ŽILINSKÁ UNIVERZITA V ŽILINE

AUTOREFERÁT
DIZERTAČNEJ PRÁCE

Žilina, apríl 2022

Ing. Veronika Šalgová

Žilinská univerzita v Žiline
Fakulta riadenia a informatiky

Ing. Veronika Šalgová

Autoreferát dizertačnej práce

Indexing, Deployment and Searching Algorithms in Large Databases

Algoritmy indexovania, rozmiestňovania a vyvažovania v rozsiahlych databázach

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

v študijnom odbore:
informatika

Žilina, apríl 2022

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

Predkladateľ: **Ing. Veronika Šalgová**
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: **doc. Ing. William Steingartner, PhD.**
Katedra počítačov a informatiky
Fakulta elektrotechniky a informatiky
Technická univerzita v Košiciach

Oponent: **prof. Ing. Marcel Harakaľ, PhD.**
Katedra informatiky
Akadémia ozbrojených síl gen. M. R. Štefánika

Autoreferát bol rozoslaný dňa: .....

Obhajoba dizertačnej práce sa koná dňa ..... o ..... 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 dňa .....

prof. Ing. Karol Matiaško, PhD.
predseda 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

1 ÚVOD

Ukladanie a spravovanie dát je významnou časťou informačných technológií. Moderná éra informatizácie prináša explóziu informácií, ktoré je potrebné uchovávať a spracovávať. Relačné databázové systémy pokrývajú hlavnú časť súčasnej správy dát informačných technológií. Sú veľmi dôležitou súčasťou mnohých informačných systémov, od komerčných, cez technické a technologické systémy, webové a mobilné aplikácie až po správu vedeckých dát v rôznych oblastiach. Veľké množstvo dát si vyžaduje veľa úložného priestoru, preto sa do popredia dostávajú veľmi veľké databázy s obrovským množstvom dát. Dáta sú organizované v procese normalizácie databázy, čím sa znižuje redundancia a závislosť informácií. Tieto dáta sa následne použijú na vykonanie dotazu a získanie požadovaných výstupov.

Rýchly prístup k dátam je v dnešnej dobe čoraz dôležitejší a veľký dôraz sa kladie na jeho skvalitňovanie. Pri veľkom množstve dát však ich spracovanie zvyčajne trvá dlho, preto sa v našej dizertačnej práci zameriavame na analýzu a implementáciu rôznych rozšírení, ktoré zaisťujú vyšší výkon a rýchlejší prístup k dátam.

2 AKTUÁLNY STAV PROBLEMATIKY

Zložitosť a rozsah informačných systémov sa neustále zvyšuje. To vytvára väčšiu potrebu čo najjednoduchšej a najrýchlejšej manipulácie s informáciami. Je tiež potrebné, aby funkcie, ktoré poskytujú podporu a prístup k údajom, boli dostatočne efektívne, rýchle a robustné. Základnou technológiou, ktorá by mala zabezpečiť kvalitný chod celého systému, je databázová technológia. Ide o jednotný súbor pojmov, prostriedkov a techník používaných na vytváranie informačných systémov.

Hlavnú časť súčasnej správy dát informačných technológií pokrývajú relačné databázové systémy. Dáta sa formujú do relácií spojených pomocou vzťahov. Sú zastúpené v tabuľkách, v ktorých každý riadok predstavuje záznam s jedinečným ID. Každá n-tica je fyzicky uložená v databáze prevádzkovanej procesmi na pozadí inštancie [57]. Používateľský dotaz sa prenesie na server, analyzuje a spracuje. Dáta sa odosielajú späť používateľovi ako súbor výsledkov. Hlavnou časťou spracovania a vyhodnocovania dotazov je práve prístup k samotným údajom [26] [38].

Transakcia je hlavnou vlastnosťou relačnej paradigmy. Každá zmena dát (pomocou príkazov Insert, Update alebo Delete) je súčasťou transakcie, ktorú je možné následne akceptovať alebo odmietnuť. Pred tým, ako sú údaje verejne viditeľné, musia byť teda potvrdené [18]. Transakcie zabezpečujú komplexnú správnosť a spoľahlivosť údajov. Sú dôležité nielen pre zmeny údajov, ale aj pre dotazy typu Select. Z fyzického hľadiska sú dáta vždy prístupné zo štruktúry vyrovnávacej pamäte [52]. V optimistickom prípade môžu byť požadované údaje priamo dostupné, pričom sú prítomné v pamäti. Takto je zostavený súbor výsledkov a odoslaný používateľovi. Ak sa tam dáta nenachádzajú, alebo nejaká časť z nich chýba, musia sa načítať z fyzického databázového úložiska do pamäte, kde sa vykonajú ďalšie kroky spracovania. Fyzické úložisko databázy je ohraničené dátovými súbormi patriacimi do tabuľkových priestorov. Interne sa každý dátový súbor skladá z jednotlivých blokov rovnakej veľkosti. Počas spracovania sa celý blok prenesie do pamäte, kde sa vyhľadajú a lokalizujú príslušné dáta. Výber bloku s relevantnými údajmi je možné vykonať pomocou dvoch techník – sekvenčného skenovania alebo pomocou indexu [17] [37].

Relačná databáza ponúka komplexnosť, robustnosť a bezpečnosť dát uložených v nej. Základným prvkom, ktorý odlišuje databázu a súborový systém, je práve podpora transakcií.

Jeho správa zaisťuje spoľahlivosť dát, ktoré spĺňajú všetky požiadavky a obmedzenia. Teória transakcií definuje štyri aspekty práce s databázami – atomicitu, konzistentnosť, izoláciu a trvanlivosť. *Atomicita* zaisťuje, že dáta prevádzkované v rámci transakcií (pridanie alebo zmena existujúcich n-tíc) sú buď schválené or úplne zamietnuté. Transakcia sa teda považuje za jeden neoddeliteľný prvok, ktorý nemožno čiastočne riadiť a hodnotiť. *Konzistentnosť* sa zaoberá integritou tým, že kladie dôraz na obmedzenia, ktoré by mali byť schválené najneskôr do konca transakcie. Preto transakcia presúva databázu z jedného platného konzistentného obrazu na iný, taktiež konzistentný. Tretím aspektom je *izolácia*. Zmeny vykonané vo vnútri transakcie sú viditeľné hneď po jej schválení rozšírením operácií do celého prostredia. Nakoniec, *trvanlivosť,* na ktorú sa vzťahuje logovanie, zabezpečuje, že schválené zmeny transakcií sú viditeľné a budú v systéme prítomné aj po páde systému [9].

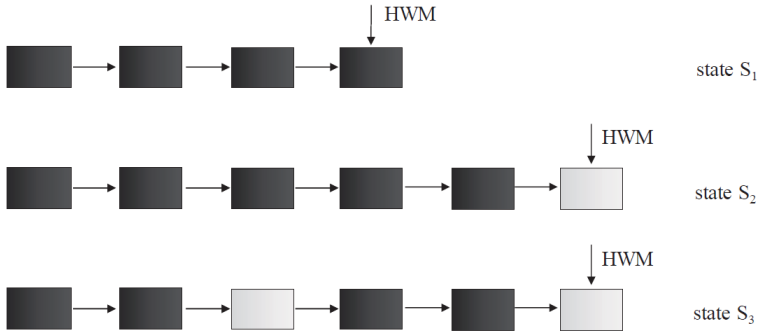
Spracovanie dotazu pozostáva z niekoľkých krokov, ktoré sa vykonávajú postupne. Výstup z jednotlivého kroku sa prenesie ako vstup nasledujúceho. *Parser* vykonáva *syntaktickú* (gramatika príkazov) a *sémantickú* (existencia objektu a prístupové práva) analýzu dotazu a prepisuje pôvodný plán na množinu operácií relačnej algebry. *Optimalizátor* navrhuje najefektívnejší spôsob získania výsledkov dotazov na základe optimalizačných metód, vytvorených indexov a zozbieraných štatistík. Vyberie teda najlepší (suboptimálny) plán vykonávania dotazu, ktorý sa použije v ďalšom kroku generátora zdrojov riadkov. Vytvára plán vykonania pre daný SQL dotaz vo forme stromu, ktorého vrcholy sú tvorené jednotlivými zdrojmi riadkov. Potom sa vykoná SQL dotaz s dôrazom na poskytnutý plán vykonania. Výsledný súbor je skonštruovaný a odoslaný klientovi [20] [35]. Najdôležitejším krokom z hľadiska efektivity spracovania, optimalizácie a pravidiel prístupu je práve *plán vykonania*, ktorý určuje využitie indexov [48].

3 MASTER INDEX

Obmedzenie použitia kategórie metódy *Full index scan* je založené na skutočnosti, že kontext vyjadrený podmienkami klauzuly *Where* možno vyhodnotiť priamo v listovej vrstve indexu. Aj keď poradie atribútov v indexe nezodpovedá dotazu, relevantné atribúty sú prítomné, avšak v nevhodnom poradí. V dôsledku toho, ak sa *ROWID* na listovej vrstve vyberie odovzdaním podmienky *Where* z dotazu, je isté, že záznam bude obsahovať dáta potrebné na vytvorenie sady výsledkov. Do vyrovnávacej pamäte sa teda nenačíta žiadny irelevantný dátový blok, okrem problému s migrovaným riadkom [5] [8].

Naše riešenie využíva iný princíp. V prípade prijatia sa v každom prípade použije definovaný index. Ak index založený na atribútoch nie je vhodný pre dotaz, použije sa len ako prístupová cesta k blokom údajov s reálnymi údajmi. Dôležitosť definície nášho riešenia je popísaná v nasledujúcom príklade. Majme štyri riadky s údajmi pre tabuľku. Pre jednoduchosť predpokladajme, že každý dátový riadok je na začiatku umiestnený v samostatnom dátovom bloku. Potom vložme dva nové riadky, ktoré sa budú nachádzať v rovnakom dátovom bloku. Bloky sú priradené k objektu vo forme jednotlivých rozsahov, nie bloky priamo. Predpokladajme teda, že rozsah obsahuje dva bloky. Po vykonaní tak bude použitých šesť blokov a posledný bude prázdny. Teraz v treťom kroku odstráňme údaje tretej n-tice. Tretí blok bude priradený k tabuľke, no bude úplne prázdny. Je zrejmé, že pre vyhodnotenie sú relevantné iba štyri dátové bloky, len obsahujú rovnaké dátové časti. Podľa indexu sú prístupné cez hodnoty *ROWID* v indexe. Ak však v tomto prípade index neobsahuje atribúty charakterizujúce podmienku dotazu, použije sa metóda prístupu *Table Access Full (TAF)*. Metóda TAF bohužiaľ nemá žiadne informácie o prázdnych blokoch spojených s tabuľkou, nedochádza k defragmentácii ani migrácii údajov z dôvodu vplyvu na výkon – takáto tabuľka by bola počas takéhoto procesu neprístupná, čo je neprijateľné. Navyše, v súčasnosti je počet

Update príkazov vysoký a stále narastá, takže konsolidácia dát by si vyžadovala príliš časté vykonávanie, aby sa zabezpečila výhoda, ale je príliš náročná na zdroje. Celkovo by zlepšenie bolo minimálne, ak by vôbec nastalo. Použitím *TAF* v opísanej situácii by sa teda do pamäte načítalo šesť blokov, z ktorých však dva neposkytujú žiadne dáta. Globálna efektivita by bola 4/6 – len o niečo viac ako 66 %. Iste, ide len o jednoduchú demonštráciu problému, v reálnom prostredí by sa dosahoval výkon výrazne pod 50 %, takže viac ako polovica systémovej práce by bola zbytočná na vyhodnotenie a spracovanie. To je, samozrejme, obrovský problém z hľadiska výkonu a rastu požiadaviek na dáta a zložitosti. Jednotlivé kroky spracovania sú znázornené na obrázku 1.



