Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Optimalizace SQL dotazů v databázovém systému Oracle

Diplomová práce

Autor: Bc. Ondřej Sucharda

Studijní obor: N1802 Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2016

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 21. 4. 2015 Bc. Ondřej Sucharda

Poděkování

Touto cestou bych chtěl poděkovat doc. Ing. Filipu Malému, Ph.D. za odborný dohled a cenné rady, které mi poskytl při zpracování diplomové práce.

Anotace

Diplomová práce se zabývá optimalizacemi SQL dotazů v databázovém systému Oracle a související problematikou. Popisuje možnosti optimalizace na několika vrstvách, přičemž největší důraz klade na design databáze a SQL dotazy. Práce se dále věnuje optimalizátoru, který je nezbytnou součástí celého procesu optimalizace a technikám pro jeho ovlivňování. Následně je proces optimalizace demonstrován na několika vybraných SQL dotazech, kde je čtenář seznámen s jednotlivými kroky optimalizace od identifikace problémových dotazů až po zátěžové testování a vyhodnocení výsledků.

Annotation

Title: SQL query optimization in Oracle database

The diploma thesis deals with the optimizing SQL queries in Oracle database system and related issues. It describes the optimization options on several layers with the greatest emphasis on database design and SQL queries. The thesis is dedicated to optimizer, which is the essential part of the whole optimization process and techniques for its influencing. Subsequently, the optimization process is demonstrated on several selected SQL queries, where the reader is made familiar with the individual steps of the optimization from the identification of problematic queries to the load testing and evaluation of the results.

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

Seznam kódů

Seznam příloh

Příloha č. 1: dotaz1/sql/*.sql

Příloha č. 2: dotaz1/data.xlsx

Příloha č. 3: dotaz2/sql/.*sql

Příloha č. 4: dotaz2/data.xlsx

Příloha č. 5: dotaz3/sql/.*sql

Příloha č. 6: dotaz3/data.xlsx

1 Úvod

V dnešním světě IT je proces zrychlování a zvyšování výkonu každodenní záležitostí. Nejinak je tomu u databázových systémů, které jsou využívány v převážné většině aplikací pracujících s daty. Databázové systémy se využívají v nejrůznějších oblastech, jako jsou elektronické obchody, informační systémy či v oblastech, kde se pracuje s obrovským množstvím dat, např. Business Intelligence nebo Data Mining. Složitost samotných systémů je někdy až abnormální a při práci s větším množstvím dat je výkon databáze velmi dobře rozpoznatelný. Rychlost přístupu k datům udává v dnešní době konkurenční výhodu.

Při práci s databází existuje mnoho možností jak zvýšit výkon. Lze jej ovlivnit na hardwarové úrovni (přidáním větší paměti, rychlejších disků atp.), pomocí konfigurace databázového systému (úpravou inicializačních parametrů), při designu datového modelu, správnou implementací dotazů, indexů, používáním materializovaných pohledů, apod. Tato práce je zaměřena především na optimalizaci SQL dotazů. Nicméně zde budou popsány i ostatní vrstvy pro optimalizace a to z toho důvodu, že spolu v mnoha případech úzce souvisí a také proto, aby čtenář získal alespoň základní vědomosti o databázovém systému Oracle a optimalizacích obecně.

Pro optimalizaci dotazů musí mít vývojář jisté odborné znalosti a často se setkává se složitými úlohami. Pro databázové odborníky je to však výzva a optimalizace se může stát jedním z mála úkolů, kde mohou vedle každodenní rutiny naplno využít svou kreativitu a inteligenci. Moderní databázové systémy dnes dovolují téměř úplnou kontrolu nad způsobem provádění dotazu. V databázovém systém Oracle lze využívat například tzv. Hintů, kterými je možné ovlivnit pořadí zpracovávaných tabulek, určit způsob přístupu ke každé z nich nebo zvolit metody pro spojení tabulek v dotazu. Databázové systémy tak nabízejí silný nástroj, který v rukou databázového experta vyřeší spoustu složitých problémů při zpracování dat.

Tato diplomová práce si klade za cíl uvést čtenáře do optimalizací SQL dotazů v databázovém systému Oracle. Veškeré postupy uvedené v práci se budou týkat verze 12c. Pro správné pochopení všech náležitostí souvisejících s optimalizacemi bude nejprve čtenáři vysvětlen pojem „optimalizace“ a principy správného návrhu databáze, který s tématem velice úzce souvisí. Z tohoto důvodu budou v práci charakterizovány vybrané

databázové objekty, jako jsou tabulky, indexy, pohledy, materializované pohledy a jejich specifické typy včetně příkladů a možností jejich použití. Tyto objekty lze považovat za jakési základní kameny před samotnou optimalizací SQL. Klíčová část práce se zabývá SQL dotazy. V této části bude vysvětlen proces jejich zpracování databázovým systémem. Kapitola se také věnuje optimalizátoru, což je komponenta starající se o výběr nejefektivnější metody vykonání příkazu. Vysvětleny budou i nástroje a techniky pro ovlivňování optimalizátoru. Poté následuje kapitola popisující nástroje pro detekci problematických SQL dotazů a možnosti jejich ladění. V poslední části práce budou získané znalosti použity na optimalizaci několika vybraných SQL dotazů, které jsou součástí aplikace používané v produkčním prostředí již řadu let. Tím bude čtenáři představen postup při optimalizaci SQL v praktickém užití.

2 Základní pojmy

Tato kapitola je věnována pojmům, které se v této práci vyskytnou a nejsou vysvětleny v samotných kapitolách, je proto nutné mít o těchto výrazech alespoň základní znalosti.

Databáze a instance

Autoři Bryla a Loney [1, s. 26-27] definují databázi jako kolekci dat, která je fyzicky uložena na databázovém serveru v jednom nebo i více souborech. Soubory jsou rozděleny na databázové a nedatabázové, podle dat, která obsahují. Databázové soubory obsahují databázová data a metadata, nedatabázové soubory obsahují inicializační parametry, protokolovací informace atp. Samotná databáze je uložena na pevném disku serveru, přičemž instance databáze je pouze v operační paměti serveru.

Instance je složena z bloku paměti, který je vyhrazený v systémové globální oblasti (SGA) a dále z procesů, které jsou spuštěné na pozadí a komunikují s oblastí SGA a databázovými soubory na disku [1, s. 26-27].

SGA

SGA (System Global Area, někdy též Shared Global Area) je skupina sdílených paměťových struktur, které obsahují data a řídící informace pro běh instance [2]. Každá instance má vlastní SGA.

PGA

PGA (Program Global Area, někdy též Process Global Area) je oblast paměti, která obsahuje data a řídící informace pro proces serveru. Jedná se o nesdílenou paměť, u které může operace čtení a zápis provádět pouze databázový systém Oracle [2].

DML

DML (Data Manipulation Language) je jazyk sloužící k úpravě dat v rámci stávajících tabulek [3]. Příkladem jsou příkazy INSERT, DELETE, UPDATE a MERGE.

DDL

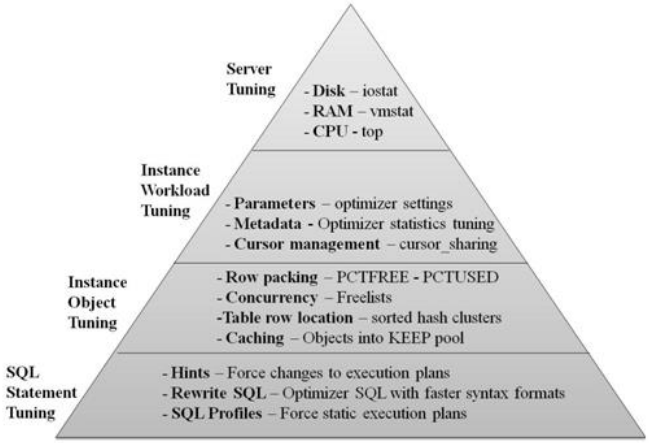
DDL (Data Definition Language) je jazyk používaný pro správu databáze [3]. Umožňuje definovat databázi a specifikovat strukturu, datové typy, a vazby. Jedná se o příkazy CREATE, ALTER, DROP.

3 Úvod do optimalizací

Pro začátek je velmi důležité pochopit pojem „optimalizace“, jaký je její účel a jaká kritéria se při optimalizaci zkoumají. Pod pojmem optimalizace si lze představit jakékoli úkony zvyšující efektivitu a snižující náklady databázového systému. Jedná se především o minimalizaci nákladů na zdrojový čas a kapacitu paměti.

3.1 Vrstvy pro optimalizace

Optimalizovat databázi lze na několika úrovních (viz obrázek 1) a každou úroveň má většinou na starosti jiná osoba.



Obrázek 1: Vrstvy pro optimalizace v DBS Oracle [28]

I přesto, že se tato práce zabývá především vrstvou pro optimalizaci SQL, tak zde budou povrchově popsány i ostatní vrstvy, aby bylo jasné, co která vrstva řeší a jakým způsobem je lze optimalizovat. Vrstvy jsou hierarchicky seřazené, podle priority a zároveň pořadí, v jakém konfigurace probíhá.

Optimalizace DB serveru

Databázový server je první místo, které je potřeba vždy správně optimalizovat. Optimalizace na této vrstvě spočívá ve správné konfiguraci a často je navržena již při plánování samotné databáze. Při konfiguraci serveru je potřeba vzít v úvahu přibližný počet uživatelů přistupující k databázi, architekturu (nejčastěji klient-server), která bude použita, úroveň zabezpečení (většinou použité některé z RAID disků) [4]. Vzhledem k postupně narůstající velikosti databáze a k navyšujícímu se počtu uživatelů přistupujících k databázi, většinou přestávají být systémové prostředky dostačující. Proto je nutné

sledovat výkon databáze i na této vrstvě, a pokud je to nutné, tak učinit investice do HW komponent.

Optimalizace databáze

Optimalizací na této vrstvě především rozumíme správné nastavení globálních parametrů, které mají vliv na instanci databáze [1, s. 26-27]. Nastavují se zde například inicializační parametry, které řídí konfiguraci SGA a ovlivňují celkové chování databáze.

Optimalizace databázových objektů

Jakmile je správně nakonfigurovaná databáze, je nutné začít s optimalizací databázových objektů. V této fázi je možné upravit nastavení několika parametrů, které ovlivňují chování tabulek a indexů, tudíž mohou mít i velký dopad na výkon databáze jako takové. Jak Thomas Kyte [5, s. 255-260] ve své knize uvádí, jediný parametr, který by se měl vždy uvážlivě nastavit, je parametr PCTFREE. Ten určuje, jak velký prostor by měl být v bloku rezervován pro aktualizaci tabulky. V případě, že se řádky z důvodu nedostatku volného místa nevejdou do bloku celé, musí být přesunuty do jiného bloku nebo zřetězeny, což je z hlediska výkonu nežádoucí. Pokud se předpokládá častá aktualizace tabulky, je vhodné tento parametr nastavit na vyšší hodnotu než výchozí, která je 10 %. Další dva parametry, které lze nastavit jsou PCTUSED a FREELIST (Seznam volných bloků). Ty lze však nastavit pouze při použití módu **Manual Segment Space Management** (MSSM). Jelikož počet souběžných operací nad tabulkou pro správné nastavení parametru FREELIST ve většině případů není možné odhadnout a ani určit optimální hodnotu PCTUSED, tak se tento mód příliš nepoužívá. Při špatném nastavení totiž dochází k plýtvání místem. V praxi se více používá metoda **Automatic Segment Space Management** (ASSM), která automaticky řídí vnitřní prostor datových bloků a tím eliminuje potřebu nastavení parametrů PCTUSED a FREELIST [1, s. 26-27].

Optimalizace SQL

Optimalizace na této vrstvě jsou věnovány výhradně SQL příkazům. Optimalizaci SQL lze definovat jako iterativní proces zvyšování výkonu SQL příkazů tak, aby splňovaly konkrétní měřitelné a dosažitelné cíle [6].

Protože se práce zabývá především touto vrstvou, budou jednotlivé metody a možnosti optimalizace popsány v samostatných kapitolách.

3.2 Kritéria a cíl SQL optimalizace

Při optimalizaci je nutné znát kritéria, díky kterým lze měřit výkon dotazů. Za taková kritéria jsou považovány dvě standardní metriky:

- Časová odezva – Představuje dobu mezi spuštěním dotazu a obdržením odpovědi s daty.
- Propustnost – Udává objem práce zpracované za určitý čas.

Snahou optimalizace je co nejvíce snížit časovou odezvu, tedy dobu, za kterou obdržíme požadovaná data a maximalizovat propustnost a tím snížit množství systémových prostředků (CPU, RAM, atp.) potřebných k provedení dotazu [6].

4 Návrh databáze

Správný návrh databáze velice úzce souvisí s následnou optimalizací dotazů. Pokud se špatně navrhnou tabulky nebo se neuvážlivě používají indexy, pohledy atp., pak není možné dosáhnout optimálního výkonu SQL dotazů i při jejich správném sestavení. Autoři Bob Bryla a Kevin Loney [1, s. 145-148] považují za základní problém při návrhu databáze fakt, že vývojáři neznají všechny způsoby využívání dat a procesy, které budou v databázi probíhat. Výsledkem toho pak je, že už v první verzi aplikace existují komponenty s nízkým výkonem. U jiných objektů se problémy mohou objevit až po nasazení do produkčního prostředí. Náprava některých problémů je relativně rychlá – přidá se index, upraví se inicializační parametr atp. V jiných případech nelze výkon zvýšit bez zásahu do architektury databáze, což bývá problematické a časově náročné.

Tato kapitola se proto zabývá metodami pro správný návrh databáze a obecnými zásadami, které mají přímý vliv na výkon databáze. Popsány zde budou základní objekty databáze Oracle, které je potřeba při návrhu bezprostředně znát. Jedná se především o tabulky, indexy, pohledy a materializované pohledy.

4.1 Databázové schéma

Obecně platí, že při návrhu databázového schéma by neměl být kladen důraz na dosažení teoretické dokonalosti. Pro uživatele aplikace není rozhodující, jestli byl dodržen čistě relační návrh tabulek nebo zda je struktura databáze normalizovaná do třetí nebo dokonce čtvrté normální formy. Tento návrh je sice logicky správný, ale s výjimkou prostředí OLTP je obvykle nepoužitelný. Uživatelé požadují aplikaci, která jim pomůže jejich práci dokončit v co možná nejkratším čase. Při návrhu by se tedy mělo dbát především na zjednodušení procesů, které jsou často vykonávány. Vždy je potřeba uvědomit si počet databázových operací, které budou nutné k provedení daných transakcí. Operace vkládání by měly zahrnovat co nejmenší množství tabulek, což často bývá v rozporu s plně normalizovanou formou. Ukládání dat u složitějších transakcí může obecně zahrnovat i čtení z mnoha další tabulek.

Při návrhu tabulek by se měl vývojář vždy zamyslet nad tím, jak bude k datům přistupovat uživatel. Po vyhodnocení takové otázky se může ukázat, že plně relační návrh struktury tabulek při zpracování rozsáhlých dotazů nevyhovuje. Problematické bývají především dotazy, které vracejí větší množství sloupců. Vrácené sloupce většinou nebývají pouze z jedné tabulky, ale z několika, proto je nutné v takových dotazech provádět operace

spojení. Pokud je součástí spojení příliš velká tabulka, může se výrazně snížit výkon celého dotazu.

Proto by se mělo vždy uvážit do jaké míry akceptovat logicky správné teoretické postupy. Je vhodné nejprve navrhnout strukturu tabulek ve třetí normální formě a poté provést jejich denormalizaci (viz kapitola 4.1.1).

4.1.1 Denormalizace

Nejprve je potřeba vysvětlit si pojem normalizace. Normalizace je proces odstraňování redundance dat z datového modelu, kdy jsou měněny sloupce jednotlivých tabulek a zaváděny mezi nimi vhodné vztahy. Při tomto procesu je brán ohled na řešení problémů s nekonzistencí dat. Normalizované modely se snadněji udržují, ale v mnoha případech je díky nim dosaženo nižšího výkonu. Z důvodu zvýšení výkonu databáze se pravidlo normalizace v některých případech porušuje a záměrně se provádí denormalizace datového modelu.

Denormalizace je proces opětovného zavedení redundantních dat se záměrem zvýšení výkonu databáze. Harrison Guy uvádí několik rizik, které s sebou tento proces nese (převzato a volně přeloženo z [8, s. 89-90]):

- Denormalizace sice může zvýšit výkon některých klíčových transakcí nebo dotazů, avšak na druhou stranu může jiné transakce udělat mnohem méně efektivními. Na často opakující se operace je denormalizace vhodná, protože se nemusí provádět operace spojení. Problém může vzniknout u agregačních funkcí, jako například výpočet průměru. V denormalizovaném tvaru bude potřeba data nějakým způsobem filtrovat a průměr vypočítávat až poté.
- Denormalizace téměř vždy vede k vyšší režii při vkládání a aktualizaci dat.
- Protože při denormalizaci vznikají redundantní data, může dojít k nekonzistenci informací. Problémy s nekonzistentními daty mohou být obrovské, je totiž velice obtížné tyto nesrovnalosti odladit a opravit.

Denormalizace by se neměla brát na lehkou váhu. Nejprve je dobré uvážit jaká rizika a jaké přínosy každá navrhovaná denormalizace přinese. Popřípadě je dobré vyzkoušet výkon denormalizovaného modelu před konečnou realizací.

Pro údržbu denormalizovaného modelu existují nástroje, kterými je možné rizika snížit. Mezi tyto nástroje patří databázové triggery a materializované pohledy (viz kapitola 4.5), díky kterým je možné řešit ukládání a aktualizaci redundantních dat automatizovaně.

Vertikální dělení a spojování

Vertikální dělení je metoda, kdy se rozdělí atributy jedné entity do dvou či více relací. Používá se především u tabulek, které obsahují velký počet atributů nebo pokud mohou sloupce obsahovat dlouhé textové řetězce. Nevýhodou této metody je následné spojování tabulek v SQL dotazech nebo vytváření pohledů, které by jinak nebyly potřeba.

Vertikální spojování je přesným opakem vertikálního dělení. Vertikálním spojením je zvýšen počet atributů v tabulce. Používá se v případě, kdy se předpokládá časté použití dvou entit najednou a bylo by potřeba vždy provádět operaci spojení. Nevýhodou vertikálního spojování je replikace sloupců.

Souhrnné tabulky

Dotazy, které generují součty nebo různé agregace jsou často velmi náročné na využití výpočetních zdrojů. Jedním z řešení je udržovat tabulku s agregačními daty, aby byl k těmto informacím snadný přístup. Harrison Guy [8, s. 91] popsal dva způsoby, jakými lze souhrnné tabulky udržovat:

- Pokud je požadováno udržovat data v reálném čase, je potřeba aktualizovat data ihned při změně zdrojových dat. Aktualizace lze provádět pomocí databázových triggerů nebo materializovaných pohledů. I když tento přístup umožňuje udržovat aktuální data s agregacemi bez režie, které vznikají při používání agregačních funkcí, může mít negativní vliv na zpracování transakcí.
- Pokud není požadováno udržovat data v reálném čase, souhrnná tabulka může být pravidelně aktualizována pomocí naplánovaných databázových jobů[1]. Taková naplánovaná úloha by se měla pouštět v době, kdy k databázi přistupuje nejméně uživatelů. Tento přístup má tu výhodu, že eliminuje režii při aktualizaci zdrojových dat. Nevýhodou však je fakt, že data v souhrnné tabulce nemusí být aktuální.

[1] Úloha, která může být pomocí plánovače spouštěna v určený čas.

4.2 Tabulky

Tabulka je v databázi základní úložnou jednotkou. Bez ohledu na typ tabulky jsou data vždy uložena v řádcích a sloupcích.

V této kapitole jsou popsány různé typy tabulek, které lze v databázi Oracle použít. Výběrem vhodného typu je možné následně docílit efektivnějšího SQL dotazu.

4.2.1 Tabulky organizované jako halda

Tento druh tabulky je nejpoužívanějším typem v databázi Oracle. Tabulka je organizovaná jako halda (heap), to znamená, že řádky tabulky nejsou uloženy v žádném pořadí. Při vkládání nových záznamů je použito první volné místo v segmentu, do kterého se data vejdou [5, s. 413].

Pro vytvoření tohoto typu tabulky je sice možné zadat klauzuli **organization heap**, ale není to nutné, protože tento typ je v databázi Oracle definován jako výchozí.

```
CREATE TABLE zamestnanec (
      jmeno      VARCHAR2(100 CHAR) NOT NULL,
      prijmeni   VARCHAR2(100 CHAR) NOT NULL
) ORGANIZATION HEAP;
```

Kód 1: Vytvoření tabulky organizované jako halda

4.2.2 Dočasné tabulky

Dočasná tabulka (**temporary table**), má stejné vlastnosti jako relační tabulka až na to, že data vložená do dočasné tabulky jsou k dispozici pouze po dobu trvání relace nebo transakce. Dočasné jsou tedy ve smyslu dočasnosti dat, nikoli dočasnosti definice tabulky jako takové. Pokud je potřeba, aby s tabulkou mohli pracovat všichni uživatelé připojující se k databázi, je potřeba vytvořit globální dočasnou tabulku. Rozdíl proti relační tabulce je ten, že každý uživatel v tabulce vidí pouze svá data. V případě provedení operace smazání všech dat z tabulky se také smažou pouze data daného uživatele. O dočasných tabulkách více v knize autora Davida Procházky [9].

Existují dva typy dočasných tabulek, u obou je perzistence řízena klauzulí **on commit**:

1. Dočasnost z hlediska trvání relace – Při vytváření dočasné tabulky je potřeba definovat tento typ použitím klauzule **on commit preserve rows**.

2. Dočasnost z hlediska trvání transakce – Při vytváření dočasné tabulky je potřeba použít klauzuli **on commit delete rows**. Tím se zajistí, že se po vykonání příkazu **commit** data z tabulky automaticky odstraní.

Pokud se při vytváření dočasné tabulky typ nedefinuje, Oracle implicitně volí typ dočasnosti z hlediska trvání transakce. Příklad vytvoření globální dočasné tabulky z hlediska trvání relace:

```
CREATE GLOBAL TEMPORARY TABLE zamestnanec (
      jmeno      VARCHAR2(100 CHAR) NOT NULL,
      prijmeni   VARCHAR2(100 CHAR) NOT NULL
) ON COMMIT PRESERVE ROWS;
```

Kód 2: Vytvoření dočasné tabulky

4.2.3 Indexově orientované tabulky (IOT)

Vytvoření indexu usnadňuje a zefektivňuje databázi Oracle vyhledávání konkrétního řádku v tabulce. Samozřejmě výhoda indexů s sebou nese i určitou režii. Oracle musí udržovat data jak v tabulce, tak i v indexu. Indexově orientované tabulky jsou extrémním případem využití indexů a měly by se používat jen v případech, kdy tabulka obsahuje malé množství sloupců a kde se k hodnotám přistupuje přes primární klíč. Využívají se například u vazebních tabulek. Definice primárního klíče je jedním z hlavních rozdílů oproti relační tabulce, indexově orientovaná tabulka ho vždy musí obsahovat. Při použití tohoto typu tabulky jsou jednotlivé řádky ukládány do indexu typu B-tree, kde jsou data setříděna podle primárního klíče. Indexům a jejich typům je věnována kapitola 4.3, kde je detailně popsán i zmíněný B-tree index.

Výhodou indexově orientované tabulky je, že databáze místo udržování dvou struktur, tedy tabulky a indexu, udržuje pouze jednu. To znamená, že i operace pro vkládání, aktualizaci a mazání dat jsou prováděny pouze na úrovni indexu. Tento typ tabulky má ale i svoje nevýhody. Jak už bylo řečeno, tabulka vždy musí obsahovat primární klíč a nemůže být součástí clusteru (viz kapitola 4.2.4). Problematiku indexově orientovaných tabulek podrobně popsali autoři Bob Bryla a Kevin Loney [1, s. 33-34].

Pro vytvoření indexově orientované tabulky je nutné definovat primární klíč a uvést klauzuli **organization index**.

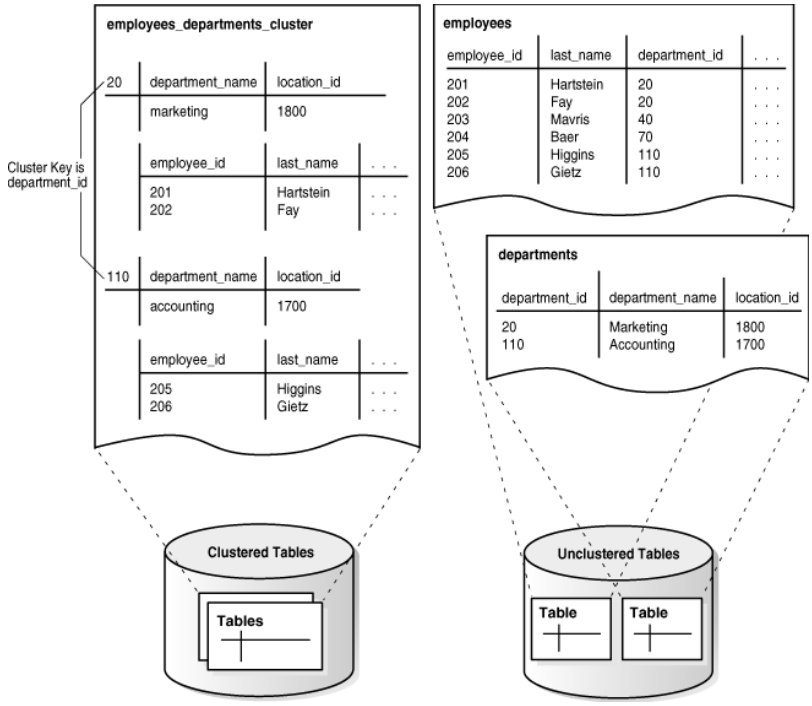
```
CREATE TABLE zamestnanec (
      id          NUMBER(18) PRIMARY KEY NOT NULL
      jmeno       VARCHAR2(100 CHAR) NOT NULL,
      prijmeni    VARCHAR2(100 CHAR) NOT NULL
) ORGANIZATION INDEX;
```

Kód 3: Vytvoření indexově orientované tabulky

4.2.4 Seskupené tabulky (Clustery)

Seskupená tabulka neboli **cluster** se používá, pokud je potřeba přistupovat ke dvěma či více tabulkám současně. Při použití seskupené tabulky je možné uložit data ze všech tabulek, které ji tvoří, ve stejném bloku (lze vidět na obrázku 2). Tím se sníží počet V/V operací, které je potřeba provést pro získání požadovaných dat. U seskupených tabulek je nutné definovat hodnotu klíče tzv. **cluster key value**. Klíčem je sloupec, který mají tabulky společný (může jich být i více). Obecně by se seskupené tabulky neměly využívat v případě, kdy je počet příkazů pro zápis dat vyšší než počet příkazů pro získání dat.

