ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia a informatiky

AUTOREFERÁT
DIZERTAČNEJ PRÁCE

Žilina, 2024 **Ing. Martina Hrínová Durneková**

Žilinská univerzita v Žiline
Fakulta riadenia a informatiky

Ing. Martina Hrínová Durneková

Autoreferát dizertačnej práce

Analytické spracovanie dát

na získanie akademického titulu **„philosophiae doctor“ (v skratke PhD.)**
v študijnom programe doktorandského štúdia
aplikovaná informatika

v študijnom odbore:
informatika

Žilina, máj 2024

Dizertačná práca bola vypracovaná v dennej forme doktorandského štúdia **na Katedre informatiky, Fakulte riadenia a informatiky Žilinskej univerzity v Žiline**

Predkladateľ: **Ing. Martina Hrínová Durneková**
Katedra informatiky
Fakulta riadenia a informatiky
Žilinská univerzita v Žiline

Školiteľ: **doc. Ing. Michal Kvet, PhD.**
Katedra informatiky
Fakulta riadenia a informatiky
Žilinská univerzita v Žiline

Oponent: **prof. RNDr. Valerie Novitzká, PhD.**
Katedra počítačov a informatiky
Fakulta elektrotechniky a informatiky
Technická univerzita v Košiciach

Oponent: **doc. Ing. Jarmila Škrinárová, PhD.**
Katedra informatiky
Fakulta prírodných vied
Univerzita Mateja Bela v Banskej Bystrici

Autoreferát bol rozoslaný dňa: 27. júna 2024

Obhajoba dizertačnej práce sa koná dňa 22.08.2024 o 10:30 h. pred komisiou pre obhajobu dizertačnej práce schválenou pracovnou skupinou odborovej komisie v študijnom odbore **informatika v študijnom programe aplikovaná informatika**, vymenovanou dekanom Fakulty riadenia a informatiky Žilinskej univerzity v Žiline.

prof. Ing. Ľudmila Jánošíková, PhD.
predsedníčka pracovnej skupiny odborovej komisie
v študijnom odbore **informatika**
v študijnom programe **aplikovaná informatika**

Fakulta riadenia a informatiky
Žilinská univerzita
Univerzitná 8215/1
010 26 Žilina

Úvod

Dáta sú všade okolo nás. Ich množstvo každý deň enormne narastá. No nie vždy je dostatočne využitý ich potenciál. Nie vždy sa dokážu správnym spôsobom pochopiť a interpretovať. Preto sa v súčasnosti kladie čoraz väčší dôraz na dátovú analýzu. Vďaka nej sme schopní dostupné dáta spracovať, vykonať nad nimi rôzne analýzy, štatistiky, reporty, na základe ktorých vieme získať hodnotné informácie, ktoré sú pre spoločnosť, ale i pre mnohé firmy tak dôležité. Firmy a organizácie na základe získaných poznatkov vedia robiť strategické rozhodnutia, ktoré vedú k optimalizácii procesov, zvýšeniu efektivity a celkovej prosperite. Okrem toho vedia pružne reagovať na rýchlo sa rozvíjajúci priemysel a mnohé ďalšie požiadavky [2]. Pre zabezpečenie bezproblémovej používateľskej skúsenosti počas analýzy je rýchle vyhľadávanie údajov a spracovanie príkazov nevyhnutné. To znamená, že je potrebné minimalizovať celkovú dobu čakania na výsledky [15].

1 Analýza súčasného stavu

Dátová analýza je proces čistenia, transformácie, modelovania a interpretácie údajov. Jej cieľom je získať užitočné informácie a poznatky, na základe ktorých je možné vykonať kvalitnejšie rozhodnutia [15]. Vďaka dátovej analýze je možné lepšie pochopiť predchádzajúce udalosti, zamerať sa na lepšie rozhodnutia súčasnosti a predpokladať smerovanie do budúcnosti. Všetky sféry podnikania sa dnes spoliehajú na dáta a ich analýzu pri prijímaní dôležitých rozhodnutí [2], [12].

Analytické spracovanie dát sa skladá z niekoľkých fáz: *Požiadavky na dáta, Zber údajov, Spracovanie údajov, Analýza dát* a *Interpretácia*.

1.1 Dátový sklad

Dátový sklad je centrálne úložisko integrovaných dát, ktoré prichádzajú z jedného alebo viacerých zdrojov [15]. Považuje sa za hlavný komponent pre systém Business Intelligence, ktorý je určený na analýzu dát. Uchováva historické údaje na jednom mieste po dlhé časové obdobie [4]. Tieto dáta sa následne využívajú na analýzu a tvorbu reportov, ktoré slúžia na zlepšenie kvality strategických rozhodnutí v organizáciách [2], [13]. Dátové sklady sú navrhnuté hlavne na rýchle vykonávanie dotazov, preto často obsahujú nenormalizovanú schému, ktorá vedie k optimalizácii výkonu a rýchlejšej odozve [22]. Na zabezpečenie vysokej priepustnosti údajov a rýchleho vykonávania dotazov je potrebné dátový sklad vhodne navrhnúť. Dátové sklady využívajú multidimenzionálny prístup k dátam [6]. Taktiež poskytujú OLAP nástroje, ktoré pomáhajú efektívne a interaktívne analyzovať údaje [4], [12].

Dátový sklad využíva dva typy tabuliek. ***Tabuľka faktov*** obsahuje merateľné atribúty, zatiaľ čo ***tabuľka dimenzií*** obsahuje atribúty, ktoré popisujú prvky organizácie. Dimenzie reprezentujú kategórie, ktoré sa používajú na organizáciu dát, ukladanie a načítanie hodnôt [7]. Existuje niekoľko typov schém dátových skladov, ktoré vytvárajú a logicky popisujú určitú databázu. Medzi tieto schémy patrí [6] – *Star schéma, Snowflake schéma* a *Fact constellation schéma*.

Dátový sklad má niekoľko charakteristík, ktoré popísal vedec Bill Inmon, ktorý je považovaný za otca dátových skladov [7], [8]. Medzi tieto charakteristiky patrí:

- **Orientácia na predmet -** dátové sklady tvoria štruktúru vhodnú na analyzovanie dát, ktoré sú zamerané na konkrétny predmet. Predmet môže reprezentovať predaj, distribúciu a iné.
- **Integrácia -** Dátové sklady obsahujú veľké množstvo dát z rôznych zdrojov, preto je potrebné zaistiť ich konzistenciu a eliminovať akúkoľvek nekonzistenciu medzi

dátami. Pre zabezpečenie konzistencie je potrebné dodržať určitý spôsob merania premenných, konvenciu pomenovávania a niekoľko ďalších pravidiel.

- **Časová zložka** - Dáta v dátovom sklade reprezentujú historické dáta zozbierané v rámci dlhodobého časového horizontu, preto je pri analýze potrebné zamerať sa na zmeny v čase.
- **Stabilita** - Dáta v dátovom sklade sú nemenné, takže nie je možné ich akokoľvek meniť, modifikovať alebo mazať ich hodnoty. Dáta sú určené výlučne na čítanie.

Dátové sklady majú množstvo odlišností od klasických transakčných databáz. Nasledujúca *Tabuľka 1* obsahuje porovnanie dátových skladov a transakčných databáz.

Tabuľka 1: Porovnanie dátových skladov a transakčných databáz

Dátový sklad	Transakčná databáza
Udržiava historické dáta	Udržiava aktuálne dáta
Stabilita, dáta je možné len pridávať	Dáta je možné modifikovať
Orientácia na predmet	Orientácia na procesy
Zameriava sa na výstup informácií	Zameriava sa na dáta vstupujúce do systému
Uchováva veľké množstvo dát	Uchováva menšie množstvo dát
Navrhnuté pre OLAP	Vytvorené pre OLTP
Založené na troch typoch schém	Založené na ERA modeli

1.2 Optimalizácia výkonnosti v databázových systémoch

Výkonnosť databázy v rámci informačných systémov je v dnešnej dobe kľúčový faktor pre organizácie, ktoré využívajú tieto systémy na dennej báze. Preto cieľom optimalizácie v databázových systémoch je zabezpečiť, aby vykonávanie dotazov bolo čo najefektívnejšie. Výkon databázy ovplyvňuje niekoľko faktorov ako napríklad už samotný návrh databázy, jej štruktúra, množstvo dát a ďalšie [11], [14]. Aby sme vedeli zlepšiť a zrýchliť aplikácie využívajúce obrovské množstvo dát, je potrebné pochopiť, čo sa deje na pozadí databázového systému, v akých krokoch sa vykonávajú dotazy, ako prebieha ich spracovanie, a aké sú možnosti optimalizácie.

Databázové systémy využívajú hlavne dva typy optimalizácie dotazov. Prvou z nich je ***Optimalizácia založená na pravidlách*** (Rule-Based Optimization, RBO) a druhou je ***Cenovo orientovaná optimalizácia*** (Cost-Based Optimization, CBO). Oba spôsoby optimalizácie sú navrhnuté tak, aby určili najefektívnejší spôsob vykonania SQL dotazov. Hlavný rozdiel medzi nimi je spôsob prístupu, ktorý používajú pri vyhodnotení a výbere najvhodnejšieho plánu vykonania [1], [14].

Nástroj, ktorý slúži na určenie najefektívnejšieho spôsobu vykonania dotazu sa nazýva ***optimalizátor***. Je to nástroj, ktorý obsahuje rôzne prístupové scenáre, z ktorých sa pomocou heuristiky vyberá najlepší, prípadne sub-optimálny plán vykonania. Medzi dôležité údaje pre rozhodnutie najvhodnejšieho plánu vykonania patria popisné údaje, štatistiky databázy, porovnanie odhadovaných nákladov na spracovanie blokov a náklady na sekvenčné spracovanie blokov. Takýto plán vykonania sa dočasne uloží do Shared pool pamäte inštancie, takže sa pri opätovnom použití daného príkazu môže využiť tento plán vykonania. ***Plán vykonania*** obsahuje jednotlivé kroky na vykonanie SQL príkazu. Tieto kroky sú vyjadrené prostredníctvom databázových operátorov, ktoré produkujú riadky. O poradí týchto operátorov a ich implementácii rozhoduje optimalizátor. Plán vykonania využíva stromovú reprezentáciu, zatiaľ čo na displeji sa zobrazuje jeho tabuľková reprezentácia [18].