Obr. 1: Modelovanie dátových blokov

Cieľom nami navrhovaného riešenia je obmedziť potrebu používania sekvenčného skenovania dátových blokov vykonávaného metódou TAF. Naša technológia využíva Master index, ktorý sa nepoužíva na samotné vyhodnotenie, pričom nespĺňa podmienky dopytu. Základom je, že obsahuje všetky ukazovatele na dáta na listovej vrstve. Preto sa ako lokátor údajov používa samotný index. Existujú dva navrhované modely, ktoré zdôrazňujú definíciu *Master indexu*. Prvý je založený na granularite riadkov dát a druhý je posunutý do blokovej identifikácie a používa dva indexy. Na základe dosiahnutých výsledkov najlepšie riešenie odráža granularitu bloku. V tomto prípade, v porovnaní s pôvodnou metódou *TAF*, bola výkonnosť v čase spracovania znížená na 13%, ak by sa v súbore výsledkov mala poskytnúť jedna desatina údajov. Princíp spočíva v odstránení vyhodnocovania voľných blokov, ktoré je potrebné štandardným spôsobom preniesť do pamäte. Naopak, nároky na veľkosť vzrástli približne o 12 %. Je to spôsobené potrebou vyvinúť nový index na blokovej granularite pre dotazy.

4 ZNÍŽENIE ČASU PRÍSTUPU K DÁTAM POMOCOU PARTÍCIÍ

Efektívnosť prístupu k údajom je jednou z najdôležitejších úloh pri zabezpečovaní výkonu systému. Množstvo dát neustále narastá, a preto je potrebné tieto dáta nejakým spôsobom deliť. Pri využívaní veľkého množstva dát sa kladie veľký dôraz na prístupový čas. Z tohto dôvodu môže vytváranie partícií a následné rozdelenie dát do nich priniesť výrazné zlepšenie času prístupu k dátam.

V našich experimentoch sa porovnávali prístupové časy pre dáta v tabuľkách s vytvorenými partíciami a tabuľkami bez partícií. Výsledky výrazne ukázali, že prístup k dátam uloženým v partíciách môže výrazne skrátiť čas prístupu k dátam a priniesť tak vyššiu efektivitu. Použitie techniky vytvárania partícií poskytlo výrazne zlepšený prístupový čas, keď sa pristúpilo k menej ako 30 z 34 oblastí. Pri prístupe k 30 a viac partíciám sa ako

vhodnejšie ukázalo ukladanie dát bez partícií. Pri použití metódy *List partitioning* došlo k ešte výraznejšiemu zlepšeniu času výkonu a toto zlepšenie pretrvalo aj pri prístupe k viacerým alebo všetkým partíciám. Na záver možno z výsledkov vyhodnotiť, že rozdelenie na partície môže výrazne skrátiť čas prístupu k dátam.

5 VPLYV PARTÍCIÍ A INDEXOVANIA

Vytváranie indexov, partícií a ich rôznych kombinácií môže priniesť výrazné zlepšenie času prístupu k údajom v mnohých situáciách. Zaoberali sme sa vplyvom partícií a indexovania na čas prístupu k údajom, ktorý bol porovnávaný v siedmich rôznych scenároch rôznych kombinácií partícií vytvorených nad tabuľkami a indexmi.

V experimentoch sa porovnávali prístupové časy pre dáta v tabuľke s vytvorenými partíciami a tabuľke bez akýchkoľvek partícií. Testovali sme niekoľko situácií, ktoré sa líšili typom indexov vytvorených v oboch tabuľkách, ako napríklad indexy neparticiované, globálne particiované, lokálne particiované s prefixom a lokálne particiované bez prefixu.

Výsledky ukázali, že čas prístupu k údajom sa v rôznych scenároch značne líšil. Pre prvé rozsahy vybraných údajov bol najhorší čas prístupu spôsobený použitím neparticiovanej tabuľky bez indexov. Tu bol prístupový čas na začiatku asi 9-13krát horší ako vo zvyšných scenároch. Približne od polovice rozsahov, ku ktorým sa pristupovalo, však došlo k výraznému spomaleniu času prístupu v scenári s particiovanou tabuľkou bez indexov a začalo to byť približne 10-krát pomalšie ako scenár s neparticiovanou tabuľkou bez indexov.

Všetky scenáre, v ktorých sa použili partície, indexovania, alebo ich kombinácie, sa začali približne v rovnakom čase prístupu k dátam, a to 4, 5 a 6 milisekúnd. Čo sa týka prístupu k údajom patriacim do prvých 11 z 25 partícií, najlepšie výsledky mal scenár s particiovanou tabuľkou bez indexov. Po tejto hranici pri 11 partíciách sa však stal výrazne najpomalším scenárom.

Z výsledkov experimentov je možné vydedukovať, že vytváranie a použitie partícií, indexov alebo ich kombinácií môže výrazne urýchliť čas prístupu k dátam. Pri častom prístupe k veľkému množstvu údajov, ku ktorým sa viažu partícií alebo vrcholy indexu, je však čas prístupu podobný ako v scenári bez partícií a indexov, alebo v niektorých situáciách môže byť dokonca oveľa horší.

Preto je dôležité zvážiť vhodný výber typov partícií a indexov v závislosti od frekvencie a objemu údajov, ku ktorým sa pristupuje.

6 VPLYV INDEXOV NA DML OPERÁCIE

Zaoberali sme sa vplyvom počtu indexov na výkon operácií DML, ako je Insert, Update a Delete. Indexy sú veľmi užitočné na urýchlenie času prístupu k dátam. V každej situácii pri vkladaní, aktualizácii a vymazávaní údajov sme vykonali sériu týchto operácií. V každom kroku bolo ovplyvnených približne jeden milión riadkov. Pre každú operáciu bolo testovaných šesť situácií. Rozdiel medzi situáciami bol v počte vytvorených indexov vzťahujúcich sa k upraveným údajom. Pridaním nového záznamu do tabuľky sa nový uzol pridá aj k príslušným indexom. Potom sa vykoná opätovné vyváženie indexu stromovej štruktúry. Na základe výsledkov je zrejmé, že vytváranie indexov súvisiacich s vkladanými údajmi má pomerne významný vplyv na čas vykonania príkazu Insert. Kým v situácii bez indexov trvalo vkladanie riadkov 85,09 sekúnd, po vytvorení jedného indexu sa čas predĺžil na 92,43 sekúnd a pri 5 indexoch ovplyvnených vloženými údajmi to trvalo dokonca až

218,58 sekúnd. Nárast času potrebného na aktualizáciu údajov v prípade absencie indexov v porovnaní s prípadom s piatimi indexmi je podstatne väčší ako pri vkladaní údajov. Kým v situácii bez indexov trvala aktualizácia údajov 17,03 sekundy, pri 5 indexoch to bolo až 72,10 sekundy. Príkaz Update musí premiestniť príslušné uzly indexu, aby sa zachovalo poradie indexu. Na tento účel je potrebné odstrániť starý záznam a pridať nový na nové miesto Štruktúra musí tiež zostať vyvážená. Čas potrebný na vykonanie operácií vymazania sa zvýšil v podobnom rozsahu ako v prípade aktualizácie údajov. Pri odstraňovaní riadku index vymaže referenciu na riadok a štruktúru indexu stromu je potrebné vyvážiť, aby sa zachovala potrebná vlastnosť B-stromu, resp. B+ stromu. Indexy sú veľmi užitočné na zrýchlenie času prístupu k údajom. Z výsledných časov našich experimentov však možno usúdiť, že počet indexov má skutočne významný vplyv na zvýšenie času na vykonanie operácií vkladania, aktualizácie a odstraňovania. Je preto veľmi dôležité stanoviť si pri práci s údajmi primeraný počet indexov. Väčší počet indexov je vhodný len pre dáta, ktoré sa často nemenia a slúžia len na vyhľadávanie. V situáciách s častými a veľkými zmenami údajov je lepšie použiť menší počet zodpovedajúcich indexov.

7 VPLYV KOMPRESIE TABUĽKY A INDEXU

Množstvo uložených údajov rýchlo rastie a prináša to značné výzvy. Enormný rast objemu dát robí z úložiska jednu z najväčších nákladových položiek. Pre tento účel sa veľmi často používajú relačné databázy. Rýchly prístup k dátam je čoraz dôležitejší a veľký dôraz sa kladie na jeho zlepšovanie. Mnoho počítačových systémov v súčasnosti používa kompresiu. Používa sa pre zvukové a obrazové údaje v multimediálnych systémoch, na zálohovanie, komprimovanie invertovaných indexov pri získavaní informácií, odosielanie súborov cez internet a ukladanie veľkých súborov a softvérových balíkov. Kompresia údajov je široko používaná pri správe údajov, aby sa šetril úložný priestor a šírka pásma siete a znížili sa náklady na úložné médiá. Mnoho algoritmov na spracovanie dotazov dokáže manipulovať s komprimovanými údajmi rovnako dobre ako s dekomprimovanými údajmi a spracovanie komprimovaných údajov môže dokonca urýchliť spracovanie dotazov oveľa väčším faktorom, než je faktor kompresie. Výkon databázy silne závisí od množstva dostupnej pamäte. To znamená, že všetka dostupná pamäť by mala byť využívaná čo najefektívnejšie a na uchovávanie a manipuláciu s údajmi v pamäti v komprimovanej forme [12].