Oracle ve své dokumentaci [11] uvádí dva typy seskupených tabulek. Prvním typem je Indexovaná seskupená tabulka, která pro přístup k datům využívá index. Druhým typem je hashovaná seskupená tabulka, která přistupuje k datům pomocí hashovací funkce.



Obrázek 2: Data v seskupené tabulce [11]

Indexovaná seskupená tabulka (Indexed cluster)

Indexovaná seskupená tabulka využívá pro přístup k datům index. Index je typu B-tree a je umístěn na hodnotu klíče. Při ukládání společného sloupce se snižují nároky na potřebné místo, protože klíč je uložen v indexu seskupení (**cluster index**). Příklad vytvoření indexované seskupené tabulky:

```
-- Vytvoření clusteru s hodnotou klíče (cluster key value)
CREATE CLUSTER zamestnanec_oddeleni_cluster (
  oddeleni_id NUMBER(18)
) SIZE 512;

-- Vytvoření indexu seskupeni (cluster index)
CREATE INDEX idx_zam_odd_cluster
  ON CLUSTER zamestnanec_oddeleni_cluster;

-- Vytvoření 1. tabulky seskupené do clusteru
CREATE TABLE zamestnanec(
  zamestnanec_id  NUMBER(18) NOT NULL,
  jmeno           VARCHAR2(100 CHAR),
  prijmeni        VARCHAR2(100 CHAR) NOT NULL,
  oddeleni_id     NUMBER(18) NOT NULL
) CLUSTER zamestnanec_oddeleni_cluster (oddeleni_id);

-- Vytvoření 2. tabulky seskupené do clusteru
CREATE TABLE oddeleni(
  oddeleni_id     NUMBER(18) NOT NULL,
  nazev           VARCHAR2(20 CHAR) NOT NULL
) CLUSTER zamestnanec_oddeleni_cluster (id);
```

Kód 4: Vytvoření indexované seskupené tabulky

Hashovací seskupená tabulka (Hash cluster)

Hashovací seskupená tabulka je podobná indexované seskupené tabulce. Liší se v přístupu k datům, kde místo indexu klíče používá hashovací funkci. Neexistuje žádný index nad seskupenou tabulkou, ale data jsou sama indexem [10].

Při použití indexované seskupené tabulky nebo Indexově orientované tabulky musí Oracle při vyhledávání nebo ukládání provést minimálně dvě V/V operace:

- Jedna nebo více V/V operací k získání nebo uložení hodnoty klíče v indexu
- Další V/V operace ke čtení nebo zápisu řádku v tabulce nebo clusteru

Oracle ve své dokumentaci [11] uvádí, že pro ukládání a vyhledávání databáze provádí pouze jedinou operací, protože data jsou sama indexem. Při těchto úkonech totiž databáze v hashovací seskupené tabulce aplikuje hashovací funkci na hodnotu klíče v daném řádku, jejímž výsledkem je hodnota odpovídající konkrétnímu bloku v clusteru.

Z hlediska výkonu je hashovací seskupená tabulka vhodnější ve chvíli, kdy je při získávání dat použita podmínka na hodnotu klíče s operátorem rovnosti [11].

Při vytváření hashovací seskupené tabulky je potřeba uvést klauzuli **hashkeys** s hodnotou specifikující maximální počet unikátních hodnot generovaných hashovací funkcí. Následuje příklad vytvoření hashovací seskupené tabulky:

```
-- Vytvoření clusteru s hodnotou klíče a s alokovanim 100 bloku
CREATE CLUSTER zamestnanec_oddeleni_cluster (
  oddeleni_id NUMBER(18)
) SIZE 8192 HASHKEYS 100;

-- Vytvoření 1. tabulky seskupené do clusteru
CREATE TABLE zamestnanec(
  zamestnanec_id  NUMBER(18) NOT NULL,
  jmeno           VARCHAR2(100 CHAR),
  prijmeni        VARCHAR2(100 CHAR) NOT NULL,
  oddeleni_id     NUMBER(18) NOT NULL
) CLUSTER zamestnanec_oddeleni_cluster (oddeleni_id);

-- Vytvoření 2. tabulky seskupené do clusteru
CREATE TABLE oddeleni(
  oddeleni_id     NUMBER(18) NOT NULL,
  nazev           VARCHAR2(20 CHAR) NOT NULL
) CLUSTER zamestnanec_oddeleni_cluster (oddeleni_id);
```

Kód 5: Vytvoření hashovací seskupené tabulky

4.2.5 Rozdělené tabulky

Účelem této metody je rozdělení rozsáhlých tabulek na menší oddíly. Metoda rozdělení se mimo jiné používá i u indexů. Při správném použití rozdělení lze docílit mnohem efektivnějších SQL dotazů a mimo jiné velmi často usnadňuje práci databázovému správci. Při poruše jednoho diskového oddílu je totiž znemožněn přístup pouze do tohoto oddílu, zatímco na ostatní oddíly tabulky se lze i nadále dotazovat [12]. Metoda rozdělení do oddílů se používá především u aplikací s požadavkem na extrémně vysokou dostupnost.

Oracle ve své dokumentaci [13] uvádí čtyři možné typy rozdělení: oddíly definované rozsahem, oddíly definované hashovací funkcí, oddíly definované seznamem, a složené (kombinované) oddíly.

Bez ohledu na typ oddílu může každý řádek v rozdělené tabulce existovat pouze v jediném oddílu a uložení řádku do oddílu řídí tzv. **klíč rozdělení**, který se může skládat až z 16 sloupců tabulky [13].

Rozdělení definované rozsahem

Data jsou rozdělena podle rozsahu hodnot. Rozdělení definované rozsahem se používá především u časových dat. Například rozdělení dat podle měsíců, kde data v každém měsíci patří do samostatného oddílu. Při vytváření je nutné uvést klauzuli **partition by range**, dále specifikovat klíč rozdělení, podmínku, podle které budou data rozdělena, a název oddílu [13].

```
CREATE TABLE objednavka (
  id    NUMBER(18) NOT NULL,
  cena  NUMBER(8) NOT NULL,
  datum_objednani DATE  NOT NULL
)PARTITION BY RANGE(datum_objednani)
(
 PARTITION objednavky_leden2016 VALUES LESS THAN(TO_DATE('01/02/2016')),
 PARTITION objednavky_unor2016 VALUES LESS THAN(TO_DATE('01/03/2016')),
 PARTITION objednavky_brezen2016 VALUES LESS THAN(TO_DATE('01/04/2016'))
);
```

Kód 6: Vytvoření rozdělené tabulky definované rozsahem

Rozdělení definované hashovací funkcí

Data jsou rozdělena pomocí hashovací funkce. Při vytváření je potřeba specifikovat sloupce, které se v hashovací funkci použijí a dále počet dostupných oddílů. Samotné oddíly se na rozdíl od oddílů definovaných rozsahem nespecifikují. Přidělení řádku do příslušného oddílu zajistí databáze Oracle automaticky a zajistí i to, aby byla data rozdělena do oddílů rovnovážně. Tento typ rozdělení je vhodnější než rozdělení definované rozsahem v situacích, kdy nelze jednoduše určit rozsahy oddílů nebo v případech, kdy by se mohla velikost dat v oddílech výrazně lišit [13].

```
CREATE TABLE objednavka(
   id           NUMBER(18) NOT NULL,
   cena         NUMBER(10),
   zakaznik_id  NUMBER(18) NOT NULL
)
PARTITION BY HASH(zakaznik_id)
PARTITIONS 4;
```

Kód 7: Vytvoření rozdělené tabulky definované hashovací funkcí

Rozdělení definované seznamem

Data jsou rozdělena pomocí konkrétních hodnot. Při vytváření je nutné specifikovat seznam hodnot, kterými databázi předáme informace o tom, do kterého oddílu data náleží. Oproti rozdělení definovaných rozsahem a hashovací funkcí, nemůže tento typ obsahovat klíč definovaný více sloupci, ale je definován vždy právě jedním. Ve většině případů se při vytváření definuje oddíl s hodnotou DEFAULT pro data, která nespadají ani

do jedné z hodnot určených seznamem [13]. Pokud v definici rozdělení takový oddíl není definován, pak databáze Oracle nepovolí vložit jiné hodnoty, než jsou definované v seznamu hodnot.

```
CREATE TABLE objednavka(
   id            NUMBER(18) NOT NULL,
   cena          NUMBER(10),
   misto_prodeje VARCHAR2(50 CHAR) NOT NULL
)
PARTITION BY LIST(misto_prodeje)
(
   PARTITION prodej_hkk VALUES('Hradec Králové','Jičín','Náchod'),
   PARTITION prodej_pha VALUES('Praha 1','Praha 2','Praha 3', 'Praha 4'),
   PARTITION prodej_pak VALUES('Pardubice','Chrudim'),
   PARTITION prodej_ostatni VALUES(DEFAULT)
);
```

Kód 8: Vytvoření rozdělené tabulky definované seznamem

Složené rozdělení

Oracle ve své dokumentaci [13] uvádí ještě složené (kombinované) rozdělení. Složené rozdělení je speciálním typem, které umožňuje rozdělit tabulku pomocí kombinace předchozích typů. Příkladem je rozdělení tabulky podle rozsahu hodnot, kde je dále každý jednotlivý oddíl rozdělen pomocí seznamu, hashovací funkce nebo rozsahu. Databáze Oracle podporuje kombinace všech typů rozdělení. Následující příklad demonstruje složené rozdělení typu rozsah-hashovací funkce.

```
CREATE TABLE objednavka(
   id           NUMBER(18) NOT NULL,
   cena         NUMBER(10),
   datum_objednani DATE  NOT NULL,
   zakaznik_id  NUMBER(18) NOT NULL
)
PARTITION BY RANGE(datum_objednani)
SUBPARTITION BY HASH(zakaznik_id)
(
   PARTITION objednavky_leden2016 VALUES LESS THAN(TO_DATE('01/01/2016')),
   PARTITION objednavky_unor2016 VALUES LESS THAN(TO_DATE('01/02/2016')),
   PARTITION objednavky_brezen2016 VALUES LESS THAN(TO_DATE('01/03/2016'))
);
```

Kód 9: Vytvoření složené rozdělené tabulky typu rozsah-hashovací funkce

4.3 Indexy

Autoři Ashdown a Kyte [14] popisují index jako strukturu postavenou nad databázovou tabulkou nebo clusterem, jejímž cílem je poskytnout rychlý přístup k požadovaným řádkům tabulky. Index je možné vytvořit nad jedním nebo více sloupci tabulky. Používají z jednoho jediného důvodu a tím je zrychlení přístupu k datům. Když se

začne zpracovávat požadavek na data, databáze použije index k nalezení identifikátoru řádku. Díky tomu přesně ví, o jaké řádky se jedná a jejich nalezení je mnohem rychlejší. Vyhledávání v indexu je většinou mnohem rychlejší, protože neobsahuje všechny sloupce tabulky a data v něm jsou na rozdíl od tabulky setříděná. Důležité je také vědět, že všechny indexy kromě bitmapového neobsahují hodnoty NULL. To znamená, že při dotazování se na prázdné řádky se indexy nepoužijí.

Indexy jsou logicky a fyzicky nezávislé na datech tabulky, tzn., že jejich přítomnost nebo absence samotná data nijak neovlivní. Po přidání nebo odebrání indexu u daných sloupců tabulky není potřeba upravovat SQL dotaz, který do těchto sloupců přistupuje. Indexy jsou jednou z mnoha možností jak snížit V/V operace disku. Pokud tabulka uspořádaná do haldy neobsahuje index, pak při dotazu do tabulky musí databáze provádět metodu Full table scan. Při použití metody Full table scan musí databáze Oracle projít tabulku řádek po řádku. Takové chování u tabulek s velkým množstvím dat zabere databázi značné množství času. Metoda úplného procházení tabulky je detailněji popsána v kapitole 5.2.

Používání indexů sebou nese i několik nevýhod. Databáze Oracle musí udržovat kromě dat v tabulkách ještě data v další struktuře, tedy v indexech. Při jakékoliv změně dat v tabulce musí databáze indexy v oné tabulce přepočítat, aby byla data aktuální. Tím se zpomaluje provádění DML příkazů. Další nevýhodou může být kapacita na disku, kterou indexy zabírají. Při rozumném množství to problém není, proto by se měly indexy vytvářet s rozvahou a indexy, které už nejsou potřeba, by se měly smazat. Databázi Oracle tak odpadá i práce s jejich údržbou.

Obecně lze říci, že index je vhodné vytvořit ve chvíli, kdy je konkrétní sloupec nejčastějším predikátem v dotazu a dotaz vrací jen malou část řádků tabulky. Použít by se měl i v případě, kdy sloupec v tabulce obsahuje omezení referenční integrity. Databáze Oracle pak nemusí uzamykat celou tabulku v případě aktualizace nebo mazání dat rodičovské tabulky.

Databázový systém Oracle automaticky vytváří indexy pro primární a unikátní klíče. Jak už bylo řečeno, index se vytváří buď na jednom, nebo více sloupcích zároveň. Index, který se skládá z více sloupců tabulky, se nazývá složený index. U složených indexů záleží na pořadí, ve kterém jsou sloupce definovány při vytváření indexu. Složený index

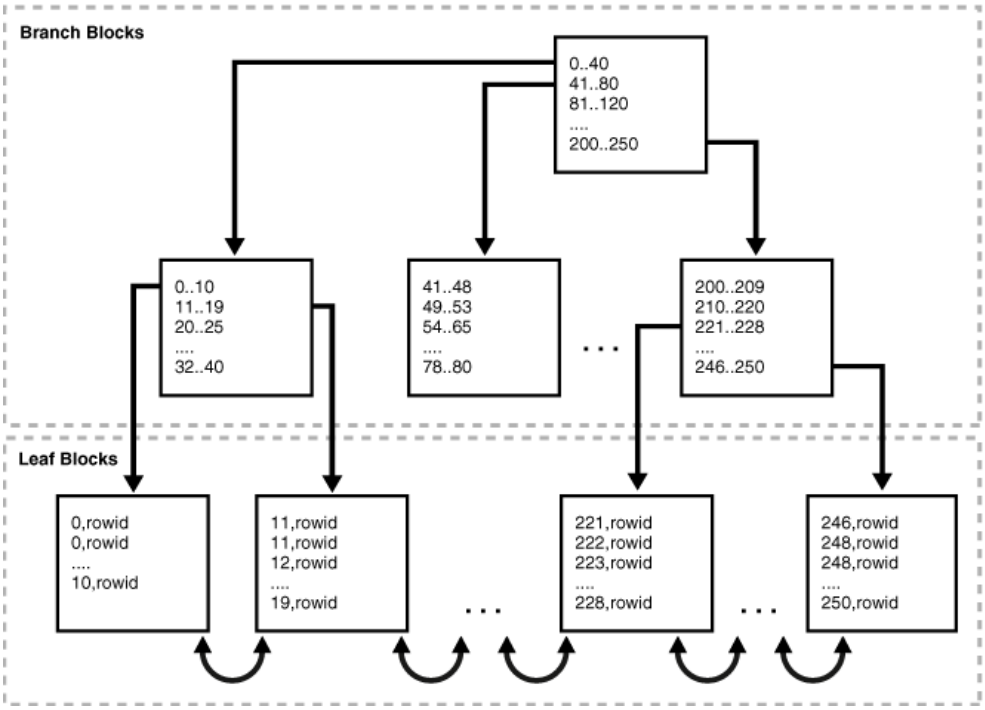
může velmi zrychlit SQL dotaz, který obsahuje podmínku na všechny nebo většinu sloupců definovaných v indexu. Proto je pořadí sloupců při vytváření indexu důležité. Obecně lze říci, že sloupce, které mají nejvyšší kardinalitu (mají největší počet různých hodnot), by se ve složeném indexu měly definovat jako první.

Databáze Oracle podporuje několik typů indexů. Každý se používá v jiné situaci, záleží například na typu tabulky, metodě přístupu i aplikačním prostředí. Jednotlivé typy indexů a jejich výhody jsou popsány v následujících podkapitolách.

4.3.1 B-tree index

B-tree index je standardním a nejpoužívanějším typem v databázi Oracle. Jak už název tohoto typu napovídá, je založen na stromové struktuře.

Ashdown a Kyte [14] popisují B-tree index jako seřazený seznam hodnot, ve kterém jsou data rozdělena do bloků. Bloky jsou dvou typů, tzv. Branch bloky a Leaf bloky (viz obrázek 3). Každý typ bloku má jiný účel, branch bloky jsou důležité z hlediska vyhledávání hodnot a v leaf blocích jsou uloženy samotné hodnoty spolu s adresou ROWID, který určuje aktuální řádek v datovém segmentu. Vyšší vrstvy branch bloků obsahují ukazatele do nižších vrstev a při procházení indexu se tak databáze dostane až k leaf blokům, kde se nacházejí samotné hodnoty.



Obrázek 3: Struktura B-tree indexu [14]

Jak je vidět na obrázku, každý uzel v leaf blocích je propojený se svými sousedícími bloky. Díky tomu jsou B-tree indexy výkonné jak při vyhledávání konkrétní hodnoty, tak i podle rozsahu. V jednotlivých blocích B-tree indexu je množství hodnot vyvážené, protože všechny leaf bloky mají stále stejnou hloubku [14]. To znamená, že každý blok obsahuje přibližně stejný počet záznamů. Tím pádem trvá databázi získání jakéhokoliv záznamu v jakémkoliv bloku zhruba stejný čas. S vyhledáváním v indexu souvisí pojem výška indexu. Výška indexu určuje počet bloků z kořenového bloku k leaf blokům [14].

Kromě běžného B-tree indexu existuje několik typů indexů, které jsou na B-tree indexu postavené. Mezi nejpoužívanější patří unikátní index, index s reverzním klíčem a sestupný index.

Unikátní index

Unikátní index se obvykle používá pro implementaci primárního klíče tabulky. Kromě samotného indexu zajišťuje kontrolu na unikátnost hodnot ve sloupci, to znamená, že v takto indexovaném sloupci se nebudou žádné hodnoty opakovat [1, s. 40]. Pro vytvoření tohoto typu indexu je důležité uvést klíčové slovo **unique**.

```
CREATE UNIQUE INDEX idx_objednavka_zak_id
   ON objednavka(zakaznik_id);
```

Kód 10: Vytvoření unikátního indexu

Index s reverzním klíčem

Index s reverzním klíčem ukládá bajty hodnoty klíče v opačném pořadí, než běžný B-tree index [1, s. 40]. Využívá se především v prostředí OLAP. Při vytváření se tento typ definuje klíčovým slovem **reverse**. Protože jsou data při jednotlivých vkládáních rozdělena podél uzlů v leaf blocích, snižuje se soupeření o prostředky v případě, kdy současně více uživatelů vkládá nová data [1, s. 40].

```
CREATE INDEX idx_objednavka_cislo_obj
   ON objednavka(CISLO_OBJEDNAVKY) REVERSE;
```

Kód 11: Vytvoření indexu s reverzním klíčem

Sestupný index

Základní B-tree index defaultně řadí data vzestupně, časová data tedy od nejstarších po nejnovější, čísla od nejmenšího po největší atd. Sestupný index, jak už název

napovídá, používá sestupné řazení. Využívá se v případech, kdy je požadavkem řadit data sestupně. Při sestavování dotazu pak není nutné používat klauzuli **order by** pro seřazení dat. Oracle využije řazení v indexu místo toho, aby bylo řazení provedeno v některé z dalších operací [1, s. 41]. Typ řazení lze při vytváření indexu definovat pro jednotlivé sloupce.

```
CREATE INDEX idx_objednavka_cislo_obj_desc
   ON objednavka(CISLO_OBJEDNAVKY DESC);
```

Kód 12: Vytvoření sestupného indexu

4.3.2 Bitmapový index

Bitmapový index má oproti B-tree indexu naprosto odlišnou strukturu. V bitmapovém indexu je pro každou možnou hodnotu přiřazen řetěz bitů, jehož délka je stejná jako počet řádků v tabulce. Mapovací funkce, kterou bitmapový index využívá, mapuje pozice bitů na aktuální ROWID [1, s. 42]. Hlavním rozdílem proti B-tree indexu je využívání jednoho klíče pro libovolné množství řádku. Díky tomu bitmapový index výrazně šetří místo. V B-tree indexu vždy jeden klíč odkazuje právě na jeden konkrétní řádek. Dalším velkým rozdílem proti všem ostatním typům indexů je vytváření klíčů i pro NULL hodnoty.

Bitmapový index se využívá především pro sloupce s velmi nízkou kardinalitou, tedy s malým počtem unikátních hodnot. Pokud sloupec obsahuje například tři různé hodnoty, tak budou v indexu uloženy pouze tři bitové hodnoty. Avšak v případě, že bude sloupec obsahovat v řádcích pouze unikátní hodnoty, tak bude indexu obsahovat identický počet bitových řetězců, jako je řádků v tabulce [14]. V takovém případě je mnohem efektivnější použít tradiční B-tree index.

Jak již bylo řečeno, bitmapový index není vhodné používat u sloupců s vysokou kardinalitou a dále v případě, kdy se data v tabulce často modifikují. Při aktualizaci řádku s bitmapově indexovaným sloupcem totiž dojde k uzamčení všech řádků tabulky, které mají stejnou hodnotu bitu jako právě modifikovaný řádek [14].

Vytvoření bitmapového indexu lze vidět v následujícím příkladu, stačí uvést klíčové slovo **bitmap**.

```
CREATE BITMAP INDEX idx_objednavka_zpusob_platby
   ON objednavka(zpusob_platby);
```

Kód 13: Vytvoření bitmapového indexu

Na bitmapovém indexu jsou postaveny bitmapové spojované indexy. Tento index se vytváří především nad sloupcem tabulky, který je často využíván ke spojení s jinými tabulkami. Své využití nalezl především v prostředí datových skladů, kde jsou základní tabulky spojovány s rozměrovými. Při vytváření bitmapového spojovaného indexu se zároveň vytvoří struktura spojení tabulek, tím šetří prostředky procesoru a V/V operace při provádění operace spojení [1, s. 42]. Vytvoření bitmapového spojovaného indexu vypadá následovně:

```
CREATE BITMAP INDEX idx_objednavka_zak_id
   ON objednavka (zakaznik.prijmeni)
   FROM objednavka, zakaznik
   WHERE objednavka.zakaznik_id = zakaznik.id;
```

Kód 14: Vytvoření bitmapového spojovaného indexu

4.3.3 Funkční index

Bob Bryla a Kevin Loney [1, s. 42] popisují funkční index jako index, který obsahuje transformované hodnoty na základě použité funkce při vytváření indexu. Index může být definován buď jako B-tree index nebo jako bitmapový index, přičemž se jeho struktura odvíjí od použitého typu. Funkce, která je pro index použita, může být založena na aritmetickém výrazu, na výrazu obsahujícím SQL funkci, dokonce může být použita i vlastní PL/SQL funkce a package funkce. Funkční indexy jsou velmi efektivní a používají se jak pro jednoduché funkce jako je například UPPER a LOWER pro vyhledávání v řetězcích bez ohledu na velikost písmen, tak pro složité vlastní funkce obsahujících desítky operací. V SQL dotazu se již jednoduše použije hodnota z indexu místo toho, aby databáze hodnoty počítala ve chvíli vykonávání dotazu. Funkční index vlastně slouží pro předpočítávání složitě počítaných hodnot. Následující příklad demonstruje vytvoření indexu s využitím funkce:

```
CREATE INDEX idx_osoba_prijmeni
   ON osoba(upper(prijmeni));
```

Kód 15: Vytvoření funkčního indexu

4.4 Pohledy

Pohledy jsou logickou reprezentací jedné nebo více tabulek. Ze své podstaty jsou také nazývané jako uložené dotazy, nic jiného klasické pohledy totiž nejsou. Data vrácená z pohledu jsou pouze data z tzv. základních tabulek, které vrací dotaz, jímž je pohled definován. Základní tabulky v definici pohledu mohou být obyčejné databázové tabulky nebo i jiné pohledy. Samotný pohled neukládá žádná data, to znamená, že databáze Oracle nemusí pro takový pohled alokovat žádné místo v databázovém segmentu [1, s. 42-43]. V datovém slovníku je uložena pouze definice pohledu, kterou je SQL dotaz. V případě, kdy se začne zpracovávat dotaz obsahující pohled, databázový systém Oracle nahradí pohled vlastním SQL dotazem z definice pohledu a daný dotaz zpracuje stejně, jako by pohled neexistoval [1, s. 42-43]. Tím databáze zajistí, že se při zpracování takového dotazu využijí všechny indexy, které jsou nad tabulkou vytvořené.

Pohledy se používají i z důvodu bezpečnosti. Například tím, že pohled neobsahuje určité sloupce s citlivými daty, jako jsou například platy zaměstnanců. V definici pohledu lze také pomocí klauzule **read only** databázi říct, že pohled má sloužit pouze pro čtení [14]. Bez klauzule **read only** je totiž v pohledu možné měnit data. Aktualizaci dat zabrání kromě klauzule **order by** ještě další konstrukce, jedná se o operátor **distinct**, agregační funkce nebo klauzuli **group by** [14]. Následující příklad demonstruje vytvoření pohledu pro čtení s informacemi o zákaznících a jejich objednávkách:

```
CREATE VIEW objednavky_info AS
   SELECT
       z.jmeno,
       z.prijmeni,
       o.cislo_objednavky,
       o.castka,
       o.datum_objednani
   FROM objednavka o
   INNER JOIN zakaznik z ON z.id = o.zakaznik_id
WITH READ ONLY;
```

Kód 16: Vytvoření pohledu pro čtení

4.5 Materializované pohledy

Kromě základních pohledů popsaných v kapitole 4.4, podporuje databáze Oracle ještě tzv. materializované pohledy, které umožňují kromě definice dotazu ukládat i výsledky dotazu [1, s. 43-44]. S obyčejnými pohledy si jsou podobné v jedné věci, jejich definice je uložena v datovém slovníku, tím podobnost obyčejných a materializovaných pohledů končí.

Materializovaný pohled si alokuje místo v databázovém segmentu, do něhož ukládá výsledky provedení SQL dotazu z definice [14]. Definice SQL dotazu se může skládat z tabulek, pohledů nebo jiných materializovaných pohledů. Velmi často se používají pro replikaci kopie tabulky nebo jako tabulka faktů v datových skladech.

Dokumentace Oracle [14] uvádí několik vlastností, které mají materializované pohledy společné s databázovými indexy:

- Obsahují aktuální data s tím, že potřebují mít alokovaný prostor v databázovém segmentu.
- Data jsou aktualizována v případě, že se změní data ve zdrojových tabulkách.
- Používají se z důvodu zvýšení výkonu provádění SQL dotazů

4.5.1 Aktualizace pohledů

Pro aktualizaci dat materializovaného pohledu je možné využívat buď kompletní aktualizaci, nebo způsob pro aktualizaci pouze přírůstkové části.

