UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Knihovna pro analýzu SQL dotazů

Bc. Miroslav Moravec

Diplomová práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Miroslav Moravec**
Osobní číslo:	**I11393**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Knihovna pro analýzu SQL dotazů**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V teoretické části bude provedena rešerše problematiky optimalizace na úrovni SQL dotazů. Cílem práce bude vytvořit knihovnu v jazyce C♯, která zanalyzuje zadaný SQL dotaz a poskytne spolu s datovým slovníkem informace o tabulkách a sloupcích ze kterých je dotaz tvořen, dále o podmínkách, které jsou uvedeny a nabídne možné využití indexů (pokud pro sloupce v podmínkách existují).
Dále bude vytvořena jednoduchá aplikace, která otestuje fungování knihovny.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

LACKO, Luboslav. Oracle: správa, programování a použití databázového systému. 1. vyd. Praha: Computer Press, 2002, 464 s. ISBN 80-722-6699-3.
LONEY, Kevin. Mistrovství v Oracle Database 10g. 1. vyd. Brno: Computer Press, 2006, 700 s. ISBN 80-251-1277-2.
KYTE, Thomas. Expert Oracle database architecture: Oracle database 9i, 10g, and 11g programming techniques and solutions. 2nd ed. Berkeley, Calif.: Apress, 2010. ISBN 978-143-0229-469.
www.oracle.com
msdn.microsoft.com/en-us/library

Vedoucí diplomové práce: **Ing. Jiří Zechmeister**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 31. 07. 2013 Bc. Miroslav Moravec

Anotace

Práce je zaměřena na optimalizaci příkazů v databázovém systému Oracle 11gR2. Zabývá se širokou škálou zásad, činností a nástrojů, jenž by měly být pro výkonný provoz systému ctěny, vykonávány a využívány. V rámci práce byla vytvořena aplikace založená na knihovně v jazyce c#, která umožňuje dle definovaných syntaktických pravidel parsovat zadaný SQL příkaz SELECT a výstupní datovou strukturu dále využívá k zobrazení detailních informací o tomto dotazu. Implementovány jsou i jednodušší algoritmy, jejichž účelem je zvážit prospěšnost vytvoření indexů pro analyzovaný dotaz.

Klíčová slova

optimalizace SQL dotazů, Oracle 11gR2, databázový systém, ladění výkonu, parser, příkaz select

Title

Library for analysis SQL queries

Annotation

The work is focused on optimizing the commands in the Oracle 11gR2 database system. It deals with a wide range of principles, actions and tools that should be revered, exercised and used for efficient system operation. The work included creating an application based on library in c# language that allows to parse the specified SQL SELECT statement by the defined syntax rules, and then used the output data structure to display the detailed informations about the query. Also are implemented simplier algorithms whose purpose is to consider the benefits of indexes for the analyzed query.

Keywords

optimizing SQL queries, Oracle 11gR2, database system, performance tuning, parser, the "select" command

Obsah

Seznam zkratek

ADDM	Automatic Database Diagnostic Monitor
AMM	Automatic Memory Management
AWR	Automatic Workload Repository
BNF	Backusova-Naurova forma
CBO	Cost-Based Optimization
CPU	Central Processing Unit
DDL	Data Definition Language
DML	Data Manipulation Language
EM	Oracle Enterprise Manager
GUI	Graphical User Interface
I/O	Input/Output
IT	Information technology
ODP	Oracle Data Provider
OLTP	Online Transaction Processing
PGA	Program Global Area
PL/SQL	Procedural Language/Structured Query Language
SGA	System Global Area
SMB	SQL Management Base
SQL	Structured Query Language
STS	SQL Tuning Sets

Seznam obrázků

Seznam tabulek

Úvod

V oblasti vědy a techniky platí, že rychlost je jedním z atributů pokroku. Zrychlují dopravní prostředky, přenosové sítě i informační technologie a systémy. Především právě v oboru IT je tento vývoj nejznatelnější. Přestože výkon počítačového hardware v posledních letech roste opravdu závratným tempem - co bylo rychlé před deseti lety, je dnes mnohdy zastaralé - stále je třeba s jeho prostředky pracovat efektivně, neboť se zvyšují i nároky na funkčnost aplikací a převládá snaha dostupný výkon využít maximálně.

Jinak tomu není ani u databázových systémů, jejichž optimalizací se tato práce zabývá. Vzhledem k moderním technologiím je možné provádět operace s daty mnohem rychleji než dříve, ovšem právě i díky těmto možnostem se nelze spokojit se stejnými a stejně rychle prováděnými operacemi nad srovnatelně velkými množinami údajů - uživatelé aplikací vyžadují stále složitější operace nad mnohdy obrovskými databázemi a potřebné výsledky jim musí být vráceny co nejdříve.

Z těchto důvodu optimalizace výkonu databáze je a v budoucnu zřejmě stále bude aktuální disciplínou. Efektivnost provedení SQL dotazu zdaleka není určena pouhou podobou příkazu samotného, právě naopak, je závislá na mnoha faktorech. Tato diplomová práce se zaměřuje především na optimalizaci na úrovni SQL dotazu - tedy zdánlivě způsoby, jak zefektivnit vykonávání dotazu pomocí změny jeho syntaktické podoby. Ovšem tyto možnosti jsou značně omezené a nejlepších výsledků lze dosáhnout pouze s využitím dalších prostředků, které je tedy nutné do problematiky tohoto textu také zařadit.

Vlastností moderních technologií je i zjednodušení práce IT vývojářů a správců, které umožňuje vývoj či administraci systémů prostřednictvím specializovaných GUI aplikací. Vývojář se v podstatě stává uživatelem a zdrojový kód vyvíjené aplikace vytváří aplikace jiná. Analogicky správce systémů má k dispozici sadů nástrojů, které mu práci usnadňují, nebo ji dokonce dělají za něj. Databáze Oracle není výjimkou, produkt je stále rozšiřován o nové funkce, které poskytují komfortní uživatelské rozhraní pro vývoj a správu databáze, případně některé činnosti automatizují. Jejich významný podíl je určen právě k zajištění optimálního provozu databáze, tudíž zde budou tyto nástroje také popsány.

Práce je ve své většině zaměřena na databázový systém Oracle 11gR2. Sedmá kapitola ovšem stručně pojednává o vylepšeních v problematice optimalizace, která přinesla verze 12c, jenž byla v době psaní těchto řádků novinkou. Jelikož lze zřejmě jen obtížně nalézt kompetentnější zdroj, vychází následující text především z oficiální dokumentace k databázovému systému Oracle, odkud je také převzata většina praktických ukázek.

V rámci praktické části bylo třeba implementovat knihovnu v jazyce c#, jenž bude poskytovat nástroje pro parsování zadaného příkazu SELECT a moduly, které na základě vytvořené datové struktury shromáždí informace o tomto dotazu. K textu práce se váže především komponent *SelectInfo*, jenž by dle zadání měl zvážit vhodnost použití indexů. Aby bylo možné ověřit funkčnost knihovny, požadavkem také bylo vytvořit jednoduchou

GUI aplikaci, která umožní zadat daný vstupní dotaz a následně přehledně zobrazí výstupy činností uvedených algoritmů.

1 Databázové optimalizační statistiky

Tato kapitola vychází zejména ze zdrojů [2], [3] a [15]. Optimalizační statistiky jsou údaje o databázi a jejich vybraných objektech uložené zpravidla v datovém slovníku databáze. Konkrétně zahrnují především charakteristiky uvedené v následující tabulce (Tabulka 1).

Tabulka 1 - Základní databázové optimalizační statistiky

Typ	Charakteristika	Možnost zobrazení	
		Sloupec	Pohled
Tabulkové statistiky	Počet řádků	NUM_ROWS	ALL_TAB_STATISTICS DBA_TAB_STATISTICS USER_TAB_STATISTICS DBA_ALL_TABLES
	Počet bloků	BLOCKS	
	Průměrná délka řádku	AVG_ROW_LEN	
Statistiky sloupců	Počet jedinečných hodnot	NUM_DISTINCT	ALL_TAB_COL_STATISTICS DBA_TAB_COL_STATISTICS USER_TAB_COL_STATISTICS ALL_TAB_COLUMNS
	Počet hodnot null	NUM_NULLS	
	Rozdělení dat - histogram	-	ALL_TAB_HISTOGRAMS DBA_TAB_HISTOGRAMS USER_TAB_HISTOGRAMS
	Rozšířené statistiky	-	ALL_STAT_EXTENSIONS DBA_STAT_EXTENSIONS USER_STAT_EXTENSIONS
Statistiky indexů	Počet listových bloků	LEAF_BLOCKS	ALL_IND_STATISTICS DBA_IND_STATISTICS USER_IND_STATISTICS
	Úrovně	BLEVEL	
	Clustering faktor	CLUSTERING_FACTOR	
Systémové statistiky	I/O výkon a využití	-	V$SYSSTAT V$SESSTAT
	CPU výkon a využití	-	

Další pohledy datového slovníku, které obsahují statistické údaje (namísto prefixu DBA může případně být USER nebo ALL):

- DBA_TABLES a DBA_OBJECT_TABLES,
- DBA_TAB_COLS,
- DBA_INDEXES,
- DBA_CLUSTERS,
- DBA_TAB_PARTITIONS a DBA_TAB_SUBPARTITIONS,
- DBA_IND_PARTITIONS a DBA_IND_SUBPARTITIONS,
- DBA_PART_COL_STATISTICS,
- DBA_PART_HISTOGRAMS,
- DBA_SUBPART_COL_STATISTICS,
- DBA_SUBPART_HISTOGRAMS.

Význam těchto informací spočívá v jejich využití při výběru optimálního exekučního plánu příkazu optimalizátorem (viz kapitola 2 Exekuční plán). Čím více údajů algoritmus uvažuje, tím kvalitnější je logicky i jeho výstup. Samozřejmě není důležitý jen objem dat, ale také to, zda jsou pravdivá. Stejně tak, jako může být nalezen nevhodný plán v případě

úplné absence statistik, mohou volbu optimalizátoru znehodnotit i zastaralé statistiky, které již necharakterizují skutečný stav databáze.

Optimalizační statistiky hrají v rozhodování optimalizátoru zásadní roli. Pokud neexistují nebo nejsou autentické, je jeho činnost značně znehodnocena, neboť dochází k chybnému vyhodnocení podmínek v dotazu, následuje nevhodná volba přístupových cest, pořadí spojení tabulek, atd. V konečném důsledku dojde k mylnému stanovení ceny a samotného exekučního plánu.

Problematika shromažďování a aktualizace statistik je popsána v následujícím oddíle.

1.1 Sběr optimalizačních statistik

Při sběru se využívá několik přístupů. Jak je v této práci častokrát uváděno, vývoj databáze Oracle (a nejen jí) směřuje k určité automatizaci úkonů a také obnova statistik je zpravidla prováděna automaticky během předem naplánované údržby systému. Ovšem v některých uvedených případech je toto řešení nevhodné či nemožné.

1.1.1 Sběr pomocí dynamického vzorkování (dynamic sampling)

Výskyt stavů, kdy pro optimalizační krok zpracování příkazu nejsou k dispozici nutně potřebné statistiky, je vyřešen jejich shromážděním pomocí dynamického vzorkování. Tento proces nelze používat v případech vzdálených (*remote*) a externích tabulek, pro které jsou v uvedené situaci použity defaultní statistiky. Pokud se tedy nejedná o objekt některého z těchto druhů a nakonfigurovaná úroveň dynamického vzorkování (viz dále) je dostatečně vysoká, je v případě absence nezbytných statistik během procesu optimalizace vždy proveden jejich sběr. V opačném případě jsou optimalizátoru k dispozici pouze defaultní statistiky uvedené v tabulkách níže (Tabulka 2 a Tabulka 3).

Tabulka 2 - Defaultní tabulkové statistiky

Tabulková statistika	Defaultní hodnota používaná optimalizátorem
Mohutnost	num_of_blocks * (block_size - cache_layer) / avg_row_len
Průměrná délka řádku	100 bytů
Počet bloků	100 nebo skutečná hodnota založená na rozsahu mapy
Vzdálená mohutnost	2000 řádků
Průměrná délka vzdálených řádků	100 bytů

Tabulka 3 - Defaultní statistiky indexů

Statistika indexu	Defaultní hodnota používaná optimalizátorem
Počet úrovní	1
Počet podřízených bloků	25
Počet podřízených bloků/klíč	1
Počet datových bloků/klíč	1
Počet jedinečných klíčů	100
Clustering faktor	800

Důležité je, že chybějící statistiky získané touto metodou nejsou úplné a zpravidla ani tak přesné, jako v případě výsledků ostatních způsobů sběru (viz dále). Tento fakt je dán kompromisem mezi snahou o dostatečnou podporu rozhodování optimalizátoru a minimalizací doby zpracování dotazu.

Dynamické vzorkování může být použito, i když statistiky existují. Mechanismus určující vhodnost jeho uplatnění je poměrně složitý. Pro paralelní dotaz systém předpokládá, že jeho provádění bude nákladné a vyplatí se proto investovat čas nutný pro vzorkování. Dále, pokud jsou stávající statistiky shledány nedostatečnými, lze tímto způsobem optimalizátoru pomoci. Například v případě výskytu komplexního predikátu, pro který ovšem neexistují rozšířené statistiky (viz pododdíl 1.2.7) nesoucí užitečnou informaci, lze dle dokumentace Oracle [2] absenci těchto charakteristik kompenzovat dynamickým vzorkováním. Naopak u jednorázových OLTP dotazů, které jsou prováděny sériově, je dynamické vzorkování zpravidla nic nepřinášející ztrátou času.

Úrovně dynamického vzorkování

Databáze Oracle rozlišuje 11 úrovní parametru OPTIMIZER_DYNAMIC_SAMPLING, které specifikují:

- kdy je dynamické vzorkování prováděno,
- velikost vzorku (*sample size*), jedná se o počet bloků objektu (tabulky nebo indexu), které optimalizátor zahrnuje do procesu shromažďování statistik.

Defaultní úroveň je 2. Úplné potlačení vzorkování je vynuceno pouze, pokud je nastavena nejnižší úroveň 0. Se zvyšováním této hodnoty roste parametr *sample size* a také okruh podmínek, pro které je provedeno. Pokud platí:

OPTIMIZER_DYNAMIC_SAMPLING=10,

dojde k dynamickému vzorkování při zpracování všech příkazů a optimalizátor při něm bude pracovat se všemi bloky objektu.

1.1.2 Automatický sběr statistik během údržby infrastruktury

Pro tabulky, jejichž objem a profilový obsah je poměrně ustálený, lze shromažďovat charakteristiky v rámci úloh automatické údržby infrastruktury během tzv. *maintenance windows*. Tyto „okna údržby“ jsou předem naplánované a pravidelně prováděné sekvence úloh, které mohou dále například vyhledávat databázové segmenty vhodné pro defragmentaci, spouštět uživatelem definované procedury, nebo identifikovat systémově nákladné SQL příkazy (viz pododdíl 6.3.1 Automatic SQL Tuning Advisor).

Zastaralé statistiky vhodné pro obnovu jsou rozlišeny pomocí funkce monitorování modifikací, jejíž aktivita je podřízena nastavení parametru STATISTICS_LEVEL. Pokud je jeho hodnota nastavena na defaultní, tedy TYPICAL, nebo na ALL, je monitoring aktivní a od poslední aktualizace shromažďuje charakteristiky:

- přibližný počet nově vložených řádků (operací INSERT),
- přibližný počet editovaných řádků (operací UPDATE),
- přibližný počet odstraněných řádků (operací DELETE),
- informace o provedení příkazu TRUNCATE.

K daným údajům lze přistoupit prostřednictvím pohledu USER_TAB_MODIFICATIONS v datovém slovníku. Pokud je naopak parametru STATISTICS_LEVEL přiřazena hodnota BASIC, automatický sběr tyto údaje k dispozici nemá a nedokáže tak rozpoznat neaktuální statistiky. V tomto případě je třeba proces provést manuálně.

Povolení automatického sběru lze provést následovně:

```
BEGIN
  DBMS_AUTO_TASK_ADMIN.ENABLE(
    client_name => 'auto optimizer stats collection',
    operation => NULL,
    window_name => NULL);
END;
```

Zakázání je analogické:

```
BEGIN
  DBMS_AUTO_TASK_ADMIN.DISABLE(
    client_name => 'auto optimizer stats collection',
    operation => NULL,
    window_name => NULL);
END;
```

Konfigurace úloh údržby je v režii administrátora. Zpravidla jsou plánovány na období, kdy je systém minimálně zatížen běžným provozem - tedy například v pozdních nočních hodinách. Z toho vyplývá zásadní nedostatek používání automatického sběru: tedy, že podoba obsahu specifických typů tabulek může být v tomto období velmi odlišná od té ve chvílích běžného provozu, případně se mezi jednotlivými intervaly aktualizací rapidně mění. Existují i typy statistik, které - nehledě na jejich autenticitu - jednoduše není umožněno shromáždit automaticky v rámci okna údržby. Jedná se o statistiky systémové a fixních objektů (viz podtituly GATHER_SYSTEM_STATS a GATHER_FIXED_OBJECTS_STATS v pododdíle 1.2.1). V těchto situacích je žádoucí nespoléhat na automatizaci a využít možnosti manuálního sběru statistik prostřednictvím procedur balíčku DBMS_STATS, které jsou popsány dále v oddíle 1.2.

1.1.3 Manuální sběr statistik

Jak je uvedeno v pododdíle 1.1.1, sběr statistik je možné uskutečňovat dynamicky při provádění každého dotazu nad danými tabulkami. Proces se tak stává součástí kroku optimalizace zpracování příkazu (viz kapitola 3 Zpracování příkazu). K docílení tohoto chování je třeba nastavit parametr OPTIMIZER_DYNAMIC_SAMPLING na defaultní hodnotu 2 nebo vyšší a pomocí příkazů:

```
BEGIN
  DBMS_STATS.DELETE_TABLE_STATS('OWNER_NAME', 'TABLE_NAME'');
  DBMS_STATS.LOCK_TABLE_STATS('OWNER_NAME','TABLE_NAME');
END;
```

odstranit a uzamknout dané statistiky - tedy zabránit jejich sběru během provádění údržby.

Pokud je znám reprezentativní vzor, jehož odlišnost od obsahu tabulky ve všech daných okamžicích provozu systému je považována za přijatelnou, lze uchovat statistiky pro tento vzor a poté opět zakázat jejich aktualizaci. Podobné řešení se nabízí i u tabulek, které jsou plněny či modifikovány dávkovými operacemi. Manuální sběr je prováděn vždy na závěr dané operace jako její součást.

1.2 Balíček DBMS_STATS

Tento balíček obsahuje procedury pro práci s optimalizačními statistikami všech typů uvedených v tabulce 1. Paleta činností, které pomocí něho lze vykonat, je velmi široká. Neomezuje se pouze na tvorbu, odstraňování a zobrazení - statistiky lze také importovat, uzamykat, porovnávat, obnovovat atd. Pomocí těchto nástrojů lze pracovat s údaji nejen v jejich standardním úložišti, tedy v datovém slovníku, ale statistiky mohou být ukládány i v prostoru uživatelských tabulek. Možnosti balíčku jsou dále popsány podrobněji.

1.2.1 Sběr optimalizačních statistik pomocí procedur balíčku DBMS_STATS

Pokud je třeba shromáždit statistiky mimo rámec úloh automatické údržby, nabízí nástroj DBMS_STATS následující procedury:

- GATHER_DATABASE_STATS,
- GATHER_DICTIONARY_STATS,
- GATHER_FIXED_OBJECTS_STATS,
- GATHER_INDEX_STATS,
- GATHER_SCHEMA_STATS,
- GATHER_SYSTEM_STATS,
- GATHER_TABLE_STATS,
- GENERATE_STATS.

GATHER_DICTIONARY_STATS

Statistiky objektů ve schématech datového slovníku SYS, SYSTEM a RDBMS komponentů. K provedení procedury je třeba oprávnění SYSDBA nebo ANALYZE ANY DICTIONARY a ANALYZE ANY zároveň.

GATHER_FIXED_OBJECTS_STATS

Statistiky „pevných“ objektů obsahujících informace o parametrech a událostech v systému. Tyto údaje jsou aktualizovány průběžně během činnosti databáze, proto se dané objekty nazývají dynamické výkonové tabulky, které lze zobrazit pomocí pohledu V$FIXED_TABLE. Tyto tabulky jsou virtuální - z hlediska jejich paměťového uložení se

tedy nejedná o klasické databázové tabulky. Jako fixní jsou označovány, protože nemohou být odstraněny a také jejich struktura je neměnná.

Využití procedury je opět podmíněno vlastnictvím potřebných práv, v tomto případě oprávnění SYSDBA nebo ANALYZE ANY DICTIONARY.

GATHER_SYSTEM_STATS

Systémové statistiky popisují charakteristiky hardware systému, kterými jsou například rychlost CPU, propustnost a rychlost přenosu I/O, doba čtení bloku, atd. Tyto údaje nelze aktualizovat automaticky a v dokumentaci systému Oracle [2] je důrazně doporučeno provádět tento proces manuálně pomocí procedury GATHER_SYSTEM_STATS, neboť poté má optimalizátor přesnější informace pro odhad nákladů na dotaz, potažmo pro určení výsledného exekučního plánu.

Statistiky systému jsou rozděleny dle způsobu jejich zaznamenání. První kategorií jsou údaje typu *noworkload*, jejichž hodnoty jsou určeny pomocí náhodných čtení datových souborů. Naopak statistiky *workload* jsou měřeny v určitém intervalu skutečného provozu databázového systému. V případě vhodné volby tohoto časového úseku přináší daný přístup reálnější data, která jsou využívána přednostně před charakteristikami prvně uvedeného typu. Statistiky *noworkload* jsou tedy využity jen v případech, kdy *workload* nejsou k dispozici.

GATHER_TABLE_STATS

Statistiky tabulek, jejich sloupců a příslušných indexů.

GENERATE_STATS

Tato procedura umožňuje generovat statistky indexů z již nashromážděných statistik objektů, od kterých jsou odvozeny. Lze ji aplikovat na vyhledání údajů indexů bitmapových a klasických typu *b-tree*.

1.2.2 Přenos

Zejména pro možnost přenosu statistik produkční databáze na odpovídající testovací systém jsou užitečné exportní a importní procedury. Využití naleznou i v rámci jedné databáze. Jelikož optimalizátor pracuje pouze se statistikami v datovém slovníku, ty, které jsou uloženy v uživatelských tabulkách, je pro jejich využití třeba do tohoto prostoru importovat. Mezi tyto procedury například patří:

- EXPORT_TABLE_STATS,
- IMPORT_TABLE_STATS,
- EXPORT_DATABASE_STATS,
- IMPORT_ DATABASE_STATS.

1.2.3 Uzamknutí

Uzamknutí je využíváno zpravidla v případech, kdy je současná podoba statistik považována za optimální a nepředpokládá se zásadní změna objektů, které charakterizují (viz pododdíl 1.1.3). Dané statistiky v tomto stavu nemohou být modifikovány. Pokud jsou zmraženy charakteristiky tabulky, vztahuje se zámek i na statistiky jejich sloupců a k nim vztažených indexů. Balíček DBMS_STATS poskytuje šest procedur:

- LOCK_SCHEMA_STATS,
- LOCK_TABLE_STATS,
- LOCK_PARTITION_STATS,
- UNLOCK_SCHEMA_STATS,
- UNLOCK_TABLE_STATS,
- UNLOCK_PARTITION_STATS.

1.2.4 Obnova

Pokud aktualizace zapříčiní zhoršení práce optimalizátoru, lze i modifikované statistiky obnovit do jejich původní podoby, podobně jako samotná data tabulek. Procedury pro obnovy vyžadují jako parametr časový údaj typu TIMESTAMP a anulují všechny změny od tohoto milníku. Daný údaj lze získat pomocí pohledů:

- DBA_OPTSTAT_OPERATIONS,
- ALL_TAB_STATS_HISTORY,
- DBA_TAB_STATS_HISTORY,
- USER_TAB_STATS_HISTORY,

obsahujících historii operací se statistikami, respektive historii úprav tabulkových statistik. Dle dokumentace [2] jsou v defaultním nastavení statistiky pro obnovu k dispozici 31 dní od jejich modifikace, poté jsou během automatického procesu odstraněny. Dostupné procedury jsou:

- RESTORE_DICTIONARY_STATS,
- RESTORE_FIXED_OBJECTS_STATS,
- RESTORE_SCHEMA_STATS,
- RESTORE_SYSTEM_STATS,
- RESTORE_TABLE_STATS,
- ALTER_STATS_HISTORY_RETENTION,
- PURGE_STATS,
- GET_STATS_HISTORY_AVAILABILITY.

RESTORE_DICTIONARY_STATS

Obnova statistik objektů ve schématech datového slovníku SYS, SYSTEM a RDBMS komponentů. K provedení procedury je třeba oprávnění SYSDBA nebo ANALYZE ANY DICTIONARY a ANALYZE ANY zároveň.

RESTORE_TABLE_STATS

Obnovení statistik tabulek, jejich sloupců a příslušných indexů.

ALTER_STATS_HISTORY_RETENTION

Nastavení časové hodnoty, po kterou budou k dispozici statistiky pro obnovu. Defaultní doba je 31 dní. Údaj lze ověřit pomocí funkce GET_STATS_HISTORY_RETENTION.

PURGE_STATS

Odstranění návratových statistik vytvořených před zadanou časovou hranicí. K provedení procedury je třeba oprávnění SYSDBA nebo ANALYZE ANY DICTIONARY a ANALYZE ANY zároveň

GET_STATS_HISTORY_AVAILABILITY

Funkce vracející nejstarší časové razítko návratových statistik.