Zaoberali sme sa vplyvom kompresie tabuliek a indexov na čas prístupu k dátam a náklady na CPU. Porovnávalo sa v ôsmich rôznych scenároch, ktoré sa líšili stavom kompresie tabuľky, prípadne indexu vytvoreného nad tabuľkou a tiež počtom riadkov tabuľky. Atribúty tabuliek obsahovali nejedinečné hodnoty, a teda tabuľka mala nízku mohutnosť, efekt kompresie bol väčší ako v predchádzajúcich experimentoch s jedinečnými hodnotami. Pozitívny efekt kompresie bol viditeľný v niektorých situáciách, najmä pri použití kompresie tabuľky s 1 miliónom riadkov. Okrem časov prístupu k dátam bol vplyv ešte viditeľnejší pri porovnaní nákladov na CPU. Tento efekt sa však menil v závislosti od počtu riadkov v tabuľkách a kompresie indexu.

8 KOMPLEXNOSŤ PROCESU ZÍSKAVANIA DÁT POMOCOU ROZŠÍRENIA INDEXU

Početné prístupové metódy boli zavedené a diskutované v priebehu desaťročí. Identifikáciu a umiestnenie údajov je možné vykonať sekvenčným skenovaním alebo

pomocou indexov. Rozhodovanie vykonáva databázový optimalizátor, ktorý ukazuje na aktuálne štatistiky. Neaktualizujú sa však automaticky, ale ich obnovovacie operácie sa plánujú [42]. Keď je prítomný robustný dátový tok a dáta sa rýchlo vyvíjajú, ani aktuálne dokončené obnovenie už nie je relevantné. Môže to mať za následok nesprávne rozhodovanie, pričom vstupné údaje sú nesprávne. Správa optimalizátora je založená na heuristike, ktorá ovplyvňuje výber metód prístupu. Preto je potrebné zamerať sa na viacero aspektov týkajúcich sa plánu realizácie. Navrhujeme nové techniky na zabezpečenie aktuálnych štatistík. Ak sa výrazne zmení vzor údajov alebo množstvo, skontroluje sa niekoľko plánov vykonávania. Vo všeobecnosti sa používa už vypočítaný plán, ak je priamo dostupný. V našom navrhovanom riešení je však definícia rozšírená o zhrnutie dátovej štruktúry, množstva dát a perspektívy reflexie. Vďaka tomu môže databázový systém autonómne vyhodnocovať existujúci SQL plán s dôrazom na aktuálny dátový obraz. Aj keď indexy tvoria robustné riešenia na zabezpečenie výkonu, možno identifikovať rôzne toky vylepšení. Pri navrhovaní našich techník sa zameriavame na nasledujúce segmenty:

- identifikácia a reflexia migrovaných riadkov,
- automatické vyváženie indexu,
- hodnotenie a zváženie efektívnosti indexovej štruktúry,
- riadenie indexu mimo hlavnej transakcie,
- kontrola nedefinovaných hodnôt (NULL),
- identifikácia relevantného bloku dát,
- optimalizácia dynamického plánu vykonania.

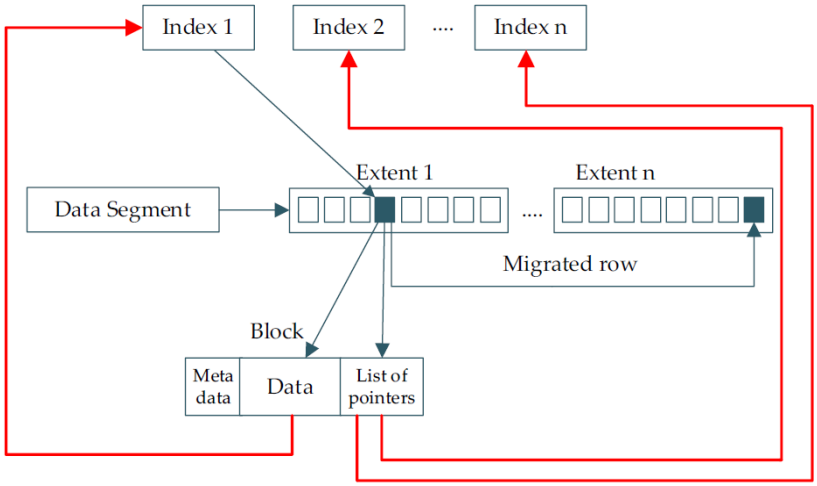
Riešenie je založené na indexe B+ stromu ako základnom prvku, ktorý je rozšírený o rôzne ďalšie dátové štruktúry a navrhované procesy na pozadí na jeho spracovanie. Dosiahnutím komplexnosti navrhovanej architektúry je možné identifikovať významné zlepšenia výkonu.

8.1 IDENTIFIKÁCIA A REFLEXIA MIGRÁCIE RIADKOV

Prepojenie medzi pamäťou a databázou je tabuľkovým priestorom, ktorý vymedzuje veľkosť bloku. Migrované riadky sa vytvoria, ak sa pôvodný záznam po zmene už nezmestí do pôvodného bloku. Systém fyzicky vyhľadá nové úložisko tak, že nájde blok, ktorý dokáže spracovať konkrétny riadok. Ak nie je k dispozícii žiadne miesto, pridelí sa nový rozsah (sada blokov). V zásade teda nie je problém nájsť nové miesto. Môže to však ovplyvniť indexy. Neexistuje žiadny konkrétny ukazovateľ opačného smeru - definícia z bloku na príslušné indexy. Na použitie migrovaného riadku by bolo potrebné oskenovať všetky indexy, čo je reprezentované pridaním nového ukazovateľa z pôvodného do nového úložiska. Výsledkom je, že na nájdenie riadku pomocou indexu je potrebné načítať viacero blokov. Vo všeobecnosti nemusí ísť len o dva bloky – pôvodné a nové blokové úložisko, ale štruktúra môže byť väčšia, v závislosti od celkovej prevádzkyschopnosti tabuľky. Migrácia jedného riadku sa teda môže rozložiť na viacero blokov, všetky v reťazci sa musia načítať postupne a nakoniec je potrebný len posledný. Ak je teda frekvencia operácií Update a Delete vysoká, výkon sa môže v priebehu času výrazne zhoršiť. Momentálne sa to dá vyriešiť len úplným prebudovaním indexu. Vzhľadom na vstupný tok (prevádzkovaný rôznymi operáciami manipulácie s dátami) a evolúciu dát takýto prístup nie je vhodný a je potrebné aplikovať a obmedzovať migrácie dynamicky. Aby sme analyzovali dopad a vyhodnocovali prínosy, navrhujeme a diskutujeme o rôznych riešeniach. Ako je zrejmé, základným prvkom je identifikácia migrovaných riadkov a odstránenie ďalších ukazovateľov, aby sa zabezpečilo, že

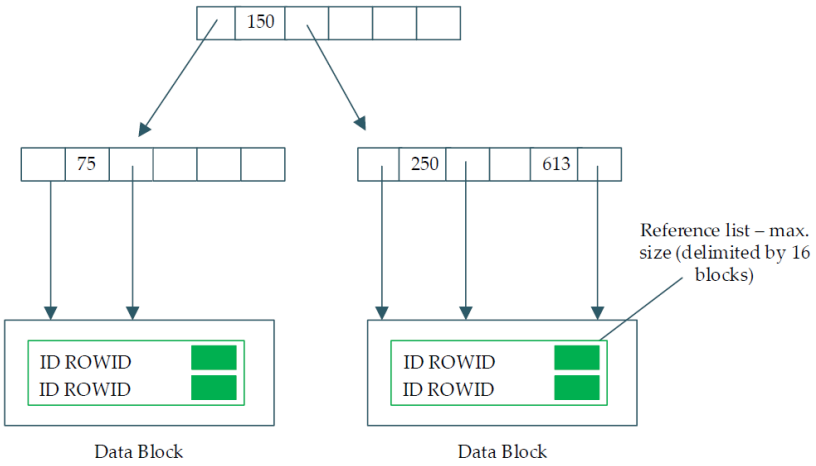
existujúca množina indexov vždy odráža aktuálne bloky. Z architektonického hľadiska sú navrhnuté tri riešenia.

Prvé riešenie, ktoré obmedzuje vplyv migrácií, je založené na rozšírení štruktúry dátových blokov. Pre každý riadok údajov je uvedený zoznam ukazovateľov na množinu indexov. Vďaka tomu je ľahké nájsť ľubovoľný index, ukazuje na konkrétny listový blok, ktorý obsahuje referenciu ROWID. Pôvodná hodnota ROWID je teda nahradená novou adresou, na ktorej sa údaje aktuálne nachádzajú. Z fyzického hľadiska je pre každý riadok alokované nové dynamické pole vo vnútri hlavného bloku. Ak sa vytvorí nový index, na odkaz na index sa musí použiť nový prvok poľa.



Obr. 2: Architektúra riešenia 1 – zoznam štruktúry ukazovateľov

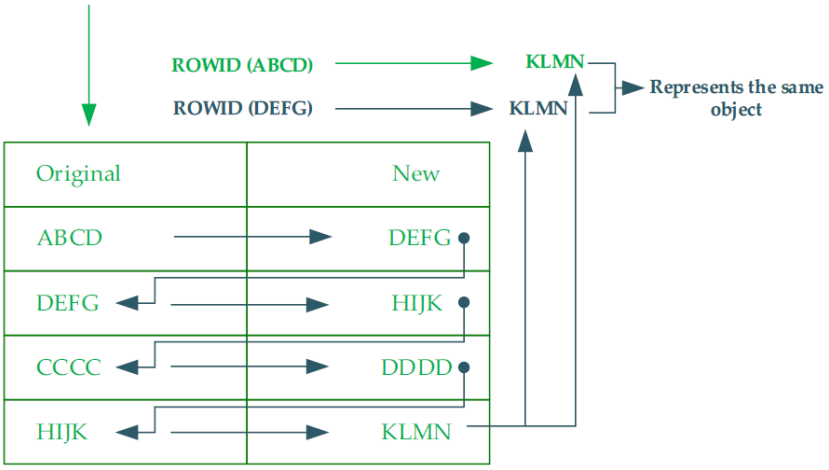
Princípy druhého riešenia sú rovnaké, ale pole adresy je extrahované do samostatnej dátovej štruktúry, tiež blokovo orientovanej. Tieto polia adries majú stromovú štruktúru, takže vyhľadávanie môže byť efektívnejšie. Každý prvok má identifikátor objektu rozšírený o polohu fyzických údajov (ROWID) a zoznam odkazov na indexy. Takýto zoznam odkazov môže byť uložený buď priamo v indexe (*riešenie 2a*) alebo mimo v samostatnom poli blokov (*riešenie 2b*). Výhoda *riešenia 2a* je založená na predpoklade, že akýkoľvek vrchol a konkrétny referenčný zoznam sú v tom istom bloku. Implementácia však nie je robustná z hľadiska pridávania alebo odstraňovania existujúcich indexov (vykonáva sa napríklad automatickým indexovaním). Naopak, rozšírenie súboru indexov alebo reflexia existujúcich prístupov k indexom sú prospešné, ak je zoznam referencií uložený samostatne. V takom prípade je možné kedykoľvek rozšíriť konkrétne referenčné pole. Počet blokov, ktoré sa majú načítať, sa však zvýši o jeden pre ľubovoľný riadok údajov. Naopak, rozšírenie zoznamu referencií sa môže nachádzať v štruktúre blokov pretečenia, ktoré sú spojené s každým zoznamom referencií vrcholov, v prípade naplnenia bloku.



