Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Automatizace podnikových procesů prostřednictvím databázových triggerů

Tomáš Jukl

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Tomáš Jukl

Informatika

Název práce

Automatizace podnikových procesů prostřednictvím databázových triggerů

Název anglicky

Automation of business processes via database triggers

Cíle práce

Bakalářská práce je zaměřena na problematiku využití databázových triggerů v informačním zabezpečení podnikatelských subjektů. Náplní této práce bude:
a) objasnit teoretické principy relačně databázové technologie, konkrétně datové integrity v kontextu s problematikou databázových triggerů,
b) zmapovat momentální stav této problematiky a vymezit její relevantnost včetně požadavků na ni kladených,
c) navrhnout možnosti přijatelných uplatnění této problematiky v souladu s identifikovanými požadavky,
d) ověřit funkčnost navržených záležitostí,
e) ověřené záležitosti zobecnit pro další možná uplatnění.

Metodika

Použitá metodika zadané bakalářské práce bude založena na studiu a analýze dostupných informačních zdrojů a 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 bakalářské práce a následně zobecněny pro další možná použití

Závazný harmonogram:

Vymezení teoretických principů řešené problematiky, literární rešerše – do 4.9.2017: předmět 1. zápočtu z BP,

Zmapování současné situace řešené problematiky a navržení odpovídajícího řešení – do 20. 12. 2017,

Ověření navrženého řešení – do 20.1.2018: předmět 2. zápočtu z BP

Zobecnění navrhovaných záležitostí – do 14.3.2018: předmět 3. zápočtu z BP.

Doporučený rozsah práce

45-55 stran

Klíčová slova

Relačně db technologie, datová integrita,databázový trigger, SQL, zpracování hromadných dat

---

Doporučené zdroje informací

BORONCZYK, T.: MySQL okamžitě. 1.vydání. Computer press, 2016. 144 stran. ISBN: 9788025147375.
KROENKE, D.: Databáze. 1.vydání. Computer press, 2015. 496 stran. ISBN: 978-80-251-4352-0.
LACKO, L.: 1001 tipů a triků pro SQL. 1.vydání. Computer press, 2011. 416 stran.
PROCHÁZKA, D.: Oracle. 1 vydání. Grada, 2009. 168 stran. ISBN: 978-80-247-2762-2.

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

doc. Dr. Ing. Václav Vostrovský

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 1. 2018

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 1. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 12. 03. 2018

Čestné prohlášení

Prohlašuji, že jsem svou bakalářskou práci "Automatizace podnikových procesů prostřednictvím databázových triggerů" vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14.03.2018 _____

Poděkování

Rád bych tímto poděkoval doc. Ing. Václavu Vostrovskému, Ph.D. za vedení při vypracování práce, za poskytnutí rad a připomínek, trpělivost a čas věnovaný konzultacím. Dále bych rád poděkoval mým nejbližším, kteří mi při vypracování této práce poskytli oporu.

Automatizace podnikových procesů prostřednictvím databázových triggerů

Abstrakt

Tato bakalářská práce (“Automatizace podnikových procesů prostřednictvím databázových triggerů“) je zaměřena zejména na automatizování procesů, které probíhají při manipulaci s relačně databázovými technologiemi, konkrétně firmy Oracle. V části zaměřené na teoretická východiska jsou probírány postupně základní informace o databázích, stručně se zde pojednává o historii a předchůdcích. Následně je probrán databázový model a relačně databázová řešení, včetně jazyka PL/SQL. Navazuje vymezení teoretických principů relací a způsobu ukládání dat této technologie. Zejména se zde hovoří o problematice datové integrity, včetně pojmů doménová, entitní a referenční integrita. Práce pokračuje teoretickými principy databázových triggerů v PL/SQL. Jsou stanoveny oblasti, ve kterých je vhodné triggery použít. Je uveden příklad, na kterém jsou zobrazeny hlavní části triggerů a následně základní syntaxe, z níž se při konstrukci vychází. Tato část práce obsahuje typy triggerů a příkazy sloužící pro jejich administraci.

V praktické části je již konkrétně uvedeno řešení vybraných triggerů, které lze využít při automatizaci podnikových procesů, se zaměřením na DML operace. Na závěr je provedeno ověření funkcionalit navrhovaného řešení, včetně otestování integritních omezení. Na základě tohoto ověření a otestování jsou formulovány výsledky práce a její závěr.

Klíčová slova: Relační db, datová integrita, databázový trigger, databázová spoušť, SQL, zpracování hromadných dat

Automation of business processes via database triggers

Abstract

This bachelor thesis ("Automation of business processes via database triggers") is focused mainly on the automation of processes that take place when dealing with relational database technologies, namely Oracle. In the part about the theoretical basis, basic information about the databases is briefly discussed, and it briefly goes through the history and predecessors. The database model and relational database technology, including PL / SQL, are described in this part. Next follows the definition of the theoretical principles of relations and the way of how the data is stored within this technology. In particular we talk about the issues of data integrity, including the concepts of domain, entity and reference integrity. The work continues with the theoretical principles of database triggers in PL / SQL. There are defined areas where triggers are appropriate. There is an example showing the main parts of the triggers, and then the basic syntax which they are commonly based on. This section also describes the types of triggers and commands used for their administration.

In the part of personal solutions there is an overview of specific selected triggers, which can be used in the automation of business processes with an emphasis on DML operations. Finally, the functionality of the proposed solutions is verified, including the testing of integrity constraints. Based on this validation and testing, the results of this work and its conclusion are formulated.

Keywords: Relational db, data integrity, database trigger, SQL, bulk data processing

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Tato práce je zaměřena na automatizaci podnikových procesů pomocí databázových triggerů prostřednictvím DBMS Oracle. Důraz je kladen zejména na automatizaci těch procesů, které by jinak časově vytěžovaly lidské zdroje, což by v důsledku mohlo znamenat zbytečné plýtvání finančními zdroji. Zmiňovaná problematika je velice důležitá také kvůli propojení velkého množství podnikových systémů, které jsou založeny na centrálním databázovém řešení a veškeré další aplikace jsou na něj napojeny. Místo rutinního obstarávání běžných činností, které mohou být automatizovány, lze následně využít pracovní sílu podniku efektivnějším způsobem.

K automatizaci procesů uvnitř databází postupně přechází stále větší počet podniků a díky tomu se tak stává běžnou praxí, že nejen velké podniky, ale i ty malé nacházejí prostor k použití databázových spouští. Tomu odpovídá i fakt, že žádné jiné řešení není ve světě pro správu a ukládání dat tak využíváno, jako relační databázové systémy. V některých případech se bohužel stává, že je potenciál pro rozvoj podniku a přínos automatizace zcela opomíjen a přehlížen, proto se tato práce snaží problematiku automatizace procesů prostřednictvím spouští přiblížit a alespoň z určité části objasnit.

Přímo úměrně s přibývajícím množstvím dat rostou zároveň i náklady na jejich správu a násobí se složitost jejich zpracování. Bez automatizace se musí data zpracovávat manuálně, což je při velkém objemu velice složité a neefektivní, u větších subjektů téměř nemožné. Při manuálním zpracování se s větším objemem dat výrazně projevuje riziko narušení integrity a konzistence dat.

Automatizace přináší výhody plynoucí z odstranění problémů, které by souvisely s manuálním zpracováním. Především lze zmínit značné urychlení podnikových procesů, snadnou reprodukci a replikaci automatizovaných úkonů. Zvýšení kontroly nad probíhajícími procesy, především z hlediska odstranění lidského faktoru, a tedy i odstranění rizika náhodné chyby. Ačkoliv jsou tyto důvody nezanedbatelné, bývá vývoj v této oblasti stále podniky přehlížen.

Proto je zaměření se na správné použití integritních omezení a databázových spouští naprosto nezbytné. Integritní omezení zajišťují vložení pouze určitého typu dat, pomocí podmínek kladených na vkládané hodnoty. Automatizace procesů je řízena pomocí databázových spouští, které v předem definovaném momentu spouští nadefinovanou sadu příkazů. Pomocí těchto spouští lze provádět velice širokou škálu operací, například lze kontrolovat vkládaná data, zamezit jejich vložení, případně je upravit. Případně lze vytvářet různé objekty, monitorovat a logovat činnosti prováděné uživatelem. Díky jednotnému a přesně definovanému přístupu k datům a jejich manipulaci je vyloučen lidský faktor, je tedy do jisté míry omezeno operační riziko. Databázové spouště jsou v mnohých případech nenahraditelné a nezastupitelné. Díky svému širokému spektru využití a zmiňované nezastupitelnosti budou dříve či později implementovány každým subjektem, spravujícím větší objem dat. Širokého využití se tomuto řešení dostává především v bankovním sektoru, kde je objem uchovávaných dat tak velký a operační a reputační riziko tak vysoké, že je využití automatizace zcela nezbytné. Důsledkem tohoto faktu je existence interních podpůrných skupin, které se zaměřují na údržbu aktuálně běžícího systému a na vývoj nových funkcionalit, jako například aplikaci nových spouští.

Téma této bakalářské práce bylo zvoleno s ohledem na mojí aktuální pracovní zkušenost, s databázovými technologiemi se setkávám a v sofistikované automatizaci jejich správy, zpracování a kontroly vidím velký potenciál do budoucnosti. Všechna uvedená řešení, která práce v praktické části obsahuje, jsou napsána a otestována v jazyce PL/SQL pro databázové technologie firmy Oracle. Tato řešení byla vypracována na základě znalostí získaných četbou odborných zdrojů.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této závěrečné práce je navrhnout funkční a přínosnou realizaci zabezpečení informačních systémů podniků prostřednictvím databázových triggerů. V tomto duchu následně navrhnout a finálně realizovat řešení poskytující zdokonalení především z hlediska časové náročnosti a náročnosti provádění rutinních procesů.

Dílčími cíli v rámci takto nedefinovaného hlavního cíle pak budou následující záležitosti:

- nalezení možných oblastí využití databázových triggerů
- identifikování konkrétních překážek a limitů v rámci zabezpečení podniků pomocí automatizace procesů
- zmapovat aktuální situaci využití databázových triggerů u podnikatelských subjektů
- vymezit současnou relevantnost této problematiky a stanovit požadavky kladené na zlepšení
- Na základě nalezených oblastí vhodných k aplikaci databázových triggerů navrhnout odpovídající řešení nabízející určitá zlepšení daného sektoru, respektující limity a bariéry použití

Problematika, ze které se u navržení vlastního řešení vychází, bude představena v teoretické části práce. Stručně bude nastíněna historie a trend vývoje databázové technologie, současný stav této technologie a oblasti možného uplatnění databázových triggerů. Cílem je taktéž navrhovaná řešení prakticky sestavit, otestovat a demonstrovat jejich zamýšlenou funkcionalitu a přínos. Tato ověřená řešení posléze zobecnit pro další vhodná uplatnění v podnikatelských subjektech. Část práce obsahující vlastní řešení poskytne konkrétní vypracované praktické příklady v jazyce PL/SQL, zajišťující automatizaci stanovených procesů, kdy bude pokaždé zobrazena jak konstrukce, tak i samotná funkce a vysvětlení, jakým způsobem daná spoušť ušetří čas.

2.2 Metodika

Pro zpracování této práce je použita metodika vycházející ze studia a analýzy dostupné odborné literatury, oficiální dokumentace firmy Oracle a dalších relevantních informačních zdrojů, včetně existujících řešení zabývajících se problematikou relačně databázových systémů a zejména databázových spouští realizovaných pomocí programovacího jazyka PL/SQL v kontextu automatizace podnikových procesů. Pro tuto práci budou významné především metody a techniky relačně databázového řešení, které jsou svým využitím soustředěny na aplikaci automatizace procesů v databázových systémech firmy Oracle, především se tedy bude jednat o programovací jazyk PL/SQL, triggery, integritní omezení, DCL a DML příkazy.

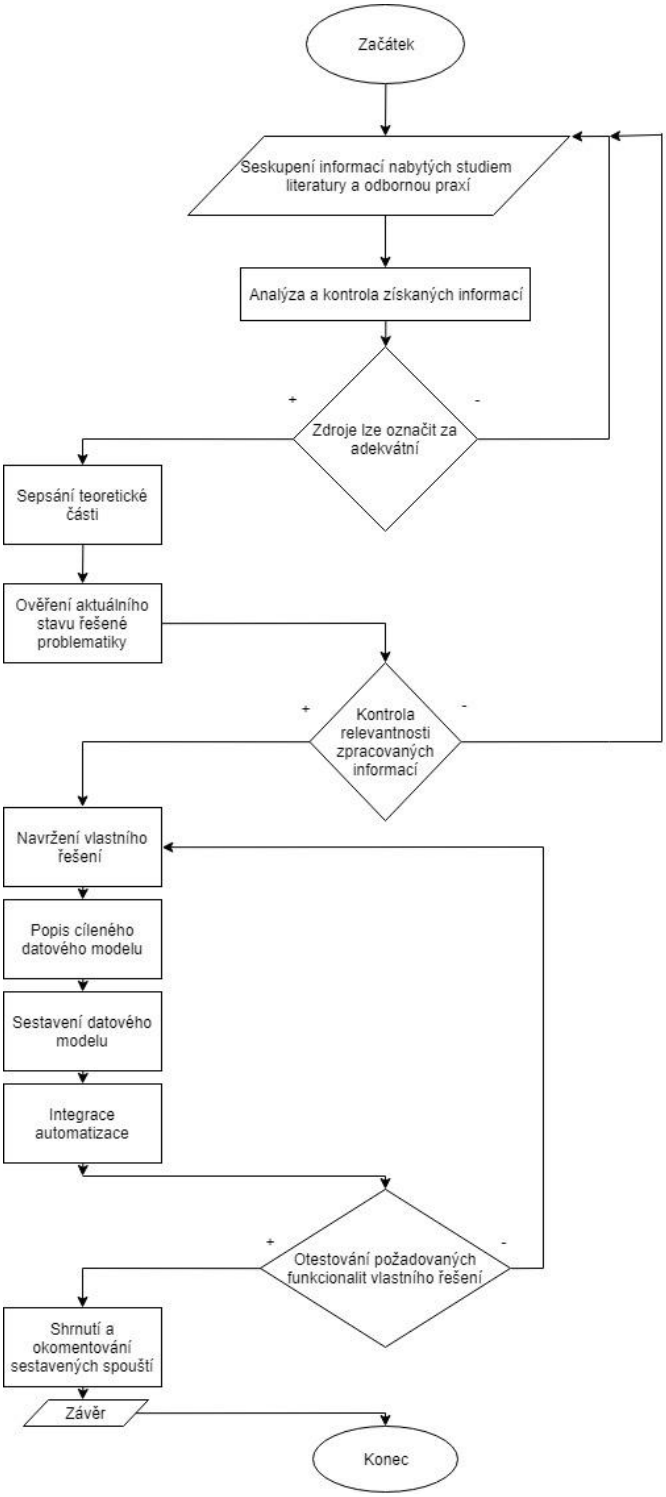
Na základě studia a rozboru dostupných fyzických, ale i elektronických odborných zdrojů budou získané znalosti seskupeny a zpracovány za účelem vytvoření relevantní a adekvátně obsáhlé literární rešerše, jež je obsahem teoretických východisek této práce.