Nasledujúci príklad (*def. príkazu 1*) znázorňuje jednoduchý príkaz SELECT, ktorý zobrazuje identifikačné číslo nehody, dátum nehody identifikačné číslo poveternostných podmienok počas nehody a popis. Spája dve tabuľky, tabuľku *nehody* a tabuľku *poveternostné podmienky*, pričom vyberáme len tie záznamy, v ktorých bolo identifikačné číslo poveternostných podmienok 2 (hmla).

```
SELECT id_nehody, p2a_datum,id_poveternostnych_podmienok, popis
 FROM nehody
  JOIN p18_poveternostne_podmienky
   USING(id_poveternostnych_podmienok)
 WHERE id_poveternostnych_podmienok = 2;
```

Definícia príkazu 1: Jednoduchý príkaz SELECT spájajúci dve tabuľky

Tabuľková reprezentácia plánu vykonania vyššie uvedeného príkazu SELECT je zobrazená na nasledujúcom *Obrázku 1*.

```
-------------------------------------------------------------------------------------------------------
| Id  |  Operation          | Name                        |  Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 |  SELECT STATEMENT   |                             |  87448 |  4099K|  7744   (1)| 00:00:01 |
|*  1 |   HASH JOIN         |                             |  87448 |  4099K|  7744   (1)| 00:00:01 |
|*  2 |    TABLE ACCESS FULL| P18_POVETERNOSTNE_PODMIENKY |      1 |    25 |     3   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| NEHODY                      |    699K|    15M|  7737   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------------
```

Obrázok 1: Plán vykonania - tabuľková reprezentácia

Výsledný plán vykonávania, ktorý vyberie optimalizátor, je len jeden z viacerých alternatívnych plánov vykonávania pre daný príkaz SELECT. Optimalizátor vyberie taký plán vykonávania, ktorý má najnižšie náklady na vykonanie, pričom náklady predstavujú internú hodnotu optimalizátora pre odhadované využitie zdrojov pre daný plán. Čím sú náklady na príkaz SELECT nižšie, tým bude plán vykonávania efektívnejší. Preto hovoríme, že optimalizátor je nákladovo orientovaný. Celkové náklady a náklady každej operácie sú uvedené v pláne vykonávania [18].

Aby sa optimalizátor vedel správne rozhodovať a vybrať najefektívnejší plán vykonania je potrebné evidovať aktuálne ***štatistiky***, vďaka ktorým je možné odhadnúť náklady plánov vykonania pre definovaný SQL príkaz [14]. Tieto štatistiky predstavujú súbor údajov, ktoré popisujú objekty v databáze a databázu ako takú [19]. Generovanie štatistík, by sa malo vykonávať pre každý objekt, ktorý sa nachádza v SQL dotaze a po každej rozsiahlejšej zmene by sa mali štatistiky taktiež pregenerovať, aby boli vždy aktuálne. Databázový systém rozoznáva 3 druhy štatistík – *Systémové, Tabuľkové* a *Stĺpcové štatistiky*.

1.2.1 Indexy

Na optimalizovanie výkonu vyhľadávania údajov v databázových tabuľkách je možné využiť indexy. Index je databázový objekt, vďaka ktorému je možné zrýchliť vykonávanie dotazov. Výhodou indexov je, že na lokalizovanie príslušných dát prostredníctvom indexov nie je potrebné prehľadávanie databázovej tabuľky riadok po riadku, avšak je potrebné vedieť, že pri operáciách *INSERT, DELETE* a *UPDATE* spôsobuje dodatočnú réžiu, pretože okrem upravenia dátových blokov je potrebné upraviť i samotný index [4].

Index môže byť tvorený jedným alebo viacerými stĺpcami tabuľky. Poradie stĺpcov má veľký vplyv na celkové náklady vykonávaného SQL dotazu. Okrem poradia stĺpcov je pri vytváraní indexu dôležité myslieť na podmienky definované v klauzule *WHERE,* podmienky spájania tabuliek a tiež atribúty, ktoré sú získané v klauzule *SELECT*. Index pozostáva z dvoch častí. Obsahuje kľúč a referenciu na dáta v databázovej tabuľke. V rámci

indexov je možné zistiť selektivitu, ktorá je definovaná pomerom medzi počtom odlišných hodnôt k celkovému počtu riadkov. Jej rozsah je možné určiť na intervale <0, 1> [4].

Existuje niekoľko typov indexov. Štandardne sa vytvára index typu *B+strom*. Okrem neho je možné vytvoriť aj iný typ indexovej štruktúry, ako napríklad *Hash index, Bitmap index, Funkcionálny index, Cluster index* a ďalšie. V prípade veľkých tabuliek je výhodné využiť rozdelenie tabuliek alebo indexov na partície, ktoré sa vytvárajú na základe zoznamu hodnôt intervalového rozdelenia alebo hodnoty hash funkcie [4]. Dátové sklady najčastejšie využívajú indexovú štruktúru typu *B+strom* a *Bitmap index*.

1.2.2 Partitioning

Nárast objemu dát v databázových tabuľkách môže viesť k spomaleniu a zníženiu efektivity operácií spracovania. Jednou z ďalších optimalizačných metód je rozdelenie tabuliek na menšie, ľahšie udržiavateľné časti nazývané ***partície*** [17], [23]. Táto optimalizačná metóda zjednodušuje údržbu databázy a zároveň prináša nárast výkonu a efektivity pri spracovaní dotazov [20], [24]. Partitioning je spôsob, ako rozdeliť databázové tabuľky a indexy, na menšie časti. Každá partícia je definovaná menom a špecifickými nastaveniami. Partitioning znižuje celkové náklady spracovania dotazov [23]. Práca s tabuľkou rozdelenou na partície na nelíši od práce s klasickými tabuľkami. Stále je to jedna entita, ku ktorej môžeme pristupovať pomocou DML operácií rovnakým spôsobom ako pri nepartíciovaných tabuľkách.

Za vytvorenie partícií je zodpovedný kľúč, ktorý môže pozostávať z jedného alebo viacerých stĺpcov tabuľky. Na základe hodnoty kľúča sa určuje, do ktorej partície sa daný záznam zaradí [10].

2 Dáta

Analytické spracovanie bude vykonávané nad reálnymi dátami, z ktorých sme vybrali dáta z leteckej dopravy a dáta z dopravných systémov, nad ktorými budú vytvárané rôzne štatistiky.

2.1 Dáta z leteckej dopravy

V rámci výskumu pre spoločnosť Helios sme pracovali s reálnymi dátami z leteckej dopravy, ktoré nám boli poskytnuté organizáciou EUROCONTROL. EUROCONTROL je celoeurópska, civilno-vojenská organizácia, ktorá sa snaží o to, aby európske letectvo bolo efektívnejšie, bezpečnejšie a s minimálnym dopadom na životné prostredie [5]. Cieľom projektu bola analýza dát, na základe ktorej bolo možné určiť vplyv letov na zmenu podnebia či vynaložené náklady na jednotlivé lety.

Dáta, s ktorými sme pracovali reprezentovali jednotlivé lety nad Európou počas 4 rokov. Všetky záznamy boli uložené v súboroch v CSV formáte. Dáta boli zozbierané z rokov 2015 – 2018, pričom pre každý rok sme mali k dispozícii údaje za každý kvartál. Dáta obsahovali letové informačné regióny (FIR), reguláciu a riadenie leteckých informácií (AIRAC), plánované a aktuálne preletené lety spolu so všetkými informáciami o nich. V *Tabuľke 2* je zobrazená štruktúra bodov aktuálnych letov z roku 2015 z prvého kvartálu

Tabuľka 2: Štruktúra dát z leteckej dopravy

ID letu	Poradové číslo bodu	Čas	Letová výška	Latitude	Longitude
184408024	0	01-03-2015 1:00:00	0	41.98000	-87.90500
184408024	1	01-03-2015 1:47:31	330	42.06361	-84.32950
184408024	2	01-03-2015 2:05:33	330	42.03611	-80.75360
...	...	...	...	...	...
184408024	33	01-03-2015 8:19:07	0	52.30806	4.76417

ID letu	Poradové číslo bodu	Čas	Letová výška	Latitude	Longitude
184408028	0	01-03-2015 0:03:00	0	49.00972	2.54778
184408028	1	01-03-2015 0:21:00	96	49.01222	2.36139
184408028	2	01-03-2015 0:22:07	143	49.03833	2.14667
. . .	. . .	. . .	. . .	. . .	. . .

2.2 Dáta z dopravných systémov

Ďalšie údaje, ktoré tvoria základ pre vyhodnocovanie, experimentovanie a overovanie vytvorených metód, indexov a funkcií, sú reálne dáta z dopravných systémov Českej republiky. Tieto datasety sú verejne dostupné na webovej stránke polície Českej republiky *https://www.policie.cz/default.aspx*. K dispozícii sú údaje z dopravných systémov pre každý mesiac od roku 2015 do 2023. Datasety obsahujú niekoľko súborov, ktoré zaznamenávajú informácie o tom, kde vznikli nehody, na akom type cestnej komunikácie došlo k nehode, aký druh vozidla spôsobilo nehodu a mnoho ďalších parametrov. Súčasťou množiny dát je aj súbor s informáciami o chodcoch. Poskytnuté údaje sú v podstate modelom dátového skladu, kde je centrálna tabuľka faktov s odkazom na jednotlivé dimenzie (číselníky) v osobitných súboroch.

3 Výskum

Experimenty boli vykonávané na databázovom prostredí Oracle, pretože Oracle je výkonný a všestranný databázový systém, ktorý ponúka širokú škálu funkcií, ktoré sú relevantné pre analytické prostredie, poskytuje technológie zamerané na analytické spracovanie, ako napríklad Oracle In-Memory Analytics, Oracle Data Science Cloud, Oracle Data Warehouse. Okrem toho má množstvo výhod. Je jedným najrozšírenejších databázových systémov. Poskytuje riešenia s vysokým výkonom a škálovateľnosťou, zaručuje vysokú dostupnosť a spoľahlivosť a zároveň ponúka aj rozsiahly súbor nástrojov pre zálohovanie a obnovenie dát. Z pohľadu analýzy dát, dokáže spracovávať veľké objemy dát s vysokou rýchlosťou a efektivitou čím je ideálny pre Big Data analýzu a aplikácie.

