Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra Informačního Inženýrství

Diplomová práce

Možnosti optimalizace přístupu k databázově evidovaných datům

Bc. Miroslav Smutný

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Miroslav Smutný

Systémové inženýrství a informatika
Informatika

Název práce

Možnosti optimalizace přístupu k databázově evidovaným datům

Název anglicky

Options of database-registered data access optimization

Cíle práce

Diplomová práce je zaměřena na problematiku optimalizace přístupu k relačně db evidovaným datům. Náplní a smyslem této práce je:
a) objasnit teoretické principy relačně databázové technologie v kontextu s problematikou optimalizace přístupu k relačně db evidovaným datům,
b) zmapovat momentální stav této problematiky a vymezit její relevantnost včetně požadavků na ni kladených,
c) navrhnout možnosti přijatelných řešení této záležitosti se zřetelem na identifikované požadavky,
d) ověřit funkčnost navržených záležitostí v souladu s výše identifikovanými požadavky,
e) ověřené záležitosti zobecnit pro další možná uplatnění.

Metodika

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

Závazný harmonogram:

Teoretické principy řešené problematiky, literární rešerše – předmět 1. zápočtu z DP: do 5.9.2018

Zmapování momentální situace řešené problematiky, identifikace požadavků s tím spojených: do 30.11.2018

Navržení možného řešení a jejich následné ověření: předmět 2. zápočtu z DP: do 31.1.2019

Zobecnění navržených záležitostí pro další možná použití – předmět 3. zápočtu z DP: do 25.3.2019

Doporučený rozsah práce

55-65

Klíčová slova

relačně databázová technologie, SQL, indexování, SŘBD Oracle

Doporučené zdroje informací

BRYLA, B., LONEY, K.: Mistrovství v Oracle Database 10g. Brno 2006. EAN 978802512779

CONOLLY, T.: Profesionální průvodce tvorbou efektivních databází. Computer Press Praha 2006. ISBN 978-80-251-2328-7

LACKO, L.: ORACLE Správa, programování a použití databázového systému. Brno 2007. EAN 9788025114902

MCLAUGHLIN, M.: Oracle Database 12c PL/SQL programming. New York: McGraw-Hill Education, 2014. ISBN 978-0071812436.

POKORNÝ, J., KOČMÍDOVÁ, H.: Učíme se SQL. Praha: Plus, 1993. ISBN 80-85297-47-7.

Předběžný termín obhajoby

2019/20 ZS – PEF (únor 2020)

Vedoucí práce

doc. Dr. Ing. Václav Vostrovský

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 04. 04. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Možnosti optimalizace přístupu k databázově evidovaným datům" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 06.04.2020 _____

Poděkování

Rád bych touto cestou poděkoval doc. Dr. Ing. Václavu Vostrovskému za odborné vedení této práce a veškerý věnovaný čas. Dále bych chtěl poděkovat všem svým bývalým i současným kolegům za cenné rady, znalosti a zkušenosti, které mi věnovali v průběhu zpracování této diplomové práce. Nakonec bych chtěl poděkovat své rodině a přátelům za podporu v průběhu celého studia a v průběhu zpracování této diplomové práce.

Možnosti optimalizace přístupu k databázově evidovaným datům

Abstrakt

Diplomová práce se zabývá problematikou optimalizace přístupu k databázově evidovaným datům a jejím možnostem. Teoretická část popisuje metody přístupu k datům, analýzu exekučních plánů, optimalizační techniky a samotný proces optimalizace.

V praktické částí je nejprve provedeno zmapování současné situace, po kterém následuje prezentace příkladu optimalizace v nejmenované společnosti. Poté je navržen datový model pro správu životopisů, na kterém jsou identifikovány datové výstupy. Pro definovaný datový výstup je sestaven text dotazu. Následuje zátěžový test, jehož průběh je po dokončení analyzován. Na základě analýzy je navrženo použití optimalizačních technik, které jsou následně analyzovány a porovnány s prvotní analýzou. Pro každou optimalizační techniku je spuštěn samostatný ověřovací zátěžový test. Všechny testy jsou následně porovnány s průběhem prvního testu a je identifikována nejvhodnější optimalizační technika. Na závěr je řešení zhodnoceno a řešení je zobecněno pro další využití v praxi.

Klíčová slova: relačně databázová technologie, SQL, indexování, SŘBD Oracle

Options of database-registered data access optimization

Abstract

Diploma thesis deals with issues of optimization of database registered data access and its options. Theoretical part describes methods of data access, analysis of execution plans, optimization techniques and process of optimization itself.

Practical part evaluates current situation, after which example of optimization is presented. After that, data model for resume management is desinged and data output is identified. For defined data output is build query text. Follows a performance test and its progress is analysed when finished. On basis of analysis, usage of optimization techniques is designed, which are analysed and compared to first analysis. Individual verificaton performance tests are executed for each optimization technique. All tests are then compared to first performance test and most appropriate technique is identified. At the end, solution is evaluated and generalized for implementation.

Keywords: relational database technology, SQL, indexing, DBMS Oracle

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

Zkratka	Vysvětlivka	Překlad
CBO	Cost Base Optimizer	Nákladově zaměřený optimalizátor
SQL	Structured Query Language	Strukturovaný dotazovací jazyk
RIS	Referenční Informační Systém	
PIN	Personal Identification Number	Osobní identifikační číslo
GSD	General Static Database	Obecná statická databáze
GB	Gigabyte	
HDD	Hard Disk Drive	
SSD	Solid State Drive	
I/O	Input / Output	Operace vstup / výstup
RAC	Real Application Cluster	Technologie multi-uzlové databáze

Seznam příloh

1 Úvod

Data jsou základním prvkem pro získávání informací na základě znalostí. Pro většinu společností jsou informace jedním z nejdůležitějších aktiv. Informace pomáhají při strategickém rozhodování, v marketingu i ve vztahu se zákazníky. Je tedy zřejmé, že informace jsou součástí všech úrovní podniku. Na vyšších úrovních společností mohou postačit informace sloučené, takzvané reporty, na nižších úrovních jsou důležitá data detailní a individuální pro specifický přístup ke konkrétnímu klientovi.

Přístup k datům je ovlivněn mnoha různými faktory. V případě cesty od klientského zařízení se mohou vyskytnout problémy s internetovým připojením, kdy uživatelské požadavky a aplikační odezvy nedorazí dostatečně rychle na stranu poskytovatele služby. Další problémy na cestě se mohou vyskytnout v samotné aplikaci, která může být naprogramovaná neefektivně a nezvládá rychle zpracovávat uživatelské požadavky. Omezení mohou také vzniknout i z důvodu pomalé rychlosti propojení aplikačního a databázového serveru. Posledním faktorem je tedy samotný databázový server, kterému se detailně věnuje tato diplomová práce.

Pomalý přístup k datům může mít negativní dopad na tvář společnosti. V častých případech jde o špatné hodnocení společnosti v sociálních okruzích nebo na internetu, případně klienti rovnou odchází k jiné společnosti. Tento přístup se začal projevovat až při nástupu internetu a digitalizace, kdy společnosti začaly poskytovat online nástroje pro klienty k vyřizování svých potřeb. Klienti přesto stále častěji vyžadují rychlejší a efektivnější nástroje. Z toho důvodu je nutné znát způsoby optimalizace přístupu k datům a dokázat je aplikovat na svých systémech. Pravděpodobně nejvíce se to týká finančních institucí, operátorů, leteckých společností, přepravních společností, společností poskytující sociální sítě a internetových obchodů.

Data se ukládají s již definovanými vztahy. K uložení slouží takzvané relační databázové systémy, jejichž účelem je zpřístupňovat data aplikacím nebo uživatelům.

Databázových systémů existuje celá řada, některé od velkých společností, některé vyvíjené komunitou. Každý z těchto systémů má určité výhody a nevýhody co se týče optimalizace. Principy jsou většinou podobné a jsou definovány použitím standardu

strukturovaného dotazovacího jazyku SQL. Rozdílným jsou však způsoby, jakými je těchto principů dosaženo.

Databázové systémy od velkých společností jsou ve většině případů vyvíjené od 70. let minulého století a byly do nich v průběhu let investovány miliardy dolarů. Příkladem takových společností je Oracle (Oracle Database), IBM (DB2) a Microsoft (Microsoft SQL Server). Tyto databázové systémy jsou vylepšovány velmi rychlým tempem a poskytují efektivnější nástroje a pokročilé možnosti s jakými je lze provozovat. Těchto možností je ohromné množství, počínaje úpravami tabulek, přes možnosti šifrování, komprese až k replikaci dat nebo pokročilým clusterům pro dosažení vysokého výkonu.

Oproti tomu komunitní databáze pocházejí z počátku tohoto století a spoléhají na vývoj dobrovolníků s téměř minimální nebo nulovou investicí. Komunitní vývojáři jsou schopní implementovat podobné pokročilé funkce, jako v případě databázových systémů od velkých společností, které prozatím nejsou stejně výkonné. Příkladem jsou PostgreSQL, MongoDB a SQLite. Nicméně i komunitní databáze byly adoptovány několika společnostmi, které se detailně seznámí s architekturou databáze na úrovni zdrojového kódu, investují do vývoje nástrojů a poskytují podporu. Příkladem může být EDB (Enterprise DB, vycházející z PostgreSQL).

Optimalizace přístupu k datům je díky vysokému množství databázových systémů a jejich principů poměrně složitá disciplína a společnosti investují značné množství financí k dosažení lepších výsledků v konkurenčním boji. Přesto existuje mnoho nechvalně proslulých klientských a zaměstnaneckých aplikací, které se i po několika letech provozu a vývoje nepodařilo zefektivnit.

Efektem optimalizace, mimo snížení odezvy a zlepšení sociálního postavení společnosti, může být také snížení nákladů na provoz systémů, kdy není nutné mít tak výkonný hardware.

2 Cíl práce a metodika

2.1 Cíl práce

Diplomová práce je zaměřena na problematiku optimalizace přístupu k relačně databázově evidovaným datům. Hlavním cílem je navrhnout postup optimalizace přístupu k datům, který povede ke snížení časové a zdrojové náročnosti přístupu na základě požadavků identifikovaných v teoretické části. Využity budou postupy optimalizace jako transkripce dotazů, úprava exekučních plánů, indexace sloupců, tabulkové oddíly, nápovědy optimalizátoru exekučních plánů nebo úprava parametrů. Dílčími cíli je seznámení s principy a kritérii optimalizace náročnosti přístupu k datům. Dalšími dílčími cíli je zmapování momentálního stavu, navržení a ověření řešení na vzorové databázové evidenci, Navržené řešení je zobecněno pro další použití.

2.2 Metodika

Použitá metodika této diplomové práce bude založena na studiu a analýze dostupných informačních zdrojů a existujících řešení v oblasti optimalizace přístupu. Stěžejními metodami této práce budou principy optimalizace, mezi které patří úprava exekučních plánů, indexace sloupců, tabulkové oddíly, nápovědy optimalizátoru exekučních plánů nebo úprava parametrů. Navrhované řešení bude zohledňovat identifikované požadavky a očekávání spojená s optimalizací přístupu. Na podkladě syntézy teoretických poznatků a dosažených výsledků optimalizace identifikovaných dotazů budou formulovány závěry této diplomové práce a následně zobecněny pro další možná použití.

Zvolená metodika pro realizaci praktické části této diplomové práce spočívá v těchto krocích:

1. Analýza shromážděných zdrojů a jejich kritické zhodnocení
2. Modelování a vytvoření vzorové databáze s důrazem na pravidla principů relačních databázových evidencí
3. Analýza požadovaných dotazů a jejich sestavení
4. Analýza zátěže sestavených dotazů
    a. Monitorování zátěže databázové instance

   b. Monitorování statistik běžících dotazů
   c. Analýza exekučních plánů
   d. Monitorování databázových systémových statistik pomocí pohledů a časových snímků.
5. Optimalizace problematických dotazů za pomocí možností popsaných v teoretické částí práce
   a. Úprava zápisu dotazu
   b. Změna fyzického přístupu nákladných operací
   c. Změna parametrů optimalizátoru
   d. Nápovědy optimalizátoru
6. Vyhodnocení optimalizovaných dotazů

3 Teoretická východiska

3.1 Relační databázové systémy

Relační databázové systémy jsou široce využívaná technologie pro ukládání a přístup k datům ve většině společnostech. Jejich předností je výkonnost, dostupnost, robustnost, bezpečnost za předpokladu dodržení doporučených principů pro návrh a provoz. (1)

Hlavní součástí relačních databázových systémů jsou ve většině případů tabulky. Tabulky se skládají z atributů, které představují vlastnosti objektů reálného světa, a řádků, které reprezentují jeden konkrétní objekt a jeho vlastnosti. Mezi tabulkami se vytváří relace, definující vztahy mezi jednotlivými řádky (objekty) společně s jejich násobností. (2)

3.2 Strukturovaný dotazovací jazyk

Strukturovaný dotazovací jazyk, z anglického překladu Structured Query Language, zkráceně SQL, je deklarativní jazyk určený pro přístup k datům v relačně databázových systémech. SQL pracuje s instrukcemi typu dotaz a příkaz. Dotaz je typ textové instrukce, který zobrazí na výstupu data, odpovídající definované projekci a selekci. Příkaz je typ instrukce, který po odeslání pouze vrátí kód ukončení operace, v případě neúspěšného dokončení také chybovou hlášku. (1)

3.2.1 Jazyk definice dat

Jazyk definice dat z anglického překladu Data Definition Language, zkráceně DDL, je část SQL používané pro definici databázových objektů. Pomocí DDL se objekty vytváří, upravují a mažou. Typy objektů, které je možné pomocí DDL definovat, jsou tabulky, pohledy, indexy, sekvence, synonyma a spouštěče. Všechny typy jsou součástí DDL popisu při exportování schématu, uživatele nebo databáze. (2)

```
CREATE  typ_objektu objekt parametry;
ALTER   typ_objektu objekt parametry;
DROP    typ_objektu objekt parametry;
```

3.2.2 Jazyk manipulace s daty

Část SQL pro manipulování s daty, anglicky Data Manipulation Language, zkráceně DML, se používá pro zobrazení, vkládání, úpravu a mazání dat. (2)

```
SELECT sloupce FROM objekt WHERE podmínka;
INSERT INTO objekt (sloupce) VALUES (hodnoty);
UPDATE objekt SET sloupce=hodnoty WHERE podmínka;
```

3.2.3 Jazyk kontroly nad daty

Jazyk kontroly nad daty z anglického Data Control Language, zkráceně DCL, je skupinou SQL příkazů, které se používají pro udělování nebo odebírání práv uživatelům. (1)

Práva mohou být systémová nebo uživatelská. Systémová práva opravňují uživatele k použití systémových objektů, jako jsou tabulky nebo procedury. Uživatelská práva jsou přidělována vlastníkem objektů jiným uživatelům nebo rolím. (2)

```
GRANT právo TO uživatel;
GRANT právo ON objekt TO uživatel;
REVOKE právo FROM uživatel;
REVOKE právo ON objekt FROM uživatel;

CREATE ROLE role;
GRANT právo TO role;
GRANT právo ON objekt TO role;
REVOKE právo FROM role;

GRANT role TO uživatel;
```

3.2.4 Jazyk kontroly nad transakcemi

Poslední částí SQL je jazyk kontroly nad transakcemi, anglicky Transaction Control Language, zkráceně TCL, určené pro uživatelskou práci s transakcemi. Skládá se celkem ze čtyř operací, kterými jsou uložení, návrat, bod záchrany a nastavení transakčních parametrů. (2)

Operace uložení informuje databázový systém, že provedené změny v daném připojení jsou trvalé a je možné je fyzicky uložit do objektů. Pomocí operace návratu se všechny provedené změny smažou a databázový systém je zahodí z paměti. (2)

Bod záchrany je mechanismus, díky kterému je možné označit jednotlivé kroky a není tedy nutné se v případě potřeby vracet na začátek celé transakce. (1)

Nastavení transakčních parametrů se liší v každém databázovém systému. Slouží k definování způsobu, jakým má k této transakci databázový systém přistupovat. Může se jednat o automatické uložení transakce, úroveň izolace nebo způsob uzamčení objektů. (1)

3.2.5 Datové typy

Každý databázový systém poskytuje k použití množinu datových typů. Typy lze rozdělit do několika skupin.

Do skupiny číselných typů patří celá čísla a čísla s desetinnou čárkou. Číselné typy pro celá čísla se dále rozdělují podle počtu bitů pro ukládání hodnot. Těchto typů je možné, v závislosti na databázovém systému, možné definovat až pět. Velmi malé číslo, pro které je maximální rozsah 255, malé číslo, pro které je maximální rozsah 65 tisíc, následuje střední číslo s maximálním rozsahem 16,7 milionu. Do běžného čísla je možné vložit maximální rozsah 4,3 miliard. Posledním typem je velké číslo s maximálním rozsahem 18,4 trilionů.