1.2.5 Neověřené (pending) statistiky

Od verze Oracle 11.2 existuje možnost uložit nově shromážděné statistiky jako neověřené a tím potlačit jejich použití během procesu optimalizace. Význam tohoto odlišení nově získaných údajů spočívá v možnosti ověření jejich přínosu pro rozhodování optimalizátoru. Tuto validaci lze provést na samotném provozním systému nebo v testovacím prostředí, kam jsou zkušební statistiky exportovány. *Pending* charakteristiky tabulek a indexů lze zobrazit pomocí pohledů:

- USER_TAB_PENDING_STATS a
- USER_IND_PENDING_STATS.

Protipólem jsou statistiky označované jako *publish*, uložené v pohledech:

- USER_TAB_STATISTICS a
- USER_IND_STATISTICS.

Systém defaultně pracuje v režimu, kdy nové statistiky ihned označí jako *publish* a automaticky je začne využívat v optimalizačním procesu. Uvedené nastavení platí, pokud následující dotaz vrací TRUE.

SELECT DBMS_STATS.GET_PREFS('PUBLISH') PUBLISH FROM DUAL;

V případě návratové hodnoty FALSE jsou čerstvé statistky označeny jako *pending* a v tomto stavu setrvají, dokud nejsou odstraněny či převedeny na *publish* příslušnou procedurou.

Pro tuto problematiku disponuje balíček následujícími procedurami:

- DELETE_PENDING_STATS,

- PUBLISH_PENDING_STATS,
- EXPORT_PENDING_STATS.

DELETE_PENDING_STATS

Odstranění *pending* statistik.

PUBLISH_PENDING_STATS

Změna *pending* statistik na *publish*.

EXPORT_PENDING_STATS

Umožňuje exportovat *pending* statistiky do dané cílové tabulky, která leží například v testovacím systému.

1.2.6 Porovnávání

Balíček DBMS_STATS poskytuje možnost porovnání statistik umístěných ve dvou různých úložištích. Využít lze funkce, které v případě odlišnosti daných vstupů vracejí sjednocení statistik obou těchto množin:

- DIFF_TABLE_STATS_IN_PENDING,
- DIFF_TABLE_STATS_IN_STATTAB,
- DIFF_TABLE_STATS_IN_HISTORY.

1.2.7 Rozšířené statistiky

Jako rozšířené statistiky jsou označovány dva druhy těchto údajů:

- Vícesloupcové statistiky;
- Statistiky výrazu.

Vícesloupcové statistiky

Volbu optimálního plánu dotazu, který v klausuli WHERE obsahuje více vzájemně závislých sloupců jedné tabulky, lze podpořit vytvořením souhrnné statistiky nad všemi těmito sloupci. Takový údaj představuje pro optimalizátor užitečnou informaci o korelaci, jaké se mu od jednoduchých statistik nedostává.

Statistiky výrazu

Častým jevem je výskyt funkce v predikátu dotazu. Jejím parametrem může být sloupec tabulky. Takové případy jsou pro optimální provedení dotazu poměrně nepříznivé a Oracle vyvinul opatření, která mají v této situaci pomoci. Kromě případného zefektivnění přístupu k datům definováním funkčních indexů jde právě o možnost vytvoření statistik výrazu. Tu lze využít v případě, kdy podoba výskytu funkčního výrazu odpovídá následujícímu vzoru:

FUNKCE (SLOUPEC) = KONSTANTA.

Jelikož funkce daný sloupec „obaluje“, optimalizátor nemůže z jeho běžných charakteristik vyvodit míru selektivity této podmínky a je vhodné vytvořit rozšířenou statistiku, která bude poskytovat informaci o návratových hodnotách funkce aplikované na tento sloupec.

Rozšířené statistiky definované v systému lze zobrazit pomocí pohledu datového slovníku USER_STAT_EXTENSIONS. Sběr lze provádět, obdobně jako u statistik jednoduchých, automaticky nebo manuálně. Pro jejich správu slouží následující nástroje:

- CREATE_EXTENDED_STATS,
- SHOW_EXTENDED_STATS_NAME,
- DROP_EXTENDED_STATS.

CREATE_EXTENDED_STATS

Funkce vytvoří dle zadaných parametrů statistiku výrazu nebo vícesloupcovou. Umožnění provedení této operace podléhá několika podmínkám a omezením. Návratovou hodnotou je jméno nově vytvořeného statistického záznamu.

SHOW_EXTENDED_STATS_NAME

Zjištění jména rozšířené statistiky po dosazení údajů, které ji charakterizují. Pokud taková neexistuje, dojde k vyvolání chyby.

DROP_EXTENDED_STATS

Procedura má obdobnou funkčnost jako SHOW_EXTENDED_STATS_NAME, ale dojde k odstranění nalezené statistiky.

1.2.8 Další možnosti balíčku DBMS_STATS

Kromě výše popsaných činností umožňuje balíček DBMS_STATS ještě řadu dalších možností spojených s:

- nastavením, odstraněním a zobrazením statistik,
- uživatelsky definovanými statistikami.

1.3 Náklady na správu statistik

1.3.1 Vzorkování

Oracle pro sběr statistik používá technologii vzorkování. Už z označení je patrné, že technika nepracuje se všemi řádky v tabulkách, ale pouze s jejich částí. Nedochází tak k úplnému průchodu a řazení tabulek.

Při dynamickém vzorkování určuje podíl zahrnutých bloků objektu nastavení parametru OPTIMIZER_DYNAMIC_SAMPLING (viz pododdíl 1.1.1). Také procedury balíčku DBMS_STATS pro shromažďování statistik obsahují parametr ESTIMATE_PERCENT představující procentuální hodnotu objemu řádků, které budou během procesu využity. Oracle disponuje mechanismy na určení hodnoty této veličiny, která je optimální vzhledem

k vyváženému poměru mezi přesností statistik a náklady na jejich určení. Doporučuje se je využívat, tedy použít pro daný parametr konstantu DBMS_STATS.AUTO_SAMPLE_SIZE.

1.3.2 Paralelní sběr statistik

Další zefektivnění představuje paralelní sběr. Podobně jako výše uvedený parametr ESTIMATE_PERCENT, obsahují procedury nástroje DBMS_STATS také vstupní argument DEGREE. Opět je doporučeno využívat konstantu DBMS_STATS.AUTO_DEGREE, pro kterou procedura využije algoritmy k určení vhodného stupně paralelizmu na základě velikosti objektu, počtu CPU a inicializačních parametrů.

1.3.3 Statistiky rozdělených (partitioned) objektů

V pododdíle 5.2.3 je stručně popsán způsob, jímž lze rozdělit tabulky (a jejich indexy) do oddílů dle požadovaných kritérií. Běžně se využívá dělení na časové období, kdy jsou například obchodní údaje firmy ukládány v sekcích představujících jednotlivé kvartály a dotazy na záznamy z konkrétního čtvrtletí lze poté provádět efektivněji. Oddíly mohou být dále děleny na pododdíly a tvořit tak hierarchii, tento stav se označuje jako kompositní dělení (*composite partitioning*).

Obdobně i statistiky těchto objektů jsou logicky děleny a lze pracovat s údaji separátně charakterizujícími daný pododdíl, oddíl, atd. V procedurách sběru lze specifikovat argument GRANULARITY, čímž definujeme, zda chceme například získat pouze globální statistiky tabulky, nebo i specifičtější charakteristiky oddílů. Defaultní hodnota je AUTO, pro kterou opět necháváme rozhodnutí na systému Oracle, jenž sám určí granularitu maximální vzhledem k použitému dělení. Dále je možné použít například variantu ALL, čímž proceduru instruujeme ke sběru statistik všech pododdílů, oddílů i globálních charakterizujících kompletní objekt.

Pokud jsou přidávána nová data v době, kdy již existují oddíly s odpovídajícími statistikami, zpravidla nenáleží do všech těchto stávajících sekcí a není tedy třeba aktualizovat kompletní soubor statistik. Lze využit argumentu procedur INCREMENTAL, který, pokud je TRUE, vede k přírůstkové aktualizaci záznamů.

2 Exekuční plán

Tato kapitola vychází především ze zdrojů [4], [5], [6] a [7]. Exekuční plán je návod na provedení příkazu vytvořený optimalizátorem. Představuje konkrétní postup určující, ve kterém pořadí a jakým způsobem budou provedeny potřebné operace, jako je například načtení záznamů z tabulky, spojení dvou množin dat či jejich seřazení.

Pro zvýšení efektivity samotného provedení příkazu má zásadní význam právě volba vhodného plánu, která se tak stává jedním ze základních konceptů optimalizace v systému Oracle.

Hojné využití mají nástroje na zobrazení plánu vybraného dotazu, které umožňují analyzovat a testovat chování optimalizátoru. Těmto možnostem je dále věnován příslušný oddíl. Nejprve však budou popsány základní principy exekučního plánu, potažmo přístupu k získávání a formování dat pro výsledek dotazu.

2.1 Princip exekučního plánu

Plán má podobu stromu, jenž se označuje jako strom řádkových zdrojů (*row source tree*). Tuto hierarchii tvoří jednotlivé řádkové zdroje, přičemž každý z nich odpovídá určité operaci v procesu provádění příkazu a poskytuje nějakou množinu řádků, která je buď dále zpracována rodičovským řádkovým zdrojem, nebo je vrácena jako výsledný výstup příkazu.

Vstupem každé položky stromu řádkových zdrojů tedy může být buď výsledek práce jejího potomka, nebo záznamy získané čtením z objektů databáze. Datové operace, které exekuční plán rozlišuje, jsou následující:

- čtení z tabulek (případně indexů),
- spojování množin,
- agregace,
- filtrování,
- řazení.

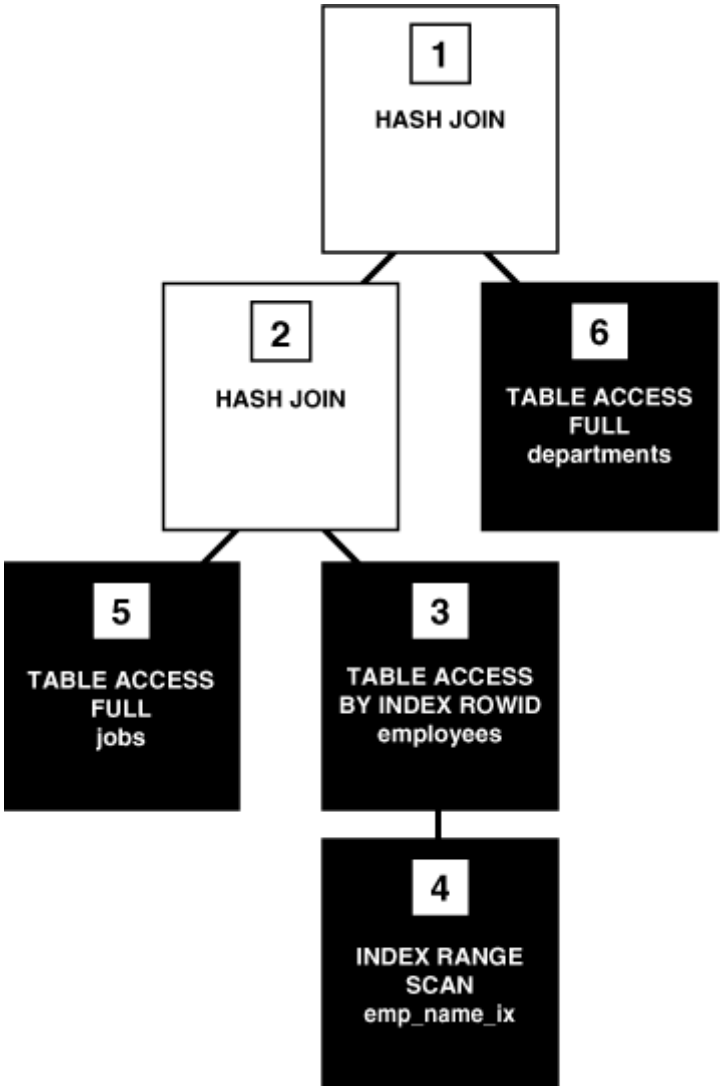
Dále plán poskytuje řadu informací charakterizujících tyto výše uvedené úkony:

- přístupové metody použité pro čtení z daných tabulek,
- způsob a pořadí spojení,
- optimalizační údaje - cena operace, kardinalita,
- dělení (*partitioning*),
- paralelní zpracování.

Samotné provádění exekučního plánu probíhá od listů stromu řádkových zdrojů k jeho kořeni. Například pro dotaz:

```
SELECT e.last_name, j.job_title, d.department_name
FROM   hr.employees e, hr.departments d, hr.jobs j
WHERE  e.department_id = d.department_id
AND    e.job_id = j.job_id
AND    e.last_name LIKE 'A%';
```

může optimalizátor a generátor řádkového zdroje (viz oddíl 3.4) stanovit průběh dle stromu zobrazeného na následujícím obrázku. Odpovídající exekuční plán je uveden a popsán dále v pododdíle 2.4.1 Orientace v exekučním plánu.



Obrázek 1 - Strom řádkových zdrojů, převzato z [4]

2.2 Zobrazení exekučního plánu

Během práce v prostředí specializovaného softwaru, jakým je například SQL Developer, lze zobrazit plán v rámci GUI dané aplikace. Tento způsob je uživatelsky přívětivý, ovšem na pozadí zpravidla používá základní nástroje systému Oracle, jejichž možnosti je tedy vhodné podrobněji popsat.

2.2.1 Explain plan

Nástroj EXPLAIN PLAN umožňuje pro příkazy SELECT, INSERT, UPDATE a DELETE zobrazit exekuční plán, dle kterého by byly v daný okamžik provedeny. Samotný příkaz tento plán pouze uloží do tabulky, kterou je zpravidla defaultní PLAN_TABLE. V tomto případě je tato tabulka vytvořena automaticky při volání příkazu. Plán lze použitím varianty EXPLAIN PLAN INTO vložit i do předem vytvořené uživatelské tabulky, jejíž struktura má odpovídající podobu.

PLAN_TABLE$

Jedná se o globální dočasnou tabulku umístěnou ve schématu datového slovníku SYS, do které je defaultně a transparentně ukládán výstup příkazu EXPLAIN PLAN. Pro přístup k ní se často využívá její veřejné synonymum PLAN_TABLE, které je rovněž vytvářeno automaticky. Řádek tabulky odpovídá konkrétní operaci exekučního plánu (tedy konkrétnímu řádkovému zdroji) a uchovává údaje uloženy ve 36 sloupcích, mezi které patří například:

- *STATEMENT_ID* - Volitelná hodnota, kterou lze zadat během provádění příkazu EXPLAIN_PLAN. Defaultně je null. Tento identifikátor usnadňuje orientaci v tabulce, případně jej lze využít například v podmínce pro filtrování výstupu dotazu na PLAN_TABLE atd.
- *ID* - identifikátor jednotlivých kroků v rámci exekučního plánu.
- *PARENT_ID* - identifikátor nadřazené operace, v rámci které je krok prováděn.
- *OPERATION* - operace uskutečňovaná krokem.
- *OPTIONS* - rozšiřující informace o dané operaci.
- *OBJECT_NAME* - název tabulky nebo indexu.
- *CARDINALITY* - odhadovaný počet řádků zpracovaných během operace.
- *BYTES* - odhadovaný počet bytů zpracovaných během operace.
- *CPU_COST* - odhad zatížení CPU.
- *IO_COST* - odhad I/O nákladů.
- *COST* - cena operace určená na základě hodnot CPU_COST a IO_COST.
- *TIME* - odhadovaná délka operace v sekundách.
- *ACCESS_PREDICATES* - podmínky, dle nichž se prostřednictvím indexů vyhledávají řádky během jejich načítání z databáze.
- *FILTER_PREDICATES* - podmínky použité k selekci již zpřístupněných řádků.

Pro uživatelsky přívětivé zobrazení záznamů uložených v PLAN_TABLE$ lze využít následující nástroje:

- *UTLXPLS.SQL* - skript, jenž zobrazí plán příkazu.
- *UTLXPLP.SQL* - skript, jenž zobrazí plán rozšířen o informace vztažené k případnému paralelnímu zpracování.
- *DBMS_XPLAN.DISPLAY* - tabulková funkce umožňující přizpůsobit formát zobrazení použitím vstupních parametrů.

Nástrojem EXPLAIN_PLAN lze zobrazit pouze plán, který by v okamžik jeho použití systém zvolil pro realizaci příkazu. Samotný příkaz tak prováděn není. Oracle ovšem také disponuje možnostmi pro zobrazení plánu, dle kterého byl příkaz skutečně vykonán a umožňuje tak získat sadu informací rozšířenou o veličiny charakterizující reálný průběh operací. Jedná se o pohled V$SQL_PLAN a především ladící funkci AUTOTRACE.

2.2.2 Pohled V$SQL_PLAN

Tento pohled obsahuje plány v minulosti provedených příkazů, které jsou uloženy v rámci *shared SQL area* (viz oddíl 3.2 Kontrola sdíleného fondu). Obsahuje obdobné informace, jako tabulka PLAN_TABLE$. Další užitečné charakteristiky lze zobrazit pomocí pohledů:

- V$SQL_PLAN_STATISTICS,
- V$SQL_PLAN_STATISTICS_ALL.

2.2.3 Funkce Autotrace

Nástroj AUTOTRACE umožňuje, podobně jako EXPLAIN PLAN, zobrazit exekuční plán. Podstatný rozdíl je v tom, že při využívání této funkce dojde k provedení příkazu a mohou tak být zaznamenány i informace týkající se nákladů, které byly reálně vynaloženy. Seznam základních charakteristik převzatý z knihy Oracle - Návrh a tvorba aplikací [15] je uveden níže v tabulce (Tabulka 4).

Tabulka 4 - Statistické informace funkce AUTOTRACE

Veličina	Popis
recursive calls	Počet provedených příkazů nutných k provedení uživatelského příkazu SQL
db block gets	Celkový počet bloků načtených v aktuálním režimu z vyrovnávací mezipaměti
consistent gets	Počet konzistentních načítání ve vyrovnávací paměti vyžadovaných pro blok, konzistentní načítání může vyžadovat načtení informací potřebných k vrácení akce (undo - rollback), tato načítání budou počítána také
physical reads	Počet fyzických načtení z datových souborů do vyrovnávací mezipaměti
redo size	Celkový počet opakovaných akcí generovaných v bajtech během provádění příkazu
bytes sent via SQL*Net to client	Celkový počet bajtů odeslaných klientovi ze serveru
bytes received via SQL*Net from client	Celkový počet bajtů přijatých od klienta
SQL*Net round-trips to/from client	Celkový počet zpráv programu odeslaných klientovi a přijatých od klienta, číslo zahrnuje přenosy týkající se načítání sady výsledků s více řádky
sorts (memory)	Počet provedených setřídění v paměti relace uživatele (oblast třídění), řízeno prostřednictvím parametru databáze sort_area
sorts (disk)	Počet setřídění na disku (dočasný tabulkový prostor), protože při třídění došlo k překročení velikosti oblasti třídění uživatele
rows processed	Řádky zpracované úpravami nebo vrácené z příkazu SELECT

Aktivita probíhá transparentně - pokud je funkce povolena, během zpracovávání každého DML příkazu automaticky sbírá uvedené údaje, které následně zobrazí. Oracle 11gR2 dle dokumentace [5] rozlišuje pět režimů činnosti v závislosti na nastavení systémové proměnné AUTOTRACE:

- *SET AUTOTRACE OFF* - výchozí nastavení, funkce negeneruje žádné sestavy, dotazy jsou spouštěny obvyklým způsobem.
- *SET AUTOTRACE ON EXPLAIN* - dotazy jsou spouštěny obvyklým způsobem, sestava funkce AUTOTRACE zobrazuje pouze průběh výpočtu optimalizátoru.
- *SET AUTOTRACE ON STATISTICS* - dotazy jsou spouštěny obvyklým způsobem, sestava funkce AUTOTRACE zobrazuje pouze statistické informace týkající se provádění příkazů SQL.
- *SET AUTOTRACE ON* - dotaz je proveden a sestava funkce AUTOTRACE zahrnuje průběh výpočtu optimalizátoru i statistické informace týkající se provedení příkazů SQL.
- *SET AUTOTRACE TRACEONLY* - podobné příkazu SET AUTOTRACE ON, ale je vynechán tisk výstupu uživatelského dotazu, užitečné při ladění dotazu vracejícího klientovi rozsáhlou sadu výsledků, je výhodnější potlačit zobrazení 1000 řádků výstupu a nečekat na jejich vytištění a posun po obrazovce.

Popis jednotlivých konfigurací je převzat z knihy Oracle - Návrh a tvorba aplikací [15].

2.3 Cost based optimalizace

Jak již bylo uvedeno, optimalizátor Oracle vybírá optimální exekuční plán pro provedení příkazu. Měřítkem, dle kterého se rozhoduje, je odhad nákladů označovaný jako cena příkazu. Tomuto procesu se říká optimalizace založená na nákladech a komponent, jenž ji provádí, se nazývá CBO optimalizátor (*cost based optimizer*). Více informací o těchto pojmech se nachází v kapitole 4 Optimalizátor Oracle.

2.4 Analýza exekučního plánu

2.4.1 Orientace v exekučním plánu

Operace ve standardně formátovaném exekučním plánu probíhají v pořadí, ve kterém jsou zobrazeny, prvotně zprava doleva a - v případě shodného odsazení - shora dolů. Nejdříve jsou tedy provedeny operace s největším odsazením doprava, poté jejich méně odsazené rodičovské bloky, atd. Pokud jich je více odsazeno shodně, postupuje se shora dolů.

Například pro dotaz, uvedený v oddíle 2.1, lze tedy kupříkladu s pomocí funkce AUTOTRACE zobrazit následující plán (převzatý z dokumentace [4]):

```
Execution Plan
----------------------------------------------------------
Plan hash value: 975837011

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |     3 |   189 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN                    |             |     3 |   189 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN                   |             |     3 |   141 |     5  (20)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |     3 |    60 |     2   (0)| 00:00:01 |
|*  4 |     INDEX RANGE SCAN          | EMP_NAME_IX |     3 |       |     1   (0)| 00:00:01 |
|   5 |    TABLE ACCESS FULL          | JOBS        |    19 |   513 |     2   (0)| 00:00:01 |
|   6 |   TABLE ACCESS FULL           | DEPARTMENTS |    27 |   432 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   2 - access("E"."JOB_ID"="J"."JOB_ID")
   4 - access("E"."LAST_NAME" LIKE 'A%')
       filter("E"."LAST_NAME" LIKE 'A%')
```

Oracle tento příkaz zpracuje dle následujícího postupu:

1. *Id 4* - Čtení indexu EMP_NAME_IX během kterého jsou vyhledány ROWID řádků, jejichž LAST_NAME začíná na znak A.
2. *Id 3* - Přístup k tabulce EMPLOYEES, z níž jsou pomocí hodnot ROWID získaných v předchozím kroku s id 4 načteny požadované záznamy s LAST_NAME začínajícím na znak A.
3. *Id 5* - Plný průchod tabulkou JOBS za účelem načtení všech jejich řádků.
4. *Id 2* - Technikou *hash* jsou spojeny řádkové zdroje získané předchozími operacemi s id 3 a 5. Každému načtenému řádku z tabulky JOBS je přiřazován odpovídající záznam z řádkového zdroje 3, tedy z tabulky EMPLOYEES.
5. *Id 6* - Plný průchod tabulkou DEPARTMENTS za účelem načtení všech jejich řádků.
6. *Id 1* - Další *hash* spojení, tentokrát je řádkový zdroj 6 spojován s výsledkem první operace HASH JOIN s id 2. Každému v kroku 6 načtenému řádku tabulky DEPARTMENTS je přiřazován odpovídající záznam z řádkového zdroje s id 2.
7. *Id 0* - Kompletní příkaz. Klientovi je vrácen výsledek operace HASH JOIN s id 1.

2.4.2 Problémy a jejich řešení

Exekuční plán je pouze informativní nástroj, možné příčiny problémů s výkonem databáze prostřednictvím něj nelze přímo řešit, ale jen odhalit. Během jeho analýzy je třeba věnovat pozornost především následujícím jevům:

- Úplné prohledávání tabulky za účelem výběru malého množství záznamů a naopak využití indexů v případě malé selektivity.
- Procházení velkého rozsahu indexu za účelem výběru malého množství záznamů.
- Filtrování dat v pozdějších fázích provádění plánu. Platí, že je efektivnější z tabulky dle podmínek vybrat malé množství záznamů a s ním dále pracovat, než vybrat vše a selekci provést až například poté, co byla tato úplná množina seřazena či spojena s jinou.

- Nevhodné pořadí spojení. Tento případ je principiálně podobný předchozímu. Pokud existuje sekvence spojení, její průběh bude efektivnější, když po první operaci sloučení bude každému dalšímu kroku předán co nejmenší objem záznamů. Vždy by měly být vysoce selektivní operace prováděny v co nejranějších fázích provádění plánu.
- Výskyt nežádoucího kartézského součinu, který je pro výslednou informaci dotazu nepotřebný.
- Nevhodné metody spojení.
- Nevhodné přístupové cesty využité během spojení. Pokud se povaha spojení blíží kartézskému součinu, vzhledem k nízké selektivitě podmínky spojení není vhodné přistupovat k datům prostřednictvím indexu. Výjimkou mohou být případy, kdy je výhodné načítat záznamy v pořadí, v němž jsou v rámci indexu řazeny. Například pokud je použita metoda *sort merge join* nebo dotaz požaduje jejich seřazení.

Rozhodování optimalizátoru nejnovějších systémů je již natolik vyspělé, že tyto nežádoucí postupy jsou jím zpravidla zavrženy. Pokud je přesto daný plán shledán nevhodným, je třeba optimalizátoru nařídit, doporučit nebo umožnit, aby volil odlišný. Možnosti, jak toho docílit, jsou popsány zejména v oddíle 5.4. Jednou z variant je také využití tzv. *fixed SQL plan baselines*, které umožňují napevno přiřadit příkazu vybraný exekuční plán, dle kterého tak bude vždy prováděn (viz dále v podtitulu Fixed SQL Plan Baselines pododdílu 2.5.1).