Vlastní navrhované řešení bude prezentováno formou praktické ukázky koncipované na platformě PL/SQL firmy Oracle, konkrétní verzí je 11g XE. Realizace těchto praktických řešení je založena především na splnění identifikovaných požadavků a naplnění stanovených očekávání, včetně demonstrace přínosu každé uvedené spouště.

Na podkladě syntézy teoretických poznatků a dosažených výsledků vlastního řešení budou formulovány závěry této bakalářské práce a následně zobecněny pro další možná použití.

Zvolený postup řešení lze graficky znázornit následovně - viz. obrázek č.1:

Obrázek č. 1 – Diagram postupu řešení



Zdroj: Autor

3 Teoretická východiska

3.1 Databáze

Databáze (též datová základna, či databanka) je organizovaným souhrnem dat s pevnou strukturou záznamů. Jedná se o kolekci schémat, tabulek, dotazů, pohledů, triggerů a dalších objektů. Data jsou typicky organizována takovým způsobem, aby co nejlépe vystihla modelovanou reálnou situaci i s jejími patřičnými aspekty.

S databázemi samotnými jsou úzce spojené, respektive v širším slova smyslu jsou jejich součástí softwarové prostředky, které zprostředkovávají interakci uživatele s databází a umožňují mu tak přístup a manipulaci s daty. Uživatel tudíž nikdy nemá přístup k datům přímo. Tyto softwarové prostředky jsou označovány jako Systém řízení báze dat (neboli zkráceně SŘBD). SŘBD by běžně mělo umožnit definování, tvorbu, dotazování, aktualizaci a administraci databází. [7]

3.2 Vznik databází

Vznik databází se v jejich současné podobě datuje zpět do 50.let dvacátého století. Potřeba záznamu, a tedy uchování, případně následné třídění dat je ale výrazně starší. První doložený nález snahy uchovávat data ve větší míře pochází ze 12. století z území Sýrie – Ugaritu. Zde bylo nalezeno velké množství hliněných tabulek s diplomatickými texty. Lze tedy hovořit o snaze data hromadně organizovat a ukládat, nikoliv ale třídit.

Hovoří-li se však o přímém předchůdci počítačových databází, stává se zásadním 18. století za přírodovědce Carla Linnaea, který zavedl pro třídění přírodovědných druhů systém, ve kterém byl každý zástupce druhu uváděn na samostatném listu papíru. Tento systém je znám jakožto kartotéka. Díky kartotékám bylo možno záznamy snadno organizovat a doplňovat. [8]

3.2.1 Digitalizace

Díky přispění státních úřadů představila roku 1951 firma IBM nový stroj pod názvem UNIVAC I. Jedná se o první sériově vyráběný digitální počítač pro komerční využití. První databázový jazyk se objevuje roku 1959 pod názvem COBOL. Ačkoliv oproti předcházejícím způsobům ukládání představoval nepopiratelný posun, časem se taktéž projevila neefektivnost, a to při používání strojového kódu pro databázové úlohy. [8] [2]

3.3 Nástup SŘBD

Roku 1961 byl Charlesem Bachmanem z General Electric představen první datový sklad. Prvky a znaky databázového managementu z tohoto datového skladu vedly roku 1965 na CODASYL konferenci k myšlence a následně vytvoření koncepce databázových systémů. Vznikl výbor Database Task Group, který tuto myšlenku dále rozvedl a představil pojmy: Systém řízení báze dat, integrita dat, datový model, schéma databáze, atomita, entita, atribut entity, vazba mezi entitami. [9]

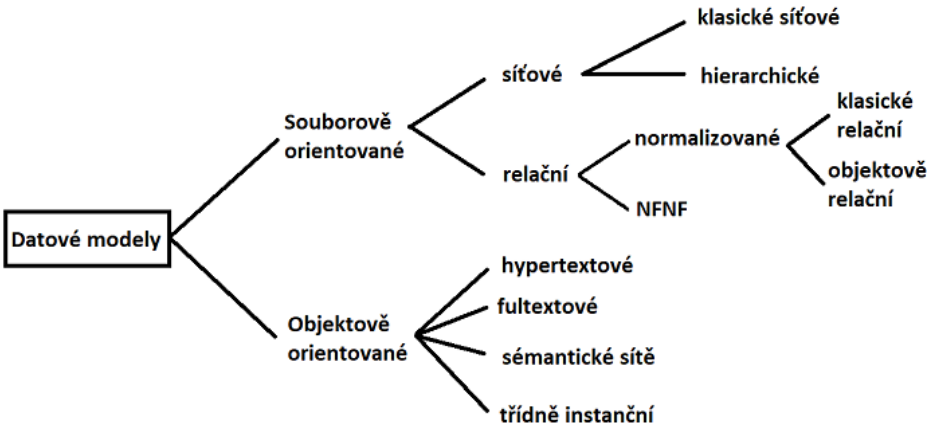
3.3.1 Entita a atribut

Databáze je souhrnem dat sloužících k popisu reálného světa (např. evidence kantorů, sklad skript, evidence knihovny a jiné). Entitou je prvek reálného světa (např. student, kantor, škola atp.), jenž je popsán pomocí svých vlastností (charakteristik). Tyto charakteristiky jsou považovány za atributy (jméno studenta, příjmení, studovaný ročník, stav, plat kantora, vyučované předměty a další). [8]

3.3.2 Databázový model

V této době (1965-1970) vzniká pojem databázový model. Původně byl zaveden matematiky, v první řadě jako prostředek k popisu databáze. Zpočátku se dělily databázové systémy především na dva základní modely – hierarchický a síťový. Pro zařazení a úplnost je na obrázku č. 2 uvedeno plné třídění druhů datových modelů. [10] [12]

Obrázek č. 2 – Druhy datových modelů [2]



Zdroj: KROENKE, D.: Databáze. 1.vydání. Computer press, 2015

3.4 Relační model báze dat

V 70. letech se ukázalo, jak nedostatečné tyto dva základní modely jsou. To dalo vzniknout novému modelu, s kterým přichází roku 1970 Edgar Frank Codd. Toho roku ve svém díle, nazvaném Model of Data for Large Shared Data Banks, kritizoval předchozí struktury, a především databáze stromových struktur. Přišel s tvrzením, že daleko praktičtější je uspořádání dat do tabulek, tak aby se s nimi dalo pracovat nezávisle na informacích o vzájemných vazbách, řazení a indexování. Jednalo se o relační model báze dat, jenž se stal standardem a používá se dodnes. Relační teorie zahrnuje základní operace při práci s daty, především pak selekci, projekci, spojení, sjednocení, kartézský součin a rozdíl. Díky zahrnutí těchto funkcí bylo možné uskutečnit veškeré potřebné operace. Při tvorbě dotazovacího jazyka se Frank Codd soustředil na použití srozumitelných výrazů z běžné angličtiny, tak aby jeho použití bylo co možná nejintuitivnější. [1]

3.5 Structured Query Language

Larry Ellison myšlenky Edgara Codda a jazyk SQL implementoval do vlastního databázového serveru nazvaného Oracle. Při uvádění svého produktu na trh se mu dokonce podařilo porazit konkurenční IBM, a tak se roku 1979 Oracle stává prvním komerčně dostupným relačním systémem řízení báze dat.

Roku 1987 byl jazyk SQL přijat jako standard, a to jak americkou organizací ANSI (American National Standards Institute), tak i mezinárodní organizací ISO (International Organization for Standardization). Na konci tohoto desetiletí trhu jednoznačně vévodil systém Oracle s SQL. [2]

SQL se řadí mezi tzv. neprocedurální programovací jazyky. V praxi to znamená, že se kód jazyka SQL nepíše v samostatném programu (příkladem může být jazyk C nebo Pascal), ale vkládá se do jiného programovacího jazyka, jež je procedurální. Tím pádem se se samotným jazykem SQL může pracovat pouze pokud se terminálem uživatel připojí na SQL server a na příkazový řádek bude zadávat přímo příkazy jazyka SQL. Platí zde pravidlo, že se nikdy nepracuje přímo s daty. Místo toho jsou na server odesílány požadavky na jejich načtení, přidání, změnu či vymazání. Server za uživatele následně požadované operace provede a předá mu výsledky. [1]

SQL se skládá z několika částí – nástrojů. Každá tato část má své určité uživatele (administrátory, návrháře databázových systémů, uživatele a programátory). V první řadě se skládá z jazyka DDL – Data Definition Language. Jde o jazyk sloužící k vytváření

databázových schémat a katalogů. Dále DCL – Data Control Language, což je jazyk sloužící k řízení přístupu k databázi. Následuje jazyk definující způsob ukládání tabulek SDL – Storage Definition Language. Další částí sloužící návrhářům a správcům je VDL – View Definition Language, který určuje vytváření pohledů (pohledem lze rozumět virtuální tabulku složenou z jiných tabulek). Poslední částí je jazyk sloužící pro práci se samotnými daty: DML – Data Manipulation Language, obsahující základní sadu příkazů: INSERT, UPDATE, DELETE a nejhojněji užívaný příkaz SELECT. S jazykem DML nejčastěji pracují koncoví uživatelé a programátoři databázových aplikací. [3]

3.6 Způsob ukládání dat

Základem relačního modelu, respektive relační databáze je pojem relace. Pokud je vynechána matematická definice, lze relaci popsat jako dvourozměrnou tabulku, která se skládá ze sloupců a řádků. Jednotlivé vlastnosti objektů uložených v databázi (atributy entit) jsou umístěny ve sloupcích a údaje uložené v každém jednom řádku tabulky odpovídají aktuálnímu stavu světa neboli stavu entit. Všechny hodnoty nacházející se v jednom sloupci mají stejný datový typ.

Názorně je zobrazeno v jednoduché relační tabulce č. 1, která popisuje entitu pracovníka ve firmě. Ve sloupcích se nachází atributy: identifikační číslo (číslo), jméno, příjmení, datum narození (dat_nar), plat a datum podpisu smlouvy (smlouva_od). [9]

Tabulka č. 1 – Seznam pracovníků [9]

ČÍSLO	JMÉNO	PŘÍJMENÍ	DAT_NAR	PLAT	SMLOUVA_OD
1	Ondřej	Buben	11.08.90	20000	01.01.2015
2	Petr	Sokol	17.04.79	30000	08.12.1999
3	Katka	Novotná	30.05.67	40000	06.04.1990

Zdroj: SKŘIVAN, J.: Databáze a jazyk SQL

Pro budování celé databáze je základním stavebním kamenem tabulka. Relace tedy musí odpovídat celé tabulce a jednomu prvku relace odpovídá konkrétní jeden řádek tabulky. Jeden řádek se nazývá databázovým záznamem. Soubor relací (tabulek) je poté celou databází (relačním schématem). [9]

3.6.1 Relace tabulek

V relačních databázích je nutné zajištění správného propojení souvisejících tabulek pro správnou komunikaci, funkci, efektivitu a výkon. Takové propojení se navazuje pomocí atributů primárního klíče a cizího klíče (FOREIGN KEY).

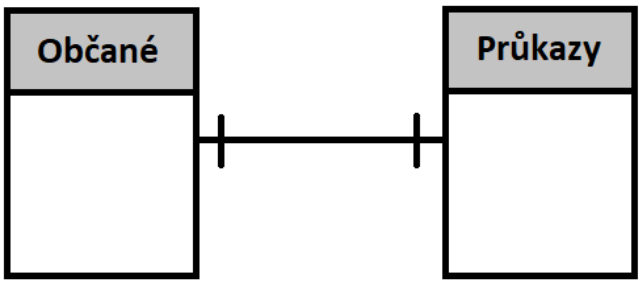
Díky propojení primárního klíče jedné tabulky a cizího klíče tabulky druhé vzniká relace. Celkem mohou být tři druhy těchto propojení. [3]

3.6.1.1 1:1

Vztah, kdy je přiřazen jeden řádek z první tabulky jednomu řádku z tabulky druhé. Jednoduchým příkladem může být občan – občanský průkaz. Každý občan má svůj jeden občanský průkaz a každý občanský průkaz patří pouze jedné osobě – viz. obrázek č.3.

Této relace se využívá pro rozdělení rozsáhlé tabulky, či k oddělení části tabulky z důvodu zabezpečení nebo z důvodu souvislosti uložených informací pouze s částí hlavní tabulky. [5]

Obrázek č. 3 – Relace 1:1[5]



Zdroj: Příručka k relacím mezi tabulkami. Microsoft

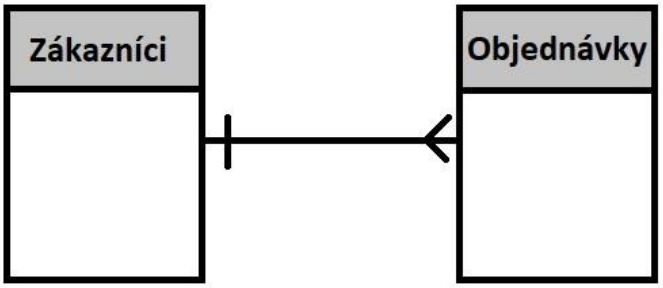
3.6.1.2 1: N

Jeden řádek z první tabulky lze propojit s jedním, nebo více řádky z druhé tabulky. Příkladem může být databáze sloužící k zaznamenávání objednávek. Tato databáze bude obsahovat tabulky Zákazníci a Objednávky. Jedním zákazníkem může být vytvořeno libovolné množství objednávek, proto každý jeden zákazník v tabulce Zákazníci může mít několik záznamů v tabulce Objednávky. [5]

Propojení těchto tabulek by muselo být provedeno pomocí atributu, který jednoznačně určí záznam na straně „1“, tedy ID zákazníka (v tabulce Zákazníci představuje primární klíč).