Skupina textových typů obsahuje typy znaky pevné délky nebo znaky proměnlivé délky. Textové typy se vždy definují s velikostí. V případě znaků pevné délky se vždy využije celá určená velikost. Pokud jsou vkládaná data menší, doplní se prázdnými znaky. Proměnlivá délka umožňuje rozdíl neukládat a tím spotřebovává méně místa. Proměnlivou délku je vhodné využít v případech, kdy není předem jasné, že vkládaná data budou vždy stejné velikosti. (2)

Do skupiny časových dat spadají typy datum a časové razítko. Typ časové razítko je oproti datu detailnější a je vhodné ho používat pro případy, kdy je nutné znát přesný čas a pořadí provedení. Tímto případem je aplikační audit. Datum ukládá den, měsíc, rok, hodinu, minutu a vteřinu. Časové razítko udržuje den, měsíc, rok, hodinu, minutu, vteřinu a setiny. (2)

Skupina XML obsahuje pouze jeden typ, XML. Tento typ v sobě může uchovávat od jednotlivých značek až po celé XML objekty. Také je možné vytvořit pouze sloupce typu XML, nebo přímo celé tabulky. (1)

Další skupinou jsou takzvané velké objekty (LOB). Ty jsou použity pro případné uložení velkých datových objemů. Velké objekty mohou být znakové, binární nebo bez určeného typu. Do velkých objektů je možné ukládat obrázky, dokumenty nebo spustitelné soubory. (2)

Skupina objektových datových typů je použita v objektově-relačních databázích a umožnují objektově orientovaný přístup k datům. V definici objektu je možné vytvořit metody, stejně jako v případě objektových databázích. Tento typ není podporován ve všech databázových systémech. (2)

3.2.6 Agregační funkce

Agregační funkce jsou používané pro získávání statistických charakteristik v definovaném souboru dat. Lze získat údaje o poloze, variabilitě, tvaru a rozložení kvantilů. Tyto funkce také umožňují provedení základních testů a regresní a korelační analýzu. (1)

Některé z pokročilých funkcí nejsou definované ve standardu SQL, ale několik společností se rozhodlo je aplikovat pro zjednodušení přístupu k datům. Většinou se jedná o pokročilé funkce, jako jsou T-test, F-test nebo regrese. V případě nutnosti je však možné tyto funkce naprogramovat v rámci vlastního databázového vývoje za předpokladu, že daný databázový systém umožňuje ukládání uživatelských funkcí nebo procedur. Pro použití agregačních funkcí je nutné aplikovat principy seskupení, tedy klauzuli GROUP BY. (2)

Tabulka č. 1 - Základní agregační funkce

Agregační funkce	Zápis v SQL
Součet	SUM(pole)
Průměr	AVG(pole)
Medián	MEDIAN(pole)
Počet	COUNT(pole)
Minimum	MIN(pole)
Maximum	MAX(pole)
Hodnost/pořadí	RANK(pole) parametry
Rozptyl	VARIANCE(pole)
Směrodatná odchylka	STTDEV(pole)

Zdroj: Autor

3.2.7 Seskupení, řazení

Dotazovací jazyk SQL umožňuje získat data z více řádků najednou a seskupit je dle definovaných sloupců. Klauzule „GROUP BY“ se zapisuje před řazením v dotazu a je následována výčtem sloupců, dle kterých se má seskupovat. (1)

Pro použití seskupení je nutné použít jednu nebo víc agregačních funkcí popsaných v kapitole 3.2.6. V případě že tomu tak není, databázový systém zahlásí chybu před zpracováním dotazu. Při operaci seskupení je možné rozlišit granulitu počtem relevantních sloupců. (2)

```
SELECT sloupec1, SUM (sloupec2)
FROM tabulka
GROUP BY sloupec1;
```

Operace řazení má za úkol seřadit výstup dotazů dle definovaných sloupců. Klauzule „ORDER BY“ se zapisuje na konec dotazu a je následována výčtem sloupců, dle kterých se má výsledek seřadit. Za výčet sloupců je také možné určit směr řazení, tedy vzestupně nebo sestupně. Řazení je možné specifikovat pro různé datové typy. Těmi jsou číselné, znakové a datumové. (2)

```
SELECT sloupec1, sloupec2
FROM tabulka
ORDER BY sloupec1 DESC|ASC;
```

3.3 Datová integrita

Datová integrita se rozděluje do tří základních typů. Entitní integrita, referenční integrita a doménová integrita. Všechny typy integrity se implementují prostřednictvím pravidel integritních omezení. (1)

Entitní integrita má za úkol kontrolovat, že každá tabulku obsahuje primární klíč, který je pro každý záznam unikátní a má nenulovou hodnotu. (1)

Referenční integrita kontroluje pomocí cizích klíčů, že data v nadřízené a podřízené tabulce souvisí tak jak bylo definováno při vytváření objektů. (1)

Doménová integrita kontroluje vkládané nebo upravované hodnoty v každém sloupci, zda mají požadovaný typ, rozsah, případně dodatečná pravidla definovaná při vytváření objektů. (2)

```
CREATE TABLE tabulka (
sloupec1 number(4),
sloupec2 number(4), - Doménová
sloupec3 char(1),
CONSTRAINT pravidlo1 PRIMARY KEY (sloupec1), Entitní
CONSTRAINT pravidlo2 FOREIGN KEY (sloupec2)
    REFERENCES tabulka2 (sloupec2) - Referenční
);
```

3.4 Přístup k datům

Většina relačních databázových systémů rozlišuje přístup k datům na typ fyzický a logický. (3) (4)

Fyzickým přístupem je myšleno načítání výsledků pro dotaz z diskového uložiště. Tento přístup je silně závislý na rychlosti diskového uložiště a je možné, že výsledný čas a náročnost dotazu je způsobena převážně tímto typem přístupu. (3) (4)

Logický přístup oproti fyzickému přistupuje pouze do paměti databázového systému a je silně závislý na velikosti paměťových struktur. Příliš malou velikostí se budou načtené výsledky odstraňovat pro uvolnění místa nových dotazům. Tím vzniká poměrně náročná režie databázovému systému, což může mít za důsledek zpomalení jiných částí. (3) (5)

Také je nutné rozlišovat, zda je zvolen způsob čtení celé tabulky nebo přes indexované sloupce. Přístup přes indexované sloupce se dále rozděluje na unikátní, rozsahový, vynechaný nebo rychlý plný. Tento způsob je ve většině případů rychlejší. Ovšem existují výjimky v případě, že tabulka je malá nebo pokud není index aktuální. (3) (5)

Výsledky dotazů s fyzickým přístupem jsou uloženy do paměťových struktur po jejich úspěšném dokončení. Náročnost opakovaného spuštění dotazu tedy může být značně odlišné od prvotního spuštění. (3) (5)

3.4.1 Indexace

Ovlivnit objem fyzického přístupu je možné pomocí indexů nad tabulkovými sloupci, které se využívají v podmínkách dotazů. Indexy snižují objem a dobu procházení diskových uložišť díky rychlému přístupu na dané sloupce. Existují dva typy indexů, které se liší fyzickou strukturou, přesto ale mají velmi podobný obsah. Konkrétně se uchovávají seřazené hodnoty z indexovaného sloupce a identifikátory řádků. (5) (6)

Index typu B-strom je vytvořen jako struktura bloků větví a bloků listů. Principem je seřazené hodnoty ze sloupce rozčlenit do oddílů dle určeného rozsahu. Blok větve, také lze pojmenovat jako kořen, slouží jako rejstřík k vyhledávání v podřízených blocích. Oproti tomu listové bloky již přímo drží kombinaci hodnoty a řádkového identifikátoru. (5) (6)

Druhým typem indexu je bitmapový index, který je vytvořen ve stejné struktuře jako B-strom, obsahem je však kombinace hodnot indexovaného sloupce, rozpětí identifikátorů řádků a samotné bitmapy. (5) (6)

Dalším typem indexů jsou takzvané funkční indexy. Definici sloupce při vytváření indexu je možné rozšířit některou z funkcí SQL. Databázový systém provede funkce definované pro sloupce, jejichž výsledky následně indexuje k odpovídajícím řádkům jako u základních typů indexů. (5) (6)

Posledním typem indexů jsou lokální nebo globální indexy, které jsou vytvářeny nad tabulkovými oddíly, které jsou detailně popsány v kapitole 3.4.2. (3) (5)

Tabulka č. 2 – Příklad bitmapového indexu

Hodnota	První ID	Poslední ID	1. řádek	2. řádek	3. řádek
Muž	AAA…	CCC…	0	1	0
Žena	AAA…	CCC…	1	0	1

Zdroj: Autor

3.4.2 Oddíly

Tabulky a indexy o velkém objemu je možné rozložit do menších dílů nazývaných oddíly, anglicky partition. Tyto oddíly je možné snadněji spravovat, distribuovat na různé diskové jednotky a měnit jejich parametry. Do přínosů oddílů lze zařadit zvýšenou dostupnost, snížený konkurenční přístup v transakčních systémech a zlepšení dotazů pro datové sklady. (3) (7) (8)

Pro každý oddíl tabulky nebo indexu je nutné zachovat stejné logické atributy objektu. Oddíly se vytváří pro jednotlivé tabulky a jako klíč pro rozdělení slouží jeden nebo více sloupců. Před DML operací se databázový systém na základě tohoto klíče rozhodne, se kterým oddílem bude pro danou operaci pracovat. (3) (7) (8)

Manipulovat s objekty je možné kolektivně nebo individuálně pomocí DDL příkazů. V případě individuální manipulace je možné rozdělit zdrojově náročné operace na vytváření, opravu a mazání oddílů, čímž se sníží nároky na fyzické zdroje. (3) (7) (8)

Pro rozdělení oddílů je možné využít jednu z dostupných strategií. V první řadě je nutné se rozhodnout, zda se využijí pouze jednoúrovňové oddíly nebo kompozitní oddíly. Následně se vybere, zda a na jaké úrovni se použijí rozsahové, seznamové, referenční nebo hash oddíly. (3) (7) (8)

Rozsahové oddíly rozdělují objekt na základě rozsah hodnot v klíčovém sloupci. Nejčastěji jsou pro klíčový sloupec používány sloupce časového typu. Může se jednat o datum vložení záznamu, který reprezentuje definovanou operaci. Příkladem může být datum prodeje, datum založení účtu nebo datum vytvoření klientského účtu. Před vložením do tabulky se na základě maximální hodnoty oddílu databázový systém rozhodne, do kterého oddílu nový řádek vloží. Podtypem rozsahových oddílů jsou intervalové oddíly, které mají na starost automatické vytváření oddílů dle definovaného intervalu. (3) (7) (8)

Pro rozdělení objektu na seznamové oddíly se využívá definovaný výčet diskrétních hodnot, které se vyskytují v klíčovém sloupci. Vložené řádky se logicky seskupují na logicky související skupiny. Typickým příkladem může být rozdělení dle typů, lokace, národnosti, kontinentu a podobně. (3) (7) (8)

Hash oddíly jsou využívány pro objekty které je nutné rozdělit, ale není již nutné rozdělovat dle nějaké logické vlastnosti. Pro rozdělení se využije interní hash funkce, která se aplikuje na hodnoty klíčového sloupce a na základě výsledku se databázový systém rozhodne do kterého oddílu nový řádek uloží. (3) (7) (8)

Referenční oddíly jsou definovány pomocí relace mezi odkazovanou a odkazující tabulkou. Odkazující tabulka neboli potomek je rozdělena na stejný typ a počet oddílů jako odkazovaná tabulka neboli rodič. Vztah je vyhodnocen pomocí cizího a primárního klíče mezi potomkem a rodičem. Tato strategie přináší několik dalších výhod. Jedná se například o snížení náročnosti na uložiště, snížení redundance klíčových sloupců a automatické operace spojení na oddílech rodiče a potomka. (3) (7) (8)

Posledním typem jsou takzvané kompozitní neboli složené oddíly. Tento typ kombinuje vždy dva typy oddílů. Kombinace je dosaženo pomocí vytvoření pododdílů. K vytvoření pododdílu je možné zvolit další klíčový sloupec. Typicky jsou využívány kombinace rozsah-hash, rozsah-list nebo rozsah-rozsah. (3) (7) (8)

Indexy nad tabulkovými oddíly je možné vytvořit dvěma způsoby, globálně a lokálně. V případě lokálních indexů jsou oddíly indexu vytvořené stejně jako tabulkové oddíly. Každý indexový oddíl je asociován právě s jedním tabulkovým oddílem. Tím je zajištěna párová nezávislost tabulky a indexu, kdy všechny hodnoty v jediném indexovém oddílu odkazují na řádky v jediném tabulkovém oddílu. (3) (8)

Globální index je, co se týče definice oddílů, oproti lokálnímu naprosto nezávislý. Každý indexový oddíl může odkazovat na jakýkoliv, případně na všechny tabulkové oddíly. Globální indexové oddíly je možné vytvářet na základě jiných klíčových sloupců, než jaké jsou definovány v tabulkovém oddílu. To je vhodné zejména ve chvíli kdy aplikační přístup vyžaduje optimalizaci jiného sloupce, než je klíčový. (3) (8)

3.4.3 Statistiky datových objektů

Pro správnou činnost optimalizátoru dotazů, kapitola 3.5.2, je nutné sbírat systémové statistiky a statistiky objektů jako jsou tabulky, sloupce a indexy. (6)

Ohledně tabulek jsou sbírány informace jako počet řádků, počet obsazených bloků a průměrná délka řádku. (6)

Pro sloupce jsou sbírány informace jako počet unikátních hodnot, počet nulových hodnot a distribuce hodnot neboli histogramy. (6)

Statistiky indexů obsahují počet listů, počet úrovní a faktor fyzického seskupení řádků ve vztahu k hodnotám sloupce. (6)

V systémových statistikách se ukládají informace o zátěži procesoru, využití procesoru, zátěži diskového uložiště a využití diskového uložiště. (6)

Z tabulkových statistik, konkrétně počtu řádků je určena kardinalita. Kardinalita určuje počet řádků vrácených v jednotlivých operacích exekučního plánu. Ke zpřesnění odhadu kardinality jsou využívány sloupcové statistiky. (6)

3.4.4 Operace spojení

Operace spojení je využívána v dotazech, při kterých se načítají data z přesně dvou objektů současně. Pro spojení je nutné specifikovat spojovací podmínku, která definuje relaci mezi objekty. Operace spojení spočívá ve spárování řádků odpovídajících podmínce. V případě, že podmínka není definována, je nad objekty proveden kartézský součin. (5) (6)

Optimalizátory používají hash spojení (hash join) pro větší datové objemy. Z menšího zdroje je pomocí deterministických funkcí vytvořena hash tabulka, do které jsou uloženy všechny řádky. Následně se skenuje objemnější datový zdroj s průběžným porovnáním s hash tabulkou pro nalezení řádků splňující podmínky spojení. (5) (6)

Proces operace vnořených smyček (nested loops) probíhá pomocí dvou iterací. V hlavní smyčce jsou data odpovídající podmínce, načtena z vnějšího datového zdroje. Ve vnořené smyčce se následně načítá každý řádek z vnitřního datového zdroje a porovná se s podmínkou spojení. V případě, že řádek odpovídá, je odeslán k zobrazení uživateli. (5) (6)

Sloučené seřazené spojení (sort merge join) je podtypem operace spojení vnořených smyček. Oba datové zdroje je nejprve nutné seřadit. Následuje, stejně jako v případě základních vnořených smyček, průchod datových smyček přes dvě iterace. (5) (6)

Kartézský součin (Cartesian Product) je typ operace, který je velmi nebezpečný při práci s velkými tabulkami. Příčinou je chybějící definice podmínek spojení. Výstupem takového dotazu je kombinace každého řádku z první tabulky s každým řádkem z druhé tabulky. Konečný objem výstupu je možné vypočítat jednoduchým vynásobením počtu řádků v první a druhé tabulce. Krom potenciálního ohromného počtu řádků je dalším problémem také nesouvislost výstupních dat a další použití takového výstupu může zapříčinit i jiné problémy. (1) (5)

3.5 Exekuční plány

Exekuční plány jsou speciální objekty určené pro definici operací a způsobů, jakým se jednotlivé dotazy mají provést. Také slouží k identifikování problematických částí a pro následnou optimalizaci. (1) (5)

Sestavení exekučních plánů má na starost SQL optimalizátor. Principy a algoritmy optimalizátorů jsou v každém databázovém systému rozdílné. To je důsledkem jiného způsobu organizace dat, objektů, principů zamykání objektů, principu sběru statistik a nastavitelných parametrů. (1) (5)

3.5.1 Analýza dotazů

V první části analýzy dotazů je zkontrolována syntaxe a sémantika, následně kontrola existujících exekučních plánů v paměti. Pro porovnání dotazů se používá vygenerované identifikační číslo dotazu na základě textu dotazu společně s číslem konkrétního plánu. V případě nalezení existujícího exekučního plánu se pokračuje postupem nazývaným měkká analýza. Pokud exekuční plán není nalezen, je nutné ho nejprve vytvořit. Poté následuje takzvaná tvrdá analýza. (1) (5)

Měkká analýza označuje situaci, kdy je nalezen exekuční plán a je možné daný dotaz pouze spustit. V případě, že se vyskytují v dotazu vázané proměnné je nutné hodnoty předat ke zpracování. Tento způsob ušetří databázovému systému spoustu času, který se projeví v celkové době běhu dotazu. (1) (5)

Tvrdá analýza značí postup analýzy, který je časově a výkonově náročný. V této části se kontrolují všechny informace o tabulkách, sloupcích, indexech, oddílech a statistikách každého typu objektu. Tyto informace je nutné načíst ze systémového katalogu. Jeden uživatelský dotaz tedy může vyvolat spoustu systémových dotazů. (1) (5)

3.5.2 Optimalizátor dotazů

Optimalizátor dotazů má na starost zajištění nejméně náročného exekučního plánu. Většina optimalizátorů je založená na odhadu nákladu dotazu, zkráceně CBO z anglického překladu Cost Based Optimalizator. Náklad je odhadnutá hodnota úměrná očekávanému využití zdrojů při exekuci konkrétního plánu. Optimalizátor vypočítá předpokládaný náklad pro přístup k datům a operacím spojení na základě skutečných výpočetních zdrojů jako jsou I/O operace, procesoru a paměti. Na základě této hodnoty se poté vybírá nejvhodnější dotaz. (1) (5)

Činnost optimalizátoru je možné rozdělit do několika obecných kroků. Výpočet odhadu, generování plánů a jejich porovnání. Pro odhad je nutné zkontrolovat statistiky objektů dotčených v konkrétním dotazu. Mimo statistiky se také kontroluje selektivita, kapitola, a kardinalita a nákladnost dotazu. Následně je možné vygenerovat exekuční plány pro porovnání. Různé plány je možné vygenerovat díky zvolení rozdílných přístupů k objektům a operací spojení. (1) (5)

3.5.3 Kritéria zátěže

Při analýze exekučních plánů je nutné znát kritické oddíly výpisu pro správnou identifikaci problémové sekce dotazu. Kritéria zátěže je možné rozdělit do několika částí. Těmi jsou zátěž na procesor, paměť a uložiště. (1) (5)

Procesorová zátěž se v případě CBO započítá do celkového nákladu. Některé databázové systémy zobrazí taktéž procesorovou zátěž v procentech. Na výpočet celkového nákladu má v tomto kritériu významný vliv zvolená operace spojení popsaná v kapitole 3.4.4. (1) (5)

Zátěž na uložiště a paměť je nutné rozlišit dle typů přístupu k datům popsaných v kapitole 3.4 s ohledem na počet řádků a celkový objem načtených bytů. (1) (5)

Optimalizace zátěže procesoru klade důraz na zlepšení operací spojení a tím ke snížení procesorového času. Zátěž na uložiště a paměť se optimalizuje směrem ke snížení celkového objemu načtených dat a ke zvolení nejlepší přístupové cesty k datům. (1) (5)

3.5.4 Selektivita

Selektivita představuje zlomek řádků z datového zdroje. Datový zdroj mohou být jak tabulky, tak výsledek operace spojení. Selektivita je vázaná na predikátory dotazu „*sloupec1 = 1234*“ nebo jejich kombinaci „*sloupec1 = 1234 AND sloupec2 = 1*“. Predikátory dotazu jsou definovány nad sloupci, které filtrují, a tedy omezují počet řádků z datového zdroje. Hodnota selektivity se pohybuje v rozmezí 0,0 až 1,0. V případě hodnoty selektivity 0,0 je řečeno, že z datového zdroje nejsou vybrány žádné řádky. Při hodnotě selektivity 1,0 jsou vybrány všechny řádky. (6)

Predikáty jsou více selektivní s přibližující se hodnotou 0,0 a méně selektivní při vyšších hodnotách. (6)

3.5.5 Kardinalita

Kardinalita je počet řádků vrácených v jednotlivých operacích v exekučním plánu. Optimalizátor určí kardinalitu operací dle komplexní sady vzorců, ve kterých jsou jako vstup použity tabulkové a sloupcové statistiky. Optimalizátor použije vzorec pro dotazy s jednoduchou omezující podmínkou nad jedinou tabulkou, pro kterou neexistují histogramy. V tomto případě optimalizátor předpokládá jednotné rozdělení a kardinalitu vypočítává jako vydělením celkového počtu řádků počtem unikátních hodnot ve sloupci s omezující podmínkou. (6)

3.6 Optimalizační techniky

Databázové systémy nabízí množství optimalizačních technik, které je možné rozdělit do několika skupin dle oblasti optimalizace. Popsané techniky jsou brány z obecného hlediska a je možné je aplikovat na většinu relačních databázových systémů.

