jedná spíše o důsledek, který je nevyhnutelný a se kterým je nutné počítat. Proto je vhodné předem stanovit, že před zavedením automatizovaného záznamu událostí v db, je nutné se na tuto implementaci připravit a ověřit, zda stav a kapacita systému odpovídá nárokům, které s sebou zvolené řešení zcela jistě přináší.

I díky těmto faktům je nutno stanovit určité předpoklady a podmínky, aby dané řešení splňovalo stanovené požadavky. Zejména musí být určitým způsobem navrženo a v případě skutečného produkčního prostředí nejdříve před puštěním v ostrém provozu testováno a optimalizováno v testovacím prostředí, nad zkušebními daty, tak aby nebyl ohrožen ani chod databáze samotné, ani informace v ní uložené. Důraz je kladen zejména na snadnost syntaxe a co možná nejnižší výpočetní náročnost. Jelikož budou části logovacího řešení frekventovaně vyvolávány, je vhodné, aby se jejich aktivace projevila na výkonu nejmenší možnou mírou. V rámci požadavku výpočetní náročnosti je taktéž důležité se zaměřit na způsob realizace samotného logování. Úpravy a operace nad uloženými informacemi lze zjišťovat a následně ukládat dvěma různými způsoby. Prvním způsobem je záznam na úrovni aplikace, druhý způsob je poté záznam v rámci databáze.

Záznam na úrovni aplikace je složitější, pracnější a výpočetně náročnější, jelikož se aplikace musí upravit a doplnit o funkčnost, jež bude uchovávat původní stav záznamu před jeho následnou úpravou a musí být zajištěn záznam atributů, které se mění. Následně musí být patřičné informace aplikací propsány do záznamové tabulky v dané databázi. Tento přístup by byl složitý zejména u rozsáhlejších databází s větším počtem tabulek, což je právě často nastávající situace v konkrétním produkčním prostředí.

Naopak záznam na úrovni databáze je možné realizovat prostřednictvím databázových spouští neboli triggerů. Tyto konstrukty se nacházejí přímo uvnitř databáze, díky čemuž není tedy ani nutné zajistit propsání požadovaných informací přes aplikaci do databáze. Umožňují nadefinování automatické reakce na námi deklarovanou vyvolávající akci, přičemž definice a konstrukce probíhá pouze jednou, následně lze spoušť aktivovat a v případě potřeby opět deaktivovat, prostřednictvím klauzulí: “ENABLE“ a “DISABLE“. Jak je již z tohoto stručného popisu zřejmé, jedná se o řešení praktičtější, jednodušší a výkonnější. I z tohoto důvodu bylo zvoleno pro tuto práci.

Manuální vedení záznamu událostí je téměř nemožný úkol. Teoreticky by totiž bylo nutné každou operaci každého uživatele nad každým záznamem zaznamenávat postupným vkládáním detailních informací do příslušné tabulky příslušné databáze. Pokud by byl tedy záznam prováděn manuálně, bylo by nutné mu věnovat nepřeberné množství času, pozornosti a úsilí. Stejný úkol se však s využitím triggerů stává po jednorázové definici velmi snadným, jelikož díky celkové automatizaci nevyžaduje zcela žádnou lidskou aktivitu. Ačkoliv prvotní sestavení, nadefinování a otestování všech spouští a příslušných záznamových tabulek včetně exportování vyžaduje nezanedbatelné množství času, stále se jedná o nepopiratelně snazší a časově kratší úkon nežli zmiňovaný manuální záznam.

Automatizace logování uživatelských událostí a jejich přístupu, tak jak je představena v rámci praktické části této diplomové práce, představuje pouze jeden z možných způsobů, kterým lze požadované funkcionality dosáhnout a je v této formě vhodné zejména pro menší a středně velké podniky. Jsou aplikována pravidla, díky nimž je zaručena konzistence dat a nenarušení datové integrity. Veškeré spouště jsou prakticky aplikovány a následně vzorově otestovány. Nejedná se však o ideální, či nejlepší možné řešení.

Praktická část této práce obsahuje soubor spouští, tabulek a XML balíčků, které představují vlastní řešení zadaného problému logování událostí v databázových evidencích. Celkem je nadefinováno a představeno 18 spouští, 4 tabulky a 2 XML balíčky.

