Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Optimalizace databáze Oracle

Bakalářská práce

Autor: Barbora Spěváková
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Barbora Tesařová, Ph.D.

Hradec Králové Duben 2016

Prohlášení:

Prohlašuji, že jsem bakalářskou zpracovala samostatně a s použitím uvedené literatury.

V Hradci Králové dne 28.4.2016 Barbora Spěváková

Poděkování:

Děkuji vedoucí bakalářské práce Ing. Barboře Tesařové, Ph.D. za metodické vedení, užitečné připomínky a cenné rady v průběhu psaní této práce.

Anotace

Bakalářská práce se zabývá optimalizací databáze Oracle. V teoretické části je popsaná optimalizace obecně. Dále se zaměřuje na optimalizátor a související informace. Protože téma optimalizace je rozsáhlé, práce se podrobněji soustředí na exekuční plány a statistiky. Praktická část se zaměřuje na vytvoření vzorové databáze, v databázovém systému Oracle Database 12c. V této části se dále nachází optimalizace SQL příkazu. Ještě před samotnou optimalizací se aktualizují statistiky. Příkaz je optimalizován pomocí SQL Tuningu. Další pozorování databáze, SQL Tuningu a statistik probíhá v Oracle Enterprise Manager Database Express 12C. Dále se pracuje s hinty, jejich vliv na optimalizátor a výsledné exekuční plány.

Annotation

Title: Optimization database Oracle

This bachelor thesis deals with optimization of Oracle database. The optimization in general is described in the theoretical part. The thesis also focuses on optimizer and information connected with it. According to the large extent of the topic the bachelor thesis is focused mainly on the execution plans and statistics. In the practical part a sample database in Oracle database 12c system is created as well as the optimization of SQL command. But before the proper optimization the statistics have to be updated. SQL command is optimized by SQL Tunning. Another observing of the database, SQL Tunning and statistics are carried out in Oracle Enterprise Manager Database Express 12c. Further on the thesis deals with hints and with their impact on optimizer and final execution plans.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Databáze jsou nedílnou součástí IT oboru, bez kterých se neobejde žádný podnik ani malá firma. Umění pracovat s těmito databázemi je důležitou dovedností získanou studiem a použitelnou v praktickém životě.

Proto, aby se tato dovednost mohla efektivně využívat, je potřeba mít v této oblasti dostatek vědomostí. Mezi tyto vědomosti určitě patří optimalizace. Ta je součástí tvorby a provozu databází. Následující kapitoly se soustředí na optimalizaci databáze a to výhradně na ty, které jsou nasazeny na databázovém systému od společnosti Oracle.

Teoretická část se zaměřuje na popis optimalizace a všechny důležité prvky databáze, které jsou s optimalizací spojené. Komplexně popisuje vybrané prvky a vysvětluje jejich funkci jak v databázi, tak i v optimalizaci. Následně vysvětluje spojení a spoluprací prvků. Protože je téma optimalizace rozsáhlé, zaměřuje se spíše na statistiky a exekuční plány. Ale soustředí se i na optimalizaci, optimalizátor a hinty.

V praktické části bude vytvořena vzorová databáze, pomocí databázového systému Oracle Database 12c. Zužitkují se zde informace vysvětlené v teoretické části. Na vzorové databázi, bude aplikována optimalizace SQL příkazu a práce s hinty.

2 Cíl práce

Bakalářská práce se snaží přiblížit oblast optimalizace a to hlavně exekučních plánů a statistik.

Cílem teoretické části je definování optimalizace, představení optimalizátoru a jeho funkce v optimalizaci. Dále budou představeny další součásti databáze a optimalizace. Podrobnější bude seznámení se statistikami a exekučními plány.

V praktické části bude vytvořená vzorová databáze s kontejnery, ve které bude předvedena optimalizace SQL příkazu pomocí statistik a exekučních plánů. Optimalizovaný SQL příkaz, bude dál zkoumán a nasazen do dalšího kontejneru databáze. Zde bude sledován a v případě potřeby bude dále optimalizován. Poslední část bude věnovaná hintům a jejich vlivu na práci optimalizátoru. V praktické části se aplikují znalosti z teoretické části.

3 Teoretická část

3.1 Optimalizace

Optimalizace je proces, kde se systém modifikuje. Systém je potom rychlejší, efektivnější, lépe využívá paměťový prostor [1]. Základem optimalizace je proces nalezení slabého místa systému. Následuje navržení variant řešení a proces výběru nejlepší varianty řešení.

Systém lze modifikovat v několika úrovních. Nejvhodnějším začátkem je optimalizace na vyšší úrovni. Ty mají větší dopad a těžko se aplikují na dokončeném projektu. Postupuje se od vyšší vrstvy k nižší vrstvě.

Architektura systému je na nejvyšší úrovni optimalizace. Návrh systému ovlivňuje jeho výkon. Proto je snaha o snížení a propojení vhodných cest. Návrh systému jde velice těžko změnit, hlavně kvůli komponentám, ze kterých se skládá systém a které se těžko nahrazují.

Další vrstva je datová a algoritmická. Ta ovlivňuje účinnost více než jakákoliv jiná vrstva. Změna datové struktury je složitější než změna algoritmů. Správné použití algoritmů v klíčových místech zvýší rychlost a sníží čas [1].

Někdy je potřeba udělat kompromisy. Tedy optimalizace jednoho aspektu se provádí na úkor aspektu jiného. Zde je potřeba si ujasnit co je důležitější a vyvarovat se tomu, aby optimalizace nebyla zdrojem jiných nepřesností.

Po provedení špatné optimalizace se mohou objevit ještě horší problémy. Ty se potom mohou vyřešit, nebo je lepší optimalizaci úplně vynechat.

Databázi a její funkčnost ovlivňuje několik faktorů. Některé nedokonalosti databáze nejsou objeveny hned po zavedení do provozu, ale až postupem času, kdy databáze obsahuje více dat, tzn. je vytíženější. To vede k dlouhým časům a nepřesnostem.

Po určení nejproblematičtějších částí databáze a zjištění důvodů problematických částí, bude provedena optimalizace a vyvarovaní se chybám.

3.1.1 Způsoby optimalizace

Nejčastější námitky a připomínky, které následně vedou k optimalizaci, mají samotní uživatelé. Ty s databází pracují a jako první si uvědomují některé nedostatky [1]. Zodpovědný za výkon, integritu a bezpečnost databáze je její správce. Ten se dále soustředí na její údržbu. Vývojář databáze má zodpovědnost za stávající a nové databáze. Stará se o jejich vývoj, plánování, shromažďování údajů, zajišťuje jejich funkčnost. Poslední je návrhář databáze, ten se zaměřuje na programovou část databáze. Dále jí navrhuje a analyzuje [2].

3.1.1.1 Optimalizace SQL dotazů

Častou příčinou optimalizace je pokles výkonu a chyby. K tomu dochází z různých důvodů. Nevhodné sestavení databáze a tabulek vede ke špatnému výkonu databáze. Zastaralé statistiky se musí aktualizovat, jinak vedou k nepřesnostem a zbytečnému úbytku výkonu databáze. Vnořené smyčky ztěžují běh SQL dotazů. Běh více úloh najednou, vede k zatížení paměti a mezi paměti [1].

Jedním z prvních kroků je určit problémový SQL příkaz. Problémový SQL příkaz se může opravit ručně, nebo se může využít nástroj od společnosti Oracle. Nástroje pracují rychle a efektivně. Tím šetří náš čas i peníze.

Po nalezení problémového SQL příkazu je důležité, určit co způsobuje jeho problémy. Prvním krokem je zjistit, zda je dotaz napsán podle obecných pravidel. Vyjmenovávají se sloupce, se kterými se pracuje, a ne pouze tabulky. Pro hledání pomocí dotazů v rozsáhlé databázi, se volí vhodné klauzule. Je snaha šetřit s daty, pamětí, výkonem. Na začátek se dávají podmínky, které vyřadí co nejvíce řádků, další podmínka potom už tolik řádků nemusí prohledávat. Důležitá je práce s indexy. Vybírají se indexy, které načtou co nejméně záznamů z tabulky. Neprohledává se celá tabulka, ale využívá se index. Dalším krokem je zjistit jak dotaz pracuje při menším a větším množstvím dat. Je potřeba dotaz testovat [1].

3.1.1.2 Více kroků do jednoho dotazu

Vede k rozsáhlým dotazům, které se provádějí déle. Tyto dlouhé dotazy se dají rozdělit na menší komponenty, které si vytvoří v každém kroku dočasné tabulky a mezi-výsledky. Dávková operace potom může slučovat data z různých tabulek, a

vkládat výsledky do nových tabulek. Dávkové operace postačí menšímu množství dat. Ve větším měřítku je potřeba operace rozdělit do několika kroků [1].

3.1.1.3 Indexy

Index je definovaná unikátní hodnota sloupce tabulky. V databázích funguje jako konstrukce, která se využívá k rychlému přístupu k datům.

Speciálním druhem indexu je primární klíč a cizí klíč. Primární klíč je jedinečný v tabulce a nemá null hodnotu. Cizí klíč se používá ke vztahu mezi dvěma tabulkami.

Index slouží ke zlepšení výkonu SQL příkazu. V rámci optimalizace je dobré indexy do databáze zavést, nebo je změnit. Pokud tabulka obsahuje originální hodnoty, je potřeba zvážit zda jsou indexy potřeba. Problém nastává, když hledaný index není nalezen, nebo neexistuje. V tomto případě se do databáze přistupuje pomocí FULL TABLE SCAN. Tento přístup neprochází databázi podle indexů, ale postupně jí prochází po řádcích tabulky. Je to způsob neefektivní a zdlouhavý [3].

Druhy indexů

- **B-Tree index**

  Skládá se z kořene (root), uzlu (node) a listu (leaf). Index vytvoří uzel o velikosti fyzického bloku disku**.** Přechod o úroveň níž zabere jednu I/O operaci.

- **Bitmap index**

  Tento druh indexu je mnohem efektivnější než B-Tree index. Bitmap index se rozděluje na dynamické a pravé. Pravé vrací výsledek pouze po přistoupení k indexu. Dynamické k indexu přistupují pomocí RowID, a jsou kombinací bitmapového indexu s B-Tree indexem. Multi-Bitmap vzniká při spojení více Bitmap indexů.

- **R-Tree index**

  R-Tree index je vhodný pro prostorově přístupové metody a indexování více rozměrných struktur. Rozdělují se na hierarchicky vkládané a potencionálně překrývané. Zde se využívá princip minimálně vázaných obdélníků.

- **MBR** - minimum bounding rectangles – (minimální vázané obdélníky)
  Každý záznam, který není listem, vkládá identifikaci dceřiného uzlu, a MBR všech záznamů dceřiného uzlu. Další výhodou je, že MBR využívá algoritmy pro vkládání, smazání, a kontrolu prvku v geometrické oblasti [4].

Správné užívaní indexů

Není podmínka používat indexy. Pokud jsou v tabulce originální hodnoty, mohou být tyto hodnoty používat jako indexy. Tímto způsobem šetří i paměť. V případě, že je tabulka využívaná ke čtení, je vhodné využít indexování. Pokud je tabulka často aktualizovaná, lepší volbou je používání méně indexů.

Není potřeba používat indexy v tabulkách, kde nejsou využívané. Tyto indexy je žádoucí sledovat. Následně index může být zrušen, nebo přeměněn na neviditelný. Když by byl index znovu potřebný, lze ho znovu zviditelnit. Další možností je vytvoření kompozitního klíče, pro více sloupců. Sloupce, které obsahují null hodnoty, nejsou vhodné k indexování [1].

3.1.1.4 Propojování tabulek

Při načítání dat je často potřeba načíst data ze dvou různých tabulek. Proto se tabulky propojují. Je dobré propojit co nejméně tabulek. Musí být předem určeno vhodné spojení. Někdy je nutné si tato spojení vyzkoušet a zjistit časy dotazu. Některé dotazy trvají déle, protože se tabulky dlouho prohledávají. Vhodné je se vyhýbat úplnému prohledávání a vybírání efektivních indexů, které vypíší z tabulky co nejméně informací. Jednotlivé cesty propojených tabulek jsou vyhodnoceny optimalizátorem [1].

3.1.1.5 Další

Eliminace logických operací čtení dat

Některá data v databázi jsou stálá a často se nemění. Přesto se na těchto datech provádí několik milionů čtení za den. Vede to k zahlcení databáze a pomalému výkonu. Řešením je změna návrhu aplikace, aby se zabránilo zbytečnému provádění příkazu.

Eliminace nadbytečných přístupů do databáze

Při optimalizaci databáze je vhodné sloučit více procedur do jedné. *„Tento přístup se seskupováním dotazů je použitelný u aplikací typu tenký klient, které sestávají z několika aplikačních vrstev."* [1]

Nepotřebné operace, přistupují zbytečně do databáze a tím spolknou prostředky procesoru serveru. Operace, které jsou nepotřebné, se neprovádí. Operace, které jsou nepotřebné ke spouštění samostatně, se slučují do jedné procedury. Tím se eliminují nadbytečné operace. Často používané hodnoty se zadávají do proměnných.

Ukládání dat způsobem dotazování

Existují dotazy na databázi, o kterých je známo, že je budou uživatelé provádět. *„Proto usnadníme Oracle databázi s transformací formátu dat v tabulkách do formátu, prezentovaného uživateli."* [1] Vyžaduje to aktualizaci materializovaných pohledů nebo tabulek pro generování sestav. Provádění dotazů je potom rychlejší, databáze provádí méně logických a fyzických operací čtení.

Zabránění opakovaným připojení k databázi

Proces připojování k databázi je zdlouhavý. Připojováním k databázi se ztrácí více času, než příkazy. Takže je opakované připojování k databázi zbytečné. Při otevření databáze se provede několik kroků, které prodlužují čas připojení databáze. Po otevření je možné s databází pracovat. Mnohem lepší je, připojení k databázi nechat připojené. Aplikace potom není pomalá. Otevřené připojení k databázi je otázkou návrhu databáze.

3.2 Optimalizátor

Optimalizátor určuje nejlepší způsob provedení SQL příkazu. Výstupem optimalizátoru je optimální plán. Přesněji generuje syntakticky správné příkazy SQL. Další službou, kterou poskytuje je indexování. Volí exekuční plán[1] s nejnižšími náklady a pro jeho vyhodnocování využívá statistiky[2]. Optimalizátor má výborné

[1] Exekuční plán – zobrazuje kroky nezbytné k provedení SQL příkazu. Podrobněji v kapitole 3.4
[2] Statistiky – Popisují detaily o databázi a databázových objektech. Podrobněji v kapitole 3.3

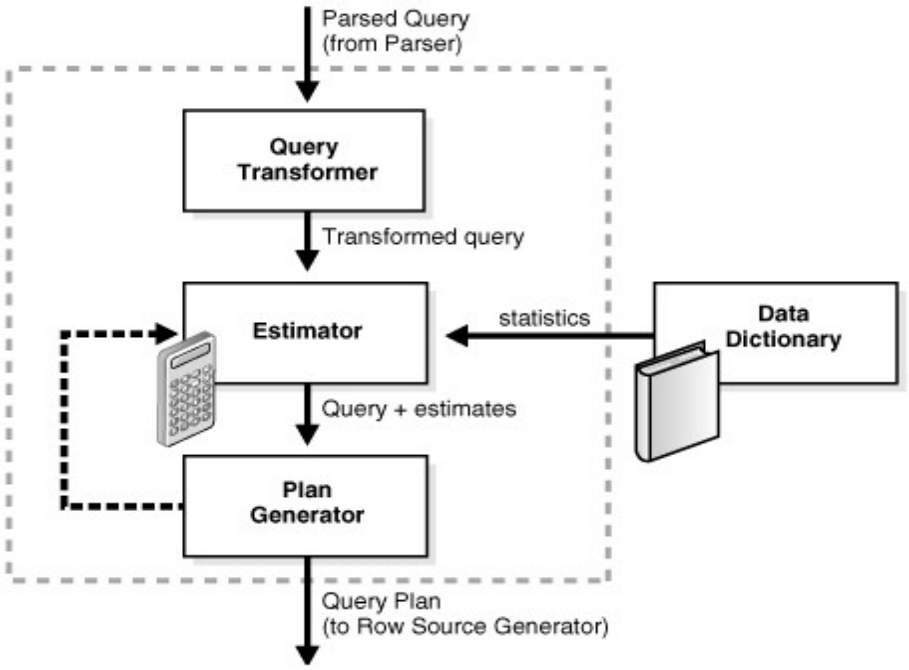
postavení při získávání statistik, protože databáze má plno interních statistik, ke kterým má přístup. Optimalizátor Oracle je uspořádává a pomocí jednotlivých kroků stanovuje sled, podle kterého SQL příkaz přistupuje k tabulkám. Mimo jiné také přiděluje metodu pro každou operaci spojení JOIN[3] a určuje operace prováděné na datech [5].

Optimalizátor se skládá ze tří komponent, které rozhodují o zvolení vhodného plánu pro určitý SQL příkaz [2].

Komponenty:

1. **Transformátor dotazu** – Ten určuje, zda je výhodné SQL příkaz přepsat. Transformátor zjistí, jestli by byla vhodnější jiná alternativa příkazu, s nižšími náklady [5].
2. **Odhadce** – Odhadce zjišťuje náklady každého plánu pomocí statistik uložených v datovém slovníku. Selektivita je vázaná na predikát dotazu jako je WHERE, LIKE, nebo kombinace těchto predikátů. Kardinalita je počet řádků vrácených každou operací v plánu provádění. Náklady představují jednoty využívání zdrojů [5].
3. **Generátor plánu** – Jeho funkcí je porovnávat plány a následně zvolit ten, který má ty nejnižší náklady. Tento zvolený plán se nazývá plán provádění – exekuční plán [5].