3.6.1 Sémantika dotazu

Optimalizace exekučních plánů dotazů je vhodná poté, co se optimalizátor rozhodne pro neoptimální exekuční plán. Neoptimální je takový plán, který využívá více zdrojů, než je nutné. (5) (6)

Jednou z možností změny je změna textu dotazu do tvaru, který nebude pro optimalizátor totožný z hlediska kontextu dotazu. Je nutné upozornit, že pouhá změna pořadí objektů není považována jako změna kontextu. Optimalizátor tedy plán nezmění. Za změnu kontextu se považují typicky vnořené dotazy. Vnořené dotazy je možné využít ve všech částech nadřazeného dotazu. Konkrétně je lze využít v podmínkách, ve výběru objektů nebo ve specifikaci zobrazení. (5) (6)

Další možností optimalizace je využití vázaných proměnných. Za pomocí vázaných proměnných se zapříčiní sdílení exekučních plánů mezi všemi uživateli, kteří použijí stejný text dotazu, a tedy není nutné pokaždé provádět tvrdou analýzu plánu. Vázané proměnné se nejvíce využívají při definování podmínek, kdy se nahradí doslovné hodnoty za proměnné. Proměnné pak musí uživatel nebo aplikace naplnit předtím, než se dotaz spustí. Databázový klient využije definovanou hodnotu proměnné, kterou následně předá databázovému systému ke zpracování. (5) (6)

Mírně odlišnou technikou optimalizace je využití nápověd dotazu. V textu dotazu je nutné doplnit přesně daný text nápovědy, který je následně vyhodnocen optimalizátorem. V některých případech je však možné, že optimalizátor nepřistoupí na doporučení v nápovědě. Typicky se jedná o nápovědy pro využití indexů, operace spojení, režimu optimalizátoru, materializace a jiné. (5) (6)

3.6.2 Parametrizace

Úpravou parametrů je možné dosáhnout určitého snížení náročnosti a nákladovosti dotazů. Parametry je možné rozlišit dle oblastí, které jsou s jejich pomocí měněny. (5) (6)

Parametry optimalizátoru slouží k vynucení změny chování optimalizátoru. Je možné měnit parametry režimu optimalizátoru, sdílení kurzorů, poměr paralelního zpracování, poměr operací spojení a mnohé jiné. (5) (6)

Parametry paměti mění strukturu paměťových fondů, což vede k dosažení zvýšení logického přístupu, zvýšení doby uchování již vytvořených exekučních plánů a snížení doby provádění operací spojení. (5) (6)

3.6.3 Využití fyzických zdrojů

Ke snížení využívání fyzických zdrojů vede několik optimalizačních technik. Typicky se jedná o indexaci sloupců nebo rozdělení tabulek do oddílů. (5) (6)

Indexace je jednou z nejjednodušších technik optimalizace. Není nutné měnit definice tabulek ani upravovat text dotazů. Nicméně existují případy, kdy ani indexace nevede ke snížení nákladovosti dotazů ani ke snížení doby trvání. V tu chvíli přichází na řadu jiné techniky. (5) (6)

Vytváření tabulkových oddílů je již náročnější technikou. Je nutné změnit definici tabulky a provádět údržbu těchto oddílů, typicky vytváření a mazání starých, pokud je to možné a pokud to obchodní a datové modely dovolují. (5) (6)

3.7 Proces optimalizace

Cílem optimalizace je snížení nákladovosti, zátěže a doby exekuce dotazů na dané databázové instanci. Ke splnění tohoto cíle slouží proces optimalizace, který se skládá z několika fází. Proces optimalizace je iterační proces, kdy odstranění prvního úzkého hrdla může vést k odhalení jiného úzkého hrdla. (9)

Před samotným začátkem procesu je nutné identifikovat, zda je nutné danou databázi optimalizovat. K tomu slouží jednoduchá zpětná vazba uživatelů. Poté následuje samotný proces optimalizace. (9)

Prvním krokem je detailnější analýza od uživatelů. Je nutné identifikovat zasažené oblasti a uživatelské cíle optimalizace. Následuje analýza statistik operačního systému, databáze a aplikace pro období kdy se výkonnostní problémy nevyskytují a pro období kdy se problémy vyskytují. Je nutné odhalit přetížené zdroje nebo chybný hardware na zúčastněných elementech infrastruktury. (9)

V druhém kroku následuje kontrola vůči běžně dopouštěným chybám srovnáním symptomů zaznamenaných v dokumentaci se symptomy nalezenými v problematické databázi. (9)

Následuje stavba konceptuálního modelu chování systému za pomocí identifikovaných symptomů. Model je nápomocný ke správné identifikaci příčin výkonnostních problémů. (9)

V dalším kroku je již návrh série opravných změn a jejich očekávaných dopadů na chování systému. Implementaci změn je vhodné provést v pořadí od nejvyššího přínosu pro systém. Po implementaci následuje měření rozdílů v chování. V ideálním případě je doporučené provést pouze jednu změnu a tu následně změřit. Tento způsob je však v produkčních prostředích téměř nemožný z důvodu udržení dostupnosti systému. Z toho důvodu je doporučeno provádět změny tak, aby bylo možné odlišit jejich působení a nezávisle měřit jejich dopady. (9)

Po návrhu a implementaci je nutné validovat, zda bylo změnami dosaženo požadovaných změn a identifikace vnímání optimalizace uživatelem. V opačném případě je nutné opakovat postup a odhalit jiná úzká hrdla v systému. (9)

Po dokončení těchto kroků je doporučeno opakovat od návrhu změn, dokud není dosaženo požadované výkonnosti nebo dokud se proces nestává nemožným z důvodu jiných omezení. (9)

Při identifikaci běžných chyb lze vycházet z předpokladu, že je možné se těchto chyb dopouštět na všech databázových systémech. Každý databázový systém je však víc náchylný na jiné chyby. Do běžných chyb lze zahrnout: (9)

- Špatná správa připojení
- Špatné využívání paměťových zdrojů
- Špatně definované dotazy
- Použití nestandardních parametrů
- Špatná konfigurace diskového uložiště
- Problémy v nastavení zachovávání transakcí
- Dlouhé plné tabulkové procházení
- Vysoký počet opakujících se systémových dotazů
- Problémy vycházející z migrací nebo nasazení

4 Vlastní práce

Možnosti optimalizace přístupu k databázově evidovaným datům popsané v teoretické části budou reprezentovány na vzorové databázi pro Referenční Informační Systém.

Vzorová databáze je implementována na relačním databázovém systému Oracle Database Enterprise Edition, verze 12.2.0.1. Tento databázový systém byl vybrán z důvodu rozšířenosti ve společnostech, jeho kvality a zkušeností autora.

4.1 Analýza současné situace

V poslední době se vyskytlo několik případů rozsáhlých problémů mediálního dopadu. Většinou se to týká sociálně známých společností nebo organizací.

Na začátku října roku 2017 se banka Unicredit Bank potýkala s jedním z největších výpadků internetového a mobilního bankovnictví za celý rok. Výpadek byl rozsáhlý a zdlouhavý, problém byl vyřešen až po pěti dnech a postihl všech 450 tisíc klientů. Důsledkem problému byla dlouhá odezva, nefunkční zobrazování historie transakcí, a platebních karet. Tiskový mluvčí uvedl, že výpadek byl způsoben z největší míry přetížením systémů. Příčinou tedy může být špatná optimalizace aplikací, špatná optimalizace databázových dotazů nebo nedostatečný výkon serverů. Problém se silně podepsal na tváři společnosti a ztrátě mnoha klientů.

Začátkem prosince roku 2017 se s výkonnostními problémy setkaly dvě z největších českých bank, Komerční banka a Československá obchodní banka. V rozmezí jednoho týdne se obě banky setkaly s problémy v internetovém a mobilním bankovnictví. V obou případech se problémy vyskytly po plánovaných odstávkách systémů. Důsledkem problémů byly dlouhé odezvy až nedostupnosti systémů, nefunkční zadávání plateb, chybové hlášky o nedostupnosti serveru, nefunkční autorizace nebo špatné zobrazování stavu účtu. Přesné příčiny ani jedna banka nezveřejnila, což je naprosto pochopitelné. Nelze tedy s jistotou určit, zda za výpadky mohou aplikace, databáze nebo jiné části infrastruktury. Problémy se podepsaly jak na tváři společností, tak na vztahu s klienty.

V lednu roku 2019 se s výpadky střetl i Státní ústav pro kontrolu léčiv se systémem eRecept. Dlouhé odezvy znepříjemnily práci lékařům, lékárníkům i čekajícím pacientům. Ústav byl silně kritizován pacienty i lékaři za odezvy systému.

Dalším příkladem problémů ve státním sektoru je výpadek registru vozidel při zavedení v roce 2012. Systém při prvotním spuštění zkolaboval na několik hodin. Důvodem byla délka odezvy, která vznikala při zadání dotazu do centrálního registru. Pravděpodobnou příčinou byla špatná optimalizace databázových dotazů nebo prostupnosti síťových prvků.

4.1.1 Příklad optimalizace

Následující příklad zobrazuje optimalizaci konkrétního dotazu využitím techniky materializovaných pohledů.

V příkladu jsou přejmenovány veškeré názvy tabulek, sloupců, indexů nebo jiných skutečností, které by měli za důsledek odhalení částí databáze. Optimalizace byla provedena na databázi, která slouží pro správu datových toků klientských objednávek nabízených produktů. Do databáze přistupují hlavně zaměstnanci na pobočkách pomocí CRM systému. Nad databází běží 3 instance v režimu RAC[1]. Další specifikace dotazu není možné publikovat z důvodu zachování mlčenlivosti v rámci organizace.

[1] RAC – Real Application Cluster – poskytuje spojení výkonu několika serverů nad jednou databází, stará se o konkurenční přístup, distribuci transakcí a jiného zpracování.

Původní dotaz:

```
SELECT /*+ PARALLEL 4 */
       p.ROWID, cm.ROWID, ci.ROWID, cd.ROWID, cp.ROWID,
       cs.ROWID, fr.ROWID, fc.ROWID, fb.ROWID, fa.ROWID,
       p.COL1, p.COL6, p.COL5, p.COL7, p.COL8,
       p.COL9, p.COL10, p.COL11, p.COL12,
       CAST(cm.VAL AS VARCHAR(30 BYTE)),
       CAST(ci.VAL AS VARCHAR(30 BYTE)),
       CAST(cd.VAL AS VARCHAR(30 BYTE)),
       CAST(cp.VAL AS VARCHAR(30 BYTE)),
       CAST(cs.VAL AS VARCHAR(30 BYTE)),
       fr.COL13, fr.COL14, fc.COL14, fb.COL13, fb.COL14, fa.COL14
FROM SCHEMA.TABLE1 p,
     SCHEMA.TABLE2 cs, SCHEMA.TABLE2 cm, SCHEMA.TABLE2 ci,
        SCHEMA.TABLE2 cd, SCHEMA.TABLE2 cp,
     SCHEMA.TABLE3 fr, SCHEMA.TABLE3 fc, SCHEMA.TABLE3 fb,
        SCHEMA.TABLE3 fa
WHERE
    p.COL1 = cs.COL2 AND cs.COL3 = 'value1'
  AND p.COL1 = cm.COL2 (+) AND cm.COL3 (+) = 'value2'
  AND p.COL1 = ci.COL2 (+) AND ci.COL3 (+) = 'value3'
  AND p.COL1 = cd.COL2 (+) AND cd.COL3 (+) = 'value4'
  AND p.COL1 = cp.COL2 (+) AND cp.COL3 (+) = 'value5'
  AND p.COL1 = fr.COL2 AND fr.COL4 = 'value6'
  AND p.COL1 = fc.COL2 (+) AND fc.COL4 (+) = 'value7'
  AND p.COL1 = fb.COL2 (+) AND fb.COL4 (+) = 'value8'
  AND p.COL1 = fa.COL2 (+) AND fa.COL4 (+) = 'value9'
  AND /* PERF_OPT_HACK__SUB_PREFIX */ p.COL5 LIKE 'value10%'
```

Z dotazu je možné identifikovat specifikaci čtyřnásobného paralelního přístupu „*/*+ PARALLEL 4 */*“, který je následně v exekučním plánu zohledněn a text plánu se oproti standardnímu přístupu razantně změní.

Části dotazu „*CAST*“ konvertují datový typ sloupce na datový typ definovaný ve specifikaci parametru, zde konkrétně „*AS VARCHAR(30 BYTE)*“, tedy na variabilní znakový typ velikosti 30 byte.

Z dotazu je taktéž možné vypozorovat použití vnější pravostranné operace spojení „*RIGHT OUTER JOIN*“. Tento predikát provede spojení na definovaných sloupcích a následně začlení všechny řádky z tabulky, jejíž sloupec je na pravé straně podmínky.

Exekuční plán původního dotazu je z důvodu čitelnosti zobrazen v přílozePříloha č. 1.

V exekučních plánech generovaných Oracle Database jsou důležité parametry Operation (operace), Cost (náklad), Time (doba trvání), Bytes, TempSpc (dočasné místo) a Name (název objektu).

Pole Operation určuje typ operace, který bude nebo byl použit při samotném spuštění dotazu. V této části se zobrazují jak operace spojení, tak operace fyzického přístupu k datům. Tato část je stěžejní pro zvolení optimalizačního postupu. Pole Cost znázorňuje celkový náklad na jednotlivé operace i na samotný dotaz. Jedná se o bezrozměrnou hodnotu, u které je žádoucí její nejmenší hodnota.

V poli Bytes je zobrazen předpokládaný objem dat, který je využit pro další zpracování nebo zobrazení.

Pole TempSpc zobrazuje využití dočasného tabulkového prostoru pro provedení operací řazení, seskupení nebo jiných částí dotazu, které není možné provést v paměti.

Poslední pole Name je pouze výpis názvů objektů, kterých se daná operace týká.

Z exekučního plánu je tedy možné vyčíst, že se načte 19 GB dat při celkovém nákladu 313 tisíc s využitím 31 MB dočasného prostoru v celkovém čase 13 vteřin. Databáze je typ OLTP[2], pro které jsou podobné dotazy značně neefektivní a měly by být použity v datových skladech po datové replikaci nebo migraci.

V exekučním plánu je možné pozorovat plný fyzický přístup „*TABLE ACCESS FULL*“ k objektu TABLE1 o velikosti 34 GB. Přesto že je ve sloupci Bytes pouze 1,2 GB, databáze skutečně musí projít všechny řádky a zkontrolovat, že se jedná o řádky splňující podmínky v dotazu. Další plné fyzické přístupy jsou také k tabulkám TABLE2, o velikosti 107 GB, a TABLE3, o velikosti 4 GB.

Další operací jsou rychlé plné fyzické přístup k indexům „*INDEX FAST FULL SCAN*“, které prochází celý index nad sloupcem nebo kombinací sloupců definovaných v podmínce dotazu. Index TABLE2_INDEX1 má celkovou velikost 88 GB. Index TABLE3_INDEX1 má velikost 3,7 GB.

[2] OLTP – Online Transaction Processing – online transakční zpracování.

Díky zvolení paralelního zpracování lze v exekučním plánu pozorovat operace typu „PX“, které společně s hodnotami ve sloupci Name, konkrétně „*:TQ10011*“, identifikují procesy pro distribuci paralelního zpracování.

Posledním typem operací je „*HASH JOIN RIGHT OUTER*“, které vypovídají o druhu operace spojení.

Původní dotaz byl převeden na materializovaný pohled. Materializovaný pohled je specifický typ objektu, jehož vytvoření spočívá v definici fyzického objektu, respektive tabulky a následné naplnění daty, které jsou výsledkem z daného dotazu. Materializovaný pohled je také možné opakovaně rychle obnovovat za pomocí automatizovaných úloh a záznamů o změnách v základních tabulkách.

```
SELECT * FROM SCHEMA.MV1;
```

Exekuční plán pohledu je z důvodu čitelnosti zobrazen v příloze Příloha č. 2.

Exekuční plán se značně zredukoval a dle výsledné hodnoty ve sloupci Cost je možné identifikovat poměrně slušné zlepšení. Z původních 313 tisíc se náklad snížil na 226 tisíc. Čas se snížil o 4 vteřiny.

Operace „*MAT_VIEW ACCESS FULL*“ značí plný přístup k tabulce materializovaného pohledu. Tabulka pro pohled MV1 má velikost 7 GB.

Druhá operace „*PARTITION LIST ALL*“ specifikuje, které oddíly tabulky byly použity. V tomto případě všechny oddíly, neboť nebyla použita žádná podmínka.

Je tedy možné prohlásit, že optimalizace byla v tomto případě úspěšná. Avšak pouze do chvíle, kdy přijde na řadu vytváření a rychlá obnova materializovaného pohledu. Doba vytváření je přibližně 45 minut. Rychlý obnova probíhá za normálního provozu poměrně rychle, v řádu vteřin. Problém s rychlou obnovou nastává v případech migrací nebo jiného dávkového zpracování, kdy se počet změn v základních tabulkách zvýší z několikaset tisíc za hodinu na několik milionů. Časový limit pro rychlou obnovu je překročen a samotná

obnova se nedokončí. Nastává problém, při kterém nejsou v materializovaném pohledu aktuální data. V tuto chvíli je nutné spustit celkovou obnovu, případně pohled smazat a znovu vytvořit. Průběh celkové obnovy nebo vytváření zamezí přístupu aplikace k datům z důvodu fyzického vymazání tabulky pro pohled.