Kompletní aktualizace

Při provádění kompletní aktualizace musí databáze Oracle provést kompletní SQL dotaz, kterým je pohled definován [14]. Takové chování může být velmi pomalé, především v případě, kdy databáze musí číst a zpracovávat obrovské množství dat. Z tohoto důvodu se při práci s velkým množstvím dat používá spíše způsob přírůstkové aktualizace.

Přírůstková aktualizace

Tento způsob aktualizace se používá z důvodu snížení doby odezvy při obnově pohledu. Databáze zpracuje pouze data, u kterých došlo ke změně [14]. Výsledkem toho je v převážné většině případů velmi rychlé provedení aktualizace.

Pro možnost provádění přírůstkové aktualizace je potřeba určit způsob, jakým databáze Oracle pozná, která data se v materializovaném pohledu změnila. Oracle ve své dokumentací [14] uvádí dvě možnosti, díky kterým dokáže určit změněná data:

- **Aktualizace založená na logování**

  Při tomto typu aktualizace je udržován log materializovaného pohledu, který zaznamenává všechny změny v hlavní tabulce, tzv. **master table**. Při aktualizaci pohledu se data porovnají s logem a databáze díky tomu ví, u kterých dat došlo ke

změně. Log materializovaného pohledu se vytvoří příkazem **create materialized view log**.

- **Aktualizace založená na sledování změn v oddílech**
  Použít tuto metodu sledování změn je možné jen v případě, kdy jsou tabulky rozděleny na oddíly. Při aktualizaci jsou odstraněna všechna ovlivněná data v daném oddílu a následně jsou přepočítána.

4.5.2 Technika přepisování dotazů

Ashdown a Kyte [14] definují metodu přepisování dotazů jako optimalizační techniku, při které dochází k transformaci uživatelského dotazu, který se dívá do hlavních tabulek, na sémanticky stejný dotaz zahrnující materializované pohledy. Tato technika se u materializovaných pohledů používá v případě, kdy základní tabulky obsahují velké množství dat, používají se agregační funkce, nebo pokud je operace spojení časově náročnou operací. Přepsání dotazu může velmi výrazně snížit dobu odezvy dotazu. Materializované pohledy již mohou spočítaná data a spojení obsahovat a není nutné takové výpočty provádět až ve fázi dotazování. Databáze je pak takových operací v danou chvíli ušetřena a dotaz díky tomu vrací ekvivalentní data v kratším čase.

K přepisu dotazů se používá databázová komponenta, tzv. **Query Transformer**. Tato komponenta je detailněji popsána v kapitole 5.5.1.

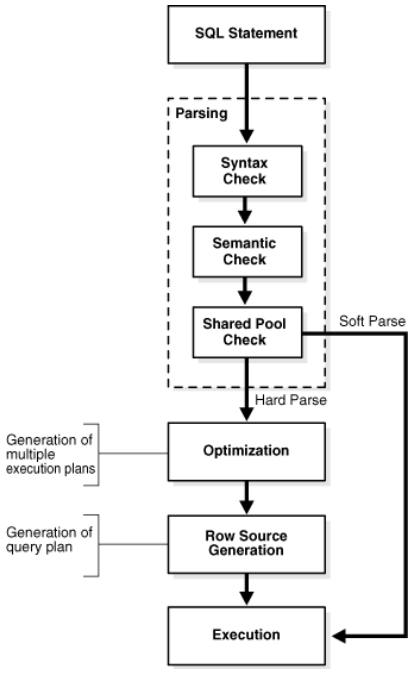
5 SQL dotazy

Tato kapitola se zabývá SQL dotazy. Rozhodně zde nejsou popsány konkrétní kroky, kterými lze obecně odladit jakýkoliv dotaz, protože taková sada kroků neexistuje. Kdyby takový algoritmus existoval, bylo by možné napsat program, který by optimalizace prováděl univerzálně. Ve skutečnosti se o zmíněnou univerzálnost ladění mnoho programů pokouší. Tyto programy většinou navrhují místa, kam přidat index, kde použít materializovaný pohled, nebo do jakého dotazu přidat jaký hint. Tyto programy však používají velmi omezenou sadu pravidel a proto by se mělo jejich doporučení vždy pečlivě uvážit. Kdyby byla jejich optimalizace použitelná univerzálně, prováděl by je optimalizátor standardně.

Kapitola se věnuje především postupu zpracování dotazů, přístupovým cestám, metodám spojení a v neposlední řadě samotnému optimalizátoru, který provádí různé operace automatizovaně.

5.1 Zpracování SQL příkazů

Kapitola popisuje postup databáze Oracle při zpracování SQL příkazů a to jak DDL k vytváření objektů, DML k úpravě dat, tak i dotazů k načtení dat. Obecný postup zpracování SQL lze vidět na obrázku 4. V závislosti na druhu příkazu mohou být některé fáze vynechány. Jednotlivé kroky při zpracování jsou popsány v následujících podkapitolách.



Obrázek 4: Postup zpracování SQL příkazů [7]

5.1.1 Analýza

Ve chvíli spuštění SQL příkazu, je potřeba provést některé operace před jeho skutečným provedením. Souhrn těchto operací nazýváme parsování (analyzování) a je to první krok při zpracování dotazu [7]. Při volání parseru se provádí kontrola syntaxe, sémantiky a kontrola sdílené oblasti.

Syntaktická analýza

Při syntaktické analýze dochází ke kontrole správnosti syntaxe jazyka SQL [7]. Následující obrázek demonstruje chybu v syntaxi.

```
SELECT * FORM employees;
ORA-00923: FROM keyword not found where expected
00923. 00000 -  "FROM keyword not found where expected"
*Cause:
*Action:
Error at Line: 1 Column: 10
```

Obrázek 5: Chybná SQL syntaxe (zdroj: autor)

Sémantická analýza

Sémantická analýza kontroluje, jestli je SQL příkaz smysluplný a zda existují objekty a sloupce, ke kterým má příkaz přistupovat [7]. Syntakticky správný dotaz se sémantickou chybou je vidět na obrázku 6.

```
SELECT * FROM nonexistent_table;
ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
Error at Line: 1 Column: 15
```

Obrázek 6: Sémantická chyba (zdroj: autor)

Kontrola sdíleného fondu

Samotný koncept sdíleného fondu je jedním z klíčových prvků architektury databáze Oracle. Thomas Kyte [5, s. 272] definoval sdílený fond takto: *„Sdílený fond je blok paměti ve sdílené globální oblasti databáze (SGA), v níž jsou uloženy dříve provedené příkazy SQL, kód PL/SQL, informace z mezipaměti slovníku a mnoho dalších položek za účelem jejich opětovného použití libovolnou relací v databázi“*.

Cílem kontroly sdíleného fondu je určit, zda je dotaz spouštěn poprvé. Oracle z každého SQL dotazu vytvoří pomocí hashovací funkce hash. Hodnota hashe je přístupná jako atribut SQL_ID v pohledu V$SQL. Při spuštění SQL dotazu se databáze vždy zeptá,

zda existuje záznam se stejnou hodnotou hashe. Pokud takový záznam neexistuje, znamená to, že je dotaz spuštěn poprvé. V takovém případě musí databáze projít všechny kroky zpracování. V průběhu toho musí alokovat paměť pro nový sdílený dotaz a uložit zparsovanou reprezentaci SQL příkazu. Takovéto zpracování se nazývá **úplná analýza** (**Hard parsing**). Mnohem jednodušší zpracování, tzv. **jemná analýza** (**Soft parsing**) se používá v případě opakovaného spuštění. Oracle si vyhledá záznam se stejnou hodnotou hashe a zjistí si naposledy použitý exekuční plán. Proto již není potřeba provádět kroky pro optimalizaci dotazu a generování řádkových zdrojů a lze přejít přímo k vykonání dotazu. Tím se ušetří strojový čas a vrácení výsledku dotazu je mnohem rychlejší. Problematika je detailněji popsána v dokumentaci Oracle [7].

5.1.2 Optimalizace

Cílem tohoto kroku je modifikovat dotaz a připravit ho tak, aby vlastní provedení bylo pro databázový systém co nejjednodušší [7]. Proces optimalizace probíhá během úplné analýzy, proto tímto procesem projdou veškeré DML příkazy vždy alespoň jednou. Samotný proces optimalizace je z hlediska procesoru velice nákladnou operací, která může trvat déle než vlastní provedení dotazu.

Proces optimalizace řídí v databázovém systému Oracle nástroj zvaný **Optimalizátor** (viz kapitola 5.5).

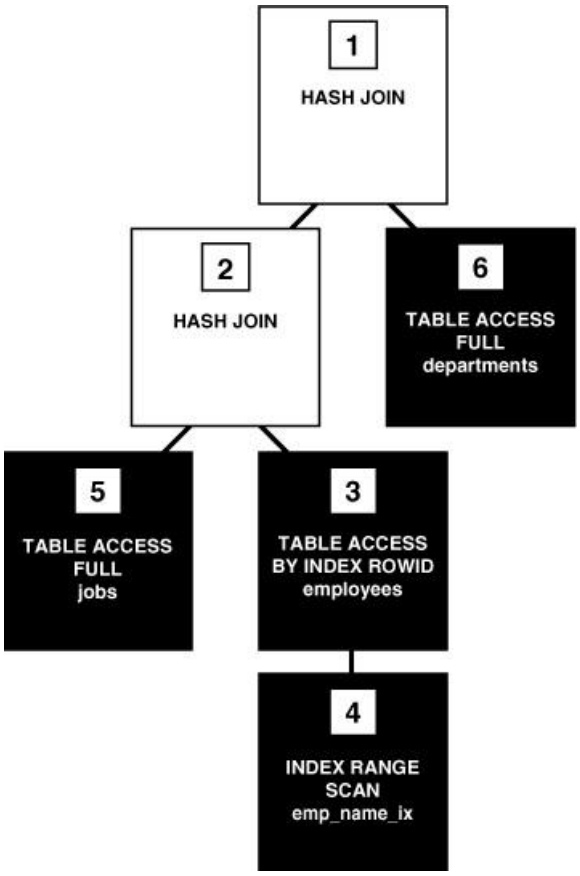
5.1.3 Generování řádkových zdrojů

Generátor řádkového zdroje je část softwaru databáze Oracle, která přijímá výstup z optimalizátoru a přemění ho na datovou strukturu, kterou dále používá zbývající část databázového systému, zejména iterační plán [7]. Iterační plán je poté modulem používán pro zpracování dotazu a generování výstupu.

Řádkový zdroj může být tabulka, pohled, nebo výsledek operace spojení. Datová struktura, jež je výstupem generátoru řádkových zdrojů, je označována jako **strom řádkových zdrojů** (viz obrázek 7) [7].

Jak už název napovídá, jedná se o stromovou strukturu, která s sebou nese následující informace:

- Pořadí tabulek, ve kterém budou zpracovány
- Přístupové metody pro každou tabulku v příkazu
- Metodu spojení pro tabulky, u kterých byla v příkazu použita operace spojení
- Operace s daty jako je filtrování, řazení, agregace atp.



Obrázek 7: Stromová struktura řádkových zdrojů [7]

5.1.4 Provedení

V tomto procesu je použit výstup z generátoru řádkového zdroje a pomocí něho je SQL příkaz proveden [7]. Provedení je jediným povinným krokem zpracování všech DML příkazů. Analýza, optimalizace i generování řádkového zdroje může databázový systém přeskočit a příkaz opakovaně provádět. To je z hlediska výkonu nejefektivnější metoda a měla by být záměrem při psaní SQL dotazů.

5.2 Přístupové cesty

Přístupové cesty jsou metody, prostřednictvím kterých databáze získává data [5, s. 479]. Neexistuje jedna jediná nejlepší cesta, databáze Oracle používá různé přístupové cesty pro různé datové struktury. Například pro indexy databáze používá jiné přístupové cesty než pro tabulky. Přehled přístupových cest pro vybrané datové struktury lze vidět v následující tabulce. Z hlediska optimalizace dotazů je velice důležité vědět, v jakých situacích se pro jakou přístupovou cestu optimalizátor databáze Oracle rozhodne.

Přístupová cesta	Tabulky organizované jako halda	B-tree indexy a IOT	Bitmapové indexy	Seskupené tabulky (clustery)
Full Table Scans	X			
Table Access by Rowid	X			
Sample Table Scans	X			
Index Unique Scans		X		
Index Range Scans		X		
Index Full Scans		X		
Index Fast Full Scans		X		
Index Skip Scans		X		
Index Join Scans		X		
Bitmap Index Single Value			X	
Bitmap Index Range Scans			X	
Bitmap Merge			X	
Cluster Scans				X
Hash Scans				X

Tabulka 1: Přístupové cesty a datové struktury [15]

Full Table Scans

Full Table Scan neboli úplné prohledávání tabulky je metoda, při níž databáze Oracle přečte všechny bloky v segmentu, který byl v nějakém místě použit. Databáze v tomto případě nejprve načte všechny řádky z tabulky a teprve poté odfiltruje řádky, které nesplňují filtrovací kritéria [5, s. 479-482]. Metoda úplného prohledávání je často odsuzována právě kvůli tomu, že je databáze nucena číst všechny řádky. Mnohokrát je ale nejrychlejší metodou pro získání řádků. Databáze může úplné prohledávání provádět

pomocí víceblokových V/V operací, to znamená, že při jedné V/V operaci přečte *N* bloků. Počet bloků lze nastavit pomocí parametru DB_FILE_MULTIBLOCK_READ_COUNT.

Obecně platí, že optimalizátor zvolí metodu úplného prohledávání v případě, pokud nelze použít jinou cestu nebo když má jiná cesta vyšší cenu. Oracle ve své dokumentaci [15] uvádí několik typických důvodů pro úplné procházení tabulky:

- Neexistují indexy nebo je nelze použít – Pokud nad tabulkou nejsou vytvořeny žádné indexy nebo indexy nelze použít, databáze je nucena k úplnému prohledání tabulky. Databáze nemůže použít například v případě, kdy je použit jiný index než funkční a predikát v dotazu aplikuje funkci nad indexovaným sloupcem.
- Tabulka je malá – V takovém případě je většinou rychlejší úplné procházení, než například metoda Index Range Scan.
- Jsou zastaralé statistiky – Příkladem může být malá tabulku, do které je následně přidáno několik sloupců a tisíce řádků. Pokud jsou v takovém případě zastaralé statistiky, pak neodráží velikost tabulky a optimalizátor nemůže vědět, že má použít index, který je v takovém případě efektivnější.
- Použití hintu pro úplné prohledávání – Při použití hintu FULL databáze vždy použije metodu Full Table Scan.

Table Access by Rowid

Hodnoty ROWID jsou v databázovém systému Oracle fyzickými adresami dat. Adresa ROWID konkrétního řádku určuje soubor a blok obsahující daný řádek a přesné umístění řádku v tomto bloku. Pokud databáze zná adresu ROWID, pak je získání řádku velmi snadné. Databáze pouze přečte blok a podle hodnoty ROWID určí požadovaný řádek v bloku. Prohledávání pomocí ROWID je tou nejrychlejší cestou k určitému řádku, ale určitě není nejrychlejší v případě, kdy je potřeba načíst tisíce řádků [5, s. 487].

Ve většině případů přistupuje databáze k řádkům tabulky pomocí adresy ROWID až po skenování indexů. Nicméně ani při použití indexů se Oracle pro využití přístupové cesty založené na ROWID nemusí v některých případech rozhodnout. Nemusí se například použít, pokud index obsahuje všechny potřebné sloupce. V takovém případě databáze spíše použije metodu Index Fast Full Scan [15].

Sample Table Scans

Při použití metody Sample Table Scan (Procházení vzorku tabulky) načte databáze náhodnou část řádků z tabulky nebo i ze složitějšího dotazu zahrnujícího pohledy a spojení [15].

Oracle tuto metodu použije tehdy, když je v klauzuli FROM použita jedna z následujících klauzulí [15]:

- SAMPLE (procenta) – databáze načte specifikované procento řádků
- SAMPLE BLOCK (procenta) – databáze načte specifikované procento bloků

Index Unique Scans

Při prohledávání pomocí metody Index Unique Scans (procházení jedinečného indexu) databáze pozná, že indexované sloupce jsou jedinečné [5, s. 490-491]. Po prohledání indexu pomocí této metody je vždy vrácen maximálně 1 řádek.

Databázový systém Oracle tuto metodu použije, pokud dotaz obsahuje podmínku s predikátem rovnosti na sloupec, nad kterým je vytvořen unikátní index [15].

Index Range Scans

Při prohledávání pomocí metody Index Range Scan (prohledávání rozsahů indexu) databáze očekává vrácení žádného, jednoho nebo i více řádků. Rozsahy indexu mohou být prohledány jedním nebo dvěma směry. Většinou databáze přečte index ve vzestupném pořadí, tzn. od nejnižší hodnoty po nejvyšší, ale nemusí tomu tak být vždy, index může být přečten i obráceně v sestupném pořadí [5, s. 491].

Optimalizátor většinou tuto metodu použije, pokud dotaz obsahuje podmínku s těmito operátory: =, <, > [15].

Index Full Scans

Index Full Scan neboli úplné procházení indexu, zpracuje všechny listové bloky indexu, ale pouze tolik bloků větví, kolik je potřeba k nalezení prvního listového bloku. Thomas Kyte [5, s. 491] uvádí, že nalezení prvního listového bloku je dostačující, protože struktura B-tree indexu je hierarchicky uspořádaná a jednotlivé listové bloky obsahují ukazatele na následující i předchozí bloky. Při úplném prohledávání indexu se používají jednoblokové V/V operace a data jsou čtena tak, jak se nachází ve struktuře B-

tree indexu, to znamená v seřazeném pořadí a proto není databáze nucena provádět operaci řazení.

Lance Ashdown [15] popsal následující případy, kdy Oracle zvažuje použití této metody:

- Predikát dotazu obsahuje sloupec, který se nachází v indexu.
- V predikátu dotazu není specifikovaný daný sloupec, ale všechny sloupce tabulky jsou součástí indexu a alespoň jeden sloupec v indexu neobsahuje hodnoty NULL.
- Dotaz obsahuje klauzuli ORDER BY nad indexovaným sloupcem, ve kterém se nevyskytují hodnoty NULL.

Index Fast Full Scans

Index Fast Full Scan (úplné rychlé procházení indexu) se od metody Index Full Scan významně liší. Úplné rychlé prohledávání načítá všechny bloky indexu, včetně bloků větví a samotná data nejsou čtena v seřazeném pořadí [5, s. 494-495]. Celá struktura indexu může být přečtena daleko rychleji než při metodě Index Full Scan, protože jsou použity víceblokové V/V operace.

Tato metoda se obecně použije v případě, pokud dotaz odkazuje pouze na indexované sloupce [15].

Index Skip Scans

Metoda Index Skip Scan se používá u složených indexů, kde není první sloupec zahrnut v predikátu dotazu nebo obsahuje několik různých hodnot a ostatní sloupce jich obsahují daleko více [15].

Index Join Scans

Index Join Scan, nazývaný jako metoda spojení indexů, je metoda, která může být zvolena, pokud nad tabulkou existuje více indexů, které dohromady obsahují všechny sloupce specifikované v dotazu [5, s. 495-496]. V takovém případě nemusí databáze přistupovat k tabulce a může k provedení dotazu využít několika indexů, které databáze pomocí hashovací funkce spojí.

Bitmap Index Single Value

Metoda Bitmap Index Single Value využívá k nalezení konkrétní hodnoty bitmapový index [15].

Optimalizátor se pro tuto metodu rozhodne v případě, kdy je v predikátu dotazu použit operátor rovnosti pro sloupec, nad kterým je vytvořen bitmapový index [15].

Bitmap Index Range Scans

Přístupová metoda Bitmap Index Range Scan využívá pro nalezení rozsahu hodnot bitmapový index. Metoda se řídí stejnými pravidly jako metoda Index Range Scan až na to, že místo klasického B-tree indexu používá bitmapový index [15].

Bitmap Merge

Bitmap Merge, neboli spojení bitmapy je metoda, která spojí několik bitmap do jedné a jako výsledek vrací právě tuto jednu bitmapu [15]. Optimalizátor většinou tuto metodou používá pro sloučení bitmap vygenerovaných z procházení rozsahů.

Cluster Scans

Cluster Scan (clusterové procházení) je metoda, která vrací všechny řádky se stejnou hodnotou klíče v index clusteru [15]. Optimalizátor tuto přístupovou cestu použije, pokud dotaz přistupuje k index clusteru, tedy indexově seskupené tabulce.

Hash Scans

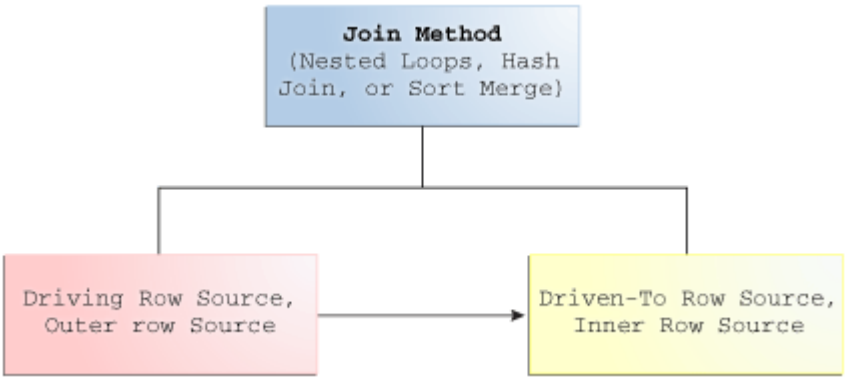
Metoda Hash Scan vrací všechny řádky se stejnou hodnotou hashe v hash clusteru [15]. Optimalizátor tuto přístupovou cestu používá, pokud dotaz přistupuje k hash clusteru, tedy k hashovací seskupené tabulce.

5.3 Metody spojení

Tato kapitola je věnována nejběžnějším metodám spojování a procesům, které při spojování nastávají. V kapitole jsou nejprve popsány nejjednodušší spojení, tedy spojení vnořenými cykly (nested loop join), poté se věnuje složitějším rozsáhlým a hromadným operacím spojení jako je spojení hash (hash join) a spojení sloučením po seřazení (sort merge join) a naposled je vysvětleno také kartézské spojení (cartesian join).

Metody spojení jsou mechanismem pro spojení dvou zdrojů a vrácení jejich výsledku jako jednořádkového setu [16]. Optimalizátor podle statistik vybere metodu s nejnižšími odhadovanými náklady.

Metoda spojení vnořenými cykly a hash spojení používají dvě tabulky, kde jedna tabulka slouží jako vnější zdroj a druhá jako vnitřní (viz obrázek 8).



Obrázek 8: Metody spojení [16]

Nested loop join

Nested loop join, neboli spojení vnořenými cykly, je pravděpodobně nejběžnější metodou spojení. Jak už samotný název napovídá, metoda je založená na procházení vnější tabulky v cyklu a pro každý řádek se hledají řádky ve vnitřní tabulce, které odpovídají podmínce spojení [16].

Podle dokumentace Oracle [16] se optimalizátor pro tuto metodu rozhodne v případě, pokud jsou spojovány tabulky s malým množstvím dat nebo pokud je spojováno velké množství a mód optimalizátoru je nastaven na FIRST_ROWS. Dalším případem, kdy optimalizátor zvolí spojení vnitřními cykly je, pokud optimalizátor vyhodnotí spojovací podmínku jako efektivní pro přístup k datům ve vnitřní tabulce.

Obecně lze říci, že metoda spojení vnořenými cykly je nejefektivnější u malých tabulek s indexovaným sloupec, který je použit v podmínce spojení.

Hash join

Hash spojení se používá především při spojování velkého množství dat. I u této metody spojení se používá vnější a vnitřní tabulka. Za optimálních podmínek použije databázový systém Oracle menší tabulku jako vnější a v paměti vytvoří odpovídající hash tabulku. Pokud není hash tabulka příliš velká, je uložena v paměti RAM. Protože jsou data z vnější tabulky uložena v paměti, může k nim databáze přistupovat velice rychle. Jakmile je vnější tabulka zatříděna do paměti, začne databáze Oracle procházet řádky vnitřní tabulky a pomocí hash tabulky uložené v paměti vybírá řádky, které odpovídají podmínce spojení. Protože se vnější tabulka nachází v privátní paměťové oblasti PGA, nevyvolá

přístup k datové struktuře blokování, které by způsobila logická V/V operace a je tím snížen počet logických V/V operací. Při použití hash spojení se sice musí čekat, než databáze provede úplné prohledání vnější tabulky a její zatřídění do paměti, poté jsou však zbývající řádky vraceny velmi rychle, někdy i rychleji než je možné je přijímat. Zpracování se mírně liší, pokud se řádky menší tabulky nevejdou do paměti. V takovém případě je tabulka rozdělena na oddíly a ty jsou postupně spojovány. To má samozřejmě negativní vliv na spotřebu oblasti PGA a dochází k častému čtení z dočasného tabulkového prostoru TEMP. Problematikou se více zabývá ve své knize Thomas Kyte [5, s. 501-502].

Optimalizátor se pro použití této metody rozhodne v případě spojování velkého množství dat, kdy je podmínka spojení definována operátorem rovnosti [16].

Sort merge join

Sort merge join, tedy spojení sloučením po seřazení nepoužívá vnější a vnitřní tabulky, v této koncepci se naprosto liší od spojení vnitřními cykly a spojení hash. Při použití této metody spojení nejprve databáze seřadí data v obou tabulkách a teprve poté provede operaci spojení [5, s. 504]. Tento druh spojení je mnohem méně používán a obecně lze říci, že je méně efektivní než hash spojení. Důvodem je především nutnost prohledání a seřazení dat v obou vstupních tabulkách.

Spojení sloučením po seřazení databáze Oracle použije v případě, kdy není v podmínce pro spojení použit predikát rovnosti, ale místo toho je použit jeden z těchto predikátů: <, <=, >, >= nebo pokud je seřazení vyžadováno jinou operací v dotazu a optimalizátor jí shledá méně nákladnou, než jinou metodu [16].

Cartesian join

Kartézské spojení, někdy též kartézský produkt, databáze použije tehdy, když jsou v dotazu uvedeny tabulky bez podmínky spojení. Optimalizátor spojí každý řádek první tabulky s každým řádkem druhé tabulky a tím vytvoří kartézský součin [16]. U většího množství dat v tabulkách je kartézské spojení velice nákladnou operací, protože vrací M*N řádků, kde M značí počet dat z první tabulky a N je počet dat z tabulky druhé.