Tento atribut se následně vloží jakožto cizí klíč do tabulky Objednávky. Relace mezi takovými tabulkami je poté typu 1: N – viz. obrázek č. 4. [5]

Obrázek č. 4 – Relace 1: N [5]



Zdroj: Příručka k relacím mezi tabulkami. Microsoft

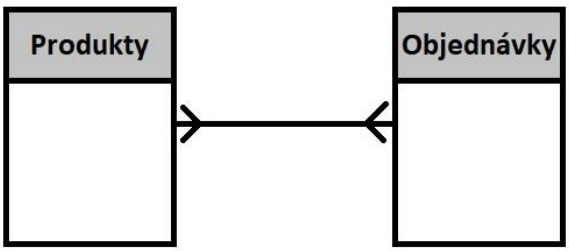
3.6.1.3 M: N

Propojení vícero řádků první tabulky s více řádky druhé tabulky.
Například, relace tabulek Produkty a Objednávky. V jedné objednávce bude více různých produktů, ale zároveň jeden konkrétní produkt může být ve více různých objednávkách. Tudíž každý záznam z tabulky Objednávky může být propojen s větším počtem záznamů z tabulky Produkty, a naopak pro každý jeden záznam z tabulky Produkty se může vyskytovat větší počet záznamů v tabulce Objednávky – viz obrázek č. 5. [9]

Takové propojení se v praxi provádí za pomoci třetí tabulky, která vzniká po rozdělení jedné výchozí relace typu M: N na dvě typu 1: N. Tato tabulka se nejčastěji nazývá spojená tabulka – viz obrázek č. 6.

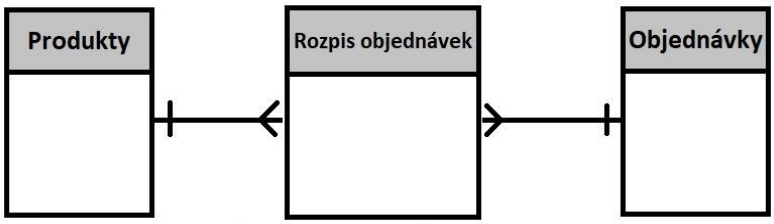
Primární klíč z výchozích dvou tabulek bude vložen do tabulky třetí, díky čemuž tato tabulka zaznamenává každý výskyt nebo instanci relace. Pokud bude uveden výše zmíněný příklad, může být vytvořena tabulka Rozpis objednávek. Relace M: N je zde následně vyjádřena dvěma relacemi 1: N (na straně N Rozpis objednávek). Díky tomu může být v jedné objednávce několik produktů a tentýž produkt v několika objednávkách. [5]

Obrázek č. 5 – Relace M: N [5]



Zdroj: Příručka k relacím mezi tabulkami. Microsoft

Obrázek č. 6 – Relace 1:M: N:1 [5]



Zdroj: Příručka k relacím mezi tabulkami. Microsoft

3.7 Datová integrita

Integrita dat je v relačních databázích řízena pomocí databázových objektů označovaných jako omezení. Tyto objekty kladou určitá pravidla na data vkládaná do sloupců tabulky, za účelem zajištění přesnosti a konzistence dat v relační databázi.

Pokaždé, když někdo vkládá data do tabulek, tak může nevědomě narušovat přesnost a konzistenci dat. Proto je potřeba prostředků, které zajistí konzistentní způsob ukládání. Protože jsou pro veškeré uživatele všechna data uložena ve stejné databázi, a jelikož je možná vícenásobná a současná aktualizace dat, tak nelze mít například od jednoho pracovníka ukládaná telefonní čísla ve formátu 123456789 a od druhého ve formátu 123 456 789.

Obecně platí, že bez aplikace omezení by správa velkého objemu dat byla velice složitá, pokud ne přímo nemožná. Pokud se ve velkých podnicích pracuje s databázemi, které obsahují tisíce až milióny záznamů, pak je nepředstavitelné, aby docházelo takovým způsobem k narušení logické struktury dat.

Zajištění integrity databáze musí být garantováno prostředky databázového serveru s tím, že již definované omezení je uplatněno bez ohledu na to, jaký prostředek je k přístupu

do databáze použit. Tato omezení nemohou být potlačena uživatelem a vytvořené aplikace musí být zcela nezávislé na jejich případných změnách. [4] [6]

3.7.1 Druhy integritních omezení

a) Entitní integrita

Pro zajištění entitní integrity je zásadní jednoznačná identifikace každého řádku dané relační tabulky. Každý řádek, který zachycuje jeden výskyt entity či vztahu objektivní reality, musí být odlišitelný od ostatních.

Proto se v každé relační tabulce musí vyskytovat minimálně jeden primární klíč, díky kterému je možné tuto identifikaci snadno provést. Tento klíč má bezpodmínečně na každém řádku tabulky jinou hodnotu. Pakliže by se hodnota na jednom či více řádcích shodovala, poté by již nemohl plnit svůj účel.

Díky zabezpečení této integrity se nemůže stát, že by došlo k uložení řádku bez vyplněného či s duplicitním primárním klíčem. [6]

b) Referenční integrita

Je zárukou konzistentních a přesných dat napříč logicky souvisejícími relacemi. Definuje vazby mezi logicky nadřízenými a podřízenými tabulkami. Tyto vazby jsou definovány pomocí cizího klíče (foreign key), který deklaruje závislost jedné hodnoty jednoho sloupce první tabulky na jedné hodnotě jednoho sloupce tabulky druhé, díky tomu jednoznačně propojuje jeden řádek nadřízené tabulky s jedním řádkem podřízené tabulky. Z této vazby lze odvodit, že primární klíč jedné tabulky bude cizím klíčem tabulky druhé. [6]

c) Doménová integrita

Během vytváření databáze se každému sloupci musí přiřadit konkrétní datový typ, ve kterém budou hodnoty do sloupců ukládány. Jinými slovy lze doménou chápat množinu všech přípustných hodnot určitého atributu. [6]

Všechna tato omezení se řadí do skupiny deklarativního typu. To znamená, že integrita je zajištěna skrze explicitní deklaraci jakožto součást databázového schématu. Jsou to tedy prostředky, díky kterým je možná specifikace integritních omezení jako přímé součásti definice struktury databáze. Prakticky se tak činí pomocí rozšíření syntaxe příkazu sloužícího k definici struktury relační tabulky, tedy v rámci příkazu CREATE TABLE či ALTER TABLE. [4]

3.7.2 Typy omezení

3.7.2.1 Omezení typu NOT NULL

Během vytváření tabulky musí být každému sloupci přiřazen konkrétní datový typ. Ten databázi určuje, jaké hodnoty lze do daného sloupce ukládat. Mezi nejobvyklejšími datovými typy jsou znak, číslo a datum. Při vytváření sloupce lze uložit omezení NOT NULL, přičemž hodnota NULL značí neznámou či chybějící hodnotu. Vždy když je vložen do tabulky nový řádek s určitým nevyplněným sloupcem, pak tento sloupec pro daný řádek nabývá hodnoty NULL. Při implementaci tohoto omezení na určitý sloupec lze tedy říci, že daný sloupec musí obsahovat data – viz přiložený příklad. [3]

```
SQL> CREATE TABLE STUDENT
   2 (STUD_ID    NUMBER(10)    NOT NULL,
   3  STUD_GN    CHAR(30)      NOT NULL,
   4  STUD_SN    CHAR(30)      NOT NULL,
   5  ADR        VARCHAR2(30),
   6  CITY       VARCHAR2(30),
   7  CNTRY      CHAR(30),
   8  ZIP        NUMBER(6) ,
   9  PHONE      NUMBER(10));
```

V tomto příkladu jsou povinné sloupce STUD_ID, STUD_GN a STUD_SN. STUD_ID, STUD_GN a STUD_SN obsahují totiž již omezení NOT NULL, jsou tedy povinné.

3.7.2.2 Omezení typu „PRIMARY KEY“

Označuje se jím jeden nebo více sloupců tabulky, které zajišťují jedinečnost každého řádku a jednoznačnou identifikaci každého záznamu v tabulce. V naprosté většině tabulek je primární klíč složen pouze z jednoho sloupce, může být však složen i z více.

V níže zmíněném příkladu je tabulka studentů, kde k jednoznačné identifikaci dostatečně slouží atribut identifikačního čísla studenta – STUD_ID. Vždy je cílem, aby každý záznam obsahoval unikátní hodnotu atributu, jež je primárním klíčem. Primární klíč se určuje opět při vytváření tabulky – viz. příklad. [4]

```
SQL> CREATE TABLE STUDENT
   2 (STUD_ID    NUMBER(10)    NOT NULL   PRIMARY KEY,
   3  STUD_GN    CHAR(30)      NOT NULL,
   4  STUD_SN    CHAR(30)      NOT NULL,
   5  ADR        VARCHAR2(30),
   6  CITY       VARCHAR2(30),
   7  CNTRY      CHAR(30),
   8  ZIP        NUMBER(6));
```

Další možná řešení jsou: definování primárního klíče až za seznamem sloupců v rámci CREATE TABLE – viz příklad a), či při doplnění až po vytvoření tabulky s využitím příkazu ALTER TABLE – viz příklad b). [6]

a)

```
SQL> CREATE TABLE STUDENT
   2 (STUD_ID    NUMBER(10)    NOT NULL,
   3  STUD_GN    CHAR(30)      NOT NULL,
   4  STUD_SN    CHAR(30)      NOT NULL,
   5  ADR        VARCHAR2(30),
   6  CITY       VARCHAR2(30),
   7  CNTRY      CHAR(30),
   8  ZIP        NUMBER(6),
   9  PHONE      NUMBER(10),
   10 PRIMARY KEY (STUD_ID));
```

b)

```
SQL> ALTER TABLE STUDENT ADD CONSTRAINT STUD_PK PRIMARY
     KEY (STUD_ID);
```

V příkladu b) byla tabulka již vytvořena a omezení – primární klíč je k ní přikládán pomocí příkazu ALTER TABLE. Primárním klíčem bude sloupec STUD_ID.

3.7.2.3 Omezení typu „FOREIGN KEY“

Vyskytuje se u tabulek, kde existuje vazba rodič – potomek. Konkrétně se jedná o jeden sloupec v tabulce potomka, jenž se odkazuje na jeden sloupec v tabulce rodiče. U relační databáze se jedná o hlavní prostředek zajištění referenční integrity napříč tabulkami. Cizí klíč v tabulce potomka představuje stejnou hodnotu, jako primární klíč rodiče. Díky tomu je

zajištěna existence odpovídajících dat v obou tabulkách. Příklad níže uvádí tabulky STUDENT a SCHOLARSHIP. [3]

```
SQL> CREATE TABLE STUDENT
   2 (STUD_ID    NUMBER(10)    NOT NULL   PRIMARY KEY,
   3  STUD_GN    CHAR(30)      NOT NULL,
   4  STUD_SN    CHAR(30)      NOT NULL,
   5  ADR        VARCHAR2(30),
   6  CITY       VARCHAR2(30),
   7  CNTRY      CHAR(30),
   8  ZIP        NUMBER(6) ,
   9  PHONE      NUMBER(10));

SQL> CREATE TABLE SCHOLARSHIP
   2 (STUD_ID    NUMBER(10)    NOT NULL,
   3  AMOUNT     INT           ,
   4  FOREIGN KEY STUD_ID_FK (STUD_ID) REFERENCES
      STUDENT(STUD_ID));
```

V tabulce SCHOLARSHIP je jako cizí klíč označen sloupec STUD_ID, jež se odkazuje skrze klauzuli REFERENCES do tabulky STUDENT na sloupec STUD_ID. Toto zaručuje existenci odpovídající hodnoty z pole STUD_ID tabulky STUDENT pro všechny hodnoty z pole STUD_ID tabulky SCHOLARSHIP.

Rodičem je tabulka STUDENT a logicky podřízeným potomkem je tabulka SCHOLARSHIP. Tímto vztahem, který je deklarován ve výše uvedeném příkladu, je dáno několik pravidel. Aby bylo možno vložit do sloupce STUD_ID potomka určitou hodnotu, musí se nejdříve tato hodnota nacházet ve sloupci STUD_ID rodičovské tabulky. A naopak, předtím, než bude odstraněna určitá hodnota ze sloupce STUD_ID rodičovské tabulky, musí být prvně odstraněna odpovídající hodnota sloupce STUD_ID z tabulky potomka. Obrázek č. 7 znázorňuje vztah rodič – potomek.

Obrázek č. 7 – Vztah rodič – potomek [6]

Zdroj: STEPHENS, R; PLEW, R; D.JONES, A.: Naučte se SQL za 28 dní

Stejně tak, jako u primárního klíče, je možné přidávat cizí klíč až po vytvoření tabulky pomocí příkazu ALTER TABLE – viz příklad níže. [6]

```
SQL> ALTER TABLE SCHOLARSHIP ADD CONSTRAINT STUD_ID_FK
FOREIGN  KEY (STUD_ID) REFERENCES STUDENT (STUD_ID);
```

3.7.2.4 Omezení typu jedinečnost

Svým způsobem se podobá primárnímu klíči, a to především unikátností každé hodnoty uložené v daném sloupci. Ne vždy je však cílem mít unikátní hodnotu pouze u primárního klíče, navíc může být vyžadována u většího počtu sloupců. Proto existuje integritní omezení UNIQUE – viz. příklad níže. [4] [6]

```
SQL> CREATE TABLE STUDENT
   2 (STUD_ID    NUMBER(10)    NOT NULL,  PRIMARY KEY,
   3  STUD_GN    CHAR(30)      NOT NULL,
   4  STUD_SN    CHAR(30)      NOT NULL,
   5  ADR        VARCHAR2(30),
   6  CITY       VARCHAR2(30),
   7  CNTRY      CHAR(30),
   8  ZIP        NUMBER(5) ,
   9  EMAIL      VARCHAR2(30) UNIQUE,
  10 PHONE      NUMBER(10) UNIQUE);
```

V takto vytvořené tabulce STUDENT není díky omezení UNIQUE možné, aby dva studenti měli shodné identifikační číslo, email a telefonní číslo. K propojení tabulek (viz. primární a cizí klíč) však lze využít pouze sloupce STUD_ID.