4.2 Databázový server

Tabulka č. 3 - Konfigurace databázového serveru

Komponenta	Hodnota
Procesor	Intel i5-8300H 2,3 GHz
Operační paměť	10 240 MB
Operační systém	Oracle Linux verze 7.4
Databázový systém	Oracle Database 12.2.0.1

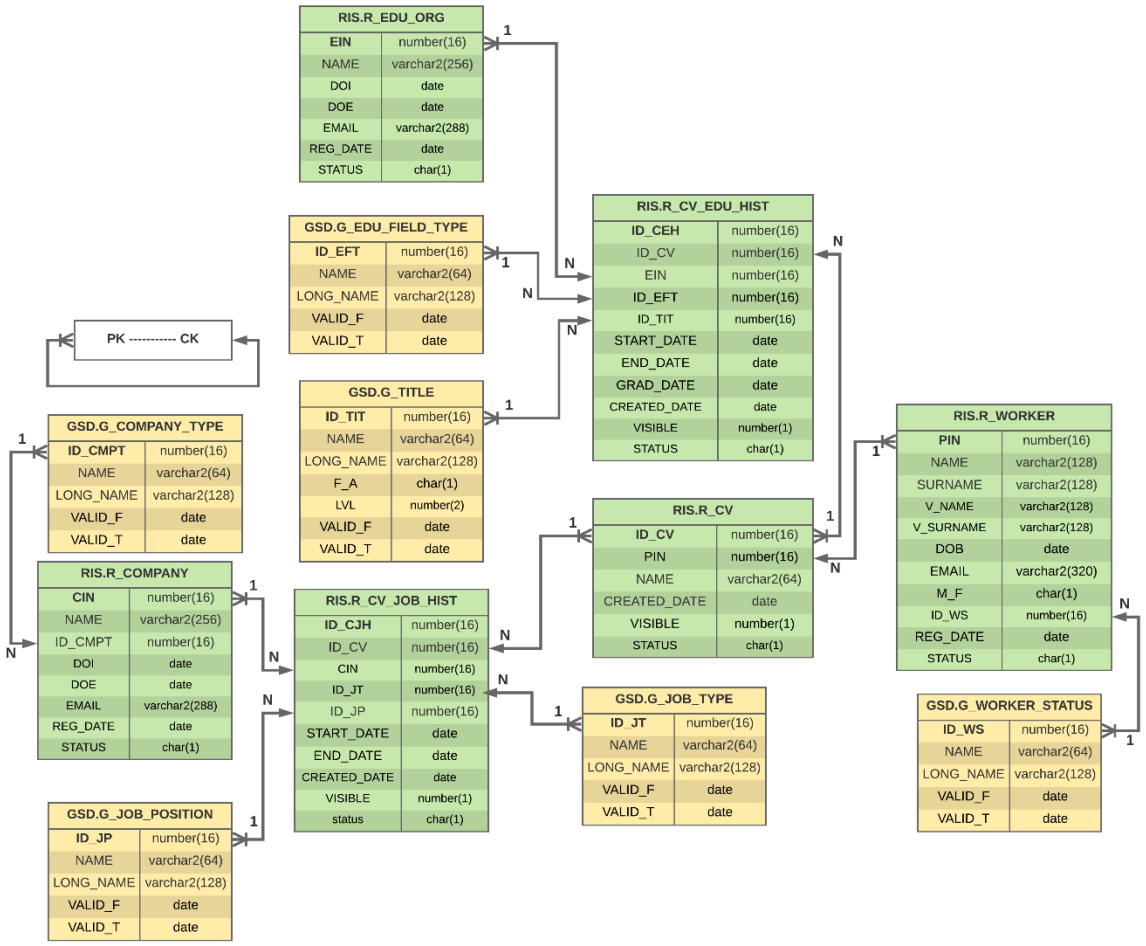
Zdroj: Autor

4.3 Referenční Informační systém

Referenční Informační Systém, zkráceně RIS, je založen na portálech s pracovními nabídkami. Přináší ovšem funkcionalitu ve formě ověření pracovních zkušeností u bývalých, případně současných zaměstnanců.

4.3.1 Datový model

Obrázek č. 1 – Diagram vztahu entit



Zdroj: Autor

4.3.2 Datový slovník

Datový slovník vychází z diagramu vztahu entit v kapitole 4.3.1.

Tabulka č. 4 – Zaměstnanec

R_WORKER	WR	Zaměstnanec
Sloupec	**Typ**	**Popis**
PIN	number(16)	Identifikační číslo zaměstnance
NAME	varchar2(128)	Jméno
SURNAME	varchar2(128)	Příjmení
V_NAME	varchar2(128	V – oslovení jménem
V_SURNAME	varchar2(128)	V – oslovení příjmením
DOB	date	Datum narození
EMAIL	varchar2(320)	Emailová adresa
M_F	char(1)	[M|F|NULL]
ID_WS	Number(16)	Číselník – status zaměstnance
REG_DATE	date	Datum vytvoření
STATUS	char(1)	[A|D|W] – Status záznamu

Zdroj: Autor

Tabulka č. 5 – Životopis

R_CV	CV	Životopis
Sloupec	**Typ**	**Popis**
ID_CV	number(16)	Identifikační číslo životopisu
PIN	number(16)	Identifikační číslo zaměstnance
NAME	varchar2(64)	Název životopisu
CREATED_DATE	date	Datum vytvoření
VISIBLE	number(1)	[0|1] – Viditelnost záznamu
STATUS	char(1)	[A|D|W] – Status záznamu

Zdroj: Autor

Tabulka č. 6 – Společnosti

R_COMPANY	COMP	Společnost
Sloupec	**Typ**	**Popis**
CIN	number(16)	Identifikační číslo společnosti
NAME	varchar2(256)	Název společnosti
ID_CMPT	number(16)	Číselník – typ organizace
DOI	date	Datum založení
DOE	date	Datum zrušení
EMAIL	varchar2(288)	Emailová adresa
REG_DATE	date	Datum registrace
STATUS	char(1)	[A|D|W] – Status záznamu

Zdroj: Autor

Tabulka č. 7 – Vzdělávací instituce

R_EDU_ORG	EO	Vzdělávací instituce
Sloupec	**Typ**	**Popis**
EIN	number(16)	Identifikační číslo organizace
NAME	varchar2(256)	Název vzdělávací instituce
DOI	date	Datum založení
DOE	date	Datum zrušení
EMAIL	varchar2(288)	Emailová adresa
REG_DATE	date	Datum registrace
STATUS	char(1)	[A|D|W] – Status záznamu

Zdroj: Autor

Tabulka č. 8 – Pracovní historie

R_CV_JOB_HIST	CJH	Pracovní historie
Sloupec	**Typ**	**Popis**
ID_CJH	number(16)	Identifikační číslo pracovní historie
ID_CV	number(16)	Identifikační číslo životopisu
CIN	number(16)	Identifikační číslo společnosti
ID_JT	number(16)	Číselník – typ pozice
ID_JP	number(16)	Číselník – pozice
START_DATE	date	Počátek zaměstnání
END_DATE	date	Konec zaměstnání
CREATED_DATE	date	Datum vytvoření
VISIBLE	number(1)	[0|1] - Viditelnost záznamu
STATUS	char(1)	[A|D|W] – Status záznamu

Zdroj: Autor

Tabulka č. 9 – Studijní historie

R_CV_EDU_HIST	CEH	Studijní historie
Sloupec	**Typ**	**Popis**
ID_CEH	number(16)	Identifikační číslo historie vzdělávání
ID_CV	number(16)	Identifikační číslo životopisu
EIN	number(16)	Identifikační číslo vzdělávací instituce
ID_EFT	number(16)	Číselník – typ oboru
ID_TIT	number(16)	Číselník – titul
START_DATE	date	Počátek studia
END_DATE	date	Konec studia
GRAD_DATE	date	Datum absolvování
CREATED_DATE	date	Datum vytvoření
VISIBLE	number(1)	[0|1] - Viditelnost záznamu
STATUS	char(1)	[A|D|W] – Status záznamu

Zdroj: Autor

Tabulka č. 10 – Typy pracovních poměrů

G_JOB_TYPE	JT	CLT – Typy pracovních poměrů
Sloupec	**Typ**	**Popis**
ID_JT	number(16)	Identifikační číslo pracovního poměru
NAME	varchar2(64)	Zkrácený název pracovního poměru
LONG_NAME	varchar2(128)	Dlouhý název pracovního poměru
VALID_F	date	Datum zadání/počátku platnosti
VALID_T	date	Datum ukončení platnosti

Zdroj: Autor

Tabulka č. 11 – Typy společností

G_COMPANY_TYPE	COMPT	CLT – Typy společností
Sloupec	**Typ**	**Popis**
ID_COMPT	number(16)	Identifikační číslo typu společnosti
NAME	varchar2(64)	Zkrácený název typu společnosti
LONG_NAME	varchar2(128)	Dlouhý název typu společnosti
VALID_F	date	Datum zadání/počátku platnosti
VALID_T	date	Datum ukončení platnosti

Zdroj: Autor

Tabulka č. 12 – Tituly

G_TITLE	TIT	CLT – Tituly
Sloupec	**Typ**	**Popis**
ID_TIT	number(16)	Identifikační číslo titulu
NAME	varchar2(64)	Zkrácený název titulu
LONG_NAME	varchar2(128)	Dlouhý název titulu
F_A	char(1)	Pozice zkratky titulu
LVL	number(2)	Úroveň titullu
VALID_F	Date	Datum zadání/počátku platnosti
VALID_T	Date	Datum ukončení platnosti

Zdroj: Autor

Tabulka č. 13 – Obory vzdělávání

G_EDU_FIELD_TYPE	EFT	CLT – Obory vzdělávání
Sloupec	**Typ**	**Popis**
ID_EFT	number(16)	Identifikační číslo oboru
NAME	varchar2(64)	Zkrácený název oboru
LONG_NAME	varchar2(128)	Dlouhý název oboru
VALID_F	date	Datum zadání/počátku platnosti
VALID_T	date	Datum ukončení platnosti

Zdroj: Autor

Tabulka č. 14 – Pracovní pozice

G_JOB_POSITION	JP	CLT – Pracovní pozice
Sloupec	**Typ**	**Popis**
ID_JP	number(16)	Identifikační číslo pozice
NAME	varchar2(64)	Zkrácený název pozice
LONG_NAME	varchar2(128)	Dlouhý název pozice
VALID_F	date	Datum zadání/počátku platnosti
VALID_T	date	Datum ukončení platnosti

Zdroj: Autor

Tabulka č. 15 – Statusy zaměstnanců

G_WORKER_STATUS	WS	CLT – Statusy zaměstnanců
Sloupec	**Typ**	**Popis**
ID_WS	number(16)	Identifikační číslo statusu
NAME	varchar2(64)	Zkrácený název statusu
LONG_NAME	varchar2(128)	Dlouhý název statusu
VALID_F	date	Datum zadání/počátku platnosti
VALID_T	date	Datum ukončení platnosti

Zdroj: Autor

4.4 Identifikace dat

4.4.1 Aplikační přístup

Do skupiny aplikačně zobrazovaných dat patří základní údaje o uživatelích, společnostech, životopisy, historie práce a studií a reference na pracovní historii.

Hlavním datovým obsahem pro aplikační zobrazení dat je několik oblastí.

- A. Zaměstnanec
  - 1. Jméno, příjmení, titul, status, email a pohlaví
- B. Životopis
  - 1. Pracovní historie
    - Pracovní pozice, Společnost, datum nástupu, datum ukončení a pracovní poměr
  - 2. Studijní historie
    - Studijní obor, škola, datum nástupu, datum ukončení, datum absolvování a titul
  - 3. Reference
    - Pracovní pozice, Společnost, datum nástupu, datum ukončení, reference, datum reference, hodnocení a referující Zaměstnanec
- C. Společnost
  - 1. CIN, název, datum založení, typ společnosti, odvětví

4.5 Sestavení dotazů

První dotaz, A1, přistupuje do tabulek R_WORKER, R_CV, R_CV_EDU_HIST, G_TITLE a G_WORKER_STATUS za účelem získání základních informací o zaměstnanci.

Text dotazu je definován následovně:

```
SELECT
    /* A11 */
    wr.PIN
    ,
    (
    SELECT
        TITLE
    FROM
        (
        SELECT
            cv.PIN
            ,tit.NAME TITLE
            ,ROW_NUMBER() OVER (PARTITION BY cv.ID_CV ORDER BY
ceh.END_DATE DESC) RWN
        FROM
            RIS.R_CV cv
            JOIN RIS.R_CV_EDU_HIST ceh ON cv.ID_CV=ceh.ID_CV
            JOIN GSD.G_TITLE tit ON tit.ID_TIT=ceh.ID_TIT
        WHERE
            ceh.STATUS='A'
        )
    WHERE
        RWN=1
        AND PIN=wr.PIN
    ) TITLE
    ,wr.NAME
    ,wr.SURNAME
    ,wr.DOB
    ,wr.EMAIL
    ,CASE
    WHEN wr.M_F = 'M' THEN 'Muž'
    WHEN wr.M_F = 'F' THEN 'Žena'
    END GENDER
    ,ROUND(((SYSDATE - wr.DOB)/365.242199),1) AS AGE
FROM
    RIS.R_WORKER wr
    JOIN GSD.G_WORKER_STATUS ws ON ws.ID_WS=wr.ID_WS
WHERE
    ws.ID_WS IN (1, 2, 3)
    AND wr.STATUS='A'
    AND cv.STATUS='A'
    AND ceh.STATUS='A'
    AND wr.PIN=číslo
;
```

4.6 Zátěžové testy

Zátěžové testování se provede za pomoci generování požadovaných vstupních dat s důrazem na co nejnižší zátěž z důvodu neovlivnění výsledků testu. Generování vstupních dat bude upraveno tak, aby odpovídalo očekávaným vstupům skutečných uživatelů.

Zátěžový test bude rozdělen do dvou fází. V první fázi bude generována aplikační zátěž. Při tomto typu zátěže je klíčová co nejrychlejší odezva s co nejnižší možnou zátěží na systémové prostředky jako jsou vstupní a výstupní (I/O) operace a zátěž procesoru. Odezvy dotazů a zátěž systému jsou ve velmi blízkém vztahu. Zvýšení I/O operací nebo zátěže procesoru má vliv na délku odezvy. Dlouhé odezvy mají za následek nespokojenost uživatelů, která může vést až k znehodnocení image společnosti provozující daný systém.

Druhá fáze testů má za cíl identifikovat problematické reportingové dotazy. Pro tento typ zátěže nemá odezva tak vysokou prioritu jako v aplikačním přístupu. Důraz je kladen na snížení zátěže na systémové prostředky. Tento rozdíl je z důvodu nižšího počtu vyvolání reportingových dotazů, které se nemusí zobrazit v tak krátkém čase.

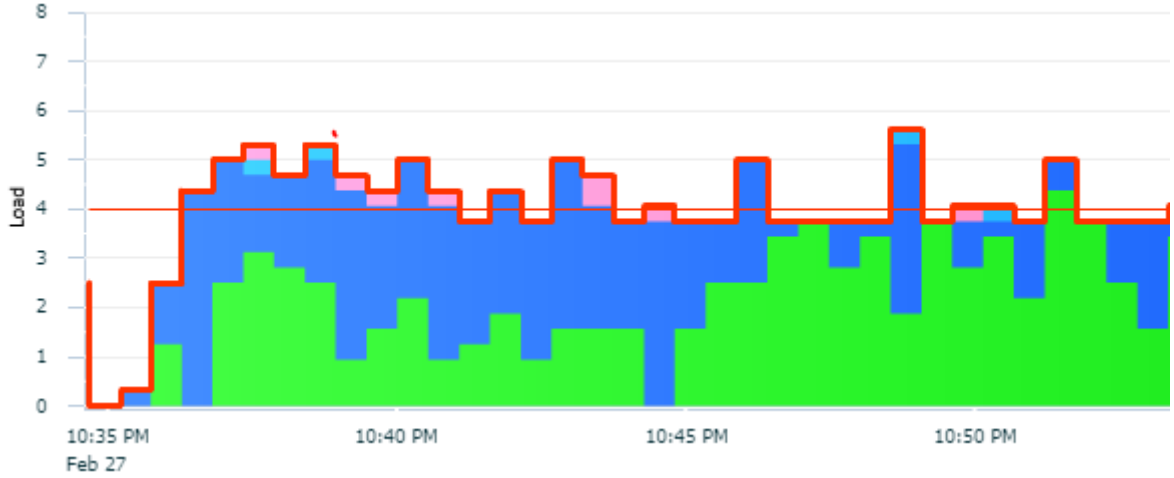
Pro otestování budou spuštěna paralelní připojení, která mají za úkol spouštět sestavené SQL dotazy v konkurenčním provozu.

4.7 Analýza zátěžových dotazů

V první fázi budou analyzovány systémové pohledy na statistiky běhu dotazů. Hledí se na zátěž procesoru, disků a paměti. V další části se následně za pomoci balíčku DBMS_XPLAN zobrazí detailní exekuční plán zátěžových dotazů. V plánu je následně potřeba identifikovat problematické části, dle kterých se upraví jednotlivé dotazy, eventuálně systémové parametry a v nejhorším případě datová struktura.

Pro kontrolu zátěže jsou stěžejní systémové pohledy do paměťové struktury sdíleného fondu, „*shared pool*“, ve které jsou uchovány probíhající nebo v nedávné historii doběhnuté SQL dotazy.

Obrázek č. 2 – Celková aktivita připojení



Zdroj: Autor

Obrázek č. 2 zobrazuje zátěž databázové instance při čtyřech paralelních přístupech. Zelená část grafu zobrazuje čekání na procesorový čas. Tmavě modrá část zobrazuje čekání na načtení dat z diskového uložiště pro nesystémové uživatele.

Tabulka č. 16 – statistika prvního zátěžového testu dotazu

Proces	Počet dotazů	Začátek	Konec	Doba běhu
25005	265	22:36:33	23:32:20	00:55:47
25543	265	22:36:33	23:32:05	00:55:32
26080	265	22:36:33	23:32:25	00:55:52
26621	265	22:36:33	23:32:25	00:55:52

Zdroj: Autor

V Tabulka č. 16 jsou zaznamenány údaje o běhu zátěžového testu ve čtyřech paralelních připojeních. Sloupec „Doba běhu“ popisuje dobu trvání jednotlivých procesů. Doba trvání zátěžového testu je maximální hodnota z tohoto sloupce. První zátěžový test měl dobu trvání 55 minut a 52 vteřin.

V průběhu testu A1 bylo odhaleno přes 1000 záznamů v systémovém pohledu „*V$SQL*“. Každý záznam udává informace o unikátním textu dotazu, jeho statistiky jako je počet spuštění, čas strávený čekáním na procesor nebo I/O operace. Vysoký počet je z důvodu lišícího se textu dotazu v části omezujících podmínek, konkrétně v omezení ve sloupci PIN.

Druhým kontrolovaným systémovým pohledem je „*V$SQL_PLAN*“, ve kterém jsou uchovány exekuční plány jednotlivých dotazů. V pohledu byl objeven totožný plán pro všech 1000 dotazů. To potvrzuje Obrázek č. 3, kde je možné ve sloupci ID vidět rozdílné identifikátory dotazů, ale ve sloupci SQL Plan Hash jsou všechny hodnoty shodné.