5.4 Exekuční plán

Exekuční plán neboli prováděcí plán definuje jednotlivé kroky, které bude databáze vykonávat při provedení SQL dotazu. Každý krok buď fyzicky načítá data z databáze, nebo je pro dotaz připravuje. Při zobrazení konkrétního plánu lze zjistit jeho

cenu, uvedenou na prvním řádku, a ceny jednotlivých operací. Cena je interní metrika databázového systému Oracle, díky které lze obecně porovnávat efektivnost jednotlivých exekučních plánů pro konkrétní dotazy. Vytváření a celkovou správu exekučních plánů má na starosti optimalizátor, který rozhoduje i o výběru nejlepšího exekučního plánu pro daný dotaz v danou chvíli. Exekuční plán pro jeden konkrétní dotaz není totiž stále stejný, ale mění se v průběhu času a v závislosti na konkrétním prostředí (různé databázové schéma, různé statistiky, atp.). Exekučnímu plánu se více věnuje autor dokumentace k databázovému systému Oracle Lance Ashdown [17].

5.4.1 Možnosti generování a zobrazení plánu

Zobrazením exekučního plánu dotazu lze zjistit mnoho užitečných informací, například pro jaké přístupové cesty a metody spojení se optimalizátor rozhodl a z jakého důvodu tak učinil. Proto jsou exekuční plány jedním z nejefektivnějších zdrojů při plánování a správě indexů a ladění výkonu SQL dotazů. Dále jsou popsány nástroje umožňující vygenerování a zobrazení exekučního plánu.

Příkaz EXPLAIN PLAN

Příkaz EXPLAIN PLAN je jeden ze způsobů, jak zjistit přesný exekuční plán pro konkrétní SQL dotaz. Tento příkaz zapříčiní vložení jednotlivých kroků exekučního plánu do tabulky PLAN_TABLE, případně lze v příkazu určit jinou tabulku, do které má být exekuční plán vložen [17]. Příkaz má následující syntaxi:

```
EXPLAIN PLAN
  SET STATEMENT_ID = 'statement_id'
  INTO nazev_tabulky
  FOR sql_prikaz;
```

Kód 17: Syntaxe příkazu explain plan

V příkazu je použito maximální nastavení, které je možné, avšak možnost *set statement_id* a *into nazev_tabulky* nejsou povinné. První možnost pouze nastaví identifikátor pro daný SQL dotaz, tím je exekuční plán v tabulce jednoduše rozpoznatelný od ostatních. Pokud v příkazu není nastaven, daný sloupec v tabulce plánů bude prázdný. Druhá možnost určuje tabulku, do které bude plán vložen. Tabulka musí mít standardní strukturu tabulky plánů. Pokud není určena, je jako výchozí použita tabulka PLAN_TABLE. Jedinou povinnou klauzulí v příkazu je *for sql_prikaz*. Tím je databázi předán SQL příkaz, pro který má vygenerovat exekuční plán.

Tabulka PLAN_TABLE

Tabulka PLAN_TABLE je při výchozím nastavení použita pro ukládání exekučních plánů. Databáze Oracle umožňuje vytvoření vlastní tabulky plánů pomocí skriptu **utlxplan.sql**, který se nachází v $ORACLE_HOME/RDBMS/ADMIN [8, s. 36]. Před verzí Oracle 10g si každý uživatel musel tabulku pro exekuční plány vytvářet, proto Oracle vytvořil globální dočasnou tabulku PLAN_TABLE, která je přístupná všem uživatelům [8, s. 36]. Protože se jedná o dočasnou tabulku, každý uživatel vidí pouze své exekuční plány a pouze po dobu trvání své relace.

Pohled V$SQL_PLAN

Místo příkazu EXPLAIN PLAN a následného zobrazení plánu z plánovací tabulky se lze na exekuční plán dotazovat do pohledu V$SQL_PLAN. Tento pohled obsahuje exekuční plány pro všechny dotazy uložené ve sdílené paměti [17]. Definice pohledu je velmi podobná plánovací tabulce PLAN_TABLE.

Funkce DBMS_XPLAN.DISPLAY

Jedním z nejběžnějších způsobů zobrazení výstupu příkazu EXPLAIN PLAN je databázová funkce DISPLAY z balíku DBMS_XPLAN. Definice hlavičky funkce vypadá následovně:

```
DBMS_XPLAN.display (
   table_name     VARCHAR2 DEFAULT 'PLAN_TABLE',
   statement_id   VARCHAR2 DEFAULT NULL,
   format         VARCHAR2 DEFAULT 'TYPICAL',
   filter_preds   VARCHAR2 DEFAULT NULL
);
```

Kód 18: Zobrazení exekučního plánu pomocí funkce dbms_xplan.display

Jak je vidět z definice hlavičky, funkce lze volat se čtyřmi parametry, přičemž každý parametr má nastavené defaultní hodnoty a je tedy možné volat funkci i bez parametrů. Guy Harrison [8, s. 42-43] popsal jednotlivé parametry funkce:

1. *table_name* – Název tabulky, ve které je plán uložen. Pokud se v parametru funkce název tabulky nepředá, databáze Oracle automaticky prohledává tabulku PLAN_TABLE.
2. *statement_id* – Identifikátor SQL dotazu, pokud byl v příkaze EXPLAIN PLAN nastaven. V případě, že se ve funkci nepředá, databáze zobrazí plán pro naposledy spuštění SQL dotaz.

3. *format* – Nastavení úrovně zobrazení. Oracle nabízí tři základní možnosti TYPICAL, BASIC a ALL, a dále několik modifikátorů, pomocí kterých lze výstup detailněji specifikovat.
    - BASIC – Ve výstupu budou uvedeny pouze informace o ID, Operaci a názvu objektu zdroje (tabulka, pohled, atp.).
    - TYPICAL – Výchozí formát zobrazení. Toto nastavení rozšiřuje předchozí nastavení o informace jako je cena, odhadovaný počet zpracovaných bajtů, odhadovaný čas na procesor, atd. Rozšiřující informace o projekci a další informace jsou zobrazeny pouze v případě, že je DBMS_XPLAN považuje za přímo související s exekučním plánem.
    - ALL – Budou zobrazeny všechny informace i přesto, že nemusí přímo souviset s exekučním plánem. Ve výstupu se dále zobrazují i názvy bloků a aliasy objektů.

Předchozí možnosti lze dále upravit pomocí několika modifikátorů. Tyto modifikátory se uvádí přímo v parametru pro formát zobrazení a uvozují se znaky +/- podle toho, zda mají být do výstupu zahrnuty či nikoliv. Pomocí těchto modifikátorů lze nastavit přesný výstup, jaký je uživatelem požadován. Následuje popis jednotlivých modifikátorů [8, s. 44]:

- BYTES – Odhadovaný počet zpracovaných bajtů v jednotlivých operacích.
- COST – Odhadovaná cena pro jednotlivé operace.
- PARTITION – Informace týkající se oddílů. Pouze v případě, kdy jsou základní tabulky do oddílů rozděleny.
- PARALLEL – Informace, které se týkají paralelního zpracování.
- PREDICATE – Informace o predikátech (v případě, že je v dotazu uveden predikát JOIN nebo WHERE)
- PROJECTION – Informace o projekci. Ve výstupu jsou uvedeny zpracovávané sloupce atd.
- ALIAS – Zobrazuje aliasy objektů při zpracování vzdáleného nebo paralelního dotazu.
- REMOTE – Zobrazuje informace o vzdáleném SQL. V případě používání distribuovaných dotazů.
- NOTE – Další různé poznámky týkající se exekučního plánu.

- IOSTATS – Zobrazení statistik o V/V operacích. Tyto statistiky budou zobrazeny pouze v případě, že je parametr STATISTICS_LEVEL nastaven na hodnotu ALL nebo v případě použití hintu GATHER_PLAN_STATS.
- MEMSTATS – Zobrazení informací o využití paměti a disku pro operace řazení a spojení.
- ALLSTATS – Zobrazení všech statistik. Alternativa k použití +IOSTATS +MEMSTATS.

Autotrace

Autotrace je vestavěnou funkcí nástroje SQL*Plus. Funkce je velmi podobná příkazu EXPLAIN PLAN, avšak jeden velký rozdíl mezi těmito nástroji je. Příkaz EXPLAIN PLAN poskytuje informace o tom, jaké kroky databáze při spuštění udělá, ale samotný příkaz není vykonán. U funkce Autotrace tomu tak není, výstup totiž zobrazí až po skutečném provedení dotazu a díky tomu umožňuje uživateli udělat si představu o tom, co skutečné provedení dotazu vyžaduje. Exekuční plány generované pomocí Autotrace jsou vytvořeny pomocí funkcí z balíku DBMS_XPLAN. Kromě exekučního plánu jsou generovány i statistiky zahrnující informace jako je počet čtení z disku a z paměti nebo síťové statistiky. V případě, že je Autotrace aktivní, jsou statistiky i exekuční plány generovány po každém provedení DML příkazu. Autotrace má několik možností nastavení, kterými lze ovlivnit jeho chování:

- SET AUTOTRACE OFF – Výchozí nastavení databáze Oracle. Funkcí Autotrace nejsou generovány žádné sestavy.
- SET AUTOTRACE ON – Autotrace generuje exekuční plány i statistické informace týkající se provedení SQL příkazu.
- SET AUTOTRACE EXPLAIN – Autotrace generuje pouze průběh výpočtu optimalizátoru, tedy exekuční plány.
- SET AUTOTRACE STATISTICS – Autotrace generuje pouze statistiky týkající se provedení SQL příkazu.
- SET AUTOTRACE TRACEONLY – Toto nastavení je velmi podobné příkazu SET AUTOTRACE ON s výjimkou vynechání uživatelského SQL dotazu ve výstupu. Takové nastavení je vhodné ve chvíli, kdy dotazy vracejí tisíce řádků, a při generování výstupu by se na tyto řádky muselo čekat.

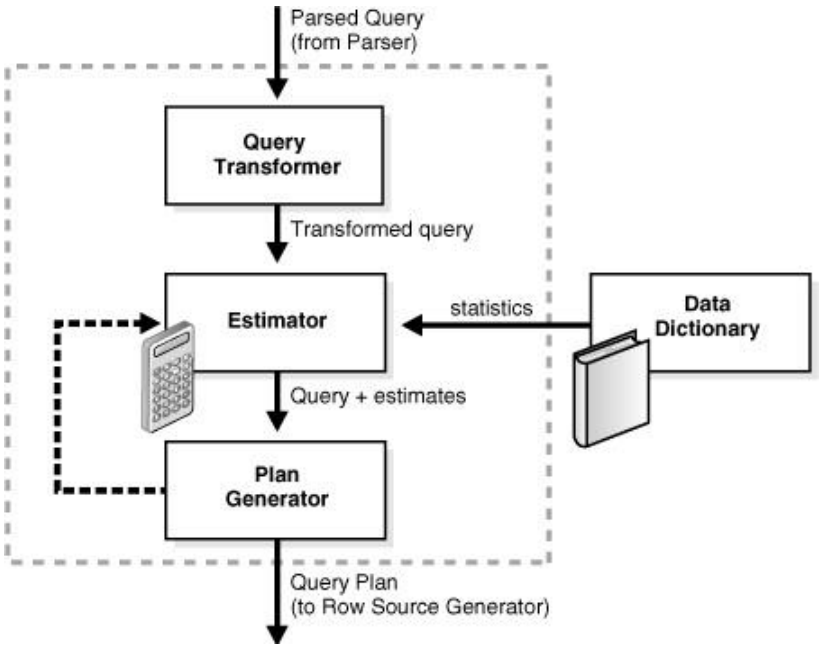
Problematice funkce Autotrace se více věnuje Simon Watt [18] v dokumentaci k databázi Oracle.

5.5 Optimalizátor

Lance Ashdown [19] popisuje optimalizátor jako vestavěný nástroj databáze Oracle, který rozhoduje o výběru nejefektivnější metody vykonání daného příkazu. Jeho úkolem je generovat co nejlepší exekuční plán pro SQL příkazy. Nejlepší exekuční plán je definován jako plán s nejnižší cenou na provedení příkazu. Výpočet ceny ovlivňuje nespočet faktorů, mezi které patří například systémové zdroje (operační paměť, CPU, V/V operace), velikost datových setů, uspořádání dat, struktura databáze, množství dat v tabulkách a množství dat vrácených dotazem, včetně toho, jak je samotný SQL dotaz napsaný. Tento přístup, který je založen na cenách dotazů, je nazývaný jako optimalizátor založený na cenách (**CBO** – Cost-Based Optimization).

Optimalizátor při výběru nejlepšího exekučního plánu pro SQL dotaz zkoumá tisíce přístupových cest a volí mezi různými druhy spojení. Pro taková rozhodnutí používá interní statistiky, z tohoto důvodu je optimalizátor ve většině případů efektivnější než kdyby exekuční plán sestavoval sám uživatel [19]. Proto je optimalizátor využíván ve všech SQL příkazech.

Oracle optimalizátor obsahuje tři hlavní komponenty, které lze vidět na následujícím obrázku:



Obrázek 9: Komponenty optimalizátoru [19]

5.5.1 Query Transformer

Tato komponenta již byla zmíněna v kapitole 4.5, kde byla popsána technika přepisování dotazů se zaměřením na přepis využívající materializované pohledy. Cílem komponenty je přepsat dotaz do jiné sémantické podoby a následně vyhodnotit cenu přepsaného dotazu [19]. Přepsaný dotaz samozřejmě vrací naprosto stejná data, ve většině případů s nižší cenou dotazu. Pokud je tedy cena přepsaného dotazu nižší než cena původního, použije pro exekuční plán nově přepsaný dotaz.

Oracle ve své dokumentaci [20] uvádí následující metody, které query transformer při přepisování dotazů používá:

OR Expansion

Pokud dotaz obsahuje v klauzuli WHERE operátor OR, Query Transformer takový dotaz nahradí za dva dotazy spojené operátorem UNION ALL. Databáze tuto metodu používá z různých důvodů. Například tím může umožnit zvolení lepších přístupových cest nebo lepších metod spojení.

View Merging

Používá se v případě dotazování se nad pohledy. Query Transformer sloučí dotaz pohledu se SQL dotazem, který daný pohled využívá.

Predicate Pushing

Při této metodě optimalizátor přesouvá predikáty konkrétního dotazu do dotazu definovaném v pohledu, na který nebyla aplikována metoda View Merging. V pohledech, u kterých nebyla použita metoda View Merging tím lze zvýšit efektivitu plánu, protože predikáty mohou být využity v indexech nebo jako filtrační kritéria už v dotazu pohledu.

Subquery Unnesting

Pokud dotaz obsahuje v podmínce WHERE poddotaz, Query Transformer ho transformuje do spojení pomocí operace JOIN, pokud je to možné. Tuto transformaci může optimalizátor provést pouze, pokud je jisté, že vrátí ty samé řádky jako původní dotaz a pokud poddotaz neobsahuje agregační funkce.

Query Rewrite with Materialized Views

Tuto metodu optimalizátor použije v případě existence materializovaného pohledu nad základní tabulkou. Optimalizátor nejprve zkontroluje, jestli je daný dotaz kompatibilní

s dotazem v definici materializovaného pohledu. Pokud ano, tak Query Transformer přepíše dotaz s použitím materializovaného pohledu. Této metodě byla věnována kapitola 4.5.2.

Star Transformation

Transformaci lze použít pouze v případě použití Hvězdicového schéma. Query Transformer se snaží předejít úplnému procházen tabulky faktů tím, že načítá pouze relevantní řádky. V případě, že dotazy obsahují omezující filtrační predikáty pro sloupce v tabulce dimenzí, může zkombinováním těchto filtrů výrazně snížit výpočetní čas.

Table Expansion

Používá se u tabulek, které jsou rozděleny na oddíly. Díky této metodě může optimalizátor použít index nad nejčtenější části rozdělené tabulky.

Join Factorization

Metoda pracuje s dotazy obsahujícími klauzuli UNION ALL. V případě, že takto spojené dotazy pracují alespoň částečně se stejnými tabulkami, provede optimalizátor společné výpočty pouze jednou pro všechny tyto dotazy. Tím se může optimalizátor vyhnout několikanásobnému úplnému procházení tabulek. Bez použití této metody musí databáze počítat každý kus dotazu nezávisle, čímž opakuje stejné výpočty včetně rozhodování o přístupových cestách a metodách spojení.

5.5.2 Estimator

Estimátor, je další komponentou Oracle optimalizátoru, jehož cílem je odhadnout celkové náklady daného exekučního plánu. Pro stanovení celkové ceny používá tři měřítka [19]:

- **Selektivita** – Vyjadřuje procento řádků z celkového množství dat v tabulce, které vyhovují predikátům v klauzuli WHERE. Hodnota selektivity je v rozmezí 0 až 1, přičemž predikáty s hodnotou blížící se k hodnotě 0 vyjadřují vyšší míru selektivity.
- **Kardinalita** – Vyjadřuje odhadovaný počet řádků, vrácený každou operací exekučního plánu. Estimator odhaduje kardinalitu pomocí shromažďovaných statistik z DBMS_STATS nebo pomocí predikátů dotazu či funkcí jako jsou DISTINCT, GROUP BY a dalších. Pokud je poměr hodnot ve sloupci s použitím

funkce DISTINCT vůči celkovém počtu řádků v tabulce nízký, pak se jedná o nízkou kardinalitu.

- **Cena** – Interní metrika databáze Oracle, která představuje odhadované využití systémových zdrojů pro daný plán. Nižší cena znamená menší využití zdrojů, tedy efektivnější exekuční plán.

5.5.3 Plan generator

Plan generator, tedy generátor exekučních plánů je třetí a poslední komponentou optimalizátoru. Jeho cílem je generovat různé exekuční plány pro konkrétní dotaz a vybrat plán s nejnižší cenou [19]. Pro jednotlivé plány jsou použity různé přístupové cesty, různé metody spojení, které jsou použity v různém pořadí. Díky velkému množství kombinací přístupových cest a metod spojení může být generováno obrovské množství exekučních plánů. Estimátor pak odhadne jejich cenu a generátor plánů tak může vybrat plán s nejnižší cenou.

Oracle ve své dokumentaci [19] popisuje vnitřní mechanismus, který optimalizátor používá pro včasné ukončení generování exekučních plánů. Jeho úkolem je rozhodnout, zda je cena aktuálně nejlepšího plánu dostatečně nízká, pokud ano je proces ukončen a vybrán aktuální plán. Pokud však mechanizmus rozhodne, že je cena vysoká, je spuštěno generování dalších plánů.

6 Techniky pro ovlivňování optimalizátoru

Obecně platí, že výchozí chování optimalizátoru je pro většinu operací naprosto dostačující. Nicméně, v některých případech může mít uživatel informace, které optimalizátor nezná nebo nemůže použít pro optimalizaci dotazu. V takových případech lze prostřednictvím několika technik optimalizátor ovlivnit a dosáhnout tak lepšího výkonu. Jedná se o techniky, jako jsou SQL profily, SQL Plan Management, inicializační parametry, hinty nebo i správná aktualizace statistik.

6.1 Inicializační parametry

Databáze Oracle zahrnuje několik inicializačních parametrů, kterými lze ovlivnit chování optimalizátoru, vybrané důležité parametry jsou uvedeny v tabulce 2.

Inicializační parametr	Popis
CURSOR_SHARING	Převádí hodnoty literálů na vázané proměnné. Používání vázaných proměnných zvyšuje možnost sdílení kurzorů a může kladně ovlivnit exekuční plány.
DB_FILE_MULTIBLOCK_READ_COUNT	Udává počet bloků, které bude databáze Oracle číst v jedné V/V operaci při úplném prohledávání nebo rychlém úplném prohledávání indexu.
OPTIMIZER_ADAPTIVE_REPORTING_ONLY	Řídí mód hlášení pro reoptimalizaci a adaptivní plány.
OPTIMIZER_MODE	Nastaví režim optimalizátoru při spuštění instance databáze. Možné hodnoty jsou ALL_ROWS (co nejdříve načte všechny řádky), FIRST_ROWS_n (co nejdříve načte n řádků, kde n nabývá hodnot 1,10,100,1000), FIRST_ROWS (co nejdříve načte první řádek).
OPTIMIZER_INDEX_CACHING	Vyjadřuje procento bloků indexu ve vyrovnávací paměti. Parametr ovlivňuje cenu exekučních plánů, protože ho optimalizátor bere v potaz při určování očekávaného množství V/V operací. Parametr může nabývat hodnot od 0 do 100.
OPTIMIZER_INDEX_COST_ADJ	Parametr ovlivňuje vypočítané náklady na přístup pomocí indexu. Čím menší hodnota je nastavena, tím levněji bude přístup pomocí indexu posouzen. Parametr může nabývat hodnot od 1 do 1000.
OPTIMIZER_USER_INVISIBLE_INDEXES	Parametr zapne nebo vypne používání neviditelných indexů.

Tabulka 2: Inicializační parametry pro ovlivnění chování optimalizátoru [21]

6.2 Hinty

Hinty jsou jakýmisi instrukcemi pro optimalizátor, které jsou vpisovány do SQL dotazu. Vznikly z jednoho prostého důvodu a tím je, že uživatelé někdy mají obsáhlejší informace o objektech v databázi, než má samotný optimalizátor. Hinty dovolují dělat rozhodnutí, které běžně provádí optimalizátor a díky tomu lze mnohdy zvolit efektivnější exekuční plán. Umožňují například specifikovat konkrétní přístupovou cestu, pořadí a metodu spojení, nebo index, který má být použit [8, s. 212].

Hint se vpisuje jako komentář za první slovo SQL příkazu, kterým může být SELECT, INSERT, UPDATE, MERGE, nebo DELETE [8, s. 212]. Od ostatních komentářů se hint liší tím, že ho uvozuje znak (+), jež následuje po znacích otevření komentáře (/*).

Příkladem může být hint FULL v následujícím SQL dotazu, který optimalizátoru přikazuje použít přístupovou cestu full table scan a to dokonce i tehdy, když optimalizátor vypočítá použití indexu jako levnější operaci.

```
SELECT /*+ FULL(objednavka)*/ *
FROM objednavka
WHERE datum_objednani < (SYSDATE - INTERVAL '10' DAY) ;
```

Kód 19: Použití hintu v dotazu

Nevýhodou používání hintů je další kus kódu, který je nutné spravovat. Protože může mít jakákoliv změna v databázi v hostitelském prostředí negativní vliv na výkon dotazů při použití hintů, tak se v praxi používají spíše pro účely testování a pro ovlivňování optimalizátoru jsou využívány jiné techniky [21]. I samotný Oracle doporučuje používat jiné techniky a nástroje, které skoro vždy nabízejí mnohem čistší řešení. V následující tabulce jsou uvedeny nejpoužívanější hinty.

Hint	Popis
ALL_ROWS	Použití ALL_ROWS jako cíl optimalizátoru.
APPEND	Přímé vložení řádku příkazem INSERT
CACHE(*tabulka*)	Při úplném procházení tabulky jsou umístěny bloky tabulky do vyrovnávací paměti. Opačný efekt má hint NOCACHE.
FACT(*tabulka*)	Tabulka je optimalizátorem považována za tabulku faktů. Používá se u schématu hvězda.
FIRST_ROWS(*N*)	Použití FIRST_ROWS(N) jako cíl optimalizátoru.
FULL(*tabulka*)	Optimalizátor použije jako přístupovou cestu

	Full Table Scan.
HASH(*tabulka*)	Optimalizátor použije jako přístupovou cestu Hash Scan. Hint funguje pouze u hash clusterů.
INDEX(*tabulka[index]*)	Tabulka je procházena pomocí indexu. V případě, že index specifikovaný v hintu neexistuje, tak je použit index s nejnižší cenou. Opačný efekt má hint NO_INDEX.
INDEX_COMBINE(*tabulka index index...*)	Optimalizátor použije pro přístup do tabulky více indexů.
INDEX_SS(*tabulka index*)	Optimalizátor použije přístupovou metodu Index Skip Scan.
LEADING(*tabulka...*)	Specifikuje pořadí, ve kterém budou dané tabulky spojeny.
ORDERED	Optimalizátor použije pro spojení takové pořadí, jaké je uvedeno v klauzuli FROM.
USE_HASH(*tabulka*)	Optimalizátor použije jako metodu spojení Hash join.
USE_MERGE(*tabulka*)	Optimalizátor použije jako metodu spojení Sort merge join.
USE_NL(*tabulka*)	Optimalizátor použije jako metodu spojení Nested loop join.

Tabulka 3: Nejpoužívanější hinty [8, s. 212-213]

6.3 Statistiky

Statistiky jsou nejdůležitějším zdrojem informací pro optimalizátor, který pomocí nich odhaduje cenu jednotlivých exekučních plánů. Optimalizátor shromažďuje statistiky o různých typech databázových objektů a o vlastnostech prostředí databáze. Oracle ve své dokumentaci [22] uvádí následující typy statistik:

- Tabulkové statistiky – Udržují informace o počtu řádků, počtu bloků, průměrné délce řádku.
- Sloupcové statistiky – Udržují informace o počtu různých hodnot ve sloupci, počtu NULL hodnot, histogramech, rozšířených statistikách atp.
- Indexové statistiky – Udržují informace o počtu listových bloků, počtu úrovní atp.
- Systémové statistiky – Udržují informace o využívání procesoru a V/V operacích.

Databázový systém Oracle obsahuje několik mechanismů, které se o shromažďování statistik starají, jedním z nejdůležitějších je package DBMS_STATS.

Automatický sběr statistik pomocí DBMS_STATS

Databázový systém Oracle zajišťuje automatický sběr statistik voláním procedury DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC. Procedura sbírá převážně

statistiky pro objekty, které buď žádné nemají, nebo jsou zastaralé z důvodu změn u objektu. Aby databáze sbírala informace o změnách u objektů, je nutné nastavit inicializační parametr STATISTICS_LEVEL na hodnotu TYPICAL nebo ALL. Ve výchozím nastavení je automatické shromažďování statistik povoleno, případně jej lze nastavením package DBMS_AUTO_TASK_ADMIN zakázat. Problematice statistik a jejich automatickému sběru se více věnuje Guy Harrison [8, s. 196-198].

6.4 SQL Profily

SQL Profil je databázový objekt obsahující pokročilé statistiky a podrobné informace o prostředí pro konkrétní SQL příkaz, jehož cílem je poskytnout optimalizátoru co možná nejdetailnější informace, díky kterým dokáže sestavit ještě efektivnější exekuční plán [23].

Pro správu a řízení SQL Profilů je využíván nástroj SQL Tuning Advisor (viz kapitola 7.3). Ten na vzorky dat používá specifické vstupní hodnoty a porovnává je s odhadovanými výsledky optimalizátoru [23]. V případě, že SQL Tuning Advisor zjistí významné odchylky, provede společně s SQL Profilerem nápravná opatření a doporučí databázi jejich přijetí.