3.1 Metódy importu a exportu dát

V rámci výskumu sme sa zamerali na porovnanie metód importu dát do databázy a exportu dát z databázy pre transakčné spracovanie. Celou touto problematikou sa venuje publikácia [HD4]. Cieľom výskumu bolo:

- porovnanie rýchlostí jednotlivých metód importu a exportu dát
- určenie najefektívnejšej možnosti importu a exportu dát
- porovnanie rýchlostí vykonania príkazu *SELECT* v interných a externých tabuľkách

Všetky typy experimentov boli testované na troch rôznych dátových setoch, ktoré sú znázornené v *Tabuľke 3*. Najmenší z nich obsahoval približne 20 000 záznamov, stredný obsahoval 6,5 milióna záznamov a najväčší dátový set mal takmer 32,5 milióna záznamov.

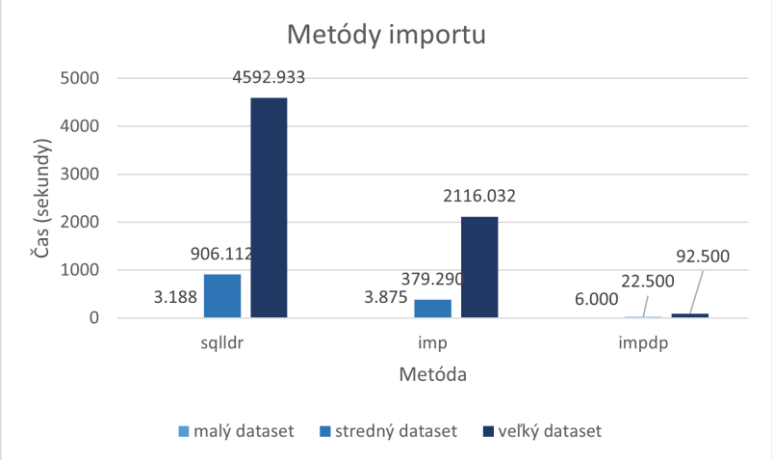
Tabuľka 3: Dátové množiny

Dátový set	Počet záznamov
Malý dátový set	19 532

Dátový set	Počet záznamov
Stredný dátový set	6 500 000
Veľký dátový set	32 400 000

3.1.1 Metódy importu dát

Metódy importu dát sú metódy, ktoré umožňujú obnoviť, respektíve vložiť dáta zo zdrojového súboru do databázy. V rámci výskumu boli testované a porovnávané tri typy importných metód: *metóda SQL*Loader, metóda IMP, metóda IMPDP*.

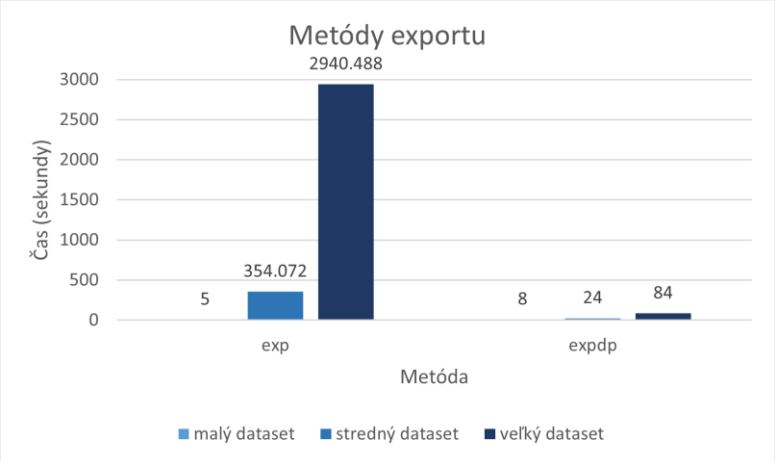


Obrázok 2: Čas trvania importu dát do databázy (sekundy)

Ako je znázornené *Obrázku 2*, s malou množinou dát sú výsledky jednotlivých metód porovnateľné. Malé odchýlky môžu byť spôsobené aktuálnym zaťažením servera. Najväčší rozdiel v čase vykonávania importu sa prejavil na veľkej množine dát. Import trval pri metóde SQLLDR viac ako hodinu a 16 minút, pri metóde IMP trval okolo 35 minút a najrýchlejšie sa vykonal pri metóde IMPDP, kde trval minútu a 30 sekúnd.

3.1.2 Metódy exportu dát

Metódy exportu dát z databázy sú metódy, ktoré umožňujú zálohovať, respektíve uložiť dáta z databázy do cieľového súboru. V rámci výskumu boli testované a porovnávané dva typy exportných metód: *metóda EXP* a *metóda EXPDP*.



Obrázok 3: Čas trvania exportu dát z databázy

Ako je znázornené na *Obrázku 3*, s malou množinou dát sú výsledky jednotlivých metód exportu porovnateľné, rovnako ako pri metódach importu. Rozdiely v trvaní exportu začínajú byť viditeľné až pri väčších množinách dát. Pri strednom dátovom sete metóda

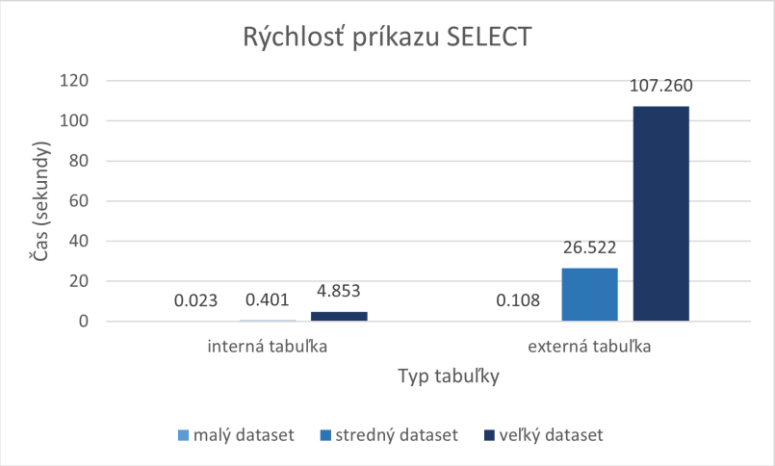
EXP trvá približne 6 minút, zatiaľ čo metóda EXPDP trvá 24 sekúnd. Najväčší rozdiel v čase vykonávania exportu sa prejavil na veľkej množine dát. Čas vykonávania metódy EXP trval približne 50 minút. Na druhej strane export prostredníctvom metódy EXPDP trval len 84 sekúnd.

Záver tohto experimentu priniesol výsledok, že najrýchlejšia metóda pre export dát do databázy je metóda EXPDP, pretože rovnako, ako pri metódach importu, aj táto metóda pracuje len na strane servera, čím sú eliminované dodatočné náklady na prepájanie klienta a servera.

3.1.3 Externé a interné tabuľky

Nasledujúci typ experimentu porovnáva rýchlosť vykonávania príkazu *SELECT* v interných a externých tabuľkách. Rozdiel medzi internými a externými tabuľkami je v tom, že externé tabuľky pristupujú k dátam, ktoré sú uložené v externom zdroji, to znamená, že dáta nie sú priamo uložené v databáze.

Ako je znázornené *Obrázku 4,* s malou množinou dát je trvanie príkazu *SELECT* v internej tabuľke v priemere 0.023 sekundy, zatiaľ čo v externej tabuľke je to 0.108 sekundy. Väčšie rozdiely sa vyskytnú pri strednej množine dát. Interná tabuľka je schopná vykonať príkaz *SELECT* v priemere za 0.401 sekundy, pričom v externej tabuľke to trvá v priemere 26.522 sekúnd. Najväčší rozdiel nastal pri veľkej množine dát. Priemerný čas trvania príkazu *SELECT* v internej tabuľke je 4.853 sekúnd, zatiaľ čo v externej tabuľke to v priemere trvalo 1 minútu a 47 sekúnd.



Obrázok 4: Rýchlosť vykonania príkazu SELECT

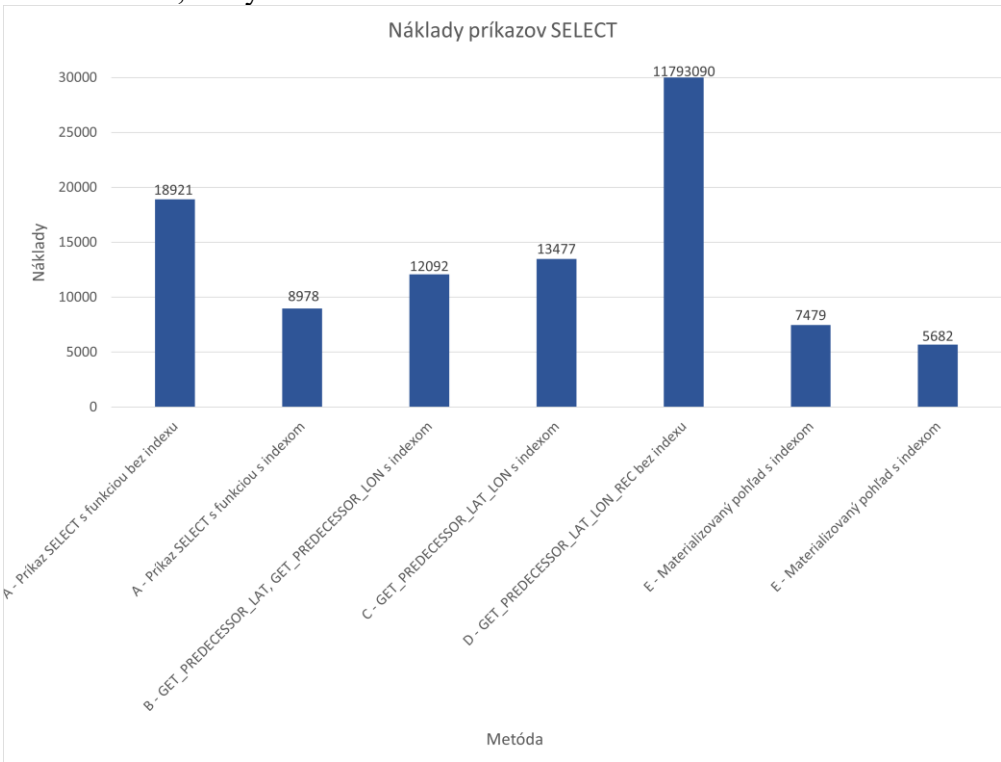
Záver tohto experimentu priniesol výsledok, že vykonanie príkazu *SELECT* je v internej tabuľke rýchlejšie ako v externej tabuľke. Je to z toho dôvodu, že sú dáta uložené priamo v databázových tabuľkách. Mapovanie externých tabuliek je veľmi rýchle, ale musia byť prepojené k externému zdroju, ktorý je uložený na strane servera, a preto je čas vykonania príkazu *SELECT* dlhší.