Obrázek č. 3 – Statistika běhu dotazu A1

ID	SQL Plan Hash	Database Time	IO Requests	SQL Text
5styrfy4zb5q3	1364083530	8.0s	3,021	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
f0pnkx3ajjm2r	1364083530	8.1s	3,018	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
g52fg8j47wspm	1364083530	7.8s	3,018	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
0vzuuzqhvq8h4	1364083530	8.5s	3,018	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
2g654a9ywyx3q	1364083530	8.1s	3,021	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
4bnpqqduxw1w2	1364083530	8.0s	3,017	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
fub0yx65jhhk7	1364083530	7.9s	3,018	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
2rz85d5fwuccu	1364083530	8.0s	3,087	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
91ckjs0077kx7	1364083530	7.9s	3,011	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...
gg6r1tc2hxhfy	1364083530	8.2s	3,011	SELECT /* A11 */ /*+ GATHER_PLAN_STATISTICS */ ...

Zdroj: Autor

Obrázek č. 4 - Statistiky časového modelu z AWR

Statistic Name	Time (s)	% of DB Time	% of Total CPU Time
sql execute elapsed time	8,344.29	95.74	
DB CPU	5,225.48	59.96	99.31
parse time elapsed	115.24	1.32	
hard parse elapsed time	110.41	1.27	
connection management call elapsed time	68.70	0.79	
PL/SQL execution elapsed time	37.24	0.43	
hard parse (sharing criteria) elapsed time	14.80	0.17	
failed parse elapsed time	14.19	0.16	
PL/SQL compilation elapsed time	8.91	0.10	
sequence load elapsed time	0.06	0.00	
repeated bind elapsed time	0.06	0.00	
hard parse (bind mismatch) elapsed time	0.00	0.00	
DB time	8,715.37		
background elapsed time	415.26		
background cpu time	36.05		0.69
total CPU time	5,261.53		

Zdroj: Autor

Díky identifikaci vysokého počtu unikátních identifikátorů dotazů je možné se ve statistikách časového modelu zaměřit na relevantní statistiky. V tomto případě „*parse time elapsed*“ (uběhlý čas analýzy dotazu) a „*hard parse elapsed time*“ (uběhlý čas tvrdé analýzy).

Nyní je nutné zjistit exekuční plán dotazů. Jelikož je hash hodnota plánu stejná, je dostačující nalezení pouze jednoho. Toho je docíleno pomocí balíčku „*DBMS_XPLAN*“ a procedury „*DISPLAY_CURSOR*“.

```
Plan hash value: 1364083530

-------------------------------------------------------------------------------------------------------
| Id | Operation                    | Name           |E-Bytes| Cost (%CPU)|   A-Time   |Buffers|Reads |
-------------------------------------------------------------------------------------------------------
|   0| SELECT STATEMENT             |                |       | 39853 (100)|00:00:00.10 |      4|     4|
|*  1|  VIEW                        |                |   140M| 39850   (1)|00:00:07.38 |  32475| 52564|
|*  2|   WINDOW SORT PUSHED RANK    |                |    96M| 39850   (1)|00:00:07.21 |  32475| 52564|
|*  3|    HASH JOIN                 |                |    96M| 13714   (1)|00:00:01.38 |  32461| 32442|
|   4|     TABLE ACCESS FULL        | G_TITLE        |   290 |     3   (0)|00:00:00.01 |      7|     6|
|*  5|     HASH JOIN                |                |    76M| 13700   (1)|00:00:00.60 |  32451| 32435|
|   6|      TABLE ACCESS FULL       | R_CV           |    13M|  2471   (1)|00:00:00.05 |   9087|  9077|
|*  7|      TABLE ACCESS FULL       | R_CV_EDU_HIST  |    46M|  6124   (2)|00:00:00.22 |  23361| 23358|
|*  8|  TABLE ACCESS BY INDEX ROWID| R_WORKER       |    66 |     3   (0)|00:00:00.10 |      4|     4|
|*  9|   INDEX UNIQUE SCAN          | R_WR_PIN_PK_IDX|       |     2   (0)|00:00:00.01 |      3|     3|
-------------------------------------------------------------------------------------------------------
```

V zobrazeném plánu je možné vidět „*TABLE ACCESS FULL*“ nad tabulkami, do kterých se přistupuje pomocí referenčních sloupců. Tyto sloupce nad sebou nemají ve výchozím nastavení vytvořené indexy a není tudíž možné zvolit jiný způsob fyzického přístupu. Operace „*TABLE ACCESS FULL*“ u nečíselníkových tabulek mají nákladovost v řádu tisíců.

Dále se v exekučním plánu vyskytují pouze hash operace spojení. Nejprve jsou spojeny tabulky „*R_CV*“ a „*R_CV_EDU_HIST*“ s nákladem 13 700. Výsledek tohoto spojení je následně spojen stejným způsobem s tabulkou „*G_TITLE*“ při nákladovosti 13 714.

Operace „*WINDOW SORT PUSHED RANK*“ je v exekučního plánu dotazu zásluhou funkce „*ROW_NUMBER*“ ve vnořeném dotazu, která má za úkol rozdělit výstup dle ID_CV a seřadit dle data ukončení.

Ve sloupci Buffers je zobrazen počet čtení z paměťového prostoru, tedy logický přístup. Nicméně ve vedlejším sloupci Reads, který zobrazuje počet čtení z diskového prostoru, tedy fyzický přístup, je hodnota přibližně stejná. Jednou z příčin tohoto chování je pravděpodobně nízká velikost paměťových struktur.

4.8 Optimalizace dotazů

4.8.1 Optimalizace vázanou proměnnou

Po zátěžovém testu jsou známé oblasti možné optimalizace dotazu. První je úprava podmínky dotazu na vázané proměnné pro dosažení sdílení a opakované použití identifikátoru a plánu dotazu.

Úprava dotazu se týká posledního řádku, "`AND wr.PIN=číslo`", který se upraví na "`AND wr.PIN=:pin`". Pro využití proměnné v dotazu je však nutné předem definovat proměnnou a následně do ní korektně přiřadit hodnotu. V knihovnách programovacích jazyků je často využíván pojem „*prepare*" pro funkce, které slouží pro naplnění proměnných v dotazu. V případě, že zdrojový kód nevyužívá vázané proměnné v dotazech, je nutná úprava i v kódu.

K ověření účinnosti změny v textu dotazu je možné spustit dotaz opakovaně s rozdílnými hodnotami v omezující podmínce. Současně je spuštěn původní dotaz pro porovnání výstupů.

```
SQL_ID        HASH_VALUE   CPU_TIME ELAPSED_TIME EXECS START              END
------------- ---------- ---------- ------------ ----- ----------------- ----------------
0zwjx2b92tc23 1364083530  6,332,893    7,289,245     1 SELECT /* A11 */  AND wr.PIN=24242
34wj8z76mps5d 1364083530  6,450,541    7,607,620     1 SELECT /* A11 */  AND wr.PIN=1342
5mxdrxhjnw8h7 1364083530  6,274,358    7,302,421     1 SELECT /* A11 */  AND wr.PIN=24263
9fgqgr06gxbay 1364083530 19,079,039   22,047,862     3 SELECT /* A11 */  AND wr.PIN=:pin
```

Ze snímku „*V$SQL*" je možné vyvodit, že dotaz s vázanou proměnnou je používán opakovaně. Ve sloupci EXECS je možné identifikovat počet exekucí. Druhým ukazatelem jsou násobně vyšší hodnoty ve sloupcích CPU_TIME a ELAPSED_TIME. Tyto sloupce popisují dobu trvání v jednotkách mikrosekund. Sloupec CPU_TIME popisuje procesorový čas z celkové doby exekuce dotazu, tedy ELAPSED_TIME.

Ze sloupce HASH_VALUE je také možné vypozorovat, že exekuční plán zůstává stejný a použití vázaných proměnných nevynutí jeho přepočítání.

4.8.2 Optimalizace nápovědou optimalizátoru

Použitím nápovědy pro paralelizaci fyzického přístupu je dosaženo souběžného čtení z diskového uložiště. Nápověda je definována ve vnořené části dotazu „`/*+ PARALLEL(4) */`“. Po otestování paralelizaci je patrná změna operací fyzického přístupu v exekučního plánu.

```
Plan hash value: 1067293009

-------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name            |Cost (%CPU)| PQ Distrib  |   A-Time  |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                 | 9931 (100)|             |00:00:00.11|
|   1 |  PX COORDINATOR                    |                 |           |             |00:00:02.16|
|   2 |   PX SEND QC (RANDOM)              | :TQ10002        | 9928   (1)| QC (RAND)   |00:00:00.01|
|*  3 |    VIEW                            |                 | 9928   (1)|             |00:00:00.01|
|*  4 |     WINDOW SORT PUSHED RANK        |                 | 9928   (1)|             |00:00:00.01|
|*  5 |      HASH JOIN                     |                 | 2472   (2)|             |00:00:00.01|
|   6 |       PX RECEIVE                   |                 |  686   (1)|             |00:00:00.01|
|   7 |        PX SEND HASH                | :TQ10000        |  686   (1)| HASH        |00:00:00.01|
|   8 |         PX BLOCK ITERATOR          |                 |  686   (1)|             |00:00:00.01|
|*  9 |          TABLE ACCESS FULL         | R_CV            |  686   (1)|             |00:00:00.01|
|  10 |       PX RECEIVE                   |                 | 1782   (2)|             |00:00:00.01|
|  11 |        PX SEND HASH                | :TQ10001        | 1782   (2)| HASH        |00:00:00.01|
|* 12 |         HASH JOIN                  |                 | 1782   (2)|             |00:00:00.01|
|  13 |          TABLE ACCESS FULL         | G_TITLE         |    2   (0)|             |00:00:00.01|
|  14 |          PX BLOCK ITERATOR         |                 | 1777   (2)|             |00:00:00.01|
|* 15 |           TABLE ACCESS FULL        | R_CV_EDU_HIST   | 1777   (2)|             |00:00:00.01|
|  16 |  PX COORDINATOR                    |                 |           |             |00:00:00.11|
|  17 |   PX SEND QC (RANDOM)              | :TQ20001        |    3   (0)| QC (RAND)   |00:00:00.01|
|* 18 |    TABLE ACCESS BY INDEX ROWID     | R_WORKER        |    3   (0)|             |00:00:00.01|
|  19 |     BUFFER SORT                    |                 |           |             |00:00:00.01|
|  20 |      PX RECEIVE                    |                 |    2   (0)|             |00:00:00.01|
|  21 |       PX SEND HASH (BLOCK ADDRESS)| :TQ20000        |    2   (0)| HASH (BLOCK)|00:00:00.01|
|* 22 |        INDEX UNIQUE SCAN           | R_WR_PIN_PK_IDX |    2   (0)|             |00:00:00.01|
-------------------------------------------------------------------------------------------------------
```

Oproti původnímu exekučnímu plánu stoupl počet operací pro fyzický přístup a také se změnilo pořadí operací spojení. V původním plánu slouží pro načtení dat operace „*TABLE ACCESS FULL*“ s čísly 9 a 15, které jsou nyní podřízené operacím typu „*PX BLOCK ITERATOR“,* které provádí paralelní načítání dat. Celé paralelní zpracování zastřešují operace „*PX COORDINATOR*“, skutečné procesy, které komunikují mezi sebou a předávají si data ke zpracování. Skutečné předání je provedeno mezi operacemi „*PX SEND QC (RANDOM)*“ a „*PX RECEIVE*“. Po předání dojde ke zpracování operací spojení pomocí „*PX SEND HASH*“.

Snížení nákladovosti je znatelné, z celkových původních 39 853 na současných 9 931. Pro jednotlivé operace se snížení také prokázalo. Pro operace „*TABLE ACCESS FULL*“ u tabulky R_CV z 2 471 na 686, u tabulky R_CV_EDU_HIST z 6 124 na 1 777. Operace spojení nyní mají nákladovost 2 472 a 1 782.

4.8.3 Optimalizace využitím tabulkových oddílů

Transformací tabulek na tabulky oddílové je možné dosáhnout změny operací fyzického přístupu v exekučního plánu. Tabulkové oddíly jsou definovány již při vytváření tabulek. V případě, že jsou oddíly nutné až po určité době provozu, je nutné připravit změny, které nezmění datový model ale pouze fyzickou strukturu. Typicky se jedná o referenční omezení a názvy objektů.

```
Plan hash value: 1960454424

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name             | Cost (%CPU)|Pstart|Pstop|   A-Time   |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                  | 40802 (100)|      |     |00:00:00.01 |
|*  1 |  VIEW                               |                  | 40800   (1)|      |     |00:00:14.46 |
|*  2 |   WINDOW SORT PUSHED RANK           |                  | 40800   (1)|      |     |00:00:14.28 |
|*  3 |    HASH JOIN                        |                  | 15769   (1)|      |     |00:00:01.54 |
|   4 |     TABLE ACCESS FULL               | G_TITLE          |     3   (0)|      |     |00:00:00.01 |
|*  5 |     HASH JOIN                       |                  | 15755   (1)|      |     |00:00:00.69 |
|   6 |      PARTITION RANGE ALL            |                  |  3283   (1)|    1 |   14|00:00:00.07 |
|   7 |       TABLE ACCESS FULL             | R_CV_P           |  3283   (1)|    1 |   14|00:00:00.06 |
|   8 |      PARTITION RANGE ALL            |                  |  7443   (2)|    1 |   16|00:00:00.25 |
|*  9 |       TABLE ACCESS FULL             | R_CV_EDU_HIST_P  |  7443   (2)|    1 |   16|00:00:00.24 |
|  10 |  PARTITION RANGE SINGLE             |                  |     2   (0)|  KEY |  KEY|00:00:00.01 |
|* 11 |   TABLE ACCESS BY LOCAL INDEX ROWID| R_WORKER_P       |     2   (0)|  KEY |  KEY|00:00:00.01 |
|* 12 |    INDEX UNIQUE SCAN                | R_WRP_PIN_PK_IDX |     1   (0)|  KEY |  KEY|00:00:00.01 |
-------------------------------------------------------------------------------------------------------------
```

Po změně definice tabulek na oddílové došlo k několika změnám v exekučním plánu. Nad operacemi čísel 7 a 9 „*TABLE ACCESS FULL*“ se nyní vyskytují operace „*PARTITION RANGE ALL*“. Tyto operace značí průchod všech oddílů tabulky. S tím souvisí i mírně zvýšená nákladovost jednotlivých operací. Tento náklad lze připsat režii databázového systému v přechodu na datové bloky jiných oddílů.

Nákladovost plánu se mírně zvýšila na 40 802. Nákladovost jednotlivých operací se také zvýšila, zejména operace spojení, jejíž nákladovost se zvýšila o přibližně 2 000.

4.8.4 Optimalizace nápovědou optimalizátoru a tabulkovými oddíly

Kombinací optimalizačních technik oddílů a paralelizace fyzického přístupu je výsledný exekuční plán sestaven za pomoci operací z předchozích exekučních plánů.

```
Plan hash value: 357377962

-----------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name             |Cost (%CPU)|PQ Distrib |   A-Time   |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                  | 9941 (100)|           |00:00:00.04 |
|   1 |  PX COORDINATOR                    |                  |           |           |00:00:02.94 |
|   2 |   PX SEND QC (RANDOM)              | :TQ10002         | 9939   (1)|QC (RAND)  |00:00:00.01 |
|*  3 |    VIEW                            |                  | 9939   (1)|           |00:00:00.01 |
|*  4 |     WINDOW SORT PUSHED RANK        |                  | 9939   (1)|           |00:00:00.01 |
|   5 |      PX RECEIVE                    |                  | 9939   (1)|           |00:00:00.01 |
|   6 |       PX SEND HASH                 | :TQ10001         | 9939   (1)|HASH       |00:00:00.01 |
|*  7 |        WINDOW CHILD PUSHED RANK    |                  | 9939   (1)|           |00:00:00.01 |
|*  8 |         HASH JOIN                  |                  | 2985   (1)|           |00:00:00.01 |
|   9 |          PX BLOCK ITERATOR         |                  |  911   (1)|           |00:00:00.01 |
|* 10 |           TABLE ACCESS FULL        | R_CV_P           |  911   (1)|           |00:00:00.01 |
|  11 |          PX RECEIVE                |                  | 2070   (2)|           |00:00:00.01 |
|  12 |           PX SEND BROADCAST        | :TQ10000         | 2070   (2)|BROADCAST  |00:00:00.01 |
|* 13 |            HASH JOIN               |                  | 2070   (2)|           |00:00:00.01 |
|  14 |             TABLE ACCESS FULL      | G_TITLE          |    2   (0)|           |00:00:00.01 |
|  15 |             PX BLOCK ITERATOR      |                  | 2065   (1)|           |00:00:00.01 |
|* 16 |              TABLE ACCESS FULL     | R_CV_EDU_HIST_P  | 2065   (1)|           |00:00:00.01 |
|  17 |  PARTITION RANGE SINGLE            |                  |    2   (0)|           |00:00:00.04 |
|* 18 |   TABLE ACCESS BY LOCAL INDEX ROWID| R_WORKER_P       |    2   (0)|           |00:00:00.04 |
|* 19 |    INDEX UNIQUE SCAN               | R_WRP_PIN_PK_IDX |    1   (0)|           |00:00:00.02 |
-----------------------------------------------------------------------------------------------------
```

Z exekučního plánu je možné vypozorovat využití „*TABLE ACCESS FULL*“ pro operace na podřízené tabulky, konkrétně u operací 10, 14 a 16. Pro hlavní tabulku je použit přístup operací s číslem 17 „*PARTITION RANGE SINGLE*“, tedy skenování rozsahu pouze jednoho oddílu. Řádky požadovaných dat jsou přímo načteny díky použití indexu nad sloupcem primárního klíče v operaci s číslem 18 „*TABLE ACCESS BY LOCAL INDEX ROWID*“.

Kombinací optimalizace nápovědou a tabulkových oddílů je nákladovost rovna 9 941. Oproti optimalizace nápovědou se tedy zvýšila nákladovost pouze o 10. Vysoká nákladovost exekučního plánu optimalizací využitím tabulkových oddílů zde není promítnuta.

4.8.5 Optimalizace transkripcí dotazu

Vhodné bude přesunout objekty z vnořených dotazů na pozici sloupců na obecnou pozici pro výběr objektů.