Pro každou tabulku výchozího modelu jsou konstruovány 3 spouště sloužící pro záznam DML operací. Veškeré DDL operace jsou zaznamenávány prostřednictvím jediné spouště aplikované na úrovni schéma. Přístup uživatelů je zaznamenán pomocí dvou spouští.

Záznam informací týkajících se DML a DDL operací realizovaných uživatelem je uchován v tabulce EVENT_LOG. Záznam informací souvisejících s přístupem k databázi je poté uchován v tabulce ACCESS_LOG. Zbývající dvě tabulky slouží pro export do formátu XML, tedy XML_EVENT_LOG a XML_ACCESS_LOG.

Na závěr, jak již název napovídá, jsou sestaveny 2 XML balíčky, sloužící exportu informací z tabulek EVENT_LOG a ACCESS_LOG do formátu XML.

Navržené řešení splňuje identifikované požadavky a předpoklady, které jsou na něj kladeny. Díky tomuto faktu jej lze aplikovat v rámci podnikatelských subjektů obdobným způsobem. Pro adekvátní a vhodné nasazení obdobného řešení by však muselo nejdříve být dále doplněno a ideálně agregováno do konkrétního, uceleného logovacího balíčku. Také by bylo vhodné věnovat další pozornost formátování v rámci exportu formátu XML. Zásadním přínosem logování je umožnění naprostého přehledu o aktivitě uživatelů a jejich manipulaci s daty uloženými v db. Díky záznamu událostí je možné realizovat rekonstrukci určitého stavu dat v databázi, a to k určitému datu a času. Není nutné hledat v zálohách, víme přesně, co i kdy se s daty stalo.

6 Závěr

Tato diplomová práce byla zaměřena na téma logování událostí v databázových evidencích v rámci informačního zabezpečení podniků.

Po zmapování aktuální situace a současného stavu v rámci odborné praxe je možno stanovit, že z hlediska správy a administrace zásadního objemu dat a informací, jsou relační databáze, a tedy i databáze společnosti Oracle, stále nejužívanějším řešením. Zároveň lze uvést, že objem ukládaných dat je na takové úrovni, kdy se jejich manuální kontrola a administrace stává značně komplikovanou, ne-li nereálnou záležitostí. Z tohoto důvodu je, stejně tak jako automatizace samotná, i automatizovaný záznam změn a aktivity za účelem kontroly a dohledávání, téměř nepostradatelnou součástí informačního zabezpečení podniků v kontextu datové základny. Těmto tvrzením odpovídá i stanovisko společnosti Oracle, které doporučuje spouště k uvedeným účelům užít.

Hlavním cílem této práce bylo navržení účelného, přínosného a zejména funkčního řešení záznamu událostí za užití databázových triggerů. Navržené řešení bylo nutné konstruovat s přihlédnutím k požadavku minimalizace náročnosti, a to jak z hlediska výpočetní náročnosti, tak i z hlediska náročnosti syntaxe. Požadavek na minimalizaci výpočetní náročnosti je uveden z důvodu předpokládaného vysokého počtu vyvolání těchto spouští.

Pro tyto potřeby bylo v rámci praktické části práce navrženo vlastní řešení, na kterém je představen způsob, kterým lze záznam událostí automatizovat, při splnění identifikovaných požadavků. Tato skutečnost byla dále otestována a okomentována.

Je nutné stanovit, že celkový přínos a potřebnost tohoto řešení roste s počtem záznamů, které se v dané databázi nachází. Přímo úměrně s počtem záznamů, a tedy i s počtem operací, které uživatelé provádějí, roste i množství času, a tedy i financí, které tato automatizace ušetří. Záznam je prováděn na úrovni databáze, díky čemuž není nutné zajistit propsání požadovaných informací přes aplikaci do databáze. Jedná se zde o řešení relativně jednoduché a díky tomu výpočetně nenáročné, zároveň však dostatečně robustní, a proto i praktické. Toto řešení má rovněž preventivní účely. Pokud by došlo v důsledku aktivity uživatele k narušení konzistence dat, či případně jiným chybám, lze tuto skutečnost díky logování odhalit podstatně rychleji a předejít tak dalšímu zdlouhavému šetření, které by v konečném důsledku taktéž mohlo přinést finanční ztráty. V práci uvedená realizace logování zcela dokazuje, že poskytuje velmi dobrou kontrolu a přehled nad stavem informací uložených

v databázi, v různých časových momentech. Díky tomu jsou data pod kontrolou, je umožněn náhled na proběhlou aktivitu a lze tak reagovat vhodně a včas na vyvstalou situaci. Automatizovaný záznam v této podobě ulehčuje práci správci databáze a poskytuje mu určitý komfort při provádění rutinních činností, jako například při vytváření, respektive generování určitých exportů, pro následné sestavení reportů.