Statistiky v SQL profilu pomáhají lépe odhadnout kardinalitu dotazu a tím umožní optimalizátoru vybrat lepší plán. Podle dokumentace k Oracle [23] poskytují SQL profily oproti jiným technikám optimalizace následující výhody:

- Na rozdíl od hintů nejsou SQL profily svázané s konkrétním exekučním plánem. SQL profily pomáhají opravit špatné odhady a umožňují flexibilitu optimalizátoru při výběru exekučního plánu v různých situacích.
- Při používání SQL profilů, na rozdíl od hintů, není potřeba zasahovat do zdrojových kódů dotazů.

6.5 SQL Plan Management

SQL Plan Management je mechanismus, který umožňuje optimalizátoru automatickou správu exekučních plánů a zajišťuje, že databáze používá pouze známé a ověřené plány [24]. Toho dosáhne tím, že si ukládá veškeré informace o exekučních plánech, jako je například identifikátor plánu, sadu hintů, vázané proměnné, nebo informace o prostředí [24].

Hlavním cílem SQL plan managementu je zabránit výkonnostním problémům způsobených změnou plánů. Druhým, neméně důležitým cílem, je přizpůsobit se nově přidaným indexům a aktualizovaným statistikám a přijmout plán pouze v případě, že dojde ke zvýšení výkonu [24]. K tomu používá mechanismus zvaný **SQL plan baseline**, což je sada akceptovaných exekučních plánů, které může optimalizátor pro daný SQL příkaz použít [24].

Lance Ashdown [24] uvádí tři po sobě jdoucí kroky, na nichž je SQL plan management založen:

1. **Zachycení plánu** – V tomto kroku jsou uloženy důležité informace o plánech pro dané SQL příkazy.
2. **Výběr plánu** – Při tomto kroku jsou optimalizátorem detekovány změny v dříve uložených plánech a pomocí SQL plan baseline jsou vybrány takové plány, které nezapříčiní zhoršení výkonu.
3. **Rozvoj plánu** – Tento krok zodpovídá za zpracování nově přidaných plánů k již existujícím SQL dotazům v SQL plan baseline.

Rozdíl mezi SQL profily a SQL plan baseline

SQL profily a SQL plan baseline mají spoustu věcí společných, jejich cílem je zvýšit výkon SQL dotazů tím, že poskytují optimalizátoru takové informace, díky kterým je schopen vybrat optimální exekuční plán. Oba dva mechanismy interně používají hinty. I přesto jsou mechanismy v několika ohledech naprosto rozdílné.

Oracle ve své dokumentací [24] uvádí, že SQL plan baseline je založen na proaktivním přístupu, kdežto SQL profily jsou založeny na reaktivním. To znamená, že SQL plan baseline se aplikuje ještě předtím, než dojde k problémům s výkonem, tím že optimalizátoru zabraňuje používat suboptimální plány. Naproti tomu, SQL profily se typicky aplikují až po zjištění výkonnostních problémů. SQL Profily jsou užitečné především tím, že napravují chyby optimalizátoru, které vedou k suboptimálním plánům. Protože jsou SQL profily založené na reaktivním přístupu, nemůžou v případě větších databázových změn zaručit stabilní výkon aplikace. Dalším rozdílem je fakt, že snahou SQL plan baseline je reprodukovat konkrétní exekuční plán na rozdíl od SQL profilů, jejichž snahou je opravovat špatné odhady estimátoru.

7 Detekce problematických SQL a možnosti ladění

Při optimalizaci je nezbytným úkolem najít SQL dotazy, které způsobují snížení výkonu databáze. Takové dotazy nadměrně využívají systémové prostředky a proto je potřeba těmto dotazům věnovat pozornost a zaměřit se na jejich optimalizaci.

Vyhledání problematických SQL dotazů může být v mnoha případech časově velmi náročné, Oracle proto nabízí několik nástrojů, které s vyhledáním takových dotazů pomáhají a často i navrhují řešení pro jejich optimalizaci. Tato kapitola je věnována právě takovýmto nástrojům.

7.1 AWR

AWR (Automatic Workload Repository) je úložiště, které uchovává a udržuje výkonové statistiky pro účely detekce a ladění problematických SQL. Tento nástroj je v databázovém systému Oracle od verze 10g a nahradil tak nástroj STATSPACK. Pomocí úložiště AWR lze generovat různé přehledy, ve kterých je možné sledovat využívání jednotlivých příkazů SQL, aktivitu relací i celkové statistické údaje. Databáze Oracle ukládá údaje o systému každou hodinu a ve výchozím stavu udržuje informace za posledních 7 dní. AWR je vysoce integrováno s nástrojem OEM[2], pomocí kterého lze snadno provádět analýzy a opravovat výkonové komplikace.

AWR lze aktivovat nastavením inicializačního parametru STATISTICS_LEVEL na hodnotu TYPICAL nebo ALL. V případě nastavení hodnoty na BASIC nebudou údaje tak podrobné a sběr je potřeba provádět manuálně.

Sestavy je možné z úložiště AWR generovat buď využitím uživatelského rozhraní nástroje OEM, nebo pomocí dodaných SQL skriptů. Jedná se o skript awrrpt.sql, který vygeneruje sestavu založenou na rozdílech mezi počáteční a koncovými statistickými údaji a o skript awrrpti.sql, který generuje sestavu založenou na počáteční a koncové sadě dat pro zadanou databázi a instanci. Skripty se nachází ve složce $ORACLE_HOME/rdbms/admin. O problematice úložiště AWR se více zmiňuje Bob Bryla a Kevin Loney [1, s. 302-304].

[2] OEM (Oracle Enterprise Manager) je soubor internetových nástrojů zaměřených na správu SW a HW vyprodukovaných společností Oracle.

7.2 ADDM

ADDM (Automatic Database Diagnostic Monitor) je samodiagnostický nástroj, jehož cílem je pravidelná analýza dat z AWR [25]. Díky pravidelným analýzám je velmi silným nástrojem pro identifikaci výkonnostních problémů v databázi, včetně určení jejich příčiny a návrhu řešení. ADDM je hojně využívaným nástrojem především databázových administrátorů a většinou je prvním místem, kde hledají příčinu možného problému.

ADDM provádí analýzu shora dolů, to znamená, že nejprve identifikuje příznaky problému a poté jeho příčiny. Hlavním cílem těchto analýz je co nejvíce snížit hodnotu metriky DB time. DB time je základní jednotkou pro měření výkonu databáze, jedná se o kumulativní čas, který databáze stráví při zpracovávání uživatelských požadavků [25].

Snížení hodnoty DB time znamená, že databáze je schopna zpracovávat více uživatelských požadavků použitím těch samých zdrojů, což má za následek zvýšení propustnosti. Problémy nahlášené pomocí ADDM jsou seřazené právě podle celkové hodnoty DB time, takže je na první pohled vidět, které problémy nejvíce zatěžují databázi, tedy které je potřeba řešit nejdříve. Hodnoty DB time si lze zobrazit i prostřednictvím pohledů V$SESS_TIME_MODEL a V$SYS_TIME_MODEL [25].

7.3 SQL Tuning Advisor

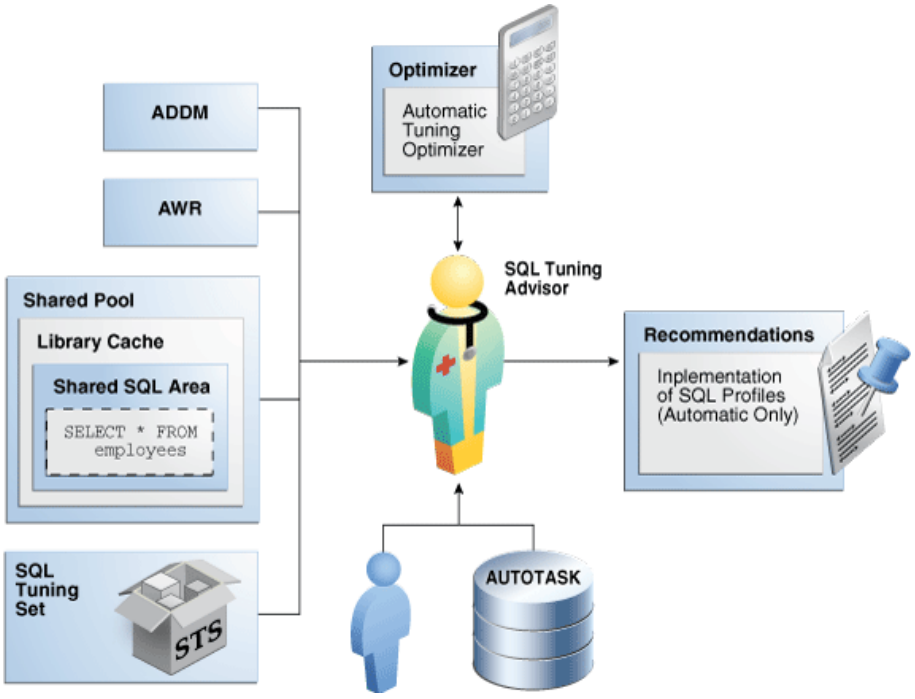
SQL Tuning Advisor je nástroj, který pomáhá řešit výkonnostní problémy SQL dotazů. Jako vstup přijímá jeden nebo i více SQL příkazů, nad kterými provádí různé analýzy a výstupem je souhrn doporučení (viz obrázek 10), jež umožňují dosáhnout následujících cílů [26]:

- **Vyhnutí se náročné manuální optimalizaci**
  SQL Tuning Advisor provádí automatickou optimalizaci dotazů prostřednictvím optimalizátoru.
- **Automatické generování doporučení a implementace SQL profilů**
  SQL Tuning Advisor umožňuje nakonfigurovat optimalizační úlohy pro SQL dotazy, které mohou být spouštěny například v noci. Pokud jsou úlohy takto spouštěny, může SQL Tuning Advisor generovat doporučení a implementovat SQL profily automaticky.

- **Analýza databázově generovaných statistik za účelem dosažení optimálních exekučních plánů**

  SQL Tuning Advisor umožňuje provádění analýz interních informací za účelem zefektivnění prováděcích plánů.
- **Možnost optimalizace SQL dotazů na testovacím systému místo produkčního**

  Pokud nastanou u SQL dotazů výkonnostní problémy na produkčním prostředí, SQL Tuning Advisor umožňuje tyto dotazy přenést na testovací prostředí, kde mohou být optimalizovány.



Obrázek 10: Architektura nástroje SQL Tuning Advisor [26]

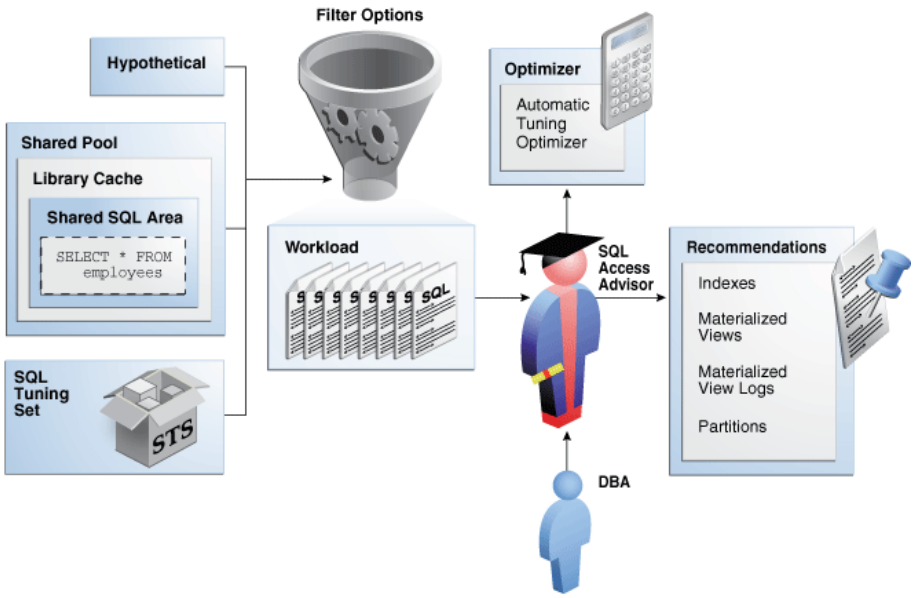
Jak je vidět na obrázku, vstupní SQL příkaz může pocházet z několika zdrojů, kterými jsou: ADDM, AWR, Sdílená SQL oblast, nebo databázový objekt STS (SQL Tuning Set).

SQL Tuning Advisor lze spouštět buď automaticky prostřednictvím tzv. Auto-tasků, u kterých je tento nástroj znám jako Automatic SQL Tuning Advisor, nebo na vyžádání. Pro spouštění nástroje manuálně lze použít buď package DBMS_SQLTUNE nebo placený nástroj Oracle Enterprise Manager Cloud Control [26].

7.4 SQL Access Advisor

SQL Access Advisor je diagnostický software, který pomáhá identifikovat a řešit problémy s výkonem SQL dotazů tím, že doporučuje použití indexů či materializovaných pohledů na konkrétních místech a pomáhá s vytvářením, mazáním a udržováním oddílů [27].

Jako vstup používá data z AWR, ale dokáže vycházet i ze stávajícího schéma databáze. AWR obsahuje jeden nebo více SQL dotazů, plus statistiky a atributy, které přesně popisují jednotlivé dotazy. Na výstupu produkuje doporučení, díky kterým lze dosáhnout lepších exekučních plánů [27].



Obrázek 11: Architektura nástroje SQL Access Advisor [27]

SQL Access Advisor lze spouštět buď prostřednictvím uživatelského rozhraní, které je součástí placeného nástroje Oracle Enterprise Manager Cloud Control nebo pomocí package DBMS_ADVISOR [27].

7.5 Quest SQL Optimizer for Oracle

Quest SQL Optimizer je produktem společnosti Quest Software, Inc, která se v červenci 2012 stala součástí společnosti Dell, Inc. Cílem Quest SQL Optimizeru je automatizace optimalizace SQL a zvyšování výkonu SQL dotazů. Dosahuje toho tak, že analyzuje, přepisuje a vyhodnocuje SQL příkazy umístěné v databázových objektech, souborech, nebo v paměťové oblasti SGA. Ve chvíli, kdy identifikuje problematické SQL dotazy, provede jejich optimalizaci a poskytne náhradní optimalizovaný kód daného

dotazu. Dále poskytuje kompletní optimalizaci indexů, to znamená, že doporučuje místa pro přidání indexů a dokáže simulovat dopad přidání indexu na výkon dotazu.

Quest SQL Optimizer je placený nástroj s možností využití 30denní zkušební verze. Pro běh vyžaduje instalaci .NET Frameworku a databázového klienta. Nástroj se skládá z následujících modulů:

Optimize SQL

Modul sloužící k optimalizaci SQL dotazů. Před samotným zahájením procesu optimalizace je nutné vybrat jeden ze dvou pracovních módů:

- **SQL Rewrite**

  Při práci v tomto módu se předpokládá možnost zasahovat do kódu dotazu. Proces optimalizace se skládá ze dvou kroků. V prvním kroku jsou s použitím transformací a hintů vygenerovány sémanticky ekvivalentní alternativy dotazu. Kromě generování alternativ umožňuje vyhledat indexy, jejichž přidáním lze zvýšit výkon dotazů. V druhém kroku provede Quest SQL Optimizer testování každé alternativy a poskytne statistiky, díky kterým dokáže nalézt nejvhodnější alternativu dotazu pro dané prostředí.
- **Plan Control**

  Plan Control mód předpokládá, že není možné zasahovat do kódu dotazu. Proces sestává ze dvou kroků. Quest SQL Optimizer nejprve vygeneruje alternativy exekučního plánu bez změny zdrojového kódu a stejně jako u módu SQL Rewrite poskytne informace, díky kterým lze vybrat nejvhodnější alternativy pro aktuální prostředí databáze. V druhém kroku lze vybraný exekuční plán zanést do SQL Baseline, čímž k němu databáze získá přístup a může jej použít. Proto je při tomto módu vyžadováno připojení k Oracle 11g.

Optimize Indexes

Tento modul se používá pro analýzu a vyhledání indexů, které by mohly pozitivně ovlivnit exekuční plán daných dotazů. Při vyhledávání takových indexů provádí analýzu všech tabulek použitých v dotazu. Po vyhledání indexů provede jejich otestování a dopad na výkon dotazu. Při testování používá virtuální indexy, to znamená, že se fyzicky žádné indexy nevytvoří a není tak ovlivněn výkon samotné databáze. Po otestování doporučí nejvhodnější indexy a nabídne script pro jejich vytvoření. Pro analýzu indexů je nutné

vybrat jeden z následujících zdrojů: AWR, Foglight Performace Analysis Repository, SGA, Scan Code.

Batch Optimize SQL

Batch Optimize se používá pro dávkovou optimalizaci. Jako vstup přijímá textové soubory, databázové objekty, zdrojové kódy SQL příkazů, nebo i příkazy uložené v Foglight PA Repository. Batch Optimize automaticky najde všechny DML příkazy a pokusí se o jejich optimalizaci. Při procesu optimalizace dochází k přepisu dotazů s použitím transformací a hintů, poté se tyto alternativní dotazy testují s cílem nalézt nejvýkonnější SQL příkaz. Po dokončení celého procesu program nabídne nejlepší alternativy a umožňuje vygenerovat script pro nahrazení původního zdrojového kódu.

Scan SQL

Scan SQL slouží k identifikaci problematických SQL příkazů. Jak už název napovídá, dochází ke skenování a extrahování DML příkazů ze zdrojového kódu. Scan SQL načte a analyzuje exekuční plány extrahovaných příkazů a provede jejich klasifikaci podle přednastavených pravidel. Jako zdroj lze použít vybrané databázové objekty, zdrojové kódy z textového nebo binárního souboru, či data z paměťové oblasti SGA. Po dokončení procesu skenování jsou SQL příkazy rozděleny do následujících skupin:

- **Problematic**
  Dotazy odkazující na 6 a více databázových objektů a provádějící Full Table Scan nad 2 a více tabulkami. Takovéto dotazy jsou doporučeny k optimalizaci. Optimalizovat je lze prostřednictvím modulů Optimize SQL nebo Batch Optimize.
- **Complex**
  Dotazy, které nespadají do skupiny Problematic, ale obsahují odkazy na 2 a více objektů a provádí Fast Full Index Scan u 3 a více tabulek.
- **Simple**
  Dotazy, které nepatří ani do jedné z předchozích skupin a jejichž exekuční plány jsou v pořádku.
- **Invalid**
  Dotazy, u kterých se nepodařilo získat exekuční plány.

Inspect SGA

Cílem tohoto modulu je zachycení, zobrazení a analyzování dotazů provedených nebo běžících v systémové paměťové oblasti databáze (SGA). Před spuštěním procesu lze nastavit kritéria a získávat tak očekávané příkazy, například dotazy konkrétního uživatele atp. Proces procházení oblasti SGA lze spustit okamžitě nebo ho naplánovat na požadovanou dobu. Po dokončení procesu lze SQL příkazy jednoduše přenést do ostatních modulů programu. Například lze pomocí Scan SQL provést jejich klasifikaci a poté optimalizovat problematické dotazy prostřednictvím modulu Optimize SQL.

Analyze Impact

Nástroj, který analyzuje dopad změn v databázovém prostředí na výkon SQL dotazů. Modul Analyze Impact si nejprve uloží exekuční plány zadaných SQL dotazů, poté provede požadované změny v databázi a nakonec si získá nové exekuční plány, které porovná s původními. Uživateli poté nabídne porovnání výkonu jednotlivých dotazů ve formě grafů a určí procentuální zvýšení nebo snížení výkonu. Nástroj umožňuje testovat dopad změn jako je přidání indexů, změna inicializačních parametrů či hardwarové a konfigurační změny.

Manage plans

Manage plans slouží pro správu uložených SQL Baselines. Prostřednictvím tohoto modulu lze uložené Baselines jednoduše zakazovat, povolovat a mazat a tím ovlivňovat optimalizace na úrovni Plan Managementu.

8 Optimalizace v praxi

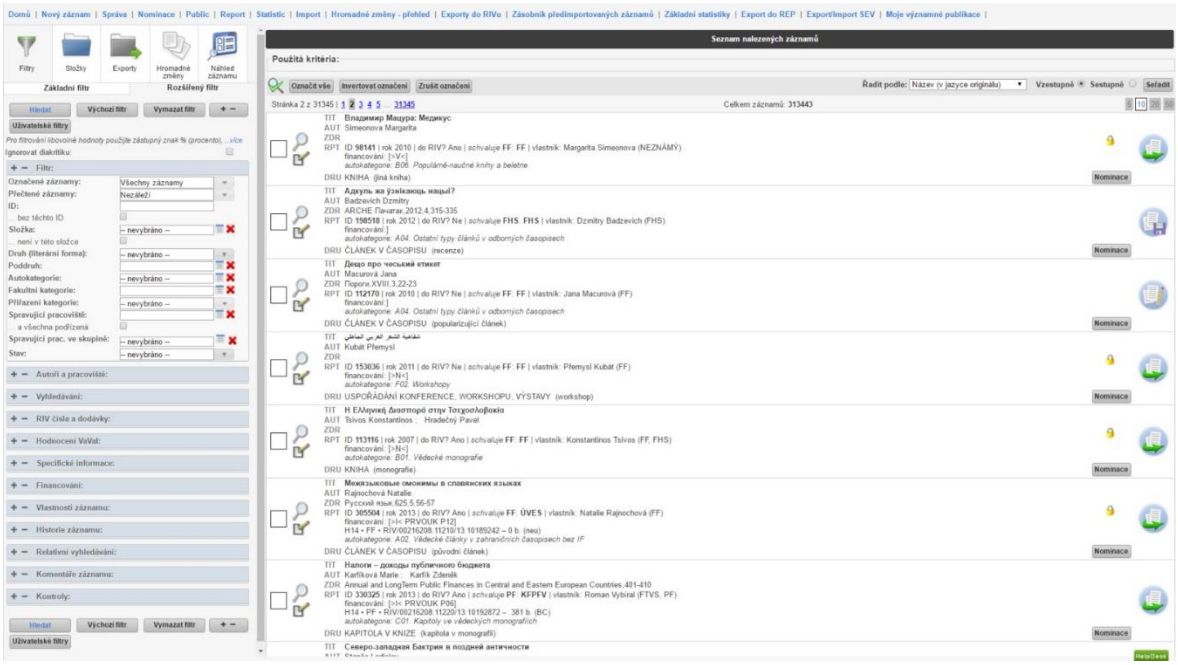
Tato kapitola je zaměřena na praktické využití znalostí a postupů uvedených v předchozích kapitolách. Je zde demonstrován postup optimalizace SQL dotazů pro již existující aplikaci OBD, která je běžně používána v produkčním prostředí. To znamená, že se optimalizace týkají samotných SQL dotazů, nikoliv úprav databázové struktury.

Cílem bylo vybrat několik SQL dotazů, ty následně optimalizovat a poté porovnat výkon původních dotazů s dotazy, jež prošly procesem optimalizace. Pro správné pochopení všech náležitostí souvisejících s procesem optimalizace je kapitola rozdělena do několika částí. Nejprve je popsána aplikace OBD, které se veškeré optimalizace týkají. Následuje popis a odůvodnění vybraných nástrojů, které byly v procesu optimalizace použity. Poté jsou popsány jednotlivé kroky procesu optimalizace od identifikace dotazů až po vyhodnocení. Poslední část se věnuje vybraným SQL dotazům a jejich optimalizacím společně s vyhodnocením výsledků.

8.1 Aplikace OBD

OBD (Osobní Bibliografická Databáze) je webová aplikace od společnosti DERS, s. r. o. Aplikace slouží ke sběru a správě záznamů o publikační a jiné vědecké činnosti a umožňuje schvalování bibliografických záznamů na základě uživatelsky definovaného workflow. Pro export záznamů systém nabízí několik formátů včetně uživatelsky definovaného, kde lze konfigurovat vlastní vzhled bibliografické citace. Výsledný soubor bibliografických záznamů lze následně prostřednictvím OBD exportovat do RIV (rejstřík informací o výsledcích).

V současné době aplikace OBD využívají 4 vědecko-výzkumné instituce a 15 univerzit, mezi které patří například i Univerzita Hradec Králové či Univerzita Karlova.



Obrázek 12: Aplikace OBD (zdroj: autor)

8.2 Výběr nástrojů pro optimalizaci

Nástroje pro optimalizaci SQL se používají velmi často, především pak u obrovských databází obsahující stovky nebo i tisíce databázových objektů. Bez používání takovýchto nástrojů je potřeba naprostý přehled o všech databázových objektech, kterých se optimalizace týká, což je časově velmi náročné. Proto byl i v této práci použit jeden z mnoha nástrojů, které se optimalizací zabývají.

Pro tuto práci bylo vyzkoušeno několik programů, z nichž se do nejužšího výběru dostaly nástroje DB Optimizer XE od společnosti Embarcadero a Quest SQL Optimizer for Oracle. Po otestování obou nástrojů byl pro další práci zvolen Quest SQL Optimizer ve verzi 8.8.1. Tento nástroj byl upřednostněn z několika důvodů. Dokáže zpracovávat SQL dotazy obsahující klauzuli WITH, v čemž DB Optimizer XE zaostává, protože takové příkazy zpracovat neumí. Další výhodou Quest SQL Optimizeru je možnost exportu výsledků do několika typů souborů, což při použití DB Optimizer XE také není možné. Jednu z mála výhod, kterou software od společnost Embarcadero nabízí je paralelní testování. Quest SQL Optimizer tuto možnost v základní instalaci nenabízí, je ale možné doinstalovat nástroj Benchmark Factory, který přístup více uživatelů simulovat dokáže a pro tuto práci byl také použit. Benchmark Factory je program umožňující vytváření virtuálních uživatelů, kteří paralelně spouští potřebné SQL dotazy.

8.3 Postup při optimalizaci

Postup, který byl při optimalizacích zvolen, lze rozdělit do několika kroků. Nejprve bylo nutné identifikovat SQL dotazy, pro které je optimalizace požadována. Tento jediný krok bylo nutné udělat právě jednou, další kroky optimalizace se již vztahují ke konkrétním dotazům, to znamená, že bylo potřeba jimi projít pro každý SQL dotaz zvlášť. Jedná se o proces manuální úpravy dotazu v případě, kdy je možné dotaz jednoduše upravit tak, že se nahradí například operátory LIKE a IN. Dalším krokem bylo generování alternativ dotazů a výběr několika z nich, u kterých byly provedeny zátěžové testy v podobě paralelního přístupu více uživatelů. V posledním kroku pak bylo nutné vyhodnotit výsledky optimalizace a rozhodnout o jejím prospěchu a popřípadě implementaci. Následuje popis jednotlivých kroků optimalizace.