Upravený text dotazu:

```
SELECT
    /* A12 */
    PIN, TITLE, NAME, SURNAME, DOB, EMAIL, GENDER, AGE
FROM
    (
    SELECT
        wr.PIN, wr.NAME, wr.SURNAME, wr.DOB, wr.EMAIL
        ,CASE
        WHEN wr.M_F = 'M' THEN 'Muž'
        WHEN wr.M_F = 'F' THEN 'žena'
        END GENDER
        ,ROUND(((SYSDATE - wr.DOB)/365.242199),1) AS AGE
        ,tit.NAME TITLE
        ,ROW_NUMBER() OVER (PARTITION BY ceh.ID_CV ORDER BY ceh.END_DATE
DESC) RWN
    FROM
        RIS.R_WORKER wr
        JOIN GSD.G_WORKER_STATUS ws ON ws.ID_WS=wr.ID_WS
        JOIN RIS.R_CV cv ON cv.PIN=wr.PIN
        LEFT OUTER JOIN RIS.R_CV_EDU_HIST ceh ON cv.ID_CV=ceh.ID_CV
        JOIN GSD.G_TITLE tit ON ceh.ID_TIT=tit.ID_TIT
    WHERE
        ws.ID_WS IN (1, 2, 3)
        AND wr.STATUS='A'
        AND cv.STATUS='A'
        AND ceh.STATUS='A'
    )
WHERE
    RWN=1
;
```

Po dokončení dotazu proběhne analýza exekučního plánu jako v jiných případech optimalizace.

```
Plan hash value: 2179299452

-------------------------------------------------------------------------------------------------------------------------------------------
| Id| Operation                          | Name                |E-Bytes| Cost (%CPU)|A-Rows |   A-Time   | Buffers | Reads |
-------------------------------------------------------------------------------------------------------------------------------------------
|  0| SELECT STATEMENT                   |                     |       |  8619 (100)|     1 |00:00:00.42 |   32456 |  32444 |
|* 1|  VIEW                              |                     |  3540 |  8619   (2)|     1 |00:00:00.42 |   32456 |  32444 |
|* 2|   WINDOW SORT PUSHED RANK          |                     |   327 |  8619   (2)|     1 |00:00:00.42 |   32456 |  32444 |
|  3|    NESTED LOOPS                    |                     |   327 |  8618   (2)|     2 |00:00:00.42 |   32456 |  32444 |
|  4|     NESTED LOOPS                   |                     |   327 |  8618   (2)|     2 |00:00:00.42 |   32454 |  32442 |
|* 5|      HASH JOIN                     |                     |   297 |  8615   (2)|     2 |00:00:00.42 |   32452 |  32441 |
|  6|       NESTED LOOPS                 |                     |    80 |  2480   (1)|     1 |00:00:00.16 |    9091 |   9083 |
|* 7|        TABLE ACCESS BY INDEX ROWID| R_WORKER            |    66 |     3   (0)|     1 |00:00:00.11 |       4 |      4 |
|* 8|         INDEX UNIQUE SCAN          | R_WR_PIN_PK_IDX     |       |     2   (0)|     1 |00:00:00.01 |       3 |      3 |
|* 9|        TABLE ACCESS FULL           | R_CV                |    14 |  2477   (1)|     1 |00:00:00.05 |    9087 |   9079 |
|*10|       TABLE ACCESS FULL            | R_CV_EDU_HIST       |   46M|  6124   (2)| 2710K|00:00:00.18 |   23361 |  23358 |
|*11|      INDEX UNIQUE SCAN             | G_TIT_ID_TIT_PK_IDX |       |     0   (0)|     2 |00:00:00.01 |       2 |      1 |
| 12|     TABLE ACCESS BY INDEX ROWID    | G_TITLE             |    10 |     1   (0)|     2 |00:00:00.01 |       2 |      2 |
-------------------------------------------------------------------------------------------------------------------------------------------
```

Snížení náročnosti dotazu je značné, co se týče nákladu a doby exekuce dotazu. Z exekučního plánu je také možné identifikovat změnu operací spojení, které jsou nyní vzestupné. První viditelnou změnou je operace s číslem 6 „*NESTED LOOPS*“, mezi řádky tabulky R_WORKER, které byly přímo načteny pomocí operace číslo 7 „*TABLE ACCESS BY INDEX ROWID*“, s tabulkou R_CV. Výsledek spojení je následně spojen s tabulkou R_CV_EDU_HIST pomocí operace číslo 5 „*HASH JOIN*“. Tento výsledek je poté spojen pomocí operace číslo 4 „*NESTED LOOPS*“ s výsledkem přečtení indexu nad primárním sloupcem tabulky G_TITLE. Nakonec je i tento výsledek spojen pomocí operace s číslem 3 „*NESTED LOOPS*“ se skutečným řádkem tabulky G_TITLE, který byl načten pomocí operace „*TABLE ACCESS BY INDEX ROWID*“.

Náročnost dotazu je snížena z 39 853 na 8 619. Doba exekuce byla snížena z jednotek vteřin na stovky milisekund. Tento plán má tedy zatím nejnižší nákladovost i dobu exekuce.

4.8.6 Optimalizace pomocí indexu

Pro další snížení nákladovosti je možné vytvořit indexy pro sloupce definované v podmínkách filtrace dotazu.

```
SQL> CREATE INDEX RIS.R_CV_PIN_IDX on RIS.R_CV(PIN);

Index created.

SQL> CREATE INDEX RIS.R_CEH_ID_CV_IDX on RIS.R_CV_EDU_HIST(ID_CV);

Index created.
```

Následuje inicializační spuštění dotazu pro ověření exekučního plánu.

```
Plan hash value: 3232848921

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                |E-Bytes| Cost (%CPU)|   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                     |       |    13 (100)|00:00:00.01 |      16 |
|*  1 |  VIEW                                    |                     |  3540 |    13   (8)|00:00:00.01 |      16 |
|*  2 |   WINDOW SORT PUSHED RANK                |                     |   327 |    13   (8)|00:00:00.01 |      16 |
|   3 |    NESTED LOOPS                          |                     |   327 |    12   (0)|00:00:00.01 |      16 |
|   4 |     NESTED LOOPS                         |                     |   327 |    12   (0)|00:00:00.01 |      14 |
|   5 |      NESTED LOOPS                        |                     |   297 |     9   (0)|00:00:00.01 |      12 |
|   6 |       NESTED LOOPS                       |                     |    80 |     6   (0)|00:00:00.01 |       8 |
|*  7 |        TABLE ACCESS BY INDEX ROWID       | R_WORKER            |    66 |     3   (0)|00:00:00.01 |       4 |
|*  8 |         INDEX UNIQUE SCAN                | R_WR_PIN_PK_IDX     |       |     2   (0)|00:00:00.01 |       3 |
|*  9 |        TABLE ACCESS BY INDEX ROWID BATCHED| R_CV                |    14 |     3   (0)|00:00:00.01 |       4 |
|* 10 |         INDEX RANGE SCAN                 | R_CV_PIN_IDX        |       |     2   (0)|00:00:00.01 |       3 |
|* 11 |       TABLE ACCESS BY INDEX ROWID BATCHED | R_CV_EDU_HIST       |    57 |     3   (0)|00:00:00.01 |       4 |
|* 12 |        INDEX RANGE SCAN                  | R_CEH_ID_CV_IDX     |       |     2   (0)|00:00:00.01 |       3 |
|* 13 |      INDEX UNIQUE SCAN                   | G_TIT_ID_TIT_PK_IDX |       |     0   (0)|00:00:00.01 |       2 |
|  14 |     TABLE ACCESS BY INDEX ROWID          | G_TITLE             |    10 |     1   (0)|00:00:00.01 |       2 |
-----------------------------------------------------------------------------------------------------------------------
```

Pro upravený dotazu jsou indexy významně efektivní. Změny v exekučním plánu se týkají pouze změny fyzického přístupu. Konkrétně se jedná o operace „*INDEX RANGE SCAN*“, které prochází vytvořené indexy a hledají řádky, jejichž sloupce jsou využity pro filtraci nebo operace spojení. Následně jsou identifikátory řádků předány pro provedení operací „*TABLE ACCES BY INDEX ROWID BATCHED*“, které naleznou množinu skutečných řádků v datových blocích daných tabulek. Snížení nákladovosti v tomto případě pouze o 8 tisíc jednotek a snížení doby trvání na jednotky milisekund.

4.9 Zátěžový test – ověření optimalizace

Pro ověření navržených a implementovaných optimalizačních technik je vhodné zjistit chování dotazů při zvýšené a paralelní zátěži. Toho bude dosaženo opakováním sady testů.

Tabulka č. 17 – statistika optimalizace vázanou proměnnou

Číslo procesu	Počet dotazů	Začátek	Konec	Doba běhu
12669	265	19:58:04	20:45:08	00:47:04
12670	265	19:58:04	20:45:13	00:47:09
12671	265	19:58:04	20:44:54	00:46:50
12672	265	19:58:04	20:45:06	00:47:02

Zdroj: Autor

Celková doba zátěžového testu optimalizace vázanou proměnnou je 47 minut a 9 vteřin. Oproti prvnímu zátěžovému testu je tento test kratší o 8 minut a 43 vteřin.

Tabulka č. 18 – statistika optimalizace nápovědou

Číslo procesu	Počet dotazů	Začátek	Konec	Doba běhu
13749	265	11:48:80	12:32:23	00:43:03
13750	265	11:48:80	12:31:47	00:42:27
13751	265	11:48:80	12:32:22	00:43:02
13752	265	11:48:80	12:32:39	00:43:19

Zdroj: Autor

Celková doba trvání zátěžového testu optimalizace nápovědou je 43 minut a 19 vteřin, což je oproti prvnímu zátěžovému testu snížení o 12 minut a 33 vteřin. Oproti testu optimalizace vázanou proměnnou je test kratší o 3 minuty a 50 vteřin.

Tabulka č. 19 – statistika optimalizace využitím tabulkových oddílů

Číslo procesu	Počet dotazů	Začátek	Konec	Doba běhu
29947	265	15:28:06	16:12:43	00:44:37
29948	265	15:28:06	16:12:29	00:44:23
29949	265	15:28:06	16:12:25	00:44:19
29950	265	15:28:06	16:12:35	00:44:29

Zdroj: Autor

Test optimalizace využitím tabulkových oddílů proběhl v celkové době trvání 44 minut a 37 vteřin. Oproti testu optimalizace vázanou proměnnou je doba trvání snížena o 11 minut a 15 vteřin. Oproti testu optimalizace nápovědou je však delší o 1 minutu a 18 vteřin.

Tabulka č. 20 – statistika optimalizace nápovědou a využitím tabulkových oddílů

Číslo procesu	Počet dotazů	Začátek	Konec	Doba běhu
6690	265	10:30:53	11:30:29	00:59:36
6691	265	10:30:53	11:30:36	00:59:43
6692	265	10:30:53	11:30:17	00:59:24
6693	265	10:30:53	11:30:17	00:59:24

Zdroj: Autor

Tento test optimalizace má nejdelší dobu trvání i oproti prvnímu testu, přičemž je tento test delší o 3 minuty a 51 vteřin. Oproti testu optimalizace vázanou proměnnou vzrostla doba trvání o 12 minut a 34 vteřin. Doba testu optimalizace nápovědou byla oproti tomuto testu kratší o 16 minut a 24 vteřin. V porovnání s testem optimalizace pomocí tabulkových oddílů je tento test delší o 15 minut a 6 vteřin.

Tabulka č. 21 – statistika optimalizace transkripcí dotazu

Číslo procesu	Počet dotazů	Začátek	Konec	Doba běhu
5595	265	22:51:35	22:53:56	00:02:21
5596	265	22:51:35	22:54:03	00:02:28
5597	265	22:51:35	22:53:48	00:02:13
5598	265	22:51:35	22:53:58	00:02:23

Zdroj: Autor

Doba trvání testu optimalizace transkripcí dotazu se pohybuje oproti předchozím testům v jednotkách minut, konkrétně v nejdelší době trvání 2 minut a 28 vteřin. Nejmenší rozdíl je oproti testu optimalizace nápovědou, a to o 40 minut a 51 vteřin.

Tabulka č. 22 – statistika optimalizace indexací

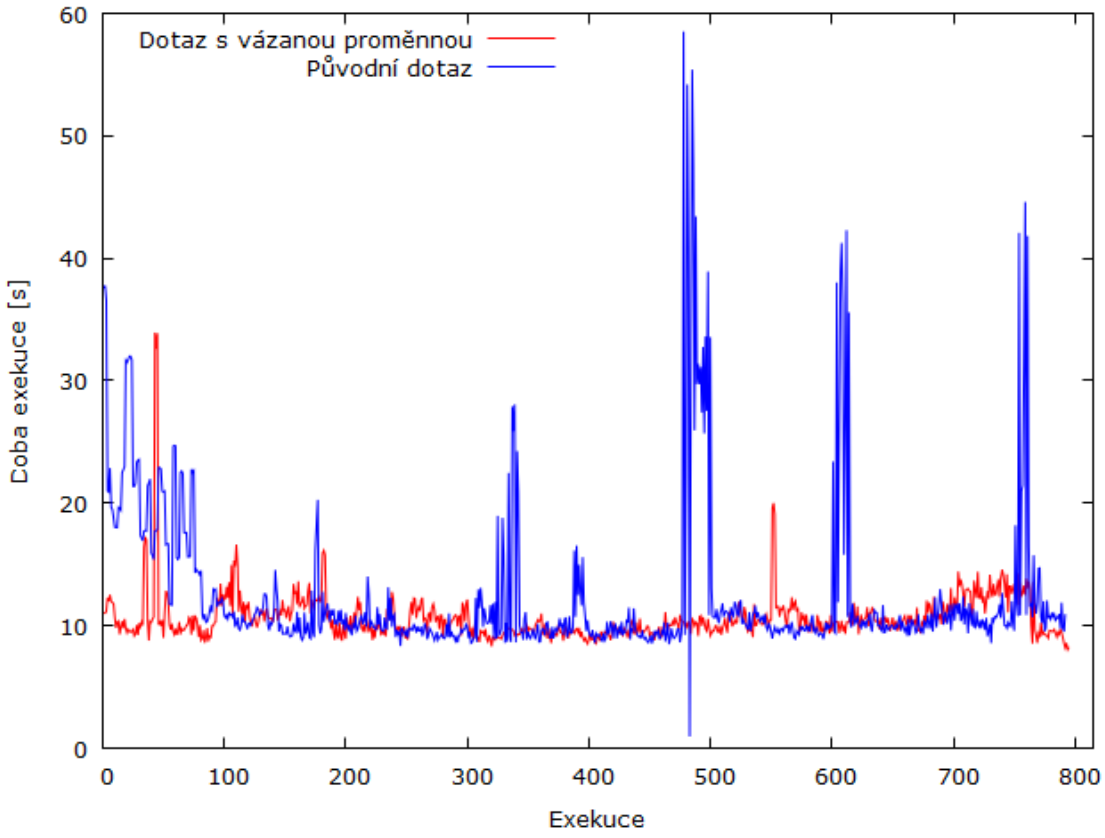
Číslo procesu	Počet dotazů	Začátek	Konec	Doba běhu
18908	265	22:17:09	22:17:10	00:00:01
18909	265	22:17:09	22:17:10	00:00:01
18910	265	22:17:09	22:17:10	00:00:01
18911	265	22:17:09	22:17:10	00:00:01

Zdroj: Autor

Poslední test, tedy test optimalizace indexací je ze všech testů nejkratší. Celková doba trvání je přibližně 1 vteřina. Rozdíl oproti testu optimalizace transkripcí je 2 minut a 27 vteřin.

4.9.1 Komparace doby exekuce dotazů optimalizačních technik

Obrázek č. 5 - Doba exekuce původního dotazu a dotazu s proměnnou



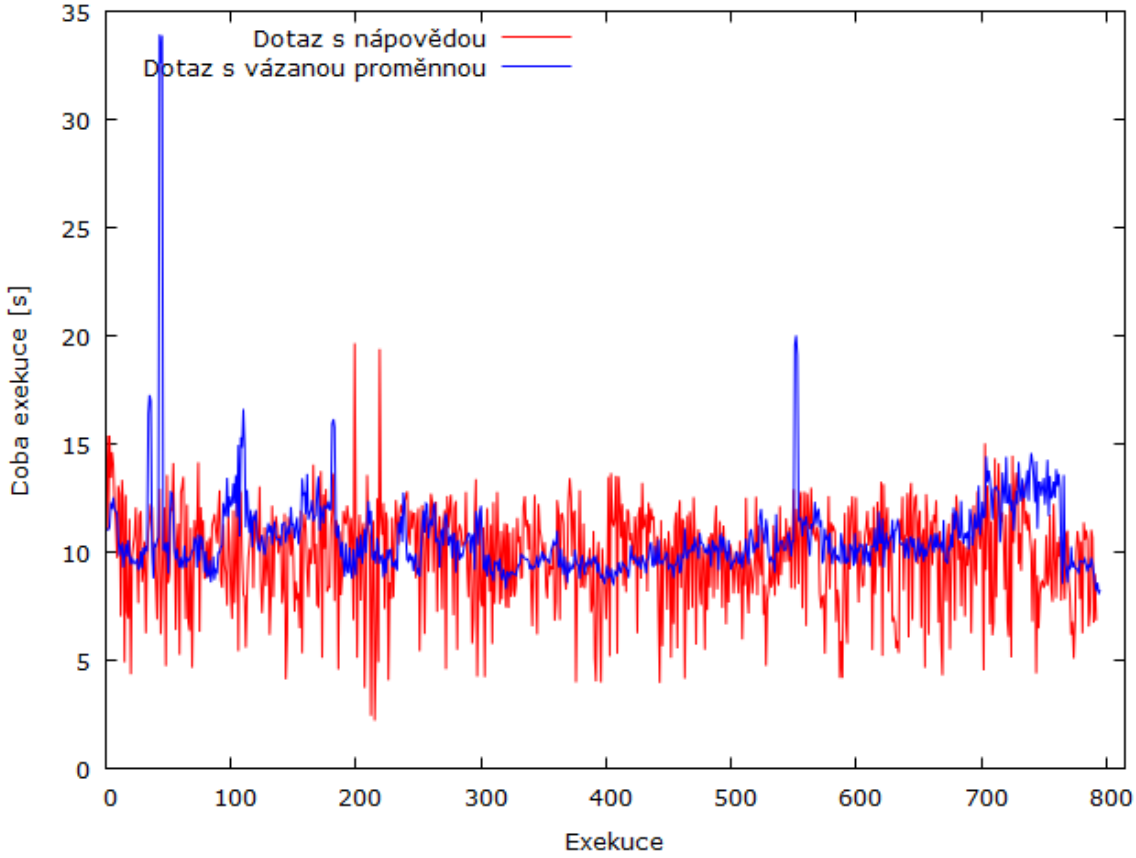
Zdroj: Autor

Obrázek č. 5 zobrazuje doby exekuce v průběhu zátěžového testu původního dotazu a dotazu s vázanou proměnnou. Z grafu je možné identifikovat průběh chování při spouštění dotazu. Původní dotaz má oproti dotazu s vázanou proměnnou značně nestabilní průběh. Objevuje se několik vrcholů s vyšším počtem exekucí. Vrcholy v intervalu exekuce 0 a 100

jsou do určité míry očekávané a běžné. Databázový systém se musí ustálit pro tento typ zátěže. Pozdější vrcholy jsou však neočekávané a silně nežádoucí. Oproti tomu exekuce dotazu s vázanou proměnnou je poměrně stabilní pouze s nízkým počtem vrcholů, které nejsou příliš vysoké a vyskytují se v počtu jednotlivých exekucí.