Obr. 3: Architektúra riešenia 2a – vložený zoznam referencií

8.2 APLIKÁCIA POST-MIGRÁCIE

Ako už bolo uvedené, problém migrácie riadkov môže byť rozšírený do rôznych blokov. Na obmedzenie vplyvu na odkaz iba na dva bloky je možné vytvoriť špecifickú štruktúru na uchovávanie migrovaných riadkov. Ak sa má migrácia vytvoriť, v štruktúre *Migration_mapper* sa uloží konkrétny pôvodný a konečný blok blokov. Dá sa priradiť k ľubovoľnej tabuľke a zaisťuje, že migrácia je relevantná len pre dva bloky a vôbec sa nedá rozširovať. Riešenie je založené na identifikácii migrácie dát na databázovom bloku. Namiesto použitia priameho ukazovateľa na ďalší blok sa optimalizátor prístupu k databáze pozrie na *Migration_mapper* a zistí úložisko poslednej fázy. Aj keď je potrebné načítať dodatočný blok, modul *Migration_mapper* je zvyčajne malý a možno ho umiestniť priamo do existujúcej vyrovnávacej pamäte inštancie databázy. Nevýhodu riešenia predstavuje rozšírenie I/O prevádzky. Požiadavky na veľkosť štruktúry *Migration_mapper* nie je možné znížiť, kým sa nevykoná operácia opätovného zostavenia celej sady indexov. Znova je potrebné zrekonštruovať všetky indexy pre konkrétnu tabuľku, aby sa skrátila funkčnosť štruktúry *Migration_mapper*. V opačnom prípade musí existovať originál, aby mohol slúžiť zvýšku prístupu do indexu. Riešenie mapovača migrácie je znázornené na obrázku 4. Pre konkrétny riadok môže existovať niekoľko úrovní migrácie. Každý index môže odkazovať na iný koreňový vrchol nástroja *Migration_mapper*. Referenčný model teda nemožno priamo kaskádovať.



Obr. 4: Architektúra riešenia 3 – Mapovač migrácie

8.3 UKLADANIE CESTY

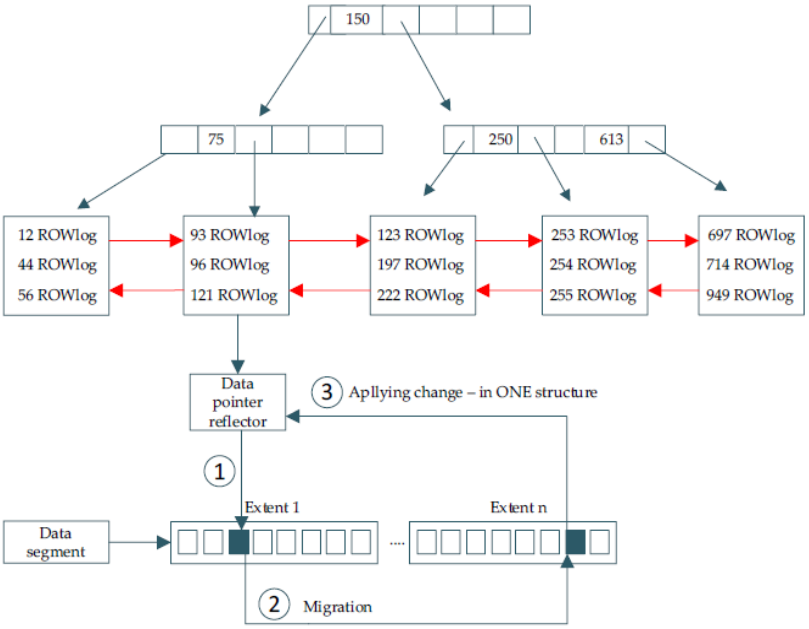
Štvrté riešenie pokrýva ďalší koncept. Aj keď je spojenie medzi indexom a údajmi len jednosmerné, počas prístupu pomocou definovaného indexu môže byť prechodová cesta dočasne uložená, takže ak je zistená migrácia údajov, podľa takejto definície môže byť ukazovateľ indexu (ROWID) aktualizované tak, aby pokrývali skutočný dátový blok, kde sa dáta nachádzajú. Na jeho implementáciu je navrhnutá dátová štruktúra *Data path reflector memory*. Je teda spojená s reláciou. Nie je zdieľaná medzi inštanciami – neprinieslo by to žiadnu výhodu, pričom operácie zmeny na konkrétnom riadku sa vždy vykonávajú vo výhradnom režime – iba jedna transakcia môže kedykoľvek zmeniť konkrétny riadok. Štruktúra *data path reflector* je preto spojená s transakciou. Po dosiahnutí príkazu na jeho ukončenie (commit/abort) je možné takúto štruktúru vyprázdniť. Vnútorne je implementovaný pomocou vrstvy ukazovateľov pre každú operáciu aktualizácie údajov. Ak sa deteguje migrovaný riadok, štruktúra reflektora údajov ovládaná procesom na pozadí Session-Reflector bude upozornená, aby použila zmenu.

Toto navrhované riešenie však môže migrované riadky obmedziť len čiastočne. Poskytnutý dátový reflektor je prepojený len s už použitou prístupovou cestou a nereflektuje ostatné indexy na konkrétnej dátovej tabuľke. Naopak, aplikuje zmenu len na jeden prvok indexu. V dôsledku toho je migrovaný riadok fragmentovaný. Pôvodný blok musí vždy uchovávať ďalší ukazovateľ bloku, aby sa zabezpečila spoľahlivosť údajov a bezpečnosť na úrovni riadkov indexu.

8.4 REFLEXIA DÁTOVÉHO UKAZOVATEĽA

Posledné navrhované riešenie v tejto kategórii nahrádza cestu priamymi ukazovateľmi. Princípy sú podobné ako pri riešení 4, rozšírené o proces na pozadí štruktúry *Index Submapper* spravujúcej infraštruktúru ukazovateľov. Z architektonického hľadiska indexy neobsahujú fyzické databázové adresy (ROWID). Namiesto toho sa používajú logické adresy k reflektoru ukazovateľa dát (ROWlog). Reflektor ukazovateľa dát je štruktúra Index Submapper, ktorá transformuje adresu logického riadka na fyzický ukazovateľ do databázy. Výhoda takéhoto prístupu je založená na jedinečnosti. Vytvorenie migrovaného riadku sa neprejaví v jednotlivých indexoch, zatiaľ čo logické ukazovatele sa nevyvíjajú – iba definujú zdroj pre reflektor ukazovateľa údajov. Takže ukazovateľ adresy fyzických údajov je v databáze uložený vždy len raz, bez ohľadu na počet definovaných indexov. S reflektorom údajového ukazovateľa sa zaobchádza ako s inline štruktúrou B+ stromového indexu na základe logickej adresy mapujúcej každú hodnotu na jeden fyzický ROWID. Ak je teda migrovaný rad vytvorený, musí byť v štruktúre reflektora aplikovaný iba raz. Hľadanie konkrétnej hodnoty je možné vykonať buď vnútornou štruktúrou B+ stromového indexu tvarovanou reflektorom (riešenie 5a) alebo je možné použiť dráhu prechodu (definovaná v riešení 4), čím sa vytvorí riešenie 5b. Traverzová cesta je uložená v oblasti špecifickej pre reláciu a je platná počas konkrétnej transakcie.

Riešenie 5b poskytuje výrazne lepší výkon, zatiaľ čo definícia dráhy prechodu je jednoduchší a najrýchlejší prístup k odkazovaniu na index. V tomto prípade posledný vrchol prechodovej cesty extrahuje adresy logických riadkov na aktualizáciu fyzického ROWID v reflektore. V dôsledku toho je migrovaný riadok len čiastkovým prvkom spracovania v rámci transakcie. Preto po potvrdení nemôžu existovať žiadne migrované riadky pre spravované údaje. Odstránenie obmedzenia migrovaného riadka sa vykoná najneskôr na konci transakcie. Dá sa teda premietnuť hneď po samotnej zmene dát alebo posunúť na koniec transakcie, kde je možné aplikovať viacero migrácií v spoločnom bloku.



Obr. 5: Architektúra riešenia 5 – Reflektor ukazovateľa dát

8.5 PRIORITNÝ MANAŽMENT

Hlavnou vlastnosťou indexu B+ stromu ako predvoleného prístupu je vyváženie, ktoré na jednej strane zabezpečuje výkon, pričom cesta od koreňa k akémukoľvek listu je vždy rovnaká. Na druhej strane jednotlivé operácie zmeny údajov musia zabezpečiť vyváženie pridaním dodatočných požiadaviek. B+ stromy sa časom nedegradujú a zaisťujú efektívnosť s rastom údajov rozdeľovaním a spájaním blokov. Prirodzene, s narastajúcim objemom dát sa index stáva čoraz komplexnejším. V zásade nie je potrebné pristupovať k údajom rovnomerne. Pri práci s časovými prvkami sa najčastejšie získavajú aktuálne platné dáta tvoriace konvenčný obraz. V priebehu času historické údaje strácajú svoj význam a dopytujú sa čoraz menej. Indexové prístupy nereflektujú takéto charakteristiky prostredia a zabezpečujú rovnakú prioritu spracovania. V našom výskume sa dôraz kladie aj na aspekt priority údajov v rámci indexu. V takom prípade nedochádza k striktnému vyváženiu kľúčov. Namiesto toho sa používa stratégia priorít, aby sa zabezpečilo, že najčastejšie používané údaje možno získať ešte výrazne lepšie.