Identifikace SQL dotazů

Prvním krokem v procesu optimalizace bylo identifikování SQL dotazů, které je potřeba optimalizovat. Většinou se jedná o dotazy, jejichž provedení a navrácení výsledků trvá déle, než je očekáváno nebo v případě nadměrného využívání systémových prostředků těmito dotazy. Způsobů, jak takové dotazy nalézt je více. Lze použít některý z nástrojů, který je k takovému účelu vytvořen nebo je lze identifikovat prostřednictvím některého z databázových objektů, který uchovává informace o provedených příkazech. Pro určení problémových dotazů lze použít například následující dotaz využívající tabulku V$SQLAREA, která poskytuje statistické údaje o SQL příkazech uložených v paměti:

```
SELECT  sql_text,
        ROUND(elapsed_time /EXECUTIONS/1E6,3),
        EXECUTIONS,
        FETCHES,
        BUFFER_GETS,
        ROWS_PROCESSED,
        plsql_exec_time,
        optimizer_mode,
        optimizer_cost
FROM V$SQLAREA
WHERE PARSING_SCHEMA_NAME = :SCHEMA
AND executions > 10
ORDER BY elapsed_time DESC;
```

Kód 20: Příklad dotazu pro identifikaci problémových SQL

Dotaz přijímá jeden parametr v podobě bindované proměnné pro určení databázového schéma a vrací data zahrnující například zaokrouhlenou hodnotu *elapsed_time*, která udává počet sekund potřebných na provedení dotazu, dále pak hodnotu *executions* znázorňující počet spuštění dotazu či hodnotu *optimizer_cost* informující o ceně

exekučního plánu. Při správném seřazení a dalším vyfiltrování dat tak lze snadno nalézt dotazy, které by bylo vhodné optimalizovat.

Pro tuto práci byl proveden výběr dotazů zaměstnancem společnosti DERS, s.r.o. Jedná se o tyto tři SQL dotazy:

- Dotaz použitý pro výpis významných publikací uživatele
- Dotaz pro výpis složek včetně řešení práv na akce
- Dotaz použitý v doplňkovém modulu OBD Public

Manuální úprava dotazů

Dalším krokem po identifikaci SQL dotazů určených pro optimalizaci byl pokus o jejich manuální přepsání. Tuto část nebylo možné provést vždy, úpravou prošly pouze dotazy, u kterých bylo na první pohled vidět, že je možné je přepsat efektivněji nebo v případě, že lze použít hint, který by optimalizátoru pomohl zpracovat dotaz rychleji.

Uplatňována byla obecná pravidla pro psaní efektivních SQL dotazů. Například při výběru všech sloupců z tabulky by se neměl používat znak *, efektivnější je všechny sloupce vyjmenovat. Databáze pak nemusí zjišťovat seznam všech sloupců. Dále jsou popsány operátory, které by neměly být používány, pokud to není opravdu nezbytné. Většina programů pro optimalizaci včetně Quest SQL Optimizeru totiž takto dotaz neupraví, i když by tím v mnoha případech pozitivně ovlivnily exekuční plán.

- **LIKE, NOT LIKE**

  Operátor LIKE se nedoporučuje u vyhledávání ve velkých textových polích a vždy je lepší zamyslet se, zda pro vyhledávání nelze použít jiná metoda.

- **IN, NOT IN**

  V mnoha případech se používá operátor IN pouze z důvodu zjištění, zda nějaká hodnota existuje. V takovém případě je mnohem efektivnější nahradit operátor IN operátorem EXISTS.

- **COUNT**

  Stejně jako operátor IN se i operátor COUNT používá méně znalými uživateli pro zjištění existence některé hodnoty. Databáze je pak nucena k počítání řádků,

přičemž to vůbec není potřeba. U takových případů by se měl vždy používat EXISTS.

- **UNION**

  UNION se používá pro spojení více dotazů. V mnoha případech se používá UNION i ve chvíli, kdy by bylo možné použít UNION ALL, který nad vrácenými řádky neprovádí operaci DISTINCT. Tato operace odstraňuje duplicitní řádky z výsledku a pro databázový systém je velmi nákladnou operací.

Generování alternativ dotazů

Pro generování alternativ vybraných SQL dotazů byl v této práci použit SQL Quest Optimizer a protože bylo možné zasahovat do zdrojového kódu dotazů, tak byl použit mód SQL Rewrite. Při použití tohoto módu program nabízí různé možnosti vyhledávání a testování alternativ dotazu. Lze využít například možnost Auto Optimize, u níž při generování alternativ zároveň probíhá jejich spuštění. Každý dotaz je spouštěn dvakrát. Další možnost, kterou Quest SQL Optimizer nabízí je volba Rewrite. Tato volba provede pouze generování alternativ bez jejich spuštění. Další volbou je Index, při které se program pokusí nalézt indexy, které by mohly zvýšit výkon dotazů. To znamená, že software podle exekučního plánu zjistí místa, kde se provádí například Full Table Scan a navrhne index, díky kterému by bylo možné provést jinou metodu. Poslední možností je volba Rewrite and Index, která byla v této práci pro generování použita. Volba kombinuje předchozí dvě možnosti, tedy vyhledávání indexů a generování alternativ dotazů.

Jelikož volba Rewrite and Index dotazy v průběhu generování neprovádí, bylo nutné všechny vygenerované dotazy spustit. Protože bylo cílem spustit každý dotaz právě desetkrát, byla použita volba Batch run Multiple All, ve které je možné nastavit počet spuštění každého SQL dotazu. Čím vyšší je zvolen počet spuštění dotazů, tím jsou výsledky přesnější, naměřené hodnoty jsou totiž průměry z více měření a nejsou v takové míře ovlivněny náhodným zatížením databázového systému.

V dotazech byly použity typické parametry. Jedná se o hodnoty, pro které dotazy vrací přibližně stejný počet řádků, který se očekává při použití v produkčním prostředí.

Výběr dotazů pro zátěžové testy

V tomto kroku procesu optimalizace bylo cílem vybrat několik dotazů z vygenerovaných alternativ, které dále projdou zátěžovými testy. Výběr byl nutný z toho

důvodu, že Quest SQL Optimizer generuje až několik set alternativ, přičemž má spoustu z nich velmi podobné exekuční statistiky a bylo by časově příliš náročné testovat je všechny.

Výběr vhodných alternativ pro další testování probíhal tak, že vygenerované alternativní dotazy byly setříděny podle hodnoty Elapsed Time, tedy doby provedení dotazu. Při výběru byly brány v úvahu následující atributy:

- Elapsed Time – Celkový čas potřebný k provedení dotazu (měří se čas, kdy databázový systém začne zpracovávat dotaz až po jeho dokončení a navrácení dat)
- First Row Time – Celkový čas potřebný k navrácení první řádku dotazu
- CPU Used by this Session – Celkové využití procesorového času
- Session Logical Reads – Celkový počet načtených databázových bloků z disku a paměti
- Sorts (Rows) – Celkový počet řádků, nad kterými databáze provádí třídění

Za pomoci předchozích parametrů bylo vždy vybráno několik alternativ, které byly dále zátěžově testovány v podobě paralelního přístupu několika desítek uživatelů.

Zátěžové testování

Zátěžové testování bylo do vyhodnocování výkonnosti dotazů začleněno z důvodu, že Quest SQL Optimizer, pomocí kterého probíhalo generování alternativ, spouští dotazy jako jeden uživatel. Některé alternativní dotazy, které jsou Quest SQL Optimizerem vyhodnoceny jako výkonnější v porovnání s původním dotazem však mohou být v některých případech pro databázi mnohem náročnější v momentě, kdy jej spouští více uživatelů paralelně. Z tohoto důvodu bylo potřeba provést zátěžové testy, kterými lze zjistit informace o časech odezvy v závislosti na počtu uživatelů spouštějících daný SQL dotaz.

Zátěžové testy probíhaly formou simulace paralelního spouštění dotazů prostřednictvím nástroje Benchmark Factory od společnosti Quest. Tento nástroj byl vybrán z důvodu kompatibility s programem Quest SQL Optimizer. Při testování bylo hlavním kritériem výkonu Response Time (čas, za který je navrácen výsledek). Před spuštěním samotného testu bylo možné nastavit minimální a maximálně počet virtuálních uživatelů, kteří budou paralelně spouštět dotazy. Tento počet byl pro každý dotaz určen zvlášť v závislosti na druhu používání dotazu. To znamená, že pro dotazy, které jsou

použité na místech, kam mají přístup běžní uživatelé, byl nastaven vyšší počet virtuálních uživatelů než pro dotazy, které může spouštět jen několik vybraných uživatelů. Nástroj dále umožňuje nastavit počet spuštění dotazu každým uživatelem, přičemž výsledná hodnota času odezvy je průměrem jednotlivých měření. Pro tuto práci byl počet spuštění každým uživatelem určen na 10.

Shrnutí výsledků a vyhodnocení optimalizace

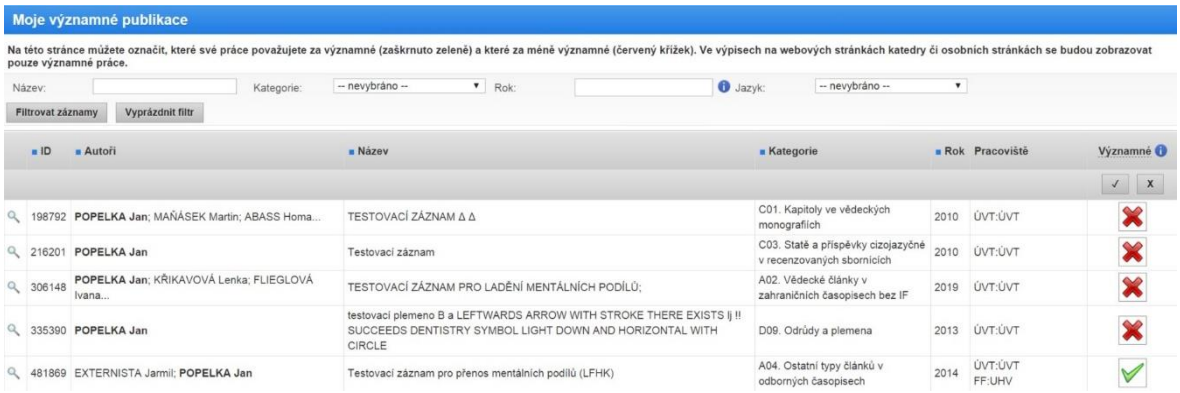
Posledním krokem v procesu optimalizace bylo vyhodnocení optimalizace na základě naměřených hodnot a provedených testů. Rozhodujícím faktorem při vyhodnocování výsledků bylo kritérium Elapsed Time. Naopak kritérium Plan Cost (cena dotazu) vůbec nebylo bráno v úvahu, protože se při testování dotazů ukázalo, že tento atribut nemá vliv na rychlost dotazu ani na využívání procesorového času a dalších systémových zdrojů. Při vyhodnocování byly kromě Elapsed Time brány v úvahu ještě kritéria CPU Used by this Session, Session Logical Reads a Sorts (Rows).

8.4 Optimalizace vybraných dotazů

Následující kapitoly představují proces optimalizace. Protože cílem této části je optimalizovat 3 vybrané SQL dotazy z aplikace OBD, obsahuje tato část právě 3 kapitoly. Každá kapitola se zabývá optimalizací jednoho SQL dotazu. V jednotlivých kapitolách je nejprve dotaz charakterizován, aby si čtenář dokázal představit, jakým způsobem je v aplikaci OBD používán. Následně jsou podrobně popsány jednotlivé kroky optimalizace.

8.4.1 Dotaz 1 – Významné publikace uživatele

Dotaz se používá na stránce, která slouží pro správu významných publikací autora. Uživatel má prostřednictvím této stránky možnost označovat jaké publikace jsou pro něho významné a které nikoliv. S tímto příznakem pak pracují další dotazy na různých místech aplikace OBD. Například ve výpisech na webových stránkách katedry a osobních stránkách se vypisují pouze publikace, které jsou autorem označeny jako významné. Na následujícím obrázku lze pro představu vidět stránku, na které je dotaz použit.



Obrázek 13: OBD – Významné publikace (zdroj: autor)

Výsledkem dotazu jsou všechny publikace, ve kterých je přihlášený uživatel uveden jako interní autor. SQL dotaz vrací následující informace:

- ID – Identifikátor publikace
- NAZEV – Název publikace v originálním jazyce
- ROK – Rok publikace
- AUTORI – První tři autoři publikace, pokud je přihlášený uživatel v pořadí autorů uveden na čtvrtém místě, vypíše se také. Pokud je uveden na páté a vyšší pozici, vypíše se za prvními třemi autory tři tečky a poté následuje výpis přihlášeného autora včetně čísla pořadí, ve kterém je v publikaci uveden.
- PRVNI_AUTOR – Interní autor, který je v publikaci uveden na první pozici
- OZNACENO – Příznak pro označení méně významných publikací
- ZOBRAZIT – Příznak pro možnost označení publikace jako méně významné
- PRACOVISTE – Pracoviště autora publikace
- KATEGORIE – Autokategorie publikace

8.4.1.1 Optimalizace a generování alternativ

Dotaz neporušoval žádnou ze zásad pro psaní optimalizovaných dotazů, a proto nebyla nutná manuální úprava dotazu. Dalším krokem bylo vyhledání indexů, které by mohly urychlit přístup k datům. V tomto případě se nepodařilo žádný takový index nalézt a další optimalizace se tedy týkaly samotného SQL dotazu. Poté následoval proces generování alternativ dotazu. Pro generování byl použit Quest SQL Optimizer,

prostřednictvím kterého bylo vygenerováno 255 alternativ, z nichž 70 dosahovalo lepších výsledků než původní dotaz. Výsledná data těchto 70 alternativ společně s výsledky původního dotazu lze nalézt v příloze č. 2.

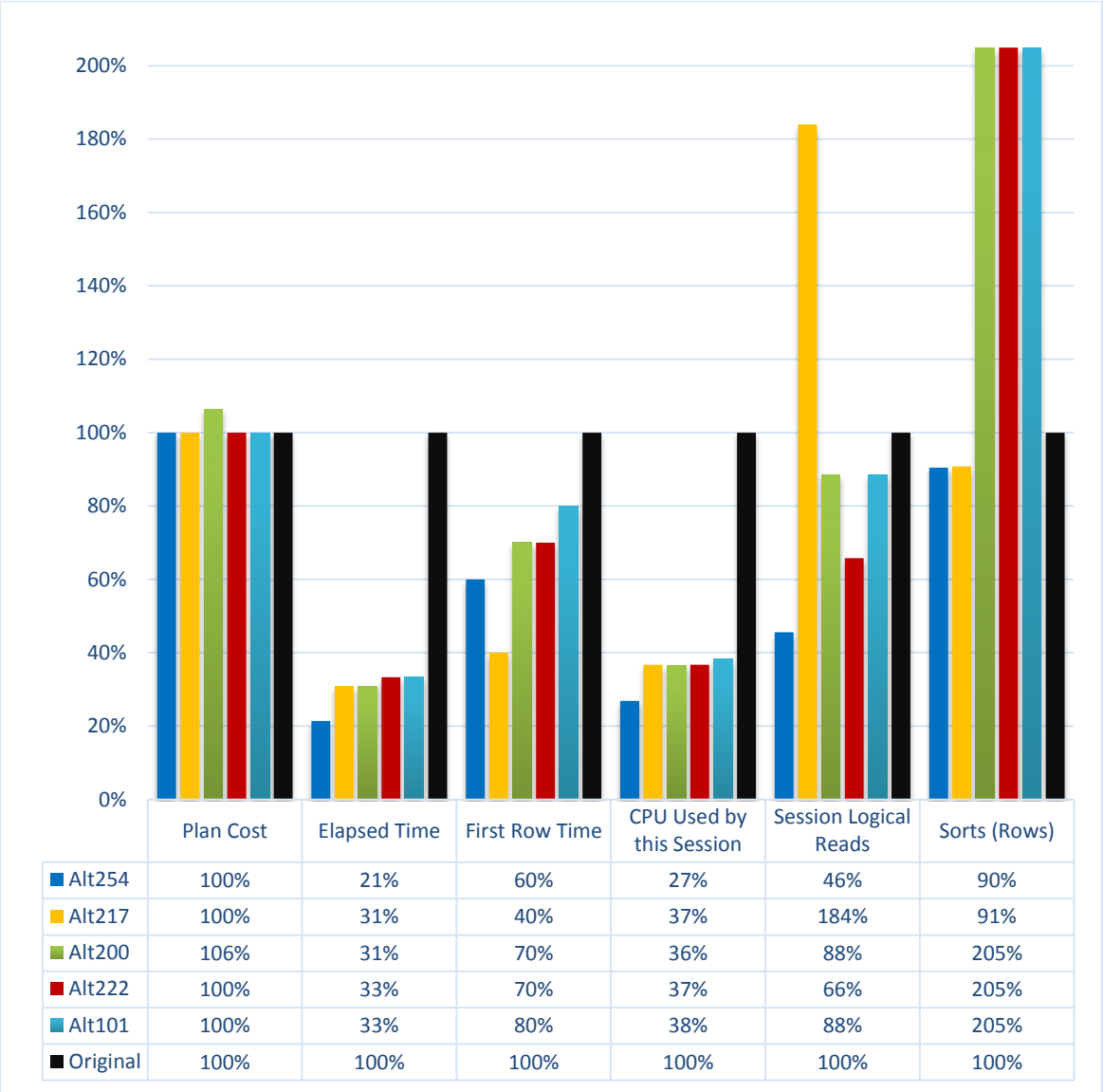
Z alternativ, které dosahovaly lepších výsledků, než původní dotaz bylo dále vybráno 5 nejvýkonnějších dotazů pro další testování. Těchto 5 alternativ je žlutě zvýrazněno v příloze č. 2. Zdrojový kód vybraných SQL dotazů lze nalézt v příloze č. 1. Následující tabulka shrnuje výsledky nejpodstatnějších atributů 5 vybraných dotazů a původního dotazu.

Scenario Name	Plan Cost	Elapsed Time (s)	First Row Time (s)	CPU Used by this Session	Session Logical Reads	Sorts (Rows)
Alt254	31	0,09	0,06	0,136	3 902	2 561
Alt217	31	0,13	0,04	0,186	15 764	2 571
Alt200	33	0,13	0,07	0,184	7 575	5 801
Alt222	31	0,14	0,07	0,186	5 633	5 803
Alt101	31	0,14	0,08	0,194	7 575	5 801
Original	31	0,42	0,10	0,506	8 568	2 832

Tabulka 4: Dotaz 1 – Naměřené hodnoty vybraných alternativ (zdroj: autor)

V předchozí tabulce lze vidět jak velký rozdíl je mezi výsledky alternativních dotazů v porovnání s dotazem původním.

Přehlednější porovnání naměřených hodnot zobrazuje následující graf, kde je zvýšení či snížení výkonu vyjádřeno v procentech.

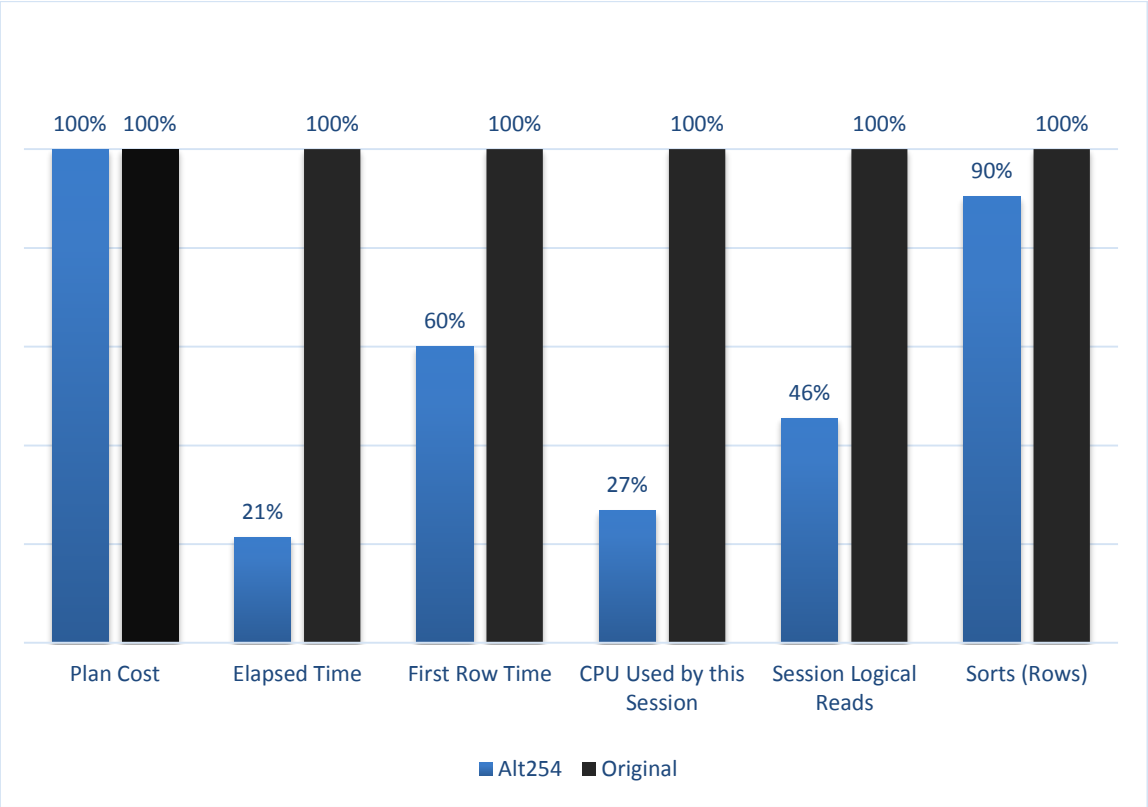


	Plan Cost	Elapsed Time	First Row Time	CPU Used by this Session	Session Logical Reads	Sorts (Rows)
Alt254	100%	21%	60%	27%	46%	90%
Alt217	100%	31%	40%	37%	184%	91%
Alt200	106%	31%	70%	36%	88%	205%
Alt222	100%	33%	70%	37%	66%	205%
Alt101	100%	33%	80%	38%	88%	205%
Original	100%	100%	100%	100%	100%	100%

Graf 1: Dotaz 1 – Procentuální zlepšení dotazu pro vybrané alternativy (zdroj: autor)

Z grafu 1 a naměřených hodnot při testování alternativ lze prozatím zhodnotit dotaz Alt254 jako nejefektivnější. Alternativa Alt217 sice dosahuje lepšího výsledku u navrácení prvního řádku, nicméně v ostatních měřených hodnotách zaostává za dotazem Alt254.

Následující graf porovnává rozdíly mezi původním dotazem a dotazem Alt254 a udává procentuální zlepšení dotazu.

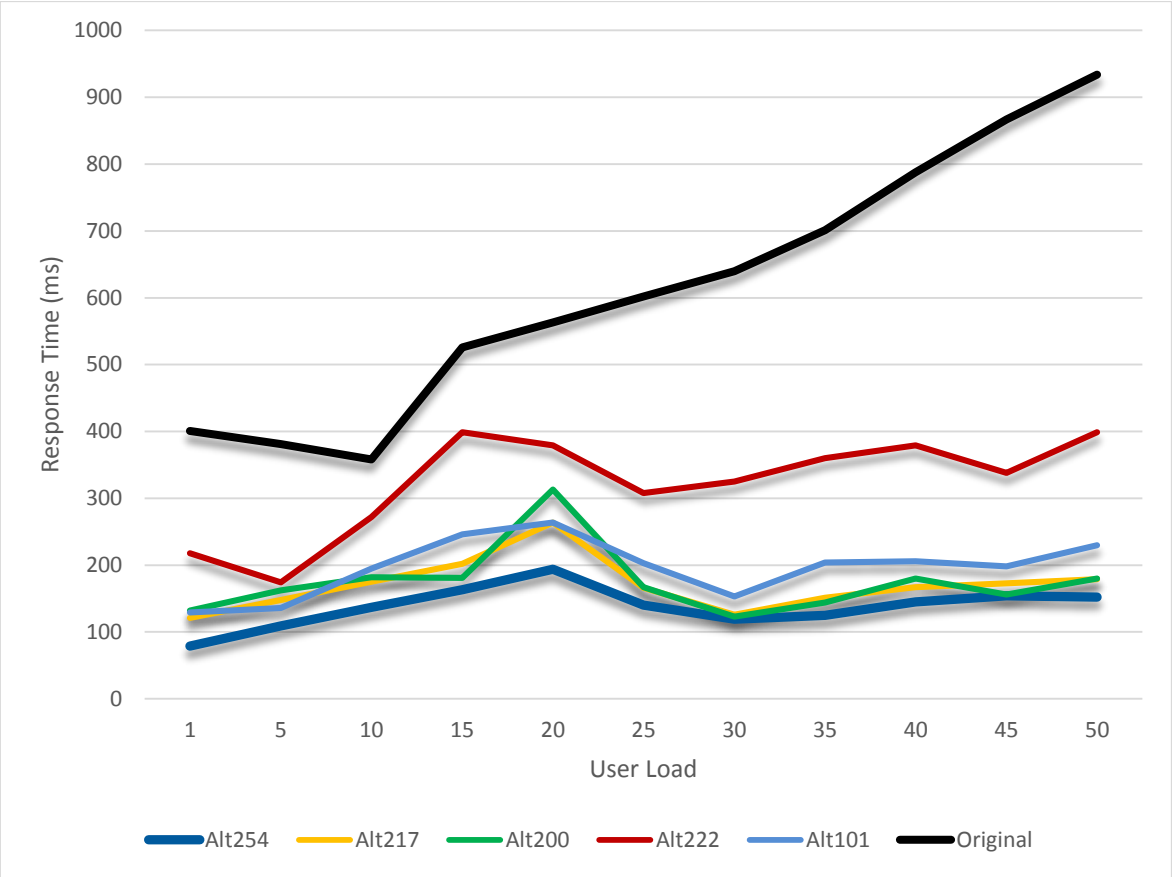


Graf 2: Dotaz 1 - Porovnání optimalizovaného a původního dotazu (zdroj: autor)

U dotazu Alt254 došlo k významnému snížení doby trvání celého dotazu z 0,42 sekund na 0,09 sekund. Výsledkem optimalizace je zrychlení dotazu o více než 79 %, přičemž využívá o 73 % méně procesorového času než původní dotaz. Jak je patrné z grafu i u ostatních měřených hodnot došlo k výraznému zlepšení.

8.4.1.2 Zátěžové testování

Paralelní test probíhal na původním dotazu a 5 vybraných alternativách. Protože je stránka s dotazem přístupná většině běžných uživatelů systému OBD, bylo cílem otestovat paralelní zátěž až pro 50 uživatelů. Výsledné hodnoty naměřené při zátěžovém testu lze vidět v následujícím grafu, který vyjadřuje závislost času odpovědi na počtu paralelně přistupujících uživatelů. Konkrétní hodnoty, ze kterých graf vychází lze nalézt v příloze č. 2.



Graf 3: Dotaz 1 - Zátěžový test (zdroj: autor)

Z přechozího grafu lze tvrdit, že nejstabilnějších výsledků při testování zátěže dosáhl dotaz Alt254. Tato alternativa měla při zatížení od 1 až po 50 paralelně přistupujících uživatelů nejmenší výkyvy co se času odezvy týče. V průběhu testu se až na drobné výjimky čas odezvy pohyboval pod 0,2 sekund.