Optimalizace pomocí vázaných proměnných je vhodná pro zvýšení stability doby exekucí.

Obrázek č. 6 – Doba exekuce dotazu s proměnnou a dotazu s nápovědou

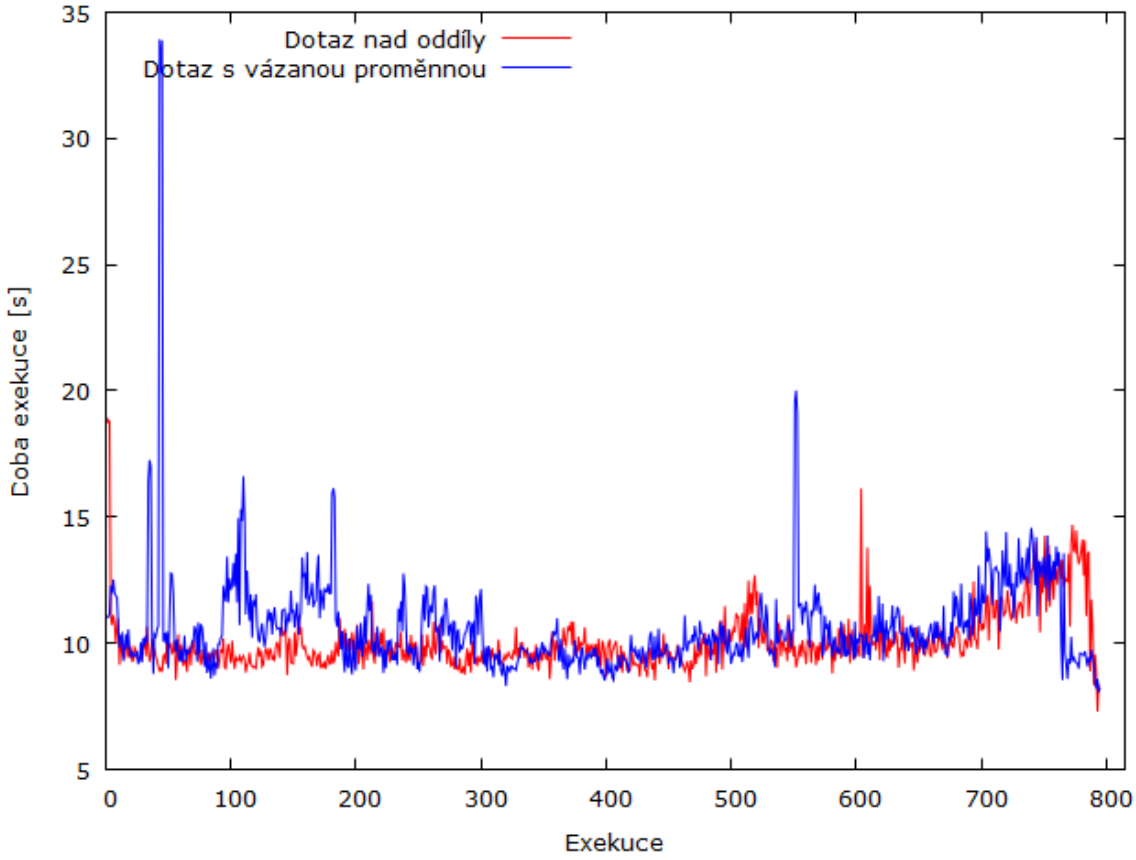


Zdroj: Autor

Obrázek č. 6 popisuje dobu exekucí dotazu s vázanou proměnnou a dotazu s nápovědou. Dotaz s nápovědou má oproti dotazu s vázanou proměnnou značně nestabilní průběh, co se týče změny doby exekucí mezi jednotlivými exekucemi. Nestabilita může být způsobena opakovaným spouštěním procesů, nedostatečným počtem jader procesoru nebo nedostatečně výkonným diskovým uložištěm. Změny se vyskytují pravidelně a doba trvání se v jejich důsledku pohybuje mezi 5 a 13 vteřinami, v extrémech až mezi 3 a 20 vteřinami.

Optimalizace pomocí nápovědy optimalizátoru tedy není příliš vhodná pro transakční systémy z důvodu nestability doby exekucí.

Obrázek č. 7 – Doba exekuce dotazu s proměnnou a dotazu nad oddíly

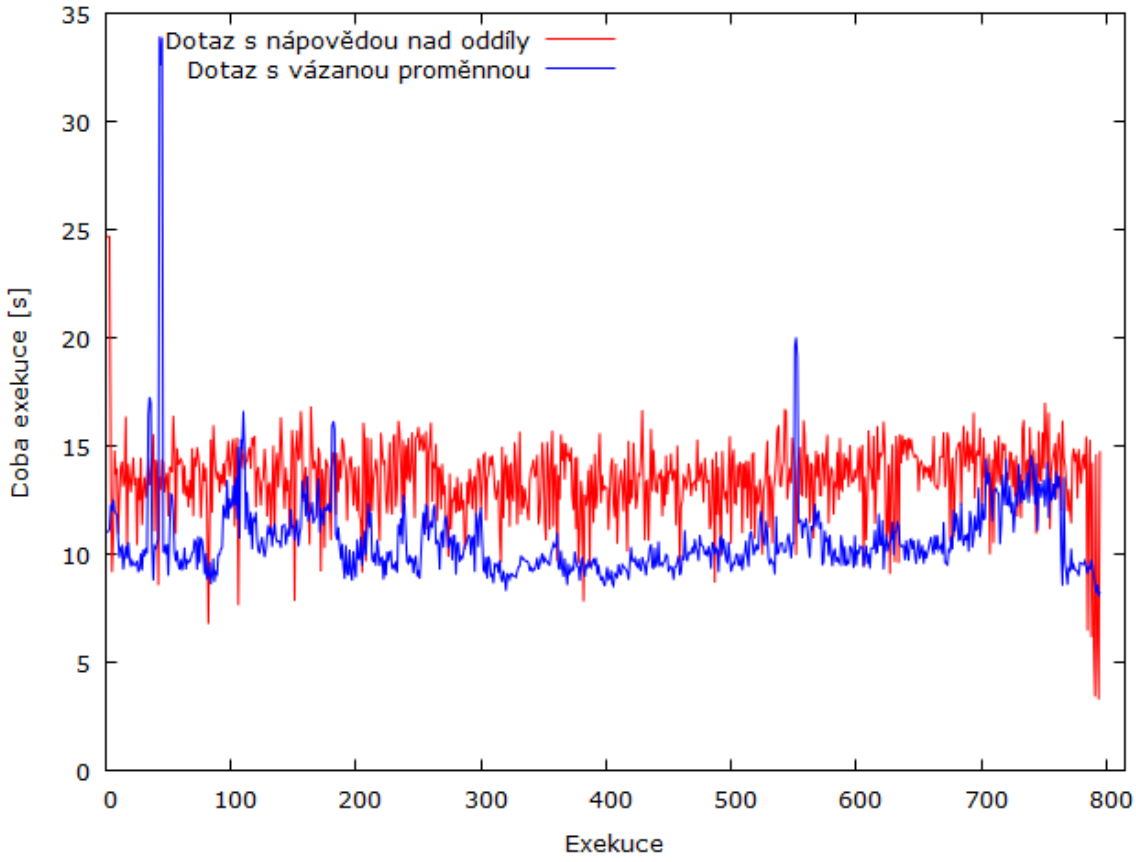


Zdroj: Autor

Obrázek č. 7 zobrazuje dobu exekucí mezi dotazy s vázanou proměnnou a dotazy nad tabulkovými oddíly. Z doby exekucí dotazu nad oddíly je možné identifikovat zvýšenou stabilitu co se týče odchýlených vrcholů. Oproti dotazu s vázanou proměnnou je možné pozorovat i zploštění mezi mírně zvýšenými vrcholy, konkrétně mezi exekucemi 100 a 200 a také mezi exekucemi 200 a 300.

Optimalizace pomocí tabulkových oddílů je tedy prospěšná pro další zlepšení stability.

Obrázek č. 8 – Doba exekuce dotazu s proměnnou a dotazu s nápovědou nad oddíly

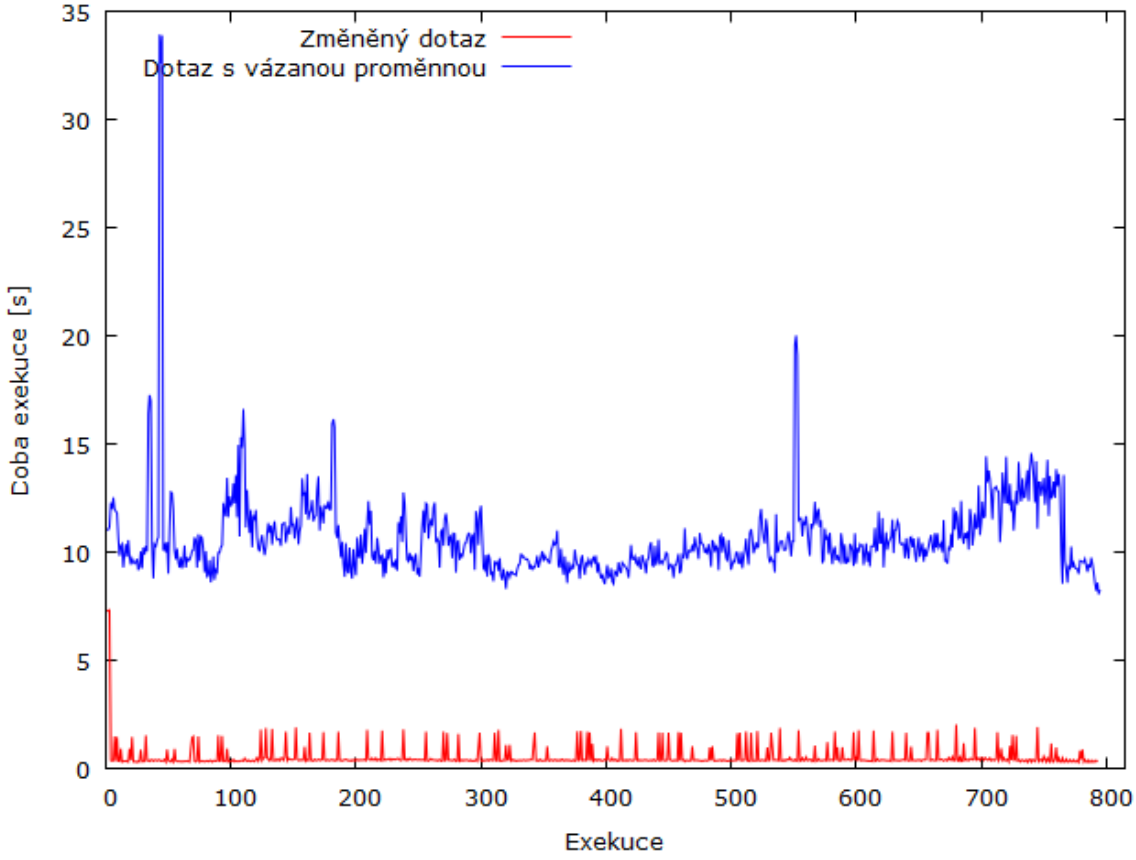


Zdroj: Autor

Obrázek č. 8 popisuje dobu exekucí dotazu s vázanou proměnnou a dotazu s nápovědou nad tabulkovými oddíly. Pro dotaz s nápovědou a tabulkovými oddíly je možné identifikovat společné charakteristiky s jednotlivými optimalizacemi pomocí nápovědy a tabulkových oddílů. Tyto charakteristiky jsou nestabilita v době trvání exekucí, kdy se doba trvání pohybuje mezi 12 a 15 vteřinami ale také charakteristika zploštění průběhu grafu bez významných vrcholů. Doba trvání se však prodloužila v důsledku paralelizace.

Optimalizace pomocí kombinace nápovědy optimalizátoru a využití tabulkových oddílů je pro tento případ použití nevhodná z důvodu nestability. Změna by nastala v případě vyššího počtu procesorových jader a v lepší distribuci dat mezi diskovým uložištěm.

Obrázek č. 9 – Doba exekuce původního dotazu s proměnnou a změněného dotazu

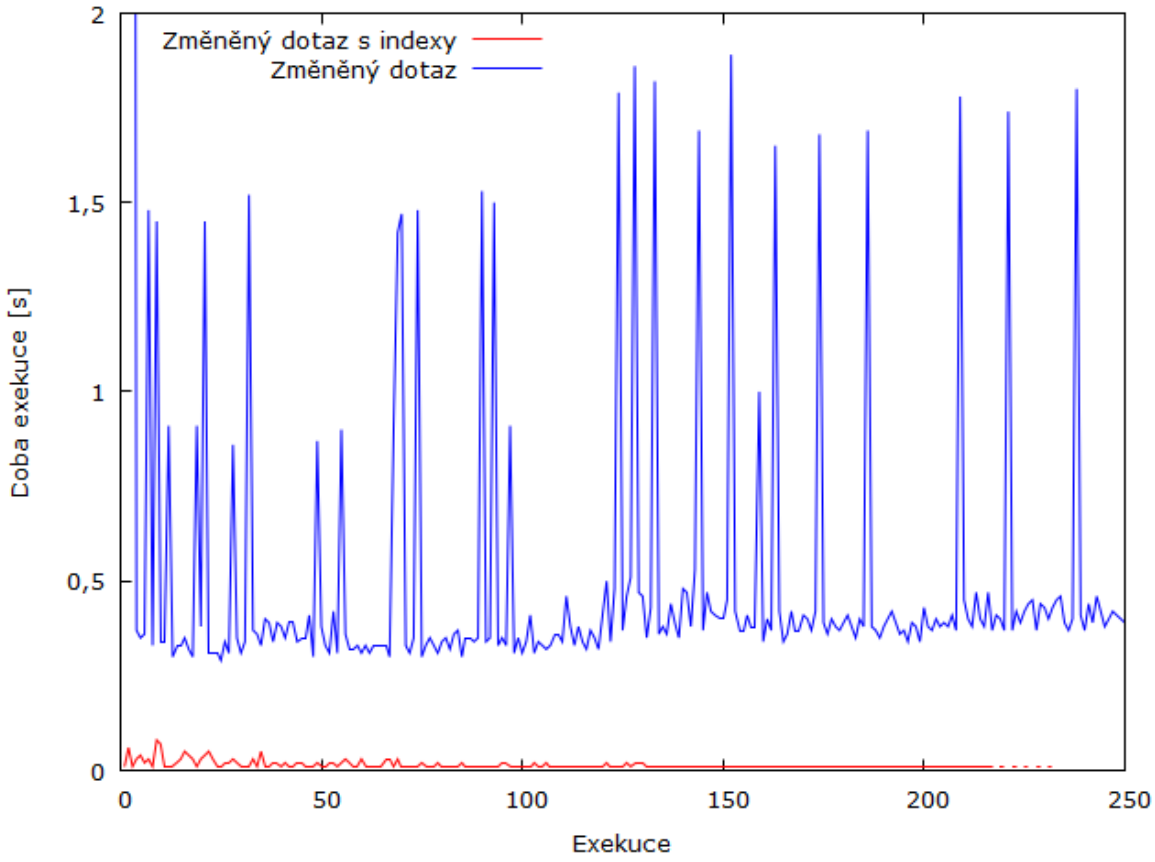


Zdroj: Autor

Obrázek č. 9 zobrazuje rozdíl mezi dobou trvání dotazu s vázanou proměnnou a změněného dotazu. Z grafu je znatelné významné zkrácení doby trvání exekucí změněného dotazu oproti dotazu s vázanou proměnnou. Doba trvání změněného dotazu se pohybuje v rozmezí 0,4 a 1,5 vteřiny bez žádných významných vrcholů ale celý průběh je mírně nestabilní. Příčinou nestability jsou operace typu „*TABLE ACCESS FULL*“, které prochází celé datové bloky tabulek a mají tendenci prodlužovat dobu trvání, pokud se hledaný řádek nachází na konci datových bloků.

Optimalizace pomocí přepisu dotazu, a tedy změny exekučního plánu je velmi efektivní za předpokladu správné identifikace a provedení. Změna exekučního plánu se silně odvíjí od typu systému, tedy zda se jedná o transakční systém nebo o datový sklad.

Obrázek č. 10 – Doba exekuce změněného dotazu a změněného dotazu s indexy



Zdroj: Autor

Obrázek č. 10 zobrazuje dobu exekuce změněného dotazu a změněného dotazu s využitím indexů. Rozsah osy exekucí je snížen z důvodu nezachycení dob exekucí nižších než 0,01 vteřiny při dokončení exekuce. Po vytvoření indexů je opět možné vidět významné doby exekucí na desítky milisekund. Po ustálení systému, tedy kolem exekuce 50, jsou doby exekucí silně stabilní bez jakýchkoliv vrcholů nebo odchylek.

Optimalizace pomocí indexace je také velmi vhodná ale je více náročná na jejich správu, co se týče údržby, kontroly stavu a použitelnosti. Zároveň je nutná kontrola správné definice omezujících podmínek v dotazech, aby docházelo k využití těchto indexů.

5 Výsledky a diskuse

Finální komparací je možné identifikovat nejlepší způsoby optimalizace. Těmi jsou využití vázané proměnné, transkripce dotazu s úpravou exekučního plánu a následná indexace. Také lze do této skupiny zařadit optimalizaci pomocí tabulkových oddílů pro zvýšení stabilitu exekučních plánů nebo pro velké datové objemy.

Naopak nápověda paralelizace pro optimalizátor je značně nestabilní způsob optimalizace. Všeobecně je doporučeno nápovědy spíše nevyužívat, nicméně existují i případy kdy je nápověda vhodná.

Nápověda optimalizátoru je nicméně nejrychlejší možností, jak optimalizovat alespoň na dobu, než je navrženo jiné řešení. Vázané proměnné a transkripce dotazu vyžadují zásah do aplikačního kódu. Indexace a vytvoření tabulkových oddílů vyžaduje zásah do struktury databázových objektů.

5.1 Zobecnění řešení

Z důvodu vysokého množství různých případů použití není možné přesné výsledky optimalizace zobecnit. Pro každý případ použití budou ideální jiné optimalizační techniky. Zobecnit je tedy možné pouze proces optimalizace.

První krokem optimalizace je identifikace problematického dotazu. Toho lze docílit zpětnou vazbou od uživatelů ale také proaktivním monitorováním stavu systému. Po identifikaci následuje analýza statistik systému ale také samotného dotazu.

Druhým krokem je kontrola stavu databáze vůči běžně dopouštěným chybám, které lze nalézt v dokumentaci pro daný databázový systém. Není vhodné kontrolovat vůči dokumentacím jiných databázových systémů. Koncepty různých databázových systémů se mohou velmi významně lišit.