Pre každý riadok údajov registrovanej tabuľky je index rozšírený o vyhodnocovanie priority, vypočítanej podľa frekvencie prístupu. Spracúva sa v granularite indexu, pričom samotná n-tica sa skladá z atribútov s rôznou frekvenciou, presnosťou a trvanlivosťou príkazov Update a Select. Každý indexový uzol je rozšírený o atribút Access Ticker, ktorý pokrýva rozsah prístupu.. Pre zabezpečenie riešenia je každá hodnota priradená nielen k dátovému vrcholu, ale zahŕňa aj použitú metódu prístupu k indexu. Nakoniec sú tieto dátové moduly časovo orientované, čo odráža využitie v čase. Vďaka tomu je možné vyvažovanie indexu vykonávať dynamicky, napríklad na konci mesiaca je možné vykonávať komplexné analýzy a reporty na základe mesačných podrobností údajov, takže index možno znova vyvážiť tak, aby slúžil procesu a zameral sa na pokryté údaje. Okrem toho je možné celý proces vykonávať dynamicky pomocou zoznamu odhadovaných aktivít.

Použiteľný index je vždy vyvážený a výkonovo optimalizovaný aplikáciou zmien priamo na index v rámci transakcie. Je zabezpečené, že samotná transakcia môže byť schválená hneď po správe údajov a indexu. Prístup k indexu je teda dôveryhodný. Súčasťou

indexu je akákoľvek časť údajov, ktorá odráža platný riadok údajov. Výsledkom je, že ak optimalizátor používa prístupovú cestu indexu, množstvo údajov, ktoré sa väčšinou týka čísla bloku, je výrazne obmedzené. Preto môže proces získavania údajov výrazne profitovať z používania indexu.

Na druhej strane ostatné operácie manipulujúce s údajmi musia aplikovať všetky zmeny na celú sadu indexov. Ak je množina indexov silná, modifikácia všetkých relevantných indexov môže rýchlo predĺžiť čas spracovania vnútornej transakcie. Pri vyhodnocovaní spracovania správy indexov počas zmeny možno pre každý index identifikovať tri operácie. Najprv je potrebné extrahovať údaje, ktoré sa majú indexovať. Potom, v druhej fáze, sú podrobné údaje smerované do indexu, čo ho núti prideliť nový prvok indexu. Takáto aktivita sa umiestni na dráhu prechodu a vrstve listu sa priradí nový uzol. Nakoniec, za predpokladu, že sa použije index B+ stromu, musí sa vykonať operácia vyváženia indexu.

V zásade môže každý riadok údajov vyžadovať vyvažovanie, ktoré možno vykonať samostatne pre každý riadok (predvolený prístup), alebo operáciu vyvažovania možno vykonať raz na úrovni granularity transakcie (append hint). Tak či onak, vyvažovanie indexov reštrukturalizuje index pomocou zámkov, aby sa zabezpečilo, že proces môže byť vykonaný bezpečne a úplne. Ostatné transakcie a operácie získavania musia visieť, aby sa zabezpečila správnosť. Ak je dátový tok vysoký, významná časť riadenia transakcií priamo súvisí s vyvážením indexu.

Nami navrhované riešenie predstavuje post-transakčnú vrstvu spojenú s indexom. Údaje sa neindexujú priamo v hlavnej transakcii. Takáto aktivita je rozdelená na dve časti. Zoznam zmien je extrahovaný z transakčných protokolov počas spracovania dát upozornením na zavedený proces na pozadí – *Index Applier*. Vo všeobecnosti údaje nie sú priamo zahrnuté v indexe. Pre každý index sa používa práve špecifická plochá dátová štruktúra – *Data Operator Module*. Ak je tam umiestnená vektorová informácia o zmene údajov, transakcia môže byť schválená a úspešne ukončená. Vďaka tomu sa skracuje čas spracovania transakcie a nedochádza k vyvažovaniu indexov. Ak sa v module *Data Operator Module* nachádzajú nejaké údaje, aktivuje sa proces na pozadí nástroja *Index Balancer*. Je to hlavný proces zodpovedný za aplikovanie zmien na príslušný index, po ktorom nasleduje vyvažovanie. Vykonáva sa však oddelene od hlavnej transakcie. Ak je zmena implementovaná a index vyvážený, definícia sa z priradeného modulu odstráni. Ako už bolo uvedené, nástroj na vyvažovanie indexu je hlavným zodpovedným procesom vytvoreným na požiadanie pre každý index. Je prítomný počas celej doby platnosti indexu ako supervízor. Inštancia má tiež rôzne pracovné procesy – *Index Balancer Worker(n),* kde „n“ predstavuje sériové číslo. Tieto procesy však nie sú priradení ku konkrétnemu hlavnému procesu, sú len zdieľaní v rámci inštancie.

Navrhovaná architektúra post-indexovania môže mať rôzne výhody. Hlavná transakcia môže byť ukončená skôr, pričom nie je prítomné žiadne vyvažovanie indexu. Vyvažovanie sa vykonáva oddelene, prevádzkuje sa pomocou pridaných procesov. Na rozdiel od existujúcich riešení, kde sa vyvažovanie vykonáva na úrovni riadkov alebo príkazov, definované riešenie využíva granularitu transakcií. Vyvažovanie môže byť rovnomerne rozdelené a môže pokryť viacero transakcií v jednej operácii. Vďaka tomu sa dá stratégia vyvažovania optimalizovať a globálne skrátiť. V jednom procese sa súčasne aplikuje viac operácií, čím sa zníži aj počet vyvažovacích operácií. Pri získavaní údajov je možné vo všeobecnosti použiť indexový prístup, ale musí sa rozšíriť o skenovanie modulu operátora údajov, ktoré sa vykonáva paralelne.

8.6 VYLEPŠENIA ARCHITEKTÚRY

V súčasnosti je najčastejšie používaným typom B+ strom založený na vyvážení indexových kľúčov. Matematické operácie sa aplikujú počas prechádzania indexu, aby sa zabezpečila správna prístupová cesta. Nedefinované hodnoty modelované pomocou NULL reprezentácie nie je možné matematicky umiestniť a porovnať. Výsledkom je, že nedefinované hodnoty nie sú súčasťou indexu. V [47] bolo navrhnutých niekoľko vylepšení na zabezpečenie pokrytia hodnoty NULL. Prvé kategorické riešenie je založené na ukladaní nedefinovaných hodnôt v plochej štruktúre združenej buď ľavou alebo pravou časťou, alebo sú nedefinované n-ticové adresy umiestnené v samostatnej štruktúre priamo prepojenej s koreňovým indexovým prvkom. Druhý prístup [62] využíva kategórie zameriavajúce sa na pôvod nedefinovanosti, ktorý je kontrolovaný spoľahlivosťou transakcie. V takom prípade sú jednotlivé ukazovatele rozdelené do segmentov typov na základe zaregistrovaných modifikátorov. Štruktúra je teda súčasťou indexového segmentu fyzicky uloženého v databáze.

Používame inú perspektívu odrážajúcu pamäť. Namiesto použitia indexu ako fyzického databázového segmentu navrhované riešenie nájde úložisko stĺpcov v pamäti. S jednotlivými atribútmi zaobchádza oddelene, a to aj pri zložených indexoch. Štruktúra uloženia stĺpca atribútov je umiestnená v pamäti v tvare B+ stromu, pričom rozširujúci modul pre správu hodnôt NULL je uložený oddelene. Procesy na pozadí a pamäťové štruktúry predstavujú inštanciu databázy. Pamäťová vyrovnávacia pamäť je hlavným úložiskom pre dátové bloky a indexy, ktoré sa majú použiť počas procesu získavania dát. Zavedený indexer pamäte je úložisko stĺpcov, ktoré priraďuje nedefinované údaje do kategórií v segmentoch správy NULL. Celá štruktúra je ovládaná procesom registrácie pamäte, ktorý je zodpovedný za vytvorenie a údržbu pamäťového modulu indexera pamäte.

V priebehu desaťročí sa zmenili štruktúry správy údajov. Oblasťou, ktorá však pretrváva dodnes, je technológia relačných databáz. Hlavnou výhodou je striktná definícia modelu pod dohľadom pravidiel integrity a konzistencie dát. Pre zabezpečenie komplexnosti dát je dôležité zaviazať transakciu týkajúcu sa podnikateľského prostredia a vnútorných požiadaviek. Je však potrebné klásť dôraz na interné databázové pravidlá, aby bola transakcia potvrdená čo najskôr, minimalizovali sa časové náklady a zmeny boli viditeľné pre ostatné systémy a relácie. Takáto požiadavka je však obmedzená vyvinutým indexom nastaveným pre konkrétnu tabuľku. Jednou silnou podmienkou je teda zabezpečenie rýchlej dostupnosti údajov pre indexy. Iný pohľad však súvisí s procesom získavania údajov, vyhodnocovania a ukladania nových n-tíc do databázy.

Správu indexov sme riešili rozšírením štruktúry. Bežným typom indexu je B+ strom, ktorý je vždy vyvážený, čo núti manažéra transakcie zabezpečiť takýto proces priamo vo vnútri transakcie. Nami navrhované riešenie vylučuje takéto operácie pri oddelení transakcií pomocou dátových indexátorov. Vďaka tomu môže byť pôvodná transakcia schválená skôr, vyvažujúce výhody zoskupovania viacerých štátov sa môžu uplatniť naraz. Vďaka navrhnutej architektúre môžu byť dáta spracované a vyhodnocované skôr a sada indexov je spoľahlivá, pokrýva všetky dáta s rozširujúcimi modulmi.

Fragmentácia je len jedným prvkom fyzickej architektúry z hľadiska blokov. Oneskorené alebo nesprávne získané a vyhodnotené údaje môžu byť následne zmenené, aby bola deklarovaná presnosť. Tieto okolnosti tvoria základ pre vytváranie migrovaných riadkov, kde pôvodný záznam po zmene už nie je možné spracovať a uložiť do pôvodného bloku z dôvodu jeho veľkosti. Migrovaný riadok je reprezentovaný uložením adresy nasledujúceho bloku, v ktorom sa záznam nachádza. Keďže samotný blok neukladá referencie indexu, vznikajú dodatočné náklady na prístup k údajom. Listový vrchol indexu získa adresu bloku

(ROWID), ktorá sa načíta do pamäte inštancie na extrahovanie záznamu. V skutočnosti tam však záznam nie je a je potrebné lokalizovať iný blok. Vo všeobecnosti môže byť potrebné spracovať viacero blokov, aby sa získal samotný potrebný záznam. Navrhli sme nové štruktúry a procesy na pozadí zodpovedné za identifikáciu migrovaných riadkov a aplikáciu zmien. Bolo navrhnutých niekoľko riešení, ktoré sa vzťahujú na výkon a obmedzenia. Štruktúra dynamického mapovania predstavuje najvýkonnejšie riešenie. Fyzické ROWID sú nahradené logickými adresami (ROWlog) do mapovacej štruktúry, v ktorej sú uložené fyzické adresy. Tým sa zavedie ďalšia vrstva, pomocou ktorej sa dá migrácia jednoducho zistiť.