[3] JOIN – Spojení mezi tabulkami. Podrobněji v kapitole 3.2.4



Obrázek 1 Komponenty optimalizátoru. Zdroj: [2]

3.2.1 Náklady

Nejvhodnějším exekučním plánem je plán, který má co nejnižší náklady. Tyto náklady se vypočítají z I/O a CPU operací. Exekuční plány se mohou lišit různými náklady, ale také odlišnými schématy. Pokud jsou, rozdílní uživatele v databázi, kteří ukazují na různá data ve stejné databázi, vede to k různým exekučním plánům. Schémata však mohou být stejná, potom se optimalizátor rozhoduje podle nákladů, které se liší [5].

Náklady a čas jsou předpokladem zdrojů, které jsou potřebné pro jedno provedení dané operace. Týká se i potomků dané operace, která je potřeba k dokončení rodičovské operace. Jako příklad je uveden vnořený dotaz.

```
select JMENO, PRIJMENI, PLAT, ZAMESTNANEC_ID
from ZAMESTNANCI A
where PLAT >
        (SELECT AV(PLAT)
        FROM ZAMESTNANCI B
        WHERE A.ZAMESTNANEC_ID = B.ZAMESTNANEC_ID);
```

Obrázek 2 Ukázka vnořeného dotazu (metody). Zdroj: Autor

Na obrázku 2, je příklad vnořeného dotazu. Rodičovská metoda A, která je zvýrazněná tučně, má vnořenou metodu B. V tomto dotazu je metoda B nezbytnou součástí metody A. Náklady a čas s ní spojeny se přičítají k rodičovské metodě.

Databáze optimalizuje každý SQL příkaz na základě statistických údajů. Při provádění plánu jsou různé přístupové cesty a připojené metody [6]. Náklady jsou číslo, které představuje odhadované využití prostředku pro plán provedení. Pro každou možnost plánu optimalizátor přiděluje cenu. Následně vybere plán, který má nejnižší náklady.
Náklady ovlivňuje:

- Objem dat a statistik
- Typy proměnných a jejich hodnoty
- Inicializační parametry, nastavení jejich hodnot globálně, nebo na úroveň relace

3.2.2 Přístupy

Důležitým rozhodnutím je, jak optimalizátor načítá data z databáze. Na přístupu závisí celé provedení SQL příkazu. Optimalizátor dokáže tyto přístupy měnit, aby získal co nejnižší náklady na provedení exekučního plánu.
FULL TABLE SCAN - Dochází k úplnému procházení tabulky a načtení všech řádků. Ty porovnává predikátem WHERE, tím hledá řádek, který splňuje požadavky. Tabulka se čte po blocích a každý blok se čte pouze jednou. Bloky se nacházejí vedle sebe a je možné k nim přistupovat MULTIBLOCK. Počet bloků N nastavuje parametr DB_FILE_MULTIBLOCK_READ_COUNT. Maximální počet bloků v MULTIBLOCK = N [7].

INDEX SCAN - Přístup INDEX SCAN, data prohledává pomocí indexů a dále se dělí [7].

UNIQUE SCAN - Unikátní index vrací jeden řádek. Je používán u primárního klíče, dále je provedena kontrola UNIQUE nebo PRIMARY KEY[4] [7].

RANGE SCAN - Skenování v určitém rozsahu, je určován predikátem WHERE. Podle počtu řádku vrací nulu nebo více řádků [7].

FULL SCAN - Tento přístup postupně projde všechny záznamy indexů [7].

FAST FULL SCAN - Přístup k datům přímo pomocí indexu. Hledání pouze ve sloupci tabulky [7].

INDEX JOIN - Spojení všech sloupců tabulky dotazovaných pomocí WHERE. Následné nalezení hodnoty pomocí indexu, který patří mezi spojené sloupce [7].

BITMAP – Je používán pro klíčové hodnoty, kde každý bit polohy je převáděn na řádek. BITMAPY efektivně slučují indexy, které odpovídají podmínkám WHERE a pomocí logických operací AND a OR [7].

ROWID SCAN - Je to nejrychlejší způsob provádění. Řádek určuje přesné umístění v databázi a datovém bloku. K prohledávání ROWID SCAN je většinou přistupováno po skenování indexů. ROWID SCAN tak nalezne přesné umístění řádků a jeho hodnoty [7].

3.2.3 Hinty

Hinty jsou používány v situaci, kdy optimalizátor nezvolí nejvhodnější plán provádění. Toho si může všimnout např. tvůrce databáze, který pracuje s daty a samotnou databází. Tvůrce ví, že plán provádění není nejvhodnější a k ovlivnění optimalizátoru použije hinty. Je to mechanismus, který instruuje optimalizátor jaký plán spuštění zvolit, na základě specifických kritérií. V některých případech, je při používaní hintů , použitá celá sada, aby byl zajištěn optimální plán. Nevýhodou může být, že hinty vedou k neočekávaným exekučním plánům. Hinty se vkládají do SQL dotazu za SELECT, UPDATE, DELETE. Důvodem pro použití je měnící se databáze. Optimalizátor a jeho statistiky jsou zastaralé a to vede ke špatné volbě exekučního plánu [5].

[4] PRIMARY KEY – primární klíč

```
SELECT/*+ FULL*/ id_zamestnanec jmeno, prijmeni
FROM aazamestnanci;
```

V příkladu je ukázáno, že se hinty vkládají do SQL dotazu za SELECT do `/*+ HINT */`. V tabulce 1, jsou uvedené nejpoužívanější HINTY s popisy jejich funkcí.

Tabulka 1 Vybrané hinty a jejich funkce. Zdroj: [8]

ALL_ROWS	Nejlepší propustnost a minimální spotřeba zdrojů.
FIRST_ROWS	Vypíše prvních n řádků velice rychle a efektivně.
FULL	Úplné prohledání tabulky.
NO_INDEX_SS	Přeskakování ze zadaných indexů na zadané tabulky.
PARALLEL	Přistupuje k tabulce pomocí paralelního zpracování.

3.2.4 Joiny

V SQL příkazu se používá klauzule FROM, která definuje tabulky, se kterými se pracuje. Joiny jsou spojení mezi těmito tabulkami. Jsou definované stromovou strukturou. V situaci kdy máme dvě tabulky přistupujeme k nim nejčastěji zleva do práva. Samozřejmě je možné přistupovat zprava doleva.
Joiny volí optimalizátor, musí zvolit nejvhodnější operaci pro obě tabulky.

1. Volba přístupové cesty, načtení dat z tabulky.
2. Volba JOIN METODY, optimalizátor musí zvolit, jakou metodou se řádky spojí.
3. Volba pořadí.

Metody jsou mechanismem spojení dvou tabulek. Optimalizátor volí metodu s nejnižšími náklady na základě statistik.
NESTED LOOPS JOINS - Je metoda vnořených smyček. Tabulky jsou rozdělené na vnější a vnitřní. Pro řádek vnitřní tabulky je hledán řádek vnější tabulky, které k sobě patří (určuje predikát). Když jsou k dispozici indexy, přistupuje se přes řádky. Tato metoda nejlépe funguje na menších tabulkách s indexy [7].
HASH JOINS - Používá se při spojování větších tabulek. Menší z tabulek se používá k vytvoření hashovacího klíče. Hashovaná tabulka se uloží do paměti. Podle hashovacího klíče, se pak prochází velká tabulka a hledají se řádky, které splňují

podmínku. Při hashování se používá deterministický algoritmus, kde jsou hodnoty 1 až N, kdy N je počet řádků hashovací tabulky. Je nejúčinnější [7].

SORT MERGE JOINS - Dvě tabulky se setřídí. Pro každý řádek první tabulky se najde odpovídající řádek druhé tabulky. Když se řádky nerovnají, řádek s menší hodnotou je nahrazen dalším řádkem tabulky. Při třídění se používají nerovnosti. Když jedna z tabulek má indexy, tak se netřídí. Třídění je nákladné [7].

CAERTESIAN JOINS - Nastává, když tabulky nemají podmínky ke spojení. Každý řádek první tabulky se spojí s každým řádkem druhé tabulky. Vzniká tak kartézský součin. Velice nákladné, vhodné pouze pro malé tabulky [7].

3.2.5 SQL Tuning

Je automatické ladění SQL příkazu. Ruční ladění je složité, opakující se a časově náročné. Při SQL Tuningu optimalizátor provádí dodatečnou analýzu a ověřuje, jestli může být exekuční plán vylepšen. Výstupem je série akcí, spolu s jejich odůvodněním a očekávaným přínosem pro výrazně lepší plán. Ladění SQL příkazu může trvat i několik minut. Při ladění je příkaz velice vytěžován. Automatic Database Diagnostic Monitor (ADDM), identifikuje zatížení SQL [5].

3.2.6 SQL Access Advisor

Je ladící nástroj, poskytující poradenství a materializované pohledy. Je určen pro náročné příkazy a docilování optimálního výkonu příkazu. Jediným mínusem je, že je časově náročný a náročný na prostorové požadavky paměti. Spouští se pomocí manageru, nebo balíčkem PL/SQL DBMS_ADVISOR. Ten se skládá z analýz a poradenských funkcí [5].

3.3 Statistiky

Statistiky jsou rozhodujícím faktorem pro vybrání nejvhodnějšího exekučního plánu optimalizátoru. Volba exekučního plánu je jen tak dobrá, jako statistiky optimalizátoru.

Popisují detaily o databázi a databázových objektech. Ke statistikám je přistupováno pomocí datového slovníku. Optimalizátor na základě výpočtů selektivity z nákladů, odhadne náklady na spuštění plánu [5]. Pomocí statistik zjistí,

jak je SQL příkaz náročný a zvolí adekvátní plán. Statistické údaje jsou uloženy v datovém slovníku.

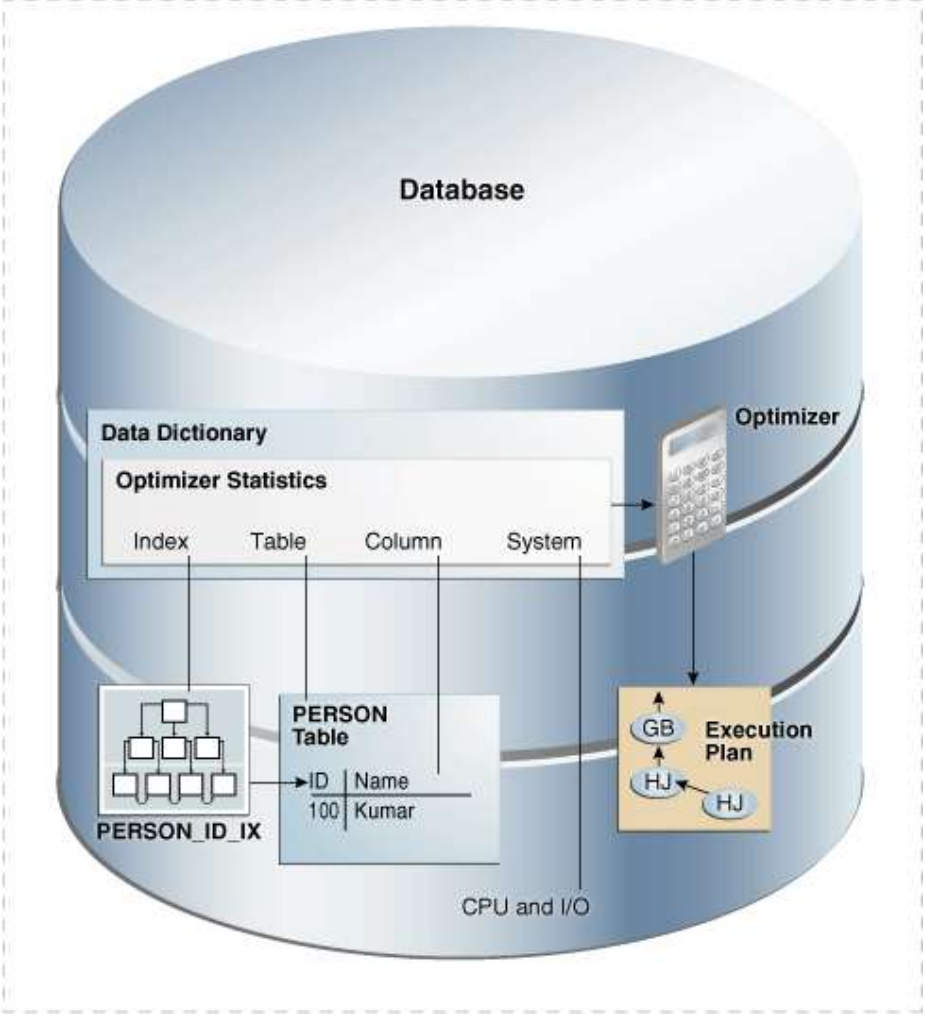
3.3.1 Datový slovník

Datový slovník definuje všechny objekty a schémata databáze (tabulky, pohledy, indexy, sekvence, procedury, apod.). Zaznamenává, kolik prostoru bylo databázi vyčleněno a kolik prostoru opravdu využívá. Zahrnuje uživatelské role a jejich práva společně s uživateli. Objekty slovníku jsou auditorované, aktualizované a přístupné. Změny se aktualizují průběžně.

Slovník je složen ze základních tabulek, ve kterých jsou informace o databázi. Přístup k nim má pouze SYSTEM zřídka uživatel. Uživatelská zobrazení jsou shrnující informace v datovém slovníku. SYSTEM je majitelem slovníku a vlastníkem všech tabulek. Běžný uživatel by se základními tabulkami neměl pracovat, mohl by ohrozit integritu dat [7].

Statistiky zahrnují následující:

- Tabulkové statistiky,
  - Počet řádků
  - Počet bloků
  - Průměrná délka řádku
- Sloupcové statistiky
  - Počet odlišných hodnot (NDV) ve sloupci
  - Počet NULL hodnot ve sloupci
  - Distribuce dat (histogram)
  - Rozšířená statistika
- Index statistiky
  - Počet listových bloků
  - Počet úrovní
  - Skupinový faktor
- Systémové statistiky
  - I/O využití výkonu
  - CPU využití výkonu



Obrázek 3 Tvorba statistik. Zdroj: převzato z www.dianarobete.com

3.3.2 PL/SQL

PL/SQL je procedurální jazyk, který je nadstavbou jazyka SQL. Tento jazyk je rozdělen do bloků. V bloku DECLARE se deklarují proměnné. Blok PROCEDURE deklaruje proměnné pomocí procedury. Blok BEGIN je povinný. Nachází se v něm procedury a SQL příkazy SELECT, INSERT, UPDATE, DELETE. Podmínky jsou v bloku EXCEPTION, který je ukončen END. Tento jazyk je používán v různých balíčcích databáze [9].

3.3.3 Typy statistik

3.3.3.1 Tabulkové statistiky

Tabulkové statistiky optimalizátor využívá k určování nákladů. Náklady se vypočítají z procházení tabulek a spojení tabulek. DBMS_STATS shromažďuje

statistické náklady za trvalé a dočasné tabulky. Databáze sleduje všechny statistiky trvalých tabulek [5].

3.3.3.1 Sloupcové statistiky

Statistiky sloupců poskytují informace o hodnotách sloupců a distribuci dat. Tyto informace optimalizátor využívá k odhadům kardinality. Dále jsou využívány k rozhodování o použití indexů, spojování objednávek, připojení metod. Rozšířené statistiky umožňují optimalizátor informovat o logických vztazích mezi sloupci [5].

3.3.3.1 Indexové statistiky

Statistiky indexů obsahují informace o počtu úrovní indexů, počtu bloků indexů a vztahy mezi indexem a datovými bloky. Optimalizátor využívá statistiky k určení nákladů procházení indexů [5]. Indexové statistiky uložené v DBA_IND_STATISTICS zobrazují následující:

Úrovně – Sloupec BLEVEL zobrazuje počet bloků potřebných k cestě z bloku kořene k bloku listu. Využívá se zde B-Tree index kde se nacházejí bloky pro vyhledávání a bloky listu, které ukládají hodnoty.

Odlišné klíče – Ve sloupci DIST_KEY se sleduje počet odlišných indexovaných hodnot.

3.3.3.1 Statistiky pro globálně dočasné tabulky

Jde o zvláštní tabulku, kde jsou ukládány relace a soukromá data po určitou dobu. ON COMMIT klauzule vytvoří dočasnou tabulku. Dále tato klauzule udává, jestli je tabulka specifická pro transakce (odstranění řádků), nebo pro relaci. Dočasná tabulka obsahuje výsledky po dobu transakce nebo relace.

Při vytvoření tabulky, lze vytvořit definici, která je viditelná pro všechny relace. Ve chvíli kdy relace vkládá data do tabulky, databáze jí přidělí úložný prostor. Data v dočasné tabulce jsou viditelná pouze pro aktuální relaci.

Pomocí tabulky GLOBAL TEMP TABLE_STATS spouštěné v databázi Oracle 12c, mohou být statistiky dočasných tabulek sdílené. Je možné sbírat statistiky dočasné tabulky v jednom sezení a následně použít statistiky pouze pro tuto relaci. Mezitím mohou uživatelé dále udržovat sdílenou verzi statistik.