3.7.2.5 Omezení typu kontrola

Slouží k omezení dat, která lze do daného sloupce vkládat. Zpravidla se užívá v rámci příkazu CREATE TABLE, jelikož v případě použití příkazu ALTER TABLE by byla kontrolována pouze data vložená až po jeho aplikaci. Využívá se pro důkladnější ochranu dat, tak jako je uvedeno na příkladu níže. [4] [6]

```
SQL> CREATE TABLE STUDENT
   2 (STUD_ID    NUMBER(10)    NOT NULL,  PRIMARY KEY,
   3  STUD_GN    CHAR(30)      NOT NULL,
   4  STUD_SN    CHAR(30)      NOT NULL,
   5  ADR        VARCHAR2(30),
   6  CITY       VARCHAR2(30),
   7  CNTRY      CHAR(3),
   8  ZIP        NUMBER(6),
   9  EMAIL      VARCHAR2(30) UNIQUE,
  10 PHONE      NUMBER(10) UNIQUE,
  11 CONSTRAINT CHK_STUD_COUNTRY
     CHECK (COUNTRY in
('CR','SK','ENG','DE','RU','UKR')));
```

Omezení typu kontrola je v tomto příkladu užito na sloupec COUNTRY, kde je požadavek na použití hodnoty z výčtu. Omezení lze použít i pro ověření číselných hodnot – viz příklad níže. [6]

a) Na konci příkazu CREATE TABLE

```
CONSTRAINT CHK_PAY CHECK (HOUR_RATE > 100));
```

b) Za definicí sloupce

```
HOUR_RATE int NOT NULL CHECK (HOUR_RATE > 100));
```

Zde je uplatňována kontrola výše hodinové mzdy, kdy je její minimálně výše 100 korun za hodinu práce. [6]

Dalším způsobem zajištění integrity dat nabízí procedurální realizace integritních omezení. Ta se na rozdíl od deklarativního způsobu provádí mimo definici struktury databáze. Toto řešení je založeno na využití speciálních databázových procedur. Tyto procedury se nazývají databázové spouště (database triggers). [6]

3.8 Databázové triggery

Jedná se o zvláštní formu uložené procedury, která se spustí v reakci na provedení specifikované operace, či události přímo databázovým serverem. Spouště se vždy váží k určité tabulce a podle toho, jak je definován, se spouští před, po nebo místo provedení určité operace. Databáze mohou detekovat pouze systémem definované události. Vlastní události nelze ručně vytvářet. [2] [14]

Lze vytvářet spouště, které se spustí v reakci na jednu z těchto operací: [13]

a) DML operace na určité tabulce nebo pohledu, které provádí jakýkoli uživatel.
   Konkrétně se jedná o spouště:
   - BEFORE INSERT /AFTER INSERT
     Ty se vyvolají před či po vložení dat do tabulky.
   - BEFORE UPDATE / AFTER UPDATE
     Spustí se před nebo po aktualizaci dat v tabulce.
   - BEFORE DELETE / AFTER DELETE
     Spouštěné před či po odstranění dat z tabulky.

b) DDL operace. Tyto spouště lze spojit jak s databází, tak se schématem.
   Zástupci jsou:
   - BEFORE CREATE / AFTER CREATE
     Sepnutí spouště probíhá před nebo po vytvoření určitého objektu.
   - BEFORE ALTER / AFTER ALTER
     Vyvolávány před či po modifikaci určitého objektu.
   - BEFORE DROP / AFTER DROP
     Po či před zahozením databázového objektu.

c) Databázové události, dělící se na:
   Systémové události:
   - SERVERERROR – lze vázat na specifickou chybu, nebo jakoukoliv
   - STARTUP – spuštění po zapnutí databáze
   - SHUTDOWN – spuštění po vypnutí databáze

Uživatelské události:

- LOGON – spuštění po přihlášení uživatele
- LOGOFF – spuštění po odhlášení uživatele

Kroky a operace, jež mají být provedeny k udržení integrity databáze či k usnadnění práce s ní, musí být explicitně naprogramovány pomocí příkazů jazyka DML. Jedná se o označenou množinu příkazů uloženou na databázovém serveru jako součást samotné databáze. Tato množina se váže na předem definované aktualizující operace s konkrétní relační tabulkou. Při výskytu definované operace následně databázový server automaticky provede tuto množinu příkazů, a to vždy bez ohledu na příčinu vyvolání, tedy může se spustit i opakovaně. Díky tomu je garantováno provedení příkazů při jakémkoli způsobu aktualizace databáze. [13]

Hlavní rozdíl mezi běžnou uloženou procedurou a spouští spočívá ve způsobu vyvolání. Uložená procedura je spuštěna explicitně uživatelem nebo aplikací či samotnou spouští. Spoušť je však spuštěna automaticky nástrojem Oracle v momentu, kdy se stane definovaná událost, bez ohledu na uživatele, který je připojen nebo jaká aplikace je zrovna používána. Lze ji ale v případě potřeby vypnout. [13]

Syntaxe obsahuje nejdříve název spouště a název tabulky, následně deklaraci příkazu, který spoušť aktivuje (INSERT, UPDATE či DELETE) a až poté vlastní množinu příkazů k provedení. [6]

3.8.1 Praktické využití triggerů

Spouště mohou mít v databázích velmi obsáhlé využití, Oracle ve své aktuální příručce (pro verzi 12c) doporučuje použití triggerů především pro tyto účely: [14]

- Logování událostí
- Sběr statistiky přístupů do tabulky
- Modifikaci tabulkových dat, když jsou DML příkazy použity na pohledy
- Zajištění referenční integrity, když se tabulky potomka a rodiče nachází na rozdílných uzlech distribuované databáze
- Publikování informací o databázových událostech a uživatelských událostech
- Blokaci DML operací na tabulky například po regulérních úředních hodinách
- Prevenci neplatných transakcí
- Zajištění komplexních úředních či referenčně integritních pravidel, která nelze definovat pomocí základních omezení:

- NOT NULL, UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DELETE CASCADE
- DELETE SET NULL [14]

3.8.2 Přínos a nevýhody triggerů

Spouště přináší zlepšení především z hlediska časové úspory a zvýšené míry dohledu nad prováděnými procesy.

Není již nutné vykonávat celou řadu úkonů ručně, ale místo toho se při předem definovaných situacích požadované operace provedou přímo SŘBD. Určité procesy lze automatizovat takovým způsobem, že je není nutné pokaždé definovat znovu a provádět množství zdlouhavých kroků opakovaně, místo toho se vždy automaticky provedou databázovým systémem, díky tomu lze urychlit a zkrátit práci zaměstnanců, kteří se místo opakovaných běžných činností mohou zabývat kvalifikovanějšími úkoly. Navíc je odstraněn lidský faktor, což v konečném důsledku znamená redukci chyb.

Spouště mají přínos také z hlediska zvýšené kontroly nad probíhajícími operacemi, pomocí automatizace totiž lze určit nejen kdy mají proběhnout, ale také jaké musí být jejich návaznosti. Díky těmto faktům je zaručena zamýšlená funkcionalita.

Spouště též umožňují využití konstrukcí, které jsou běžné pro většinu programovacích jazyků. Podporují totiž deklaraci lokálních proměnných, příkazy pro kontrolu procedur, přiřazení výsledků výrazů do proměnných a také umožňují ovládání chybových hlášek. Jejich velkou výhodou je zlepšení výkonu v prostředí client/server. Všechna pravidla stanovená pomocí spouští jsou provedena na serveru před vrácením výsledků, a to konzistentně pro každého uživatele.

Všechny tyto body budou v této práci demonstrovány na praktických příkladech.[18]

Oproti výhodám lze postavit pouze málo nevýhod, především se jedná o personální náklady spojené se zaměstnáním IT specialistů, kteří budou zastřešovat podporu pro provoz a vývoj automatizace. V případě větších firem jsou takové podpůrné skupiny řešeny interně, v případě středních a menších podniků jsou řešeny outsourcingem. [18]

3.8.3 Komparace triggerů a omezení

Ačkoli spouště, ale i omezení zajišťují dodržení určitých integritních pravidel, tak se značně liší.

Spoušť vždy pracuje pouze s novými daty. Například může zabránit DML příkazu vložit novou prázdnou hodnotu do sloupce tabulky, ale jestliže sloupec již prázdnou hodnotu obsahoval před vložením spouště do databáze či pokud byla spoušť v době vložení deaktivována, poté bude tato hodnota v tabulce uložena. [14]

3.8.4 CREATE TRIGGER

Základní konstrukce spouště bude vypadat následovně: [15]

```
CREATE [OR REPLACE] TRIGGER [schéma.]název_spouště
{BEFORE | AFTER | INSTEAD OF} {DELETE [OR] | INSERT [OR] |
UPDATE}
ON [schéma.]{ název_tabulky | název_pohledu }
FOR EACH {ROW | STATEMENT}
[WHEN (podmínka)]
BEGIN
… blok PLS/SQL kódu …
END;
```

3.8.4.1 :OLD a :NEW prefix

Pokud nastane situace, kdy je potřeba použít hodnotu z tabulky před změnou, lze použít speciálních předpon, společně s klauzulí REFERENCING. Pro novou hodnotu poté :NEW.jméno_sloupce pro původní :OLD.jméno_sloupce. [15]

3.8.5 Konstrukce triggerů

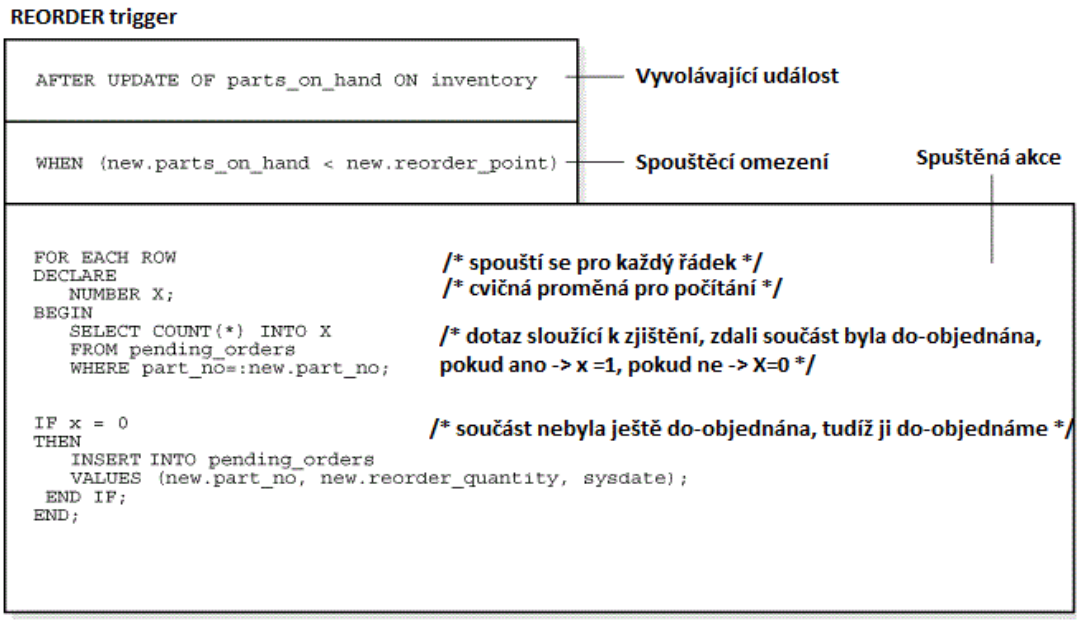
Vytvářet je lze nad tabulkou či nad pohledem příkazem CREATE TRIGGER, často též ve formě CREATE OR REPLACE TRIGGER. Pokud by již existovala spoušť se stejným názvem, jako chce uživatel použít, tak ho nahradí, resp. přepíše.

Vždy se skládají ze tří povinných částí:

1. Vyvolávající událost či příkaz
2. Spouštěcí omezení
3. Spuštěná akce

Na obrázku č. 8 je příklad spouště z označenými hlavními částmi.

Obrázek č. 8 – Vzorová spoušť [13]



Zdroj: Database Concepts. ORACLE Help Center

Vyvolávající událost je vždy složena z DML příkazů: DELETE, INSERT nebo UPDATE. Pokud bychom chtěli vytvořit spoušť, která by se spustila v návaznosti na příkaz MERGE, poté by bylo nutné vytvořit spouště na INSERT a UPDATE příkazy, jelikož na ty se operace MERGE rozkládá. Před definováním vyvolávající události se uvádí okamžik spuštění, tedy jestli se má obsah spouště provést před, po nebo místo této události. Navíc lze definovat, zdali se má spoušť spustit pouze jednou při výskytu definované události nebo pokaždé, když je spouštěcí událostí ovlivněn nějaký řádek specifikované tabulky nebo místo této události. V prvním případě se hovoří o „BEFORE/AFTER statement trigger“, též „Statement-level BEFORE/AFTER trigger“, v druhém případě o „BEFORE/AFTER each row trigger“, též „row-level BEFORE/AFTER trigger“ a ve třetím případě se jedná o spoušť typu INSTEAD OF. [11]

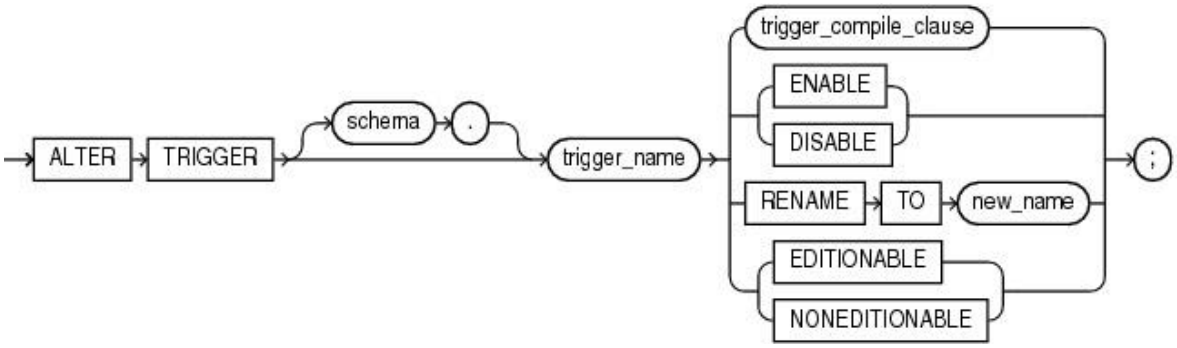
Spouštěcí omezení specifikuje Boolean výraz, tedy true/false, který musí být true, aby se spoušť vyvolala. Pokud výsledek spouštěcího omezení vrací hodnotu false nebo unknown, tak se spoušť neprovede. Dalším případem, kdy se spoušť neprovede, je situace, kdy je podmínka splněna, ale spoušť zakázána. Spouště totiž lze povolit i zakázat (enabled/disabled), primárně je přednastaveno její povolení.