2.5 SQL Plan management

SQL plan management je mechanismus, který průběžně sleduje, zaznamenává a vyhodnocuje provádění exekučních plánů příkazů. Účelem jeho aktivity je potlačení situací, kdy náhlé nevhodné změny exekučního plánu snižují výkon systému při provádění daného příkazu. Jedná se například o:

- Změny konfigurace systému a parametrů optimalizátoru.
- Změny optimalizačních statistik.
- Upgrade verze optimalizátoru.
- Změny schématu.
- Vytvoření SQL profilu.

Stěžejní koncept, který tento správce využívá, se označuje *SQL plan baselines*.

2.5.1 SQL Plan Baselines

Jedná se o sadu exekučních plánů, které vytváří a spravuje *SQL plan management*. Jedním ze segmentů datového slovníku systému Oracle je tzv. *SQL management base* (SMB), který mimo jiné obsahuje jednak historii všech plánů opakovaně prováděných příkazů nazývanou *SQL plan history*, a také její podmnožinu, která představuje právě *SQL plan baselines*. Do ní náleží pouze tzv. akceptované plány, u nichž se předpokládá, že jejich provádění bude efektivní.

Pokud optimalizátor vytvoří nový plán, který se v *SQL plan baselines* zatím nenachází, porovnává jej s těmi, které již obsaženy jsou. Když nalezne lepší plán, pak jej použije místo nově vytvořeného. Tyto uložené postupy lze tedy označit jako soubor vhodných řešení procesu volby plánu, které udržují pomyslnou laťku kvality zvoleného exekučního plánu. Aktuální *baselines* lze zobrazit prostřednictvím pohledu datového slovníku DBA_SQL_PLAN_BASELINES nebo využitím balíčku DBMS_XPLAN a jeho funkce DISPLAY_SQL_PLAN_BASELINE:

```
SELECT * FROM TABLE(
    DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE(
        sql_handle=>'SYS_SQL_209d10fabbedc741',
        format=>'basic'));
```

Problematiku mechanismů používání *plan baselines* lze rozdělit do následujících úkonů:

- Tvorba;
- Výběr;
- Vývoj.

Tvorba

Položku *SQL plan baseline* lze zachytávat automaticky, nebo vytvářet manuálně. Automatické zachytávání je defaultně neaktivní, jelikož platí konfigurace:

OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES = FALSE.

Pro povolení funkce je tedy třeba změnit hodnotu parametru na TRUE. Během aktivity automatického zachytávání je při prvním provádění konkrétního příkazu použitý plán vždy označen jako akceptovaný a uložen jako *SQL plan baseline*. Při dalších zpracováních totožného příkazu probíhají dále uvedené fáze volby a vývoje, tak aby byly používány a v *SQL plan baseline* udržovány jen efektivní plány.

V případě jednorázového manuálního vkládání je daný plán považován za ověřený a systém tak žádnou kontrolu neprovádí. Záznam lze načíst buď z *SQL tuning sets* (viz oddíl 6.5) a *AWR snapshots* (viz oddíl 6.1), nebo z umístění v *shared SQL area* (viz oddíl 3.2 Kontrola sdíleného fondu).

Pro import plánu z *SQL Tuning Sets* slouží funkce LOAD_PLANS_FROM_SQLSET balíčku DBMS_SPM:

```
DECLARE
  my_plans PLS_INTEGER;
BEGIN
  my_plans := DBMS_SPM.LOAD_PLANS_FROM_SQLSET( sqlset_name => 'tset1');
END;
```

Obdobný postup lze použít, i když se potřebný plán nachází v *AWR snapshots*. Tehdy je třeba jej nahrát do *SQL tuning sets* a poté použít výše uvedenou rutinu.

Funkce, která umožňuje vložení plánu z *library cache*, konkrétněji z její podoblasti *shared SQL area*, je LOAD_PLANS_FROM_CURSOR_CACHE. Poskytuje ji také balíček DBMS_SPM a rovněž použití je analogické. Plán je jen třeba identifikovat pomocí SQL_ID nebo SQL_TEXT namísto SQLSET_NAME:

```
DECLARE
  my_plans PLS_INTEGER;
BEGIN
  my_plans := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE( sql_id =>
'99twu5t2dn5xd');
END;
```

Výběr

Poté, co optimalizátor sestaví a dle ceny vybere exekuční plán, dochází k prohledávání *SQL plan baselines*.

Když *SQL plan baselines* neobsahují položku pro zpracovávaný příkaz, optimalizátoru nezbývá jiná možnost, než použít nově vytvořený plán, který je poté do *baselines* uložen jako akceptovaný.

Pokud pro daný příkaz záznam existuje a odpovídá nově sestavenému plánu, je tento plán použit pro provedení příkazu.

V opačném případě je nově vytvořený plán uložen do historie v *SQL management base* a označen jako neakceptovaný. Tento přívlastek mu může být změněn pouze v případě úspěšného ověření v rámci vývoje (viz dále). Poté optimalizátor dle nejnižší ceny, kterou stanoví v daný okamžik, vybere z *SQL plan baselines* nejlepší plán a ten předá dále k provedení.

Mechanismus je ošetřen i vůči změnám v systému, které mohou jeho prospěšnost degradovat. Například odstranění indexu, se kterým postupy uložené v *baselines* počítali, způsobí jejich nereprodukovatelnost a do procesu výběru nejsou zapojeny. Naopak se jej v takových případech zúčastní i nově vytvořený plán, který byl sestaven v již změněném prostředí, a pokud má nejnižší cenu právě on, bude použit.

Vývoj

Proces vývoje spočívá nejen ve vytváření *SQL plan baselines*, ale především v udržování a zvyšování kvality těchto exekučních plánů.

Pro tento úkol disponuje systém Oracle funkcí EVOLVE_SQL_PLAN_BASELINE balíčku DBMS_SPM. Jak již bylo uvedeno výše, pokud v *SQL plan baselines* během optimalizace již existuje záznam pro daný příkaz, je nově vytvořený plán uložen jako neakceptovaný do historie. Takto uchované záznamy jsou předmětem činnosti popisované funkce, která umožňuje jejich verifikaci. Ta probíhá tak, že je - dle zadaných argumentů - jeden, nebo více neakceptovaných plánů porovnáváno s údaji v *baselines* pro daný příkaz a

pokud jsou shledány výkonnějšími, dojde k jejich potvrzení. Funkce neprovede žádné změny, pokud je příslušná *baseline* fixní (viz dále podtitul Fixed SQL Plan Baselines).

Fixed SQL Plan Baselines

Každý plán v *SQL plan baselines* obsahuje atribut FIXED. Pokud je nastaven na YES, stává se fixním nejen on sám, ale také *baseline*, do které náleží. Během optimalizace mají poté fixní plány vyšší prioritu. Pokud *baseline* obsahuje více takto označených položek, vybírá se mezi nimi. Případné ostatní záznamy mechanismus uvažuje pouze, pokud jsou všechny fixní plány nereprodukovatelné z důvodu změn v systému.

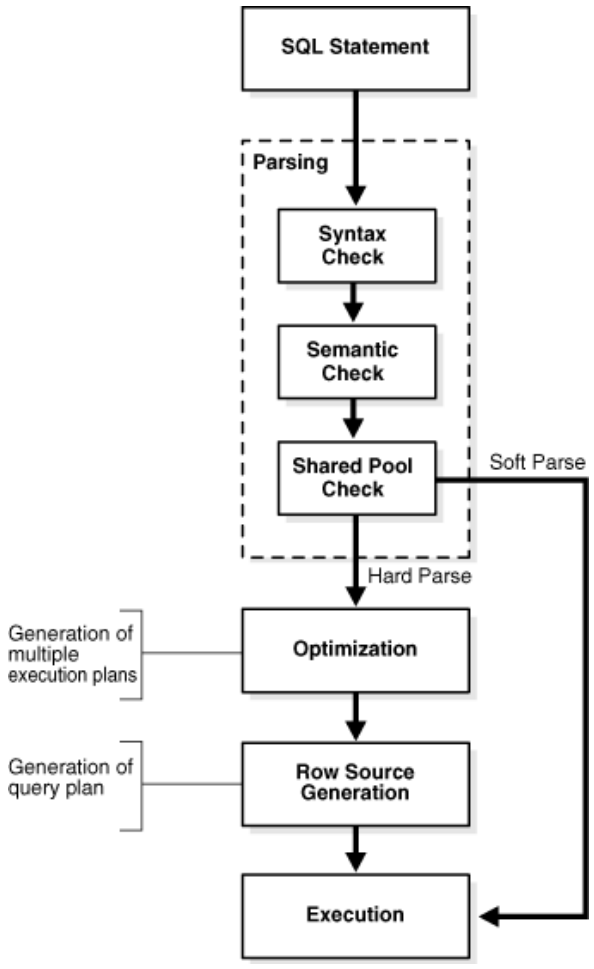
Využití této možnosti je zřejmé - pokud je znám exekuční plán, jenž představuje efektivní postup, a existuje předpoklad, že si tuto vlastnost udrží i v budoucnosti, je vhodné ho označit jako fixní. Tím je vynuceno jeho používání.

3 Zpracování příkazu

Následující text vychází především ze zdrojů [4], [8] a [15]. Neboť zpracování příkazu je v podstatě procesem, který je předmětem optimalizace, do této práce bezpochyby patří i na dané téma zaměřená kapitola. Databáze Oracle může v závislosti na typu příkazu a dalších faktorech provést následující kroky:

- Analýza;
- Optimalizace;
- Generování řádkového zdroje;
- Provedení příkazu.

Přestože většina optimalizačních mechanismů usiluje o optimalizaci samotného provedení příkazu, zásadní význam mají i takové, které umožňují vynechat určité fáze zpracování a výrazně tak urychlit jeho průběh, neboť jediné, co každý příkaz opravdu musí, je být proveden. Především etapa optimalizace bývá zpravidla značně nákladnou operací a její aktivita během všech jednotlivých průchodů procesem by výkon systému velmi znehodnotila. Zpracování příkazu je znázorněno na obrázku níže (Obrázek 2), jeho jednotlivé kroky jsou dále popsány v samostatných oddílech.



Obrázek 2 - Fáze zpracování SQL příkazu, převzato z [4]

3.1 Analýza

Během analýzy dochází k řadě úkonů, primárně je to transformace vstupu do podoby požadované datové struktury, ve které bude dále zpracováván. Dále je ověřena syntaktická a sémantická korektnost příkazu - pokud tedy jeho volání končí chybou, ve většině případů byla odhalena ve fázi analýzy a proces zpracování touto etapou skončil. Výjimkou jsou situace, kdy v dalším průběhu zpracování dojde například k deadlocku či nekorektní konverzi datových typů.

Součástí analýzy je i v úvodu kapitoly naznačený mechanismus, který umožňuje za daných podmínek vynechat nákladný krok optimalizace. Jeho anglické označení je *shared pool check* a z hlediska tématiky této práce, ve které bude označován jako kontrola sdíleného fondu, představuje nejpodstatnější úkol analýzy příkazu. Z tohoto důvodu mu je věnován samostatný oddíl.

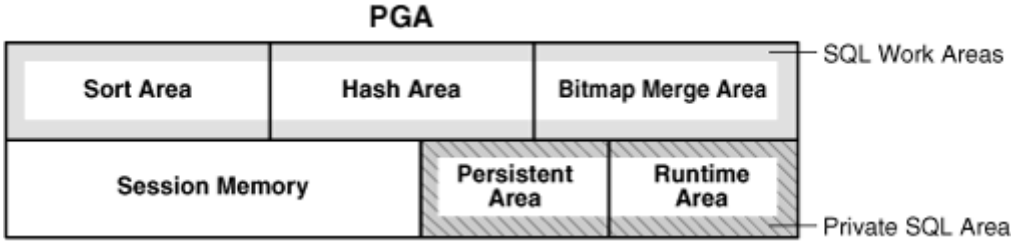
Ani fáze analýzy nemusí být povinně prováděna. O tom, zda tomu tak bude či ne, nemůže rozhodovat samotná databáze, nýbrž jen aplikace s ní pracující. Přímé provedení příkazu po přeskočení všech předchozích úkonů včetně analýzy logicky významně zvyšuje výkon systému. Nejprve budou tedy popsány možnosti jejího vynechání a následně výše uvedené kroky, které během své činnosti provádí.

3.1.1 Zpracování příkazu bez provedení analýzy

Jak je uvedeno výše, sama databáze se nemůže vzdát procesu analýzy. K tomuto kroku musí být instruována z dané aplikace, na jejímž vývojáři tedy leží rozhodnutí, jak s udělenou pravomocí naložit. Vynechání analýzy umožňuje koncept dat kursorů uložených v paměťové oblasti PGA. Nejprve budou tedy stručně popsány tyto termíny.

Program Global Area (PGA)

Program global area (PGA) obsahuje data a řídící informace pro proces serveru, které nejsou ostatním procesům sdíleny. PGA je dále logicky dělena na další segmenty sloužící například jako úložný prostor pro paměťově náročné řazení či spojování záznamů během provádění exekučního plánu. Tato úložiště se souhrnně nazývají *SQL work areas* a optimalizátor je při rozhodování ovlivněn také jejich kapacitou, jejíž konfigurace je zpravidla dle doporučení ponechána na nástroji *Automatic Memory Management* (AMM) systému Oracle. S hlediska kursorů příkazů má význam podoblast *private SQL area*.



Obrázek 3 - Struktura PGA, převzato z [8]

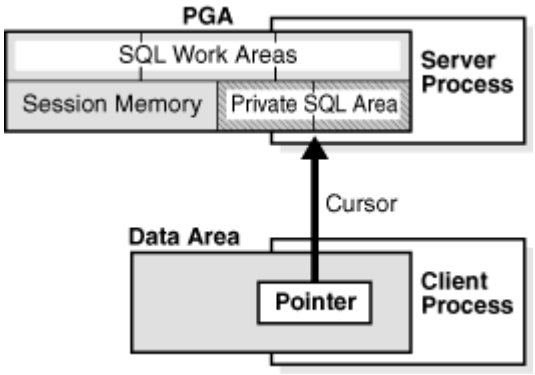
Private SQL area

Tento paměťový segment obsahuje informace o analyzovaných příkazech a další záznamy vztahující se k dané relaci. Během provádění daného příkazu jsou v *private SQL area* uchovávány údaje o stavu tohoto procesu, pracovních prostorech *work areas* nebo o hodnotách vázaných proměnných (viz pododdíl 5.2.4), atd. Oblast je dále dělena na segmenty:

- *Run-time* - uchovávající aktuální informace v rámci provádění příkazů, jako například počet dosud načtených řádků během plného průchodu tabulkou. V případě DML příkazů je run-time oblast uvolněna při jejich uzavření.
- *Persistent* - obsahuje například hodnoty vázaných proměnných, které jsou dodávány procesu provádění dotazu průběžně během jeho aktivity. Tato oblast je uvolněna při uzavření kursoru.

Kursor

Kursor je handler pro konkrétní oblast *private SQL area*, který leží na straně klienta. V podstatě se tedy jedná o ukazatel, prostřednictvím kterého klientská aplikace přistupuje k vyhrazenému paměťovému prostoru databázového serveru. Aplikace prací s kursory do velké míry zodpovídá za správu dané *private SQL area*. Databáze ji může limitovat parametrem OPEN_CURSORS, který udává maximální počet otevřených kurzorů.



Obrázek 4 - Kursor, převzato z [8]

Snížení počtu analýz

Při práci s kurzory v rámci aplikace je třeba najít vhodný kompromis mezi efektivním přístupem k uvolňování paměti *private SQL area* a možností jejich opakovaného využití.

Pokud daná aplikace provede příkaz pouze jednou, je na místě daný kurzor po této operaci okamžitě uzavřít, nebo jej případně využít pro příkaz jiný. V opačném případě může být užitečné ponechat kurzor otevřený pro opakované použití, během kterého pak aplikace nebude muset požadovat vykonání fáze analýzy a příkaz bude moci být přímo proveden.

I již uzavřený kursor lze v některých situacích opětovně využít, neboť systém Oracle jeho data může uchovávat v mezipaměti. Toto chování podléhá nastavení parametru SESSION_CACHED_CURSORS, který udává právě maximální počet takto uložených kurzorů pro danou relaci. Defaultní hodnota je 50.

Výhodným řešením je využití statických příkazů v rámci PL/SQL kódu, které jsou vždy, pokud existuje dostatek prostoru nebo nehrozí překročení hranice OPEN_CURSORS, ponechávány otevřené pro opětovné provedení.

3.1.2 Syntaktická kontrola

Zde je ověřena syntaktická správnost příkazu, kterou mohou narušit překlepy ve vyhrazených slovech, jejich vynechání či chybná posloupnost, atd. Nevyhovující příkaz nemá smysl dále zpracovávat a databáze vypíše chybu. Často dochází například k překlepu v klíčovém slovu FROM:

```
SQL> SELECT * FORM employees;
SELECT * FORM employees
         *
ERROR at line 1:
ORA-00923: FROM keyword not found where expected
```

3.1.3 Sémantická kontrola

Pokud je syntaxe správná, může ukončení zpracování příkazu dále způsobit jeho nekorektní sémantika. Ta se týká například názvů sloupců a tabulek, u kterých se syntaktická kontrola zajímá jen o to, jestli jsou uvedeny na správném místě. Až kontrola sémantiky testuje, zdali uvedené tabulky skutečně existují, obsahují požadované sloupce, či k nim daný uživatel přistupuje oprávněně. Dále je zde mimo jiné ověřena jedinečná identifikace těchto objektů v rámci příkazu. Například pokus o provedení dotazu na neexistující tabulku vyvolá následující chybu:

```
SQL> SELECT * FROM nonexistent_table;
SELECT * FROM nonexistent_table
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

Pokud příkaz splní požadavky obou kontrol, dle jeho typu pokračuje proces přechodem na příslušný další krok.

3.2 Kontrola sdíleného fondu (shared pool check)

Kontrola sdíleného fondu patří do fáze analýzy, ale pro její výraznou vazbu k tématu této práce je jí vyhrazen vlastní oddíl. K jejímu provedení dochází pouze u příkazů DML, které pracují s daty již existujících objektů vytvořených a spravovaných příkazy DDL. Neboť jen u nich má smysl a je vykonávána fáze optimalizace a tudíž je možné opakovaně používat již jednou stanovený způsob provedení - exekuční plán. Krok optimalizace naopak není nikdy aplikován na samotné příkazy DDL.

Výhradní postavení má DML příkaz SELECT, označovaný jako dotaz, protože právě na volbu jeho provedení jsou vynakládány výrazné prostředky, jelikož má za úkol dle zadaných kritérií vyhledat, zformovat a poskytnout data častokrát velmi rozsáhlých tabulek.

V následujícím textu jsou nejdříve popsány důležité termíny a dále samotný proces kontroly skládající se z:

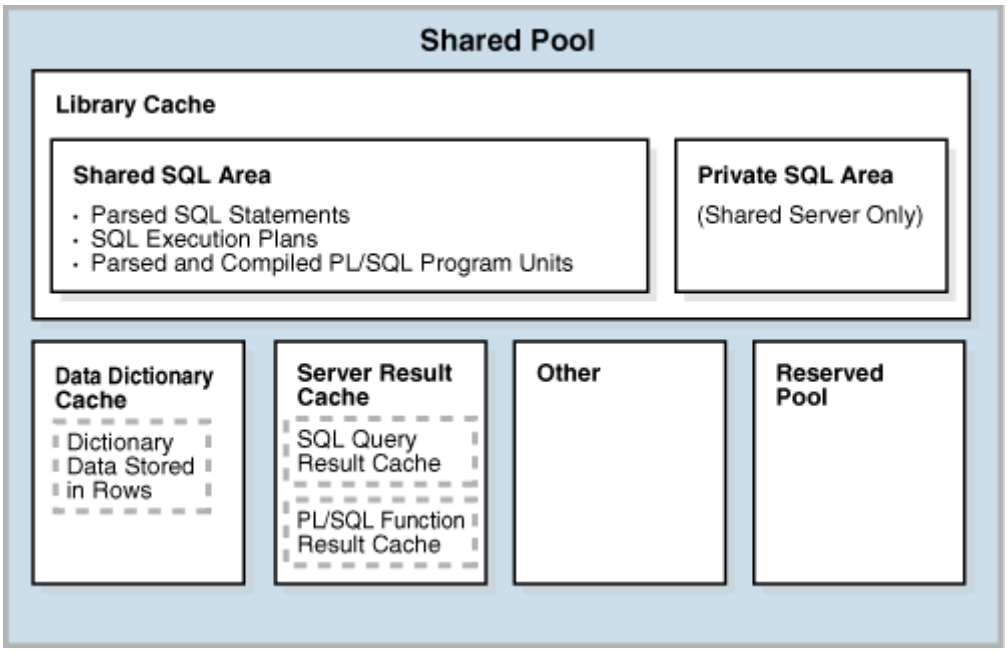
- Prohledávání *shared SQL area*;
- Kontroly sémantické shody;
- Kontroly shody prostředí.

System Global Area (SGA)

System global area (SGA) je sdílená paměťová oblast, jenž obsahuje data a řídící informace databázové instance. Konkrétně uchovává například z disku načtené bloky tabulek a indexů, informace z datového slovníku, návratový log, atd. Pro tento text má význam především její segment *shared pool*.

Sdílený fond (shared pool)

Tato část paměťové oblasti SGA slouží k uchovávání různorodých programových dat sdílených v rámci instance databáze. Je dále logicky dělena na další segmenty, které obsahují například datový slovník databáze, systémové parametry, analyzované SQL a PL/SQL příkazy, atd. Pro popisovanou problematiku má význam především podoblast označená jako *library cache* a konkrétněji její část *shared SQL area*. Základní struktura je znázorněna na následujícím obrázku (Obrázek 5).



Obrázek 5 - Shared pool, převzato z [8]

Velikost jak celé oblasti SGA, tak segmentu *shared pool* lze nastavit manuálně, pomocí parametrů:

- SGA_TARGET,
- SGA_MAX_SIZE,
- SHARED_POOL_SIZE.

Ovšem k využití této možnosti dochází sporadicky, neboť systém Oracle disponuje nástrojem *Automatic Memory Management* (AMM), jenž dokáže transparentně a flexibilně rozdělovat dostupnou paměť mezi oblasti, které ji v daný okamžik potřebují. Dostatečná kapacita je důležitým faktorem pro optimální výkon databáze nejen v případě prostoru *shared pool*, ale i dalších segmentů. Například paměti *buffer cache*, kde jsou uchovávány kopie bloků načtených z datových souborů na disku a budoucí dotazy na příslušné objekty tak mohou provádět jen méně nákladné logické čtení z tohoto úložiště. Aktuální kapacitu paměti *shared pool* a její využití lze najít v pohledu V$SGAINFO respektive V$SGASTAT.

K manuálnímu vyprázdnění sdíleného fondu lze použít příkaz ALTER SYSTEM FLUSH SHARED_POOL.

Shared SQL area

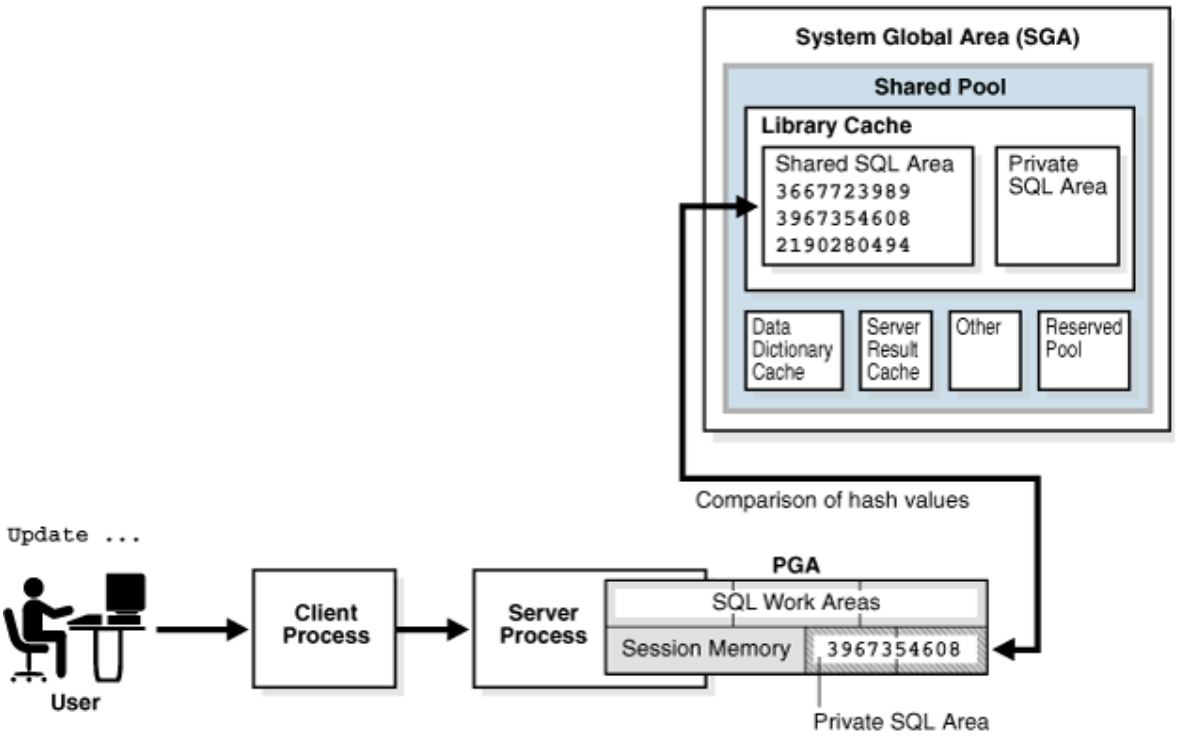
Během prvního zpracování daného příkazu je do této oblasti uložen jeho *parser tree* a optimální exekuční plán (viz kapitola 2), dle kterého byl vykonán. Data jsou poté k dispozici při fázích analýzy vyvolaných všemi uživateli a později, pokud je znovu zpracováván stejný příkaz, jsou využity pro jeho provedení. Seznam takto uložených příkazů a jejich plánů lze zobrazit prostřednictvím pohledu V$SQL respektive V$SQL_PLAN.

V případě, že dojde k aktualizaci statistik některého objektů (viz kapitola 1), dojde také k zneplatnění záznamů všech příkazů, které s ním pracují. Pokud je poté zpracováván příslušný příkaz, nelze tato uložená data využít a je provedena nová optimalizace ovlivněná současnou podobou statistik. Výjimkou jsou systémové statistiky, kde údaje v *shared SQL area* zůstávají platné a jejich aktualizovaná podoba tak má vliv jen na nově vytvářené exekuční plány doposud neoptimalizovaných příkazů.

3.2.1 Prohledávání shared SQL area

Na právě analyzovaný příkaz je aplikována funkce *hash*, která vrací jeho *hash* hodnotu, označovanou jako SQL ID. Tento identifikátor je pak použit k vyhledání odpovídajícího záznamu ve sdíleném fondu. Pokud je hledání úspěšné, je třeba provést další kontroly shody daných příkazů, neboť SQL ID je dáno pouze jejich syntaktickou textovou podobou a volba vhodného exekučního plánu provedení přirozeně závisí i na dalších faktorech. Text příkazů musí být přesně totožný. Jako odlišnost se berou i rozdíly ve velikosti písmen, výskytu komentářů, sekvencích mezer, atd. Výjimkou jsou pouze situace, kdy se dva příkazy liší v hodnotách literálů a systémovému parametru CURSOR_SHARING (viz

5.4.1 Parametry optimalizátoru Oracle) je přiřazena nedefaultní úroveň FORCE. Za takových okolností mechanismus pracuje s těmito literály jako s vázanými proměnnými (viz 5.2.4) a považuje dané dotazy za shodné. Dále případně následují fáze kontroly shody sémantiky a prostředí.



Obrázek 6 - Kontrola sdíleného fondu, převzato z [4]

3.2.2 Kontrola sémantické shody

Typickým příkladem důvodu k tomuto testu je zpracování „stejných“ dotazů různých uživatelů, kteří mají oba ve svém schématu vytvořenou stejnojmennou tabulku, jejíž data chtějí získat. Oba zadají například příkaz:

```
SELECT * FROM EMPLOYESS;
```

Ač jsou jejich textové podoby totožné, odkazují dotazy na dvě různé tabulky v příslušných schématech, jejichž odlišnost může být velká. Stejně tak se tedy mohou lišit i optimální exekuční plány těchto dotazů a kontrola sémantické shody je shledá různými.

3.2.3 Kontrola shody prostředí

Jak bude popsáno v následujících kapitolách, provedení příkazu lze ovlivnit volbou mnoha databázových parametrů, například nastavením OPTIMIZER_MODE na FIRST_ROWS, respektive ALL_ROWS. Daná konfigurace může přímo vyžadovat či nepřímo zapříčinit určitou podobu exekučního plánu a lze jí souhrnně označit právě jako prostředí, ve kterém je dotaz analyzován a proveden. Je tedy nutné shodu ověřit také z tohoto hlediska.

Pohled V$SQL obsahuje položky OPTIMIZER_ENV a OPTIMIZER_ENV_HASH_VALUE, pomocí nichž lze odlišit příslušné nastavení, ve kterém byl daný příkaz optimalizován.

Pokud jsou i dle tohoto kritéria příkazy shledány totožnými, nastává tzv. *library cache hit* a může být provedena *soft parse*. Opačný případ se nazývá *library cache miss* a vede k procesu *hard parse*. Důvody, proč při opakované analýze zdánlivě shodného dotazu nastal *library cache miss*, lze odhalit pomocí pohledu V$SQL_SHARED_CURSOR.

3.2.4 Soft parse

Když je v *shared SQL area* nalezen potřebný plán, lze jej využít k okamžitému provedení příkazu. Není tak třeba provádět fáze optimalizace a generování řádkového zdroje.

3.2.5 Hard parse

Nastává v případech, kdy daný příkaz dosud nebyl analyzován, nebo po tomto procesu došlo ke změně faktorů, které ovlivňují rozhodování optimalizátoru. Například byly modifikovány parametry systému nebo statistiky souvisejících objektů. Tehdy je nutné přistoupit k etapě optimalizace příkazu a nalézt vhodný exekuční plán.

Hard parse je postupem, jehož výskyt je pro výkonný provoz systému nutné minimalizovat. Především na škálovatelnost aplikací (viz 5.1.1) ve vysoce konkurenčním prostředí OLTP systémů by měl častý průchod tímto procesem fatální vliv, neboť analýzu a zejména optimalizaci dotazů nelze dle knihy Oracle - Návrh a tvorba aplikací [15] provádět souběžně s mnoha dalšími operacemi. Důvodem je, že během těchto úkonů databáze poměrně intenzivně přistupuje ke sdíleným informacím v datovém slovníku a *library cache*, které je nutné v daných okamžicích blokovat před přístupy procesů ostatních uživatelů, jenž by mohly provést nežádoucí modifikace. Tento serializační mechanismus se nazývá *latches* a ačkoli je jeho zapojení pro stabilitu systému nezbytné, nese s sebou nevýhody delšího času zpracování a právě nižší souběžnosti konkurenčních operací, které musí čekat na zpřístupnění potřebných sdílených datových struktur.

3.3 Optimalizace

Tato fáze má za úkol dle dostupných informací a pokynů pro analyzovaný příkaz vytvořit vhodný exekuční plán. Povaha příkazů DDL je specifická a tak jimi může být optimalizační proces vyvolán jen v případech, kdy obsahují vnořené DML dotazy, na které je aplikován. Naopak každý příkaz DML musí být proveden dle daného exekučního plánu, který je tak třeba sestavit minimálně jednou během prvního volání.

Etapu optimalizace vykonává komponent Optimalizátor, který patři k nejpozoruhodnějším modulům databáze Oracle, neboť využívá poměrně sofistikované metody a algoritmy. Jeho činnosti je dále v této práci věnována samostatná kapitola 4 Optimalizátor Oracle.

3.4 Generátor řádkového zdroje

Generátor řádkového zdroje provádí transformaci během optimalizace sestaveného exekučního plánu na iterativní plán, který se nazývá také *query plan* a je reprezentován binárním programem. Jeho formátovanou verzi lze zobrazit například pomocí funkcí EXPLAIN_PLAN nebo AUTOTRACE (viz oddíl 2.2). Ačkoli exekuční plán popsaný v

kapitole 2 má blíže k tomuto výslednému programu, rozdíl mezi podobou vstupu a výstupu generátoru není pro tuto práci příliš podstatný a proto jej nebere v úvahu.

3.5 Provedení příkazu

Poslední krok procesu zpracování příkazu je také jediným krokem, který musí být vždy proveden. Samozřejmě za předpokladu, že během případných předchozích fází není odhalena chyba.

Princip provádění příkazu je popsán v kapitole 2 Exekuční plán.

4 Optimalizátor Oracle

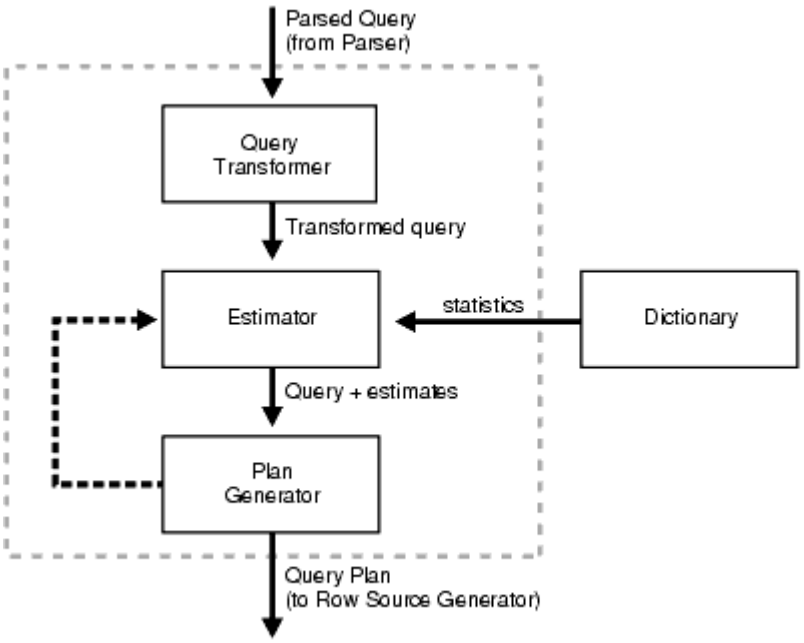
Tématika předchozího textu práce se protíná v této kapitole, neboť optimalizátor systému Oracle je logicky modulem stěžejním pro optimální provádění dotazů. Jeho aktivita je jednou z etap procesu zpracování příkazu (kapitola 3) a s využitím informací o objektech a výkonových charakteristik databáze - optimalizačních statistik (kapitola 1) - generuje výstup své činnosti, jímž je optimální exekuční plán pro provedení příkazu (kapitola 2). Postup, který k tomu využívá, lze rozepsat do třech základních kroků:

1. Generování různých variant exekučního plánu na základě daných možností a pokynů, například hintů (viz pododdíl 5.4.2).
2. Odhad ceny těchto plánů s využitím optimalizačních statistik.
3. Výběr nejlevnějšího plánu.

Uvedený proces vykonávají následujícími moduly:

- Query transformer;
- Estimator;
- Plan generator.

Obrázek 9 znázorňuje schéma těchto komponentů, které jsou dále podrobněji popsány v samostatných oddílech.



Obrázek 7 - Komponenty optimalizátoru Oracle, převzato z [9]

4.1 Query transformer

První etapa obdrží během fáze analýzy parsovaný příkaz, jehož konstrukce může být složitější. Například dotaz SELECT může obsahovat vnořené poddotazy. V takovém

případě jsou aktivovány mechanismy, které ověřují, zda pro příkaz neexistuje syntakticky odlišná, ovšem sémanticky totožná, podoba, ve které by bylo možné jej zpracovat méně nákladně.

Mezi techniky, které *query transformer* používá a případně mezi sebou vzájemně kombinuje, patří například:

- View Merging;
- Predicate Pushing;
- Subquery Unnesting;
- Query Rewrite with Materialized Views.

4.1.1 View Merging

Tato technika se týká dotazů, které pracují s pohledem. Například pokud existuje pohled *employees_50_vw*:

```
CREATE VIEW employees_50_vw AS
  SELECT employee_id, last_name, job_id, salary, commission_pct,
department_id
  FROM   employees
  WHERE  department_id = 50;
```

Na který je aplikován dotaz:

```
SELECT employee_id
FROM   employees_50_vw
WHERE  employee_id > 150;
```

Během fáze analýzy je odkaz na daný pohled nahrazen jeho definicí. Optimalizátor tedy příkaz SELECT obdrží v následujícím tvaru:

```
SELECT employee_id
FROM
  SELECT employee_id, last_name, job_id, salary, commission_pct,
department_id
  FROM   employees
  WHERE  department_id = 50;
WHERE  employee_id > 150;
```

A s využitím popisovaného mechanismu jej transformuje do podoby, pro kterou poté hledá optimální plán:

```
SELECT employee_id
FROM   employees
WHERE  department_id = 50
AND    employee_id > 150;
```

View merging lze použít pouze u jednoduchých pohledů, které jsou tvořeny operacemi selekce, projekce a spojení. V komplikovanějších případech je třeba pracovat s definicemi obsažených pohledů jako s oddělenými dotazy. Pro každý takový segment je nezávisle vytvořen samostatný exekuční plán, který je poté spojen s ostatními do výsledného

postupu. Tento způsob oddělené optimalizace zpravidla nedosahuje tak kvalitních výsledků, jako sofistikovanější varianta po uplatnění *view merging*.

4.1.2 Predicate pushing

Princip této techniky je podobný mechanismu *view merging*, který může zastoupit v situacích, kdy jej pro složitost pohledu nelze použít. Například při dotazu:

```
SELECT last_name
FROM   all_employees_vw
WHERE  department_id = 50;
```

Na pohled:

```
CREATE VIEW all_employees_vw AS
  ( SELECT employee_id, last_name, job_id, commission_pct, department_id
    FROM   employees )
  UNION
  ( SELECT employee_id, last_name, job_id, commission_pct, department_id
    FROM   contract_workers );
```

Nelze praktikovat *view merging*, neboť pohled obsahuje klausuli UNION. Je tedy využit *predicate pushing*, který dokáže vložit podmínku *department_id = 50* do obou poddotazů tvořících spojení UNION a příkaz SELECT bude dále optimalizován v následující podobě:

```
SELECT last_name
FROM   ( SELECT employee_id, last_name, job_id, commission_pct,
department_id
         FROM   employees
         WHERE  department_id=50
         UNION
         SELECT employee_id, last_name, job_id, commission_pct,
department_id
         FROM   contract_workers
         WHERE  department_id=50 );
```

Vnořené dotazy jsou sice poté zpracovávány separátně, jak je uvedeno na konci předchozího pododdílu, ale při tvorbě jejich plánů má optimalizátor kompletní informaci o selekci v dotazu a může tak například zvolit efektivnější přístup prostřednictvím indexu.

4.1.3 Subquery unnesting

Proces *subquery unnesting* představuje transformaci vnějšího dotazu a jeho vnořeného poddotazu na jejich spojení, které je poté optimalizováno. Může k němu dojít například při následujícím použití klausule IN:

```
SELECT *
FROM   sales
WHERE  cust_id IN ( SELECT cust_id FROM customers );
```

Tento dotaz bude převeden na sémanticky totožnou podobu:

```
SELECT sales.*
FROM   sales, customers
WHERE  sales.cust_id = customers.cust_id;
```

Aplikovatelnost je opět podmíněna určitým stupněm jednoduchosti vnitřního poddotazu, který nesmí obsahovat agregační funkce. Pokud toto kritérium nesplňuje, opět je použit oddělený přístup a optimalizátor provádí vnořený blok, přičemž jeho výsledné řádky používá při zpracování dotazu vnějšího.

4.1.4 Query Rewrite with Materialized Views

Pokud pro dotaz nebo jeho část existuje ekvivalentní materializovaný pohled (viz pododdíl 5.4.5), lze využít jeho záznamy a ušetřit tak významný podíl práce. K přepisu nemusí dojít, i pokud daný pohled existuje, neboť optimalizátor stanoví cenu provedení s využitím *query rewrite* a pokud je vyšší, je pohled ignorován. Toto opatření lze potlačit pomocí hintu REWRITE (viz 5.4.2 Hinty).

4.2 Estimator

Komponent *plan generator* tento modul využívá k ohodnocení exekučních plánů, které vytvořil. Pro vyjádření nákladů *estimator* využívá tři kritéria:

- Selektivita;
- Kardinalita;
- Cena.

4.2.1 Selektivita (selectivity)

Selektivita představuje míru záznamů, které vyhovují predikátu dotazu. Zapsat ji lze jako desetinné číslo v intervalu <0.0, 1.0>, přičemž platí, že čím větší toto číslo je, tím více záznamů podmínku splňuje. Pokud se naopak tato hodnota blíží k 0, hovoříme o vysoké selektivitě, kdy filtrem prochází malé procento dat.

Pro odhad této veličiny mají zásadní význam optimalizační statistiky. Jak je uvedeno v kapitole 1, pokud nejsou k dispozici plnohodnotné statistiky nebo není povoleno dynamické vzorkování, kdy:

OPTIMIZER_DYNAMIC_SAMPLING = 0,

estimator musí použít defaultní hodnoty (viz tabulka 1). V opačném případě může provést zpravidla kvalitnější odhad na základě přesnějších charakteristik.

4.2.2 Kardinalita (cardinality)

Kardinalita neboli mohutnost udává počet řádků v dané množině, kterou může být buď základní tabulka či pohled, nebo výsledek operací JOIN a GROUP BY, které jsou vykonávány v rámci provádění dotazu.

4.2.3 Cena (cost)

Náklady na jeho provedení příkazu vyjadřuje jeho cena. Zásadní odlišnosti mezi jednotlivými exekučními plány daného příkazu způsobuje především zvolený způsob přístupu k záznamům a případně použitý typ či pořadí jejich spojování. Tato metrika konkrétně zahrnuje následující vynakládané prostředky:

- Počet I/O operací;
- Využití CPU;
- Využití paměti.

4.3 Plan generator

Posledním komponentem optimalizátoru Oracle je *plan generator*, který má za úkol v kooperaci s modulem *estimator* vytvářet a oceňovat různé varianty exekučních plánů pro provedení příkazu. Jejich variabilita spočívá především v použití rozdílných přístupových technik a v případě výskytu spojení množin, také ve volbě metod a pořadí těchto operací. Množství možných plánů je úměrné zejména počtu množin řádků, které je třeba v rámci dotazu spojit.

Pokud generátor obdrží příkaz SELECT obsahující vnořené pododtazy nebo pohledy, které nemohly být transformovány technikou *view merging*, musí každý takový segment dotazu nejprve zpracovávat odděleně a generovat pro něj samostatný subplán. Proces probíhá obdobně, jako pozdější fáze provádění - postupuje se od nejvíce zanořených bloků až k výslednému kompletnímu dotazu.

Když mechanismus generování dospěje k plánu, jenž považuje za dostatečně kvalitní, pokračuje proces zpracování příkazu etapou generování řádkového zdroje dle tohoto zvoleného plánu. Vzhledem k teoreticky téměř neomezenému rozsahu variant, jakými lze složitější příkazy provést, optimalizátor vždy prověřuje jen určitý podíl možných plánů. Mechanismus tedy umí posoudit, zda je cena aktuálně nejlepší varianty stále natolik vysoká, že se vyplatí pokračovat v hledání. Pokud ne, nepředpokládá v dalším průběhu významné zlepšení a zvolí daný plán.

Důležitým úkolem modulu je stanovit výchozí exekuční plán, především tedy vhodné pořadí spojení, které případně obsahuje. Přirozeně platí, že čím blíže má tento vstup k ideální variantě, tím rychleji je zpravidla nalezena jeho dostatečně optimální podoba.

5 Optimalizace SQL dotazu

5.1 Cíle a kritéria

Pojem optimalizace konkrétně představuje snahu o dosažení následujících cílů, které mají snížit dobu odezvy systému, nebo naopak minimalizací potřebných HW prostředků zvýšit jeho propustnost:

- *Reduce the Workload* - hledání efektivní realizace daných úkolů, konkrétně tedy vhodného exekučního plánu, který příkaz zpracuje korektně, ale s nejmenšími náklady.

- *Balance the Workload* - snaha o rovnoměrné rozdělení zátěže na celé období činnosti systému. Například provádění oken údržby v nočních hodinách, kdy se systémem pracuje minimum uživatelů.

- *Parallelize the Workload* - dotaz, který pracuje s velkým objemem dat, lze pro snížení doby odezvy provádět paralelně. Tato technika je užitečná především v prostředí datových skladů, které není zatíženo velkým počtem souběžně prováděných příkazů. Naopak v provozním systému OLTP, jenž bývá limitován vysokou konkurencí mezi uživateli, může paralelismus způsobit nežádoucí efekt.

Důležitým atributem dobře fungujícího systému je také jeho škálovatelnost, kterou stručně popisuje následující pododdíl.

5.1.1 Škálovatelnost (scalability)

Škálovatelnost se týká poměru mezi objemem zátěže a na ni vynakládanými systémovými prostředky. Například pokud je na systém vyvíjena oproti běžnému provozu dvojnásobná zátěž, teoreticky by mělo být využito dvojnásobné množství zdrojů, než během klasického zatížení. Takový systém by byl označen jako lineárně škálovatelný. Ovšem s rostoucí zátěží se zvyšuje i počet režijních operací a konfliktů, kdy například příkaz jednoho uživatele nemůže přistupovat k uzamčeným datům, se kterými v danou chvíli pracuje uživatel jiný. V důsledku tohoto konkurenčního prostředí, roste křivka vytížení prostředků nelineárně a dochází ke snížení škálovatelnosti. V takových případech mohou mít po výraznějším nárůstu počtu uživatelů problémy s výkonem i ty aplikace, které během vývoje a testování pracovaly dobře.

Ideální aplikace by byla z hlediska této vlastnosti limitována pouze HW, na němž je provozována a nebyla by zatížena následujícími nedostatky, které mohou zapříčinit nízkou škálovatelnost:

- Špatný návrh schématu a SQL příkazů databázového systému.
- Špatně navržené databázové transakce z aplikace.
- Nevhodná realizace připojení aplikace k databázi.
- Provádění paměťově náročných operací bez zajištění opětovného efektivního uvolňování a obecně neefektivní nakládání s dostupnou pamětí.

K této práci se vztahuje zejména první uvedený případ. Velkým přínosem pro aplikaci je častý výskyt procesů *soft parse* či úplného vynechání analýzy (viz kapitola 3), kterého lze dosáhnout například vhodným použitím vázaných proměnných stručně popsaným dále v pododdíle 5.2.4.

5.1.2 Postup

Jak ostatně vyplývá z textu předchozích kapitol, optimalizace dotazu v databázovém systému je velmi komplexní problematikou, pro kterou společnost Oracle vytvořila a s každou novou verzí stále vyvíjí řadu mechanismů a nástrojů. Optimalizační moduly, které dokážou do velké míry pracovat samostatně, jsou popsány v další kapitole 6. Ať už proces optimalizace probíhá relativně automaticky, nebo je zcela v režii administrátora, dochází během něho zpravidla k následujícím etapám, které mohou být vykonávány opakovaně až do dosažení požadovaného výkonu:

1. Identifikace nákladných SQL příkazů.
2. Hledání příčin náročnosti na zdroje. Verifikace exekučních plánů, dle kterých jsou tyto příkazy prováděny.
3. Realizace opatření pro zlepšení výkonu.

Uvedené fáze jsou dále v této kapitole podrobněji rozepsány v oddíle 5.3 Proces optimalizace. Ideální ovšem je, pokud je není třeba vůbec realizovat. Nejprve budou tedy stručně popsány zásady a možnosti, jak lze výkonný databázový systém především vytvořit, než jak jej optimalizovat později během provozu.

5.2 Tvorba efektivních databázových struktur a příkazů

5.2.1 Efektivní návrh schématu

Profesionální přístup přirozeně vyžaduje, aby již během fáze návrhu bylo pamatováno na to, jaký účel systém bude mít, jak velký okruh uživatelů bude s aplikací pracovat, jaké dotazy budou volány často a jaké naopak minimálně, které tabulky budou obsahovat velké množství záznamů, atd. Základem dobrého databázového systému je správný návrh struktur a vazeb jeho tabulek. Nutností je dodržování normalizace, ovšem dílčí prohřešky vůči ní mohou být v některých situacích žádoucí. Například schémata datových skladů často připouští redundanci dat, která ovšem umožňuje rychlejší zpracování dotazů. Dále je třeba používat vhodné datové typy. Během fáze fyzického návrhu je nutné s vědomím konkrétní charakteristiky systému zvážit použití indexů, clusterů, dělení tabulek, atd.

5.2.2 Cluster

Cluster je skupina souvisejících tabulek, jejichž data jsou ukládána ve společném fyzickém úložišti. V rámci clusteru může jeden datový blok obsahovat záznamy z více tabulek. Základním principem a benefitem této techniky je snížení počtu bloků, které je třeba načíst během provádění příkazu. Souvislost jednotlivých záznamů je odvozena z podobnosti hodnot stanoveného klíče, dle kterých jsou umísťovány do příslušných segmentů. Například pokud bude aplikace často používat následující dotaz vracející data z tabulek *employees* a *departments* spojená prostřednictvím identifikátoru oddělení *department_id*:

```
select * from
      hr.employees
      join hr.departments
using(department_id);
```

Může být užitečné vytvořit pro tyto dvě tabulky cluster a jejich záznamy ukládat pospolu dle klíče *department_id*. Při provádění uvedeného dotazu pak načítané bloky obsahují pouze data, která budou pro sestavení výstupu opravdu potřeba a je tedy nutné provést méně I/O operací, než když řádky tabulek nejsou udržovány koherentně. Další výhodou je úspora potřebného úložného prostoru, neboť klíčové položky jsou pro všechny záznamy clusteru totožné a mohou tedy v rámci něho být uloženy pouze jedenkrát. Cluster definovaný pouze pro jednu tabulku lze dle knihy Oracle - návrh a tvorba aplikací [15] použít také jako alternativu k indexu nebo indexově orientované tabulce.

Přirozeně existují i následující případy, kdy použití clusteru není vhodné [11]:

- Časté modifikace daných tabulek (především sloupců použitých jako klíče clusteru) či aplikování příkazu TRUNCATE.
- Časté provádění plného průchodu některé z tabulek bez potřeby načítat data ostatních tabulek v clusteru.
- Počty řádků obsahujících jednotlivé klíčové hodnoty se výrazně liší.
- Všechny záznamy s danou hodnotou klíče se nevejdou pouze do jednoho bloku.

5.2.3 Dělení (partitioning)

Systém Oracle umožňuje dle zadaných kritérií rozdělit rozsáhlejší tabulky a jejich indexy na menší segmenty. Každý takový oddíl je uvnitř databáze reprezentován jako samostatný objekt, jemuž lze nastavit specifické fyzické atributy. Z pohledu aplikace lze ale s rozdělenou tabulkou pracovat obdobně jako s běžnou a při použití této techniky tak není třeba přizpůsobovat použité příkazy. Ovšem z hlediska optimalizace je vhodné při tvorbě SQL dotazů brát případné dělení v úvahu a snažit se jeho předností maximálně využít. Tyto výhody mohou být následující:

- Zvýšená dostupnost - vzhledem k separaci oddílů uvnitř databáze lze při nedostupnosti některého z nich ostatní segmenty bez problému využívat.
- Usnadněná správa - více menších oblastí se spravuje snadněji než jedna globální a potřebné operace jsou prováděny efektivněji.
- Snížená doba provádění údržby.
- Možnost využití efektivního paralelního zpracování, jenž může zvýšit škálovatelnost systému.
- Vyšší výkon dotazů - redukce soupeření o sdílené systémové prostředky v systémech OLTP a zlepšení výkonů ad hoc[1] dotazů v prostředí datových skladů.

[1] Přízvisko ad hoc charakterizuje dotaz, který je prováděn účelově a jednorázově za účelem získání specifických dat potřebných v daný okamžik. Takový příkaz zpravidla není implementován v dané provozní aplikaci, ale je volán prostřednictvím nástrojů pro správu databáze jako SQL *Plus, SQL Developer či phpMyAdmin.

Například pokud existuje dle zadaného seznamu dělená tabulka, která byla vytvořena pomocí následujícího DDL příkazu:

```
CREATE TABLE list_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY LIST (channel_id)
 (PARTITION even_channels VALUES (2,4),
  PARTITION odd_channels VALUES (3,9)
 );
```

A je na ni aplikován dotaz s podmínkou:

```
WHERE channel_id = 3;
```

Pak si je optimalizátor vědom, že záznamy vyhovující predikátu jsou obsaženy pouze v daném oddílu *odd_channels* a nezahrne do exekučního plánu procházení ostatních segmentů.

Dělení v systému Oracle lze dle povahy rozčlenit do čtyř druhů:

- rozsahové,
- seznamové,
- hash,
- kompozitní.

5.2.4 Vázané proměnné (bind variables)

Použití vázaných proměnných podporuje koncept sdílení kursorů založený na mechanismu *shared pool check*. Nalézá uplatnění u relativně podobných příkazů, které jsou, například v rámci cyklu, volány opakovaně. V takovém případě mohou být jednotlivé příkazy této série prováděny dle jednoho stejného exekučního plánu a mohou být tedy zpracovány za pomocí jediného procesu *hard parse*. Příklad je následující.