Ve třetím kroku bývá navržen konceptuální model chování systému, sloužící ke korektní identifikaci příčin problému.

Čtvrtý kro je návrh a implementace konkrétních změn systému nebo datového modelu. Může se jednat o návrhy indexace sloupců, změn fyzické struktury objektů, změn diskového

uložiště, změn chování aplikace vůči databázovému systému, změn parametrů objektů nebo systému. Po implementaci je vhodné ověřit přínosnost změny.

V pátém kroku je nutné ověřit, zda bylo dosaženo předem definovaných cílů optimalizace.

Proces optimalizace je doporučeno opakovat od třetího kroku, dokud není dosaženo cílů nebo dokud není dosaženo omezení. Do tohoto cyklu je však nutné začlenit i část prvního kroku, konkrétně kontrolu statistik systému. Každá změna může odhalit nová úzká hrdla. Při identifikaci jiného úzkého hrdla je v podstatě nutné i validovat model chování systému.

5.2 Diskuze

Na proces optimalizace a jeho dosažené výsledky má vliv několik faktorů. Prvním faktorem jsou zkušenosti a znalosti aplikačních architektů, vývojářů a aplikačních a databázových administrátorů. Od zkušeností a znalostí se odvíjí průběh procesu optimalizace. Kritickými oblastmi jsou analýza problematických dotazů, rozpoznání symptomů statistik běhu systémů a návrh změn pro dosažení optimalizace.

Druhým faktorem je dostupnost optimalizačních technik. Techniky jsou dostupné k používání pomocí různých politik. Existují databázové systémy, kde je nutné zakoupit licenci pro použití pokročilých technik. Příkladem je Oracle Database a DB2. Jiné databázové systémy (například PostgreSQL) pokročilé techniky přímo nenabízí a spoléhají se pouze na komunitní vývojáře. Tento faktor je ovlivněn finančními zdroji společnosti a také mírou rizika na které je společnost ochotna přistoupit.

Třetím faktorem je alokace zdrojů pro optimalizaci. Společnost musí alokovat dostatek lidských i finančních zdrojů pro proces optimalizace. Stále se objevují společnosti, které nedávají procesu optimalizace dostatečný důraz.

Navržené řešení je pouze jednou z několika možností, jak procesem optimalizace dosáhnout definovaného cíle. Dalšími řešeními může být úprava zdrojů serverů, úprava parametrů datových uložišť nebo úprava parametrů databázového a operačního systému.

Mírně odlišnou možností optimalizace jsou databáze běžící na více serverech, nazvané clustery. V tomto případě se nejedná o snížení nákladovosti a doby exekuce jednotlivých dotazů, ale spíše o rozložení zátěže celého rozsahu aplikačních dotazů. Díky rozložení zátěže jsou směrovány aplikační přístupy na různé servery, které následně využijí zdrojů na těchto rozdílných serverech.

Závěr

Hlavním cílem této diplomové práce bylo navrhnout postup optimalizace přístupu k datům, který povede ke snížení časové a zdrojové náročnosti přístupu na základě požadavků identifikovaných v teoretické části. Dílčími cíli bylo seznámení s principy a kritérii optimalizace náročnosti přístupu k datům. Dalšími dílčími cíli bylo zmapování momentálního stavu, navržení a ověření řešení na vzorové databázové evidenci. Navržené řešení mělo být zobecněno pro další použití.

V praktické části byly identifikovány problémy přístupu k datům v sociálně známých společnostech, jež měly mediální dopady. Poté byl prezentován příklad optimalizace z reálného produkčního prostředí nejmenované společnosti.

Pro účely analýzy a ověření optimalizačních technik byl vytvořen datový model na němž byly identifikovány aplikační dotazy. Datový model reprezentuje problematiku správy životopisů. Následoval zátěžový test, na jehož základě se identifikoval problematický dotaz.

Problematický dotaz následně prošel řadou optimalizačních technik, přičemž pro každou techniku bylo nutné analyzovat exekuční plán. Plány byly poté porovnány s původním exekučním plánem v oblastech nákladovosti, době trvání a použitých zdrojích.

Samotné ověření optimalizace dotazů probíhalo individuálně zopakováním zátěžového testu a monitorováním stejných statistik a parametrů jako v případě prvního testu. Výsledky všech testů bylo možné porovnat a ověřit prospěšnost optimalizace.

Přestože jsou relační databázové systémy hojně využívané a rozšířené, stále lze nalézt značně neoptimální datové přístupy. Pravděpodobnou příčinou je neznalost nebo nezkušenost vývojářů aplikací, architektů a v neposlední řadě i databázových administrátorů. Poměrně častými případy je nedostatečná nebo přehlcená indexace, neoptimální exekuční plány nebo nevhodný návrh datového modelu. Z těchto důvodů je absolutně nutná znalost používaného databázového systému, jeho metod optimalizace případně i principy nejlepších praktik. Stále existují aplikace, které jsou nechvalně proslulé svou špatnou odezvou a vysokou náročností.

Jednotlivé optimalizační techniky však nelze příliš generalizovat. Použitelné a efektivní techniky jsou závislé na datovém modelu, dotazech, databázových parametrech,

databázovém systému a konfiguraci serveru. Oproti tomu je možné zobecnit proces optimalizace, který se prochází fázemi identifikace problematických oblastí, jejich analýza, navržení optimalizace, její ověření a v případě nutnosti návrat k návrhu.

Přínosem diplomové práce je ověření aplikovatelnosti různých optimalizačních technik a jejich komparace. Aplikovatelnost lze pojmout jako splnění cílů optimalizace pomocí dosažení nízké náročnosti na výpočetní výkon, na diskové uložiště a na paměťové struktury v exekučních plánech dotazů při zachování stejných výstupních dat.

Dalším přínosem je také analýza chování dotazu pro rozdílných technikách optimalizace. Každá optimalizační technika má své kladné a záporné stránky, které byly do určité míry prezentovány na stabilitě a době exekuce dotazů.

V této diplomové práci byl navržen postup optimalizace přístupu k datům, který vedl ke snížení časové a zdrojové náročnosti přístupu na základě požadavků identifikovaných v teoretické části. Bylo provedeno seznámení s principy a kritérii optimalizace náročnosti přístupu k datům. Byl zmapován momentální stav a bylo navrženo a ověřeno řešení na vzorové databázové evidenci. Navržené řešení bylo z hlediska procesu optimalizace zobecněno pro další použití.

6 Seznam použitých zdrojů

1. **Roeser, Mary Beth;.** SQL Language Reference. *Oracle Help Center.* [Online] April 2017. [Citace: 29. Červen 2019.] https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/. E83703-01.

2. **Laurenčík, Marek.** *SQL - Podrobný průvodce uživatele.* Praha : Grada, 2018. 9788027107742.

3. **Ashdown, Lance a Kyte, Tom.** Database Concepts. *Oracle Help Center.* [Online] Březen 2018. [Citace: 25. Únor 2020.] https://docs.oracle.com/en/database/oracle/oracle-database/12.2/cncpt. E85769-04.

4. **Feuerstein, Steven a Pribyl, Bill.** *Oracle PL/SQL Programming.* 6th. Sebastopol : O´Reilly Media Inc., 2014. ISBN 978-1-449-32445-2.

5. **Microsoft.** SQL Docs. *Microsoft.* [Online] 2019. [Citace: 22. Únor 2020.] https://docs.microsoft.com/en-us/sql/sql-server.

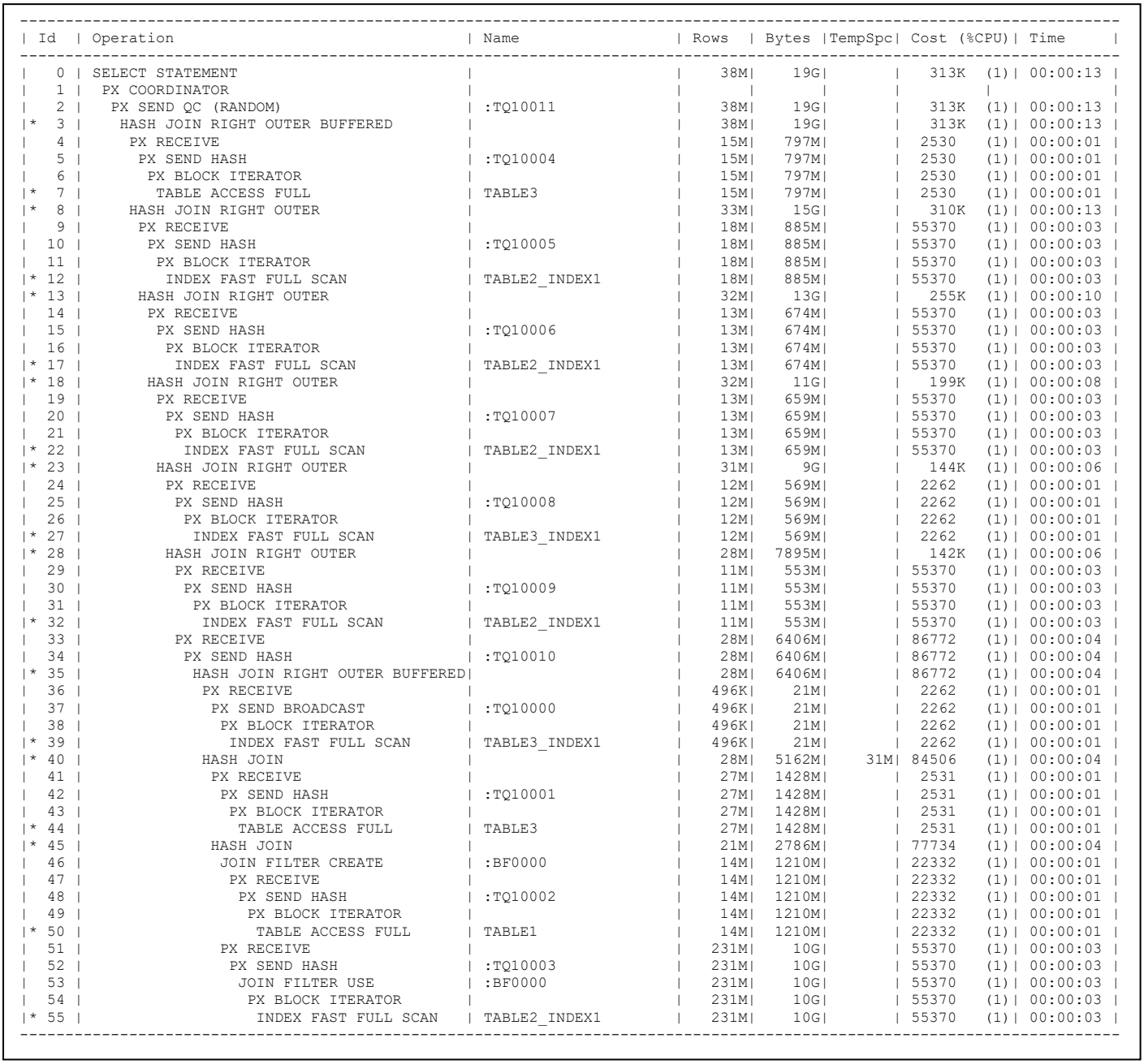
6. **Ashdown, Lance.** SQL Tuning Guide. *Oracle Help Center.* [Online] 2019. [Citace: 17. Únor 2020.] https://docs.oracle.com/en/database/oracle/oracle-database/12.2/tgsql. E85762-05.

7. **Momjian, Bruce.** *PostgreSQL: introduction and concepts.* Brno : Computer Press, 2003. 80-7226-954-2.

8. **Belden, Eric.** VLDB and Partitioning Guide. *Oracle Help Center.* [Online] 7 2017. [Citace: 20. Únor 2020.] https://docs.oracle.com/en/database/oracle/oracle-database/12.2/vldbg/. E85722-01.

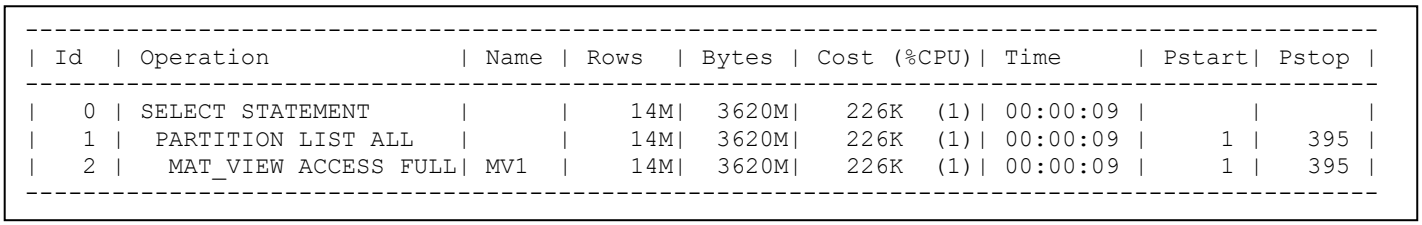
9. **Bhatiya, Rajesh, Chan, Immanuel a Ashdown, Lance.** Database Performance Tuning Guide. *Oracle Help Center.* [Online] Leden 2019. [Citace: 20. Únor 2020.] https://docs.oracle.com/en/database/oracle/oracle-database/12.2/tgdba/. E85600-04.

7 Přílohy



Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		38M	19G		313K (1)	00:00:13
1	PX COORDINATOR						
2	PX SEND QC (RANDOM)	:TQ10011	38M	19G		313K (1)	00:00:13
* 3	HASH JOIN RIGHT OUTER BUFFERED		38M	19G		313K (1)	00:00:13
4	PX RECEIVE		15M	797M		2530 (1)	00:00:01
5	PX SEND HASH	:TQ10004	15M	797M		2530 (1)	00:00:01
6	PX BLOCK ITERATOR		15M	797M		2530 (1)	00:00:01
* 7	TABLE ACCESS FULL	TABLE3	15M	797M		2530 (1)	00:00:01
* 8	HASH JOIN RIGHT OUTER		33M	15G		310K (1)	00:00:13
9	PX RECEIVE		18M	885M		55370 (1)	00:00:03
10	PX SEND HASH	:TQ10005	18M	885M		55370 (1)	00:00:03
11	PX BLOCK ITERATOR		18M	885M		55370 (1)	00:00:03
* 12	INDEX FAST FULL SCAN	TABLE2_INDEX1	18M	885M		55370 (1)	00:00:03
* 13	HASH JOIN RIGHT OUTER		32M	13G		255K (1)	00:00:10
14	PX RECEIVE		13M	674M		55370 (1)	00:00:03
15	PX SEND HASH	:TQ10006	13M	674M		55370 (1)	00:00:03
16	PX BLOCK ITERATOR		13M	674M		55370 (1)	00:00:03
* 17	INDEX FAST FULL SCAN	TABLE2_INDEX1	13M	674M		55370 (1)	00:00:03
* 18	HASH JOIN RIGHT OUTER		32M	11G		199K (1)	00:00:08
19	PX RECEIVE		13M	659M		55370 (1)	00:00:03
20	PX SEND HASH	:TQ10007	13M	659M		55370 (1)	00:00:03
21	PX BLOCK ITERATOR		13M	659M		55370 (1)	00:00:03
* 22	INDEX FAST FULL SCAN	TABLE2_INDEX1	13M	659M		55370 (1)	00:00:03
* 23	HASH JOIN RIGHT OUTER		31M	9G		144K (1)	00:00:06
24	PX RECEIVE		12M	569M		2262 (1)	00:00:01
25	PX SEND HASH	:TQ10008	12M	569M		2262 (1)	00:00:01
26	PX BLOCK ITERATOR		12M	569M		2262 (1)	00:00:01
* 27	INDEX FAST FULL SCAN	TABLE3_INDEX1	12M	569M		2262 (1)	00:00:01
* 28	HASH JOIN RIGHT OUTER		28M	7895M		142K (1)	00:00:06
29	PX RECEIVE		11M	553M		55370 (1)	00:00:03
30	PX SEND HASH	:TQ10009	11M	553M		55370 (1)	00:00:03
31	PX BLOCK ITERATOR		11M	553M		55370 (1)	00:00:03
* 32	INDEX FAST FULL SCAN	TABLE2_INDEX1	11M	553M		55370 (1)	00:00:03
33	PX RECEIVE		28M	6406M		86772 (1)	00:00:04
34	PX SEND HASH	:TQ10010	28M	6406M		86772 (1)	00:00:04
* 35	HASH JOIN RIGHT OUTER BUFFERED		28M	6406M		86772 (1)	00:00:04
36	PX RECEIVE		496K	21M		2262 (1)	00:00:01
37	PX SEND BROADCAST	:TQ10000	496K	21M		2262 (1)	00:00:01
38	PX BLOCK ITERATOR		496K	21M		2262 (1)	00:00:01
* 39	INDEX FAST FULL SCAN	TABLE3_INDEX1	496K	21M		2262 (1)	00:00:01
* 40	HASH JOIN		28M	5162M	31M	84506 (1)	00:00:04
41	PX RECEIVE		27M	1428M		2531 (1)	00:00:01
42	PX SEND HASH	:TQ10001	27M	1428M		2531 (1)	00:00:01
43	PX BLOCK ITERATOR		27M	1428M		2531 (1)	00:00:01
* 44	TABLE ACCESS FULL	TABLE3	27M	1428M		2531 (1)	00:00:01
* 45	HASH JOIN		21M	2786M		77734 (1)	00:00:04
46	JOIN FILTER CREATE	:BF0000	14M	1210M		22332 (1)	00:00:01
47	PX RECEIVE		14M	1210M		22332 (1)	00:00:01
48	PX SEND HASH	:TQ10002	14M	1210M		22332 (1)	00:00:01
49	PX BLOCK ITERATOR		14M	1210M		22332 (1)	00:00:01
* 50	TABLE ACCESS FULL	TABLE1	14M	1210M		22332 (1)	00:00:01
51	PX RECEIVE		231M	10G		55370 (1)	00:00:03
52	PX SEND HASH	:TQ10003	231M	10G		55370 (1)	00:00:03
53	JOIN FILTER USE	:BF0000	231M	10G		55370 (1)	00:00:03
54	PX BLOCK ITERATOR		231M	10G		55370 (1)	00:00:03
* 55	INDEX FAST FULL SCAN	TABLE2_INDEX1	231M	10G		55370 (1)	00:00:03

Příloha č. 1 – exekuční plán původního dotazu z příkladu optimalizace



```
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |    14M|  3620M|   226K  (1)| 00:00:09 |       |       |
|   1 |  PARTITION LIST ALL   |      |    14M|  3620M|   226K  (1)| 00:00:09 |     1 |   395 |
|   2 |   MAT_VIEW ACCESS FULL| MV1  |    14M|  3620M|   226K  (1)| 00:00:09 |     1 |   395 |
-------------------------------------------------------------------------------------------
```

Příloha č. 2 – exekuční plán materializovaného pohledu z příkladu optimalizace