Spuštěnou akcí je procedura sestavující se z bloku PL/SQL kódu, který obsahuje SQL příkazy, které budou provedeny v momentě úspěšného vyvolání spouště. [13]

3.8.6 ALTER TRIGGER

Jak již bylo zmíněno, spouště lze vypnout a zapnout (enable/disable). K této administrativě se používá klauzule ALTER TRIGGER. K tomu, aby uživatel mohl této klauzule využívat u schémat, musí mít přidělená ALTER ANY TRIGGER systémová práva nebo SYSDBA práva (pokud je vlastníkem schématu, poté je mu SYSDBA přiděleno automaticky). Na obrázku č. 9 je znázorněna syntaxe ALTER TRIGGER. [16]

Obrázek č. 9 – ALTER TRIGGER [16]



Zdroj: ALTER TRIGGER Statement. ORACLE Help Center

- Schema – název schéma, které obsahuje trigger, defaultně naše vlastní
- Trigger name – název spouště, kterou se uživatel chystá vypnout/zapnout, případně přejmenovat či jinak upravit

Příklad vypnutí a zapnutí spouště:

```
ALTER TRIGGER update_job_history DISABLE;
ALTER TRIGGER update_job_history ENABLE;
```
[16]

3.8.7 DROP TRIGGER

Mazání spouští proběhne po použití příkazu DROP TRIGGER. Opět, stejně jako u ALTER TABLE klauzule, musí mít uživatel potřebná oprávnění k použití tohoto příkazu. Konkrétně se jedná o DROP ANY TRIGGER systémové oprávnění. Pokud chce uživatel zahodit spoušť v schématu jiného uživatele, musí mít navíc systémové oprávnění ADMINISTER DATABASE TRIGGER. Na obrázku č. 10 je názorně vyobrazena syntaxe příkazu pro zahození. [17]

Obrázek č. 10 – DROP TRIGGER [17]

Zdroj: DROP TRIGGER Statement. ORACLE Help Center

Příklad zahození spouště:

```
DROP TRIGGER update_job_history; [17]
```

3.8.8 Pořadí spouštěných triggerů

Každá spoušť je databázovým nástrojem Oracle spuštěna v momentu, který je definován vyvolávající událostí, na kterou je spoušť nastavena. Pokud však nastává situace, kdy stejná událost splňuje podmínky vyvolání vícero spouští, poté se nejdříve spustí „Statement-level BEFORE trigger“, poté „Row-level trigger“, následně „Row-level AFTER trigger“ a na závěr „Statement-level AFTER trigger“. V případě, že se má spustit větší počet spouští stejného typu, poté je jejich pořadí náhodné. [14]

3.8.9 Řízení chybových hlášek

V použitých příkladech této práce bude hojně využito deklarace vlastních chybových hlášek, proto zde budou zmíněny typy a vysvětlen způsob jejich tvorby, respektive deklarace, očíslování a způsob vyvolání a následného výpisu. [20]

Existují 3 typy chybových hlášek – viz tabulka č. 2:

- Interně definované
- Předdefinované
- Uživatelsky definované

Tabulka č. 2 – Kategorizace chybových hlášek [20]

Kategorie	Definuje	Kód chyby	Název	Implicitně vyvolána	Explicitně vyvolatelná
Interně definovaná	Runtime systém	Vždy	Volitelně	Vždy	Volitelně
Předdefinovaná	Runtime systém	Vždy	Vždy	Vždy	Volitelně
Uživatelem definovaná	Uživatel	Volitelně	Vždy	Nikdy	Vždy

Zdroj: PL/SQL Error Handling. ORACLE

3.8.9.1 Interně definované chybové hlášky

Runtime systém vyvolává tento typ hlášek implicitně, tedy automaticky při každém výskytu specifické chyby. Například pokud dojde k deadlocku a následnému nedostatku paměti, tak se automaticky vypíše interně definovaná chyba: ORA-00060 (detekovaný deadlock během čekání na zdroje) a ORA-27102 (nedostatek paměti). [20]

Zde je na odkazovaném zdroji uveden výpis všech interně definovaných chyb: [19].

3.8.9.2 Předdefinované chybové hlášky

Předdefinované chybové hlášky jsou taktéž interně definované, ale na rozdíl od předchozí kategorie mají vždy konkrétní jméno. Například pokud se objeví problém s uložištěm, bude vypsána chyba ORA-06500 (PL/SQL: problém s uložištěm) s předdefinovaným názvem STORAGE_ERROR. [20]

Na uvedeném zdroji je uveden seznam všech předdefinovaných chyb: [21].

3.8.9.3 Uživatelsky definované chybové hlášky

Uživatel si může nadefinovat své vlastní chybové hlášky v deklarační části jakéhokoli PL/SQL bloku, podprogramu, nebo balíčku. Například lze mít deklarovanou chybovou hlášku, jež uživatele bude informovat o nedodržení referenční integrity.

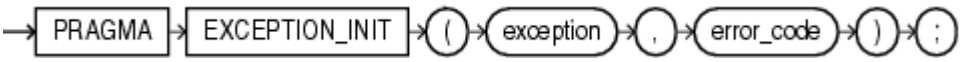
Vzhledem k faktu, že tento způsob realizace chybových hlášek bude ve vlastním řešení hojně využit, tak zde bude syntaxe a sémantika tohoto typu uvedena. [20]

K asociaci uživatelsky definované hlášky a chybového kódu slouží klauzule EXCEPTION_INIT PRAGMA. Použití je však striktně vázáno na tu samou deklarační část, ve které je deklarována ona chybová hláška. Deklarace samotné chybové hlášky musí navíc vždy předcházet této klauzuli.

Způsob zápisu je vyznačen na obrázku č.11.

Syntaxe:

Obrázek č. 11 – PRAGMA EXCEPTION_INIT [21]



Zdroj: Predefined Exceptions. ORACLE

- exception

  Jméno chybové hlášky deklarované před tímto krokem
- error_code

  Chybový kód, který chceme asociovat s chybovou hláškou. Může nabývat jakékoli záporné celočíselné hodnoty, která je větší než -10000000, kromě hodnoty -1403, jelikož tato hodnota je vyhrazena pro chybu “no data found“, jež je zástupcem předdefinovaných chybových hlášek. [21]

Na následujících stránkách je zobrazeno vzorové použití chybových hlášek v rámci spouště řešící zajištění referenční integrity.

3.9 Procedurální realizace referenční integrity

Jak již bylo zmíněno, integritní omezení lze realizovat i procedurálně. V tomto případě pomocí databázových spouští. Cílem tohoto ukázkového příkladu je názorně demonstrovat celou konstrukci spouště, a to včetně konstrukce výchozích tabulek.

3.9.1 Vytvoření výchozích tabulek

Pro demonstraci budou vytvořeny tabulky pro záznam informací o zaměstnancích a o odděleních, ve kterých pracují. V tabulce zaměstnanců (emp) bude primárním klíčem zaměstnancovo číslo (Empno) a cizím klíčem číslo oddělení (Deptno), vůči druhé uvedené tabulce bude zastupovat ve vztahu Rodič – potomek pozici potomka. Rodičovská tabulka oddělení (dept) bude mít za primární klíč atribut číslo oddělení (Deptno), je tudíž zajištěno propojení zaměstnance s oddělením, ve kterém pracuje. [14]

Kód:

```
CREATE TABLE emp (
  Empno     NUMBER NOT NULL,
  Ename     CHAR(10),
  Job       CHAR(9),
  Mgr       NUMBER(4),
  Hiredate  DATE,
  Sal       NUMBER(7,2),
  Comm      NUMBER(7,2),
  Deptno    NUMBER(2) NOT NULL);


CREATE TABLE dept (
  Deptno    NUMBER(2) NOT NULL,
  Dname     CHAR(14),
  Loc       VARCHAR2(13),
  Mgr_no    NUMBER,
  Dept_type NUMBER);
```

Aby byla zajištěná referenční integrita, je nutné zajistit, aby nebylo možné smazat řádek či změnit číslo oddělení u záznamu v tabulce oddělení, pro které existuje odpovídající záznam v tabulce zaměstnanců. Dále pro tabulku potomka (zaměstnanců) je nutno při vkládání či aktualizaci cizího klíče (čísla oddělení) ověřit přítomnost odpovídající hodnoty v tabulce rodiče (oddělení). [14]

3.9.2 Konstrukce triggerů emp_dept_check a dept_restrict

Nejdříve bude uvedena konstrukce spouště pro zajištění definovaných omezení nad tabulkou zaměstnanců a poté až konstrukce spouště pro omezení nad tabulkou oddělení. [14]

Kód spouště pro tabulku zaměstnanců:

```
CREATE OR REPLACE TRIGGER emp_dept_check
  BEFORE INSERT OR UPDATE OF Deptno ON emp
  FOR EACH ROW WHEN (NEW.Deptno IS NOT NULL)
```

/ Spoušť sepne před vložením řádku, nebo před aktualizací hodnoty DEPTNO v tabulce zaměstnanců (emp), díky čemuž je následně ověřena nová hodnota cizího klíče oproti stavájícím hodnotám v tabulce oddělení. /

```
DECLARE
  Dummy               INTEGER;
  Invalid_department  EXCEPTION;
  Valid_department    EXCEPTION;
  PRAGMA EXCEPTION_INIT (Invalid_department, -4093);
  PRAGMA EXCEPTION_INIT (Valid_department, -4092);
```

/ Deklarace testovací proměnné, která bude využita pro kurzor sloužící ke kontrole existence rodičovské hodnoty. Pokud bude hodnota nalezena, bude rodičovský řádek uzamknut, aby nedošlo k jeho smazání jinou transakcí před dokončením této transakce. Dále jsou deklarované chybové hlášky, kterým jsou přiřazena čísla. /

```
 CURSOR Dummy_cursor (Dn NUMBER) IS
    SELECT Deptno FROM dept
    WHERE Deptno = Dn
    FOR UPDATE OF Deptno;
BEGIN
  OPEN Dummy_cursor (:NEW.Deptno);
  FETCH Dummy_cursor INTO Dummy;
```

/ Ověření přítomnosti rodičovské hodnoty. Pokud hodnota nebude nalezena, bude vrácena uživatelem definovaná chybová hláška o neprovedení operace v důsledku nesplnění referenční integrity. Pokud bude hodnota nalezena, tak bude kurzor před dokončením spouštěcí události uzavřen a na výstupu bude obdržena hláška o úspěšném vložení. /

```
 IF Dummy_cursor%NOTFOUND THEN
  RAISE Invalid_department;
 ELSE
  RAISE Valid_department;
 END IF;
 CLOSE Dummy_cursor;
EXCEPTION
 WHEN Invalid_department THEN
  CLOSE Dummy_cursor;
  Raise_application_error(-20000, 'Invalid Department'
   || ' Number' || TO_CHAR(:NEW.deptno));
 WHEN Valid_department THEN
  CLOSE Dummy_cursor;
 END;
/
```

Kód spouště pro tabulku oddělení:

```
CREATE OR REPLACE TRIGGER dept_restrict
  BEFORE DELETE OR UPDATE OF Deptno ON dept
  FOR EACH ROW
```

/ Spoušť sepne před smazáním řádku, nebo před aktualizací hodnoty primárního klíče DEPTNO tabulky oddělení (dept) a následně je ověřena přítomnost závislých hodnot v tabulce zaměstnanců. Pokud jsou takto závislé hodnoty nalezeny, tak se příkaz, který spustil tuto spoušť, neprovede. /

```
DECLARE
  Dummy                   INTEGER;
  employees_present       EXCEPTION;
  employees_not_present   EXCEPTION;
  PRAGMA EXCEPTION_INIT (employees_present, -4094);
  PRAGMA EXCEPTION_INIT (employees_not_present, -4095);
```

/ Deklarace testovací proměnné, která bude využita pro kurzor sloužící ke kontrole přítomnosti závislé hodnoty. Dále jsou deklarované chybové hlášky, kterým jsou přiřazena čísla. /

```
 CURSOR Dummy_cursor (Dn NUMBER) IS
    SELECT Deptno FROM emp WHERE Deptno = Dn;
BEGIN
  OPEN Dummy_cursor (:OLD.Deptno);
  FETCH Dummy_cursor INTO Dummy;
```

/ Pokud bude závislá hodnota nalezena, bude vrácena uživatelem definovaná chybová hláška o neprovedení operace v důsledku hrozícího narušení referenční integrity. Pokud nebude hodnota nalezena, tak bude kurzor před dokončením spouštěcí události uzavřen a operace mazání či aktualizace bude provedena. /

```
  IF Dummy_cursor%FOUND THEN
    RAISE employees_present;
  ELSE
    RAISE employees_not_present;
  END IF;
  CLOSE Dummy_cursor;

EXCEPTION
  WHEN employees_present THEN
    CLOSE Dummy_cursor;
    Raise_application_error(-20001, 'Employees Present in'
      || ' Department ' || TO_CHAR(:OLD.DEPTNO));
  WHEN employees_not_present THEN
    CLOSE Dummy_cursor;
END;
/
```

[14]

4 Praktická část

V této části práce jsou uvedeny tři příklady použití spouští. Každá z těchto spouští slouží k automatizaci procesů, které by v případě jejich nesestrojení musela vykonávat pověřená osoba, tudíž by bylo třeba vynaložit podstatně více času na realizaci takových procesů. Vždy je obsažena konstrukce výchozích tabulek a syntaxe konkrétních spouští. Těmito spouštěmi jsou:

- KLIENTI_INFORM
  Spoušť informující o stavu věrnostních bodů klientů cestovní kanceláře.
- MZDY_KONTROLA
  Spoušť zastupující komplexní integritní omezení typu CHECK.
- OBJEDNAVKY_INFO_INSERT
  Spoušť zaručující uložení hodnot do výchozích dvou tabulek při pokusu o uložení řádku do pohledu, jež z těchto dvou tabulek vypisuje související záznamy.