Optimalizátor nejprve zjistí, zda dočasné tabulky mají statistiky specifické pro relaci. Pokud ano, optimalizátor je použije. V opačném případě použije sdílení statistiky [5].

3.3.3.1 Systémové statistiky

Systémové statistiky popisují hardwarové vlastnosti I/O a výkon procesoru. Statistiky umožňují systému lépe odhadnout I/O a CPU náklady při výběru plánu. Databáze analyzuje všechny nové SQL příkazy pomocí nových statistik. Dále analyzuje aktivitu systému v určitém časovém období (statistika zátěže), nebo simuluje zatížení [5].

3.3.4 Shromažďování statistik

Shromažďování statistik se dělí na automatické a manuální. Při automatickém shromažďování systém automaticky shromažďuje statistické údaje a má naplánovanou údržbu statistik. Ruční shromažďování musí probíhat ve všech schématech databáze, včetně systémových. Ruční shromažďování musí být pravidelné. Automatické shromažďování eliminuje manuální úkony a následné problémy s optimalizací dotazu. Dále snižuje riziko špatného provedení plánu z důvodů zastaralých statistik [7].

3.3.4.1 DBMS_STATS

Tento PL/SQL balíček shromažďuje a spravuje statistické údaje optimalizátoru. Patří do manuálního ovládání statistik. Eliminuje mnoho manuálních úkonů, spojených se správou optimalizátoru. Snižuje tím riziko vzniku neoptimálních exekučních plánů. V základním nastavení databáze Oracle má nastaveno automatické spuštění DBMS_STATS. Shromažďuje statistické údaje optimalizátoru a všech objektů ve schématu [5].

3.3.4.2 Dynamické statistiky

Ve výchozím nastavení, kdy statistiky chybí, jsou nedostatečné. Databáze shromažďuje dynamické statistiky. Vytváří je rekurzivním-opakovaným SQL skenováním malého náhodného vzorku tabulky bloku.

Dynamické statistiky často doplňují statistické údaje. Pomáhají tak zlepšovat plány optimalizátoru [5].

Přínosy dynamických statistik:

- Exekuční plány jsou optimální z důvodu použití složitých predikátů
- Doba shromažďování informací je kratší, než celkové provedení dotazu
- Dotaz je proveden několikrát, takže se čas vzorkování neodepisuje

3.3.4.3 Online statistiky

Databáze shromažďuje statistiky tabulky automaticky, během těchto operací:

- CREATE TABLE AS SELECT
- INSERT INTO ... SELECT

Účelem online statistik, je schopnost shromažďovat statistické údaje automaticky během hromadného zatížení. Shromáždění statistických údajů v průběhu zatížení zabraňuje dalšímu prohledávání tabulek a to zlepšuje výkon. Je vylepšená i správa. Uživatel nemusí zasahovat do statistických údajů.

Při vkládání řádků do prázdných oddílů tabulky jsou oddíly prázdné. Databáze shromažďuje statistické údaje během vložení. Statistiky jsou k dispozici po příkazu INSERT.

Při shromažďování online statistik se neshromažďují statistiky indexů a histogramy. Pokud tyto statistiky požadujeme, použijeme tento příkaz DBMS_STATS.GATHER_TABLE_STATS.

Omezení pro online statistiky:

- nejsou prázdné, provedení INSERT INTO, SELECT
- vnořené tabulky
- index-organizované tabulky
- externí tabulky
- virtuální sloupce
- statistiky jsou zamčené

V online statistikách se dají používat i hinty. Ve výchozím nastavení se shromažďují statistické údaje během hromadného zatížení. Tato funkce se dá zakázat pomocí příkazu NO GATHER OPTIMIZER STATISTICS [5].

3.4 Exekuční plány

Exekuční plán zobrazuje kroky nezbytné k provedení SQL příkazu a tím pomáhá popsat rozhodnutí optimalizátoru. Určuje jak správně provést SQL příkaz. Databáze optimalizuje každý SQL příkaz na základě statistických údajů. Při provádění plánu jsou různé přístupové cesty a připojené metody [1].

Optimalizátor přistupuje k SQL příkazu, kde pomocí datového slovníku získá data k vytvoření plánu provedení. Těchto plánů, může být více. Optimalizátor tyto plány porovná a vybere plán s nejnižšími náklady. To je výsledný exekuční plán.

Je několik rozdílů mezi exekučními plány. Dvě různé databáze mají rozdílná schémata a různí uživatelé mají různé přístupy k tabulkám ve stejné databázi. Dalším rozdílem ve schématu je používání indexů. Náklady mají různé hodnoty podle statistik, objemu dat, typů proměnných a nastavení inicializačních parametrů. Jejich struktura je stromová, stromovou strukturu je možné projít několika způsoby, mezi které patří PREORDER, INORDER a POSTORDER. Při průchodu tabulky jsou důležité dva směry a to shora dolů a zleva doprava. Nejpoužívanějším a nejvíce efektivním je způsob POSTORDER, kde se projde levý list, pak pravý list a pokračuje se ke kořenu. Tímto způsobem se procházejí řádky tabulky [5].

3.4.1 PLAN_TABLE

Tabulka PLAN_TABLE je výstupní tabulkou exekučního plánu. Slouží pro popisování exekučního plánu. Proto jí můžeme nazvat popisující tabulkou. Sloupce tabulky popisují oddíly sloupců a řádky vkládá EXPLAIN PLAN.

Optimalizátor vybere pro SQL příkaz plán, který EXPLAIN PLAN zkoumá. Zkoumaná data následně generuje do popisující tabulky [5].

Výstupní tabulku PLAN_TABLE_OUTPUT vygenerujeme pomocí příkazu EXPLAIN PLAN FOR a SELECT * FROM table(dbms_xplan.display(null, null, 'projection'));. Tím je obalen SQL příkaz.

```
EXPLAIN PLAN FOR
        SELECT      aazakaznici.jmeno, aazakaznici.prijmeni,
                    aafaktura.fakturovanacastka, aazamestnanci.jmeno,
                    aazamestnanci.prijmeni
        FROM        aazakaznici, aafaktura, aazamestnanci
        WHERE       aafaktura.fakturovanacastka < 1500 AND
                    aafaktura.id_zakaznici=aazakaznici.id_zakaznici AND
                    aafaktura.id_zamestnanec=aazamestnanci.id_zamestnanec
        ORDER BY    aazakaznici.prijmeni, aazakaznici.jmeno
SELECT * FROM table(dbms_xplan.display(null, null, 'projection'));
```