Užitečnost samotných databázových triggerů je však mnohem širší. Spočívá totiž v jejich širokém užití, kdy lze s jejich pomocí například automaticky kontrolovat zadávaná data a zamezit jejich vložení do databáze, automaticky generovat odvozené hodnoty hodnot sloupců, zamezit invalidním transakcím nebo implementovat business pravidla. Navíc lze také zajistit jinak neaplikovatelné, konkrétně například lze aplikovat referenční integritu přes různé uzly databáze, nebo kupříkladu uložit skrze pohled data přímo do příslušných tabulek, což by jinak bylo nemožné, díky faktu, že pohledy nejsou dědičně aktualizovatelné. Proto jsou v tomto ohledu nenahraditelné.

Zároveň je důležité zdůraznit, že spouště nejsou tzv. všelékem. Zásadním předpokladem správné, vhodné a účelné implementace triggerů je důsledná analýza a kvalitní návrh konkrétního řešení. Vzhledem k již zmiňovanému množství uložených dat nelze, aby navržené a následně užité spouště narušili integritu či konzistenci databáze. Vždy musí být věnován dostatečný čas a prostor pro vytvoření dostatečně přehledného, robustního a udržitelného řešení. Ty nejjednodušší spouště bývají v mnohých případech ty nejefektivnější.

Dalšími dílčími cíli bylo nalezení možných způsobů realizace logování, zmapování současného stavu a vymezení relevantních požadavků a úvod do řešené problematiky. Pro tyto dílčí cíle byla sepsána literární rešerše. Pro splnění uvedených cílů bylo užito technik a postupů, jež vychází ze znalostí, které byly získány v rámci studia a analýzy odborné literatury, vlastní odborné praxe a dalších informačních zdrojů.

Byly užity především ty metody a techniky relačně databázového řešení, jež se týkají zajištění automatizace procesů v databázových systémech firmy Oracle, zejména bylo užito dotazovacího jazyka PL/SQL, datové integrity, DDL, DML a DCL příkazů a spouští. Rešerše je zaměřena zejména na problematiku integritních omezení, triggerů, formátu XML a způsobů, kterými lze informace v tomto formátu exportovat.

V rámci podniků je současným trendem a již i běžnou praxí nejprve implementovat a testovat nová řešení na testovacím prostředí daného systému, aplikace či databáze. Tyto testovací verze jsou v naprosté většině případů téměř totožnou kopií prostředí produkčního, jsou však skryté identifikační údaje daného subjektu. Následně jsou poté nové funkcionality důsledně testovány na relevantních datech a v případě nevyhovujících výsledků testování, probíhá optimalizace a úpravy. Do produkčního prostředí se tak dostávají již ověřené a spolehlivé funkcionality.

I přes uvedené důvody a výhody použití spouští, automatizace a logování v databázových evidencích, stále existují subjekty, zejména tedy ty střední a menší, které se užití těchto metod, technik a postupů vyhýbají. Především jsou to ty podniky, jež primárně svým polem působnosti nespadají do IT sféry. U takových subjektů hrozí zvýšené riziko v případě, kdy nastává chyba během manipulace s daty, či v momentě, kdy je potřeba realizovat audit změn dat. Takové a jim podobné operace jsou poté velmi náročné na provedení a zároveň jak časově, tak i finančně nákladnější. Pasivní přistupování k automatizaci má za následek neefektivní využívání lidských zdrojů a postupné zvyšování nákladů na zpracování dat.