Navyše, akákoľvek zmena je vždy na úrovni len jedného záznamu, bez ohľadu na počet a štruktúru indexov. Takáto štruktúra mapovania teda môže priniesť významné výkonnostné výhody, zatiaľ čo migráciu je možné umiestniť a lokalizovať len na jednom mieste. Hoci sa používa dodatočná dátová štruktúra, celkové náklady a čas spracovania sú prínosom.

9 ZÁVER

V dizertačnej práci sme sa zaoberali algoritmami indexovania, rozmiestňovania a vyhľadávania vo veľkých databázach. Prvú časť práce, obsahujúcu kapitoly 2 až 7, možno považovať za teoretickú a čiastočne praktickú časť. Obsahuje úvod do problematiky jednotlivých oblastí a rozbor možností ich aplikácie. V druhej kapitole sme sa zaoberali typmi databázových systémov, či už distribuovanými, relačnými alebo objektovo-relačnými. Zamerali sme sa aj na dátové sklady a datamarty a na ich porovnávanie, či už medzi sebou, alebo s klasickou databázou. V tretej kapitole sme sa zaoberali vyhľadávaním v dátových štruktúrach. Popísali sme rôzne typy vyhľadávania a rôzne štruktúry, ktoré sú primárne určené na vyhľadávanie, ale nie sú úplne vhodné na použitie v databázových systémoch, pretože nemajú najlepší čas na vkladanie a mazanie údajov. Štvrtá kapitola bola zameraná na indexovanie. Obsahuje analýzu niekoľkých typov indexov a metód používaných na skenovanie indexov. V tejto kapitole sme sa venovali aj stratégiám vykonávania operácie join, spôsobu rekultivácie nepoužívaných diskov pomocou zmenšovania priestoru, ako aj metóde fulltextového vyhľadávania a automatickej indexácie. V piatej kapitole sme sa zaoberali vytváraním partícií. Táto kapitola obsahuje analýzu troch techník vytvárania partícií, ktoré sa potom používajú pri rôznych metódach particiovania a tiež metódy, ktorá zabezpečuje mapovanie logických partícií na fyzické skupiny súborov na základe schémy partícií. Pozornosť je venovaná aj particiovanému indexu, či už lokálnemu, alebo globálnemu. Šiesta kapitola sa zameriava na automatickú správu úložiska, ktorá zjednodušuje správu súborov, riadiacich súborov a protokolových súborov. V siedmej kapitole analyzujeme inštitúty a univerzity, ktoré sa zaoberajú podobnou problematikou a zdôrazňujeme niektoré ich výskumy.

Druhá časť práce je obsiahnutá v ôsmej kapitole a obsahuje náš vlastný prínos založený na rôznych vykonaných experimentoch. V kapitole 8.1 sme sa zaoberali master indexom, kde bolo cieľom obmedziť nutnosť používania sekvenčného skenovania dátových blokov vykonávaného metódou Table Access Full. Master index sa nepoužíva na samotné vyhodnotenie, zatiaľ čo nevyhovuje podmienkam dotazu. Základom je, že obsahuje všetky ukazovatele na údaje na listovej vrstve, takže samotný index sa používa ako lokátor údajov. V kapitole 8.2 sme sa zamerali na skrátenie času prístupu k údajom pomocou rôznych techník a metód vytvárania partícií. Vykonali sme niekoľko scenárov, ktoré sa líšia typmi a počtom vytvorených partícií. Výsledky experimentov ukázali, že prístup k dátam uloženým v partíciách môže výrazne skrátiť čas prístupu k dátam a priniesť tak vyššiu efektivitu. Kapitola 8.3 sa zaoberá efektom particiovania a indexovania. Porovnali sa prístupové časy pre dáta v

tabuľke s vytvorenými partíciami a tabuľke bez partícií. Rôzne situácie sa líšili aj v type indexov vytvorených nad oboma tabuľkami, ako napríklad neparticiované, globálne particiované a lokálne particiované indexy s prefixom alebo bez prefixu. Z výsledkov sa usúdilo, že použitie partícií, indexov alebo ich kombinácií môže výrazne urýchliť čas prístupu k dátam. Pri častom prístupe k veľkému množstvu údajov, ku ktorým je viazaný veľký počet partícií alebo indexových vrcholov, je však čas prístupu podobný scenáru bez partícií a indexov, prípadne môže byť v niektorých situáciách ešte oveľa horší. V kapitole 8.4 sme sa zaoberali vplyvom indexov na operácie DML. Experimenty sme vykonali na systéme zdieľania bicyklov. Pre každú zo situácií vkladania, aktualizácie a odstraňovania bolo testovaných šesť scenárov, ktoré sa líšili počtom vytvorených indexov súvisiacich s upravenými údajmi. Dospeli sme k záveru, že počet indexov má významný vplyv na zvýšenie času potrebného na vykonanie operácií vkladania, aktualizácie a vymazania. Veľké množstvo indexov je vhodné len pre dáta, ktoré sa často nemenia a slúžia len na vyhľadávanie. V situáciách s častými a veľkými zmenami údajov je lepšie použiť menší počet zodpovedajúcich indexov. Kapitola 8.5 sa zameriava na vplyv kompresie tabuliek a indexov na čas prístupu k údajom a náklady na CPU. Experimenty boli vykonané v ôsmich rôznych scenároch rôznych kombinácií komprimovaných a nekomprimovaných indexov nad komprimovanými a nekomprimovanými tabuľkami s neunikátnymi údajmi s rôznymi počtami riadkov. Keď atribúty tabuliek obsahovali neunikátne hodnoty, a teda tabuľky mali nízku kardinalitu, účinok kompresie bol viditeľnejší ako v našich predchádzajúcich experimentoch s tabuľkami obsahujúcimi unikátne dáta. Kapitola 8.6 sa zaoberá komplexnosťou procesu získavania údajov pomocou rozšírenia indexu. Tu sme sa zamerali na identifikáciu a reflexiu migrovaných riadkov, automatické vyvažovanie indexu, hodnotenie efektívnosti jeho štruktúry a riadenie mimo hlavnej transakcie. Zaoberali sme sa aj kontrolou nedefinovaných hodnôt, identifikáciou relevantných dátových blokov a optimalizáciou dynamického plánu vykonania.

Naše experimenty boli realizované v systéme relačnej databázy Oracle a boli testované aj v systéme relačnej databázy MySQL. Niektoré časti riešení a scenárov však prešli drobnými zmenami, pričom iné systémy riešia dostupné funkcionality inak ako Oracle alebo ich neponúkajú vôbec. Bolo tomu tak napríklad pri vytváraní indexových partícií nad neparticiovanou tabuľkou alebo naopak pri vytváraní neparticiovaného indexu nad particiovanou tabuľkou. Takéto riešenie v Oracle je možné, no MySQL ho neumožňuje. Ďalším príkladom odlišného riešenia v iných databázových systémoch sú nedefinované hodnoty. Hodnota NULL funguje v Oracle ako prázdny reťazec, kým v MySQL po spojení reťazca a hodnoty NULL dostaneme hodnotu NULL. Na dosiahnutie analogického riešenia by bolo potrebné použiť transformačnú funkciu.

Vo všeobecnosti sú tieto riešenia univerzálne a použiteľné v rôznych systémoch. Naše metódy sú použiteľné pre akúkoľvek štruktúru, ktorá môže byť indexovaná. V situáciách, ktoré sú obohatené o kolekcie, ktoré sa nedajú indexovať, je možné použiť kľúčové slovo *TABLE* na prístup ku kolekcii ako k relačnej tabuľke a potom ju indexovať. Takže by bola potrebná medzivrstva, ktorá by transformovala kolekcie na relačné tabuľky, vytvárala indexy a potom by z nich znova vytvorila kolekciu. Pri implementácii XML typu do riešenia je možné indexovať ich jednotlivé elementy. Zodpovedajú objektovo-relačným stĺpcom a tabuľkám. Vytváranie indexov B- stromu na týchto stĺpcoch a tabuľkách tak poskytuje vynikajúci spôsob efektívneho indexovania zodpovedajúcich objektov XML. XMLIndex navyše poskytuje všeobecný index špecifický pre XML, ktorý indexuje vnútornú štruktúru údajov XML. Jedným z jeho hlavných účelov je prekonať obmedzenie indexovania, ktoré predstavuje binárne úložisko XML.

Čo sa týka dátového typu JSON, neexistuje vyhradený dátový typ SQL pre dáta JSON, takže ich možno indexovať zvyčajným spôsobom. Je možné definovať JSON vyhľadávací

index, ktorý je užitočný pre ad hoc štrukturálne dotazy aj pre fulltextové dotazy. Konkrétne možno použiť index B- stromu alebo bitmapový index pre funkciu SQL/JSON *json_value*. Takéto indexovanie založené na funkciách je vhodné pre dotazy, ktoré sa zameriavajú na konkrétne funkcie, čo v kontexte funkcií SQL/JSON znamená konkrétne výrazy cesty SQL/JSON. Pre ten istý stĺpec JSON je možné definovať indexy založené na funkciách aj JSON vyhľadávacie indexy.

Prostredníctvom našich experimentov a štúdie hodnotenia výkonu sme vyvinuli metodiku na zvýšenie efektívnosti prístupu k údajom, najmä prostredníctvom využitia indexov a ich automatického vyvažovania, particiovania alebo ich kombinácie, efektu kompresie údajov alebo optimalizácie dynamického plánu vykonania. Použitie master indexu ako lokátora dát môže obmedziť potrebu použitia sekvenčného skenovania blokov údajov vykonávaného metódou Table Access Full. Particiovanie tabuliek a indexov môže výrazne skrátiť čas prístupu k dátam a priniesť tak vyššiu efektivitu. Avšak pri častom prístupe k veľkému počtu partícií alebo indexových vrcholov alebo pri častých zmenách údajov nie je vždy vhodné použiť particiovanie alebo indexovanie. Počet indexov má významný vplyv na zvýšenie času potrebného na vykonanie operácií Insert, Update a Delete Veľké množstvo indexov je vhodné len pre dáta, ktoré sa často nemenia a slúžia len na vyhľadávanie. V situáciách s častými a veľkými zmenami údajov je lepšie použiť menšie množstvo zodpovedajúcich indexov. Použitie kompresie dát je vhodnejšie pri neunikátnych hodnotách, v porovnaní s unikátnymi dátami, kedy je zlepšenie menej viditeľné Správa indexu rozšírením štruktúry využíva dátové indexátory na vylúčenie vyrovnávacích operácií v rámci transakcie, takže pôvodná transakcia môže byť schválená skôr a výhody vyvažovania zo zoskupenia viacerých stavov môžu byť použité naraz. Dáta je možné spracovať a vyhodnotiť skôr a sada indexov je spoľahlivá a pokrýva všetky údaje pomocou rozširujúcich modulov. Pri riešení fragmentácie a migrovaných riadkov sme navrhli nové štruktúry a procesy na pozadí zodpovedné za identifikáciu migrovaných riadkov a aplikáciu zmien. Štruktúra dynamického mapovania predstavuje najvýkonnejšie riešenie. Fyzické ROWID sú nahradené logickými adresami (ROWlog) do mapovacej štruktúry, v ktorej sú uložené fyzické adresy. Takáto štruktúra mapovania môže priniesť významné výhody z hľadiska výkonu, zatiaľ čo migráciu je možné umiestniť a lokalizovať len na jednom mieste. Celkové náklady a čas spracovania sa zlepšia, aj keď sa použije dodatočná dátová štruktúra.