3.2 Optimalizácia príkazu SELECT

Existuje niekoľko spôsobov ako optimalizovať SQL príkazy. V rámci optimalizácie príkazu *SELECT* sme sa zamerali na porovnanie piatich rôznych metód vytvorenia dotazu spolu s vytvorením vhodných indexov, ktoré môžu zlepšiť plán vykonania daného dotazu. Prvou metódou je klasický príkaz *SELECT* obsahujúci priamo analytickú funkciu LAG(), ktorá vracia hodnotu definovaného stĺpca s definovaným posunom pred aktuálnym riadkom. Ďalej nasledujú tri rôzne používateľom definované funkcie a poslednou porovnávanou metódou je použitie materializovaného pohľadu. Všetky porovnávané metódy vrátia

rovnaký výsledok, len iným spôsobom. Týmto výskumom sa bližšie zaoberá publikácia [HD7].

Na demonštrovanie experimentov sme využili dáta z leteckej dopravy, resp. letové údaje, v ktorých sme získali hodnoty súradníc z predchádzajúceho záznamu, ktoré sme následne použili na výpočet vzdialenosti medzi dvoma bodmi. Všetky experimenty boli testované na datasete, ktorý obsahoval 3 360 932 záznamov.



Obrázok 5: Stĺpcový graf s celkovými nákladmi porovnávaných metód

Ako je možné vidieť na stĺpcovom grafe, najvyššie náklady na vykonanie príkazu *SELECT* dosiahla používateľom definovaná funkcia GET_PREDECESSOR_LAT_LON_REC(), pretože nebolo možné nad ňou vytvoriť vhodný index, keďže návratová hodnota tejto funkcie je typu *RECORD*. V poradí druhou metódou s najvyššími celkovými nákladmi na vykonanie bol klasický príkaz *SELECT* obsahujúci priamo analytickú funkciu LAG() bez využitia indexu. Pri využití používateľom definovaných funkcií GET_PREDECESSOR_LAT(), GET_PREDECESSOR_LON() a GET_PREDECESSOR_LAT_LON(), sú celkové náklady na vykonanie porovnateľné. Najlepšie výsledky dosiahol materializovaný pohľad s vytvoreným indexom, ktorý optimalizoval daný príkaz a znížil celkové náklady na jeho vykonanie na hodnotu 5 682.

3.3 Návrh dátového skladu

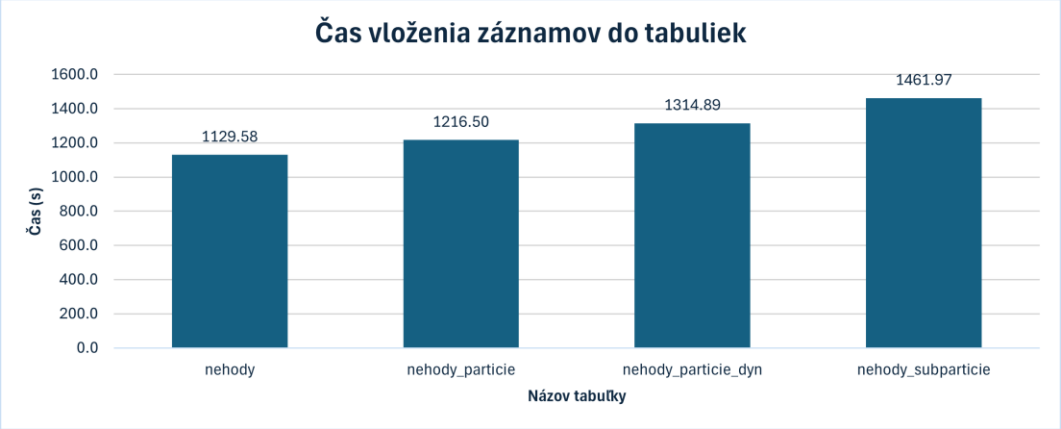
V rámci analytického spracovania dát je potrebné získané údaje určitým spôsobom uchovávať v databáze. Dátový sklad je vhodnou metódou ako tieto údaje ukladať. My sme sa v rámci nášho výskumu zamerali na vytvorenie dátového skladu pre policajné dáta, ktoré následne využívame na ďalší výskum zameraný na optimalizáciu dátovej štruktúry a optimalizáciu vykonávania dotazov v dátových skladoch.

Policajné dáta z dopravných nehôd, ktoré sa zaznamenávajú, majú jednotnú štruktúru. V rámci daného roku sú tieto dáta zbierané mesačne pre jednotlivé kraje Českej Republiky. Okrem samotných nehôd sa zaznamenávajú aj informácie o chodcoch, ktorí boli súčasťou dopravnej nehody. Datasety o nehodách, predstavujú tabuľku faktov a jednotlivé dimenzie reprezentujú číselníky, ktoré sú taktiež k dispozícii.

3.4 Optimalizácia dátovej štruktúry

Po vytvorení dátového skladu pre dáta, ktoré boli dostupné z dopravných nehôd sme pristúpili k experimentom, ktorých cieľom bolo určiť najefektívnejší spôsob vkladania a vyhľadávania údajov do dátového skladu, resp. tabuľky faktov.

V princípe sme vytvorili niekoľko tabuliek faktov pre nehody, ktoré sa líšili v spôsobe vytvorenia. Prvou z nich bola klasická tabuľka faktov s názvom *nehody*, ktorá obsahovala kľúče z jednotlivých dimenzionálnych tabuliek a ďalšie dodatočné informácie. Druhú tabuľku faktov, ktorú sme vytvorili sa nazývala *nehody_particie*. Táto tabuľka obsahuje rovnaké atribúty, avšak líši sa v tom, že má definované partície podľa dátumu nehody. Všetky partície, do ktorých budú dáta rozdelené sú vymenované v definícii tabuľky. Tretí typ tabuľky je tabuľka faktov s názvom *nehody_particie_dyn*. Tabuľka je rozdelená do partícií podľa dátumu nehody, avšak partície sa vytvárajú dynamicky, to znamená, že sa vytvorí toľko partícií, koľko rôznych rokov budú obsahovať údaje. Ďalší typ tabuľky je tabuľka s názvom *nehody_subparticie*. Táto tabuľka je rozdelená na partície podľa rokov a subpartície podľa mesiacov.



Obrázok 6: Stĺpcový graf - čas vloženia záznamov do tabuliek

Ako je možné vidieť na *Obrázku 6*, čas vkladania záznamov do klasickej tabuľky faktov bez partícií je najnižší, a to z toho dôvodu, že nepotrebuje dodatočnú réžiu na vytváranie jednotlivých partícií. Na druhej strane, tabuľka faktov, v ktorej sa vytvárali subpartície mesiacov pre jednotlivé partície rokov, mala čas vkladania záznamov do databázy najvyšší.

Cieľom ďalších experimentov bolo určiť najefektívnejší spôsob vyhľadávania údajov v rôznych tabuľkách, ktoré uchovávajú dáta rozličnými spôsobmi:

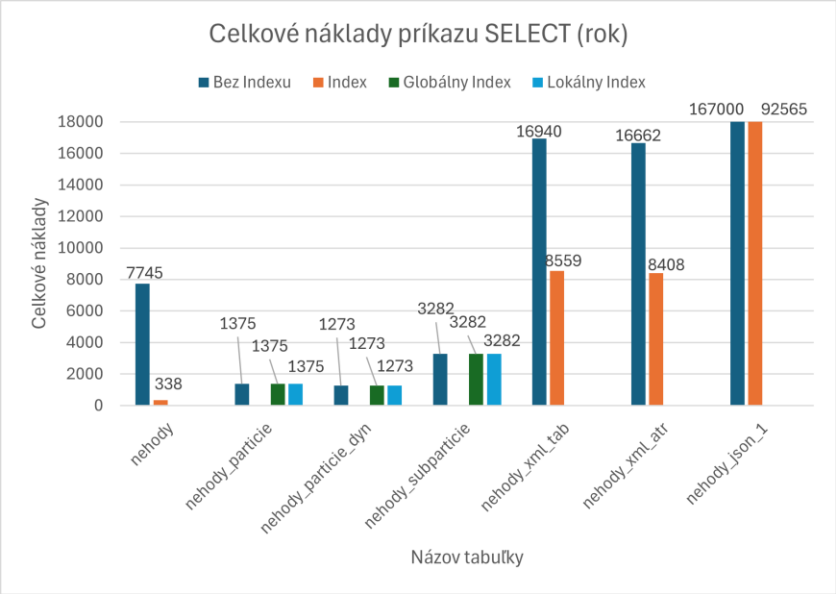
- klasická tabuľka faktov
- tabuľky rozdelené na partície (vymenovaním, dynamicky)
- tabuľka rozdelená na subpartície
- tabuľky uchovávajúce objekty typu XML a JSON

Efektívnosť vyhľadávania sme sa snažili zlepšiť vytvorením vhodných indexov, ktoré zlepšia rýchlosť vyhľadávania v jednotlivých tabuľkách. Príkaz *SELECT*, ktorý budeme testovať bude vyberať údaje na základe dátumu nehody, pričom budeme pozorovať ako sa výkonnosť príkazu mení v prípade, že vyberáme údaje podľa:

- definovaného roku
- definovaného mesiaca v danom roku

- definovaného mesiaca vo všetkých rokoch

Aby sme mohli pracovať s objektami typu XML a JSON, bolo potrebné vytvoriť ďalšie tabuľky a vložiť do nich všetky záznamy.