```
declare
  type rc is ref cursor;
  l_cursor rc;
begin
  for i in 1..5000
  loop
    open l_cursor for
    'select x from t  where x = :x' using i;
    close l_cursor;
  end loop;
end;
```

V databázi během zpracování tohoto cyklu příkazů proběhne pouze jedna fáze optimalizace, a to pro hodnotu *:x = 1*. Funkce *bind variable peeking* poskytne systému tuto

hodnotu proměnné *:x* a umožní mu sestavit konkrétní podobu predikátu a potažmo i dotazu, jenž bude předmětem optimalizace:

```
select x from t  where x = 1;
```

Dle stanoveného exekučního plánu bude proveden nejen uvedený prvně volaný příkaz SELECT, ale i jeho zbylých 4999 následníků. Ve sdíleném fondu (viz oddíl 3.2) bude tedy uložen pouze jeden záznam.

Z relativně univerzálního používání totožného exekučního plánu vyplývá zřejmá nevýhoda základního konceptu vázaných proměnných. Prováděný dotaz je sice vždy syntakticky a principiálně shodný, ovšem přesto pro různé konkrétní hodnoty daných proměnných mohou být vhodné odlišné exekuční plány. Je tedy třeba zvážit, zda bude plán vytvořený během prvního volání příkazu dostatečně efektivní i pro ostatní alternativy hodnot proměnných a nevyplatí se investovat prostředky do uskutečnění většího počtu fází optimalizace. Například v systémech datových skladů, které jsou charakteristické poměrně nesouběžným prováděním nákladných dotazů, je dle dokumentace Oracle [12] doporučeno používat konkrétní literály namísto vázaných proměnných a umožnit tak optimalizátoru lépe odhalit skutečnou selektivitu predikátů.

Adaptive cursor sharing

Výše uvedený nedostatek za cenu určitých režijních nákladů potlačuje funkce adaptivního sdílení kursoru, jenž během své aktivity rozlišuje kursory dvou stavů:

- Bind-Sensitive;
- Bind-Aware.

Pokud například existuje histogram (viz kapitola 1 Databázové optimalizační statistiky) pro sloupec, jenž je v predikátu porovnáván s danou vázanou proměnnou, předpokládá optimalizátor nerovnoměrné rozdělení jeho hodnot a kursoru lze udělit status *bind-sensitive*. Během provádění sekvence takto označených dotazů s různými hodnotami proměnných zaznamenává databáze výkonové statistiky, a pokud získá oproti dosavadním údajům extrémně odlišný výsledek, označí daný kursor jako *bind-aware*. Pro příslušný dotaz je během jeho příštího provádění sestaven nový plán s využitím dané hodnoty vázané proměnné.

5.2.5 Použití PL/SQL kódu

Použití procedurálního jazyka s sebou nese řadu výhod, například:

- Omezení množství analýz s využitím statických příkazů (viz pododdíl 3.1.1).
- Lepší škálovatelnost systémů s velkým počtem uživatelů díky snížení zatížení *dictionary cache*.
- Možnost efektivně provádět komplexní operace v rámci jednoho volání z aplikace.
- Zvýšená přehlednost aplikace.

5.2.6 Optimalizační zásady pro psaní SQL dotazu

Existuje řada univerzálních zásad a doporučení, které by měly být během tvorby dotazů dodržovány. Jedná se především o následující pravidla a tipy (dle zdrojů [13] a [15]):

- Využívání jednoznačných složených podmínek a rovnosti.
- Zamezení výskytu transformovaných sloupců v predikátu.
- Tvorba méně komplikovaných příkazů specializovaných na konkrétní úkol.
- Využívání klausule WITH pro komplikované vnořené dotazy.
- Vhodná aplikace analytických funkcí.
- Pokud je to možné, vždy používat operátor rovnosti namísto podmínky LIKE.
- U komplexních agregací využívat funkce DECODE nebo CASE.
- Eliminace vícenásobného skenování výrazem CASE.
- Používání příkazů DML s RETURNING klausulí.

Využívání jednoznačných složených podmínek a rovnosti

Kdykoli je to možné, je žádoucí sestavit potřebný predikát pouze s pomocí operátoru rovnosti = a složené podmínky řetězit pomocí logické operace AND.

Zamezení výskytu transformovaných sloupců v predikátu

Není vhodné používat v predikátu pomocí funkcí transformované sloupce, na které poté nelze aplikovat indexový přístup s výjimkou odpovídajících funkčních indexů. Pokud je nutné funkci použít, měla by být na sloupci, pro který index neexistuje nebo by pro dotaz nebyl použit.

Je třeba dbát na případné implicitní typové konverze, které mohou rovněž zamezit případnému využití indexu. Pokud je například třeba filtrovat v dotazu data prostřednictvím indexovaného sloupce *charcol* typu VARCHAR2 a podoba této je podmínky následující:

```
WHERE charcol = 5;
```

jeho index nebude využit, neboť systém provede implicitní konverzi a predikát transformuje do podoby, kterou popisuje první odstavec tohoto podtitulu:

```
WHERE TO_NUMBER(charcol) = 5;
```

Vhodná podoba podmínky je tedy následující:

```
WHERE charcol = '5';
```

Pro správný odhad mohutnosti a selektivity podmínek je třeba se vyhnout složeným výrazům jako:

```
col1 = NVL (:b1,col1)
NVL (col1,-999) = …
TO_DATE(), TO_NUMBER()
```

Tvorba méně komplikovaných příkazů specializovaných na konkrétní úkol

Optimálnějším přístupem zpravidla bývá vytvoření více jednodušších příkazů, kde každý z nich plní svůj specifický úkol, než používání jednoho universálního dotazu. Pokud je přesto třeba provádět komplexní operaci v rámci jednoho příkazu, lze využít operátor UNION ALL nebo procedurální jazyk PL/SQL.

Využívání klausule WITH pro komplikované vnořené dotazy

Pokud se v dotazu vyskytuje několik shodných vnořených poddotazů, systém vykonává zbytečně vícekrát stejnou práci. Zejména v případě jejich komplexnějších struktur je užitečné přepsat hlavní dotaz pomocí klausule WITH a na vnořený poddotaz se odkazovat prostřednictvím stanoveného identifikátoru. V takové situaci je poddotaz proveden pouze jednou a jeho výstup je udržován v dočasném tabulkovém prostoru pro opakované použití v rámci vnějšího příkazu.

Eliminace vícenásobného skenování výrazem CASE

Například pro zobrazení počtů zaměstnanců, jejichž výše mzdy je menší než 2000, větší než 4000 a mezi těmito hodnotami, lze použít následující tři samostatné příkazy:

```
SELECT COUNT (*)
  FROM employees
  WHERE salary < 2000;

SELECT COUNT (*)
  FROM employees
  WHERE salary BETWEEN 2000 AND 4000;

SELECT COUNT (*)
  FROM employees
  WHERE salary>4000;
```

Ovšem v tomto případě by měla převládat snaha o získání výstupu pomocí jednoho příkazu, což je samo o sobě efektivnějším řešením. Navíc pokud například neexistuje index, během zpracování modifikovaného dotazu dojde k zpřístupnění každého řádku tabulky pouze jednou a nikoli třikrát. Lze tedy použít výrazy CASE:

```
SELECT COUNT (CASE WHEN salary < 2000
                   THEN 1 ELSE null END) count1,
       COUNT (CASE WHEN salary BETWEEN 2001 AND 4000
                   THEN 1 ELSE null END) count2,
       COUNT (CASE WHEN salary > 4000
                   THEN 1 ELSE null END) count3
  FROM employees;
```

Používání příkazů DML s RETURNING klausulí

Tuto klausuli lze použít u příkazů INSERT, UPDATE, nebo DELETE pro výběr a modifikaci dat během jediného volání. Tím se snižuje počet těchto volání do databáze.

5.2.7 Využívání analytických funkcí

Především v prostředí datových skladů, kde jsou častokrát pomocí dotazů zprostředkovávány různorodé statistické údaje, představuje analytika mocný nástroj, který v řadě případů umožňuje sestavit výrazně efektivnější dotaz než bez jejího využití. Oracle 11gR2 dle dokumentace [14] poskytuje 32 funkcí pro řešení širokého okruhu problémů. V rámci provádění dotazu dochází k jejich aplikaci na data po provedení všech operací kromě závěrečného řazení dle klausule ORDER BY a nejsou během ní nenávratně seskupeny řádky, jako v případě funkcí agregačních. Pomocí analytiky lze účinně řešit například následující problémy:

- Vyhledání specifického záznamu - například řádku s minimálním/maximálním atributem, řádku s hodnotou vybraného sloupce, která je nejblíže dané úrovni, atd.
- Vyhledání n-záznamů, které jsou řazeny dle vybraného atributu. Vyhledávat lze v kompletní množině nebo v rámci skupin jejich řádků, do kterých byly rozděleny dle stanovených kritérií.
- Zjištění předchozího/následujícího záznamu v množině řazené dle daných kritérií. Možnost využití hodnot tohoto řádku pro další použití v rámci dotazu.

5.3 Proces optimalizace

Jen málo systémů se nestane předmětem tohoto procesu. Ať už z důvodu zanedbaného návrhu, nebo nepředvídatelných provozních změn způsobených například neočekávaným nárůstem uživatelů či objemu uchovávaných dat. Optimalizace zpravidla zahrnuje fáze, které jsou popsány v následujících oddílech.

5.3.1 Identifikace nákladných (high load) příkazů

Společným rysem těchto příkazů jsou zejména zvýšené hodnoty těchto charakteristik (v závorkách jsou uvedeny sloupce pohledů datového slovníku, pomocí kterých se dají tyto údaje zobrazit):

- Přístupy k bufferu (V$SQLSTATS.BUFFER_GETS);
- Fyzické diskové čtení I/O (V$SQLSTATS.DISK_READS);
- Počet řazení (V$SQLSTATS.SORTS).

Pro nalezení příkazů, které mohou z výše uvedených důvodů způsobovat výkonnostní problémy, lze využít mimo jiné následující funkce a možnosti:

- Automatic Database Diagnostic Monitor;
- Automatic SQL tuning;
- Automatic Workload Repository;
- V$SQL view;
- Custom Workload;
- SQL Trace.

Uvedené automatické nástroje jsou stručně popsány v kapitole 6. Potíže s výkonem aplikace lze dle rozsahu jejich příčin rozdělit do dvou skupin:

- Problémy vztahující se ke konkrétnímu programu nebo k jejich malému množství.
- Problémy vyskytující se na úrovni kompletní aplikace.

Ladění specifického programu

V případech, kdy je znám užší rozsah problémových programů, potažmo i konkrétních dotazů, je krok identifikace relativně snadnou záležitostí. Pro upřesnění daného okruhu lze využít diagnostické funkce, jenž poskytuje centrální rozhraní Oracle *Enterprise Manager*, nebo přímo zkontrolovat jednotlivé exekuční plány pomocí nástrojů jako EXPLAIN PLAN či AUTOTRACE, které jsou popsány v kapitole 2 Exekuční plán.

Pokud není k dispozici konkrétní podoba SQL příkazů, jejichž je například generována dynamicky za provozu aplikace, je možné aktivovat funkci SQL_TRACE, která zajistí ukládání informací o prováděných příkazech do trasovacího souboru. Tyto údaje lze poté zobrazit v uživatelsky přívětivém formátu pomocí nástroje TKPROF a opět dle nich přesně zaměřit hledané příčiny problémů.

Ladění kompletní aplikace

První fází tohoto procesu je zpravidla určení období, během kterého bude následně daný systém monitorován za účelem shromáždění potřebných údajů. Pro vysokou vypovídající hodnotu těchto charakteristik se zpravidla volí doba, kdy zátěž systému vrcholí. Tabulka 5 zobrazuje seznam základních statistik. Identifikaci lze provést na základě analýzy v této tabulce uvedených dynamických výkonových pohledů, při které je například pro vybraný zdroj porovnáváno celkově využívané množství oproti míře nákladů na konkrétní příkaz.

Tabulka 5 - Základní statistiky využívané při identifikaci problémů v rámci aplikace

Statistiky	Umístění v pohledu
I/O operace	V$FILESTAT
Systémové statistiky	V$SYSSTAT
SQL statistiky	V$SQLAREA, V$SQL, V$SQLSTATS, V$SQLTEXT, VSQL_PLAN, VSQL_PLAN_STATISTICS

5.3.2 Hledání příčin nákladnosti a verifikace exekučních plánů

Po vyhledání konkrétních příkazů je třeba určit, zda lze dosáhnout zlepšení jejich výkonu. Předmětem zkoumání jsou v dotazu odkazované tabulky (případně pohledy) a jejich indexy, u kterých je vhodné upnout pozornost na to, zda jsou unikátní či nikoliv. Dále je třeba ověřit dostupnost a kvalitu souvisejících optimalizačních statistik, atd. Stěžejním úkonem fáze je verifikace exekučních plánů zkoumaných příkazů, která může pomoci odhalit nevhodné rozhodování optimalizátoru a důvody, jenž jej způsobují. Rozbor exekučního plánu je podrobněji popsán v oddíle 2.4.

Pokud provedená analýza vede k rozhodnutí, že dané příkazy mohou být prováděny efektivněji, je třeba podniknout odpovídající kroky.

5.3.3 Opatření

V extrémních situacích, kdy vinu za nízký výkon prokazatelně nese kompletně špatný návrh schématu, se zpravidla vyplatí neoptimalizovat stávající systém a investovat prostředky do návrhu nového a vhodnějšího řešení. V jiných případech lze provést drobnější úpravy právě v rámci struktury databáze, jako vytvoření indexů či clusterů nebo rozdělení tabulek na oddíly. Na úrovni volby vhodného exekučního plánu lze dále využít možnosti uvedené v následujícím oddíle 5.4, které optimalizátoru mohou umožnit, doporučit nebo nařídit, aby stanovil jiný a vhodnější postup pro provádění příkazu.

5.4 Možnosti ovlivnění volby exekučního plánu

5.4.1 Parametry optimalizátoru Oracle

Optimalizátor Oracle lze konfigurovat mimo jiné pomocí následujících základních parametrů:

- OPTIMIZER_MODE,
- CURSOR_SHARING,
- DB_FILE_MULTIBLOCK_READ_COUNT,
- OPTIMIZER_INDEX_CACHING,
- OPTIMIZER_INDEX_COST_ADJ,
- STAR_TRANSFORMATION_ENABLED,
- OPTIMIZER_FEATURES_ENABLE,
- OPTIMIZER_DYNAMIC_SAMPLING.

OPTIMIZER_MODE

Tento významný parametr slouží ke specifikaci přístupu při tvorbě plánu vzhledem k požadavkům na rychlost získaní určitého podílu výsledku příkazů. OPTIMIZER_MODE může nabývat následujících hodnot:

- *ALL_ROWS* - defaultní nastavení. Optimalizátor dle ceny hledá takový plán, aby byl s co nejmenším vytížením zdrojů sestaven kompletní výsledek příkazu. Tento mód má zajistit maximální propustnost systému.
- *FIRST_ROWS_n* - optimalizátor opět používá výběr plánu dle nejnižší ceny, ovšem cílem je tentokrát nejrychlejší získání prvních n záznamů - tedy minimální doba odezvy. Za n mohou být dosazeny hodnoty z množiny {1, 10, 100, 1000}.
- *FIRST_ROWS* - podobný účel jako FIRST_ROWS_n, ovšem zde není specifikován přesný počet záznamů a cílem je tedy minimální doba potřebná k získání několika prvních řádků. Za této konfigurace optimalizátor k nalezení řešení používá vyhledávání dle ceny v kombinaci s heuristickým algoritmem. Dle dokumentace [9] je možnost této konfigurace ponechávána pouze kvůli zpětné kompatibilitě a doporučuje se používat FIRST_ROWS_n.

Mimo nastavení tohoto parametru pomocí ALTER SYSTEM respektive ALTER SESSION lze požadované chování vynutit také na úrovní konkrétního dotazu použitím hintu (viz 5.4.2).

CURSOR_SHARING

V pododdíle 3.2.1 zmíněný parametr CURSOR_SHARING se vztahuje k možnosti používat stejný plán (sdílet kursor) pro příkazy, které se liší pouze v hodnotách literálů. Nakonfigurovat lze dvě úrovně. Defaultní hodnota EXACT způsobuje sdílení pouze totožných dotazů, kdy je pro každou jedinečnou kombinaci literálů vytvořen odpovídající plán. Úroveň FORCE naopak přistupuje k literálům jako k vázaným proměnnám (viz 5.2.4). Pokud je zpracováván příkaz obsahující literály, dojde k jejich nahrazení vázanými proměnnými. Tato nová podoba je uložena v *shared pool* (zobrazit ji lze pomocí sloupce SQL_TEXT v pohledu V$SQL) a využita pro sestavení exekučního plánu, který je posléze využíván pro opakované provádění stejných příkazů, lišících se pouze v obsažených literálech. Nastavením CURSOR_SHARING na FORCE lze tedy výrazně redukovat počet procesů *hard parse*.

DB_FILE_MULTIBLOCK_READ_COUNT

Tato konfigurace určuje počet bloků načtených v rámci jedné I/O operace během úplného procházení tabulky (*full table scan*) nebo indexu (*index fast full scan*). Vyšší hodnoty parametru snižují cenu úplného procházení tabulky (indexu) a jejich použití může tedy vést k výběru této metody přístupu namísto prohledávání prostřednictvím struktury indexu. Defaultní hodnota parametru odpovídá maximální velikosti I/O přenosu dané platformou, na níž je systém provozován.

OPTIMIZER_INDEX_CACHING

Tento parametr představuje informaci o tom, kolik procent bloků indexu by se mělo nacházet ve vyrovnávací mezipaměti *buffer cache*. Hodnoty lze volit v intervalu <0,100>. Daná konfigurace zásadně ovlivňuje odhad nákladů iterací v rámci predikátu IN a spojení vnořenými cykly. Pokud optimalizátor předpokládá, že nebude třeba investovat delší čas do I/O operací pro fyzické načtení bloků indexu, spíše jej využije a pro spojení aplikuje techniku vnořených cyklů (*nested loops*). V opačných případech raději zvolí metodu typu *hash* nebo sloučení po seřazení (*sort-merge join*) založenou na plném průchodu tabulek.

Jelikož optimalizátor nedokáže předpovídat, jaké bude vytížení dostupné paměti *buffer cache*, je výchozí hodnota parametru nastavena na 0. K její manuální modifikaci je třeba přistupovat velmi obezřetně, tedy přirozeně tak, aby nová konfigurace co nejvíce odpovídala skutečnosti a optimalizátor neurčoval zkreslené ceny plánů.

Pro datové sklady je dle knihy Oracle - Návrh a tvorba aplikací [15] obvykle vhodné použít defaultní konfiguraci. U systému OLTP hodnotu 90. Přesné nastavení tohoto parametru lze vynahradit vhodnou správou systémových statistik (viz pododdíl 1.2.1), které mohou optimalizátoru poskytnout zaměnitelnou informaci.

OPTIMIZER_INDEX_COST_ADJ

Tento parametr lze chápat jako ekvivalent pro předchozí OPTIMIZER_INDEX_CACHING, který ovšem popisuje, kolik bude v mezipaměti uloženo dat tabulky, přičemž čím vyšší hodnota, tím více záznamů je k dispozici v *buffer cache* a tím méně vhodné je přistupovat k nim přes index. Konkrétně lze dle zdroje [15] brát toto číslo jako poměr nákladů na jednoblokovou I/O operaci (čtení indexu) oproti ceně operace víceblokové (úplné prohledávání tabulky). Pokud je nastavena defaultní hodnota 100, není zvýhodněn ani jeden uvedený postup. V případě hodnoty 50 jsou výdaje na indexový přístup oproti plnému průchodu považovány za poloviční.

Validní rozsah hodnot je <0,10000>. Pro datové sklady je dle knihy Oracle - Návrh a tvorba aplikací [15] obvykle vhodné použít defaultní konfiguraci. U systému OLTP hodnotu 25. Přesné nastavení tohoto parametru lze vynahradit vhodnou správou systémových statistik (1.2.1), které mohou optimalizátoru poskytnout zaměnitelnou informaci.

STAR_TRANSFORMATION_ENABLED

Správa možnosti provádět transformaci hvězdicových dotazů, které se používají zejména v datových skladech. Při tomto procesu je daný dotaz přepsán pomocí vnořených poddotazů, čímž je umožněno k rozsáhlým tabulkám faktů přistupovat efektivně pomocí bitmapových indexů.

OPTIMIZER_FEATURES_ENABLE

Tento parametr řídí chování optimalizátoru vzhledem k odlišnostem jednotlivých verzí systému Oracle. Využití této konfigurace spočívá především v možnosti potlačením novějších funkcí do jisté míry vynutit původní chování optimalizátoru při upgradu databáze. Pokud je například nově zavedena verze 11.2.0.2, lze zadat příkaz:

```
ALTER SYSTEM SET OPTIMIZER_FEATURES_ENABLE = '10.2.0.5';
```

A tím při procesu optimalizace potlačit použití funkcí implementovaných po verzi 10.2.0.5. Poté lze například v testovacím prostředí vybrané relace ověřit výkon databáze při činnosti těchto zakázaných mechanismů a po případných optimalizačních opatřeních parametr nastavit na aktuální verzi.

OPTIMIZER_DYNAMIC_SAMPLING

Parametr nastavuje úroveň dynamického vzorkování pro sběr optimalizačních statistik (viz pododdíl 1.1.1).

5.4.2 Hinty

Hinty jsou instrukce, které se ve formě komentářů vkládají přímo do dotazu a optimalizátoru nařizují nebo doporučují, jaká dílčí řešení má volit při sestavování plánu. V provozním prostředí je žádoucí věnovat pozornost především správě podkladových

informací pro optimalizátor či jeho konfiguraci a samotné rozhodování ponechat na něm. Pokud existuje užitečná informace, která nemůže být optimalizátoru předána jiným způsobem, může vést aplikace hintů k lepším výsledkům, ale na rozdíl od většiny ostatních technik není jejich chování adaptivní vůči změnám v databázi. Když například výrazně naroste objem dat v tabulkách, mohou hinty optimalizátoru instruovat nevhodné řešení a jejich permanentní použití v běžném provozu tedy dle dokumentace [16] není považováno za vhodné. Uplatnění nalézají především během testování výkonu jednotlivých dotazů, kdy umožňují ověřit například cenu vybraných přístupových cest nebo pořadí spojení. Dostupné hinty je možné zobrazit pomocí pohledu V$SQL_HINT a lze je dělit do následujících kategorií:

- Hinty pro optimalizace přístupy a cíle;
- Hinty pro správu optimalizačních funkcí;
- Hinty pro přístupové metody;
- Hinty pro pořadí spojení;
- Hinty pro metody spojení;
- Hinty pro on-line upgrade aplikace;
- Hinty pro paralelní provádění;
- Hinty pro transformace dotazu.

5.4.3 Vytvoření indexů

Indexy jsou jedním ze základních konceptů optimalizace SQL dotazu a během vývoje a správy každého kvalitního systému by jejich použití vždy mělo být zváženo. Pokud analýza exekučního plánu odhalí neefektivní plný průchod tabulkou, je na místě vytvořit index. Ovšem indexy mohou být užitečné nejen svým primárním účelem. Při požadavku řazení mohou být například díky nim záznamy načítány v již utříděném sledu. Dále pokud jsou v jeho struktuře obsaženy všechny sloupce, které jsou potřebné pro sestavení výsledku příkazu, lze se úplně vyhnout přístupu k dané tabulce. Tento princip je ještě více podpořen mechanismem spojování indexů. Je tedy také vhodné zvážit tvorbu vícesloupcových indexů a případně oprostit provádění dotazu od zbytečného načítání záznamů z tabulek. Databáze Oracle poskytuje následující druhy těchto objektů:

- B-tree indexy;
- Bitmapové indexy (bitmap, bitmap join indexes);
- Funkční indexy (function-based indexes);
- Aplikační doménové indexy (application domain indexes);
- Indexově organizované tabulky (index-Organized Tables).

5.4.4 Změna parametrů velikostí oblastí instance

Dostatek a vhodná správa paměti je důležitým atributem výkonného systému. Zásadní význam mají rychlé mezipaměti *cache* a také úložiště, jenž mohou uchovávat výsledky vykonané práce pro jejich opětovné použití (viz koncepty *shared pool* a *cursor sharing*). Dokumentace Oracle důrazně doporučuje ponechat správu dostupné paměti na nástroji

Automatic Memory Management (AMM), jehož činnost lze konfigurovat pomocí dvou parametrů:

- *MEMORY_TARGET* - celková velikost použitelné paměti instance serveru, která by měla být využívána.
- *MEMORY_MAX_TARGET* - maximální velikost použitelné paměti pro instanci serveru.

Paměťový prostor instance systému Oracle je rozdělen na dva základní segmenty SGA a PGA (viz oddíl 3.2 a pododdíl 3.1.1), které lze, mimo rámec aktivity modulu AMM, spravovat separátně pomocí následujících parametrů:

- SGA_TARGET,
- PGA_AGGREGATE_TARGET.

Při manuální konfiguraci lze využít nástroje *Memory Advisor*.

SGA_TARGET

Konfigurace velikosti paměti SGA, ve které jsou pro opakované používání uchovávány například plány analyzovaných příkazů. Dalšími důležitými segmenty jsou mezipaměti jako *buffer cache*, *server reset cache*, *data dictionary cache*, atd. Kapacitu těchto oblastí lze také nastavit separátně pomocí příslušných parametrů.

PGA_AGGREGATE_TARGET

Parametr udává souhrnnou velikost PGA paměti dostupné všem procesům serverů připojených k instanci. PGA obsahuje pracovní oblasti *SQL work areas*, které jsou při provádění příkazu používány pro operace jako je třídění či spojování tabulek. Velikost těchto prostorů má vliv na cenu plánu, který například při větší kapacitě *hash area* uvažuje nižší náklady na spojení typu *hash*. Přístup ke správě pracovních oblastí udává konfigurace WORKAREA_SIZE_POLICY. Pokud je nastavena na MANUAL, kapacita konkrétních podoblastí vychází z manuálního nastavení parametrů:

- SORT_AREA_SIZE,
- HASH_AREA_SIZE,
- BITMAP_MERGE_AREA_SIZE.

5.4.5 Materializovaný pohled

Především v prostředí datových skladů dochází k častému používání nákladných dotazů, jenž provádějí například rozsáhlé agregace či operace řazení. Pro podporu jejich výkonu lze část jejich práce uložit permanentně prostřednictvím materializovaných pohledů. Tyto objekty uchovávají výsledek zadaného dotazu analogickým způsobem, jako běžné tabulky svá data. Platí, že dosažení nižších nákladů na provedení dotazu lze dosáhnout za cenu režijních operací, jejichž úkolem je udržovat aktuálnost záznamů pohledu.

Pro proces aktualizace záznamů může být využito specializovaných tabulek označovaných jako log materializovaného pohledu (*materialized view log*). Pokud jsou pomocí DML příkazů modifikovány podkladové tabulky materializovaného pohledu, systém uloží informace o těchto změnách do příslušných logů a tyto údaje použije pro aktualizaci záznamů pohledu. Tímto způsobem je dosaženo poměrně rychlých procesů renovace dat materializovaných pohledů, které se označují jako inkrementální nebo rychlá obnova. Nevýhodou jsou samozřejmě náklady na režijní operace pro vkládání dat při provádění DML modifikací.

Pokud dané logy nejsou k dispozici, musí databáze aplikovat kompletní obnovu, během které je příslušný materializovaný pohled prakticky znovu vytvořen. Tento přístup je zpravidla časově nákladnější.

Na daný materializovaný pohled se lze v dotazu přímo odkazovat, nebo jej může optimalizátor využít transparentně pomocí mechanismu *query rewrite* (viz pododdíl 4.1.4), jehož aktivita podléhá nastavení parametru QUERY_REWRITE_ENABLED.

6 Oracle optimalizační nástroje

Systém Oracle disponuje řadou nástrojů a konceptů, které usnadňují a do určité míry automatizují správu systému. Těmto modulům je věnována následující kapitola, která stručně popisuje principy a možnosti, jenž přináší. Primárním rozhraním pro jejich obsluhu je centrální správce databáze *Enterprise Manage*r (EM), který poskytuje přehledné GUI ovládací prvky pro provádění potřebných operací a jeho používání doporučuje oficiální dokumentace [19]. Ve verzích systému, kde není EM k dispozici, a v dalších vybraných situacích lze dále popsané nástroje využívat prostřednictvím specializovaných balíčků, mezi které patří zejména soubor rutin DBMS_SQLTUNE.

6.1 Automatic Workload Repository

AWR je vestavěné úložiště, sloužící k uchovávání výkonových statistik pro detekci problémů a podporu jejich eliminace. Tyto údaje jsou shromažďovány automaticky v pravidelných intervalech dle konfigurace a představují základ pro všechny funkce zajišťující samosprávu v oblasti ladění databáze Oracle. Výsledné množiny jednotlivých sběrů charakterizující systém a aktivity uvnitř něho v daném intervalu se označují jako snímky (*snapshots*), které konkrétně obsahují následující statistiky:

- Objektové statistiky přístupu k databázovým segmentům a jejich využití.
- Statistiky časového modelu.
- Některé statistiky systému a relace.
- Nákladné SQL příkazy, které nejvíce zatěžují systém.
- *Active session history* statistiky vztahující se k nedávným aktivitám v rámci relace.

Při výchozím nastavení je snímek generován každou hodinu, přičemž jeho data jsou poté uchovávána po dobu 8 dní. Správa AWR je primárně prováděna prostřednictvím rozhraní EM, případně je možné požadovaná data zobrazit pomocí reportů nebo pohledů, jako například DBA_HIST_SNAPSHOT, který obsahuje seznam aktuálně uchovávaných snímků. Pro tvorbu, modifikaci a odstranění lze také využít procedury balíčku DBMS_WORKLOAD_REPOSITORY.

Proces ukládání záznamů do AWR je podřízen hodnotě parametru STATISTICS_LEVEL a při defaultním nastavení na úroveň TYPICAL je povolen, přičemž jeho deaktivace je dle dokumentace [17] důrazně nedoporučena. Zvolená úroveň tohoto významného parametru ovlivňuje aktivitu mnoha funkcí, jejichž seznam lze zobrazit pomocí pohledu V$STATISTICS_LEVEL.

6.2 Automatic Database Diagnostic Monitor (ADDM)

Po každém shromáždění údajů do AWR (viz předchozí oddíl 6.1) je automaticky aktivována činnost tohoto diagnostického nástroje, který provádí následující úkony:

- Analyzuje AWR data.
- Provádí diagnostiku příčin případných problémů.
- Poskytuje řešení nalezených problémů.
- Identifikuje bezproblémově pracující oblasti systému.

Analýza může být aplikována na libovolnou dvojici snímků v AWR, ovšem zpravidla je prováděna automaticky, přičemž vychází z posledních dvou záznamů. Ve výchozím nastavení tedy pracuje s daty vztahujícími se zejména k předchozí hodině provozu, která využívá k nalezení problémů, jejich příčin a případných řešení na základě snižování hodnot veličiny *DB time*. Toto kritérium představuje celkový čas databázových volání všech aktivních relací od okamžiku spuštění instance a jeho minimalizace vede k maximální propustnosti systému.

Činnost ADDM podléhá nastavení dvou parametrů. Kromě již uvedeného STATISTICS_LEVEL (viz oddíl 6.1) se jedná o parametr CONTROL_MANAGEMENT_PACK_ACCESS, jenž je v *Enterprise Edition* systému Oracle defaultně konfigurován na hodnotu DIAGNOSTIC+TUNING, která funkci povoluje.

S ADDM lze pracovat analogicky jako s ostatními zde popisovanými nástroji, tedy prostřednictvím primárního rozhraní EM nebo s využitím specializovaných procedur, které v tomto případě sdružuje balíček DBMS_ADDM. Výstupní report popsaný v následujícím pododdíle lze zobrazit také pomocí příslušných pohledů, mezi které se řadí například DBA_ADDM_FINDINGS.

6.2.1 Výstup ADDM

Každý výstup činnosti ADDM je sadou položek, které mohou být následujícího typu:

- Příčina problému s výkonem databáze.
- Příznak problémů s výkonem databáze.
- Informace o databázi, které se nevztahují přímo k výkonnostním problémům, ale jsou užitečné pro pochopení dějů v systému.
- Informace o aspektech, které mohou znehodnotit práci ADDM, jako například chybějící údaje v AWR.

V případě výskytu výkonnostního problému ADDM zpravidla poskytne seznam doporučení k jeho odstranění či potlačení, jenž může být mimo jiné tvořen těmito kroky:

- Upgrade hardware. Přidání CPU nebo změna nastavení I/O subsystému.
- Změna konfigurace inicializačních parametrů databáze.
- Změny ve schématu. Například zavedení *hash* dělení tabulek nebo vytvoření indexů.
- Změny v aplikaci jako použití vázaných proměnných v dotazech.
- Použití dalších nástrojů Oracle jako například *SQL Tuning Advisor* (viz dále).

Konkrétní report, převzatý z oficiální dokumentace [18], může mít následující podobu:

```
FINDING 1: 31% impact (7798 seconds)
-----------------------------------
SQL statements were not shared due to the usage of literals. This resulted in
additional hard parses which were consuming significant database time.

RECOMMENDATION 1: Application Analysis, 31% benefit (7798 seconds)
  ACTION: Investigate application logic for possible use of bind variables
    instead of literals. Alternatively, you may set the parameter
    "cursor_sharing" to "force".
  RATIONALE: SQL statements with PLAN_HASH_VALUE 3106087033 were found to be
    using literals. Look in V$SQL for examples of such SQL statements.
```

Uvedený příklad poukazuje na vyšší počet postupů *hard parse* způsobený nesdílením plánů z důvodu výskytu literálů v dotazech. Dále poskytuje dvě doporučení, přičemž odhaduje, že jejich aplikace ve zkoumaném období o 31% zredukuje celkový *DB time*:

- Zkontrolovat aplikační logiku a zvážit použití vázaných proměnných namísto literálů.
- Alternativně nastavit parametr CURSOR_SHARING na FORCE.

Aplikaci těchto kroků zdůvodňuje výskytem konkrétních příkazů s uvedenou PLAN_HASH_VALUE.

6.3 SQL Tuning Advisor

Tento modul přebírá jako vstup jeden nebo více dotazů a aplikuje na ně mechanismy pro ověření a případné doporučení vybraných opatření. Jeho činnost probíhá zpravidla automaticky během oken údržby systému, ale může být vyvolána i na základě manuálního pokynu či reaktivně. Tyto dva přístupy jsou dále popsány v příslušných pododdílech.

6.3.1 Automatic SQL Tuning Advisor

Během procesu automatického ladění provádí databáze následující kroky:

1. Prohledávání AWR za účelem výběru nákladných SQL příkazů, které jsou kandidáty na ladění. Zvoleny jsou pouze ty, které vykazují potenciál ke zlepšení. Ignorovány jsou například dotazy rekurzivní nebo nedávno laděné (v posledním měsíci).
2. Vybrané příkazy jsou jednotlivě laděny nástrojem *SQL Tuning Advisor*. Během těchto procesů databáze zvažuje různá opatření, pro něž může generovat report. Ovšem určené kroky nemůže sama realizovat. Výjimkou je pouze implementace SQL Profilů (viz oddíl 6.4).
3. Testování daných SQL Profilů, které databáze provádí spouštěním příkazu s jejich využitím a naopak. Pokud kontrola dopade dle daných kritérií úspěšně a parametr ACCEPT_SQL_PROFILES je nastaven na TRUE, je daný profil přijat. V opačném případě je pouze vygenerováno doporučení k jeho vytvoření.
4. Případná implementace profilů

Pro plánování činnosti nástroje *SQL Tuning Advisor* během automatických oken údržby je třeba použít funkce ENABLE respektive DISABLE balíčku DBMS_AUTO_TASK_ADMIN. Aktivita modulu je také podřízena nastavení stěžejního parametru STATISTICS_LEVEL. Konfiguraci lze provádět prostřednictvím procedury SET_AUTO_TUNING_TASK_PARAMETER balíčku DBMS_AUTO_SQLTUNE, který lze dále použít i pro zobrazení detailních reportů či spuštění poradce pomocí procedury REPORT_AUTO_TUNING_TASK respektive EXECUTE_AUTO_TUNING_TASK.

6.3.2 Manuální ladění s využitím nástroje SQL Tuning Advisor

K manuálnímu spuštění tohoto poradce zpravidla dochází na podnět vydaný ADDM, který také v dané situaci poskytne vstupní příkaz uložený v AWR. Záznamy z tohoto úložiště je možné ladit i přímo, bez zapojení správce ADDM. V takovém případě je ovšem třeba cílové příkazy v repositáři vyhledat svépomocí. Další možností je aplikovat mechanismy nástroje *Tuning Advisor* na vybrané příkazy v *shared SQL area*, jenž byly volány před pořízením posledního snímku do AWR, který je tudíž zatím neobsahuje. Vstupem mohou dále být i předem vytvořené *SQL tuning sets* (viz dále v oddíle 6.5), které umožňují procesu najednou předat množinu vybraných příkazů, jenž například mohly být prováděny v rámci běžného provozu a jejich ladění je žádoucí provést na systému testovacím.

Pro manuální ladění prostřednictvím modulu *Tuning Advisor* je opět doporučeno volit centrální GUI rozhraní EM, které poskytuje uživatelsky přívětivé prostředí. Ve specifických případech je možné požadované úkony provést pomocí balíčku DBMS_SQLTUNE, s jehož využitím proces zpravidla probíhá následovně:

1. Vytvoření *SQL Tunning sets* (pro ladění více příkazů).
2. Vytvoření úkolu (*SQL tuning task)*.
3. Provedení úkolu (*SQL tuning task)*.
4. Zobrazení výsledků úkolu (*SQL tuning task)*.
5. Realizace případných opatření.

6.4 SQL Profiles

SQL profil je sada informací vztahujících se k danému SQL příkazu, která je sestavena v rámci automatických procesů jeho ladění a posléze využívána optimalizátorem pro zvýšení porozumění problému vedoucímu k lepší volbě exekučního plánu. Konkrétní přínos těchto údajů je především v přesnějších odhadech mohutnosti a selektivity. Jak je uvedeno výše, předběžná tvorba probíhá zpravidla automaticky během činnosti modulu *SQL Tuning Advisor*, který může vydat doporučení k používání profilu podložené přehledným zdůvodněním. V daném reportu je obsažen také příkaz, jehož provedení zajistí akceptování profilu, které resultuje k jeho skutečnému využívání.

Obsažené údaje jsou poměrně nezávislé na modifikacích souvisejících s daným příkazem, jako je například změna rozdělení dat a dle dokumentace [19] obecně platí, že profily není třeba aktualizovat. Přesto se po delším časovém úseku mohou stát zastaralými. Mechanismus automatického ladění do jisté míry ošetřuje používání těchto neaktuálních

údajů, neboť v případě ztráty realističnosti vedoucí k volbě nevhodných plánů je zpravidla v rámci automatického ladění navržen profil nový.

Vytvořené profily obsahuje pohled DBA_SQL_PROFILES. Pokud je třeba pracovat mimo rámec rozhraní EM, lze opět využít balíček DBMS_SQLTUNE a dle potřeby provádět následující operace:

- *Akceptování* - ALTER_SQL_PROFILE.
- *Změna* - ALTER_SQL_PROFILE. Modifikovat lze atributy STATUS, NAME, DESCRIPTION, a CATEGORY[2].
- *Transport* - Proces probíhá dvoufázově. Nejprve je daný objekt procedurou PACK_STGTAB_SQLPROF exportován ze schématu SYS do pomocné pracovní tabulky předem vytvořené rutinou CREATE_STGTAB_SQLPROF a poté je proveden import z tohoto přechodného umístění, který je řízen procedurou UNPACK_STGTAB_SQLPROF.
- *Odstranění* - DROP_SQL_PROFILE.

6.5 SQL Tuning Sets

SQL tuning sets (STS) jsou objekty, které obsahují sadu SQL příkazů společně s informacemi souvisejícími s jejich dřívějším provedením. Pro každý jednotlivý příkaz mohou konkrétně obsahovat následující data:

- Kontext jejich provedení zahrnující například uživatelské schéma, název modulu aplikace, seznam vázaných proměnných nebo parametry prostředí, v němž byl daný kursor zkompilován.
- Základní statistiky charakterizující výkonnost provedení příkazu. Jedná se o:
  - celkový čas a čas využití CPU,
  - počet bloků načtených z mezipaměti a disku,
  - počet bloků načtených z disku,
  - počet zpracovaných řádků,
  - načtené kursory,
  - počet provedení a úplný počet provedení,
  - cena příkazu,
  - typ příkazu.
- Příslušné exekuční plány a statistiky řádkových zdrojů.

[2] Atribut CATEGORY určuje, které relace smějí použít daný profil. Jeho nastavením lze například zajistit, aby vytvořeným profilem byly ovlivněny pouze vybrané relace, ve kterých je bez rizika otestováno chování optimalizátoru při využití této sady informací.

Přínos uchovávání těchto údajů spočívá především v jejich předání optimalizačním modulům jako ADDM či *SQL Tuning* a *SQL Access Advisor*, které případně provedou potřebné ladění. Užitečná je i možnost jejich exportu - tedy přenesení problému, jenž reprezentují, z rizikového provozního prostředí na testovací systém, kde se opět mohou stát vstupní informací pro častokrát nákladné optimalizační procesy vykonávané uvedenými nástroji.

Seznam a související data *SQL tuning sets* je možné zobrazit pomocí pohledů, jako například DBA_SQLSET. Pro jejich správu opět existují dvě varianty. Doporučeno je využívat rozhraní EM. Pokud tento centrální nástroj není k dispozici, lze aplikovat již zmíněný balíček DBMS_SQLTUNE. Životní cyklus objektů STS řízený pomocí jeho procedur a funkcí zpravidla probíhá následovně:

1. Vytvoření prázdného objektu pomocí procedury CREATE_SQLSET.
2. Aplikace procedury LOAD_SQLSET pro naplnění nově vytvořené sady vybranými příkazy a k nim příslušejícími daty. Standardním zdrojem této operace je AWR, *shared SQL area* nebo jiná STS.
3. Zobrazení načteného obsahu tabulkovou funkcí SELECT_SQLSET s možností formátování výstupu pomocí řazení či aplikace filtrů.
4. Případná modifikace s využitím procedur UPDATE_SQLSET respektive DELETE_SQLSET.
5. Transport STS. Proces probíhá dvoufázově, obdobně jako u přenosu SQL profilů. Nejprve je daný objekt procedurou PACK_STGTAB_SQLSET exportován do pomocné pracovní tabulky předem vytvořené rutinou CREATE_STGTAB_SQLSET a poté je proveden import z tohoto přechodného umístění, který je řízen procedurou UNPACK_STGTAB_SQLSET.
6. Odstranění sady pomocí rutiny DROP_SQLSET.

6.6 SQL Access Advisor

Nástroj *SQL Access Advisor* poskytuje poradenství ohledně následujících optimalizačních prvků:

- Indexy;
- Materializované pohledy a jejich aktualizace s využitím logů (viz 5.4.5 Materializovaný pohled);
- Dělení (*partitioning*).

Koncept je založen, podobně jako v případě komponentu *Tuning Advisor*, především na generování doporučení k aplikaci stanovených opatření. Služby poradce lze využívat prostřednictvím rozhraní EM nebo rutin specializovaného balíčku DBMS_ADVISOR. Postup jednotlivých ladění se zpravidla skládá z následujících etap:

1. Vytvoření úkolu;
2. Definování zatížení (*workload*);

3. Generování doporučení;
4. Zobrazení a realizace doporučení.

Vytvoření úkolu

Úkol (*task*) poradce je struktura v datovém slovníku, která obsahuje informace vztažené k danému procesu ladění. K jeho tvorbě dochází automaticky při práci v rozhraní EM nebo během aplikace procedury QUICK_TUNE, která umožňuje provést kompletní proces získání doporučení pro jeden vybraný SQL příkaz a volitelně dokáže stanovené kroky i přímo realizovat. Pokud ladění není vykonáváno těmito způsoby, je třeba úkol vytvořit manuálně pomocí procedury CREATE_TASK. Parametry procesu ladění v rámci již existujícího úkolu lze konfigurovat prostřednictvím rutiny SET_TASK_PARAMETER.

Definování zatížení (workload)

V tomto kroku je třeba určit předmět ladění. *Workload* představuje určitý podíl příkazů, se kterými systém pracuje a k nim se vztahující sadu potřebných statistik a atributů. Když je optimalizována kompletní aplikace, hovoříme o tzv. plné zátěži, kdy může *advisor* doporučit i odstranění nepoužívaných indexů a materializovaných pohledů, neboť pracuje se všemi situacemi, kde mohou být použity. Pokud se naopak ladění týká jen konkrétního dotazu nebo vybrané podmnožiny, nelze vyloučit přínos těchto objektů pro výkon ostatních nezahrnutých příkazů.

Vhodným přístupem při stanovení zatížení je použití *SQL tuning sets*, které obsahují mimo samotných příkazů také velmi užitečné statistiky. Tyto sady je třeba s daným úkolem propojit pomocí procedury ADD_STS_REF.

Vytvořené úkoly i s definovaným zatížením lze uchovávat v podobě šablon, které v budoucnu umožní urychlit proces přípravy ladění.

Generování doporučení

Pro provedení úkolu je určena procedura EXECUTE_TASK, která zajistí uložení vygenerovaných doporučení do úložiště úkolů *SQL Access Advisor repository* v datovém slovníku.

Zobrazení a realizace doporučení

Výstupy poradce mohou instruovat nejen k jednoduchým operacím, ale také k poměrně komplexním sekvencím kroků. Doporučením může být například realizace rozdělení již existujících tabulek, vytvoření indexů či materializovaných pohledů a shromáždění souvisejících statistik. Pro jejich zobrazení mimo prostředí EM lze využít katalogové pohledy nebo skripty vygenerované pomocí procedury GET_TASK_SCRIPT, jenž představují sekvenci příkazů pro realizaci stanovených úkonů.

6.7 SQL Test cases

Tento nástroj je implementován od verze 11gR2. *Test cases* umožňují vytvářet skripty, pomocí nichž lze uchovávat informace vztahující se k jednomu konkrétnímu provedení daného příkazu, které tak lze opakovaně reprodukovat za relativně stejných podmínek. Pokud například v systému za určité konfigurace a charakteristiky podkladových tabulek dochází k situaci, kdy nějaký dotaz nepracuje efektivně, je vhodné tento „případ“ zaznamenat pomocí *test cases*. Výsledný skript lze poté při ladění použít k simulaci problému na testovacím systému. Výhodou je i možnost odeslat jej na technickou podporu společnosti Oracle, kde na základě něho mohou navrhnout řešení potíží. Konkrétní údaje, jež tento nástroj zaznamenává, jsou následující:

- Právě uskutečňované dotazy.
- Definice tabulek, indexů a dalších objektů.
- Volitelně lze zahrnout i data příslušných tabulek.
- Optimalizační statistiky.
- PL/SQL funkce, procedury a balíčky.
- Nastavení inicializačních parametrů.

Test cases lze vytvářet a spravovat v rámci nástroje EM nebo manuálně pomocí balíčku DBMS_SQLDIAG.

7 Novinky Oracle 12 v oblasti optimalizace

Verze systému Oracle 12c (12.1) se oproti předchozím vydáním vyznačuje především důrazem na technologii *cloud computing*, ovšem přináší i řadu vylepšení zaměřených primárně na zvýšení výkonu a optimalizaci. Kromě nových přístupů k problematikám, jakými jsou například I/O operace či víceprocesorový a vícevláknový provoz, také vylepšuje i zavádí funkce specializované přímo na zkvalitnění podpory optimalizátoru při jeho rozhodování. Ten se tak stává ještě více precizním a přizpůsobivým, přičemž z hlediska optimalizace dotazů právě jeho zvýšená adaptivita značí směr vývoje a představuje nejvýraznější pokrok, jenž přichází z verzí 12c. Systém konkrétně přináší inovace v konceptech statistik a správy plánů a využívá níže uvedené nové funkce a techniky. Významná problematika adaptivní optimalizace dotazů je na základě dokumentu [22] a oficiální dokumentace [23] dále popsána podrobněji.

- Adaptivní optimalizace dotazů;
  - Adaptivní exekuční plán;
  - Adaptivní statistiky;
- Optimalizační statistiky;
  - Nové typy histogramů;
  - On-line sběr statistik;
  - Statistiky globálních dočasných tabulek vztahující se k dané relaci;
  - Nové rutiny balíčku DBMS_STATS pro reporting statistik;
- Optimalizační techniky;
  - Partial Join Evaluation;
  - Null accepting semi-joins;
  - Scalar Subquery Unnesting;
  - Multi-Table Left Outer Join;
- Inicializační parametry;
  - OPTIMIZER_ADAPTIVE_FEATURES;
  - OPTIMIZER_ADAPTIVE_REPORTING_ONLY;

7.1 Adaptivní optimalizace dotazů

Systém Oracle disponuje poměrně kvalitními nástroji na správu optimalizačních statistik, přesto mohou nastat případy, kdy je exekuční plán sestavován na základě nereálných hodnot těchto údajů. Tuto nevhodnou volbu dokáže eliminovat právě adaptivní optimalizace, která zahrnuje koncepty adaptivních exekučních plánů a statistik.

7.1.1 Adaptivní exekuční plán

Tato funkce umožňuje během samotného vykonávání příkazu shromažďovat skutečné exekuční statistiky, na základě kterých může být realizována korekce plánu, jenž je pro dané provádění používán. Nejprve je s využitím klasických statistik sestaven výchozí adaptivní plán, jenž je ovšem složen z více alternativních subplánů, mezi kterými lze během exekuce přepínat. Tento postup také na vybraných klíčových pozicích obsahuje

kolektory statistik, které v daných fázích shromažďují reálné run-time statistiky a uchovávají výsledné řádky získané během dosavadního procesu. Pokud je na základě těchto údajů uznáno za vhodné, je pro další postup použit jiný subplán, dle kterého je poté příkaz prováděn i v budoucnosti.

K adaptivnímu chování optimalizátoru dochází na základě nastavení dvou parametrů:

- *OPTIMIZER_FEATURES_ENABLE* - parametr musí být konfigurován na verzi 12.1.0.1, nebo pozdější.
- *OPTIMIZER_ADAPTIVE_REPORTING_ONLY* - musí být nastavena defaultní hodnota FALSE.

Příkaz EXPLAIN PLAN (viz pododdíl 2.2.1) defaultně poskytuje výchozí alternativu adaptivního plánu. Jeho finální variantu pro již vykonané příkazy lze zobrazit pomocí tabulkové funkce DBMS_XPLAN.DISPLAY_CURSOR. Oba uvedené způsoby také v poznámce NOTE obsahují informaci o tom, zda je daný plán adaptivní. Tento údaj lze získat i prostřednictvím sloupce IS_RESOLVED_ADAPTIVE_PLAN pohledu V$SQL.

Konkrétními aspekty, které lze během provádění příkazu adaptovat vzhledem k run-time statistikám, jsou metody operací spojování a distribuce při paralelním zpracování.

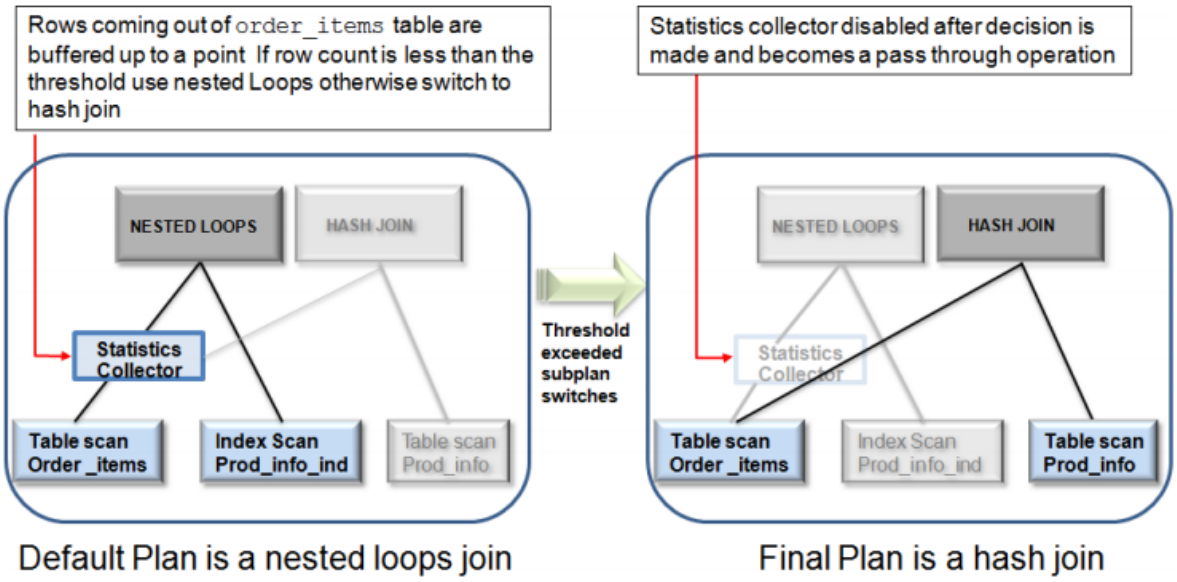
Adaptivní metody spojení

Optimalizátor verze 12.1 v rámci adaptivního plánu umožňuje přechod z typu spojení vnořenými cykly (*nested loops join*) na variantu slučování *hash* a naopak. Pokud je jako výchozí alternativa zvoleno sloučení po seřazení (*sort metge join*), je tímto přístupem zpracována kompletní operce bez možnosti adaptace.

Například při následujícím dotazu spojujícím tabulky *order_items* a *prod_info*:

```
SELECT product_name
FROM   order_items o, prod_info p
WHERE  o.unit_price = 15
AND    quantity > 1
AND    p.product_id = o.product_id;
```

Může být sestaven adaptivní plán lišící se v metodě daného spojení. Jako výchozí může být zvolena varianta vnořených cyklů. V takovém případě adaptivní systém během provádění této operace testuje skutečný počet zpracovávaných řádků pocházejících z levého zdroje. Pokud je překročena stanovená mez, dojde k změně používané techniky na metodu *hash*. Proces je znázorněn na následujícím obrázku (Obrázek 8).



Obrázek 8 - Adaptivní exekuční plán pro spojení tabulek, převzato z [22]

Adaptivní paralelní distribuční metody

Distribuční metoda pro paralelismus závisí mimo jiné také na očekávaném počtu řádků, který bude během operace zpracován, přičemž špatný odhad této hodnoty může přirozeně vést ke snížení výkonu příkazu. Tento nežádoucí jev má za úkol potlačit adaptivní technika hybridní *hash* distribuce, pro kterou optimalizátor do exekučních plánů vkládá před paralelní procesy statistické kolektory. Shromážděné údaje jsou opět využity k určení vhodné varianty metody, kterou může být například alternativa *broadcast* nebo, při překročení stanovené hranice počtu řádků, právě distribuce typu *hash*.

7.1.2 Adaptivní statistiky

Jsou používány zejména v situacích, kdy je predikát v dotazu shledán natolik komplexním, že pro kvalitní odhad jeho selektivity zřejmě nebudou dostačovat pouze základní optimalizační statistiky. Případně se uplatňují, pokud je během provádění příkazu zjištěna velká odlišnost skutečných dat od výchozího odhadu. Problematiku adaptivních statistik lze rozdělit do následujících okruhů:

- Dynamické statistiky;
- Automatická reoptimalizace;
- Direktivy SQL plánu (SQL plan directives).

Dynamické statistiky

Pokud optimalizátor rozhodne, že dostupné základní statistiky neposkytují dostatečnou nebo přesnou informaci, může použít vzorkování pro shromáždění dynamických statistik, které zahrnují například následující údaje:

- Počet bloků tabulek a indexů.
- Kardinalitu tabulek a jejich spojení.

- GROUP BY statistiky.

Koncept tohoto procesu není novinkou verze 12c, neboť ta jej pouze vylepšuje. Předchozí verze (viz pododdíl 1.1.1) byla zkvalitněna o možnost sběru statistik vztažených nejen k jednotlivým objektům, ale také k jejich spojením či k agregacím dle klausule GROUP BY. Také původní rozsah hodnot konfiguračního parametru OPTIMIZER_DYNAMIC_SAMPLING byl rozšířen o úroveň 11, která umožňuje optimalizátoru provést dynamické vzorkování vždy, kdy jej uzná za přínosné. Takto získané charakteristiky jsou uloženy persistentně a mohou být dále využívány v rámci optimalizace dalších příkazů.

Automatická reoptimalizace

Během jednoho konkrétního provádění příkazu je adaptivní korekce plánu možná jen pro určité jeho specifika, jako jsou použité metody spojení. Ovšem například výchozí pořadí spojení měnit nelze. Přesto lze získané exekuční informace využít k zefektivnění výkonu, ale pouze pro v budoucnosti sestavované exekuční plány. Na konci fáze prvního provádění příkazu optimalizátor porovnává zaznamenané statistiky s hodnotami odhadů, z nichž při volbě optimálního postupu vycházel. Pokud odhalí významné rozdíly, je během příštího zpracovávání daného příkazu provedena reoptimalizace na základě shromážděných exekučních statistik. Konkrétní formy popisovaného procesu jsou následující:

- Statistics Feedback;
- Performance Feedback.

Statistics Feedback

Tento mechanismus opravuje neefektivní plány znehodnocené nesprávnými odhady kardinality. Základní cyklus zahrnuje následující kroky:

1. Generování exekučního plánu během prvního zpracování daného příkazu a jeho provedení. Na konci procesu optimalizátor porovnává výchozí odhady kardinality výstupní množiny každé operace s jejich reálnými protějšky, které jsou v případě významné odlišnost uchovány pro pozdější využití během reoptimalizace. Při uvedené neshodě je dále daný příkaz označen jako reoptimalizovatelný (sloupec IS_REOPTIMIZABLE pohledu V$SQL je roven hodnotě Y). Optimalizátor pro tento mechanismus realizuje sledování statistik v následujících situacích:

   - Tabulka nemá statistiky.
   - Více konjunktivní nebo disjunktivní predikáty.
   - Predikáty obsahující komplexní operátory.

   Během této fáze mohou také být vytvořeny direktivy SQL plánu (viz podtitul dále), jenž umožňují získané exekuční charakteristiky využít i pro volbu plánu odlišných příkazů, které ovšem obsahují například obdobný predikát.

2. Po prvním zpracování je deaktivován sběr statistik pro *statistics feedback*.

3. V případě dalšího spuštění je daný dotaz (označený jako IS_REOPTIMIZABLE) v rámci procesu *hard parse* znovu optimalizován s využitím uložených exekučních statistik kardinalit.

Performance Feedback

Princip této funkce je obdobný jako v případě *statistics feedback*. *Performance feedback* ovšem slouží ke korekci použitého stupně paralelismu. Pokud je parametr PARALLEL_DEGREE_POLICY konfigurován na hodnotu ADAPTIVE, je tento typ reoptimalizace používán, přičemž v rámci něho jsou nebo mohou být prováděny následující kroky:

1. Během prvního zpracování daného příkazu, pro nějž je vhodné použít paralelismus, je na základě odhadu určen stupeň tohoto typu provedení. Dále jsou během samotného provádění monitorovány výkonové charakteristiky, a to i pro příkazy, pro které nebyl zvolen paralelní přístup.

2. Poté je na základě reálných zaznamenaných statistik určen optimální stupeň paralelismu, který je porovnáván s použitou hodnotou určenou na základě odhadu. Pokud tato kontrola určí významnou odlišnost, jsou opět příslušné statistiky uloženy pro budoucí použití a dotaz je označen jako adept na reoptimalizaci.

3. Při opětovném spuštění daného dotazu jsou uchovávané údaje použity pro stanovení optimálního stupně paralelismu.

Direktivy SQL plánu (SQL plan directives)

Tyto direktivy představují další užitečnou informaci pro optimalizátor, která se v principu nevztahuje k jednotlivým objektům, ani k celým konkrétním příkazům, ale přísluší k jejich výrazům, jako je například predikát obsahující konkrétní sloupce nebo klausule JOIN definující operaci spojení daných tabulek. Totožné direktivy tedy mohou být využívány k podpoře optimalizace rozdílných dotazů, které obsahují danou část, pro jejíž zpracování může direktiva doporučovat vhodné kroky. V době psaní tohoto textu byla dle dokumentu [22] implementována pouze direktiva typu DYNAMIC_SAMPLING, která instruuje optimalizátor k provedení sběru dynamických statistik.

Direktivy nemohou být vytvářeny manuálně. Vznikají tedy pouze automaticky v rámci činnosti optimalizátoru, který je ukládá do tabulkového prostoru SYSAUX. Zobrazit je lze dotazy na pohledy DBA_SQL_PLAN_DIRECTIVES a DBA_SQL_PLAN_DIR_OBJECTS. K další manuální správě je možné využít rutiny balíčku DBMS_SPD.

8 Popis vytvořené knihovny

8.1 Zadání

Předmětem praktické části práce bylo vytvoření knihovny v jazyku c#, která bude poskytovat následující funkcionality:

- Rozparsování zadaného příkazu SELECT do potřebné podkladové datové struktury.
- Výpis informací o tabulkách a podmínkách, které se nacházejí v daném dotazu. Tyto údaje budou načteny z datového slovníku databáze.
- Implementace mechanismů, které, pokud pro podmínky uvedené v daném dotazu a související tabulky v něm obsažené shledají indexový přístup efektivním, doporučí vytvoření/ využití tohoto objektu.

Dále bylo nutné vytvořit jednoduchou formulářovou GUI aplikaci, prostřednictvím které bude možné knihovnu otestovat.

8.2 Gramatika a BNF pravidla

Pro práci parseru je třeba definovat určitý vzor, oproti kterému budou dané algoritmy porovnávat zpracovávaný řetězec, respektive ve kterém budou vyhledávat tomuto řetězci odpovídající podobu. Kompletnímu popisu syntaxe jazyka se říká gramatika. Ta zpravidla představuje množinu pravidel, jenž jednotlivě definují syntaxi určité podoblasti celku, k čemuž používají dva druhy elementů, ze kterých jsou především tvořeny. Jedná se o symboly:

- terminální,
- neterminální.

Terminální symboly

Terminál je symbol, který se vyskytuje přímo v daném jazyce. Může to být například klíčové slovo, literál či operátor.

Neterminální symboly

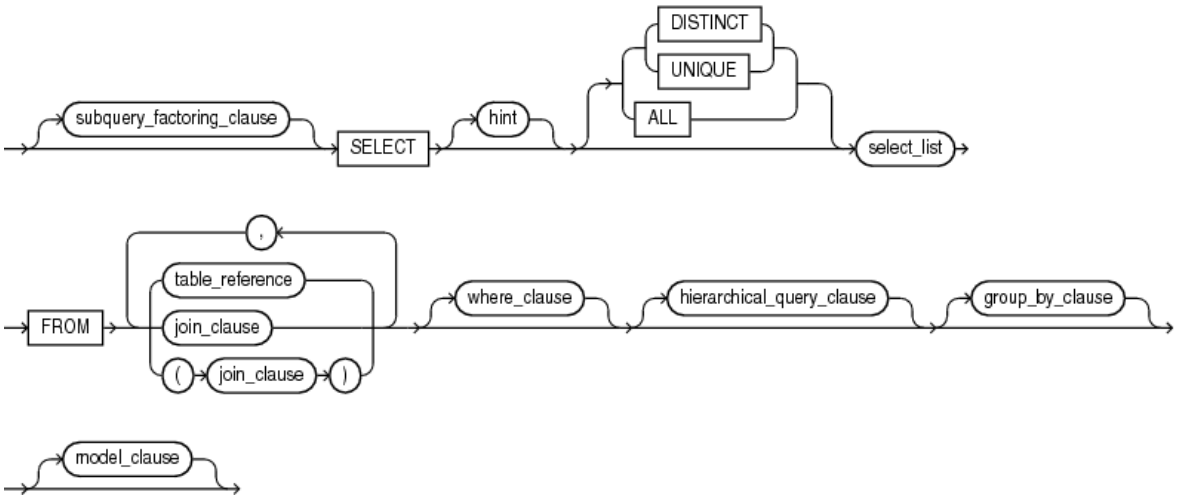
Neterminál se dá označit jako odkaz na další pravidlo, kterým je třeba tento identifikátor nahradit. Tomuto procesu se říká derivace a jeho opakováním je získán vzor, který obsahuje pouze terminální symboly.

Oficiální dokumentace systému Oracle (například [21]) využívá především dva přístupy pro vyjádření pravidel syntaxe svých příkazů. Kromě grafického znázornění prostřednictvím conwayových diagramů poskytuje také definice vyjádřené textovými řetězci. Uvedené podobě se říká Backusova-Naurova forma (BNF) a pro možnost jejího algoritmického zpracování byl tento formát zvolen pro realizaci požadované knihovny.

Například pravidlo pro významný neterminál *query_block* má v BNF podobu (terminální symboly jsou zvýrazněny tučným písmem):

```
query_block ::=
  [ subquery_factoring_clause ]
SELECT [ hint ] [ { { DISTINCT | UNIQUE } | ALL } ] select_list
  FROM { ( join_clause ) | join_clause | table_reference }
         [ , { ( join_clause ) | join_clause | table_reference } ] ...
  [ where_clause ]
  [ hierarchical_query_clause ]
  [ group_by_clause ]
  [ model_clause ]
```

A jeho conwayův diagram je následující:



Obrázek 9 - Conwayův diagram pro *query_block*, převzato z [21]

Jak je z příkladu patrné, BNF pravidla také obsahují řídící metasymboly, které umožňují přehlednějším způsobem popisovat složitější a rozsáhlejší konstrukce. Dokumentace Oracle konkrétně používá operátory uvedené v následující tabulce.

Tabulka 6 - Metasymboly v BNF pravidlech

Metasymbol	Popis
|	Alternativa. Tento znak představuje operátor OR
[…]	Volitelný blok
{…}	Povinný blok, slouží především k seskupování syntaktických jednotek
…	Tento symbol se používá v kombinaci s volitelným nebo povinným blokem, přičemž značí, že se blok může vícekrát opakovat

Pro specifikaci pravidel neterminálů na nejnižší úrovni, tedy například identifikátorů, řetězců nebo čísel, byly použity regulární výrazy. Více dále v oddíle 8.3.2.

8.3 Moduly

8.3.1 BNFScanner

Parser je postaven na algoritmech, které zpracovávají vstupní dotaz dle zadaných pravidel v BNF, jenž definují gramatiku příkazu SELECT v systému Oracle 11gR2. Tyto předpisy jsou ve formě textových řetězců uloženy v externím textovém souboru *oracle_grammar.txt*. Pro větší soudržnost knihovny by bylo vhodné je v budoucnu přesunout přímo do jejího zdrojového kódu v jazyku c#, ovšem tato pravidla byla v závěru tvorby práce stále předmětem testování, a tak byla pro větší přehlednost a snadnější modifikaci ponechána v samostatném externím umístění, odkud je do prostředí parseru načítá právě komponent *BNFScanner*. Tento modul navíc dané pravidlo transformuje z podoby textového řetězce na posloupnost elementů, které jsou označeny jako BNF tokeny[3]. Všechny tyto vytvořené objekty jsou instancemi třídy *BNFToken*, která obsahuje vlastnost *Type* výčtového typu *BNFTokenType*. Dle hodnoty tohoto atributu jsou v daných pravidlech rozlišeny následující kategorie tokenů (tedy znaků či jejich sekvencí):

- *TERMINAL_KEY_WORD* - klíčové slovo psané velkým písmem.
- *TERMINAL_SYMBOL* - terminální symbol, například operátor /.
- *NONTERMINAL* - identifikátor neterminálu psaný malým písmem.
- *OPTIONAL_BLOCK_START* - začátek volitelného bloku, znak [.
- *OPTIONAL_BLOCK_END* - konec volitelného bloku, znak].
- *REQUIRED_BLOCK_START* - začátek povinného bloku, znak {.
- *REQUIRED_BLOCK_END* - konec povinného bloku, znak }.
- *ALTERNATIVE* - znak alternativy |.
- *REPEAT_ELEMENT* - metasymbol opakování předchozího elementu … .

Pravidla v této vzniklé podobě jsou uložena v předdefinované generické struktuře *Dictionary*, která se v naplněném stavu dá označit za hlavní výstup činnosti modulu *BNFScanner*. Všechny kolekce, jenž jsou v rámci jeho aktivity sestavovány, jsou uvedeny v následující tabulce.

Tabulka 7 - Kolekce naplňované během činnosti modulu *BNFScanner*

Název	Typ	Popis
rulesTokens	Dictionary<string, BNFToken[]>	Slovník, jenž každému klíči, který je názvem neterminálu, přiřazuje pole BNF tokenů představující dané pravidlo
terminalsRegexs	Dictionary<string, Regex>	Některá pravidla na nejnižší úrovni (například identifikátory nebo řetězce) nejsou definována pomocí pole BNF tokenů, ale prostřednictvím regulárního výrazu
keyWords	SortedSet<string>	Množina klíčových slov ze všech pravidel, přičemž každé unikátní slovo je v ní obsaženo právě jednou
operators	Dictionary<string, BNFOperatorType>	Kolekce, jenž každému klíči, který je operátorem (symbolem), přiřazuje jeho typ z hlediska výskytu bílých znaků mezi ním a okolními operandy

[3] V teorii kompilátorů představuje token prvek jazyka s definovaným významem. Jedná se tedy o množinu znaků, které společně vytvářejí například klíčové slovo, operátor či identifikátor. S tokenem souvisí také pojem lexéma, jehož význam je v podstatě totožný a v této práci je používán termín token.

8.3.2 QueryScanner

Koncepce modulu *QueryScanner* spočívá v přijmutí dotazu ve formě vstupního řetězce, který transformuje do podoby posloupnosti tokenů (provádí obdobnou činnost jako *BNFScanner*, který ovšem pracuje s BNF pravidly a ne s dotazem). Tento úkol konkrétně vykonává funkce:

```
List<QueryToken> scanQuery(String query)
```

jenž v hojné míře využívá pro získávání jednotlivých tokenů regulární výrazy. Výstupem scanneru je tedy generická kolekce *List<QueryToken>*. Implementovaný algoritmus dokáže rozlišit typy tokenů uvedené v tabulce níže (Tabulka 8).

Tabulka 8 - Kategorie tokenů

Typ	Vzor	Popis
KEY_WORD	^(\p{Ll}|\p{Lu}|\p{Lt}|\p{Lo}|\p{Lm})\w* Kromě uvedeného regulárního výrazu také výčet klíčových slov v kolekci *keyWords*	Klíčové slovo
UNBOUNDED_IDENTIFIER	^(\p{Ll}|\p{Lu}|\p{Lt}|\p{Lo}|\p{Lm})(\w|\$)*	Neohraničený identifikátor
BOUNDED_IDENTIFIER	^"[^"\n]+"	Identifikátor ohraničený uvozovkami "
CHAR_SEQUENCE	^'([^']|\')*'	Sekvence znaků ohraničená apostrofy '
POSITIVE_NUMBER	^\+?(\.\d+|\d+\.\d*)((e|E)?(\+|-)?\d+)?(f|F|d|D)?	Kladné číslo
NEGATIVE_NUMBER	^-(\.\d+|\d+\.\d*)((e|E)?(\+|-)?\d+)?(f|F|d|D)?	Záporné číslo
POSITIVE_INTEGER	^\+?\d+((e|E)?(\+|-)?\d+)?(f|F|d|D)?	Kladné celé číslo
NEGATIVE_INTEGER	^-\d+((e|E)?(\+|-)?\d+)?(f|F|d|D)?	Záporné celé číslo
SYMBOL	Výčet možných operátorů (symbolů) v kolekci *operators*	Operátor (symbol)
AMBIGUOUS	-	Nejednoznačný typ tokenu, který by mohl být například zároveň operátorem nebo identifikátorem.
INCORRECT_TOKEN	-	Nekorektní token, který například obsahuje nepřijatelné znaky nebo není zakončen požadovanou uvozovkou

V případě výskytu chyby vyvolá modul výjimku typu *ScanningException*.

8.3.3 Parser

Parser je nejkomplikovanějším komponentem knihovny, který své služby poskytuje prostřednictvím následující rutiny:

```
ParserTree parseQuery(QueryToken[] queryTokensFromScanner, string
nonTerminalId = "select")
```

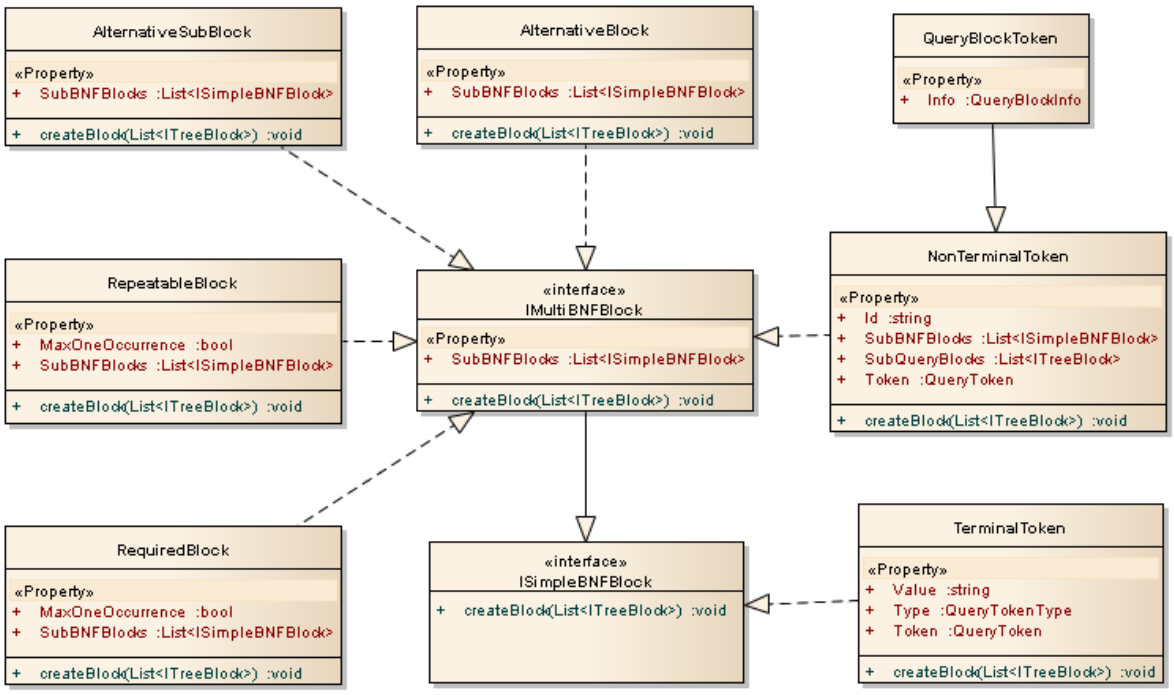
Tato funkce přijímá pole tokenů, na které je transformována výstupní kolekce modulu *QueryScanner*, a pro dotaz v této podobě vyhledává odpovídající protipól ve vzorových syntaktických pravidlech. Pokud je hledání neúspěšné, syntaxe dotazu je shledána nekorektní a parser vyvrhne výjimku typu *ParsingException*. Pokud naopak dojde k nalezení vzoru, porovnání je úspěšné a metoda vrací hierarchickou strukturu typu *ParserTree*, která je podrobněji popsána v podtitulu dále. Funkce *parseQuery* obsahuje i volitelný parametr *nonTerminalId*, který umožňuje stanovit výchozí neterminál, oproti kterému bude porovnávání probíhat. Jeho defaultní hodnota je *"select"*, neboť knihovna je určena výhradně k parsování kompletních příkazů SELECT a další text bude popisovat právě tento případ.

Algoritmus

Nejprve je vytvořena instance třídy *NonTerminalToken* představující výchozí neterminál s id *select*, pro kterou je následně zavolána její následující metoda:

```
void createBlock(List<ITreeBlock> blockList)
```

Parametr *blockList* v této situaci představuje kořen výstupní struktury *ParserTree*. Uvedená funkce je stěžejním segmentem parseru. Její signatura je obsažena v základním rozhraní *ISimpleBNFBlock* od kterého jsou odvozeny další rozhraní a třídy, jenž dle vlastní potřeby definují tělo metody. Tuto hierarchii zobrazuje obrázek 10 níže.



Obrázek 10 - Diagram základních tříd a rozhraní modulu *Parser*

Jak je uvedeno výše, mechanismy uvnitř funkce *createBlock* jsou pro každý typ odlišné. Jejich popis je společně s přehledem jednotlivých rozhraní a tříd uveden v následujících tabulkách (Tabulka 9 a Tabulka 10).

Tabulka 9 - Základní rozhraní modulu Parser

Název rozhraní	Popis
ISimpleBNFBlock	Základní rozhraní, ze kterého vycházejí ostatní rozhraní a třídy. Obsahuje signaturu metody *createBlock*, kterou tak musí implementovat všechny odvozené typy
IMultiBNFBlock	Tento potomek rozhraní *ISimpleBNFBlock* svého rodiče rozšiřuje o list prvků typu *ISimpleBNFBlock*. Zavadí tedy koncept hierarchie, kdy jeden prvek typu *IMultiBNFBlock* může obsahovat kolekci dalších bloků

Tabulka 10 - Základní třídy modulu Parser

Název třídy	Popis třídy	Chování metody createBlock
NonTerminalToken, QueryBlockToken	Základní třída pro neterminály, které mohou být definovány pomocí regulárního výrazu nebo pole prvků typu *BNFToken* (BNF pravidlo). Třída *QueryBlockToken* představuje konkrétní neterminál *query_block*. Je potomkem třídy *NonTerminalToken*, kterou rozšiřuje pouze o vlastnost *Info*, jenž uchovává sadu informací o daném elementu *query_block* používanou modulem *SelectInfo* (viz dále v pododdílu 8.3.4 SelectInfo)	Pokud je neterminál definovaný pravidlem, tedy polem prvků typu *BNFToken*, metoda naplní s využitím těchto prvků kolekci *SubBNFBlocks*. Dané pravidlo je tak transformováno do své další podoby, kterou je tato datová struktura, jenž je poté procházena a pro každý její element je opět volána funkce *createBlock*. Pokud analyzovaná sekvence tokenů odpovídá vzoru, jsou do listu *SubQueryBlocks* přidávány odpovídající prvky, čímž je tvořen výsledný analytický strom *ParserTree*
TerminalToken	Třída reprezentující terminální symboly.	Metoda porovnává hodnotu vlastnosti *Value* s hodnotou aktuálního tokenu *QueryToken*. Pokud je shledá odlišnými, vyvrhne výjimku typu *ParsingException*
RepeatableBlock	Pokud je v daném pravidle obsažen nepovinný blok uzavřený do hranatých závorek [], je později v kolekci *SubBNFBlocks* reprezentován jako objekt typu *RepeatableBlock*	Je procházena kolekce *SubBNFBlocks* a pro její prvky je opět volána jejich metoda *createBlock*. Blok je nepovinný. Pokud je uvnitř těchto rekurzivně volaných funkcí vyvolána výjimka, je odchycena a proces parsování pokračuje dále.
RequiredBlock	Obdobný význam jako u předchozí třídy *RepeatableBlock*. Ovšem v tomto případě se jedná o povinný blok ohraničený složenými závorkami {}	Obdobný princip jako u *RepeatableBlock* ale pokud je při neshodě vnitřních terminálů vyvolána chyba, není v rámci této metody odchycena a je vyvrhnuta vně
AlternativeBlock, AlternativeSubBlock	Pokud pravidlo obsahuje znak alternativy |, je tato konstrukce převedena na objekt typu *AlternativeBlock*, jenž obsahuje jednotlivé možné alternativy v listu *SubBNFBlocks* jako objekty typu *AlternativeSubBlock*	Metoda prochází kolekci *SubBNFBlocks* a pro každý její prvek volá funkci *createBlock* dokud nenarazí na shodu. Pokud nenalezne žádnou možnou alternativu, je vyvrhnuta výjimka

Důležitou roli v algoritmech parseru hrají mechanismy vyvolávání a odchytávání výjimek, na kterých je postaven celý koncept předávání chyb vzniklých pří neshodě analyzovaného dotazu se vzorovými syntaktickými pravidly. Pokud například terminál typu

TerminalToken provádí svou funkci *createBlock* a zjistí neshodu, vyvrhne tato metoda výjimku. Ovšem v případě, kdy je tato neúspěšná kontrola na shodu terminálu prováděna v rámci nepovinného bloku typu *RepeatableBlock*, je uvnitř jeho metody *createBlock* výjimka odchycena a proces parsování tak dále pokračuje. Z prvně volané funkce *createBlock* pro základní neterminál *select* a potažmo i z metody *parseQuery* je výjimka vyvržena jen v případech, kdy je jisté, že syntaxe vstupního dotazu neodpovídá požadovanému vzoru.

ParserTree

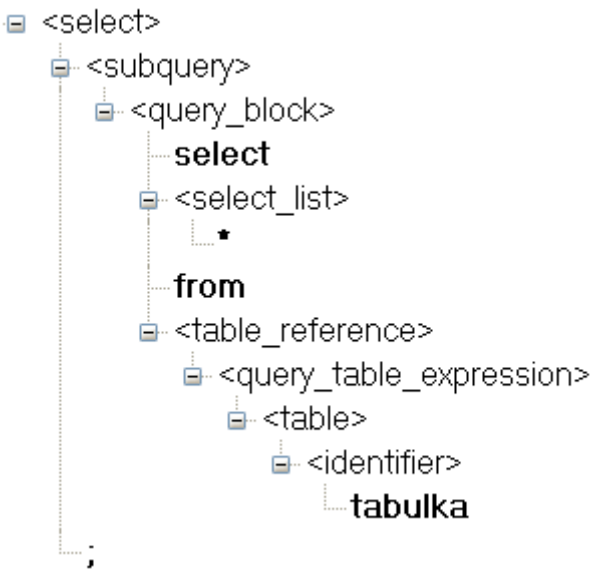
Výstupní struktura implementovaného parseru má hierarchickou podobu stromu, který se skládá pouze z:

- Neterminálních symbolů, které představují uzly.
- Terminálních symbolů, jenž představují koncové listy.

Kořenem je vždy neterminál *select*, který obsahuje list symbolů, z nichž je složen. Mezi nimi jsou mimo terminálů obsaženy také další neterminály se svými kolekcemi elementů, prostřednictvím kterých lze strukturou postupovat až ke všem koncovým listovým terminálům. Například pro jednoduchý příkaz:

```
select * from tabulka;
```

Sestaví parser následující strom (terminály jsou zvýrazněny tučným písmem):



Obrázek 11 - Struktura *ParserTree* pro jednoduchý příkaz SELECT

8.3.4 SelectInfo

V předchozím textu popsané moduly knihovny souvisejí především s teorií kompilátorů a parseru. Komponent *SelectInfo* má ze všech elementů knihovny jednoznačně nejblíže k problematice optimalizace dotazů, neboť zahrnuje optimalizační mechanismy pro zhodnocení přínosu indexů. Z tohoto důvodu mu je dále věnován samostatný oddíl.

8.4 Modul SelectInfo

Stejnojmenná třída tohoto modulu poskytuje metodu *syntaxChechAndGetInfo*, jejíž signatura je následující:

```
void syntaxChechAndGetInfo(ParserTree parserTree)
```

Vstupem činnosti modulu je tedy dotaz ve formě struktury typu *ParserTree*, jenž byla sestavena během procesu parsování. Tento komponent je jediným, který pracuje s databází. Pro připojení využívá technologii *Oracle Data Provider for .NET* (ODP.NET) a konkrétně třídy a rutiny ze jmenného prostoru *Oracle.DataAccess.Client*. Vnitřní mechanismy modulu provádějí tři základní činnosti:

- Kontrola sémantiky;
- Zjištění informací o tabulkách dotazu;
- Zvážení vhodnosti aplikace indexového přístupu.

Předmětem těchto procesů je vždy daný *query_block*, jenž představuje zásadní část konstrukce příkazu SELECT. Pokud dotaz obsahuje například klausuli *subquery_factoring_clause* nebo vnořené poddotazy a je tedy tvořen z více těchto bloků, jsou uvedené mechanismy aplikovány na každý jednotlivý *query_block*, který je ve vstupní struktuře *parserTree* reprezentován jako objekt třídy *QueryBlockToken*. Tato třída obsahuje vlastnost *Info* typu *QueryBlockInfo*, jenž představuje kontejner pro potřebné a požadované informace získané během výše uvedených aktivit modulu *SelectInfo*. Tyto kroky jsou dále popsány podrobněji.

8.4.1 Kontrola sémantiky

Během této částečné sémantické kontroly jsou provedeny následující úkony:

- Ověření existence tabulek či pohledů.
- Ověření existence odkazovaných sloupců v dotazu (i v případě jejich výskytu v podmínkách a skalárních poddotazech či výběru dat z vnořených poddotazů).
- Ověření jedinečné identifikace objektů (tabulek, pohledů, sloupců či pojmenovaných poddotazů).
- Přiřazení obsažených sloupců ke konkrétním zdrojům (tabulkám, pohledům a poddotazům).

8.4.2 Zjištění informací o tabulkách dotazu a podmínkách

S využitím datového slovníku systému Oracle jsou získány následující informace:

- Typ objektu (tabulka, pohled, materializovaný pohled).
- Všechny sloupce objektu.
- Indexy objektu.
- Počet řádků objektu.
- Seznam podmínek a informací k nim vztažených, jako například typ podmínky či počet sloupců v ní obsažených.

8.4.3 Zvážení vhodnosti aplikace indexového přístupu

Indexový přístup je spojen s výběrem určité části záznamů, k němuž vždy dochází v důsledku použití podmínky. Z tohoto důvodu se stanovená doporučení vztahují k jednotlivým podmínkám, které jsou rozlišeny do následujících kategorií:

- *ONE_OBJECT_ONE_COLUMN* - podmínka aplikovaná na jednu množinu záznamů, v níž se vyskytuje pouze jeden sloupec z tohoto zdroje.
- *ONE_OBJECT_MULTIPLE_COLUMNS* - podmínka aplikovaná na jednu množinu záznamů, v níž se vyskytuje více sloupců z tohoto zdroje.
- *JOIN_ONE_COLUMN* - podmínka spojující více množin záznamů, ve které se vyskytuje jeden sloupec z každého zdroje.
- *JOIN_MULTIPLE_COLUMNS* - podmínka spojující více množin záznamů, ve které se vyskytuje více sloupců z některého zdroje.

Pro odhad přínosnosti indexu je třeba shromáždit vybrané charakteristiky daných zdrojů a jejich sloupců. Zdrojem nemusí být pouze databázový objekt jako tabulka nebo pohled, ale knihovna získává potřebné statistiky i pro výsledek vnořeného poddotazu, se kterým se v rámci vnějšího dotazu dále pracuje jako s tabulkou. Popisovanými údaji jsou:

- Celkový počet řádků zdroje;
- Počet jedinečných řádků pro dané sloupce v podmínce;
- Celkový počet řádků výsledku po aplikaci podmínky;
- Počet jedinečných řádků výsledku pro dané sloupce v podmínce po její aplikaci.

Na základě těchto uvedených statistik doporučuje knihovna použití indexů pro sloupce tabulek dle dvou následujících kritérií, jenž obě využívají koeficient poměru defaultně nastaven na hodnotu 0,2:

- *Poměr celkového počtu řádků zdroje k celkovému počtu řádků výsledku* - nejprve je ověřena selektivita podmínky. Pokud je výsledný počet řádků menší než jedna pětina původní kardinality tabulky, považuje mechanismus použití indexu za vhodné. Když je tomu naopak a předmětem je podmínka spojení, pokračuje proces kontrolou poměru počtů jedinečných řádků.
- *Poměr počtu jedinečných řádků pro sloupce zdroje k počtu jedinečných řádků pro dané sloupce výsledku* - toto kritérium je užitečné pro určování selektivity podmínek spojení, kterou výše popsaná základní kontrola nemusí určit správně. Je aplikováno pouze, pokud je podmínka jednoho ze dvou typů vztahujícího se ke spojení, tedy verze JOIN_ONE_COLUMN nebo JOIN_MULTIPLE_COLUMNS. Jak popisuje příklad dále, výsledek této operace může být objemný, i když se v něm vyskytuje jen malý podíl řádků z jedné připojované tabulky.

Knihovna také doporučuje použití bitmapového indexu, pokud je počet jedinečných hodnot sloupce menší než stanovená prahová hodnota (defaultně 4) a celková kardinalita tabulky naopak převyšuje určenou hranici (defaultně 30), je pro daný sloupec doporučeno využití bitmapového indexu a výše uvedené kontrolní mechanismy již pro něj neprobíhají.

8.4.4 Příklad

Pokud existují tabulky:

- *hr.employees* - obsahuje 107 řádků a pro sloupec *department_id* existuje 12 jedinečných variant (včetně NULL), ovšem největší podíl má hodnota 50, která se vyskytuje ve 45 řádcích, tedy téměř v polovině záznamů.
- *hr.departments* - obsahuje 27 řádků, přičemž sloupec department_id je primárním klíčem a pro jeho hodnotu tedy existuje 27 různých variant.

Nad kterými je vykonán následující příkaz:

```
select * from
(select * from hr.employees where department_id = 50)
join hr.departments using(department_id);
```

Dotaz obsahuje dvě podmínky popsané dále v příslušných podtitulech a jejich tabulkách (Tabulky 11-15):

- *department_id = 50*;
- *using(department_id)*.

Podmínka A (department_id = 50)

Tabulka 11 - Charakteristiky podmínky A

Atribut	Hodnota
Podmínka	HR.EMPLOYEES.DEPARTMENT_ID = 50
Typ podmínky	ONE_OBJECT_ONE_COLUMN
Kardinalita výsledku	45
Počet sloupců v podmínce	1

Tabulka 12 - Statistiky sloupce *department_id* pro podmínku A

Atribut	Hodnota
Schéma	HR
Objekt	EMPLOYEES
Sloupec	DEPARTMENT_ID
Typ zdroje	TABLE
Počet existujících indexů	1
Kardinalita zdroje	107
Kardinalita výsledku	45
Procentuální kardinalita výsledku	42,06 %
Počet jedinečných hodnot sloupců zdroje	12
Počet jedinečných hodnot sloupců výsledku	1
Procentuální počet jedinečných hodnot sloupců výsledku	8,33 %
Doporučení použít index	**Ne**

Z údajů v tabulkách vyplývá, že pro tuto jednoduchou podmínku knihovna index nad sloupcem *department_id* nedoporučuje. Filtrem projde cca 42% řádků, což představuje

relativně nízkou selektivitu a není tedy splněno první kritérium. Druhé kritérium není bráno v úvahu, neboť se nejedná o podmínku spojení.

Podmínka B (using(department_id))

Tabulka 13 - Charakteristiky podmínky B

Atribut	Hodnota
Podmínka	using(DEPARTMENT_ID)
Typ podmínky	JOIN_ONE_COLUMN
Kardinalita výsledku	45
Počet sloupců v podmínce	2

Tabulka 14 - Statistiky sloupce *department_id* z vnořeného poddotazu pro podmínku B

Atribut	Hodnota
Schéma	-
Objekt	-
Sloupec	DEPARTMENT_ID
Typ zdroje	SUBQUERY
Počet existujících indexů	-
Kardinalita zdroje	45
Kardinalita výsledku	45
Procentuální kardinalita výsledku	100 %
Počet jedinečných hodnot sloupců zdroje	1
Počet jedinečných hodnot sloupců výsledku	1
Procentuální počet jedinečných hodnot sloupců výsledku	100 %
Doporučení použít index	-

Tabulka 15 - Statistiky sloupce *department_id* z tabulky *departments* pro podmínku B

Atribut	Hodnota
Schéma	HR
Objekt	DEPARTMENTS
Sloupec	DEPARTMENT_ID
Typ zdroje	TABLE
Počet existujících indexů	1
Kardinalita zdroje	27
Kardinalita výsledku	45
Procentuální kardinalita výsledku	166,67 %
Počet jedinečných hodnot sloupců zdroje	27
Počet jedinečných hodnot sloupců výsledku	1
Procentuální počet jedinečných hodnot sloupců výsledku	3,7 %
Doporučení použít index	**ANO**

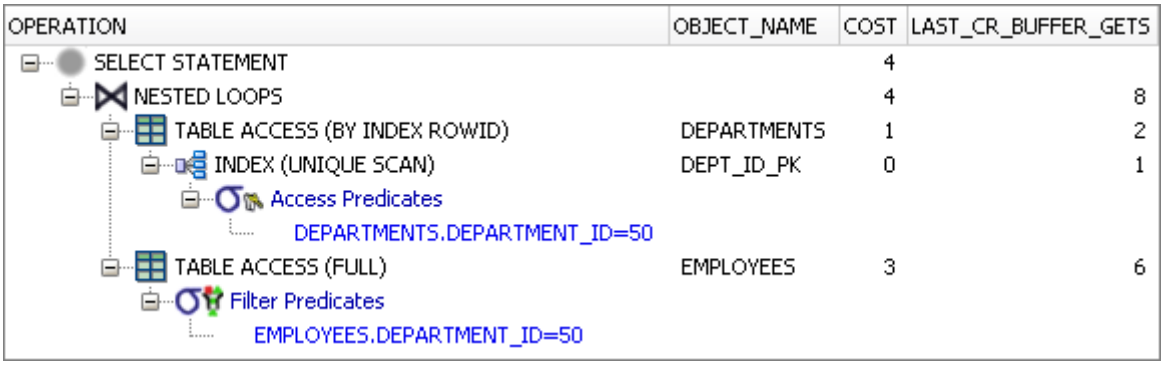
Druhá podmínka spojuje výsledek vnořeného poddotazu s tabulkou *departments*. V tomto případě knihovna považuje za vhodné přistupovat k této tabulce přes index. První kritérium sice splněno není - naopak kardinalita výsledku spojení je téměř dvakrát větší než počet řádků v tabulce *departments*. Ovšem jelikož se jedná o podmínku spojení, je uplatněno i druhé kritérium počtů jedinečných řádků. Tím je odhalen fakt, že pro výsledek operace je potřeba pouze jediná hodnota z 27 záznamů a bude tedy zřejmě vhodné uplatnit indexový

přístup ke sloupci *department_id* potažmo k tabulce *departments*. Toto rozhodnutí by však bylo mylné, pokud by sloupec *department_id* nebyl v tabulce *departments* primárním klíčem a obsahoval by velký podíl řádků s hodnotou 50. Chování algoritmu by tedy mohlo být předmětem dalších inovací, které by do jeho rozhodování zahrnuly například právě fakt, zda sloupec je unikátním klíčem či nikoliv.

Optimalizátor Oracle samozřejmě pracuje na daleko vyšší úrovni, kdy například dokáže rozpoznat, že vnořený poddotaz na tabulku *employees* lze vynechat a dotaz transformovat do podoby spojení dvou tabulek:

```
select * from hr.employees
join hr.departments using(department_id)
where department_id = 50;
```

pro kterou určí exekuční plán zobrazený na následujícím obrázku (Obrázek 12):



Obrázek 12 - Exekuční plán testovacího dotazu

Ovšem je patrné, že volba použitých přístupových metod se v tomto případě neliší od výstupů mechanismů knihovny. Pro tabulku *departments* je indexový přístup využit a na tabulku *employees* je naopak aplikován úplný průchod.

Závěr

Systém Oracle disponuje stále inteligentnějšími mechanismy, které dokážou eliminovat problémy s výkonem způsobené nevhodně formulovanými příkazy, respektive pojem nevhodně formulovaný příkaz z hlediska optimalizace již téměř ztrácí význam, neboť jakákoliv výchozí textová podoba dotazu je jen dočasná a dá se říci, že představuje pouze jednu ze vstupních informací pro optimalizátor, který s jejím využitím sám stanoví efektivní postup. Z těchto důvodů se optimalizace na úrovni SQL dotazu stává stále banálnější.

Ovšem vzhledem k různorodosti provozovaných aplikací, jenž jsou postaveny v podstatě na jedné verzi databáze Oracle, je nutné tento univerzální systém přizpůsobit dle konkrétní povahy a požadavků. Přestože i v tomto směru prochází produkt Oracle vývojem, který zvyšuje jeho schopnost automatického řízení své činnosti, stále je třeba dbát na optimalizaci, ovšem především ve formě vhodné architektury, konfigurace a průběžné správy systému. Efektivnějších výsledků lze přirozeně dosáhnout také uvědomělým přístupem při tvorbě samotné aplikace, která databázi využívá. Ignorování možností a zásad, které mohou vést k zlepšenému výkonu, by kvalitu provozu aplikace v některých případech značně, vždy ale zbytečně znehodnotilo.

V rámci praktické části byla dle zadání vytvořena funkční aplikace využívající požadovanou knihovnu. Tento software pro testovací příklady pracuje poměrně spolehlivě. Ovšem spektrum možných variant dotazu SELECT a klauzulí, které obsahuje, je v podstatě neohraničené. Pro vylepšení funkčnosti by tedy do budoucna bylo vhodné investovat delší čas do testování programu. Algoritmus parseru je dle dosavadního provozu plně funkční, rezervy knihovny jsou očekávány v některých syntaktických pravidlech, která parser pro svou činnost využívá. Po plném prověření a následné vhodné modifikaci používané syntaxe by bylo možné považovat vyvinutý parser za pevný základ pro, kromě již implementovaných modulů, případné další softwarové nadstavby.

Vylepšení by se přirozeně mohlo dostat i samotné GUI aplikaci a ostatním komponentům. Kvalita optimalizátoru by mohla být zvýšena vyvinutím sofistikovanějších algoritmů pro doporučení tvorby indexů, nebo mechanismů, jenž by tyto objekty automaticky vytvořily. Dále například implementací testování kritérií, dle nichž by knihovna dokázala odvodit vhodnost vytvoření materializovaných pohledů, použití dělení tabulek, clusterů, vázaných proměnných, atd.

Literatura

[1] **Oracle Corporation.** *Oracle Database Performance Tuning Guide 11g Release 2 (11.2) - Performance Tuning Overview* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/perf_overview.htm>.

[2] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Managing Optimizer Statistics* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/stats.htm>.

[3] **Oracle Corporation.** *Oracle® Database PL/SQL Packages and Types Reference 11g Release 2 (11.2) - DBMS_STATS* [online]. 2011 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/appdev.112/e25788/d_stats.htm>.

[4] **Oracle Corporation.** *Oracle® Database Concepts 11g Release 2 (11.2) - SQL* [online]. 2011 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e25789/sqllangu.htm>.

[5] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Using EXPLAIN PLAN* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/ex_plan.htm>.

[6] **Oracle Corporation.** *SQL*Plus® User's Guide and Reference Release 11.2 - Tuning SQL*Plus* [online]. 2010 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16604/ch_eight.htm>.

[7] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Using SQL Plan Management* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/optplanmgmt.htm>.

[8] **Oracle Corporation.** *Oracle® Database Concepts 11g Release 2 (11.2) - Memory Architecture* [online]. 2011 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e25789/memory.htm>.

[9] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - The Query Optimizer* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/optimops.htm>.

[10] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Designing and Developing for Performance* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/design.htm>.

[11] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Using Indexes and Clusters* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/data_acc.htm>.

[12] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Configuring and Using Memory* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/memory.htm>.

[13] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - SQL Tuning Overview* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/sql_overview.htm>.

[14] **Oracle Corporation.** *Oracle® Database SQL Language Reference 11g Release 2 - Analytic Functions* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e26088/functions004.htm>.

[15] **KYTE, Thomas.** *Oracle - návrh a tvorba aplikací*. Vydání první. Brno : Computer Press, a.s., 2007. 694 s. ISBN 80-251-0569-5.

[16] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Using Optimizer Hints* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/hintsref.htm>.

[17] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Automatic Performance Statistics* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/autostat.htm>.

[18] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Automatic Performance Diagnostics* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/diag.htm>.

[19] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - Automatic SQL Tuning* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/sql_tune.htm>.

[20] **Oracle Corporation.** *Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) - SQL Access Advisor* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e16638/advisor.htm>.

[21] **Oracle Corporation.** *Oracle® Database SQL Language Reference - SELECT* [online]. 2012 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E11882_01/server.112/e26088/statements_10002.htm>.

[22] **COLGAN, Maria.** *Optimizer with Oracle Database 12c* [online]. 2013 [cit. 2013-07-31]. Dostupné z WWW: <http://www.oracle.com/ocom/groups/public/@otn/documents/webcontent/1963236.pdf>.

[23] **Oracle Corporation.** *Oracle® Database SQL Tuning Guide 12c Release 1 - Query Optimizer Concepts* [online]. 2013 [cit. 2013-07-31]. Dostupné z WWW: <http://docs.oracle.com/cd/E16655_01/server.121/e15858/tgsql_optcncpt.htm>.