V blízkej budúcnosti by sme sa chceli zamerať na dynamické vyvažovanie dát vo vytvorených partíciách, ako aj na situácie s hybridným particiovaním. Zaujímavou témou nášho výskumu by mohol byť aj multi-index, v ktorom by bolo možné oba indexy spracovávať paralelne a prípadne komprimovať. Chceli by sme sa zaoberať aj závislosťou výkonu od štruktúry a hierarchie tabuliek a vzťahov medzi nimi, a teda od vplyvu úrovní spájania na čas vykonania. Zamerali by sme sa tiež na vplyv veľkostí blokov a typov diskov, na ktorých sú vytvorené partície, na výkon a časy prístupu k dátam.

ZOZNAM VLASTNÝCH PUBLIKÁCIÍ

[I] ŠALGOVÁ, V., KVET, M. (2019). *Optimization Methods in Bike-Sharing Systems*. 17th International Conference on Emerging eLearning Technologies and Applications (ICETA). - pp. 687-692 - doi: 10.1109/ICETA48886.2019.9040079.

[II] ŠALGOVÁ, V., KVET, M. (2019). *Intelligent transport and parking systems.* Dopravná infraštruktúra v mestách: zborník príspevkov z 10. Medzinárodnej konferencie. – 1.vyd. – Žilina: Žilinská univerzita v Žiline. – pp. 1-8. - ISBN: 978-80-554-1594-9.

[III] KVET, M., ŠALGOVÁ, V., KVET, M., MATIAŠKO, K. (2019). *Master Index Access as a Data Tuple and Block Locator.* Proceedings of the 25th Conference of Open Innovations Association (FRUCT). – pp. 176-183 – ISBN: 978-952-69244-1-0 - doi. 10.23919/FRUCT48121.2019.8981531.

[IV] ŠALGOVÁ, V., MATIAŠKO, K. (2020). *Analysis of Very Large Database Indexing.* International Scientific Days 2020: Innovative Approaches for Sustainable Agriculture and Food Systems Development. – ISBN: 978-963-269-918-9.

[V] ŠALGOVÁ, V., MATIAŠKO, K. (2020). *Reducing Data Access Time using Table Partitioning Techniques*. 18th International Conference of Emerging eLearning Technologies and Applications (ICETA). – pp. 564-569 – ISBN: 978-0-7381-2366-0.

[VI] ŠALGOVÁ, V., MATIAŠKO, K. (2021). *The Effect of Partitioning and Indexing on Data Access Time.* Proceedings of the 29th Conference of Open Innovations Association (FRUCT). – pp. 301-306 – ISBN: 978-952-69244-5-8 - doi: 10.23919/FRUCT52173.2021.9435500.

[VII] ŠALGOVÁ, V. (2021). *Effect of indexes on DML operations.* 14th International Scientific Conference on Sustainable, Modern and Safe Transport (TRANSCOM). – pp. 1368-1372 – ISSN 2352-1465 - Code: 146198.

[VIII] KVET, M., ČEREŠŇÁK, R., ŠALGOVÁ, V. (2021). *Use of Machine Learning for the Unknown Values in Database Transformation Processes.* 11th International Scientific Conference on Communication and Information Technologies (KIT). – pp. 1-7 – doi: 10.1109/KIT52904.2021.9583753.

[IX] ŠALGOVÁ, V., KVET, M. (2021). *The Impact of Table and Index Compression.* 19th International Conference of Emerging eLearning Technologies and Applications (ICETA) - pp. 327-332, doi: 10.1109/ICETA54173.2021.9726601.

[X] ŠALGOVÁ, V. (2022). *The Impact of Table and Index Compression on Data Access Time and CPU Costs.* 10th World Conference on Information Systems and Technologies (WorldCIST).

REFERENCIE

[1] ABDELHAFIZ, B.M. (2020). *Distributed Database Using Sharding Database Architecture*. IEEE Asia-Pacific Conference on Computer Science and Data Engineering, CSDE 2020. doi: 10.1109/CSDE50874.2020.9411547.

[2] ADAIR, B. (2019). *BI/DW: What is Business Intelligence and Data Warehousing?* Retrieved from: https://www.selecthub.com/business-intelligence/business-intelligence-and-data-warehousing/.

[3] ADAMU, F. B., HABBAL, A., HASSAN, S., COTTRELL, R. L., WHITE, B., ABDULLAHI, I. (2016). *A Survey on Big Data Indexing Strategies*. The 4th International Conference on Internet Applications, Protocols and Services.

[4] AGHAV, S. (2010). *Database compression techniques for performance optimization*. 2nd International Conference on Computer Engineering and Technology. IEEE. p. V6-714-V6-717.

[5] AHSAN, K., VIJAY, P. (2014). *Temporal Databases: Information Systems.* Booktango.

[6] AL-SANHANI, A.H. et all. (2017). *A comparative analysis of data fragmentation in distributed database*. ICIT 2017 - 8th International Conference on Information Technology, Proceedings. pp. 724–729. doi: 10.1109/ICITECH.2017.8079934.

[7] ARORA, G., KALRA, S., BHATIA, A., TIWARI, K. (2021). *PalmHashNet: Palmprint Hashing Network for Indexing Large Databases to Boost Identification*. IEEE Access. 9, pp. 145912–145928. doi: 10.1109/ACCESS.2021.3123291.

[8] ASHDOWN, L., KYTE, T. (2015). *Oracle database concepts.* Oracle Press.

[9] BURLESON, D.K. (2001). *Oracle high-performance SQL tuning*. McGraw-Hill, Inc.

[10] CAMBAZOGLU, B. et all. (2013). *A term-based inverted index partitioning model for efficient distributed query processing*. ACM Transactions on the Web, Volume 7, Issue 3. pp 1-23. doi: 10.1145/2516633.2516637.

[11] CERI, S., NEGRI, M., PELAGATTI, G. (1982). *Horizontal data partitioning in database design*. ACM SIGMOD International Conference on Management of Data. pp. 128-136.

[12] CHAN, H.L., HON, W.K., LAM, T.W. (2004). *Compressed index for a dynamic collection of texts*. In*: Annual Symposium on Combinatorial Pattern Matching.* Springer, Berlin, Heidelberg. p. 445-456.

[13] CHEN, H., LI, J. (2013). *The Research of Embedded Database Hybrid Indexing Mechanism Based on Dynamic Hashing*. Lecture Notes in Electrical Engineering. 211 LNEE, pp. 691–697. doi: 10.1007/978-3-642-34522-7_74.

[14] CHENG, C.H., WEI, L.Y., LIN, T.C. (2007). *Improving relational database quality based on adaptive learning method for estimating null value*. Second International Conference on Innovative Computing, Information and Control, ICICIC 2007. doi: 10.1109/ICICIC.2007.350.

[15] CORNEJO, R. (2018). *Dynamic Oracle Performance Analytics*. doi: 10.1007/978-1-4842-4137-0.

[16] DARGAHI NOBARI, A., RAFIEI, D. (2021). Efficiently Transforming Tables for Joinability. doi: arxiv-2111.09912.

[17] DATE, C.J., LORENTZOS, N., DARWEN, H. (2015). *Time and Relational Theory: Temporal Databases in the Relational Model and SQL.* Morgan Kaufmann.

[18] DELPLANQUE, J., ETIEN, A., ANQUETIL, N., AUVERLOT, O. (2018). *Relational database schema evolution: An industrial case study.* IEEE International Conference on Software Maintenance and Evolution ICSME 2018 - pp. 635-644.

[19] DESAI, M., MEHTA, R., RANA, D. (2019). *A Survey on Techniques for Indexing and Hashing in Big Data*". doi: 10.1109/CCAA.2018.8777454.