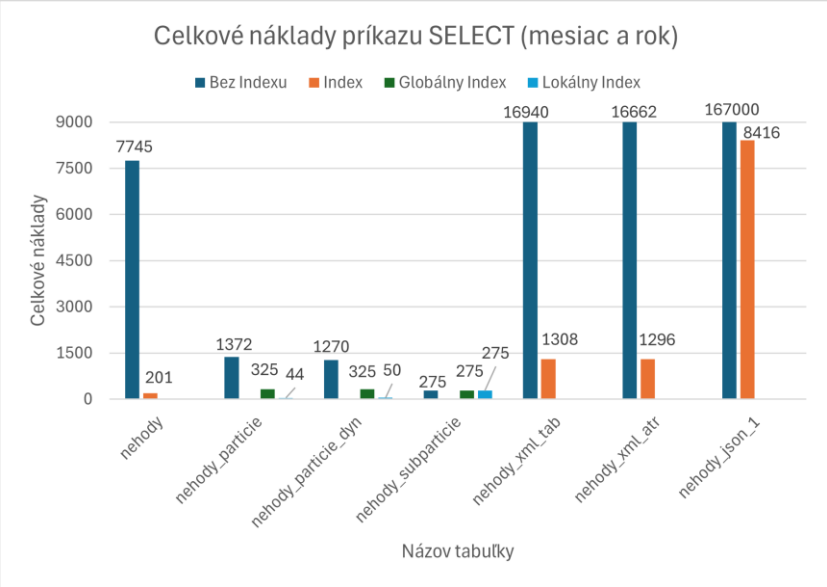


Obrázok 7: Stĺpcový graf celkových nákladov príkazu SELECT (rok)

Na *Obrázku 7* je znázornený stĺpcový graf pre prvý typ príkazu *SELECT*, ktorý vyberá záznamy len pre rok 2018. Tmavomodrou farbou sú zobrazené náklady na vykonanie príkazu bez využitia indexov. Vidíme, že s vytvorením indexov sa náklady na vykonanie príkazov značne znížili. Stĺpce označené oranžovou farbou predstavujú náklady na vykonanie príkazov s využitím klasických indexov. Tabuľky rozdelené na partície a subpartície majú 2 typy indexov, a preto sú stĺpce označené zelenou a slabomodrou farbou. Tabuľka, ktorá obsahovala JSON dokumenty sa ukázala ako najmenej efektívna, pretože náklady sú príliš vysoké aj po vytvorení indexu. Medzi ďalšie menej efektívne možnosti ukladania záznamov sa zaradili tabuľky, ktoré obsahujú XML dokumenty. Tabuľka so subpartíciami dosahuje relatívne dobré výsledky, avšak rovnako ako aj pri tabuľkách s partíciami náklady na vykonanie príkazov sú rovnaké, keďže vytvorené indexy neboli použité pre tieto príkazy a to z toho dôvodu, že vďaka tomu, že sme pristupovali priamo k partícii daného roku nebola potrebná podmienka *WHERE*. Klasická tabuľka faktov bez využitia indexov nie je tak efektívna, ako tabuľky rozdelené na partície, na druhej strane hodnota celkových nákladov na vykonanie príkazu s vytvoreným indexom oproti ostatným možnostiam sa javí ako najefektívnejšia možnosť keďže ako jedinej sa podarilo dosiahnuť náklady 338.

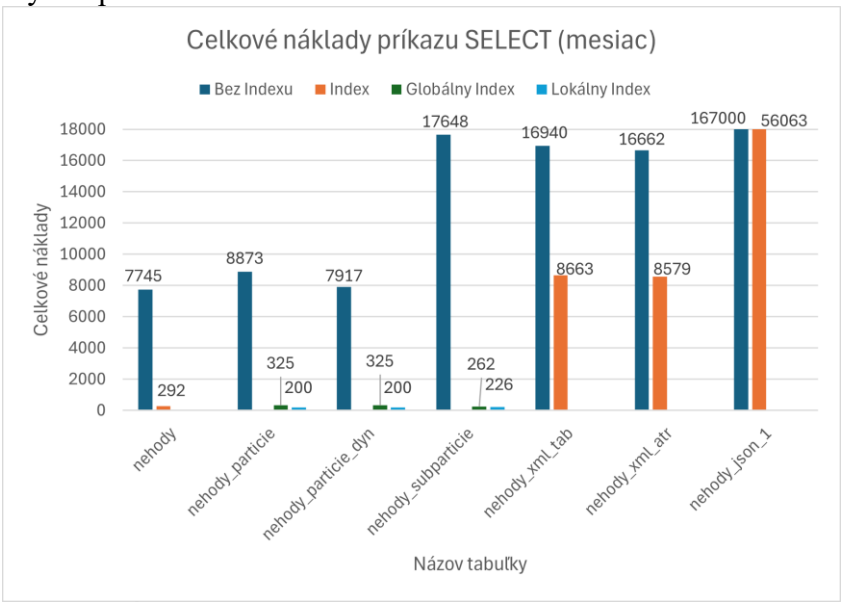
Na *Obrázku 8* je znázornený stĺpcový graf pre druhý typ príkazu *SELECT*, ktorý obsahoval podmienku vyberajúcu záznamy z mája roku 2018. Na obrázku vidíme, že náklady na vykonanie príkazov pre jednotlivé tabuľky bez využitia indexov sú omnoho vyššie ako náklady na vykonanie príkazov s vytvorenými indexami. Opäť je najmenej efektívna možnosť vyhľadávania záznamov v tabuľke uchovávajúcej JSON dokumenty, či bez vytvorených indexov alebo s indexami. XML tabuľky sú druhou najmenej efektívnou možnosťou. Náklady na vykonanie sú však v tabuľke, ktorá uchováva XML dokumenty ako jeden zo stĺpcov, menšie ako tabuľka XML dokumentov. Tabuľka subpartícií má stabilné hodnoty celkových nákladov, keďže počas experimentov s indexami optimalizátor nevyužil ani jeden, avšak v rámci príkazu sa pristupovalo priamo k záznamom z 5 mesiaca roku 2018. Tabuľky rozdelené na partície majú hodnoty nákladov porovnateľné, rozdiel by mohol nastať v prípade, ak by systém dynamicky vytvoril viac partícií, ako keď by sme v definícií

tabuľky definovali pevne stanovené partície. V tomto prípade však dosahujú najlepšie výsledky vyhľadávania záznamov spomedzi všetkých tabuliek s vytvoreným lokálnym indexom. Klasická tabuľka faktov bez vytvorených indexov nedosahuje až tak dobré výsledky ako s vytvoreným indexom, vďaka ktorému sa hodnota celkových nákladov dokázala znížiť na 201.



Obrázok 8: Stĺpcový graf celkových nákladov príkazu SELECT (mesiac, rok)

Na *Obrázku 9* je znázornený stĺpcový graf pre posledný typ príkazu *SELECT*, ktorý vyberal záznamy len podľa mesiaca.



Obrázok 9: Stĺpcový graf celkových nákladov príkazu SELECT (mesiac)

Na obrázku vidíme, že náklady na vykonanie príkazu bez vytvorených indexov sú podstatne vyššie ako s indexami. Tabuľka, ktorá uchováva JSON dokumenty je aj v tomto prípade najmenej efektívnou metódou pre vyhľadávanie záznamov aj napriek vytvoreným indexom. Tabuľky obsahujúce XML dokumenty majú približne rovnaké náklady na vykonanie príkazu bez vytvorených indexov, ale aj s indexami. Hodnota nákladov v tabuľke rozdelenej na subpartície bez indexov bola porovnateľná ako hodnoty nákladov pre XML dokumenty. S vytvorenými indexami sa náklady na vykonanie príkazu v tabuľke rozdelenej na subpartície značne znížili a vďaka lokálnemu indexu dosahovala takmer najlepšie hodnoty celkových nákladov na vykonanie definovaného príkazu. Klasická tabuľka faktov

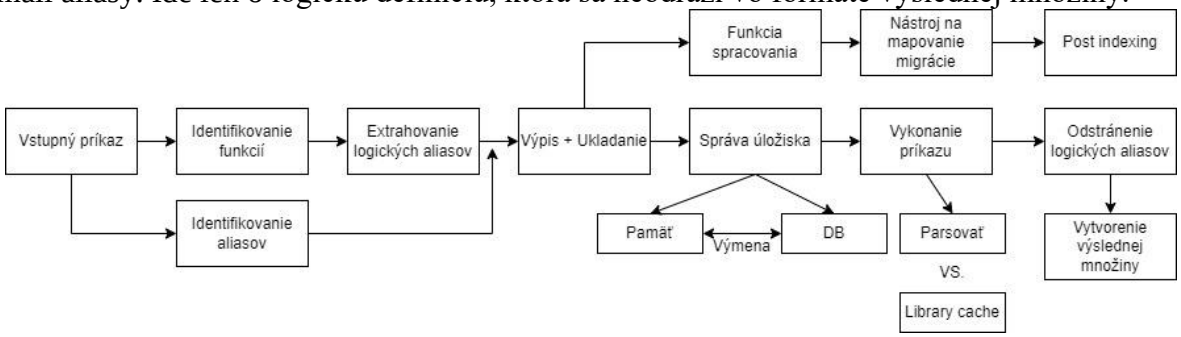
a tabuľky rozdelené na partície majú bez indexov približne rovnaké náklady na vykonanie dotazu. Rozdiel nastáva pri vytvorení indexov, avšak celkové náklady v týchto tabuľkách sú približne rovnaké. Každopádne tabuľky rozdelené na partície spolu s vytvoreným indexom dosahujú najlepšie výsledky. Taktiež je možné vidieť rozdiel medzi globálnymi a lokálnymi indexami v tabuľkách rozdelených na partície a subpartície. Lokálne indexy dosahujú lepšie výsledky ako globálne indexy.

3.5 Referencie funkcií v príkaze SELECT

Mnoho informačných systémov je zameraných na dátovú analytiku, v rámci ktorej sa produkujú výsledky v akejkoľvek grafickej podobe, na to aby umožnili lepšie pochopiť procesy a spôsob rozhodovania sa na základe vytvorených prognóz [21]. Za získané výstupy môžeme vďačiť dátovej vrstve a jej metódam na prístup, manipuláciu a spracovanie dát. V rámci výskumu sme sa zamerali na spracovanie údajov a optimalizáciu výkonu pričom sa orientujeme na spracovanie a referencie funkcií. Existujúce riešenia správy funkcií zahŕňajú ukladanie výsledkov do vyrovnávacej pamäte, funkcionálne indexy, materializované pohľady, virtuálne stĺpce, alebo optimalizáciu funkcií tak, aby sa dali priamo aplikovať v SQL alebo PL/SQL.