8.4.1.3 Shrnutí výsledků

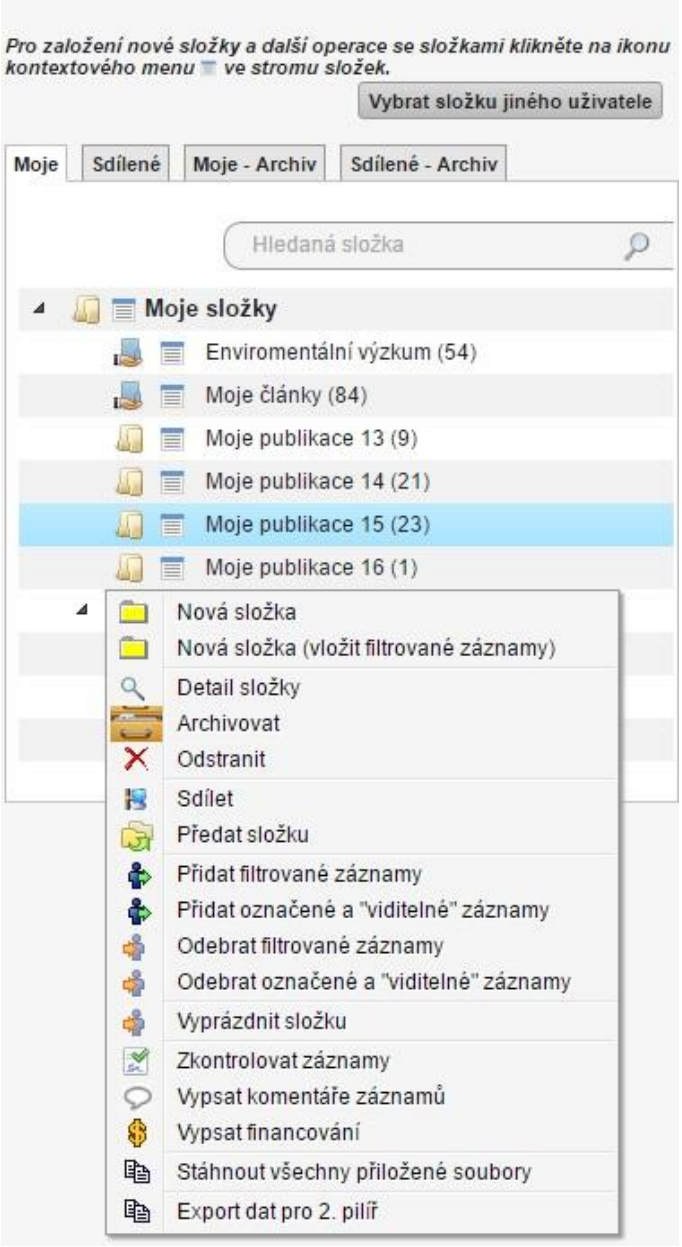
Nejefektivnějším řešením byl v tomto případě bezesporu dotaz Alt254, který u převážné většiny zkoumaných kritérií dosáhl nejoptimálnějších výsledků. Tento dotaz byl téměř 5krát rychlejší než původní dotaz a zároveň nevytěžoval systémové zdroje v takové míře jako původní dotaz. Například počet Session Logical Reads klesl o 54 % a také využíval o 73 % méně procesorového času.

Alternativa Alt254 měla zároveň nejstabilnější výsledky při testování zátěže dotazu, kde se testoval paralelní přístup až 50 uživatelů, přičemž se v převážné většině naměřených časů odpovědi držely pod 0,2 sekund. V porovnání s výsledky paralelní zátěže u původního dotazu, kde se s přibývajícím počtem paralelních přístupů výrazně zvyšoval

čas odpovědi a pro 50 zároveň přistupujících uživatelů trvala odpověď více než 0,9 sekund, se jedná o významnou optimalizaci.

8.4.2 Dotaz 2 – Složky

Tento dotaz je v OBD používaný pro výpis složek uživatele. Seznam těchto složek se zobrazuje na základě vlastnických práv, které jsou řešeny v rámci stejného dotazu. Každý uživatel vidí vlastní složky (uživatel je sám vytvořil) a dále se podle oprávnění zobrazují sdílené složky či složky jiných uživatelů. Zobrazení složek v systému OBD lze pro představu vidět na následujícím obrázku.



Obrázek 14: OBD - Složky (zdroj: autor)

Složky v aplikaci OBD slouží pro třídění záznamů do logických celků dle potřeby každého uživatele. Přiřazování záznamů do složek je pouze logická operace, to znamená, že se nejedná o skutečný přesun záznamů jako takových. Z tohoto důvodu může být konkrétní záznam umístěn v několika složkách. Protože jsou vázané na uživatele, vidí každý uživatel pouze své vlastní složky nebo ty, které jsou jinými uživateli sdílené. Složky jsou rozděleny do čtyř základních záložek (viz obrázek 14):

- **Moje**

  Tato sekce je nastavena jako výchozí a každému uživateli zobrazuje seznam jeho vlastních složek.
- **Sdílené**

  V případě, že ostatní uživatelé sdílejí své vlastní složky s přihlášeným uživatelem, jsou vypsány právě na této záložce.
- **Moje – Archiv**

  Záložka obsahuje složky přihlášeného uživatele, které jsou označeny jako archivované. Složky lze do archivu přesunout pomocí speciální akce přístupné v kontextového menu vybrané složky.
- **Sdílené – Archiv**

  Záložka pro výpis sdílených složek, které jejich vlastník přesunul do archivu.

Všechny tyto sekce jsou řešeny prostřednictvím parametru v dotazu. Výsledkem dotazu jsou všechny složky přihlášeného uživatele pro aktivní sekci. Protože OBD umožňuje vytváření podsložek, které jsou načítány samostatně při rozkliknutí nadřízené složky, je v dotazu předáván ještě parametr pro určení rodičovské složky. Dotaz tedy přijímá celkem 3 parametry pro určení uživatele, záložky a nadřízené složky a vrací následující informace:

- ID – Identifikátor složky
- IDRODIC – Identifikátor nadřízené složky (v případě, že žádná nadřízená složka neexistuje, vrací hodnotu 0)
- INS_KDY – Datum vytvoření složky
- INS_KDO – Uživatelské jméno uživatele, který složku vytvořil
- UPD_KDO – Datum poslední úpravy složky
- NAZEV – Název složky

- POPIS – Popis složky
- SDILENA – Příznak určující sdílené složky
- VLASTNIK – Uživatelské jméno aktuálního vlastníka složky
- PVIEW – Příznak (pro sdílené složky), který určuje právo na zobrazení složky
- PADD – Příznak (pro sdílené složky), který určuje právo pro přidávání záznamů do složky
- PDELS – Příznak (pro sdílené složky), který určuje právo pro mazání záznamů, které do složky sám přiřadil
- PDELP – Příznak (pro sdílené složky), který určuje právo pro mazání záznamů, které byly přidány uživatelem se stejným pracovištěm jako má přihlášený uživatel
- PDELV – Příznak (pro sdílené složky), které určuje právo pro mazání všech záznamů ze složky
- TYP_SDILENI – V případě sdílené složky určuje typ sdílení (sdílet složku lze pro uživatele, roli, pracoviště)
- NAZEV_TOOLTIP – Název, který se zobrazuje jako nápověda u složky
- POCET_VE_SLOZCE – Počet záznamů přiřazených do složky
- MA_DETI – Příznak určující, zda složka obsahuje podsložky

8.4.2.1 Optimalizace a generování alternativ

Přestože dotaz splňoval všechny zásady pro psaní optimalizovaných dotazů, byl dotaz doplněn hintem MATERIALIZE. Dotaz obsahoval klauzuli WITH, ve které byl obsažen dotaz pro počítání publikací ve složce, dotaz tedy používal funkci COUNT. Poté při dotázání se na tuto hodnotu v hlavním SELECT dotazu musela databáze vždy do dané tabulky přistoupit a provést výpočet. V případě použití hintu MATERIALIZE je optimalizátor instruován k vytvoření globální dočasné tabulky, do které insertuje agregovaná data z dotazu v klauzuli WITH. Po této úpravě už databáze není nucena provádět výpočet znovu a znovu, ale pouze přistoupí do dočasné tabulky, která již vypočtené hodnoty obsahuje.

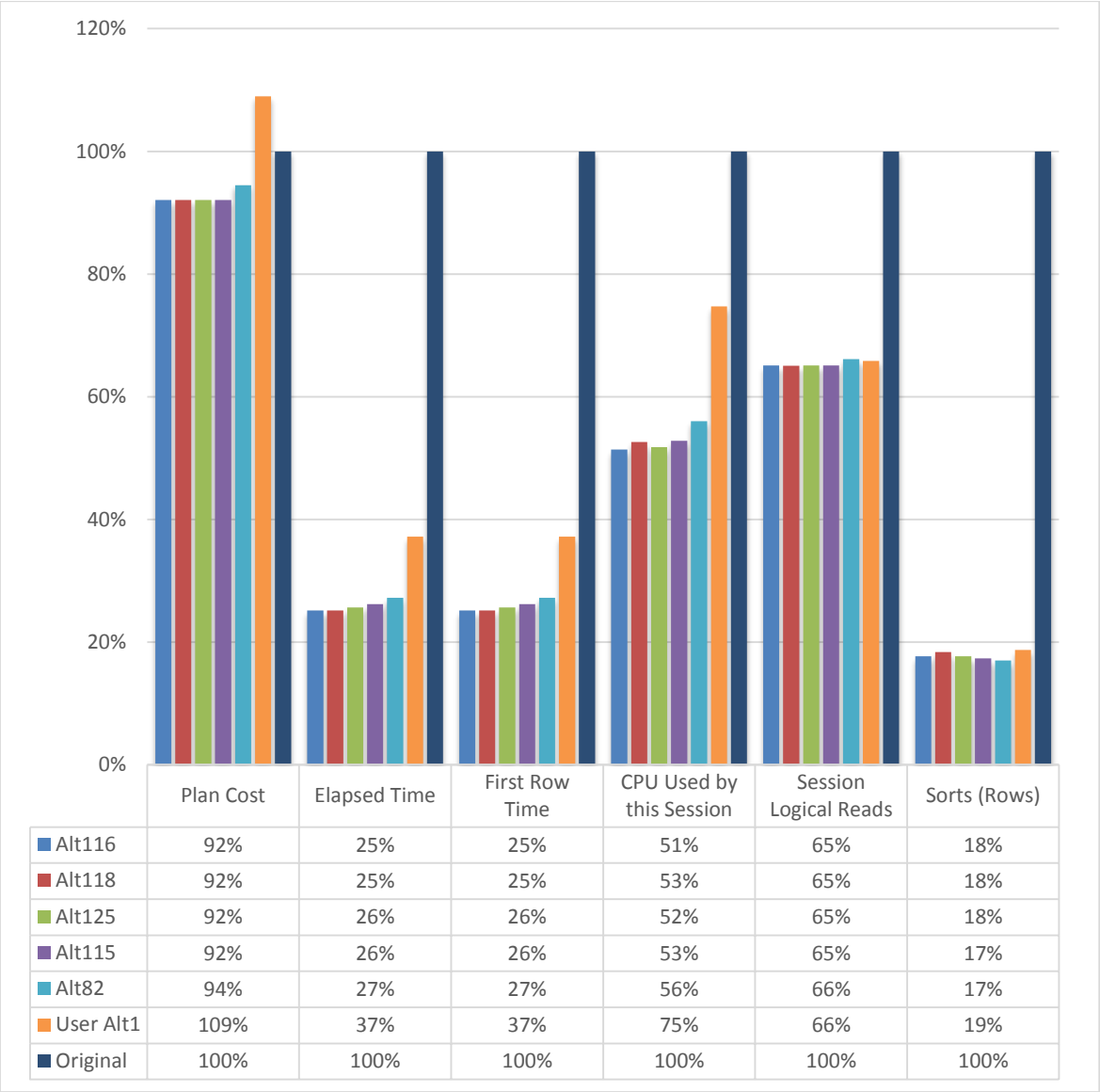
Po manuální úpravě dotazu proběhlo vyhledání indexů, díky kterým by mohl být přístup k datům rychlejší. Nicméně se pro tento dotaz nepodařilo žádný takový index nalézt. Následně probíhalo generování alternativ dotazu. Do těchto alternativ byl zahrnut i manuálně upravený dotaz s přidaným hintem. Včetně upraveného dotazu bylo

vygenerováno celkem 135 alternativ, ze kterých bylo 63 výkonnějších než původní dotaz. Naměřená data těchto 63 dotazů lze nalézt v příloze č. 4.

Podle naměřených hodnot bylo dále vybráno 5 alternativ s nejlepšími výsledky a pro porovnání byl přidán i manuálně upravený dotaz. Těchto 6 vybraných alternativ společně s původním dotazem (žlutě zvýrazněné v příloze č. 4) prošlo zátěžovým testem. Zdrojový kód vybraných dotazů lze nalézt v příloze č. 3. Následující tabulka shrnuje výsledné naměřené hodnoty vybraných dotazů a pro objektivnější pohled na data jsou stejné informace zobrazeny v grafu 4, kde lze lépe pozorovat rozdíly mezi alternativami a původním dotazem.

Scenario Name	Plan Cost	Elapsed Time (s)	First Row Time (s)	CPU Used by this Session	Session Logical Reads	Sorts (Rows)
Alt116	349	0,48	0,48	0,516	1 703	102
Alt118	349	0,48	0,48	0,528	1 702	106
Alt125	349	0,49	0,49	0,520	1 703	102
Alt115	349	0,50	0,50	0,530	1 703	100
Alt82	358	0,52	0,52	0,562	1 730	98
User Alt1	413	0,71	0,71	0,750	1 722	108
Original	379	1,91	1,91	1,004	2 616	577

Tabulka 5: Dotaz 2 – Naměřené hodnoty vybraných alternativ (zdroj: autor)



	Plan Cost	Elapsed Time	First Row Time	CPU Used by this Session	Session Logical Reads	Sorts (Rows)
Alt116	92%	25%	25%	51%	65%	18%
Alt118	92%	25%	25%	53%	65%	18%
Alt125	92%	26%	26%	52%	65%	18%
Alt115	92%	26%	26%	53%	65%	17%
Alt82	94%	27%	27%	56%	66%	17%
User Alt1	109%	37%	37%	75%	66%	19%
Original	100%	100%	100%	100%	100%	100%

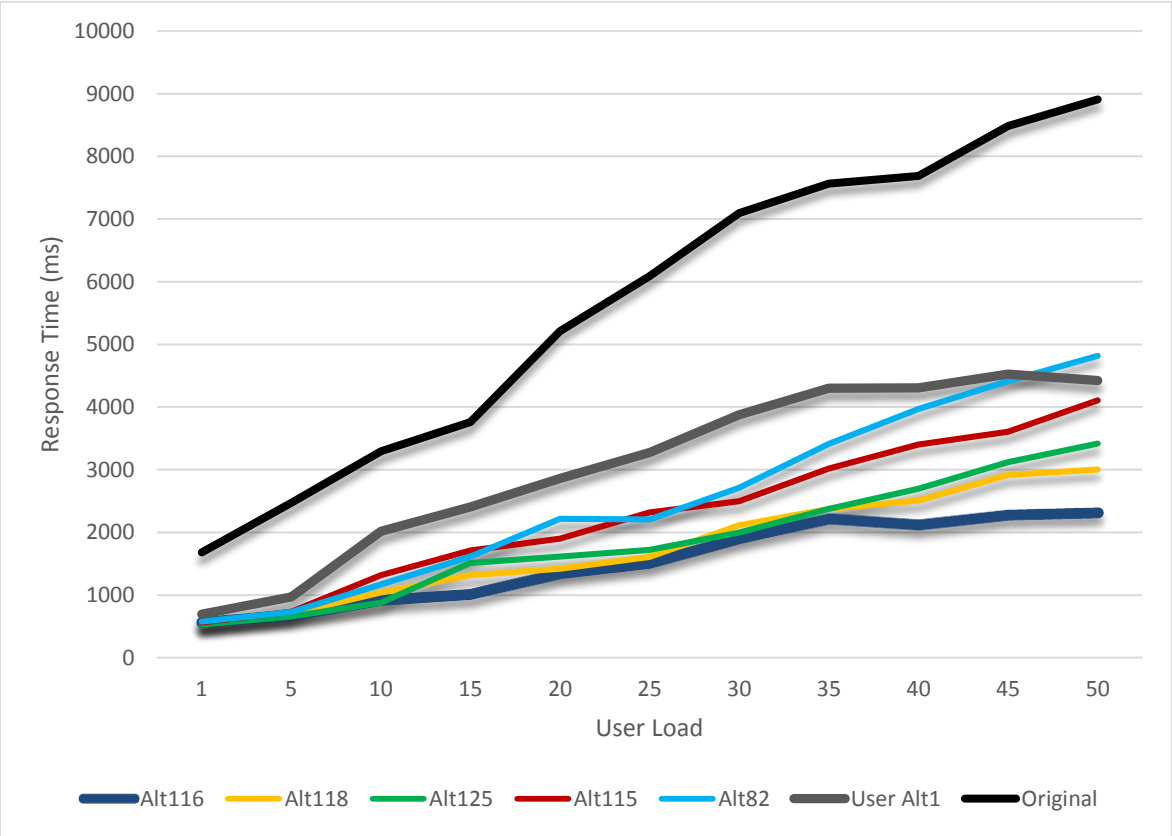
Graf 4: Dotaz 2 – Procentuální zlepšení dotazu pro vybrané alternativy (zdroj: autor)

Jak lze vidět v předchozím grafu a tabulce s naměřenými hodnotami, alternativní dotazy dosahovaly velmi podobných výsledků, přičemž jsou výrazně výkonnější než původní dotaz. Dotaz s manuálně přidaným hintem byl také několikanásobně rychlejší, ale nedosahoval takového výsledku, který by mohl být oproti vygenerovaným alternativám zanedbatelný. Z naměřených hodnot nebylo možné určit jeden nejvýkonnější dotaz z důvodu srovnatelných výsledků u vygenerovaných alternativ.

8.4.2.2 Zátěžové testování

Zátěžový test probíhal celkem na sedmi dotazech. Do těchto dotazů byl zařazen původní dotaz, manuálně upravený dotaz s přidaným hintem a 5 vybraných alternativ. Protože jsou složky v OBD přístupné všem běžným uživatelům aplikace, bylo cílem

otestovat zátěž až pro 50 uživatelů. Výkon jednotlivých dotazů lze vidět v následujícím grafu, který udává čas odpovědí dotazů v závislosti na počtu paralelně přistupujících virtuálních uživatelů. Konkrétní hodnoty, ze kterých tento graf vychází lze nalézt v příloze č. 4.



Graf 5: Dotaz 2 - Zátěžový test (zdroj: autor)

Z hlediska zátěžového testu dosáhl nejlepších výsledků dotaz Alt116. Tato alternativa měla nejstabilnější výsledky, které se ve většině případů držely těsně pod hodnotami ostatních vygenerovaných alternativ.

8.4.2.3 Shrnutí výsledků

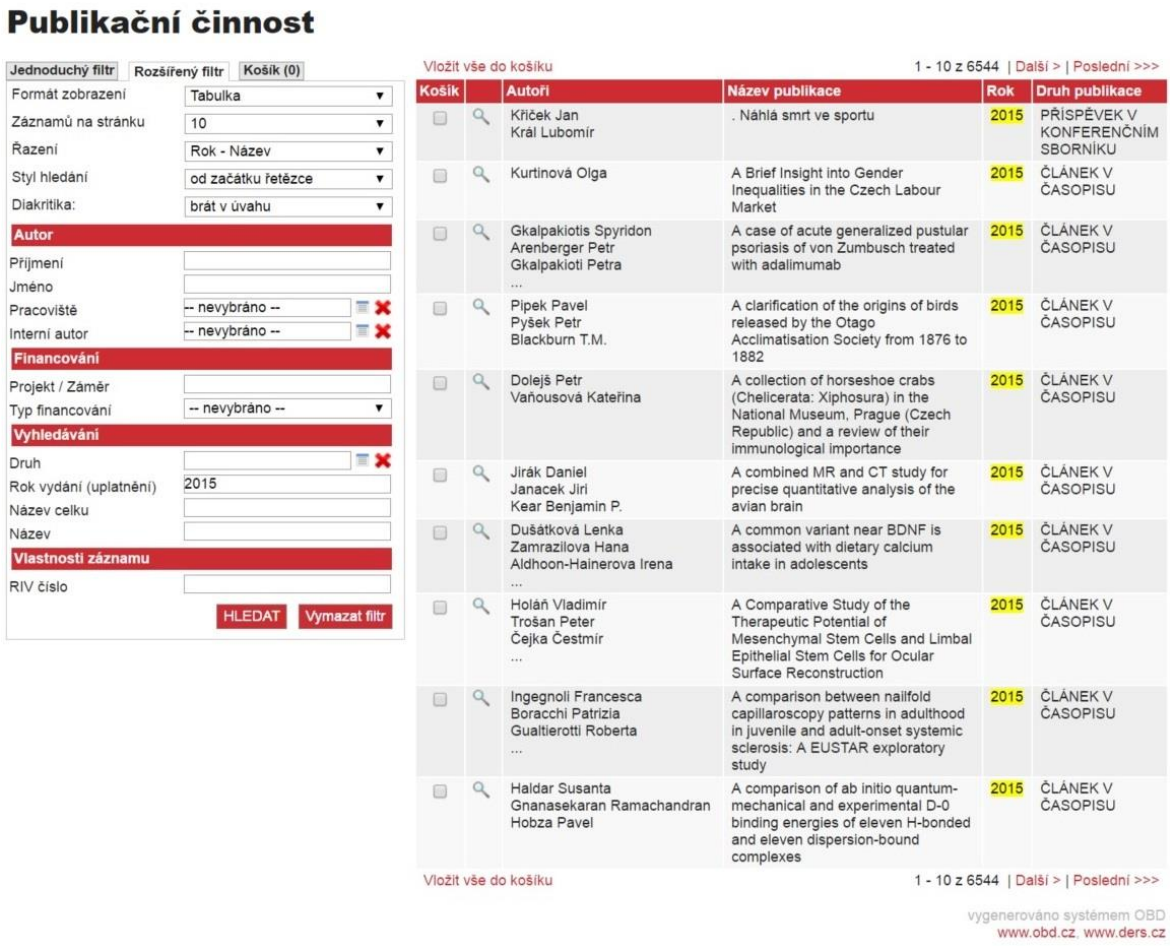
U tohoto dotazu, který se používá při práci se složkami, se podařilo zvýšit výkon pomocí manuálně přidaného hintu. Takto upravený dotaz je přibližně o 63 % rychlejší a využívá o 25 % méně procesorového času. Nicméně se po vygenerování alternativních dotazů nezařadil mezi prvních 5 dotazů s nejlepšími výsledky. Alternativy vygenerované pomocí Quest SQL Optimizeru dosahovaly lepších výsledků. Prostřednictvím alternativ byla data vracena o 10-12 % rychleji než uživatelsky upravený dotaz a využívaly pouze 27-25 % procesorového času oproti původnímu dotazu. Vygenerované dotazy dosahovaly v prvních testech velmi podobných výsledků, a proto nebylo z počátku možné určit

nejlepší dotaz. Po testu zátěže byl již výsledek jednoznačnější. Alternativy Alt116, Alt118 a Alt125 sice dosahovaly velmi podobných výsledků v případě paralelního přístupu pro 1 až cca 35 uživatelů, při vyšším počtu se však začaly výkony těchto dotazů více projevovat a díky tomu bylo možné určitě nejvhodnější dotaz, kterým byl Alt116. Tato alternativa projevovala v zátěžovém testu nejstabilnější výsledky.

Dotaz Alt116 byl přibližně 4krát rychlejší než původní dotaz a využíval o 49 % méně procesorového času. Dále klesl počet Session Logical Reads o 35 % a celkový počet řádků, nad kterými musí databáze provádět třídění, se snížil o 82 %.

8.4.3 Dotaz 3 – OBD Public

Dotaz je používaný v modulu OBD Public pro výpis informací souvisejících s publikační činností. Modul OBD Public je doplňkovým modulem systému OBD, který slouží ke zveřejnění záznamů z databáze OBD pro uživatele bez oprávnění pro vstup do jiných částí aplikace. Součástí modulu jsou jednoduché vyhledávací filtry a záznamy lze řadit podle několika kritérií. Filtry jsou rozděleny do 3 záložek na jednoduchý filtr, rozšířený filtr a košík. Jednoduchý filtr zobrazuje pouze základní filtrovací kritéria, kde lze hledat například podle názvu publikace, autora, RIV čísla, DOI atp. Rozšířený filtr rozšiřuje základní filtr o vyhledávání podle financování a dalších specifických informací jako je typ zdroje, ISSN, ISBN a tak dále. Záložka košík plní funkci filtru označených publikací. Každá publikace v OBD Public lze jednoduše pomocí zaškrtávátka označit a vložit jí tak do košíku. Vyfiltrováním v košíku jsou vráceny právě tyto označené publikace. OBD Public dále umožňuje zobrazení výsledků v několika různých formátech a tyto výsledky lze také exportovat do MS Word či MS Excel.