[20] ERLANDSSON, M. et all. (2016) *Spatial and temporal variations of base cation release from chemical weathering a hisscope scale*. In Chemical Geology, Vol. 441, pp. 1-13.
[21] EZEIFE, C.I., ZHENG, J. (1999). *Measuring the performance of database object horizontal fragmentation schemes*. Proceedings of the International Database Engineering and Applications Symposium, IDEAS. pp. 408–414. doi: 10.1109/IDEAS.1999.787292.
[22] FERRAGINA, P., MANZINI, G. (2001). *An experimental study of a compressed index*. Information Sciences. 135.1-2: 13-28.
[23] FEUERSTEIN, S. (2007). *Oracle PL/SQL Best Practices: Write the Best PL/SQL Code of Your Life*. O´Reilly Media. Inc. 978-0596514105.
[24] FINIS, J. et all. (2015). *Indexing highly dynamic hierarchical data*. Proceedings of the VLDB Endowment. 8. 986-997. 10.14778/2794367.2794369.
[25] FREITAG, M. et all. (2020). *Adopting Worst-Case Optimal Joins in Relational Database Systems*. Proceedings of the VLDB Endowment, Volume 13, Issue 12. pp 1891–1904. doi: 10.14778/3407790.3407797.
[26] GARCÍA-MOLINA H., ULLMAN J.D., WIDOM J. (2009). *Database Systems: The Complete Book (Second Edition).* New Jersey.
[27] GRAEFE, G. (2003). *Sorting And Indexing With Partitioned B-Trees*. *CIDR*. Vol. 3.2.
[28] GRAEFE, G., GUY, W., SAUER, C. (2016). *Instant Recovery with Write-Ahead Logging: Page Repair, System Restart, Media Restore, and System Failover, Second Edition*. Synthesis Lectures on Data Management. 8, pp. 1–113.
[29] HAN, W.S., LEE, K.H., LEE, B. (2003). *An XML storage system for object-oriented/object-relational DBMSs. Journal of Object Technology*. 2. 113-126. 10.5381/jot.2003.2.3.a2.
[30] HAJMOOSAEI, A., KASHFI, M., KAILASAM, P. (2011). *Comparison plan for data warehouse system architectures*. The 3rd International Conference on Data Mining and Intelligent Information Technology Applications*, Macao, pp. 290-293.*
[31] HAY, D.C. (2018). *Achieving Buzzword Compliance: Data Architecture Language and Vocabulary*. Technics Publications.
[32] HEMALATHA, G., THANUSKODI, K. (2010). *A Survey on Join Techniques*. Annual International Conference on ADPC. doi: 10.5176/978-981-08-7656-2 A-41.
[33] HONISHI, T., SATOH, T., INOEU, U. (1992). *An index structure for parallel database processing.* Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing.
[34] HUANG, H., LUAN, H. (2021). *Rethinking Insertions to B + -Trees on Coupled CPU-GPU Architectures*. pp. 993–1001. doi: 10.1109/ISPA-BDCLOUD-SOCIALCOM-SUSTAINCOM52081.2021.00139.
[35] INMON, W.H. (1992). *Building the Data Warehouse*. John Wiley & Sons, Inc., USA. ISBN 978-81-265-0645-3.
[36] JIN, D., CHEN, G., HAO, W., BIN, L. (2020). *Whole Database Retrieval Method of General Relational Database Based on Lucene*. Proceedings of 2020 IEEE International Conference on Artificial Intelligence and Computer Applications, ICAICA 2020. pp. 1277–1279. doi: 10.1109/ICAICA50127.2020.9182496.
[37] JOHNSTON, T. (2014). *Bi-temporal data – Theory and Practice.* Morgan Kaufmann.
[38] JOHNSTON, T., WEIS, R. (2010). *Managing Time in Relational Databases.* Morgan Kaufmann.
[39] KAO, M., CERCONE, N., LUK, W.S. (1988). *Providing Quality Responses with Natural Language Interfaces: The Null Value Problem*. IEEE Transactions on Software Engineering. 14, pp. 959–984. doi: 10.1109/32.42738.

[40] KHATRI, H., FAN, J., CHEN, Y., KAMBHAMPATI, S. (2007). *QPIAD: Query processing over incomplete autonomous databases*. Proceedings - International Conference on Data Engineering. pp. 1430–1432. doi: 10.1109/ICDE.2007.369028.

[41] KUHN, D., ALAPATI, S., PADFIELD, B. (2016). *Partitioned Indexes.* In: Expert Oracle Indexing and Access Paths. Apress. Berkeley. CA. doi: 10.1007/978-1-4842-1984-3_6.

[42] KUHN, D., KYTE, T. (2021). *Expert Oracle Database Architecture*. APress. Berkeley. doi: 10.1007/978-1-4842-7499-6.

[43] KUMAR, A. (2018). *Architecting Data-Intensive Applications: Develop scalable, data-intensive, and robust applications the smart way*. Packt Publishing Ltd.

[44] KVET, M. (2021). *Database Index Balancing Strategy*. Conference of Open Innovation Association, FRUCT 2021. pp. 214–221. doi: 10.23919/FRUCT52173.2021.9435452.

[45] KVET, M. (2021). *Relational data index consolidation*. 28th Conference of Open Innovation Association, FRUCT 2021. pp. 215-221. doi: 10.23919/FRUCT50888.2021.9347614.

[46] KVET, M. (2022). *Study of duplicate tuple management*. pp. 3081–3088. doi: 10.1109/SMC52423.2021.9658726.

[47] KVET, M., KVET, M. (2021). *Relational pre-indexing layer supervised by the DB-index-consolidator background process*. Conference of Open Innovation Association, FRUCT 2021. doi: 10.23919/FRUCT50888.2021.9347573.

[48] KVET, M., MATIAŠKO, K. (2014). *Transaction Management in Temportal System.* IEEE Conference CISTI 2014 – pp. 868-873.

[49] KVET, M., MATIAŠKO, K. (2019). *Efficiency of the relational database tuple access*. INFORMATICS 2019 - IEEE 15th International Scientific Conference on Informatics, Proceedings. pp. 231–236. doi: 10.1109/INFORMATICS47936.2019.9119325.

[50] KVET, M., MATIAŠKO, K. (2020). *Analysis of current trends in relational database indexing*. International Conference on Smart Systems and Technologies (SST). pp. 109-114. doi: 10.1109/SST49455.2020.9264034.

[51] KVET, M., MATIAŠKO, K. (2021). *Data Loading and Migration Methods in the Cloud Environment*. In: 2021 Communication and Information Technologies (KIT). pp. 1–6.

[52] LI, S., QIN, Z., SONG, H. (2016). *A Temporal-Spatial Method for Group Detection, Locating and Tracking.* IEEE Access, volume 4.

[53] LIEBEHERR, J., OMIECINSKI, E. R., AKYILDIZ, I. F. (1993). *The effect of index partitioning schemes on the performance of distributed query processing*. In IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 3, pp. 510-522. doi: 10.1109/69.224201.

[54] LO, Y.-L., TAN, C.-Y. (2012). *A Study on Multi-Attribute Database Indexing on Cloud System*. Lecture Notes in Engineering and Computer Science. 2195, 299–304.

[55] LORENZINI, M., KIM, W., AJOUDANI, A. (2022). *An Online Multi-Index Approach to Human Ergonomics Assessment in the Workplace*. IEEE Transactions on Human-Machine Systems. doi: 10.1109/THMS.2021.3133807.

[56] MATIAŠKO, K., KVET, M. (2020). *Temporálne databázy – 1. vyd.* Žilinská univerzita, Žilina. ISBN 978-80-554-1662-5.

[57] MATIAŠKO, K., VAJSOVÁ, M., AND KVET M. (2017) *Pokročilé databázové systémy 1. diel*. EDIS.

[58] MEI, Y., JI, K., WANG, F. (2013). *A Survey on Bitmap Index Technologies for Large-Scale Data Retrieval*. 6th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), Shenyang. pp. 316-319, doi: 10.1109/ICINIS.2013.88.

[59] PÖSS, M., POTAPOV, D. (2003). *Data compression in Oracle.* In: Proceedings 2003 VLDB Conference. Morgan Kaufmann, p. 937-947.

[60] PUSHPA, S., VINOD, P. (2007) *Binary Search Tree Balancing Methods: A Critical Study*. International Journal of Computer Science and Network Security 7. pp. 237-243.

[61] RADAIDEH, M.A. (2015). *A Distributed and Parallel Model for High-Performance Indexing of Database Content*. 26, 257–262 (2015). doi: 10.1080/1206212X.2004.11441747.

[62] RAOUF, A.E.A., ABO-ALIAN, A., BADR, N.L. (2021). *A Predictive Multi-Tenant Database Migration and Replication in the Cloud Environment*. IEEE Access. 9, pp. 152015–152031. doi: 10.1109/ACCESS.2021.3126582.

[63] ROLIK, O. et all. (2022). *Increase Efficiency of Relational Databases Using Instruments of Second Normal Form*. pp. 221–225. doi: 10.1109/ATIT54053.2021.9678605.

[64] SAYOOD, K. (2017). *Introduction to data compression*. Morgan Kaufmann.

[65] SHANBHAG, A. (2016). *An adaptive partitioning scheme for ad-hoc and time-varying database analytics*. Massachusetts Institute of Technology.

[66] SHARMA, V. (2020). *How indexing helps in improving performance of databases*. Retrieved from https://www.clariontech.com/blog/how-indexing-helps-in-improving-performance-of-databases.

[67] SPÄRCK JONES, K. (1974). *Automatic Indexing*. Journal of Documentation, Vol. 30 No. 4, pp. 393-432.

[68] TANG, Y., CHEN, L., LIU, J., LI, D. (2016). *Speeding up virtualized transaction logging with vtrans*. Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS. 0, pp. 916–923. doi: 10.1109/ICPADS.2016.0123.

[69] THAT, D.-H.T., GHAREHDAGHI, M., RASIN, A., MALIK, T. (2022). *LDI: Learned Distribution Index for Column Stores*. pp. 376–387. doi: 10.1109/BIGDATA52589.2021.9671318.

[70] TIMS, J., GUPTA, R., SOFFA, M. (1998). *Data Flow Analysis Driven Dynamic Data Partitioning*. In: Selected Papers from the 4th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers. pp. 75–90. Springer-Verlag, Berlin, Heidelberg.

[71] TUZHILIN, A. (2016). *Using Temporal Logic and Datalog to Query Databases Evolving in Time.* Forgotten Books.

[72] UNNIKRISHNAN, K., PRAMOD, K. (2009). *On Implementing Temporal Coalescing in Temporal Databases Implemented on Top of Relational Database Systems*. In: Proceedings of the International Conference on Advances in Computing, Communication and Control. pp. 153–156. doi: 10.1145/1523103.1523135.

[73] VINAYAKUMAR, R., SOMAN, K., MENON, P. (2018). *DB-Learn: Studying Relational Algebra Concepts by Snapping Blocks*. 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT). pp. 1-6, doi: 10.1109/ICCCNT.2018.8494181.

[74] WANG, M., XIAO, M., PENG, S., LIU, G. (2017). *A hybrid index for temporal big data*. Future Generation Computer Systems, 72, 264–272. doi: 10.1016/j.future.2016.08.002.

[75] WANG, Q., DU, Z., AND LIU, N. (2012). *Design and realization of database online migration*. 2nd International Conference on Computer Science and Network Technology, Changchun. pp. 1195-1198, doi: 10.1109/ICCSNT.2012.6526138.
[76] WANG, R., SALZBERG, B., LOMET, D. (2009). *Transaction support for log-based middleware server recovery*. Proceedings - International Conference on Data Engineering. pp. 353–356. doi: 10.1109/ICDE.2009.45.
[77] WESTMANN, T., KOSSMANN, D., HELMER, S., MOERKOTTE G. (2000). *The implementation and performance of compressed databases*. ACM Sigmod Record. 29(3). pp. 55-67.
[78] WU, E., MADDEN, S. (2011). *Partitioning techniques for fine-grained indexing*. IEEE 27th International Conference on Data Engineering. pp. 1127-1138. doi: 10.1109/ICDE.2011.5767830.
[79] ZYGIARIS, S. (2018). *Database Management Systems: A Business-Oriented Approach Using ORACLE, MySQL and MS Access*. Emerald Group Publishing.