Oracle Database 23c predstavila niekoľko možností zlepšenia výkonu a nový prístup k aliasom výrazov a stĺpcov. Nami navrhované riešenie sa zameriava na identifikáciu a extrakciu aliasov stĺpcov, výrazov a funkcií, ukladanie referencií v pamäti a databázovej vrstve, optimalizáciu prenosu medzi nimi pomocou výmeny (presunu), ako aj na migráciu kontrolných bodov a volania funkcií. Poskytuje komplexnú architektúru, ktorá zahŕňa riadenie transakcií. Okrem toho poskytuje aj robustnú vrstvu, pričom každá vrstva je detailne analyzovaná z hľadiska výkonu jej výhod a obmedzení. Naša navrhovaná architektúra prináša výkonnostné benefity pre komplexné analytické dotazy, a taktiež, môže byť aplikovaná pri online spracovaní transakcií. Cieľom je extrahovať aliasy a zabezpečiť ich použiteľnosť v rámci celého procesu vykonávania dotazov, pričom sa snažíme znížiť časové nároky a celkové náklady na spracovanie dotazu. Celý popis štúdie a navrhovaného riešenia spolu s ďalšími podrobnosťami výskumu je možné nájsť v publikácii [9].

Zavádzame vlastné identifikátory a logické rámce aliasov pre volania funkcií, vďaka ktorým je možné sa na tieto funkcie priamo odkazovať a spracovávať ich rovnako ako keby mali aliasy. Ide len o logickú definíciu, ktorá sa neodrazí vo formáte výslednej množiny.



Obrázok 10: Schéma navrhovaného riešenia

Navrhované riešenie začína s identifikáciou a extrakciou aliasov umiestnených v klauzule *SELECT*, po ktorej nasleduje výpis aliasov. Pre tento účel sa zavádza nový proces na pozadí QUAM, ktorý predstavuje hlavný proces dohliadajúci na činnosti worker procesov QUAWn. Extrahované aliasy a volania funkcií sa umiestnia do optimalizovanej dátovej štruktúry - Query Alias Manager Structure, ktorá sa nachádza v pamäti inštancie. Na základe evaluačnej štúdie sme dospeli k záveru, že ukladanie výhradne v pamäti je nepraktické kvôli nárokom na zdroje a kapacitu pamäte. Preto sa štruktúra presunula z privátnej oblasti do zdieľaného úložiska, čo umožňuje zdieľanie rovnakých plánov

vykonania a extrahovaných prvkov. Preto sme zaviedli lokálne úložisko dotazov, definované výmenou medzi pamäťou inštancie a databázou. Na zistenie konzistentnosti, spoľahlivosti, odolnosti voči chybám a obnoviteľnosti sa používali transakcie. To však prinieslo dodatočné nároky na spracovanie, obmedzenie extrahovania aliasov a správy volania funkcií v samostatných autonómnych transakciách. Okrem toho sme zaviedli pravidlá transakcií a checkpointing správy aliasov na zabezpečenie výkonu a robustnosti. Schéma celého riešenia je znázornená na *Obrázku 10.*

3.6 Spracovanie odkazov temporálnych funkcií v relačných databázach

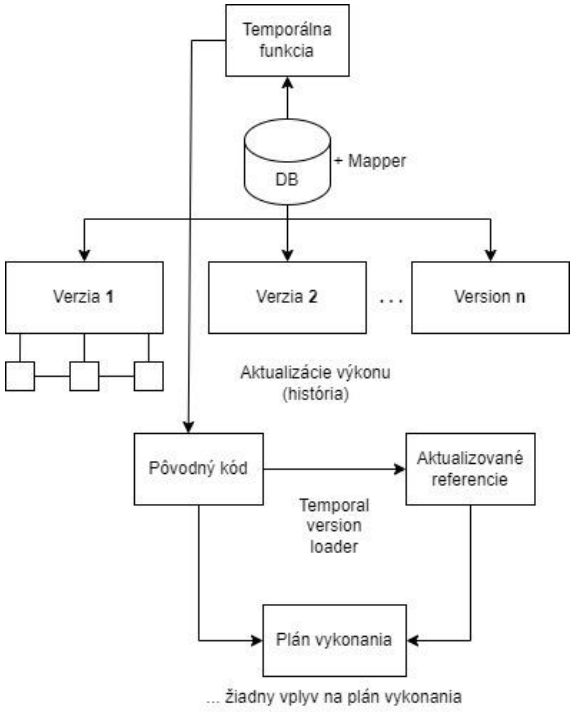
Počas vývoja databázových technológií a paradigmy SQL bol zavedený koncept temporálnych databáz. S vývojom objektov a jednotlivých reprezentácií stavov je potrebné riešiť aj odkazy na funkcie a ich vývoj v čase. Preto sme sa bližšie zamerali na riešenia správy funkcií, ktorých kód sa môže v priebehu času meniť, a to nie len z hľadiska optimalizácie kódu, ale aj z hľadiska produkcie výstupov. Výpočty a spracovanie dát sa môžu vyvíjať a pre rovnaké vstupy môžu produkovať rôzne výstupy na základe časových verzií funkcie. Naším cieľom je poskytnúť robustné riešenie transformácie a mapovania, ktoré kladie dôraz na jednotlivé verzie funkcií a volá relevantné funkcie na základe odkazovaných časových rámcov. Vďaka poskytnutej spoľahlivej transformačnej vrstve môže existovať viacero verzií funkcií, pričom sa rešpektuje platná verzia funkcie, ktorá existovala v čase platnosti príslušnej n-tice [3] [16]. Navrhované riešenie je založené na správe multiverzií prostredníctvom nových klauzúl, príkazov a procesov na pozadí, ktoré autonómne spravujú celý systém. Navyše jednotlivé verzie môžu byť neskôr vylepšené o optimalizácie výkonu, ako sú sofistikovanejšie dátové štruktúry, vďaka čomu môže mať každá verzia viacero implementácií. Celý popis štúdie a navrhovaného riešenia spolu s ďalšími podrobnosťami výskumu je možné nájsť v publikácii [HD8].

V našom navrhovanom riešení sa primárne snažíme zabezpečiť ***vysokú mieru paralelizmu***, ktorý sa snažíme dosiahnuť obmedzením nutnosti parsovania, načítavania a transformácie medzi verziami, ***dynamickému mapovaniu verzií***, pri ktorom zostáva hlavička jednej temporálnej funkcie rovnaká pre všetky verzie, ***identifikácii a registrácii verzii*** na obmedzenie testovacích prípadov, ***aktualizácii a implementácii verzii*** na zlepšenie výkonu. Cieľom tohto riešenia je zabezpečiť aby sa vždy vykonala najefektívnejšia implementácia danej verzie a ***registrácii novej verzie funkcie***, ktorá sa stane platnou v definovanom časovom bode.

Na *Obrázku 11* sa nachádza architektúra navrhovaného riešenia. Jednotlivé verzie sú kontrolované procesom Temporal_version_leader na pozadí. Tento proces zabezpečuje načítanie verzie, ako aj správne mapovanie implementácie jednotlivých verzií, pričom sa kladie dôraz na ich platnosť. Navyše sa tu nachádza proces Temporal_registrator, ktorý je zodpovedný za zaznamenávanie novej verzie v temporálnej vrstve riadiacich funkcií. Okrem toho kontroluje konzistenciu časových rámcov, aby sa obmedzilo prekrývanie verzií v čase. Správca temporálnych funkcií poskytuje:

- Registráciu temporálnej funkcie
- Registráciu novej verzie (platná okamžite, platná v budúcnosti)
- Registráciu zmien implementácie verzie
- Kompiláciu po registrácii novej verzie / zmeny implementácie verzie
- Zakázať VŠETKO – to znamená, že temporálna funkcia už nebude prijímať žiadne nové verzie ani zmeny v implementácii danej verzie

- Zakázať verziovanie – to znamená, že temporálna funkcia už neprijíma žiadne nové vezie, ale implementácia existujúcej verzie sa môže zmeniť, ak to má vplyv na zlepšenie výkonu
- Zlúčenie verzií – táto funkcionalita sa použije v prípade, ak sa kód niekoľkokrát kompiluje, čím sa vytvárajú nové verzie, ak je kód rovnaký, tak sa tieto verzie môžu zlúčiť dohromady.



Obrázok 11: Navrhovaná architektúra verziovania temporálnych funkcií

3.7 Problematika nedefinovaných hodnôt, chybné dáta

Problematika nedefinovaných hodnôt a chybných dát pri ukladaní a spracovaní v databázových systémoch je dôležitou témou, keďže v rámci dátových skladov s obrovským množstvom dát je potrebná ich správna interpretácia. Neúplné dáta, prípadne nevedomosť správneho použitia funkcií môže viesť ku skresleným výpočtom a chybnej prezentácií výsledkov, čo môže mať nežiadúce následky pre spoločnosť, pre ktorú sa vykonávala daná analýza.

Veľmi často sa stretávame s tým, že niektoré hodnoty stĺpcov nie sú vyplnené, sú prázdne a tým nadobúdajú hodnotu NULL. Ako s takýmito stĺpcami pracovať aby sme si boli istí, že výsledky budú správne?

V prípade, že chceme nejakým spôsobom ošetriť NULL hodnoty a pracovať s nejakou nami definovanou hodnotu namiesto NULL, existujú funkcie, ktoré dokážu transformovať NULL hodnoty na nami definovanú hodnotu. Medzi tieto funkcie patrí: NVL(), COALESCE(), DECODE(). Pri NULL hodnotách je taktiež dôležité vedieť, že ich nie je možné porovnávať cez znamienko =, ale cez podmienku IS NULL prípadne IS NOT NULL.

3.7.1 NULL hodnoty a partície

Partície v databázových systémoch slúžia na rozdelenie tabuľky na menšie časti, s cieľom zvýšiť výkon dotazov a zlepšiť udržiavanie tabuliek. Pri definícií partícií je potrebné uviesť stĺpec tabuľky, podľa ktorého sa budú partície vytvárať. Pokiaľ je stĺpec definovaný ako NOT NULL stĺpec, nie je to žiadny problém. Ale čo v prípade ak stĺpec podľa ktorého vytvárame partície môže nadobúdať aj NULL hodnoty? Do akej partície takýto stĺpec zaradiť? Vloží sa taký záznam do tabuľky?