Pro tyto spouště budou konstruovány tabulky:

- KLIENTI
  Obsahuje číslo klienta (kli_id), pro jednoznačnou identifikaci; jména a příjmení (kli_jmeno; kli_prijmeni); Kontakt na klienta formou emailové adresy a telefonního čísla (kli_mail;kli_tel); Informaci o tom, zdali je klient smluvní (smluvni); počet věrnostních bodů (kli_body).
- ZAMESTNANCI
  Skládající se z čísla zaměstnance (zam_cislo), sloužícího k identifikaci každého zaměstnance; jména a příjmení (zam_jmeno; zam_prijmeni); práce, kterou vykonává (pozice); data nástupu (dat_nast); výše výplaty (mzda) a čísla oddělení, ve kterém pracuje (od_id) .
- MZDY
  S položkami mzdového stupně (mzda_stup); nejnižší možné mzdy (niz_mzda); nejvyšší možné mzdy (vys_mzda) pro danou klasifikaci pozice (poz_klas).
- ZAKAZNICI
  Složena z identifikačního čísla zákazníka (zak_id); jména a příjmení zákazníka (zak_jmeno; zak_prijmeni).

- OBJEDNAVKY
  Obsahující číslo objednávky (obj_id), přes kterou lze každou objednávku dohledat; datum objednávky (obj_dat); status objednávky (obj_stat) a identifikační číslo zákazníka (zak_id).

4.1 KLIENTI_INFORM

Tato spoušť má sloužit k vypsání informací o stavu a změně množství věrnostních bodů, které klienti fiktivní cestovní kanceláře mají na svých účtech. Díky tomu bude ušetřena práce, a tedy i téměř polovina času na provedení operací, které by bylo jinak nutno vykonat manuálně. Pověřená osoba již nebude muset provádět tři úkony, ale jen jeden. Pokaždé, když je tabulka KLIENTI ovlivněna DML příkazy INSERT, UPDATE nebo DELETE, tak se spustí spoušť, jež vypíše automaticky starý počet bodů před použitím DML operace, aktuální stav a rozdíl bodů před a po. Pokud je klient zástupcem smluvní společnosti, neaplikují se na něho věrnostní body, jelikož má např. fixní slevu či jiné další bonusy, proto se touto spouští nevypíše.

4.1.1 Vytvoření tabulky KLIENTI

V tabulce jsou uchovávány údaje o klientech. Pro identifikaci každého klienta slouží unikátní hodnota klientova ID, jež je PRIMARY KEY. Dále tabulka obsahuje jméno, příjmení. Kontaktní informace – email a telefonní číslo, na které se aplikuje omezení UNIQUE. Informace typu true/false – zdali je tento klient smluvní, a nakonec právě počet věrnostních bodů.

Kód:

```
CREATE TABLE klienti (
  kli_id            NUMBER       NOT NULL   PRIMARY KEY,
  kli_jmeno         CHAR(10)     NOT NULL,
  kli_prijmeni      CHAR(10)     NOT NULL,
  kli_mail          VARCHAR2(25) NOT NULL   UNIQUE,
  kli_tel           NUMBER(13)   NOT NULL   UNIQUE,
  smluvni           NUMBER       NOT NULL,
  kli_body          NUMBER);
```

4.1.2 Konstrukce triggeru KLIENTI_INFORM

Syntaxe kódu pro vytvoření spouště:

```
CREATE OR REPLACE TRIGGER KLIENTI_INFORM
  BEFORE DELETE OR INSERT OR UPDATE ON klienti
  FOR EACH ROW
  WHEN (NEW.smluvni <> '1')
DECLARE
  body_rozdil  NUMBER;
BEGIN
  body_rozdil  := :NEW.kli_body  - :OLD.kli_body;
  DBMS_OUTPUT.PUT(:NEW.kli_prijmeni || ': ');
  DBMS_OUTPUT.PUT('Stary pocet bodu = ' || :OLD.kli_body || ', ');
  DBMS_OUTPUT.PUT('Novy pocet bodu = ' || :NEW.kli_body || ', ');
  DBMS_OUTPUT.PUT_LINE('Rozdil: ' || body_rozdil);
END;
/
```

4.2 MZDY_KONTROLA

Pokaždé, když bude zaměstnanci změněna výše platu, případně pokud bude novému zaměstnanci vkládán nově plat, bude jeho výše z tabulky ZAMESTNANCI porovnána s maximální a minimální mzdou dané pracovní pozice v tabulce MZDY. Pokud bude mzda, kterou uživatel vloží do tabulky zaměstnanců, vybočovat ze stanovených limitů, poté spoušť vypíše chybovou hlášku upozorňující na tuto skutečnost a vložení hodnoty se neprovede. Díky této spoušti bude celý proces vložení přípustné mzdy na dané pozici zkrácen o automatizované porovnávání a díky tomu může být ušetřena až polovina doby tohoto procesu.

U této spouště je vycházeno z tabulek MZDY a ZAMESTNANCI, konstrukce je uvedena níže.

4.2.1 Vytvoření tabulky ZAMESTNANCI

Aby mohla spoušť MZDY_KONTROLA fungovat, je nutné nejdříve vytvořit náležitou tabulku. V tomto příkladu je vycházeno z tabulky ZAMESTNANCI sloužící k zaznamenávání údajů o zaměstnancích fiktivního podniku. Entitní integrita je zaručena použitím primárního klíče, jímž je číslo zaměstnance, tudíž každý zaměstnanec bude mít nezaměnitelnou hodnotu tohoto atributu. Žádná z položek navíc nemůže obsahovat nulovou hodnotu, čehož je dosaženo omezením NOT NULL.

Kód:

```
CREATE TABLE zamestnanci (
  Zam_cislo         NUMBER       NOT NULL   PRIMARY KEY,
  zam_jmeno         CHAR(10)     NOT NULL,
  zam_prijmeni      CHAR(10)     NOT NULL,
  pozice            NUMBER       NOT NULL,
  dat_nast          DATE         NOT NULL,
  mzda              NUMBER       NOT NULL,
  od_id             NUMBER       NOT NULL);
```

4.2.2 Vytvoření tabulky MZDY

Tabulka uchovávající hodnoty mzdového stupně, nejnižší a nejvyšší přípustné hodnoty pro danou pozici, jež je vyjádřena unikátním klasifikačním číslem. Unikátnost je zaručena použitím integritního omezení UNIQUE.

Kód:

```
CREATE TABLE mzdy (
 mzda_stup        NUMBER,
 niz_mzda         NUMBER        NOT NULL,
 vys_mzda         NUMBER        NOT NULL,
 poz_klas         NUMBER        UNIQUE);
```

4.2.3 Konstrukce triggeru MZDY_KONTROLA

Syntaxe kódu pro vytvoření spouště:

```
CREATE OR REPLACE TRIGGER mzdy_kontrola
  BEFORE INSERT OR UPDATE OF mzda, pozice ON zamestnanci
  FOR EACH ROW
DECLARE
  Min_mzda        NUMBER;
  Max_mzda        NUMBER;
  Mzda_mimo_meze  EXCEPTION;
  PRAGMA EXCEPTION_INIT (Mzda_mimo_meze, -4096);
```

```
/*    Definice proměnných pro minimální a maximální přípustné
hodnoty výplat. */

BEGIN
  SELECT niz_mzda, vys_mzda INTO Min_mzda, Max_mzda
  FROM mzdy
  WHERE poz_klas = :NEW.pozice;

/*   Vložení hodnot z tabulky MZDY do proměnných. */

  IF (:NEW.mzda < Min_mzda OR :NEW.mzda > Max_mzda) THEN
    RAISE Mzda_mimo_meze;
  END IF;
EXCEPTION
  WHEN Mzda_mimo_meze THEN

/*   Pokud zaměstnancova nová mzda přesáhne horní hranici, či
     naopak nedosáhne spodní hranice, zavolá Oracle chybovou
     hlášku Mzda_mimo_meze. Příkaz, který spoušť vyvolal se
     ruší a uživatelem zadaná operace se nevykoná. */

Raise_application_error (
       -20300,
       'Mzda '|| TO_CHAR(:NEW.mzda) ||' mimo meze pro '
       || 'pozici cislo ' ||:NEW.pozice
       ||' zamestnance ' || :NEW.zam_jmeno || :NEW.zam_prijmeni
       );
  WHEN NO_DATA_FOUND THEN
    Raise_application_error(-20322, 'Neplatna klasifikace');
END; /
```

4.3 OBJEDNAVKY_INFO_INSERT

Spouště typu INSTEAD OF lze využít především při manipulaci s pohledy, které nejsou dědičně aktualizovatelné. Cílem tohoto příkladu je zajistit provedení DML operace INSERT volané nad pohledem tak, aby hodnoty byly ukládány do výchozích tabulek.

Bude vytvořen pohled s hodnotami čerpanými ze dvou výchozích tabulek. Data jsou čerpána z tabulek ZAKAZNICI a OBJEDNAVKY, pohled z nich čerpající je OBJEDNAVKY_INFO.

Díky této spoušti bude očekáváno zkrácení a usnadnění celého procesu. Pověřená osoba již nebude muset řešit do které tabulky patří jaké údaje, místo toho je bude ukládat skrze pohled do patřičných tabulek DBMS automaticky. Lze očekávat zkrácení procesu opět o přibližně polovinu času.

4.3.1 Vytvoření tabulky ZAKAZNICI

Tabulka obsahuje seznam zákazníků, kteří jsou identifikováni pomocí jejich unikátního identifikačního čísla. Žádná z položek v této tabulce nemůže být prázdná, jelikož všechny sloupce obsahují integritní omezení NOT NULL. Propojení s tabulkou OBJEDNAVKY bude provedeno právě přes atribut zak_id.

Kód:

```
CREATE TABLE zakaznici (
  zak_id          NUMBER(5)        NOT NULL        PRIMARY KEY,
  zak_jmeno       CHAR(15),        NOT NULL
  zak_prijmeni    CHAR(15),        NOT NULL
);
```

4.3.2 Vytvoření tabulky OBJEDNAVKY

Tato tabulka sdružuje informace o jednotlivých objednávkách. Jednoznačná identifikace každé objednávky se provádí pomocí atributu obj_id. Propojení s tabulkou ZAKAZNICI je zajištěno cizím klíčem – zak_id, který je v tabulce ZAKAZNICI primárním klíčem. Opět žádný ze sloupců nemůže nabývat nulové hodnoty díky použití omezení NOT NULL.

Kód:

```
CREATE TABLE objednavky (
  obj_id      NUMBER(5)      NOT NULL PRIMARY KEY,
  obj_dat     DATE           NOT NULL,
  obj_stat    CHAR(10)       NOT NULL,
  zak_id      NUMBER         NOT NULL,
  FOREIGN KEY (zak_id) REFERENCES zakaznici(zak_id),
  CONSTRAINT CHK_obj_stat CHECK (obj_stat in
('prijata','zpracovani','expedovana','vyrizena','zrusena'))
);
```

4.3.3 Vytvoření pohledu OBJEDNAVKY_INFO

Díky tomuto pohledu je možné zobrazit data ze dvou tabulek v jednom objektu. Bohužel však následkem tohoto spojení primární klíč tabulky objednávky není unikátní, a proto ani tento pohled není dědičně aktualizovatelný. V takových případech je proto nutné vytvořit spoušť, která provede uživatelem požadovanou operaci nad výchozími tabulkami.

Kód:

```
CREATE OR REPLACE VIEW objednavky_info AS
  SELECT z.zak_id, z.zak_jmeno, z.zak_prijmeni,
      o.obj_id, o.obj_dat, o. obj_stat
  FROM zakaznici z, objednavky o
  WHERE z.zak_id = o.zak_id;
```

4.3.4 Konstrukce triggeru OBJEDNAVKY_INFO_INSERT

Syntaxe kódu pro vytvoření spouště:

```
CREATE OR REPLACE TRIGGER objednavky_info_insert
  INSTEAD OF INSERT ON objednavky_info
  DECLARE
   duplikatni_info EXCEPTION;
   PRAGMA EXCEPTION_INIT (duplikatni_info, -00001);
  BEGIN
   INSERT INTO zakaznici
    (zak_id, zak_jmeno, zak_prijmeni)
   VALUES (
   :new.zak_id,
   :new.zak_jmeno,
   :new.zak_prijmeni);
  INSERT INTO objednavky (obj_id, obj_dat, obj_stat, zak_id)
  VALUES (
   :new.obj_id,
   :new.obj_dat,
   :new.obj_stat,
   :new.zak_id);
  EXCEPTION
   WHEN duplikatni_info THEN
    RAISE_APPLICATION_ERROR (
     num=> -20107,
     msg=> 'Duplikatni zakaznikovo ID nebo cislo objednavky');
  END objednavky_info_insert;
/
```

4.4 Otestování navrhovaných řešení

4.4.1 KLIENT_INFORM

Nejprve byla vytvořena tabulka KLIENTI, viz. výstup níže:

```
SQL> CREATE TABLE klienti (
  2    kli_id       NUMBER NOT NULL PRIMARY KEY,
  3    kli_jmeno    CHAR(10) NOT NULL,
  4    kli_prijmeni CHAR(10) NOT NULL,
  5    kli_mail     VARCHAR2(25) NOT NULL UNIQUE,
  6    kli_tel      NUMBER(13) NOT NULL UNIQUE,
  7   smluvni       NUMBER       NOT  NULL  CHECK  (smluvni  in
      (1,0)),
  8    kli_body   NUMBER);
Table created.
```

Do tabulek byly následně vloženy testovací hodnoty pro následné ověření funkcionality spouště:

```
SQL> INSERT INTO klienti VALUES
(1,'Petra','Sumna','petra.sumna@em.cz','720431254',0,20000);
1 row created.

SQL> INSERT INTO klienti VALUES
(2,'Alena','Kralova','alena.kralova@em.cz','721005468',0,15000);
1 row created.

SQL> INSERT INTO klienti VALUES
(3,'Jan','Buben','jan_buben@em.cz','721489645',0,7500);
1 row created.

SQL> INSERT INTO klienti(kli_id, kli_jmeno, kli_prijmeni, kli_mail,
kli_tel, smluvni)
2  VALUES
(4,'Oldrich','Seminko','ACD_business@acd.cz','680489984',1);
1 row created.
```

Při pokusu o vložení hodnoty porušující integritní omezení UNIQUE, bychom měli obdržet chybovou hlášku o nemožnosti provedení požadované operace:

```
SQL> INSERT INTO klienti VALUES
(5,'Petra','Sumnakova','petra.sumna@em.cz','726831253',0,0);
    INSERT INTO klienti VALUES
(5,'Petra','Sumnakova','petra.sumna@em.cz','726831253',0,0)
    *
    ERROR at line 1:
    ORA-00001: unique constraint (SYSTEM.SYS_C008397) violated
```