S přihlédnutím k uvedeným skutečnostem bych označil záznam událostí, stejně tak jako použití spouští samotných a aplikace automatizace u vybraných procesů, za velmi vhodné, ne-li nezbytné řešení efektivního řízení operačního rizika, nákladů na správu a ideálně i bezchybný chod databáze. Dle mého názoru bude s postupným růstem shromažďovaných dat a postupným zdokonalováním technologií relačních databází naprosto nevyhnutelné, aby naprostá většina subjektů, bez ohledu na jejich velikost a zaměření, v případě že, využívají takové technologie, přistoupila k implementaci těchto a jim podobným řešením.

7 Seznam použitých zdrojů

[1] AUER, D J. -- KROENKE, D. Databáze. Brno: Computer Press, 2015. ISBN 978-80-251-4352-0.

[2] CODD, E. F.: A Relational Model of Data for Large Shared Databanks. Communication of the ACM (červen 1970): 377 – 387

[3] LACKO, L.: 1001 tipů a triků pro SQL. 1.vydání. Computer press, 2011. 416 stran.

[4] LACKO, Ľ.: Oracle : Správa, programování a použití databázového systému. Brno: Computer Press, 2002. ISBN 80-7226-699-3.

[5] LACKO, Ľ.: SQL : Hotová řešení : pro SQL Server, Oracle a MySQL. Brno: Computer Press, 2003. ISBN 80-7226-975-5.

[6] PROCHÁZKA, D.: Oracle. 1 vydání. Grada, 2009. 168 stran. ISBN: 978-80-247-2762-2.

[7] STEPHENS, R; PLEW, R; D.JONES, A.: Naučte se SQL za 28 dní. Computer Press, 2012. 728 stran. EAN: 9788025127001.

[8] VALENTA, M. -- POKORNÝ, J.: Databázové systémy. Praha: České vysoké učení technické v Praze, 2013. ISBN 978-80-01-05212-9.

[9] BROOK, C.: Data protection 101, 2019 [online]. [citováno 2019-24-11]. Dostupné na: https://digitalguardian.com/blog/what-data-integrity-data-protection-101

[10] KARAM, S.: Oracle dbms_xmlgen tips. 2014. [online] [citováno 2019-30-11]. Dostupné na: http://www.dba-oracle.com/t_dbms_xmlgen.htm

[11] ORACLE: Oracle Database Release 12.2. ORACLE, 2019. [online] [citováno 2019-20-08]. Dostupné na: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/.

[12] ORACLE: Data Integrity. ORACLE, 2019. [online] [citováno 2019-20-07]. Dostupné na: https://docs.oracle.com/cd/B19306_01/server.102/b14220/data_int.htm.

[13] ORACLE: Database Concepts. ORACLE, 2015. [online] [citováno 2019-20-11]. Dostupné na: https://docs.oracle.com/cd/E11882_01/server.112/e40540/toc.htm

[14] ORACLE: Database PL/SQL Language Reference. ORACLE, 2019. [online] [citováno 2019-20-11].
Dostupné na: https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/toc.htm

[15] ORACLE: Database SQL Reference. ORACLE, 2019. [online] [citováno 2019-20-11]. Dostupné na: https://docs.oracle.com/cd/B19306_01/server.102/b14200/

[16] ORACLE: XML DB Developer's Guide. ORACLE, 2014. [online] [citováno 2019-29-11]. Dostupné na:

[17] PEARLMAN, S.: What is data integrity, 2019 [online]. [citováno 2019-24-11]. Dostupné na: https://www.talend.com/resources/what-is-data-integrity/
[18] SKŘIVAN, J.: Databáze a jazyk SQL, 2000 [online]. [citováno 2019-15-07]. Dostupné na: https://www.interval.cz/clanky/databaze-a-jazyk-sql/
[19] ŠELKO, M.: 1. Normální forma, 2019 [online]. [citováno 2019-15-11]. Dostupné na: https://blog.root.cz/databaze/1-normalni-forma/
[20] VALENTA, M.: Integritní omezení (IO). Praha: FIT ČVUT, 2011. [online] [citováno 2019-20-11].
Dostupné na: https://users.fit.cvut.cz/valenta/doku/lib/exe/fetch.php/bivs/dbs2_02_io_ddl.pdf
https://docs.oracle.com/cd/E11882_01/appdev.112/e23094/xdb13gen.htm#ADXDB1600.