NULL hodnoty v partíciách je možné riešiť niekoľkými spôsobmi. Dôležité je správne definovať partíciu, kam by tieto NULL hodnoty spadali. Hlavným cieľom nasledujúceho výskumu je vytvoriť metodiku spracovania záznamov s NULL hodnotami v tabuľkách rozdelených na partície a ukázať ako je potrebné definovať tabuľky a ich partície tak, aby bolo možné vkladať záznamy s NULL hodnotami kľúčových stĺpcov partícií do tabuliek. Celý popis štúdie a navrhovaného riešenia spolu s ďalšími podrobnosťami výskumu je možné nájsť v publikácii [HD9].

Pre experimenty sme vybrali databázový systém Oracle, ktorý poskytuje širokú škálu možností riešenia tejto problematiky. Navyše podporuje niekoľko druhov rozdeľovania tabuliek na partície. V našom výskume sme sa zamerali na 3 typy partícií: *Range, List* a *Hash partitioning*. Pre každý typ rozdelenia tabuliek na partície sme vytvorili osobitné databázové tabuľky, do ktorých sme vložili záznamy s rôznymi hodnotami, pričom niektoré z týchto záznamov obsahovali aj NULL hodnoty kľúčových stĺpcov, podľa ktorých sa tabuľka delí na partície.

Range partitioning

Range partitioning umožňuje vytvárať partície priamo alebo dynamicky. V prvom kroku si ukážeme, akým spôsobom je potrebné definovať tabuľku, tak aby boli pokryté aj záznamy s NULL hodnotami.

```
CREATE TABLE tab_particie_null(
  id_zaznamu          NUMBER NOT NULL PRIMARY KEY,
  datum_vytvorenia    DATE,
  text                VARCHAR2(30))
PARTITION BY RANGE (datum_vytvorenia)(
     PARTITION part_2020
          VALUES LESS THAN (TO_DATE('01.01.2021', 'DD.MM.YYYY')),
     PARTITION part_2021
          VALUES LESS THAN (TO_DATE('01.01.2022', 'DD.MM.YYYY')),
     PARTITION part_2022
          VALUES LESS THAN (TO_DATE('01.01.2023', 'DD.MM.YYYY')),
     PARTITION part_20XX
          VALUES LESS THAN (MAXVALUE))
/
```

Definícia príkazu 2: Tabuľka partícií definovaných priamo

Databázová tabuľka (*def. príkazu 2*) je rozdelená na štyri partície, pričom prvá partícia pokrýva všetky záznamy, ktoré boli vytvorené do roku 2021, druhá partícia pokrýva záznamy vytvorené v roku 2021, tretia partícia pokrýva záznamy vytvorené v roku 2022 a posledná partícia uchováva záznamy s hodnotou roku vytvorenia vyššou ako 2022. Do takto vytvorenej tabuľky je možné vložiť aj záznamy s NULL hodnotou stĺpca, podľa ktorého sa vytvárajú partície. Všetky takéto záznamy budú automaticky uložené do poslednej partície, definovanej kľúčovým slovom *MAXVALUE*.

Druhou možnosťou, ako vytvárať partície, je pomocou intervalu. Tieto partície sa vytvárajú automaticky. Na začiatku sa určí prvá, počiatočná partícia. Pri vkladaní záznamu sa vezme hodnota záznamu v stĺpci, podľa ktorého sa delí tabuľka na partície, a vloží sa do vhodnej partície. Ak táto partícia neexistuje vytvorí sa automaticky nová partícia. Ak by sme

sa snažili vložiť záznamy s NULL hodnotami kľúčových stĺpcov do takejto tabuľky, systém by zobrazil chybu ORA-14300.

Riešenie tohto problému s NULL hodnotami pre kľúčový stĺpec, pomocou ktorého sa tabuľka rozdeľuje na partície, by mohol priniesť virtuálny stĺpec tabuľky, ktorý bude nadobúdať existujúcu hodnotu dátumu vytvorenia, alebo transformuje NULL hodnotu dátumu vytvorenia na nami preddefinovanú hodnotu. Tabuľku *tab_interval_particie_null* rozšírime o virtuálny stĺpec *DATUM_DEF*, ktorý sa bude automaticky počítať na základe toho, či je alebo nie je dátum vytvorenia vyplnený. Ak bude dátum vytvorenia nadobúdať hodnotu NULL, tak sa nastaví preddefinovaný dátum 01.01.2999. V časti *PARTITION BY RANGE* je taktiež potrebné zmeniť stĺpec *DATUM_VYTVORENIA* na *DATUM_DEF*. Tabuľku popisuje *def. príkazu 3*.

```
CREATE TABLE tab_interval_particie_null(
  id_zaznamu          NUMBER NOT NULL PRIMARY KEY,
  datum_vytvorenia    DATE,
  text                VARCHAR2(30),
  datum_def           DATE GENERATED ALWAYS AS
           (COALESCE(datum_vytvorenia,
                     TO_DATE('01.01.2999', 'DD.MM.YYYY'))) VIRTUAL)
PARTITION BY RANGE (datum_def)
     INTERVAL( numtoyminterval(1, 'year'))(
  PARTITION part_1
           VALUES LESS THAN (TO_DATE('01.01.2021', 'DD.MM.YYYY')))/
```

Definícia príkazu 3: Tabuľka s virtuálnym stĺpcom

List partitioning

List partitioning umožňuj podobný spôsob rozdelenia tabuľky na partície ako Range partitioning, avšak rozdiel je v tom, že List partitioning definuje diskrétne hodnoty partícií, ktoré môžu záznamy nadobúdať. List partitioning ponúka dve možnosti riešenia problému NULL hodnôt. Prvou možnosťou je použitie partície typu *DEFAULT*, no pri tejto voľbe je dôležité uvedomiť si, že okrem záznamov s NULL hodnotami sa do nej budú ukladať aj záznamy s hodnotami, ktoré nepatria do žiadnej inej partície. Druhou možnosťou je použiť partíciu typu *NULL*, takže sa do tejto partície budú ukladať len záznamy s NULL hodnotami kľúčového stĺpca partície. Definíciu tabuľky popisuje *def. príkazu 4*.

```
CREATE TABLE tab_list_particie_null(
  id_zaznamu          NUMBER NOT NULL PRIMARY KEY,
  datum_vytvorenia    DATE,
  kod_statu           VARCHAR2(2)
  text                VARCHAR2(30))
PARTITION BY LIST (kod_statu)(
     PARTITION part_1 VALUES ('SK', 'CZ'),
     PARTITION part_2 VALUES ('BE', 'FR'),
     PARTITION part_3 VALUES ('DE', 'AT'),
     PARTITION part_4 VALUES ('HU', 'PL'),
     PARTITION part_null VALUES (NULL),
     PARTITION part_default VALUES (DEFAULT))/
```

Definícia príkazu 4: Definícia tabuľky - List partitioning

Hash partitioning

Ďalší spôsob ako rozdeliť databázovú tabuľku na partície je Hash partitioning. Nie je potrebné explicitne špecifikovať partície ako pri delení tabuľky s použitím Range alebo List partitioningu. V tomto prípade špecifikujeme len stĺpec, podľa ktorého sa vytvorí partícia

a počet partícií, ktoré sa vytvoria. Ak počet partícií nie je zahrnutý v definícii, systém vytvorí štandardne jednu partíciu.

Def. príkazu 5 reprezentuje vytvorenie tabuľky, ktorá využíva Hash partitioning. Partície sa vytvárajú pomocou stĺpca *DATUM_VYTVORENIA* a implicitne sme definovali vytvorenie štyroch partícií.

```
CREATE TABLE tab_hash_particie_null(
  id_zaznamu          NUMBER NOT NULL PRIMARY KEY,
  datum_vytvorenia    DATE,
  text                VARCHAR2(30))
PARTITION BY HASH (datum_vytvorenia)
  PARTITIONS
/
```

Definícia príkazu 5: Definícia tabuľky - Hash partitioning

Hash partitioning je navrhnutý tak, že dokáže bez problémov spracovať záznamy s NULL hodnotami kľúčových stĺpcov partícií. Systém vloží záznam s nedefinovanými hodnotami do partície na základe výpočtu jeho hash funkcie. Vo všeobecnosti v prípade Hash partitioningu, záznamy s NULL hodnotou kľúčového stĺpca partícií môžu byť uložené v ľubovoľnej partícii.

Záver

Dizertačná práca sa zameriava na oblasť analytického spracovania dát, cieľom ktorého je transformovať dostupné dáta na hodnotné informácie, ktoré sú kľúčové pre úspešné fungovanie mnohých firiem a organizácií. Tieto výstupy sa stávajú základom pre strategické rozhodnutia, vedúce k optimalizácii procesov, nárastu efektivity a celkovému napredovaniu.

Naším cieľom bolo zamerať sa na pozadie týchto procesov, efektivitu ukladania veľkého objemu dát do databáz a dátových skladov, pričom kľúčová je rýchlosť a výkonnosť vyhľadávania položiek v rámci úložiska. Dôraz sme kládli na správny návrh a optimalizáciu dátových a indexových štruktúr, ktoré sú základom pre dosiahnutie požadovanej efektivity. Okrem toho sa práca zaoberá návrhom vhodných algoritmov pre analytické spracovanie dát, v rámci čoho je dôležitá práca s agregačnými a analytickými funkciami. Pre zaistenie spoľahlivosti výsledkov je podstatná konzistencia a bezchybnosť spracovávaných dát. Z toho dôvodu bola práca zameraná aj na návrh a implementáciu vlastných algoritmov pre identifikáciu a odstránenie chybných, nedefinovaných údajov.

Výskum našej práce by sme mohli rozdeliť do dvoch častí – optimalizačná časť a algoritmická časť. Na začiatku sme sa venovali porovnaniu rôznych metód importu dát do databázy, pričom metóda dosahujúca najlepšie výsledky bola metóda importu pomocou dátovej pumpy. Obdobne to bolo aj pri exporte dát do databázy, pretože export pomocou dátovej pumpy dosahoval najlepšie výsledky. Bolo to z toho dôvodu, že import aj export pomocou dátovej pumpy sa vykonávajú len na strane servera, takže nie je potrebné prepájať stranu klienta so stranou servera. Ďalšou časťou výskumu bolo porovnanie rýchlostí príkazu *SELECT* v interných a externých tabuľkách, pričom interné tabuľky dosiahli omnoho lepšie výsledky.