Stejná chybová hláška by byla zobrazena v případě pokusu o vložení záznamu s již použitým telefonním číslem. Pokud bude vkládána hodnota porušující integritní omezení CHECK, zobrazí se chybová hláška viz. příklad níže:

```
SQL> INSERT INTO klienti VALUES
(5,'Petra','Sumnakova','petra.sumnakova@em.cz','726831253',0,0);
    INSERT INTO klienti VALUES
(5,'Petra','Sumnakova','petra.sumnakova@em.cz','726831253',0,0)
    *
    ERROR at line 1:
    ORA-02290: check constraint (SYSTEM.SYS_C008395) violated
```

Takto nyní vypadá tabulka klientů naplněná daty – viz. tabulka č. 3:

Tabulka č. 3 – Tabulka klientů naplněná daty;

```
SQL> SELECT * FROM klienti;
```

KLI_ID	KLI_JMENO	KLI_PRIJMENI	KLI_MAIL	KLI_TEL	SMLUVNI	KLI_BODY
1	Petra	Sumna	petra.sumna@em.cz	720431254	0	20000
2	Alena	Kralova	alena.kralova@em.cz	721005468	0	15000
3	Jan	Buben	jan_buben@em.cz	721489645	0	7500
4	Oldrich	Seminko	ACD_business@acd.cz	680489984	1	-

Zdroj: Autor

Pokud například cestovní kancelář vyhlásí vánoční marketingovou akci, v jejímž rámci navýší všem klientům s nenulovým počtem věrnostních bodů jejich zůstatek o 10%, jako vánoční dárek, bude použit příkaz UPDATE, po jehož provedení bude očekáván výpis změněných polí s informacemi o původním množství bodů novým množstvím a rozdílem mezi těmito hodnotami.

```
SQL> UPDATE klienti
  2  SET kli_body = kli_body * 1.1
  3  WHERE kli_body > 10000;
```

Výsledný výstup:

```
Sumna: Stary pocet bodu = 20000, Novy pocet bodu = 22000, Rozdil:
2000
Kralova: Stary pocet bodu = 15000, Novy pocet bodu = 16500, Rozdil:
1500
```

4.4.2 MZDY_KONTROLA

Nejdříve byly založeny tabulky MZDY a ZAMESTNANCI, tak jak je zobrazeno na výstupu:

```
SQL> CREATE TABLE mzdy (
  2    mzda_stup NUMBER,
  3    niz_mzda NUMBER NOT NULL,
  4    vys_mzda NUMBER NOT NULL,
  5    poz_klas CHAR(9) NOT NULL UNIQUE);
Table created.

SQL> CREATE TABLE zamestnanci (
  2    Zam_cislo     NUMBER NOT NULL PRIMARY KEY,
  3    zam_jmeno     CHAR(10) NOT NULL,
  4    zam_prijmeni  CHAR(10) NOT NULL,
  5    pozice        NUMBER(9) NOT NULL,
  6    dat_nast      DATE NOT NULL,
  7    mzda          NUMBER NOT NULL,
  8   od_id          NUMBER NOT NULL);
Table created.
```

Do tabulek byly následně vloženy testovací hodnoty pro následné ověření funkcionality spouště:

Tabulka mezd:

```
SQL> INSERT INTO mzdy VALUES(1,19000,26000,1);
1 row created.

SQL> INSERT INTO mzdy VALUES(1,23000,29000,2);
1 row created.
```

Tabulka zaměstnanců:

```
SQL> INSERT INTO zamestnanci
VALUES(1,'Petr','Svislik',1,TO_DATE('22/3/2016','DD/MM/YY'),25000,1)
;
1 row created.

SQL> INSERT INTO zamestnanci
VALUES(2,'Jonas','Kliste',2,TO_DATE('20/6/2017','DD/MM/YY'),29000,1)
;
1 row created.
```

Po naplnění daty byla konstruována samotná spoušť:

```
SQL> CREATE OR REPLACE TRIGGER mzdy_kontrola
  2  BEFORE INSERT OR UPDATE OF mzda, pozice ON zamestnanci
  3  FOR EACH ROW
  4  DECLARE
  5          Min_mzda NUMBER;
  6          Max_mzda NUMBER;
  7          Mzda_mimo_meze EXCEPTION;
  8          PRAGMA EXCEPTION_INIT (Mzda_mimo_meze, -4096);
  9
  10  BEGIN
  11  SELECT niz_mzda, vys_mzda INTO Min_mzda, Max_mzda
  12  FROM mzdy
  13  WHERE poz_klas = :NEW.pozice;
  14
  15    IF (:NEW.mzda < Min_mzda OR :NEW.mzda > Max_mzda) THEN
  16      RAISE Mzda_mimo_meze;
  17    END IF;
  18  EXCEPTION
  19    WHEN Mzda_mimo_meze THEN
  20  Raise_application_error (
```

```
  21  -20300,
  22  'Mzda '|| TO_CHAR(:NEW.mzda) ||' mimo meze pro '
  23  || 'pozici cislo ' ||:NEW.pozice
  24  ||' zamestnance ' || :NEW.zam_jmeno || :NEW.zam_prijmeni
  25  );
  26
  27    WHEN NO_DATA_FOUND THEN
  28   Raise_application_error(-20322, 'Neplatna klasifikace');
  29  END;
  30  /
  Trigger created.
```

Následuje vkládání řádků, u kterých se očekává vrácení chybové hlášky, za předpokladu cíleného fungování spouští.

```
SQL> INSERT INTO zamestnanci
VALUES(3,'Petr','Svislik',2,TO_DATE('2/8/2017','DD/MM/YY'),5000,1);
INSERT INTO zamestnanci
VALUES(3,'Petr','Svislik',2,TO_DATE('2/8/2017','DD/MM/YY'),5000,1)
                *
    ERROR at line 1:
    ORA-20300: Mzda 5000 mimo meze pro pozici cislo 2 zamestnance
    Petr Svislik
    ORA-06512: at "SYSTEM.MZDY_KONTROLA", line 17
    ORA-04088: error during execution of trigger
    'SYSTEM.MZDY_KONTROLA'
```

U tohoto vložení je hodnota vkládána mimo meze, proto databáze zobrazí příslušnou chybovou zprávu o nedodržení podmínek pro vložení hodnot.

```
SQL> INSERT INTO zamestnanci
VALUES(3,'Petr','Svislik',3,TO_DATE('2/8/2017','DD/MM/YY'),25000,1);
INSERT INTO zamestnanci
VALUES(3,'Petr','Svislik',3,TO_DATE('2/8/2017','DD/MM/YY'),25000,1)
                *
```

```
ERROR at line 1:
ORA-20322: Neplatna klasifikace
ORA-06512: at "SYSTEM.MZDY_KONTROLA", line 25
ORA-04088: error during execution of trigger
'SYSTEM.MZDY_KONTROLA'
```

Zde byla vkládána klasifikace pozice, jež neexistuje v tabulce mezd, nelze ji tedy kontrolovat a databáze vrací odpovídající chybovou hlášku

4.4.3 OBJEDNAVKY_INFO_INSERT

Nejprve byly založeny tabulky ZAKAZNICI a OBJEDNAVKY viz. výstup:

```
SQL> CREATE TABLE zakaznici (
  2    zak_id NUMBER(5) NOT NULL PRIMARY KEY,
  3    zak_jmeno CHAR(15) NOT NULL,
  4    zak_prijmeni CHAR(15) NOT NULL);
Table created.

SQL> CREATE TABLE objednavky (
  2    obj_id NUMBER(5) NOT NULL PRIMARY KEY,
  3    obj_dat DATE NOT NULL,
  4    obj_stat CHAR(10) NOT NULL,
  5    zak_id NUMBER,
  6    FOREIGN KEY (zak_id) REFERENCES zakaznici(zak_id),
  7    CONSTRAINT CHK_obj_stat CHECK (obj_stat in
('prijata','zpracovani','expedovana','vyrizena','zrusena'))
  8  );
Table created.
```

Do tabulek byly následně vloženy testovací hodnoty pro ověření funkcionality spouště:

Tabulka zákazníků:

```
SQL> Insert into zakaznici values(1,'Zbysek','Kulhavy');
1 row created.

SQL> Insert into zakaznici values(2,'Petra','Kratka');
1 row created.
```

Tabulka objednávek:

```
SQL> Insert into objednavky
values(1,TO_DATE('4/8/2017','DD/MM/YY'),'vyrizena',1);
1 row created.

SQL> Insert into objednavky
values(2,TO_DATE('4/11/2017','DD/MM/YY'),'zpracovani',2);
1 row created.
```

Nyní bude vytvořen pohled pro jednotné zobrazení záznamů z těchto tabulek:

```
SQL> CREATE OR REPLACE VIEW objednavky_info AS
  2     SELECT z.zak_id, z.zak_jmeno, z.zak_prijmeni,
  3            o.obj_id, o.obj_dat, o. obj_stat
  4     FROM zakaznici z, objednavky o
  5     WHERE z.zak_id = o.zak_id;

    View created.
```

Takto nyní vypadá tento pohled – viz tabulka č. 4:

Tabulka č. 4 – Pohled OBJEDNAVKY_INFO před vložením hodnot;

```
SQL> SELECT * FROM objednavky_info;
```

ZAK_ID	ZAK_JMENO	ZAK_PRIJMENI	OBJ_ID	OBJ_DAT	OBJ_STAT
1	Zbysek	Kulhavy	1	04-AUG-17	vyrizena
2	Petra	Kratka	2	04-NOV-17	zpracovani

Zdroj: Autor

Nyní bude zadán příkaz pro vložení hodnoty do tohoto pohledu, pro potvrzení výroku, že tato tabulka není dědičně upravitelná.

```
SQL> INSERT INTO objednavky_info
VALUES(3,'Ondrej','Plachy',3,TO_DATE('5/11/2017','DD/MM/YY'),'expedo
vana');
    INSERT INTO objednavky_info
VALUES(3,'Ondrej','Plachy',3,TO_DATE('5/11/2017','DD/MM/YY'),'expedo
vana')
    *
    ERROR at line 1:
    ORA-01779: cannot modify a column which maps to a non key-
    preserved table
```

Pro provedení vložení do tabulek zákazníků a objednávek na základě vkládání do pohledu je tedy skutečně nutné použít spoušť.

```
SQL> CREATE OR REPLACE TRIGGER objednavky_info_insert
  2     INSTEAD OF INSERT ON objednavky_info
  3     DECLARE
  4       duplikatni_info EXCEPTION;
  5       PRAGMA EXCEPTION_INIT (duplikatni_info, -00001);
  6     BEGIN
  7       INSERT INTO zakaznici
  8         (zak_id, zak_jmeno, zak_prijmeni)
  9       VALUES (
 10        :new.zak_id,
 11        :new.zak_jmeno,
 12        :new.zak_prijmeni);
 13     INSERT INTO objednavky (obj_id, obj_dat, obj_stat,
        zak_id)
 14     VALUES (
 15       :new.obj_id,
 16       :new.obj_dat,
 17       :new.obj_stat,
 18       :new.zak_id);
 19     EXCEPTION
 20       WHEN duplikatni_info THEN
 21         RAISE_APPLICATION_ERROR (
 22           num=> -20107,
 23           msg=> 'Duplikatni zakaznikovo ID nebo cislo
              objednavky');
 24     END objednavky_info_insert;
 25  /
   Trigger created.
```

Pro ověření správného fungování bude provedeno několik zkušebních vložení:

```
SQL> INSERT INTO objednavky_info
VALUES(3,'Ondrej','Plachy',3,TO_DATE('5/11/2017','DD/MM/YY'),'expedovana');
    1 row created.
```

Příkaz, který byl před vytvořením spouště neplatný, nyní vrací zprávu o vytvoření nového řádku, spoušť tedy funguje správně.

Následuje vložení záznamu, který bude nejdříve obsahovat již použité identifikační číslo a následně již použité číslo objednávky, opět by měly být zobrazeny chybové hlášky s odpovídajícím zněním.

Duplicitní ID zákazníka:

```
    SQL> INSERT INTO objednavky_info
VALUES(3,'Jana','Bezdekova',4,TO_DATE('6/11/2017','DD/MM/YY'),'zruse
na');
    INSERT INTO objednavky_info
VALUES(3,'Jana','Bezdekova',4,TO_DATE('6/11/2017','DD/MM/YY'),'zruse
na')
                *
    ERROR at line 1:
    ORA-20107: Duplikatni zakaznikovo ID nebo cislo objednavky
    ORA-06512: at "SYSTEM.OBJEDNAVKY_INFO_INSERT", line 19
    ORA-04088: error during execution of trigger
'SYSTEM.OBJEDNAVKY_INFO_INSERT'
```

Duplicitní ID objednávky:

```
    SQL> INSERT INTO objednavky_info
VALUES(4,'Jana','Bezdekova',3,TO_DATE('6/11/2017','DD/MM/YY'),'zruse
na');
    INSERT INTO objednavky_info
VALUES(4,'Jana','Bezdekova',3,TO_DATE('6/11/2017','DD/MM/YY'),'zruse
na')
                *
    ERROR at line 1:
    ORA-20107: Duplikatni zakaznikovo ID nebo cislo objednavky
    ORA-06512: at "SYSTEM.OBJEDNAVKY_INFO_INSERT", line 19
    ORA-04088: error during execution of trigger
'SYSTEM.OBJEDNAVKY_INFO_INSERT'
```

Korektní vložení stejného řádku:

```
    SQL> INSERT INTO objednavky_info
VALUES(4,'Jana','Bezdekova',4,TO_DATE('6/11/2017','DD/MM/YY'),'zruse
na');
    1 row created.
```

Zobrazení pohledu a tabulek pro závěrečnou kontrolu – viz tabulky č. 5; 6; 7:

Tabulka č. 5 – Pohled OBJEDNAVKY_INFO

```
SQL> SELECT * FROM objednavky_info;
```

ZAK_ID	ZAK_JMENO	ZAK_PRIJMENI	OBJ_ID	OBJ_DAT	OBJ_STAT
1	Zbysek	Kulhavy	1	04-AUG-17	vyrizena
2	Petra	Kratka	2	04-NOV-17	zpracovani
3	Ondrej	Plachy	3	05-NOV-17	expedovana
4	Jana	Bezdekova	4	06-NOV-17	zrusena

Zdroj: Autor

Tabulka č. 6 – Tabulka OBJEDNAVKY

```
SQL> SELECT * FROM objednavky;
```

OBJ_ID	OBJ_DAT	OBJ_STAT	ZAK_ID
1	04-AUG-17	vyrizena	1
2	04-NOV-17	zpracovani	2
3	05-NOV-17	expedovana	3
4	06-NOV-17	zrusena	4

Zdroj: Autor

Tabulka č. 7 – Tabulka ZAKAZNICI:

```
SQL> SELECT * FROM zakaznici;
```

ZAK_ID	ZAK_JMENO	ZAK_PRIJMENI
1	Zbysek	Kulhavy
2	Petra	Kratka
3	Ondrej	Plachy
4	Jana	Bezdekova

Zdroj: Autor

5 Diskuse

Přirozeným důsledkem technického rozvoje a rozvoje moderní společnosti obecně je téměř až nepředstavitelný nárůst shromažďovaných informací. Výsledkem tohoto obrovského růstu zpracovávaných dat je jejich podstatně komplikovanější správa a administrace. Úkony, které byly dříve lehko proveditelné a časově nenáročné, se stávají v současné době v manuální podobě prakticky nerealizovatelné. A to ať už se jedná o ukládání záznamů, dodržování integritních omezení či například výpis pozměněných dat. Z těchto důvodů je nutné automatizovat, a to především rutinní činnosti a procesy, kterým člověk svojí přímou účastí nedává žádnou přidanou hodnotu.