Obrázek 15: OBD Public (zdroj: autor)

Výsledkem dotazu, který je v modulu OBD Public použit, jsou všechny publikace, které jsou v OBD ve stavu *Publikovaný* (v rámci instituce, která OBD používá lze nastavit i jiné stavy či stavů více). Dotaz pro výpis publikací vrací následující informace:

- ID – Identifikátor publikace
- ROK – Rok publikace
- DRUH_PUBLIKACE – Literární forma použitá u publikace
- PRVNI_AUTOR – Autor, který je u publikace přiřazen na prvním pozici
- AUTORI – Interní autoři publikace
- TITUL_ORIG – Název publikace v originálním jazyce
- TITULY – Názvy publikace včetně překladů do jiných jazyků
- OZNACENO – Příznak určující označení publikace

8.4.3.1 Optimalizace a generování alternativ

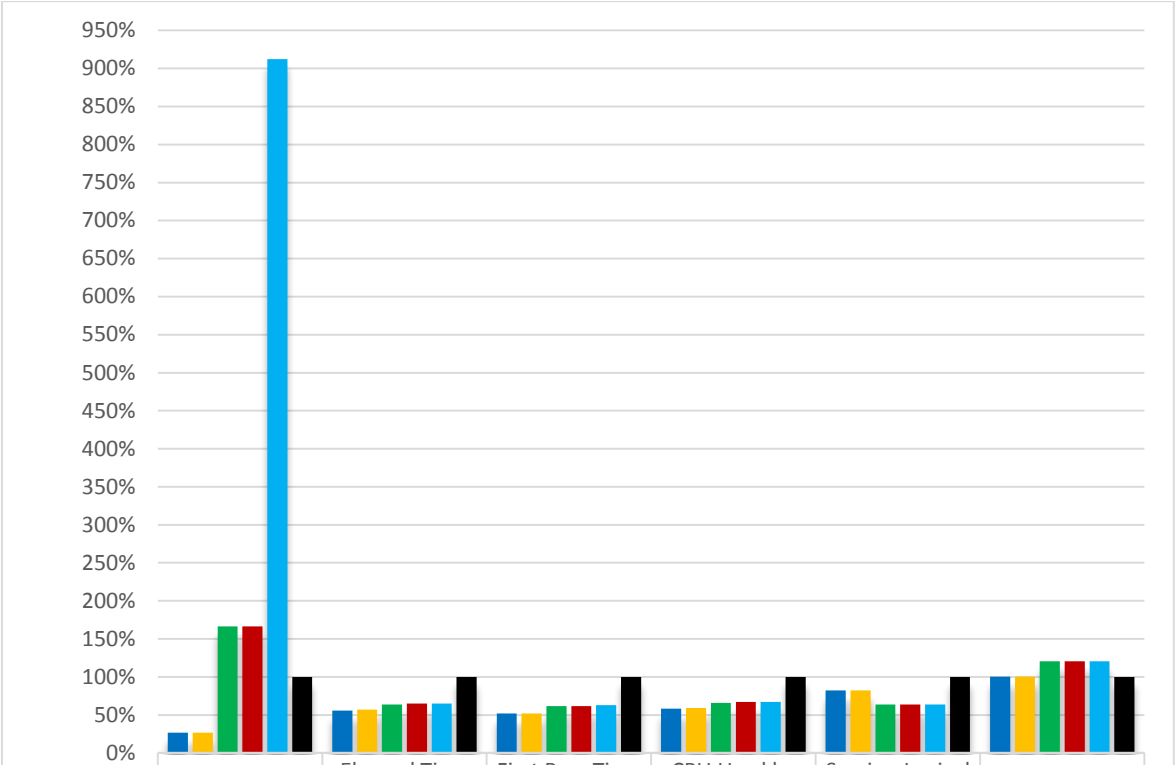
U tohoto dotazu se nepodařilo nalézt žádný index, který by pomohl ke zrychlení dotazu a nebylo ani nutné dotaz manuálně upravovat. Dalším krokem bylo generování alternativních dotazů. Prostřednictvím Quest SQL Optimizeru bylo vygenerováno 74 alternativ, ze kterých 30 dosahovalo lepších výsledků než původní dotaz. Naměřená data těchto 74 dotazů lze nalézt v příloze č. 6.

Z alternativ, které dosahovaly lepších výsledků v porovnání s původním dotazem, bylo dále vybráno 5 dotazů pro zátěžové testy. Vybrané alternativy jsou žlutě vyznačeny v příloze č. 6 a jejich zdrojové kódy lze nalézt v příloze č. 5. Následující tabulka shrnuje naměřené hodnoty dotazů určených pro zátěžové testy.

Scenario Name	Plan Cost	Elapsed Time (s)	First Row Time (s)	CPU Used by this Session	Session Logical Reads	Sorts (Rows)
Alt73	839	0,49	0,42	0,531	38 257	25 887
Alt71	838	0,50	0,42	0,539	38 240	25 887
Alt12	5 252	0,56	0,50	0,601	29 628	31 133
Alt35	5 253	0,57	0,50	0,612	29 645	31 133
Alt58	28 820	0,57	0,51	0,613	29 628	31 133
Original	3 159	0,88	0,81	0,916	46 593	25 823

Tabulka 6: Dotaz 3 – Naměřené hodnoty vybraných alternativ (zdroj: autor)

Jak lze pozorovat v předchozí tabulce, prostřednictvím alternativ došlo ke zrychlení dotazu z 0,88 sekund na hodnoty kolem 0,5 sekund. Tento fakt udává více než 40% zrychlení dotazu. I hodnoty ostatních kritérií dosahovaly ve většině případů lepších hodnot. Přehlednější porovnání lze vidět v následujícím grafu, který zobrazuje procentuální zvýšení či snížení výkonu u jednotlivých kritérií.



	Plan Cost	Elapsed Time (s)	First Row Time (s)	CPU Used by this Session	Session Logical Reads	Sorts (Rows)
Alt73	27%	56%	52%	58%	82%	100%
Alt71	27%	57%	52%	59%	82%	100%
Alt12	166%	64%	62%	66%	64%	121%
Alt35	166%	65%	62%	67%	64%	121%
Alt58	912%	65%	63%	67%	64%	121%
Original	100%	100%	100%	100%	100%	100%

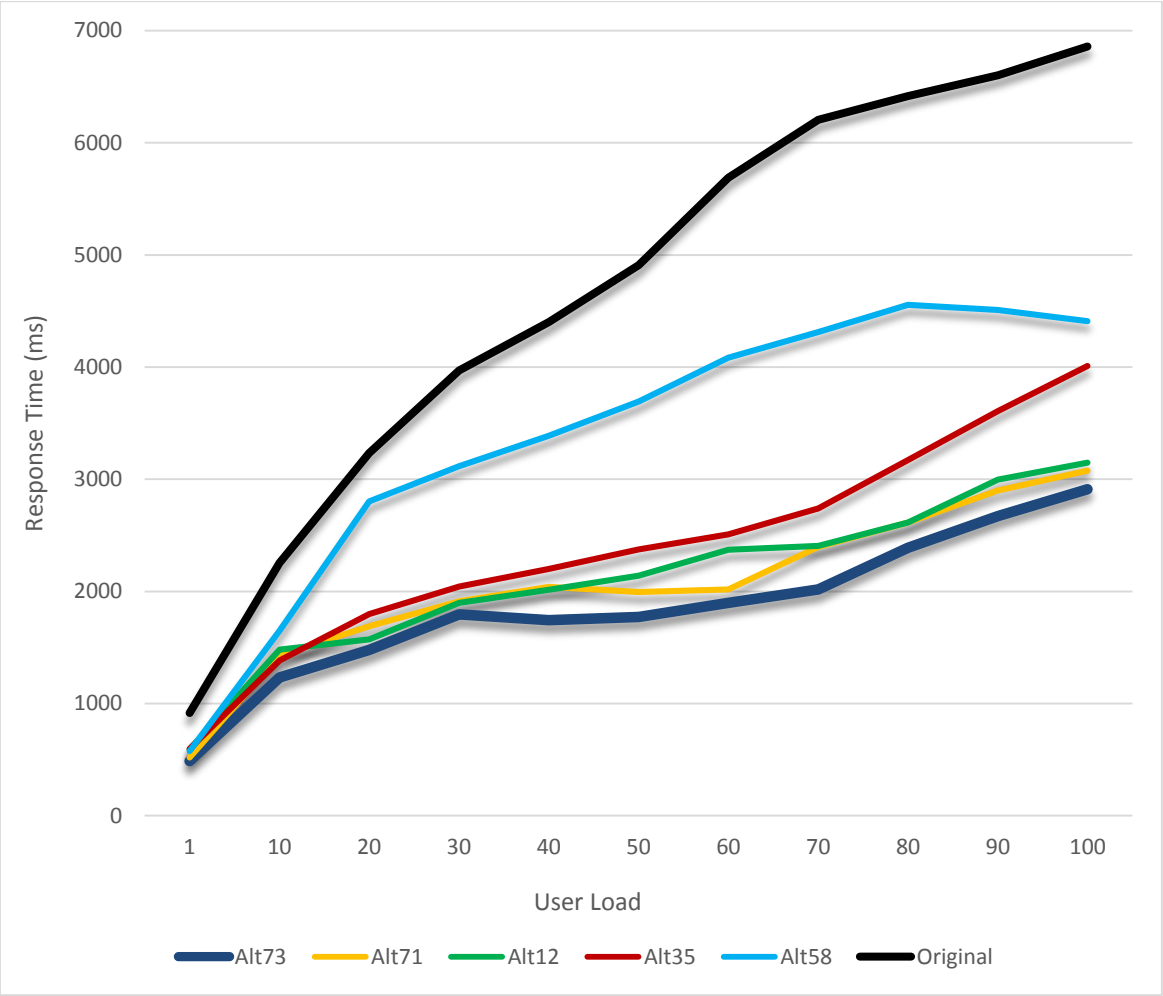
Graf 6: Dotaz 3 – Procentuální zlepšení dotazu pro vybrané alternativy (zdroj: autor)

Jak lze na první pohled z grafu vyčíst, i když měla alternativa Alt58 9krát vyšší cenu dotazu, tak i přesto vracela data rychleji než původní dotaz. Z tohoto důvodu nebyl na toto kritérium brán ohled při vyhodnocování výsledků. Po otestování dotazů jedním uživatelem dosahovaly nejlepších výsledků dotazy Alt73 a Alt71. Protože tyto alternativy měly téměř srovnatelné výsledky, tak prozatím nebylo možné rozhodnout o jednom nejlepším dotazu.

8.4.3.2 Zátěžové testování

Testování zátěže podstoupil původní dotaz a 5 vybraných alternativ. Protože je dotaz používán v modulu OBD Public, který slouží ke zveřejňování publikací široké veřejnosti, tak bylo cílem otestovat dotaz až pro 100 virtuálních uživatelů přičemž testování začínalo pro 1 uživatele s krokem po 10. Výsledné hodnoty zátěžového testu lze

vidět v následujícím grafu, který vyjadřuje závislost času odpovědí na počtu virtuálních uživatelů. Konkrétní hodnoty, ze kterých graf vychází lze nalézt v příloze č. 6.



Graf 7: Dotaz 3 - Zátěžový test (zdroj: autor)

Při testování zátěže jednotlivých dotazů dosáhl nejlepších výsledků dotaz Alt73. Tento dotaz se v paralelní testování pro 1 až 100 uživatelů vždy držel pod hodnotami ostatních alternativ, přičemž pro 1 uživatele databázi trvalo přibližně 0,5 sekund a pro 100 paralelně přistupujících uživatelů necelé 3 sekundy. V porovnání s původním dotazem, u kterého pro 1 uživatele databáze vracela data více než 0,9 sekund a pro 100 uživatelů dokonce téměř 7 sekund, se jedná o průměrné zrychlení cca 60 %.

8.4.3.3 Shrnutí výsledků

Podle průměrů uvedených v tabulce 6 je patrné, že nejlepšího výsledku dosáhl dotaz Alt73. Z tohoto měření však nebylo možné tvrdit zcela s jistotou, že tento dotaz je nejvhodnější. Dotaz Alt71 měl totiž výsledky téměř srovnatelné. Z naměřených dat je možné pozorovat zrychlení dotazu o více než 40 %, tedy z 0,88 sekund na hodnoty kolem

0,5 sekund. Po paralelním testování, kde rozdíly mezi výkony jednotlivých dotazů byly mnohem více viditelné, již bylo možné určit nejlepší alternativu. V zátěžovém testu byly nejlepší hodnoty naměřeny u dotazu Alt73. Tento dotaz se v průběhu celého měření, které probíhalo pro 1 až 100 uživatelů s krokem po 10, pohyboval pod hodnotami všech ostatních dotazů. S přibývajícím počtem virtuálních uživatelů byly rozdíly ve výkonu mezi původním dotazem a alternativou Alt73 čím dál více viditelné. Průměrné zrychlení dotazu dosahovalo přibližně 60 %. Například pro 100 uživatelů se jednalo o zrychlení z téměř 7 sekund na necelé 3. Dotaz Alt73 dále snížil požadavky na systémové zdroje. Pro zpracování dotazu databáze spotřebovala o 42 % méně procesorového času a Session Logical Reads se snížil o 18 %. U počtu řádků, pro které databáze musela provádět třídění, však ke zlepšení nedošlo, hodnota tohoto parametru zůstala stejná jako u původního dotazu.

9 Shrnutí výsledků

Tato práce představuje proces optimalizace SQL dotazů v databázovém systému Oracle. Vymezila postupy a pojmy související s návrhem databáze, jako jsou tabulky, indexy, pohledy či materializované pohledy. Značná část práce se věnovala SQL dotazům a související problematice. Dále byl podrobně popsán postup zpracování SQL příkazů, metody spojení, přístupové cesty a především optimalizátor a jeho úloha v procesu optimalizace. Jelikož nemusí být rozhodnutí optimalizátoru vždy tou nejlepší alternativou, nabízí databázový systém Oracle několik nástrojů a technik, kterými lze optimalizátor ovlivnit. Této problematice byla věnována samostatná kapitola, ve které byly vymezeny pojmy, jako jsou hinty, statistiky, SQL profily či SQL Plan management. Důležitým a zároveň prvním krokem v procesu optimalizace je identifikace problematických SQL. Práce shrnuje několik vybraných nástrojů, které se na detekci problematických SQL zaměřují. Převážná většina z nich zároveň nabízí možnosti pro jejich ladění.

Každý vývojář pracující s databázovým systémem Oracle musí znát mnoho pojmů a principů. Již při návrhu databáze nabízí Oracle nespočet možností. Neznalost či špatné použití databázových objektů může mít za následek velmi výrazný pokles výkonu celé databáze. V případě špatného návrhu je pak velice časově náročné databázi optimalizovat. Tato práce pomůže nahlédnout do problematiky designu databáze a představuje možnosti, které jsou pro správný návrh nezbytné. K lepšímu pochopení principů designu pomohou kusy kódů, které jsou součástí vykládané problematiky.

Nejčastějším úkonem vývojářů vzhledem k databázi je získávání dat. V některých aplikacích je využito technologií pro objektově relační mapování, v jiných se píšou nativní dotazy v jazyce SQL. Práce je v tomto směru určena především pro ty, jež sami sestavují SQL dotazy. Práce shrnuje důležité pojmy související s psaním SQL dotazů a vysvětluje postup databáze při jejich zpracování. Pro lepší pochopení a uvedení čtenáře do procesu optimalizace je součástí práce kapitola, která se věnuje optimalizacím dotazů v praxi. Optimalizací prošly dotazy aplikace OBD, kterou využívá většina vysokých škol v ČR již řadu let. Tato kapitola provede čtenáře procesem optimalizace a představí, k jak obrovskému zvýšení výkonu lze dospět, pokud je optimalizacím věnována dostatečná pozornost. V kapitole byly optimalizovány celkem 3 dotazy a u všech se jednalo o velmi znatelný nárůst výkonu. U prvního dotazu došlo k téměř pětinásobnému zrychlení a poklesu využití procesorového času o 73 %. Druhý dotaz byl zrychlen téměř 4krát,

přičemž využíval o 49 % méně procesorového času a celkový počet řádků, nad kterými databáze provádí třídění, klesl o 82 %. U posledního dotazu bylo dosaženo 40% zrychlení a při testování zátěže v podobě paralelního přístupu až 100 uživatelů dosahovalo průměrné zrychlení přibližně 60 %. Optimalizace se týkala pouze úpravy SQL. To znamená, že nebyla změněna struktura databáze a v tomto případě nebyly použity ani jiné struktury, jako jsou například indexy.

10 Závěry a doporučení

Cílem práce bylo uvést čtenáře do optimalizací v databázovém systému Oracle. Práce se zaměřuje na SQL dotazy a související náležitosti, které jsou k pochopení celého procesu optimalizace nezbytné. Jedná se o velmi rozsáhlou problematiku, která přesahuje rozsah této práce. Proto se práce zaměřuje pouze na čistě relační databázi a její běžné využití. Z tohoto důvodu nepopisuje postupy pro optimalizaci objektově-relačních databází ani databází distribuovaných.

V první části byl čtenář uveden do optimalizací v obecné rovině, kde byla popsána místa, na kterých je možné optimalizace provádět. Velká část práce je zaměřena na design databáze, který velice úzce souvisí s následnou optimalizací SQL dotazů. Tato část práce představila vybrané databázové objekty, které jsou v praxi nejčastěji používány a vzhledem k optimalizacím je velmi důležité znát možnosti, výhody a nevýhody každého z nich. Cílem této části bylo představit čtenáři rozmanité možnosti, které Oracle pro návrh databáze nabízí. Při volení nesprávných typů tabulek, nepoužívání nebo naopak nadměrnému používání indexů může dojít k velmi výraznému poklesu výkonu celé databáze.

Stěžejní část práce se zabývala SQL dotazy. V této kapitole byl podrobně popsán proces zpracování SQL dotazů, přístupové cesty, metody spojení a pojem exekuční plán. Znalost všech těchto pojmů je pro všechny, jež se chtějí zabývat optimalizacemi SQL v Oracle, naprosto nezbytná. Dále kapitola vysvětluje úlohu optimalizátoru. Tento nástroj zodpovídá za výběr nejvýkonnější metody provedení dotazu. Výkon posuzuje především podle ceny exekučního plánu, nicméně se na reálných příkladech ukázalo, že cena jakožto měřítko pro zkoumání výkonu dotazů není dostačující. Cena je jakési univerzální měřítko, jejíž výhodou je, že ji optimalizátor spočítá bez nutnosti spuštění příkazu. Na druhou stranu je velkou nevýhodou fakt, že vybraný exekuční plán nemusí být zdaleka nejvýkonnější. Z tohoto důvodu dnes moderní databázové systémy nabízí silné nástroje pro ovlivňování optimalizátoru a tak je možné převzít téměř úplnou kontrolu nad způsobem provádění dotazu. Nástroje a techniky pro ovlivňování optimalizátoru byly v této práci detailně popsány včetně možností jejich používání. Mezi tyto nástroje patří například hinty, které jsou v praxi často využívané i přesto, že je samotný Oracle nedoporučuje používat.

Další důležitou částí procesu optimalizace je detekce problémových dotazů. Této problematice byla věnována samostatná kapitola, která popisuje několik nástrojů, které lze pro identifikaci problémových SQL použít. Představeny byly nástroje produkované přímo společností Oracle a jeden, vytvořený a spravovaný společností Quest Software, která je již řadu let součástí společnosti Dell.

Poslední část práce byla zaměřena na využití znalostí z předchozích kapitol a demonstrování optimalizací v praxi. Optimalizovány byly 3 dotazy používané v aplikaci OBD, která je běžně používána v produkčním prostředí na většině vysokých škol v ČR. Představen byl celý proces optimalizace od identifikace dotazů až po zátěžové testování a vyhodnocení výsledků. Při optimalizaci bylo zajímavé zjištění, že cena exekučního plánu nemá vliv na výkon dotazů. Ač informace popsané v dokumentaci Oracle tvrdí, že cena je jakýmsi univerzálním měřítkem výkonu databáze a platí, že nižší cena = vyšší výkon, při optimalizaci dotazů toto tvrzení neplatilo. Naopak se v mnoha případech ukázalo, že dotazy s několikanásobně vyšší cenou byly rychlejší a zároveň využívaly zlomek systémových zdrojů. Při vyhodnocování výsledků optimalizace proto nebyl brán na cenu dotazu ohled. Proces optimalizace sestával ze 4 kroků. Prvním krokem byla manuální úprava dotazu v případě, že bylo na první pohled vidět, že lze pro zvýšení výkonu provést jednoduché úpravy. Takto upraven byl pouze jeden dotaz a úprava spočívala v přidání hintu MATERIALIZE, díky němuž byl výsledek vrácen o 63 % rychleji a využíval o 25 % méně procesorového času. Poté následovalo generování alternativ dotazů prostřednictvím nástroje Quest SQL Optimizer for Oracle. Tento nástroj generuje alternativy tak, že pomocí transformací a hintů upravuje původní dotaz. Alternativy jsou poté spouštěny a jejich výkon je posuzován podle několika kritérií. Po dokončení procesu generování alternativ bylo vždy vybráno několik z nich, které se podrobily zátěžovému testování v podobě paralelního přístupu několika desítek virtuálních uživatelů. V posledním kroku procesu optimalizace byly vyhodnoceny výsledky podle dat získaných ze zátěžových testů a z hodnot naměřených při generování alternativ. Ve všech třech případech se optimalizace zdařila a dotazy byly vráceny až několikanásobně rychleji.

11 Seznam použité literatury

[1] BRYLA, Bob, Kevin LONEY a Jiří HUF (překl.). *Mistrovství v Oracle Database 11g*. Vyd. 1. Brno: Computer Press, 2009, 700 s. ISBN 978-80-251-2189-4.

[2] CYRAN, Michele. Oracle Database Concepts: Memory Architecture. *Oracle Database: Documentation Library* [online]. 2005 [cit. 2015-11-17]. Dostupné z: http://docs.oracle.com/cd/B19306_01/server.102/b14220/memory.htm

[3] ROESER, Mary Beth. Oracle Database SQL Language Reference: Types of SQL Statements. In: *Oracle Database: Documentation Library* [online]. 2015 [cit. 2015-11]. Dostupné z: https://docs.oracle.com/database/121/SQLRF/statements_1001.htm

[4] CHAN, Immanuel a Lance ASHDOWN. Oracle Database Performance Tuning Guide: Performance planning. *Oracle Database: Documentation Library* [online]. 2014 [cit. 2015-11-17]. Dostupné z: https://docs.oracle.com/cd/E11882_01/server.112/e41573.pdf

[5] KYTE, Thomas. *Oracle: návrh a tvorba aplikací*. Vyd. 1. Brno: CP Books, 2005, 694 s. Programování (CP Books). ISBN 80-251-0569-5.

[6] ASHDOW, Lance. Oracle Database SQL Tuning Guide: Introduction to SQL Tuning. *Oracle Database: Documentation Library* [online]. 2013 [cit. 2015-11-16]. Dostupné z: http://docs.oracle.com/database/121/TGSQL/tgsql_intro.htm

[7] ASHDOW, Lance. Oracle Database SQL Tuning Guide: SQL Processing. *Oracle Database: Documentation Library* [online]. 2013 [cit. 2015-11-22]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm

[8] HARRISON, Guy. *Oracle performance survival guide: a systematic approach to database optimization*. Upper Saddle River, NJ: Pearson/Prentice Hall, 2010, xxv, 730 p. ISBN 01-370-1195-4.

[9] PROCHÁZKA, David. *Oracle: průvodce správou, využitím a programováním nad databázovým systémem*. 1. vyd. Praha: Grada, 2009, s. 45. Průvodce (Grada). ISBN 978-80-247-2762-2.

[10] URBANO, Randy. Oracle Database Administrator´s Guide: Managing Hash Clusters. *Oracle Database: Documentation Library* [online]. 2014 [cit. 2015-12-01]. Dostupné z: https://docs.oracle.com/cd/E24628_01/server.121/e41484.pdf

[11] ASHDOWN, Lance a Tom KYTE. Oracle Database Concepts: Tables and Table Clusters. *Oracle Database: Documentation Library* [online]. 2014 [cit. 2015-29-11]. Dostupné z: https://docs.oracle.com/database/121/CNCPT/schemaob.htm

[12] Database VLDB and Partitioning Guide: Partitioning Concepts. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2015-12-02]. Dostupné z: https://docs.oracle.com/cloud/latest/db121/VLDBG.pdf

[13] ASHDOWN, Lance a Tom KYTE. Oracle Database Concepts: Partitions, Views, and Other Schema Objects. *Oracle Database: Documentation Library* [online]. 2014 [cit. 2015-12-02]. Dostupné z: https://docs.oracle.com/database/121/CNCPT/schemaob.htm

[14] ASHDOWN, Lance a Tom KYTE. Oracle Database Concepts: Indexes and Index-Organized Tables. *Oracle Database: Documentation Library* [online]. 2014 [cit. 2015-12-05]. Dostupné z: https://docs.oracle.com/database/121/CNCPT/indexiot.htm

[15] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Optimizer Access Paths. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2015-12-21]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_optop.htm

[16] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Joins. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2016-01-02]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_join.htm

[17] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Generating and Displaying Execution Plans. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2015-12-13]. Dostupné z: http://docs.oracle.com/database/121/TGSQL/tgsql_genplan.htm

[18] WATT, Simon. SQL*Plus User's Guide and Reference: Tuning SQL*Plus. *Oracle Database: Documentation Library* [online]. 2013 [cit. 2015-12-20]. Dostupné z: https://docs.oracle.com/database/121/SQPUG/ch_eight.htm

[19] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Query Optimizer Concepts. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2015-12-13]. Dostupné z: http://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm

[20] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Query Transformations. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2015-12-13]. Dostupné z: http://docs.oracle.com/database/121/TGSQL/tgsql_transform.htm

[21] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Techniques for Influencing the Optimizer. *Oracle Database: Documentation Library* [online]. 2015

[cit. 2016-01-03]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_influence.htm

[22] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Optimizer Statistics Concepts. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2016-01-03]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_statscon.htm

[23] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Managing SQL Profiles. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2016-01-09]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_profiles.htm

[24] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Managing SQL Plan Baselines. In: *Oracle Database: Documentation Library* [online]. 2015 [cit. 2016-01-10]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_spm.htm

[25] BHATIYA, Rajesh a Immanuel CHAN. Oracle Database Performance Tuning Guide: Automatic Performance Diagnostics. *Oracle Database: Documentation Library* [online]. 2014 [cit. 2016-01-05]. Dostupné z: https://docs.oracle.com/database/121/TGDBA/pfgrf_diag.htm

[26] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Analyzing SQL with SQL Tuning Advisor. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2016-01-03]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_sqltune.htm

[27] ASHDOWN, Lance. Oracle Database SQL Tuning Guide: Optimizing Access Paths with SQL Access Advisor. *Oracle Database: Documentation Library* [online]. 2015 [cit. 2016-01-03]. Dostupné z: https://docs.oracle.com/database/121/TGSQL/tgsql_sqlaccess.htm

[28] Burleson Consulting: Oracle SQL tuning - Tune individual SQL statements. In: *Oracle Burleson Consulting* [online]. [cit. 2016-01-19]. Dostupné z: http://www.dba-oracle.com/art_sql_tune.htm

12 Zadání závěrečné práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Kombinovaná
Obor/komb.: Aplikovaná informatika (ai2-k)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Sucharda Ondřej	Bulharská 944, Nová Paka	I14719

TÉMA ČESKY:

Optimalizace SQL dotazů v databázovém systému Oracle

TÉMA ANGLICKY:

SQL query optimization in Oracle database

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl:
Cílem práce je popsat postupy optimalizace SQL dotazů. Představit proces optimalizace na vybraných SQL dotazech a poukázat na jejich výkon oproti dotazům původním.

Osnova:
1. Úvod
2. Základní pojmy
3. Návrh databáze
4. Optimalizace SQL, práce optimalizátoru
5. Detekce problematických SQL a možnosti ladění
6. Optimalizace vybraných dotazů, měření výkonnosti a interpretace výsledků
7. Závěry a doporučení
8. Seznam použitých zdrojů

SEZNAM DOPORUČENÉ LITERATURY:

Podpis studenta: ..... **Datum:** 30. 9. 2015

Podpis vedoucího práce: ..... **Datum:** 30. 9. 205