Výskum zameraný na optimalizáciu príkazu *SELECT*, ktorý obsahoval analytické funkcie porovnával päť rôznych spôsobov. Naším cieľom bolo porovnať celkové náklady na vykonanie príkazov a optimalizovať ich tak, aby náklady boli čo najmenšie. Po vykonaní experimentov sme dospeli k záveru, že príkaz, ktorý využíval materializovaný pohľad, nad ktorým sme vytvorili index dosahoval najmenšie náklady na vykonanie príkazu obsahujúceho analytické funkcie.

Celý výskum využíva reálne dáta, ktoré bolo potrebné určitým spôsobom spracovať a navrhnúť dátové modely / dátové sklady na ich ukladanie. Pri návrhu sme vykonali

experimenty zamerané na rýchlosť vkladania veľkého množstva údajov do rôznych typov tabuliek, ktoré sme následne porovnali. Porovnávali sme štyri rôzne typy tabuliek, z ktorých sa ukázalo, že do klasickej tabuľky faktov sa údaje vložia najrýchlejšie. Zvyšné tabuľky mali čas vkladania dlhší, hlavne z dôvodu dodatočnej réžií pri vytváraní partícií a vkladania záznamov do vhodnej partície.

Na základe toho sme ďalej pokračovali optimalizáciou dátovej štruktúry, pričom sme porovnávali rýchlosti vyhľadávania záznamov v rôznych typoch tabuliek. Porovnávali sme celkové náklady troch príkazov *SELECT*. Prvý príkaz vyberal záznamy z definovaného roku, druhý príkaz z definovaného mesiaca a roku a posledný len z definovaného mesiaca. Aj keď sa pri vkladaní záznamov ukázalo, že do klasickej tabuľky faktov je možné vložiť záznamy najrýchlejšie, vyhľadávanie záznamov v klasickej tabuľke faktov neprinieslo také výsledky. V tomto prípade sa spomedzi vyhľadávania dokázalo, že tabuľky rozdelené na partície dosahujú najlepšie výsledky pri vyhľadávaní záznamov, čo môže byť kľúčové pokiaľ pracujeme s obrovským množstvom dát.

Ďalším komplexným predmetom skúmania bolo referencovanie funkcií v príkaze *SELECT,* v ktorom sme navrhli a experimentálne otestovali nami navrhnuté zlepšenie odkazovania sa na aliasy funkcií v rôznych častiach príkazu *SELECT*. Toto riešenie sme porovnali s existujúcim riešením predstaveným v Oracle 23c.

Okrem toho sme navrhli, implementovali a experimentálne otestovali spracovanie odkazov temporálnych funkcií v relačných databázach. V tomto výskume bližšie rozoberáme možnosti verziovania temporálnych funkcií a naše riešenia porovnávame už s existujúcimi.

Druhú časť výskumu sme venovali problematike nedefinovaných hodnôt a chybných dát, kde sme vytvorili metodiku pre spracovanie NULL hodnôt v stĺpcoch, ktoré predstavujú kľúčové stĺpce pre vytváranie partícií. Okrem toho sme sa snažili ukázať prácu s NULL hodnotami, akým spôsobom ich spracovať aby boli výsledky dosiahnutých riešení správne. S touto problematikou sme sa stretli aj pri práci na projekte, kde sme museli navrhnúť a implementovať príkazy umožňujúce doplnenie nedefinovaných hodnôt, pre ďalší výskum. V rámci výskumu sme pracovali aj na implementácií procedúr a funkcií, slúžiacich na analýzu leteckých dát.

Vďaka tejto práci môžeme ďalej rozvíjať poznatky v rôznych oblastiach, ako je napríklad návrh metodiky na automatizované vytváranie indexov na základe dopytov, jej implementácia a testovanie v databázovom prostredí. Do ďalšej oblasti výskumu by sme mohli zaradiť predikciu neúplných dát, s čím sa spája identifikácia a vytváranie vzorov, na základe ktorých by sme vedeli získať chýbajúce informácie. Medzi ďalšiu oblasť výskumu by sme mohli zaradiť návrh optimalizačných stratégií využitia analytických funkcií pre časovo orientované databázy. Okrem vyššie uvedených oblastí, existuje mnoho ďalších tém, ktoré by sa dali skúmať v rámci tejto problematiky.

Zoznam vlastných publikácií

[HD1] Durneková M., Michalková V., Kvet M. (2019). *Parking Information System.* Journal of Information, Control and Management Systems. – pp. 111-121 – ISSN: 1336-1716

[HD2] Hrínová Durneková M., Kvet M. (2020). *School Information System.* 18th IEEE International Conference on Emerging Technologies and Applications: Information and Communication Technologies in Learning.(ICETA) – pp. 176-181 – ISBN: 978-0-7381-2366-0

[HD3] Durneková M. (2021). *Principles of Data Warehouses in Data Analytics.* Mathematics in Science and Technologies (MIST). – pp. 21-25 – ISBN: 9798748088183

[HD4] Durneková M., Kvet M. (2021). *Data Import and Export Methods*. 29th Conference of Open Innovations Association (FRUCT). – pp. 423-428 – ISBN: 978-952-69244-5-8

[HD5] Durneková M., Kvet M. (2021). *Using Multi-Index Database Layer in Ad-Hoc Communication Networks*. 28th International Conference on Systems, Signals, and Image Processing (IWSSIP). – pp. 285-296 – ISBN: 978-80-227-5112-4

[HD6] Durneková M., Kvet M. (2022). *Information System for Asset Management in Buildings*. 20th Anniversary of IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA). – pp. 141-146 – ISBN: 979-8-3503-2032-9

[HD7] Hrínová Durneková M., Kvet M. (2023). *Optimization of the SELECT Statement Containing Window Functions*. International Conference on Information and Digital Technologies (IDT). – pp. 267-272 – ISBN: 979-8-3503-0586-9

[HD8] Hrínová Durneková M., Kvet M., Papan J. (2024). *Treating Temporal Function References in Relational Database Management System*. IEEE Access. – pp. 54535-54552 – doi: 10.1109/ACCESS.2024.3387046

[HD9] Hrínová Durneková M., Kvet M. (2024). *Referring Null Values in Partitioned Tables*. 35th Conference of Open Innovations Association (FRUCT). – aktuálne prijatý, ale nie je indexovaný

Zoznam použitej literatúry

[1] Bharti J., R.D.Morena, "An Efficient Query Optimization for Object Oriented Database", International Conference on Computing, Communication, Control and Automation, 2017.

[2] Chiheb F., Boumahdi F., Bouarfa H., "A New Model for Integrating Big Data into Phases of Decision-Making Process", Procedia Computer Science - *The 2nd International Conference on Emerging Data and Industry 4.0*, 2019, vol 151, pp. 636-642, ISSN 18770509, Doi: 10.1016/j.procs.2019.04.085

[3] Chavan S., Hopeman A., Lee S., Lui D., Mylavarapu A., Soylemez E., "Accelerating Joins and Aggregations on the Oracle In-Memory Database", *34th International Conference on Data Engineering*, 2018, pp.1453-1464

[4] Datta A. a Thomas H., „The cube data model: a conceptual model and algebra for on-line analztical processing in data warehouses," *Decision Support System*, zv. 27, pp. 289-301, 1999.

[5] EUROCONTROL, „EUROCONTROL Supporting European Aviation," [Online]. Available: www.eurocontrol.int.

[6] Golfarelli M., Rizzi S., "From Star Schemas to Big Data: 20+ Years of Data Warehouse Research", A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years , 2018, vol 31, pp 93-107

[7] Harvey R., Oracle Cloud Data preparation in Oracle Analytics Cloud, 2017.

[8] Khan R.A., Quadri S.M.K., "Business Intelligence: An Integrated Approach", Business Intelligence Journal, 2012, vol. 5, pp. 64-70

[9] Kvet M., a Papan J., "Enhancing Analytical Select Statements Using Reference Aliases", IEEE Access, 2024, vol. 12, pp 27311-27330

[10] Lim L., "Elastic Data Partitioning for Cloud based SQL Processing Systems", *International Conference on Big Data*, 2013, pp 8-16

[11] Maabreh K.S., "Optimizing Database Query Performance Using Table Partitioning Techniques", International Arab Conference on Information Technology, 2018, ISBN 978-172810385-3

[12] Malik U., Goldwasser M. a Johnston B., SQL for Data Analytics, Birmingham, 2019, ISBN: 978-1-78980-735-6
[13] Mannino M., Hong S. N. a Choi I. J., „Efficiency evaluation of data warehouse operations,“ *Decision Support Systems*, zv. 44, pp. 883-898, 2008.
[14] Matiaško K., Vajsová M. a Kvet M., Pokročilé databázové systémy 2. diel - Architektúra, programovanie s objektmi a XML, Žilina: EDIS, 2017.
[15] Moktadir A., Istiak Chowdhury N.M., "Subject Oriented Data Partitioning - A Proposed Data Warehousing Schema", *1st International Conference on Advances in Science, Engineering and Robotics Technology*, 2019, ISBN: 978-172813445-1
[16] Nindito H., Sano A.V.D., Condrobimo A.R., "Comparative Study of Storing Unstructured Data Type Between BasicFile and SecureFile in Oracle Database 12c", *International Conference on Information Management and Technology*, 2016, pp. 146-149
[17] Oracle, "Oracle Partitioning", 2019
[18] Oracle, "The Oracle Optimizer Explain the Explain Plan", 2017
[19] Oracle, "Understanding Optimizer Statistics", 2012
[20] Oracle, VLDB and Partitioning GUIDE, Web: https://docs.oracle.com/en/database/oracle/oracle-database/23/vldbg
[21] Rani G., Sharma T., a Sharma A., "Future Database Technologies for Big Data Analytics", Proceedings of the 2023 International Conference on Intelligent Systems for Communication, IoT and Security, 2023, pp. 349-354
[22] Surianszah B., Ilham A.A., Paundu A.W., "Optimization of Data Warehouse Architecture to Improve Information System Performance", *International Conference on Computer Science, Information Technology and Engineering*, 2023, pp 240-245, ISBN: 979-835032095
[23] Šalgová V., Matiaško K., "The Effect of Partitioning and Indexing on Data Access Time", *Conference of Open Innovation Association, FRUCT*, 2021, pp. 301-306
[24] Wisal K., Cheng Z., Bin L., Teerath K., Ejaz A, "Robust Partitioning Scheme for Accelerating SQL Database", *International Conference on Emergency Science and Information Technology, ICESIT*, 2021, pp 369-376