Automatizace procesů je velmi rozsáhlá a komplikovaná problematika, kterou nelze komplexně obsáhnout v rámci jedné bakalářské práce. Tato práce má sloužit jako úvod do problematiky a seznámení s jejími hlavními aspekty, uvedení vhodných oblastí užití a způsobu realizace databázových spouští. Triggery jsou jedním z možných prostředků, díky kterým lze procesy automatizovat.

Přínos triggery představují z hlediska zjednodušení a zrychlení procesů, jelikož jejich konstrukce probíhá, na rozdíl od manuální realizace každého úkonu, jen jednou a poté je již SŘBD automaticky spouští při každém výskytu deklarované spouštěcí události. Pro jejich konstrukci je ve většině případů potřeba vynaložit větší množství času, než který by byl jednorázově čerpán pro vykonání neautomatizovaných procesů. Pokud se však sečte čas, který by na vykonání takovéhoto periodického úkonu byl třeba vykonat pro tisíce záznamů, poté automatizace představuje nespornou úsporu obrovského objemu času. Jednorázové nastavení triggerů je tudíž moderní a efektivní postup, který zadavatelům ušetří mzdové náklady a umožní využití kvalifikované pracovní síly na náročnější úkoly – ať již při samotném zpracování dat, tak při automatizaci nezbytných kontrolních procesů.

Úspory nákladů rostou přímou úměrou s každým vyvoláním triggeru, protože spoušť se již znovu nedefinuje, ale pouze bude opakovaně proveden její obsah. Pro názornost je v praktické části uveden příklad, ve kterém není nutné vkládat data do dvou výchozích tabulek, ze kterých je složen pohled, ale díky nadefinované spoušti lze ukládat data rovnou do pohledu, ze kterého jsou informace následně automaticky uloženy do těchto výchozích tabulek. Již díky této jednoduché spoušti lze ušetřit téměř polovinu času, který by byl jinak na stejnou operaci vynaložen při výhradně manuálním zpracování.

Dalším příkladem je spoušť vypisující požadované informace o rozdílu nově vložené hodnoty a původní hodnoty. Díky této spoušti uživatel již nemusí manuálně dohledávat původní hodnotu a následně počítat, jaký je rozdíl mezi původní a novou hodnotou, ale místo toho jsou mu tyto údaje přímo vypsány spouští. Místo provedení tří manuálních operací je díky užití triggeru nutné vykonat pouze jednu.

Poslední ukázka slouží pro kontrolu maximální a minimální výše hodnoty atributu, které jsou dané kategorizací každé entity. Pokud budou při vkládání či aktualizaci těchto hodnot porušeny meze, bude uživatel o této skutečnosti informován a jím zadávaná operace se neprovede. Místo toho, aby musel uživatel pokaždé dohledávat maximální a minimální meze, je o jejich porušení informován při neplatném vložení, touto spouští tak bude ušetřen čas, který by musel být vynaložen pro nalezení a opravu chybně vložených hodnot, eliminuje tedy chyby způsobené lidským faktorem.

Spouště sestavené v praktické části představují ukázky způsobu automatizace specifikovaných podnikových procesů bez narušení integritních pravidel, nejedná se o konečné a ideální řešení, pouze o způsob, jakým lze automatizaci realizovat. Integritní omezení byla pokaždé v rámci ověření navržených řešení vzorově testována. Jako první byla představena spoušť, jež zaručuje výpis těch záznamů, u kterých došlo použitím DML příkazu INSERT, UPDATE nebo DELETE k úpravě a navíc zobrazí, o jakou změnu se jedná a jaký je rozdíl mezi původní a novou hodnotou. V pořadí druhá spoušť zajišťuje důkladnou kontrolu výše specifikovaných hodnot vůči nadefinovaným hranicím, které odpovídají kategorizaci daného záznamu. V tomto případě to je nejvyšší a nejnižší přípustná mzda pro danou pracovní pozici, na kterou zaměstnanec nově nastupuje. Poslední představená spoušť umožňuje manuálně nerealizovatelné uložení hodnot po použití DML příkazu INSERT do výchozích dvou tabulek skrze jeden pohled, jež není dědičně aktualizovatelný. Ověření obsahuje úspěšné testování integritních omezení PRIMARY KEY a správné uložení nových hodnot do obou tabulek.

Navrhovaná řešení obsahují požadované funkcionality a lze je úspěšně využít k automatizaci podnikových procesů v typově podobných případech, bylo by však nutné jejich funkcionality podstatně rozšířit a doplnit, ideálně v rámci určitého automatizačního balíčku. Jejich zásadním přínosem je zrychlení a zjednodušení práce. Jednodušší práce znamená snížení operačního rizika, zkrácení času potřebného k vykonání různých úkonů přináší vyšší efektivitu, která znamená ve svém důsledku snížení nákladů podniku.

6 Závěr

Tématem této bakalářské práce byla problematika automatizace procesů prostřednictvím databázových triggerů v kontextu informačního zabezpečení podniků.

Aktuálně lze na základě mapování momentálního stavu říci, že pro správu a administraci velkého množství dat jsou relační databáze nejužívanějším řešením. Množství takto ukládaných dat se však neustále navyšuje a jejich kontrola a údržba se stává velice komplikovanou. Právě díky tomuto faktu se stává využití automatizace naprostou nutností pro všechny podnikatelské subjekty, bez ohledu na jejich velikost, čemuž odpovídá i aktuální doporučení firmy Oracle k užití spouští.

Hlavním cílem práce bylo přiblížení, navržení a realizace automatizace procesů. Pro tento záměr byla navržena v praktické části tři možná řešení, na kterých jsou demonstrovány možné postupy vytváření spouští a užití automatizace v DBMS firmy Oracle. Dalšími dílčími cíli bylo vymezení relevantnosti, stanovení aktuálního stavu a úvod do řešené problematiky. Pro tento účel byla vytvořena literární rešerše. Pro splnění cílů byly použity postupy založené na znalostech získaných ze studia, analýzy odborné literatury, vlastní odborné praxe a dalších zdrojů.

Řešení navrhovaná v praktické části práce jsou zaměřena na usnadnění a zkrácení periodických podnikových procesů, což v konečném důsledku znamená ušetření finančních zdrojů podniků. V této souvislosti je však nutno podotknout, že využití spouští je tím efektivnější, čím větší počet záznamů se nachází v dané databázi.

Bylo užito především metod a technik relačně databázového řešení, které se věnují zajištění automatizace procesů v databázových systémech firmy Oracle, zejména bylo užito programovacího jazyka PL/SQL, integritních omezení, DCL a DML příkazů a triggerů. Rešerše se věnuje hlavně principům relačně databázových systémů a problematice datové integrity v kontextu se spouštěmi.

Automatizace procesů ušetří podnikům jak velký objem financí, tak i velké množství času, který by jinak musel být pro správu dat vynaložen. Spouště navíc omezují zásah lidského faktoru do databáze, snižují tedy riziko chyby, která může nastat při každé manuální modifikaci dat a způsobit jejich nekonzistentnost. Užitečnost databázových triggerů spočívá i

v jejich širokém užití, lze pomocí nich například automaticky kontrolovat zadávaná data a zamezit jejich vložení do databáze, automaticky generovat odvozené hodnoty hodnot sloupců, zamezit invalidním transakcím, implementovat business pravidla, logovat události v databázi nebo kupříkladu modifikovat data v tabulce, pokud DML příkaz používá pohled. Navíc lze také zajistit jinak neaplikovatelnou referenční integritu přes různé uzly databáze, proto jsou v tomto ohledu nenahraditelné.

Podmínkou úspěšné implementace spouští za účelem automatizace procesů je kvalitní analýza a návrh. Při tak zásadním objemu dat nelze vytvořit spoušť, která by narušovala integritu databáze, či přinášela jiná bezpečnostní rizika. Proto musí být věnován dostatek času a pozornosti vytvoření dostatečně průhledné, robustní, a zejména udržovatelné formy spouští. Často platí, že nejlepší spouště jsou ty nejjednodušší.

Tuto myšlenku lze rozvést především směrem k případnému hledání chyb, či dalšímu rozšiřování a optimalizaci. Ve většině podniků je v současné době běžnou praxí nejdříve nové funkcionality nasazovat do testovacího prostředí dané aplikace, jež je kopií prostředí produkčního, se skrytými identifikačními údaji předmětu podnikání daného subjektu. V testovacím prostředí jsou pak nové funkcionality na relevantních datech důkladně testovány a optimalizovány. Do produkčního prostředí jsou nasazovány až po průkazném ověření funkčnosti.

I přes všechny zmiňované výhody lze stále narazit na podniky, jež se automatizaci vyhýbají. Především se jedná o ty subjekty, které svým zaměřením nespadají do IT sféry. U takových podniků je většina opakovatelných a rutinních úkonů spouštěna ručně, což přináší velké riziko lidské chyby a narušení konzistence dat. Důsledkem této pasivity je zbytečné zvýšení nákladů na zpracování dat a neefektivní využití lidských zdrojů.

S ohledem na tato fakta bych v současné době označil použití spouští v relačních databázích za velice užitečné, pro efektivní řízení operačního rizika a nákladů téměř nezbytné. Vzhledem k předpokladu pokračujícího růstu množství ukládaných dat a zdokonalování databázových technologií budou dle mého názoru postupně muset všechny podnikatelské subjekty a organizace přistoupit na toto řešení.

7 Seznam použitých zdrojů

[1] BORONCZYK, T.: MySQL okamžitě. 1.vydání. Computer press, 2016. 144 stran. ISBN: 9788025147375.

[2] KROENKE, D.: Databáze. 1.vydání. Computer press, 2015. 496 stran. ISBN: 978-80-251-4352-0.

[3] LACKO, L.: 1001 tipů a triků pro SQL. 1.vydání. Computer press, 2011. 416 stran. ISBN: 978-80-251-3010-0.

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

[5] Příručka k relacím mezi tabulkami. Microsoft [online] [cit 2017-20-10]. Dostupné z: https://support.office.com/cs-cz/article/P%C5%99%C3%ADru%C4%8Dka-k-relac%C3%ADm-mezi-tabulkami-30446197-4fbe-457b-b992-2f6fb812b58f#__toc269752114

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

[7] Databáze [online]. [cit 2017-19-07]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/databaze/.

[8] KOPAL, O.: Historie a trendy ve vývoji databází [online]. [cit 2017-19-07]. Dostupné z: http://www.web-integration.info/cs/blog/historie-a-trendy-ve-vyvoji-databazi/.

[9] SKŘIVAN, J.: Databáze a jazyk SQL [online]. [cit 2017-22-07]. Dostupné z: https://www.interval.cz/clanky/databaze-a-jazyk-sql/.

[10] VALENTA, M.: Databázové modely. ČVUT [prezentace]. [cit 2017-22-07]. Dostupné z: https://users.fit.cvut.cz/valenta/doku/lib/exe/fetch.php/bivs/dbs_02_databazove_modely.pdf.

[11] SKŘÍVAN, J.: Jak na triggery [online]. [cit 2017-27-07]. Dostupné z: https://www.interval.cz/clanky/sql-jak-na-triggery/.

[12] POKORNÝ, J; HALAŠKA, I.: Databázové systémy. ČVUT [online] [cit 2017-02-08]. Dostupné z: webzam.fbmi.cvut.cz/szabozol/ISZ/Kauler/506.doc

[13] Database Concepts. ORACLE Help Center [online] [cit 2017-30-10]. Dostupné z: https://docs.oracle.com/cd/B19306_01/server.102/b14220/triggers.htm

[14] Database PL/SQL Language Reference. ORACLE Help Center [online] [cit 2017-30-10]. Dostupné z: https://docs.oracle.com/database/122/LNPLS/plsql-triggers.htm#LNPLS020

[15] CREATE TRIGGER Statement. ORACLE Help Center [online] [cit 2017-08-11]. Dostupné z: https://docs.oracle.com/cloud/latest/db112/LNPLS/

create_trigger.htm#LNPLS01374
[16] ALTER TRIGGER Statement. ORACLE Help Center [online] [cit 2017-08-11]. Dostupné z: https://docs.oracle.com/database/122/LNPLS/ALTER-TRIGGER-statement.htm#LNPLS99996
[17] DROP TRIGGER Statement. ORACLE Help Center [online] [cit 2017-08-11]. Dostupné z: https://docs.oracle.com/database/122/LNPLS/DROP-TRIGGER-statement.htm#LNPLS99990
[18] MySQL Triggers [online] [cit 2017-18-11]. Dostupné z: https://www.w3resource.com/mysql/mysql-triggers.php#BFT
[19] Database Error Messages. ORACLE [online] [cit 2017-25-11]. Dostupné z: https://docs.oracle.com/database/122/ERRMG/toc.htm
[20] PL/SQL Error Handling. ORACLE [online] [cit 2017-25-11]. Dostupné z: https://docs.oracle.com/database/122/LNPLS/plsql-error-handling.htm#LNPLS99872
[21] Predefined Exceptions. ORACLE [online] [cit 2017-25-11]. Dostupné z: https://docs.oracle.com/database/122/LNPLS/plsql-error-handling.htm#LNPLS00703
[22] EXCEPTION_INIT Pragma. ORACLE [online] [cit 2017-25-11]. Dostupné z: https://docs.oracle.com/database/122/LNPLS/EXCEPTION_INIT-pragma.htm#LNPLS01315