Výsledkem ukázky je samotná výstupní popisující tabulka PLAN_TABLE_OUTPUT na obrázku 4.

```
-------------------------------------------------------------------------------------------
| Id  | Operation            | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                |   300 | 15600 |    14   (0)| 00:00:01 |
|   1 |  SORT ORDER BY       |                |   300 | 15600 |    14   (0)| 00:00:01 |
|*  2 |   HASH JOIN          |                |   300 | 15600 |    14   (0)| 00:00:01 |
|*  3 |    HASH JOIN         |                |   300 |  9900 |     8   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| AAZAMESTNANCI  |   199 |  3980 |     3   (0)| 00:00:01 |
|*  5 |     TABLE ACCESS FULL| AAFAKTURA      |   300 |  3900 |     5   (0)| 00:00:01 |
|   6 |    TABLE ACCESS FULL | AAZAKAZNICI    |  1402 | 26638 |     6   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Obrázek 4 Ukázka PLAN_TABLE_OUTPUT. Zdroj: Autor

Dále bude vysvětleno, co jednotlivé části tabulky znamenají a jak se generují jejich hodnoty.

ID - Číslo přiřazené každému kroku v exekučním plán.

Operation - Název prováděné operace.

Name - Název tabulky, kde se daná operace provádí.

OBJECT_NAME – Název tabulky (uváděno v grafických nebo tabulkových exekučních plánech).

Rows - Očekávaný počet řádků prováděním této operace.

Bytes – Očekávaný počet bajtů. Vypočítává se pomocí řádků a velikosti řádků.

Cost (%CPU) - Očekávané náklady pro jedno provedení této operace včetně jejich potomků. Ve sloupci cost je hodnota nákladů přečtených informací I/O operace.

Hodnoty ve sloupci (%CPU) jsou náklady úměrné systémovým cyklům potřebných pro provoz.

Time - Uplynulý čas hodiny:minuty:sekundy, potřebné k provádění operace. V čase jsou zahrnuty i potomci metody. Čas je obvykle zaokrouhlován. Operace může být provedena za 6 milisekund a tento čas je zaokrouhlen na 1 sekundu.

Kardinalita - Odhadovaný počet řádků vracených každou operací provádění. Optimalizátor určuje kardinalitu na základě komplexního souboru vzorců. Optimalizátor rovnoměrně rozdělí a vypočítává kardinalitu pro dotaz vydělením celkového počtu řádků [5].

3.4.2 Pohledy

V některých případech se může stát, že i plán, který je vybrán, není úplně efektivní. Zde je potřeba zvážit jestli je opravdu nutné využívat indexy anebo místo nich využít k ušetření pohledy.

Tabulka 2 Pohledy a jejich popis. Zdroj: [10]

Pohled	Popis
V$SQL_PLAN	Tento pohled obsahuje plán provádění pro každé prohlášení uložené v SQL oblasti. Výhodou využití tohoto pohledu na rozdíl od použití EXPLAIN je, že není potřeba znát prostředí kompilace.
V$SQL_PLAN_STATISTIC	Poskytuje skutečné statistické spuštění pro každou operaci plánu jako je uplynulý čas, výstupní řádky.
V$SQL_PLAN_STATISTIC_ALL	Tento plán umožňuje srovnání vedle sebe odhadů, které optimalizátor poskytuje pro počet řádků a uplynulý čas. Zde kombinuje V$SQL_PLAN a zároveň V$SQL_PLAN_STATISTIC.

3.4.3 SQL plan management a novinky v Oracle 12C

SQL plan management zajišťuje, že výkon nedegraduje kvůli změně plánu provádění. Nový plán je sledován a vyhodnocován až ve chvíli, kdy dokazuje znatelné zlepšení běhu [10].

1. **Získání plánu:**

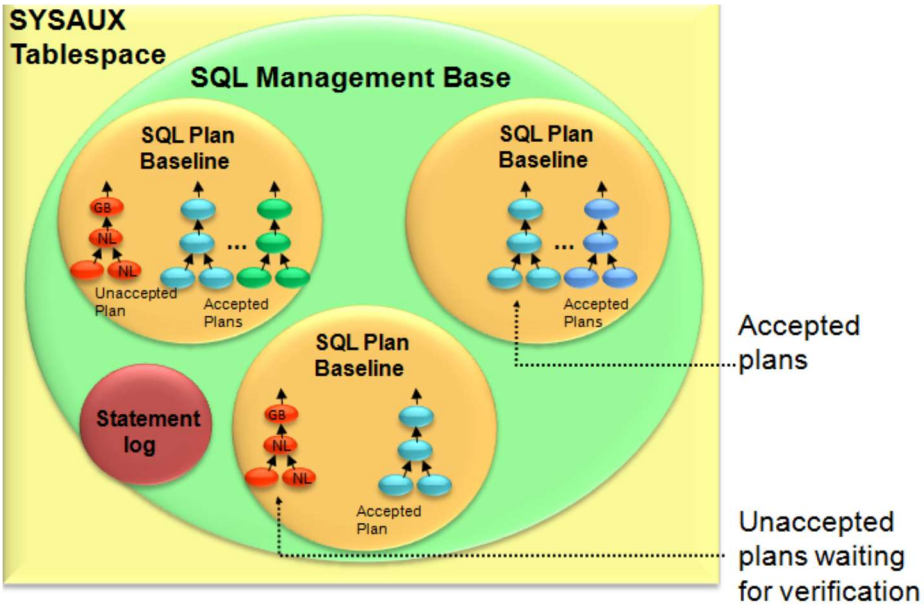
   Všechny uznané exekuční plány pro příslušné SQL příkazy jsou ukládané. Plány jsou uložené do SYSAUX do tabulkového prostoru.

2. **Plán selekce:**

   Zajišťuje provádění pouze přijatých exekučních plánů. Zaznamenává všechny nové exekuční plány.

3. **Plán vývoje:**

   Vyhodnocení všech nepřijatých exekučních plánů pro daný příkaz. Plány, které vedou ke zlepšení výkonu, jsou uznané.

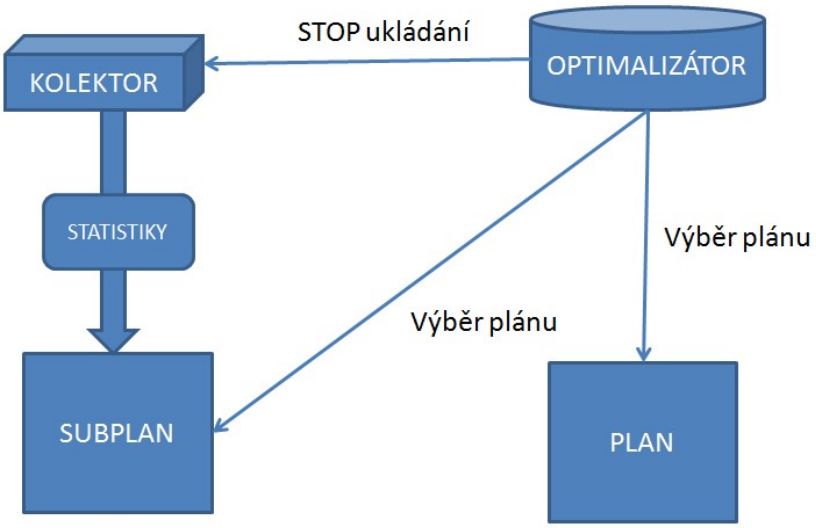


Obrázek 5 SQL plán management. Zdroj: [10]

3.4.4 SQL plan baselines & Adaptive Plans

Adaptivní plány byly představeny v databázi Oracle 12c. Umožňují optimalizátoru odložit rozhodnutí o konečném plánu až do doby provedení. Nástroje optimalizátoru vyberou plán, jehož hodnoty zjistí při běhu. Při běhu kolektor shromažďuje statistiky, které vkládá do vyrovnávající paměti do řádku. Statistiky tak vytváří SUBPLAN. Potom co optimalizátor zvolí plán, kolektor statistik

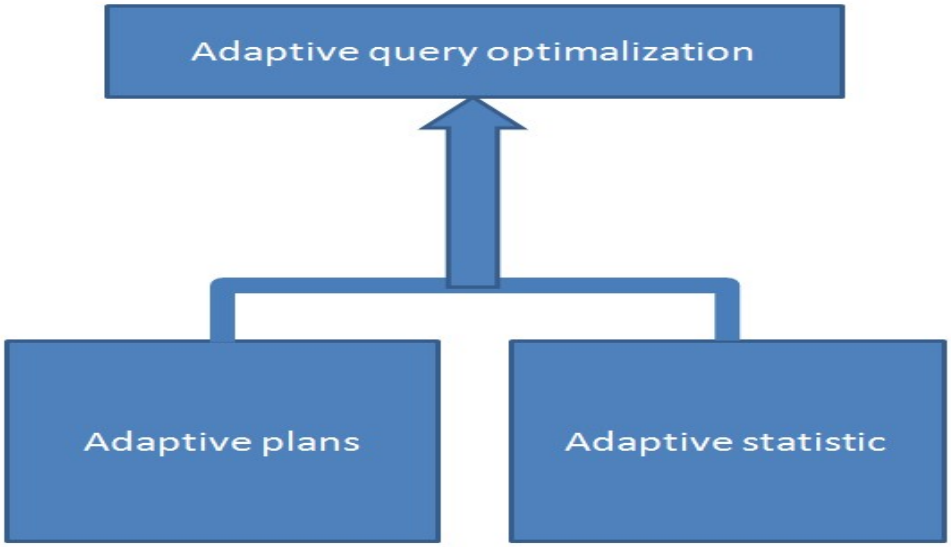
zastaví ukládání informací. Optimalizátor zakáže ukládání do vyrovnávací paměti a zvolí se vybraný konečný plán. Pokud se odhadované a reálné hodnoty liší, tak se plán nebo jeho části upravují. Tím se zabrání neoptimálnímu výkonu SQL příkazu [11].



Obrázek 6 SQL plan baselines. Zdroj: Autor

3.4.5 Adaptive query optimalization

Další příjemnou změnou, kterou přinesla databáze Oracle 12c je Adaptive query optimalization. Adaptivní optimalizace dotazu je sada funkcí, která umožňuje optimalizátoru za běhu upravovat plány a objevovat další informace, které vedou k lepším statistikám. V předešlých verzích Oracle databáze byly statistiky dodatečné k vytvoření optimálního plánu. Teď jsou jeho součástí. Adaptive query optimalization přispívá ke zlepšování dotazů a adaptivních statistik [10].



Obrázek 7 Adaptive query optimalization. Zdroj: Autor

3.5 Oracle Database 12c

Oracle database 12c je objektově relační databázový systém. Jak písmeno C naznačuje, umožňuje rychlou konsolidaci mnoha databází a správu v rámci cloudové služby. Mezi další inovace patří nové úrovně efektivity, výkonu, zabezpečení a dostupnosti.

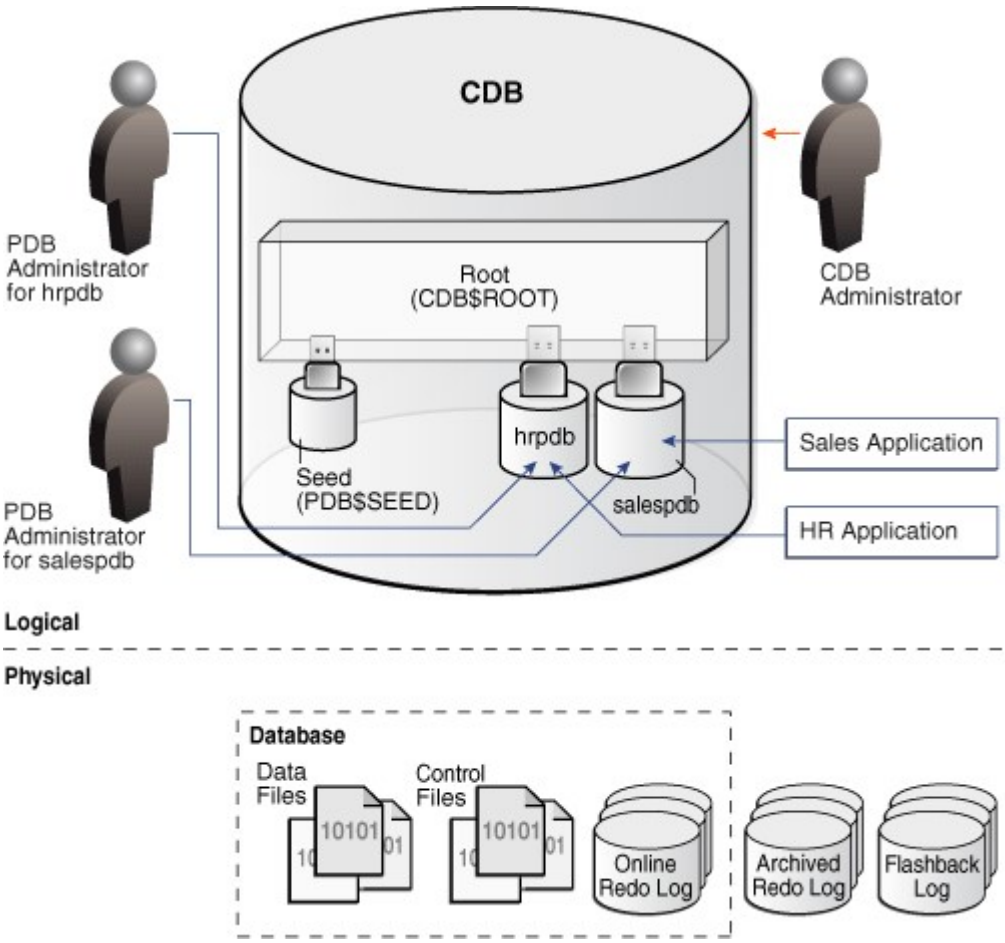
Databáze využívá MULTITENANT architekturu. Což znamená, že v databázi (CDB), je umožněné vytvářet kontejnery (PDB). Tato možnost nebyla v předešlých verzích možná.

Velké podniky používají až tisíce databází, které běží na různých platformách a serverech. Dnes je úroveň serverů mnohem lepší a zvládnou vyšší zatížení. Tím databáze využívají zlomek serverové a hardwarové kapacity. Proto je možné sjednocení více databází do jednoho počítače.

V databázi je možné vytvářet několik kontejnerů, které mají originální název – identifikátor. Každý z kontejnerů má vlastní datový slovník.

Databáze má hlavní kontejner CDB$ROOT, který ukládá systémová metadata do podřazených kontejnerů. Ten spravuje správce databáze.

Všechny kontejnery se dají snadno připojit a odpojit. To ovlivňuje využití I/O a CPU.



Obrázek 8 Kontejnery v databázi. Zdroj: [12]

Na obrázku 8, je vidět databáze obsahující hlavní kontejner CDB$ROOT, který ukládá systémová metadata, potřebné pro správu podřazený kontejnerů. ROOT kontejner spravuje správce databáze. Ke každému kontejneru může přistupovat správce databáze, nebo přímo správce samotného kontejneru.

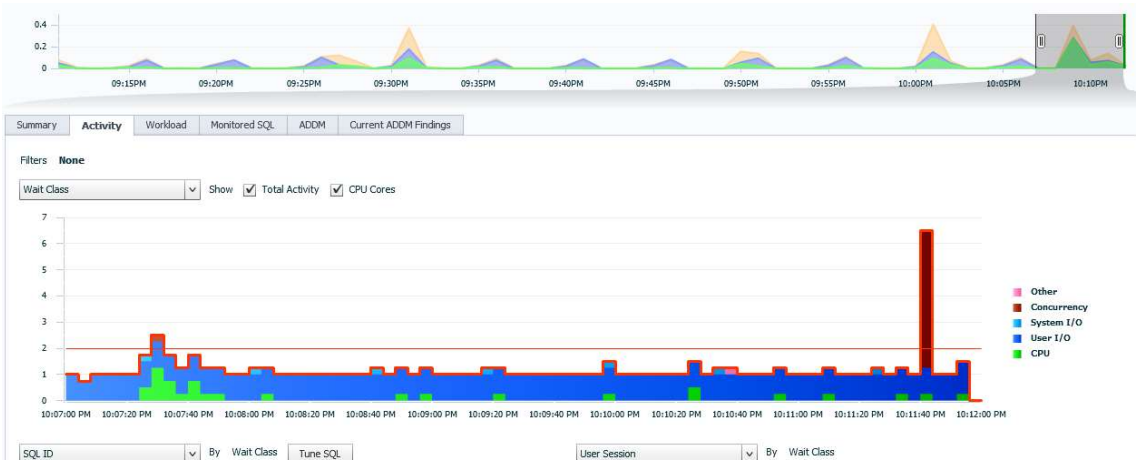
S Oracle Database 12c, jsou propojené další produkty pro správu SQLPlus, SQL Developer a Oracle Enterprise Manager Database Express 12c.
SQLPlus je nejzákladnějším nástrojem pro ovládání databáze Oracle. Databáze je tímto nástrojem ovládána pomocí příkazů v příkazovém řádku.

Oracle SQL Developer, pomocí grafického rozhraní, umožňuje uživateli spouštět příkazy, ladit a testovat [12].

3.5.1 Oracle Enterprise Manager Database Express 12c – OEMDE12C

Je webový nástroj pro správu databáze. Poskytuje podporu pro základní administrativní úkony, poskytuje komplexní řešení diagnostiky, ladění a výkonosti. Tento webový nástroj např. vytváří kontejnery a dále s nimi pracuje. Celkově práci

a administraci databáze usnadňuje. I každý nový vytvořený kontejner má vlastní OEMDE12C webovou aplikaci.



Obrázek 9 PERFORMANCE HUB. Zdroj: Autor

Dalším kladem OEMDE12C, je sledování vytížení databáze. Na obrázku, je zobrazeno okno PERFORMANCE HUB. V horní části se nachází časová osa, která zobrazuje vytížení I/O (fialová), CPU (zelená), ostatní (oranžová). PERFORMANCE HUB umožňuje sledování různých parametrů v rozmanitém grafickém zpracování [13].

3.5.1.1 SQL Tuning Advisor

Je automatické ladění provádějící se nástrojem serveru SQL Tuning Advisor. Na vstupu vyvolává optimalizaci SQL příkazu a na výstupu jsou doporučení s odůvodněním a přínosem [5].

3.5.2 ENTERPRISE MANAGER CLOUDE CONTROL 13C - EMCC13C

Další možností spravování databáze je novinka EMCC13C. Tento produkt lze dodatečně stáhnout na stránkách Oracle. Poskytuje kompletní řízení a správu podniku v cloudu. Databázi lze real-time[5] integrovat. Vylepšené je sledování statistik a jejich správa. Další možnosti jsou v testování a sledování databáze. Lze si vytvořit úlohy (SQL příkazy), které se následně testují a upravují [14].

[5] Real–time – skutečný čas.

4 Praktická část

Praktická část bakalářské práce bude zaměřena na teoretické dovednosti, které budou aplikované na vzorovou databázi. Po představení a popsání databáze budou přiblížena různá doporučení pro práci s programy, ve kterých bude databáze vytvářena a následně testována. Další v pořadí jsou statistiky, které musí být aktualizované. Bude popsána možnost jejich sledování v rámci OEMDE12C. Samotné testování je rozděleno na tři části, které probíhají v jedné databázi s kontejnery. Každá část se provádí v samostatném kontejneru, který byl pro ni vytvořen. V první části je optimalizovaný SQL příkaz pomocí SQL Tuningu. Druhá část se zaměřuje na použití optimalizovaného příkazu v dalším kontejneru databáze s identickými daty. Třetí část se soustředí na použití hintů a jejich vliv na exekuční plány.

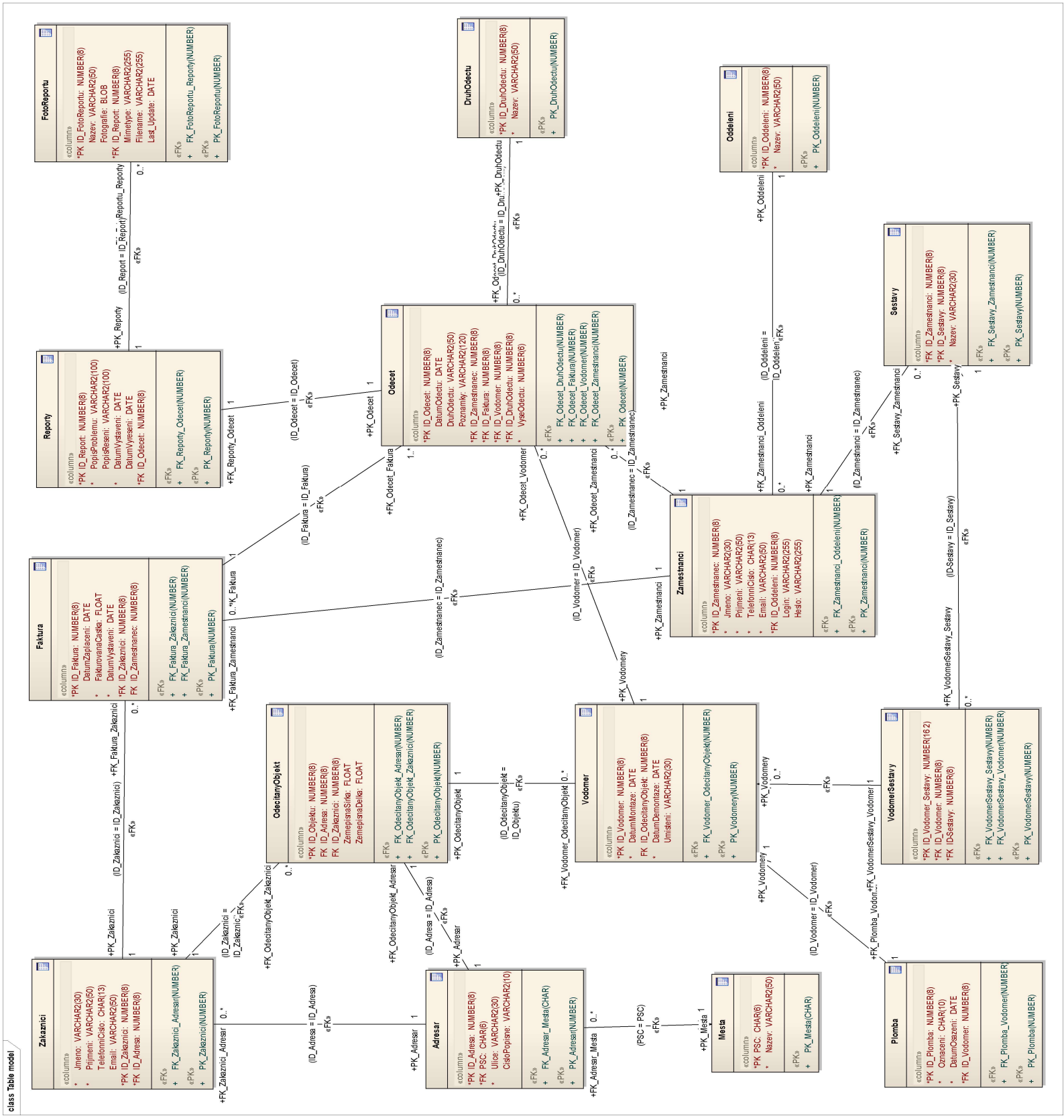
4.1 Popsání databáze

Pro ukázkové účely byla zvolena databáze aplikace pro odečty vodoměrů. Aplikace slouží jak do terénu samotným odečítačům, tak i pracovníkům spravujícím faktury pro vlastníky. Do databáze se zaznamenává adresa vodoměru, kde se provádí odečet. Adresa vlastníka je zároveň fakturační adresou. Plomba vodoměru je úřední ověření, kde se zaznamenává číslo plomby, datum osazení, přesná lokace vodoměru, kontakt na majitele a poznámky. Fakturace probíhá po každém odečtu, o jednotlivých fakturách se uchovávají informace o posledním období, ceně za kubík, stavu faktury apod. Poslední stav vodoměru je nejnovějším stavem odečtu. V databázi se vyskytuje patnáct tabulek, ve kterých je přibližně dvacet tisíc záznamů. Struktura ukázkové databáze byla vytvořena v rámci předmětu DBS2 studenty Barborou Spěvákovou, Kristýnou Kamenickou a Karlem Laštůvkou. Studenti souhlasili s použitím struktury databáze.

4.1.1 Složitosti databáze

Databáze je využívána při plánování tras. Z tohoto důvodu musí být nalezeny všechny adresy vodoměrů v určité lokalitě. Dané lokality se určují podle poštovního směrovacího čísla. Vytvořená aplikace následně určí trasu. Při vytváření faktury je potřeba vyhledat majitele s jeho fakturační adresou a adresami vodoměru.

4.1.2 Database model



Obrázek 10 Database model. Zdroj: Spěváková, Kamenická, Laštůvka

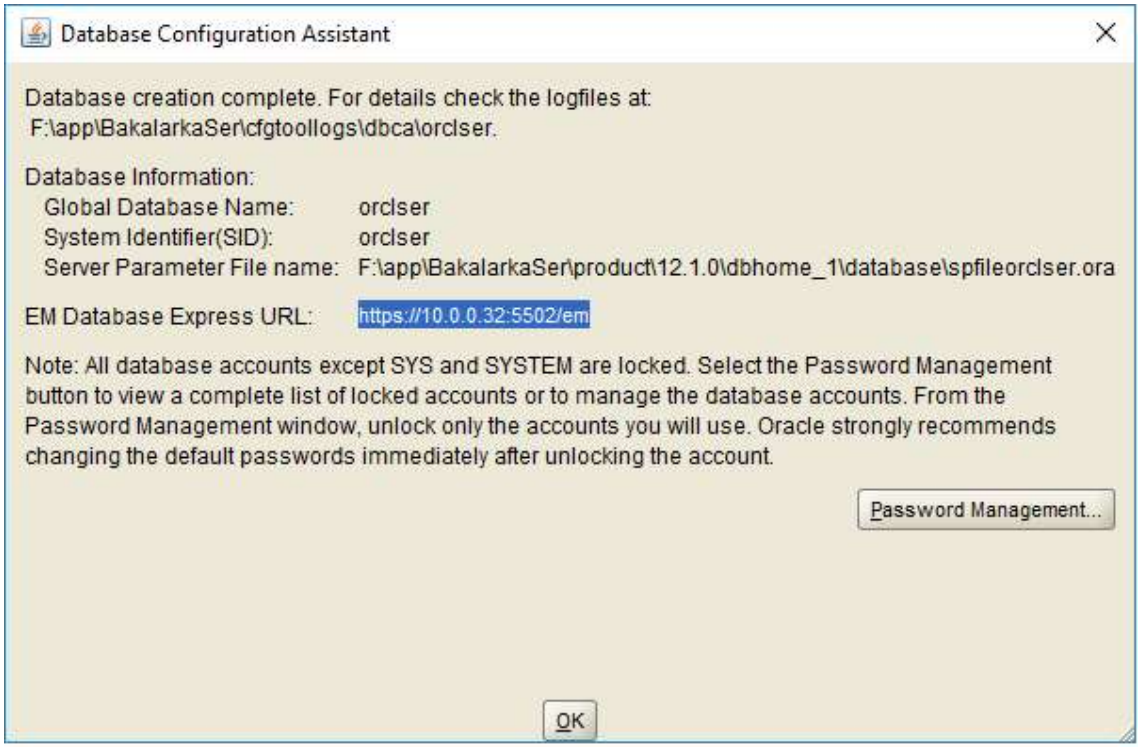
4.2 Práce s programy Oracle

Ještě před instalací je dobré zkontrolovat a nastavit zařízení, se kterým se bude pracovat. Základem je znát jeho parametry, podle kterých je pak instalován vhodný program. Uživatel tak předejde řadě problémů. Všechny instalované programy mají popsané své požadavky na stránce www.oracle.com. Je důležité projekt zálohovat a zaznamenávat si různé informace o databázi. S přesnějšími zápisky je následná práce mnohem jednoduší.

Instalace je doprovázená nápovědami. Chybová hlášení lze najít kdekoli na internetu. K dispozici jsou různé manuály a dokumentace přímo od Oracle. Pokud problém přetrvává, je možné se obrátit přímo na Oracle Help Center.

4.2.1 Oracle Database 12c – OD12C

Na konci instalace se automaticky spustí konečné okno, kde je možné se dostat k oknu konfiguračních asistentů.



Obrázek 11 Konečné okno instalace Oracle Database 12c. Zdroj: Autor

Okno nabízí spravování hesel uživatelských účtů. Je důležité nastavit hesla u SYS a SYSTEM, popřípadě u další zvolených uživatelů. Ty se využívají k přihlášení do databáze. Po kliknutí na tlačítko OK se pokračuje v instalaci. Uživatele a hesla se pak nemusí nastavovat pomocí příkazů v programu SQLPlus.

Součástí konečného okna je také URL adresa OEMDE12C, která umožňuje webový přístup do spravující aplikace databáze.

Po úspěšné instalaci je potřeba upravit dokument tnsname.ORA, kde je přidán kontejner. Při instalaci kontejneru nebyl kontejner přidán do souboru. To usnadní přístup k databázi.

Příklad vytvořeného kontejneru:

```
PDBORCLBP =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = pdborclBP)
    )
  )
```

Tento konfigurační dokument, je vhodné využít při spojení s databází, zejména položky jako protocol, host, port a service name. Kontejner PDBORCLBP je instancí databáze, ke které je potřeba se připojit a pracovat s ní.

Při přihlašování pomocí SID s databází někdy nelze navázat spojení. Při připojování pomocí kontejneru, přes servise name, je přihlašování úspěšnější. V některých případech i zadání kontejneru do SID.

4.2.2 Nastavení služeb – services

Všechny služby patřící k databázi a aplikacím mají název začínající Oracle. Stačí vyhledat všechny služby, které začínají Oracle. Služby mají dva stavy spuštěno, spouštěno. Služby lze spustit, restartovat, zastavit, aktualizovat.

V následující tabulce 3 je doporučené nastavení služeb pro fungování databázové aplikace.

Tabulka 3 Seznam spouštěných služeb. Zdroj: Autor

Název	Stav	Typ spouštění	Účet pro přihlášení
OracleJobScheduler ORCLNAZEV		Zakázáno	Název databáze
OracleOraDB12HomeMTSRecoveryService	Spuštěno	Automaticky	Název databáze
OracleOraDB12HomeTNSListener	Spuštěno	Automaticky	Název databáze
OracleRemExecServiceV2	Spuštěno	Ručně	Local System
OracleServiceORCL	Spuštěno	Automaticky	Název databáze
OracleVssWriterORCL	Spuštěno	Automaticky	Název databáze

V tomto složení by měla databázová aplikace fungovat. Při stavu spouštění vše nefunguje. Proto je potřeba počkat na stav spuštěno, nebo službu aktualizovat. Každá akce se službou je různě časově náročná. Jde i o druh služby.

4.2.3 Nastavení httpsportu kontejneru pomocí SQLPLUS

Pro samotnou databázi byl httpsport vytvořen při instalaci. Pro práci s instancí databáze, je potřeba kontejneru samostatně přidat httpsport. Nejvhodnějším nástrojem pro nastavení je SQLPlus. Následuje nastínění krátkého postupu nastavení httpsportu:

1) Příkaz: SELECT con_id, name, open_mode FROM v$pdbs;

   Výsledek:

```
CON_ID       NAME                           OPEN_MODE
---------- ------------------------------ ---------- ----------
         2        PDB$SEED                       READ ONLY
         3        PDBORCLBP                      READ WRITE
         4        KONTEJNER1                     READ WRITE
         5        KONTEJNER2                     READ WRITE
         6        KONTEJNER3                     READ WRITE
```

Po provedení tohoto příkazu se vypíší kontejnery databáze, společně s id, názvem kontejneru, modem.

2) Příkaz: ALTER session SET container=kontejner1;

   Výsledek: Session altered.

   Pro práci bude otevřen kontejner1.

3) Příkaz: SHOW *con_name*;

   Výsledek:

```
CON_NAME
------------------------------
KONTEJNER1
```

   Ověření otevření kontejneru1.

4) Příkaz: SELECT dbms_xdb_config.gethttpsport from dual;

   Výsledek:

```
GETHTTPSPORT
------------
           0
```

   Zjištění přiřazeného httpsportu. Pokud je jeho hodnota nula, httpsport musí být nastaven.

5) Příkaz: execute dbms_xdb_config.sethttpsport(5504);

6) Výsledek:

```
GETHTTPSPORT
------------
        5504
```

   Příkaz nastavil httpsport na hodnotu 5504. Pro ověření se opakuje krok 4.

Httpsport byl úspěšně nastaven. Dále je použit v odkazu na OEMDE12c.
Příklad: https://localhost:5504/em/

Pomocí webové aplikace OEMDE12C lze pracovat s instancí databáze, kontejnerem KONTEJNER1.

4.2.4 Oracle Enterprise Manager Database Express 12c - OEMDE12C

K další práci s databází bude zvolen libovolný program. Uživatelsky příjemným řešením je OEMDE12C. Pomocí této webové aplikace uživatel pohodlně a přehledně spravuje vytvořenou databázi. Hned na úvodní stránce jsou zobrazené všechny parametry databáze. Graficky je zobrazen výkon, zdroje, relace nebo paměť. Pozadu nejsou ani SQL příkazy, kde se nachází trvání, typ, id, uživatele, čas a samotný SQL příkaz.

Při další práci s databází mohou nastat chyby, nebo dotazy na změny parametrů databáze. Většinou tato situace nastává při přidávání doplňků, nebo instalací dalších produktů pro správu databáze. Parametry se dají měnit pomocí SQL příkazů, nebo v OEMDE12C.

Při přenastavení parametrů pomocí SQLPlus databáze nezaznamená změnu ani po dalším SQL dotazu na změněný parametr, zdali je změněný. Vzhledem k těmto zkušenostem, je vhodné pracovat s OEMDE12C. V konfiguraci se zvolí inicializace parametrů. Zde se nachází kategorie, ve kterých jsou parametry databáze. Nalezený požadovaný parametr se změní na požadovanou hodnotu. Změny se provedou a uloží v rámci několika sekund.

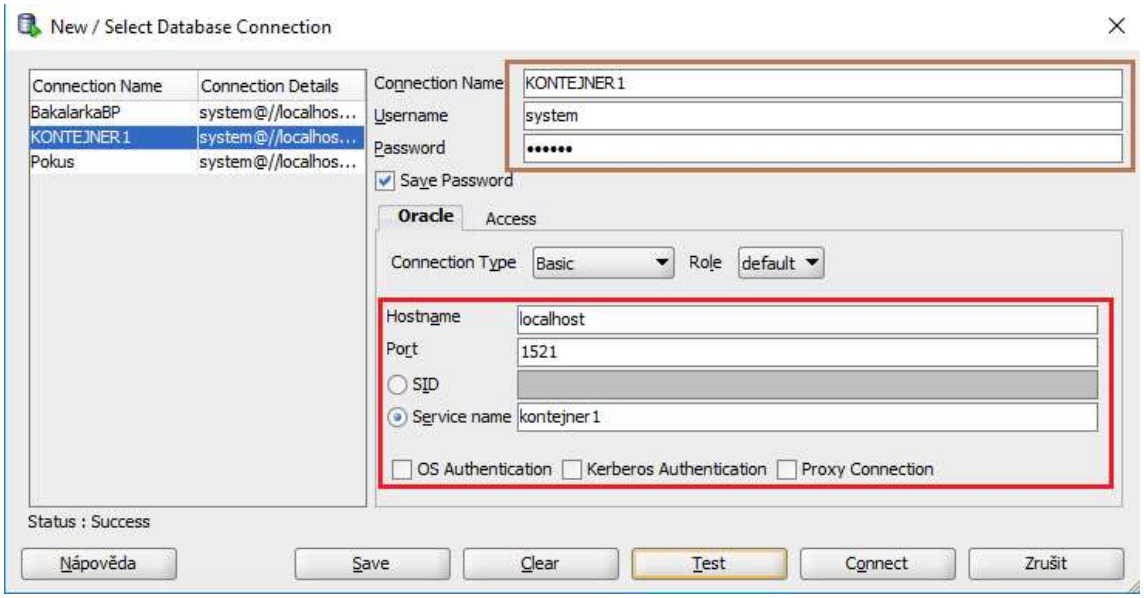
Aplikace je rozdělená pomocí httpsportů. Po instalaci je databázi přidělen httpsport 5501. Pomocí odkazu se spravuje celá databáze. V této hlavní části, je možnost spravovat kontejnery databáze. Kontejnery lze vytvářet, odebírat, připojovat, odhlašovat. Pro spravování samostatného kontejneru je potřeba vytvořit vlastní httpsport. Tím je vytvořena jeho vlastní adresa, pomocí které je možné se připojit k jeho spravování.

4.2.5 Oracle SQL Developer - OSD

OSD je vývojové prostředí usnadňující správu a vývoj Oracle Database. Společně s databází je připravená (instalovaná) verze 2013. Velkou výhodou je propojení všech produktů databáze. Všechny SQL příkazy, exekuční plány a statistiky můžeme sledovat ve webové aplikaci OEMDE12C.

4.2.5.1 Připojení k databázi

Při vytváření nového připojení, se zobrazí přihlašovací okno. Zde se využijí předešlé informace, které byly získány při a po instalaci OD12C.



Obrázek 12 Database Connection. Zdroj: Autor

Na obrázku 12, je zobrazeno připojení k instanci databáze, kontejneru KONTEJNER1. V horní části, je zvolen libovolný název připojení. K databázi se lze připojit pomocí vytvořeného uživatele SYSTEM. V části 4.2.1 OD12C, je nastavován soubor tnsname.ORA. Do souboru je přidáván vytvořený kontejner. V příkladu byl uveden hostname, port, service name. Tyto informace lze využít k připojení. Kliknutím na tlačítko test, program zjistí úspěšnost připojení. Pokud bude status success-úspěšně připojený, pokračuje se tlačítkem connect. Připojení je úspěšné a pokračuje se k samotné práci s databází.

4.3 Statistiky

Prvním krokem, kterým je potřeba začít, je aktualizace statistik. Optimalizátor musí mít aktuální statistiky, aby odváděl adekvátní práci. K aktualizaci statistik je vhodné využít OSD. Ten pomocí balíčku DBMS_STATS, statistiky aktualizuje.

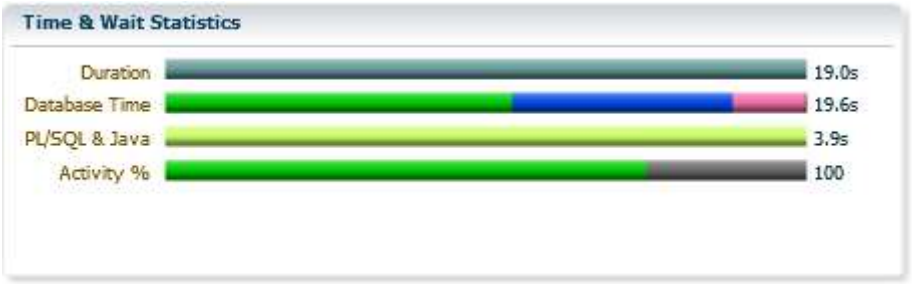
```
BEGIN
DBMS_STATS.GATHER_SCHEMA_STATS ( ownname => 'SYSTEM', estimate_percent => 100);
 END;
```

V aplikaci OEMDE12C, je možná kontrola a stav statistik databáze. V horní části menu se objevuje PERFORMANCE a dál PERFORMANCE HUB. Na stránce je potřeba zvolit MONITORED SQL. Zde je SQL příkaz, který uživatele odkáže na

samotné monitorování statistik. Pomocí grafů jsou zobrazeny různé parametry statistik v návaznosti s časem.

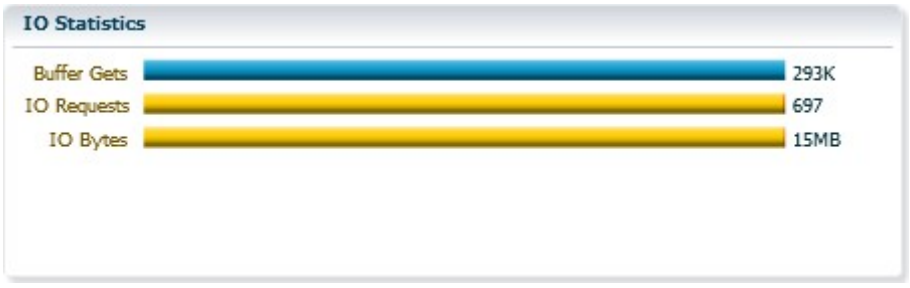
Obrázek 13 MONITORED SQL. Zdroj: Autor

1) V GENERAL jsou veškeré informace o textu SQL, id označení a uživatel, který statistiky spustil.



Obrázek 14 Průběh statistik. Zdroj: Autor

2) Čas a průběh statistik zobrazuje trvání, databázový čas, PL/SQL a JAVA, aktivity. Databázový čas je složen z času CPU (zelená 54%), I/O(modrá 34%) a ostatní (růžová 11%).



Obrázek 15 IO statistiky. Zdroj: Autor

3) V IO statistikách je zobrazena velikost vyrovnávací paměti, IO žádosti a IO bytes.

Obrázek 16 Detail. Zdroj: Autor

4) Část detail má dva stavy zobrazení aktivity a metrik. V aktivitách jsou různobarevně zobrazeny SQL příkazy v závislosti na čase. V metrikách jsou zobrazeny grafy CPU využití (zelená), IO propustnost (žlutá), paměť a IO žádosti.

4.4 Testování

Testování je rozděleno na tři části. V první části je optimalizován SQL příkaz, pomocí SQL Tuningu. Příkaz byl vytvořen právě k testování a záměrně není dobře napsaný. V rámci testování 1, si příkaz projde všemi postupy rozhodování optimalizátoru. Cílem je, vytvoření co nejvíce optimalizovaného příkazu, pomocí statistik, exekučních plánů a SQL Tuningu. K testování byl vytvořen samostatný kontejner TEST, kde optimalizace probíhá. V kontejneru je vytvořená vzorová databáze. V rámci testování byly zaznamenávané časy, které budou dále rozebrány ve výsledcích. Časy jsou zaznamenané po prvním spuštění. Jejich spolehlivost není přesná, v závislosti na různých faktorech. Například čas SQL Tuningu v OSD a čas v SQL Tuning Advisor v OEMDE12C, jsou rozdílné. Proto tyto výsledky nebudou brány jako stěžejní, ale ukázkové.

V druhé části bude použit, již optimalizovaný SQL příkaz, v nově vytvořeném kontejneru TEST2. Tento kontejner má identicky vytvořenou databázi, jako je

v první části. Podobně budou zaznamenány výsledné časy. Účelem druhé části, je aplikace a fungování optimalizovaného SQL příkazu a jeho vliv na čas.

Ve třetí části bude přiblížená práce s HINTY a sledování jejich dopadu na exekuční plány. V rámci třetí části budou opět sledované časy dále zmíněné ve výsledcích.

4.4.1 Optimalizace SQL příkazu

SQL příkaz byl vytvořen pro ukázkové účely v rámci databáze. Příkaz projde všemi komponentami optimalizátoru. Příkaz pracuje nepřesně a není kompletní. Pomocí SQL Tuningu bude vylepšován. Postupná cesta k dokonalému příkazu je dokumentována se všemi postupy a během testování jsou zaznamenávány časy.

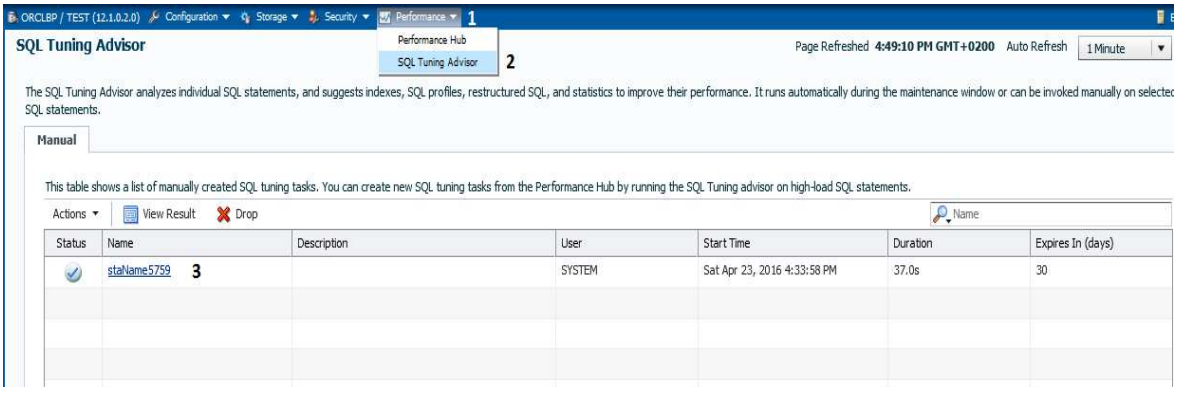
Příkaz:

```
SELECT  aafaktura.datumvystaveni, aafaktura.fakturovanacastka, aazakaznici.jmeno,
aazakaznici.prijmeni, aaadresar.ulice, aaadresar.cislopopisne, aaadresar.psc, aamesta.nazev,
aaodecet.datumodectu, aadruodectu.nazev, aazamestnanci.jmeno, aazamestnanci.prijmeni
FROM aazakaznici, aafaktura, aazamestnanci, aaadresar, aamesta, aaodecet, aadruodectu
WHERE aafaktura.datumvystaveni BETWEEN '01.01.14'
AND '31.12.14'
AND aafaktura.fakturovanacastka>=9000
AND aazamestnanci.prijmeni='Hyde';
```

Příkaz prochází sedm tabulek a hledá faktury konkrétního zaměstnance Hyde. Časové rozmezí faktur je mezi 1. 1. 2014 až do 31. 12. 2014. Hledají se faktury nad 9 000 Kč.

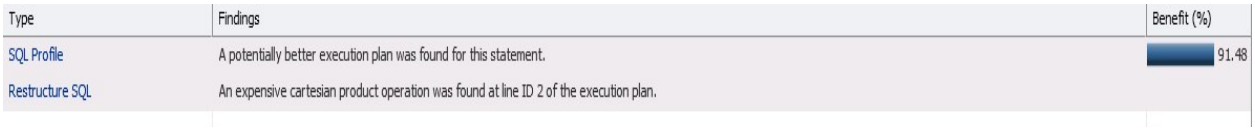
4.4.1.1 První spuštění

Prvním krokem bude spuštění SQL Tuningu v OSD.



Obrázek 17 1-P, 2-STA, 3-Úkoly. Zdroj: Autor

Další zkoumání probíhá v OEMDE12C. Ten je umístěn v kartě PERFORMANCE-P, SQL Tuning Advisor-STA. V tabulce MANUAL jsou ručně vytvořené tuningové úkoly, např. spuštěné v OSD. Úkoly jsou seřazené časově, od nejaktuálnějších (umístěné nahoře), až po ty nejstarší (umístěné dole). U každého úkolu je zaznamenaný status, název úkolu, popis úkolu. Uživatel, který tuning spustil, čas startu, doba provádění v sekundách a čas vypršení ve dnech.



Type	Findings	Benefit (%)
SQL Profile	A potentially better execution plan was found for this statement.	91.48
Restructure SQL	An expensive cartesian product operation was found at line ID 2 of the execution plan.	

Obrázek 18 SQL detaily. Zdroj: Autor

Po rozklinutí úkolu, se zobrazí jeho stránka. Na stránce jsou zobrazené SQL detaily, kde se nachází název úkolu, majitel úkolu, id SQL, schéma, název daného kontejneru. Dále je SQL text, kde je zkoumaný SQL příkaz. Daleko zajímavější je tabulka doporučení. V tabulce se může nacházet více doporučení jako na obrázku 18. Jen jedno doporučení by mělo být implementováno. SQL Tuning upozorňuje na dva lepší exekuční plány. Dále upozorňuje na lepší provedení za pomocí indexů. Celkový benefit z těchto vylepšení, by měl být 91,48%.

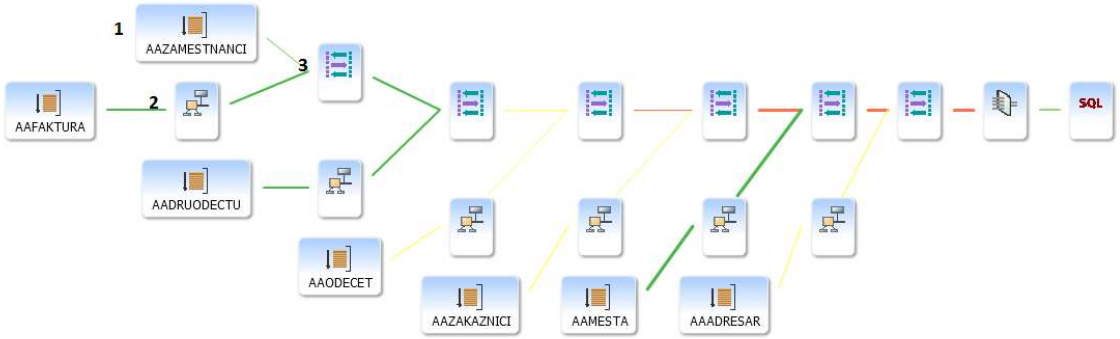
Exekuční plán:

SELECT STATEMENT		0			2,884T	262P
FILTER		1				
MERGE JOIN CARTESIAN		2			2,884T	262P
MERGE JOIN CARTESIAN		3			1,924G	131T
MERGE JOIN CARTESIAN		4			5,884M	335G
MERGE JOIN CARTESIAN		5			4,197K	176M
MERGE JOIN CARTESIAN		6			1,810	62K
MERGE JOIN CARTESIAN		7			37	999
TABLE ACCESS FULL	AAZAMESTNANCI	8		3	1	15
BUFFER SORT		9			34	408
TABLE ACCESS FULL	AAFAKTURA	10		5	34	408
BUFFER SORT		11		15	49	343
TABLE ACCESS FULL	AADRUODECTU	12			49	343
BUFFER SORT		13		15K	2,319	19K
TABLE ACCESS FULL	AAODECET	14		8	2,319	19K
BUFFER SORT		15		17M	1,402	21K
TABLE ACCESS FULL	AAZAKAZNICI	16		4	1,402	21K
BUFFER SORT		17		3,215M	327	3,597
TABLE ACCESS FULL	AAMESTA	18		1	327	3,597
BUFFER SORT		19		3,669G	1,499	34K
TABLE ACCESS FULL	AAADRESAR	20		2	1,499	34K

Obrázek 19 První exekuční plán. Zdroj: Autor

Na obrázku 19, je zobrazen originální exekuční plán. Exekuční plány byly představené v teoretické části, proto ho není potřeba zdlouhavě popisovat. Zajímavé je jeho grafické zpracování. Hodnoty nereprezentují pouze čísla, ale jsou graficky zpracované. Například OPERATION COST, hodnotu reprezentuje číslo, ale grafická část je dále rozdělená na I/O a CPU. I/O je zobrazeno modrou barvou a po najetí myší se zobrazí přesná hodnota pro I/O. CPU je zobrazeno zeleně. To umožňuje pouze OEMDE12C. V operacích je zobrazeno, jak je ke každé tabulce přistupováno přes TABLE ACCESS FULL. To je zdlouhavé a je potřeba nalézt vhodnější řešení.

Grafický exekuční plán:



Obrázek 20 První grafický exekuční plán. Zdroj: Autor

Na obrázku 20, je grafické znázornění aktuálního exekučního plánu.

1. U tabulky AAZAMESTNANCI dochází k úplnému prohledávání tabulky pomocí TABLE ACCESS FULL. Jak je na obrázku 20 vidět, tímto způsobem jsou prohledávány i ostatní tabulky příkazu.
2. Další je seřazení BUFFER SORT.
3. Tabulky jsou spojené MERGE JOIN CARTESIAN.

Jak je na příkladu vidět, tabulky se prohledávají celkově. To je zdlouhavé a nepřesné. Dalším problémem je kartézský součin. Který vykazuje hodně dat a má vysoké náklady. Dalším krokem je doporučená implementace. Když SQL Tuning možnost implementace nabízí, je vhodné jí implementovat.

4.4.1.2 Druhé spuštění

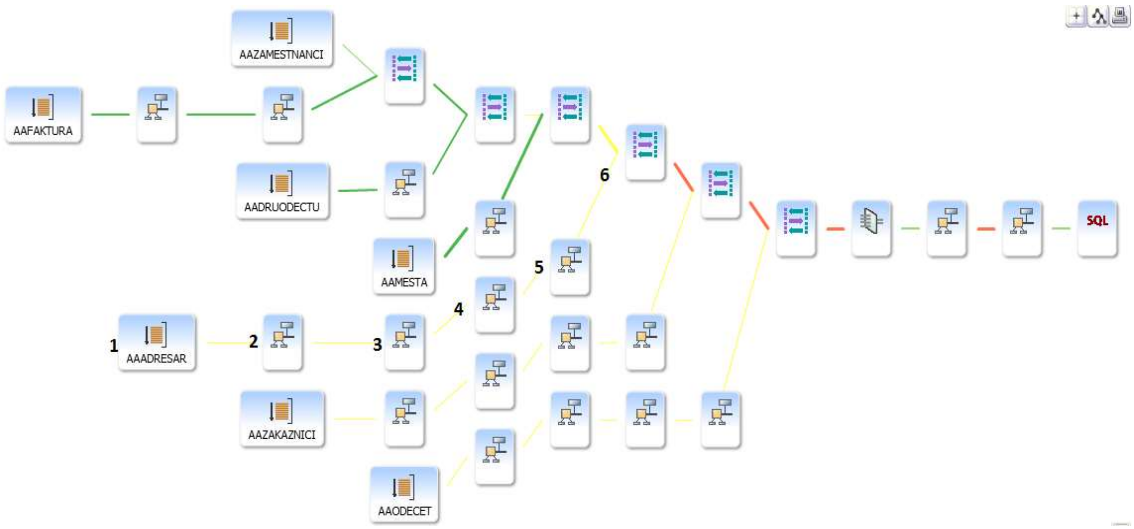
V OSD se provede další spuštění SQL Tuningu. V OEMDE12C je nový úkol a nové vylepšení SQL příkazu. SQL Tuning dále upozorňuje na kartézský součin, který je potřeba řešit propojením indexů. Určitá změna nastala v exekučním plánu, která je dále popsaná u grafického modelu.

Exekuční plán:

Operation	Object	Line ID	Predicate	Pruning	Operation Cost	Estimated Rows	Estimated Bytes
SELECT STATEMENT		0				2,884T	262P
PX COORDINATOR		1					
PX SEND QC (RANDOM)	:TQ10003	2				2,884T	262P
FILTER		3					
MERGE JOIN CARTESIAN		4				2,884T	262P
MERGE JOIN CARTESIAN		5				1,244G	103T
MERGE JOIN CARTESIAN		6				887M	60G
MERGE JOIN CARTESIAN		7				592K	27M
MERGE JOIN CARTESIAN		8				1,810	62K
MERGE JOIN CARTESIAN		9				37	999
TABLE ACCESS FULL	AAZAMESTNANCI	10			2	1	15
BUFFER SORT		11			1	34	408
PX BLOCK ITERATOR		12				34	408
TABLE ACCESS FULL	AAFAKTURA	13			2	34	408
BUFFER SORT		14			8	49	343
TABLE ACCESS FULL	AADRUODECTU	15			2	49	343
BUFFER SORT		16			237	327	3,597
TABLE ACCESS FULL	AAMESTA	17			2	327	3,597
BUFFER SORT		18			223K	1,499	34K
PX RECEIVE		19				1,499	34K
PX SEND BROADCAST	:TQ10000	20				1,499	34K
PX BLOCK ITERATOR		21				1,499	34K
TABLE ACCESS FULL	AAADRESAR	22			2	1,499	34K
BUFFER SORT		23			223M	1,402	21K
PX RECEIVE		24				1,402	21K
PX SEND BROADCAST	:TQ10001	25				1,402	21K
PX BLOCK ITERATOR		26				1,402	21K
TABLE ACCESS FULL	AAZAKAZNICI	27			2	1,402	21K
BUFFER SORT		28			313G	2,319	19K
PX RECEIVE		29				2,319	19K
PX SEND BROADCAST	:TQ10002	30				2,319	19K
PX BLOCK ITERATOR		31				2,319	19K
TABLE ACCESS FULL	AAODECET	32			2	2,319	19K

Obrázek 21 Druhý exekuční plán. Zdroj: Autor

Grafický exekuční plán:



Obrázek 22 Druhý grafický exekuční plán. Zdroj: Autor

1. Tabulka AAADRESAR je prohledávána TABLE ACCESS FULL.
2. PX BLOCK ITERATOR –Rozdělení řádků na bloky, pro lepší skenování.
3. PX SEND BROADCAST – Přeposílá získané řádky tabulky do nadřazeného procesu.
4. PX RECEIVE – Přijímá řádky tabulky z podřazeného procesu.
5. BUFFER SORT – Seřazení získaných řádků tabulky.
6. MERGE JOIN CARTESIAN – Kartézský součin.

Předávání řádků tabulky je zbytečně zřetězené. Vzhledem ke kartézskému součinu, který je velmi složitý a nákladný je tedy důležitý tento problém řešit. Jedním z řešení je, tabulky propojit přes indexy. Proto bylo za podmínku WHERE do příkazu doplněné indexové propojení tabulek.

Upravený SQL příkaz:

```
SELECT  aafaktura.datumvystaveni, aafaktura.fakturovanacastka, aazakaznici.jmeno,
aazakaznici.prijmeni, aaadresar.ulice,
aaadresar.cislopopisne, aaadresar.psc, aamesta.nazev, aaodecet.datumodectu, aadruodectu.nazev,
aazamestnanci.jmeno, aazamestnanci.prijmeni
FROM aazakaznici, aafaktura, aazamestnanci, aaadresar, aamesta, aaodecet, aadruodectu
WHERE aafaktura.datumvystaveni BETWEEN '01.01.14' AND '31.12.14'
AND aafaktura.fakturovanacastka>=9000 AND aazamestnanci.prijmeni='Hyde'
AND aafaktura.id_zakaznici=aazakaznici.id_zakaznici
AND aafaktura.id_zamestnanec=aazamestnanci.id_zamestnanec
AND aazakaznici.id_adresa=aaadresar.id_adresa
AND aaadresar.psc=aamesta.psc
AND aafaktura.id_faktura=aaodecet.id_faktura
AND aaodecet.id_druhodectu=aadruodectu.id_druhodectu;
```

4.4.1.3 Třetí spuštění

Další spuštění tentokrát upraveného SQL příkazu. Následuje kontrola SQL Tuningu.

Type	Findings	Benefit (%)
Index	The execution plan of this statement can be improved by creating one or more indices.	50.16
	SYSTEM.IDX$$_001F0002 on SYSTEM.AAZAKAZNICI("ID_ZAKAZNICI");	
	SYSTEM.IDX$$_001F0004 on SYSTEM.AAMESTA("PSC");	
	SYSTEM.IDX$$_001F0003 on SYSTEM.AAADRESAR("ID_ADRESA");	
	SYSTEM.IDX$$_001F0005 on SYSTEM.AAODECET("ID_FAKTURA");	
	SYSTEM.IDX$$_001F0001 on SYSTEM.AAZAMESTNANCI("PRIJMENI");	
	SYSTEM.IDX$$_001F0006 on SYSTEM.AADRUODECTU("ID_DRUHODECTU");	

Obrázek 23 SQL detaily. Zdroj: Autor

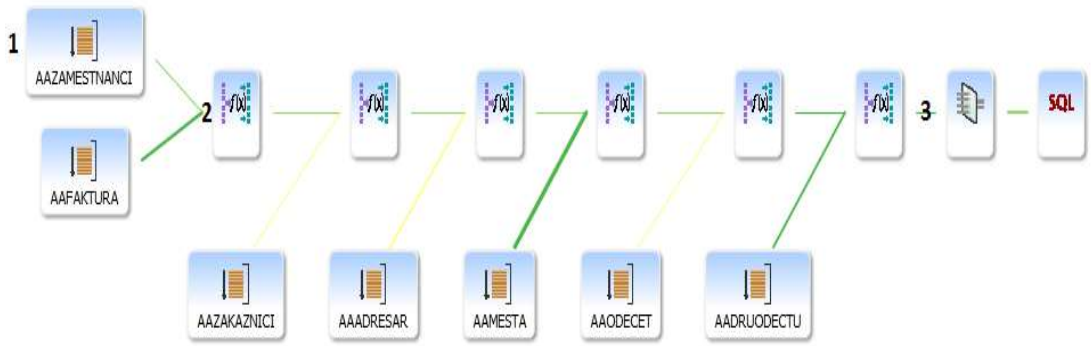
Po rozkliknutí např. vylepšení přes indexy se zobrazí doporučené podrobnosti. V horní části se nachází text, kde je vysvětleno, jak indexy zvýší výkon SQL. Sníží potřebu úplného prohledávání tabulky, které je velice časově náročné. SQL tuning dále doporučuje spustit SQL Access Advisor pro zjištění zatížení SQL. Další součástí jsou zobrazené exekuční plány. Ty jsou rozdělené na originální exekuční plán a doporučený k použití. Ty je možné zobrazit v tabulce, nebo graficky. Benefit po optimalizaci je 50,16%.

Exekuční plán:

Operation	Object	Line ID	Predicate	Pruning	Operation Cost	Estimated Rows	Estimated Bytes
SELECT STATEMENT		0				2	284
FILTER		1					
HASH JOIN		2				2	284
HASH JOIN		3				2	262
HASH JOIN		4				1	113
HASH JOIN		5				1	95
HASH JOIN		6				1	69
HASH JOIN		7				1	46
TABLE ACCESS FULL	AAZAMESTNANCI	8			3	1	20
TABLE ACCESS FULL	AAFAKTURA	9			5	34	884
TABLE ACCESS FULL	AAZAKAZNICI	10			6	1,402	32K
TABLE ACCESS FULL	AAADRESAR	11			4	1,499	39K
TABLE ACCESS FULL	AAMESTA	12			2	327	5,886
TABLE ACCESS FULL	AAODECET	13			10	2,319	42K
TABLE ACCESS FULL	AADRUODECTU	14			2	49	539

Obrázek 24 Třetí exekuční plán. Zdroj: Autor

Grafický exekuční plán:



Obrázek 25 Třetí grafický exekuční plán. Zdroj: Autor

Jak je v exekučních plánech vidět, kartézské součiny jsou vyřešené. Stále se prochází celé tabulky.

1. Tabulka AAZAMESTNANCI a i ostatní tabulky se procházejí pomocí TABLE ACCESS FULL.
2. Spojování tabulek je řešeno lépe pomocí HASH JOIN.
3. Konečný filtr, ve kterém je podmínka TO_DATE('31.12.14')>=TO_DATE('01.01.14').

Dalším krokem je implementace doporučení.

4.4.1.4 Čtvrté spuštění

Po posledním spuštění SQL Tuningu není nalezeno lepší řešení exekučního plánu. To bylo cílem optimalizace SQL příkazu.
Výsledný optimalizovaný SQL příkaz vypadá takto:

```
SELECT  aafaktura.datumvystaveni, aafaktura.fakturovanacastka, aazakaznici.jmeno,
aazakaznici.prijmeni, aaadresar.ulice,
aaadresar.cislopopisne, aaadresar.psc, aamesta.nazev, aaodecet.datumodectu, aadruodectu.nazev,
aazamestnanci.jmeno, aazamestnanci.prijmeni
FROM aazakaznici, aafaktura, aazamestnanci, aaadresar, aamesta, aaodecet, aadruodectu
WHERE aafaktura.datumvystaveni BETWEEN '01.01.14' AND '31.12.14'
AND aafaktura.fakturovanacastka>=9000 AND aazamestnanci.prijmeni='Hyde'
AND aafaktura.id_zakaznici=aazakaznici.id_zakaznici
AND aafaktura.id_zamestnanec=aazamestnanci.id_zamestnanec
AND aazakaznici.id_adresa=aaadresar.id_adresa
AND aaadresar.psc=aamesta.psc
AND aafaktura.id_faktura=aaodecet.id_faktura
AND aaodecet.id_druhodectu=aadruodectu.id_druhodectu;
```

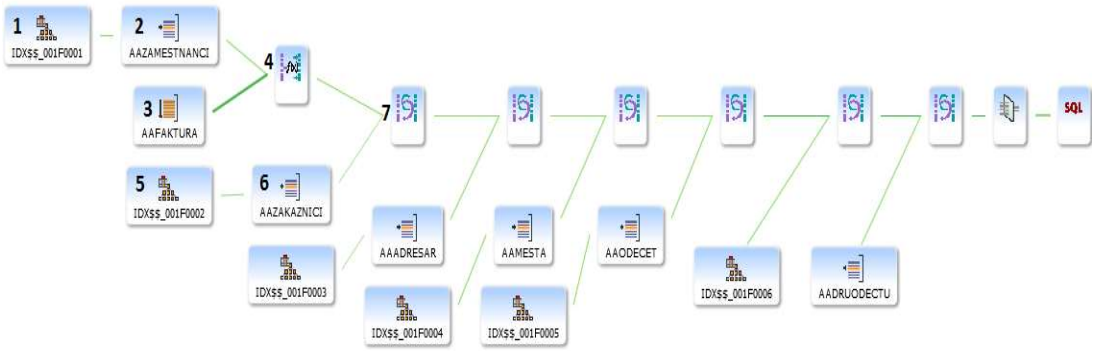
Exekuční plán optimalizovaného příkazu:

Operation	Object	Line ID	Predicate	Pruning	Operation Cost	Estimated Rows	Estimated Bytes
SELECT STATEMENT		0				2	284
FILTER		1					
NESTED LOOPS		2				2	284
NESTED LOOPS		3			2	2	284
NESTED LOOPS		4				2	262
NESTED LOOPS		5				1	113
NESTED LOOPS		6				1	95
NESTED LOOPS		7				1	69
HASH JOIN		8				1	46
TABLE ACCESS BY INDEX ROWID BATCHED	AAZAMESTNANCI	9			1	1	20
INDEX RANGE SCAN	IDX$$_000B0001	10			1	1	
TABLE ACCESS FULL	AAFAKTURA	11			5	34	884
TABLE ACCESS BY INDEX ROWID BATCHED	AAZAKAZNICI	12			1	1	23
INDEX RANGE SCAN	IDX$$_000B0002	13			1	1	
TABLE ACCESS BY INDEX ROWID BATCHED	AAADRESAR	14			1	1	26
INDEX RANGE SCAN	IDX$$_000B0003	15			1	1	
TABLE ACCESS BY INDEX ROWID BATCHED	AAMESTA	16			1	1	18
INDEX RANGE SCAN	IDX$$_000B0004	17				1	
TABLE ACCESS BY INDEX ROWID BATCHED	AAODECET	18			1	1	18
INDEX RANGE SCAN	IDX$$_000B0005	19			1	1	
INDEX RANGE SCAN	IDX$$_000B0006	20				1	
TABLE ACCESS BY INDEX ROWID	AADRJODECTU	21			1	1	11

Obrázek 26 Čtvrtý exekuční plán. Zdroj: Autor

Při porovnání exekučního plánu po prvním spuštění jsou vidět výrazné rozdíly. Náklady jsou nižší, k tabulkám se přistupuje pomocí indexů, spojování tabulek pomocí JOINŮ hash a nested loops je nejefektivnější.

Grafický model :



Obrázek 27 První grafický exekuční plán. Zdroj: Autor

1. Tabulka AAZAMESTNANCI se prochází pomocí INDEX RANGE SCAN v rozsahu "AAZAMESTNANCI"."PRIJMENI"='Hyde'.
2. TABLE ACCESS BY INDEX ROWID SCAN přistupující k tabulce AAZAMESTNANCI.
3. Jediná tabulka AAFAKTURA se prochází pomocí TABLE ACCESS FULL.
4. Tabulky AAFAKTURA a AAZAMESTNANCI se spojují pomocí HASH JOIN.
5. Tabulka AAZAKAZNICI se prochází pomocí INDEX RANGE SCAN v rozsahu "AAFAKTURA"."ID_ZAKAZNICI"="AAZAKAZNICI"."ID_ZAKAZNICI".
6. Tabulka AAZAKAZNICI se prochází pomocí INDEX RANGE SCAN v rozsahu "AAFAKTURA"."ID_ZAKAZNICI"="AAZAKAZNICI"."ID_ZAKAZNICI".
7. TABLE ACCESS BY INDEX ROWID SCAN přistupující k tabulce AAZAKAZNICI.
8. Spojení NESTED LOOPS.

4.4.2 Aplikace optimalizovaného příkazu

V kontejneru TEST2, byla vytvořena identická databáze jako v kontejneru TEST.

Z předešlé části, byl použit již optimalizovaný SQL příkaz, který bude spuštěn v kontejneru TEST2. Účelem tohoto spouštění je zjištění, zda bude potřeba SQL příkaz dál optimalizovat. Dále budou sledované časy.

4.4.2.1 První spuštění

Spuštění příkazu bylo mnohem rychlejší, protože nebylo nutné dělat úpravy, které byly již provedeny v prvním testování. Následuje kontrola SQL Tuningu.

Type	Findings	Benefit (%)
Index	The execution plan of this statement can be improved by creating one or more indices.	50.16
	SYSTEM.IDX$$_000B0002 on SYSTEM.AAZAKAZNICI("ID_ZAKAZNICI");	
	SYSTEM.IDX$$_000B0004 on SYSTEM.AAMESTA("PSC");	
	SYSTEM.IDX$$_000B0003 on SYSTEM.AAADRESAR("ID_ADRESA");	
	SYSTEM.IDX$$_000B0005 on SYSTEM.AAODECET("ID_FAKTURA");	
	SYSTEM.IDX$$_000B0001 on SYSTEM.AAZAMESTNANCI("PRIJMENI");	
	SYSTEM.IDX$$_000B0006 on SYSTEM.AADRUODECTU("ID_DRUHODECTU");	

Obrázek 28 SQL detaily. Zdroj: Autor

Jde o stejnou úpravu, jako v posledním kroku se stejným benefitem 50,16%. Cílem je přístup přes indexy, který je méně nákladný, než procházení celé tabulky. Následuje jeho implementace úpravy.

4.4.2.2 Druhé spuštění

Poslední spuštění bylo rychlejší a bylo dosáhnuto stejného výsledku, jako u prvního příkladu. Exekuční plán, je totožný jako na obrázku 26 a stejné to je i s grafickým modelem na obrázku.

Konečný exekuční plán:

Operation	Object	Line ID	Predicate	Pruning	Operation Cost	Estimated Rows	Estimated Bytes
SELECT STATEMENT		0				2	284
FILTER		1					
NESTED LOOPS		2				2	284
NESTED LOOPS		3			2	2	284
NESTED LOOPS		4				2	262
NESTED LOOPS		5				1	113
NESTED LOOPS		6				1	95
NESTED LOOPS		7				1	69
HASH JOIN		8				1	46
TABLE ACCESS BY INDEX ROWID BATCHED	AAZAMESTNANCI	9			1	1	20
INDEX RANGE SCAN	IDX$$_000B0001	10			1	1	
TABLE ACCESS FULL	AAFAKTURA	11			5	34	884
TABLE ACCESS BY INDEX ROWID BATCHED	AAZAKAZNICI	12			1	1	23
INDEX RANGE SCAN	IDX$$_000B0002	13			1	1	
TABLE ACCESS BY INDEX ROWID BATCHED	AAADRESAR	14			1	1	26
INDEX RANGE SCAN	IDX$$_000B0003	15			1	1	
TABLE ACCESS BY INDEX ROWID BATCHED	AAMESTA	16			1	1	18
INDEX RANGE SCAN	IDX$$_000B0004	17				1	
TABLE ACCESS BY INDEX ROWID BATCHED	AAODECET	18			1	1	18
INDEX RANGE SCAN	IDX$$_000B0005	19			1	1	
INDEX RANGE SCAN	IDX$$_000B0006	20				1	
TABLE ACCESS BY INDEX ROWID	AADRUODECTU	21			1	1	11

Obrázek 29 Konečný exekuční plán. Zdroj: Autor

4.4.3 HINTY

Jak bylo uvedeno v teoretické části, hinty instruují optimalizátor. V této části, bude názorně ukázáno používání hintů. Nejprve bude zjištěno, jaký exekuční plán zvolí samotný optimalizátor. V dalších částech se v příkazu nachází hinty a výsledkem jsou jejich exekuční plány. Během testování, byly zaznamenávány časy, které jsou brány jako doplněk. Při zvolené metodě spojování tabulek HASH JOIN, bude uveden celý spuštěný SQL příkaz. Jak je uvedeno v teoretické části, HINT se vkládá v příkazu za SELECT, UPDATE, DELETE. V dalších případech bude uveden samotný HINT.

4.4.3.1 Samostatný SQL příkaz

V příkazu se nenachází žádný HINT, takže optimalizátor pracuje samostatně a zvolil nejvhodnější exekuční plán. Použitý SQL příkaz je modifikací předešlého příkazu, kde jsou jiné podmínky.

SQL příkaz:

```
SELECT aafaktura.datumvystaveni, aafaktura.fakturovanacastka, aazakaznici.jmeno,
aazakaznici.prijmeni, aaadresar.ulice,
aaadresar.cislopopisne, aaadresar.psc, aamesta.nazev, aaodecet.datumodectu, aadruodectu.nazev,
aazamestnanci.jmeno,
aazamestnanci.prijmeni
FROM aazakaznici, aafaktura, aazamestnanci, aaadresar, aamesta, AAODECET, aadruodectu
WHERE aafaktura.fakturovanacastka < 1500
AND aafaktura.fakturovanacastka > 1300
AND aafaktura.id_zakaznici=aazakaznici.id_zakaznici
AND aafaktura.id_zamestnanec=aazamestnanci.id_zamestnanec
AND aazakaznici.id_adresa=aaadresar.id_adresa
AND aaadresar.psc=aamesta.psc
AND aafaktura.id_faktura=aaodecet.id_faktura
AND aaodect.id_druhodectu=aadruodectu.id_druhodectu
AND aaadresar.psc>50000
AND aadruodectu.nazev LIKE 'cc%'
ORDER BY aazakaznici.prijmeni, aazakaznici.jmeno;
```

Exekuční plán SQL příkazu:

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	142	26 (4)	00:00:01
1	SORT ORDER BY		1	142	26 (4)	00:00:01
2	NESTED LOOPS		1	142	25 (0)	00:00:01
3	NESTED LOOPS		1	142	25 (0)	00:00:01
* 4	HASH JOIN		1	124	24 (0)	00:00:01
5	NESTED LOOPS		3	339	22 (0)	00:00:01
6	NESTED LOOPS		3	339	22 (0)	00:00:01
* 7	HASH JOIN		2	190	18 (0)	00:00:01
* 8	HASH JOIN		40	2760	14 (0)	00:00:01
* 9	HASH JOIN		40	1840	8 (0)	00:00:01
* 10	TABLE ACCESS FULL	AAFAKTURA	40	1040	5 (0)	00:00:01
11	TABLE ACCESS FULL	AAZAMESTNANCI	199	3980	3 (0)	00:00:01
12	TABLE ACCESS FULL	AAZAKAZNICI	1402	32246	6 (0)	00:00:01
* 13	TABLE ACCESS FULL	AAADRESAR	75	1950	4 (0)	00:00:01
* 14	INDEX RANGE SCAN	IDX$$_001F0005	1		1 (0)	00:00:01
15	TABLE ACCESS BY INDEX ROWID	AAODECET	1	18	2 (0)	00:00:01
* 16	TABLE ACCESS FULL	AADRUODECTU	1	11	2 (0)	00:00:01
* 17	INDEX RANGE SCAN	IDX$$_001F0004	1		0 (0)	00:00:01
18	TABLE ACCESS BY INDEX ROWID	AAMESTA	1	18	1 (0)	00:00:01

Obrázek 30 Exekuční plán. Zdroj: Autor

Pro kontrolu byl příkaz spuštěn ještě v SQL Tuningu. Ten neodhalil žádné lepší řešení exekučního plánu. Jak je vidět na obrázku, tabulky jsou procházeny kombinovaně přes indexy, nebo celé. Tabulky jsou kombinovaně spojované pomocí HASH JOIN a NESTED JOIN.

4.4.3.2 HINT – HASH JOIN

V druhé fázi bude vyzkoušeno spojení tabulek pomocí HASH JOIN. Jedná se o nejúčinnější metodu spojování tabulek.

PŘÍKAZ + HINT :

```
SELECT
/*+USE_HASH(aafaktura,aazakaznici)USE_HASH(aafaktura,aazamestnanci)USE_HASH(aazakaznici,
aaadresar) USE_HASH(aaadresar,aamesta) USE_HASH(aafaktura,aaodecet)
USE_HASH(aaodecet,aadruodectu)*/
aafaktura.datumvystaveni, aafaktura.fakturovanacastka, aazakaznici.jmeno, aazakaznici.prijmeni,
aaadresar.ulice,
aaadresar.cislopopisne, aaadresar.psc, aamesta.nazev, aaodecet.datumodectu, aadruodectu.nazev,
aazamestnanci.jmeno,
aazamestnanci.prijmeni
FROM aazakaznici, aafaktura, aazamestnanci, aaadresar, aamesta, AAODECET, aadruodectu
WHERE aafaktura.fakturovanacastka < 1500
AND aafaktura.fakturovanacastka > 1300
AND aafaktura.id_zakaznici=aazakaznici.id_zakaznici
AND aafaktura.id_zamestnanec=aazamestnanci.id_zamestnanec
AND aazakaznici.id_adresa=aaadresar.id_adresa
AND aaadresar.psc=aamesta.psc
AND aafaktura.id_faktura=aaodecet.id_faktura
AND aaodect.id_druhodectu=aadruodectu.id_druhodectu
AND aaadresar.psc>50000
AND aadruodectu.nazev LIKE 'cc%'
ORDER BY aazakaznici.prijmeni, aazakaznici.jmeno;
```

Exekuční plán:

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	142	33 (4)	00:00:01
1	SORT ORDER BY		1	142	33 (4)	00:00:01
* 2	HASH JOIN		1	142	32 (0)	00:00:01
* 3	HASH JOIN		1	124	30 (0)	00:00:01
* 4	HASH JOIN		3	339	28 (0)	00:00:01
* 5	HASH JOIN		2	190	18 (0)	00:00:01
* 6	HASH JOIN		2	150	15 (0)	00:00:01
* 7	HASH JOIN		40	1960	11 (0)	00:00:01
* 8	TABLE ACCESS FULL	AAFAKTURA	40	1040	5 (0)	00:00:01
9	TABLE ACCESS FULL	AAZAKAZNICI	1402	32246	6 (0)	00:00:01
* 10	TABLE ACCESS FULL	AAADRESAR	75	1950	4 (0)	00:00:01
11	TABLE ACCESS FULL	AAZAMESTNANCI	199	3980	3 (0)	00:00:01
12	TABLE ACCESS FULL	AAODECET	2319	41742	10 (0)	00:00:01
* 13	TABLE ACCESS FULL	AADRUODECTU	1	11	2 (0)	00:00:01
14	TABLE ACCESS FULL	AAMESTA	327	5886	2 (0)	00:00:01

Obrázek 31 Exekuční plán HASH JOIN. Zdroj: Autor

Zdlouhavé může být celkové prohledávání tabulek. Metoda spojení tabulek je v tomto případě HASH JOIN. Zajímavé je, že exekuční plán zvolený optimalizátorem má vyšší náklady než exekuční plán příkazu s HINTEM – HASH JOIN.

4.4.3.3 HINT - NESTED LOOPS

Další metoda spojení je vnořených smyček – NESTED LOOPS.

HINT:

```
/*+
USE_NL(aafaktura,aazakaznici)USE_NL(aafaktura,aazamestnanci)USE_NL(aazakaznici,aaadresar)
USE_NL(aaadresar,aamesta) USE_NL(aafaktura,aaodecet) USE_nl(aaodect,aadruodectu)*/
```

Exekuční plán:

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	142	257 (1)	00:00:01
1	SORT ORDER BY		1	142	257 (1)	00:00:01
2	NESTED LOOPS		1	142	256 (1)	00:00:01
3	NESTED LOOPS		1	122	253 (1)	00:00:01
4	NESTED LOOPS		1	104	251 (1)	00:00:01
5	NESTED LOOPS		2	156	245 (1)	00:00:01
6	NESTED LOOPS		2	110	235 (1)	00:00:01
7	NESTED LOOPS		68	1972	12 (0)	00:00:01
* 8	TABLE ACCESS FULL	AADRUODECTU	1	11	2 (0)	00:00:01
* 9	TABLE ACCESS FULL	AAODECET	47	846	10 (0)	00:00:01
* 10	TABLE ACCESS FULL	AAFAKTURA	1	26	3 (0)	00:00:01
* 11	TABLE ACCESS FULL	AAZAKAZNICI	1	23	5 (0)	00:00:01
* 12	TABLE ACCESS FULL	AAADRESAR	1	26	3 (0)	00:00:01
* 13	TABLE ACCESS FULL	AAMESTA	1	18	2 (0)	00:00:01
* 14	TABLE ACCESS FULL	AAZAMESTNANCI	1	20	3 (0)	00:00:01

Obrázek 32 Exekuční plán NESTED LOOPS. Zdroj: Autor

Tabulky jsou opět prohledávány celkově a jsou spojeny metodou NESTED LOOPS. Náklady exekučního plánu jsou vysoké. Po spuštění příkazu v SQL Tuningu je navrhnuto vhodnější řešení. Po prohlednutí navrhovaného exekučního plánu bylo zjištěno, že jde o plán s metodou spojení tabulek HASH JOIN, stejně jako je v předešlém případě. Benefit z této implementace by byl 98,08%.

4.4.3.4 HINT - MERGE

Spojení tabulek pomocí metody setřídění MERGE.

HINT:

```
/*+
USE_merge(aafaktura,aazakaznici)USE_merge(aafaktura,aazamestnanci)USE_merge(aazakaznici,aaadresar)
USE_merge(aaadresar,aamesta) USE_merge(aafaktura,aaodecet) USE_merge(aaodecet,aadruodectu)*/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	142	45 (29)	00:00:01
1	SORT ORDER BY		1	142	45 (29)	00:00:01
2	MERGE JOIN		1	142	44 (28)	00:00:01
3	SORT JOIN		1	124	41 (27)	00:00:01
4	MERGE JOIN		1	124	40 (25)	00:00:01
5	SORT JOIN		1	104	36 (25)	00:00:01
6	MERGE JOIN		1	104	35 (23)	00:00:01
7	SORT JOIN		3	279	32 (22)	00:00:01
8	MERGE JOIN		3	279	31 (20)	00:00:01
9	SORT JOIN		2	150	20 (25)	00:00:01
10	MERGE JOIN		2	150	19 (22)	00:00:01
11	SORT JOIN		40	1960	14 (22)	00:00:01
12	MERGE JOIN		40	1960	13 (16)	00:00:01
13	SORT JOIN		40	1040	6 (17)	00:00:01
* 14	TABLE ACCESS FULL	AAFAKTURA	40	1040	5 (0)	00:00:01
* 15	SORT JOIN		1402	32246	7 (15)	00:00:01
16	TABLE ACCESS FULL	AAZAKAZNICI	1402	32246	6 (0)	00:00:01
* 17	SORT JOIN		75	1950	5 (20)	00:00:01
* 18	TABLE ACCESS FULL	AAADRESAR	75	1950	4 (0)	00:00:01
* 19	SORT JOIN		2319	41742	11 (10)	00:00:01
20	TABLE ACCESS FULL	AAODECET	2319	41742	10 (0)	00:00:01
* 21	SORT JOIN		1	11	3 (34)	00:00:01
* 22	TABLE ACCESS FULL	AADRUODECTU	1	11	2 (0)	00:00:01
* 23	SORT JOIN		199	3980	4 (25)	00:00:01
24	TABLE ACCESS FULL	AAZAMESTNANCI	199	3980	3 (0)	00:00:01
* 25	SORT JOIN		327	5886	3 (34)	00:00:01
26	TABLE ACCESS FULL	AAMESTA	327	5886	2 (0)	00:00:01

Obrázek 33Exekuční plán MERGE. Zdroj: Autor

Jako v předešlých případech se tabulky prohledávají kompletně. Tabulky jsou spojovány metodou MERGE JOIN. V rámci této metody dochází ještě k seřazení, proto jsou náklady dvakrát vyšší.

5 Shrnutí výsledků

V této kapitole, se nachází rekapitulace průběhu testování v praktické části. Část optimalizace obsahuje souhrn optimalizace SQL příkazu a jeho nasazení do totožné databáze v dalším kontejneru. Část věnována HINTŮM, shrnuje použití a dosažené výsledky. Během testování byly zaznamenávány časy, které jsou ve výsledcích porovnány.

5.1 Optimalizace

V prvním testování si SQL příkaz skutečně prošel všemi komponentami optimalizátoru. Už po prvním spuštění se objevil kartézský součin a celkové prohledávání tabulek. SQL Tuning Advisor navrhoval dvě řešení a jedno z nich bylo implementováno. Po implementaci nastal další problém s řetězeným předáváním řádků tabulky. Proto bylo přistoupeno k přepsání příkazu. Po třetím spuštění byly problémy vyřešeny, ale byla navrhnuta další implementace. Čtvrté spuštění už vykazovalo optimalizovaný SQL příkaz. Na začátku špatně napsaný a neoptimalizovaný SQL příkaz, se pomocí SQL Tuningu stal optimalizovaným. První testování probíhalo v kontejneru TEST.

V druhém testování na identické databázi, bylo zkoumáno, zda optimalizovaný dotaz je potřeba dále upravovat nebo optimalizovat. Zde nastala jenom poslední implementace spuštění SQL Access Advisor. Po které SQL příkaz nabyl stejného exekučního plánu, jako u prvního testování. Druhé testování probíhalo v kontejneru TEST2.

Během prvního a druhého testování byly zaznamenány časy, po prvním spuštění operace. Zaznamenané časy optimalizace jsou v následující tabulce 4. Z OSD byly získány časy run, autotrace, exekuční plán, SQL Tuning. Poslední časy jsou zaznamenány z OEME12C a to v části SQL Tuning Advisor – Duration a benefit.

Tabulka 4 Výsledky měření optimalizace. Zdroj: Autor

Kontejner	Run	Autotrace	Exekuční plán	SQL Tuning OSD	SQL Tuning OEMDE12C	Benefit
TEST-1	0,953 s	0,301 s	0,051 s	40,662 s	37,0s	91,48%
TEST-2	0,945 s	0,521 s	0,201 s	6,102 s	4 s	X
TEST-3	0,03 s	1,758 s	0,151 s	3,901 s	2 s	50,16%
TEST-4	0,094 s	0,408 s	0,101 s	4,622 s	2 s	X
TEST2-1	0,072 s	0,412 s	0,103 s	7,951 s	5 s	50,16%
TEST2-2	0,031 s	0,21 s	0,052 s	4,779 s	3 s	X

V rámci prvního testování je možné pozorovat postupnou optimalizaci příkazu, která se projevila i v časech provedení. Řádek TEST-1 se vyznačuje dlouhými časy a odpovídá to neoptimalizovanému příkazu. Ve sloupci s časy z OEMDE12C kontejneru TEST, je vidět výrazné zlepšení. Neoptimalizovaný příkaz trval 37 sekund a po optimalizaci dosáhl 2 sekund. Časy z OSD jsou rozmanité, ale i u nich je vidět zlepšení.

U druhého testování v kontejneru TEST2 je vidět také zlepšení. Časy prvních spuštění v kontejnerech porovnávat nelze. Ale zajímavé porovnání je u posledního spuštění v kontejneru TEST a prvního spuštění v kontejneru TEST2. Časy v kontejneru TEST2 jsou logicky delší, protože chybí poslední implementace. Po implementaci se dá očekávat, že časy v kontejneru TEST2 budou stejné nebo lepší. Ve sloupci OEMDE12C u posledního spuštění kontejnerů je rozdíl jedna sekunda a TEST2 má delší čas 3 sekundy. V rámci fungování databáze má i jedna sekunda velký vliv. Tento výsledek si lze vysvětlit dvěma způsoby. Rozdíl je možný zaokrouhlením času, nebo delší prací optimalizátoru s příkazem. Časy z OSD jsou spíše kratší pro kontejner TEST2.

Dosažené výsledky jsou zajímavé. Postup optimalizace probíhal bez problémů. Očekávání u nasazení příkazu do dalšího kontejneru bylo trochu jiné, ale muselo dojít k implementaci pro lepší fungování. Výsledné časy odpovídají optimalizaci, ale mohou být ovlivněné zaokrouhlováním nebo prací optimalizátoru s příkazem.

5.2 HINTY

Pro samotný SQL příkaz optimalizátor zvolil podle něho nejlepší exekuční plán. Pomocí HINTŮ byl optimalizátor ovlivněn k jiným metodám spojování tabulek. Jejich exekuční plány, měly různé náklady s velkým rozmezím. Paradoxně nejnižší náklady měl exekuční plán HINTU- HASH, i když byly tabulky celkově prohledávány. V dalších případech, optimalizátor tuto metodu spojování tabulek volil jako ideální exekuční plán.

Tabulka 5 Výsledky měření hinty. Zdroj: Autor

HINT	Run	Autotrace	Exekuční plán	SQL Tuning OSD	SQL Tuning OEME12C
HASH	0,394 s	0,26 s	0,052 s	3,653 s	2 s
SQL samostatný	0,049 s	0,309 s	0,101 s	5,303 s	3 s
MERGE	0,045 s	0,267 s	0,051 s	3,451 s	2 s
NESTED LOOPS	0,071 s	0,408 s	0,051 s	4,201 s	3 s

V tabulce jsou hinty a samostatný SQL příkaz seřazeny od exekučního plánu s nejnižšími náklady po nejvyšší. Další časy byly zaznamenávány během testování. Z OSD byly získány časy run, autotrace, exekuční plán, SQL Tuning. Poslední čas je zaznamenán z OEME12C a to v části SQL Tuning Advisor – Duration.
Podle uvedených časů, které jsou spíše doplňující, má nejlepší časy HINT – MERGE. Další časy se hodně rozchází.

Výsledkem je, že optimalizátor skutečně lze ovlivňovat pomocí hintů. Pokud správce databáze ví, že lze zvolit lepší exekuční plán a zná cestu k lepšímu exekučnímu plánu, může dosáhnout daleko lepších výsledků a dokonalejší optimalizace.

6 Závěry a doporučení

Databázový systém Oracle Database 12c, má jednoznačné výhody oproti předešlým verzím. V rámci bakalářské práce byla využita možnost tvorby kontejnerů, která byla novinkou. To umožnilo samostatné spouštění SQL příkazů, bez ovlivnění optimalizátoru. Vytváření nových kontejnerů a práce s nimi nabízí nové možnosti a bylo to velkou zkušeností v rámci bakalářské práce. Plusem společnosti Oracle, je jejich výborná dokumentace ke všem produktům. V rámci práce bylo čerpáno z různých dokumentací, manuálů a doporučení. Dokumenty jsou výborně napsané a obsahují veškeré teoretické i praktické informace. Optimalizace SQL příkazu pomocí SQL Tuningu a OEMDE12C, ověřila a aplikovala informace z teoretické části. Bylo zajímavé si vyzkoušet závislost optimalizátoru na statistikách, podle kterých vytváří exekuční plány, v praxi v rámci praktické části práce. Zajímavá byla práce s SQL Tuningem. Ten doporučoval řešení optimalizace a předpokládal jejich dopad. To umožňuje optimalizovat i uživateli, se základními znalostmi. Další zkušeností bylo používání HINTŮ a jejich dopad na exekuční plány.

V rámci testování byly zaznamenávány časy, které bohužel nelze považovat za důvěryhodné. Všechny testy probíhaly na jednom počítači, který byl určitým způsobem vytížen, a to mohlo časy ovlivnit. Dalším faktorem je cachování.

K dalšímu zkoumání, by mohl být využit systém Oracle Enteprise Manager. Ten umožňuje rozsáhlé sledování a testování databáze. Pro OD12C by byl nejvhodnější Oracle Enteprise Manager 12c. Nedávno vydanou novinkou je verze Oracle Enteprise Manager 13c.

Cíle bakalářské práce byly splněny. Teoretická část se věnovala optimalizaci v návaznosti na optimalizátor a další důležité součásti optimalizace a databáze. Podrobněji se zaměřovala na statistiky a exekuční plány. V praktické části byly zužitkovány teoretické znalosti. Pomocí systému Oracle Database 12c, byla nainstalována kontejnerová databáze. V každém využitém kontejneru byla vytvořená identická data. Dále se pokračovalo optimalizací SQL příkazu. Plně optimalizovaný SQL příkaz se pak použil v dalším kontejneru, kde se na stejných datech testoval dopad úplně optimalizovaného příkazu. V další části se ovlivňoval optimalizátor pomocí hintů a sledoval dopad na exekuční plány. Všechny zkoumání proběhly úspěšně.

7 Seznam použité literatury

[1] Bryla, Bob a Loney, Kevin. *Mistrovství v Oracle Database 11g.* Brno : ComputerPress, 2009.

[2] Václav, Nidrle. Optimalizace obecně. *Nidrle Vaclav.* [Online] 8. Listopad 2003. [Citace: 24. Srpen 2014.] http://nidrle.vaclav.sweb.cz/oracle2/optimalizace.html.

[3] Alapati, Sam R., Kuhn, Darl a Padfield, Bill. *Oracle Database 12c Performance Tuning Recipes.* místo neznámé : Apress, 2013.

[4] Ashdown, Lance a Kyte, Tom. Oracle Database Concepts, 11g Release 2 (11.2). *Oracle Help Center.* [Online] Květen 2015. [Citace: 24. Srpen 2015.] https://docs.oracle.com/cd/E11882_01/server.112/e40540.pdf.

[5] Ashdown, Lance, Colgan, Maria a Kyte, Tom. SQL Tuning Guide 12c Release 1 (12.1). *Oracle Help Center.* [Online] Prosinec 2014. [Citace: 10. Červenec 2015.] http://docs.oracle.com/database/121/TGSQL/E49106-09.pdf.

[6] Cyran, Michele. Oracle8i Designing and Tuning for Performance. *Oracle Help Center.* [Online] Prosinec 1999. [Citace: 12. Duben 2016.] http://docs.oracle.com/cd/A87862_01/NT817CLI/server.817/a76992/toc.htm.

[7] Green, Connie Dialeris. Oracle9i Database Performance Tuning Guide and Reference, Release 2 (9.2). *Oracle Help Center.* [Online] Říjen 2002. [Citace: 19. Duben 2016.] https://docs.oracle.com/cd/B10500_01/server.920/a96533/title.htm.

[8] Nyffenegger, René. Oracle SQL hints. *René Nyffenegger's collection of things on the web.* [Online] 29. Srpen 2014. [Citace: 5. Srpen 2015.] http://www.adp-gmbh.ch/ora/sql/hints/.

[9] Oracle Database 12c PL/SQL. *Oracle.* [Online] Oracle, 17. Prosinec 2015. [Citace: 23. Duben 2016.] http://www.oracle.com/technetwork/database/features/plsql/index.html.

[10] Colgan, Maria. *SQL Plan Management with Oracle Database 12c.* Redwood Shores : Oracle, 2013.

[11] Rich, Bert. *2 Day DBA 12c Release 1 (12.1).* Redwood Shores : Oracle Database, 2014.

[12] Kyte, Tom a Ashdown, Lance. Oracle Database Concepts , 12c Release 1 (12.1). *Oracle Help Center.* [Online] Září 2015. [Citace: 23. Duben 2016.] https://docs.oracle.com/database/121/CNCPT/E41396-13.pdf.

[13] Oracle Database 12c: EM Database Express . *Oracle.* [Online] [Citace: 23. Duben 2016.] http://www.oracle.com/technetwork/database/manageability/emx-intro-1965965.html.

[14] Oracle Enterprise Manager Cloud Control Introduction, 13. *Oracle Help Center.* [Online] Únor 2016. [Citace: 23. Duben 2016.] http://docs.oracle.com/cd/E63000_01/EMCON/EMCON.pdf.

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Barbora Spěváková**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Barbora Tesařová**

Název práce:
Optimalizace databáze Oracle

Název práce v AJ:
Optimization database Oracle

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je ukázka možností optimalizace databáze s pomocí nástrojů Oracle.

Osnova práce:
1. Úvod
2. Teoretická část
   - Optimalizace
   - Možnosti optimalizace
   - Optimalizace-Exekuční plány
   - Optimalizace- Statistiky
   - Optimalizace-Testy databáze
3. Praktická část
   - Vytvoření ukázkové databáze
   - Ukázky
   - Řešení
4. Závěr

Projednáno dne: 15.10.2014

Podpis studenta Spěváková

Podpis vedoucího práce