

SVIG
CONSULTING

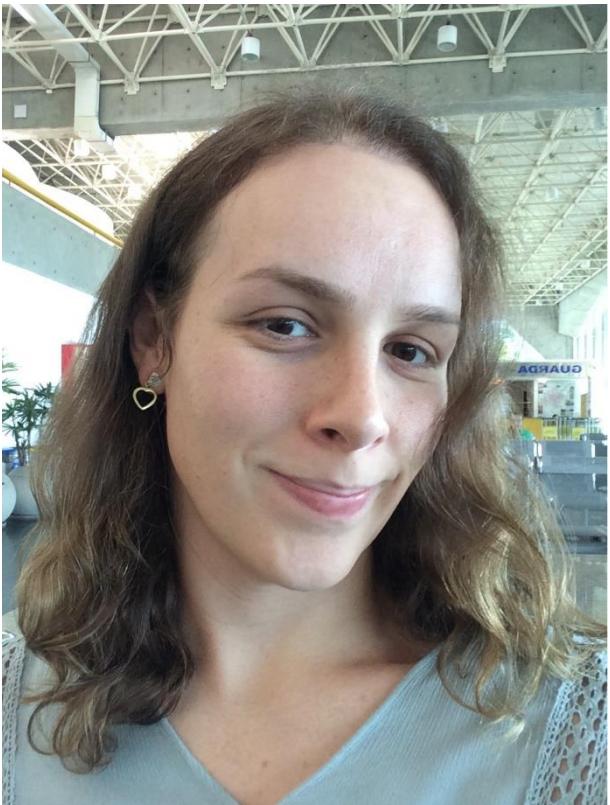
Performance Tuning com Oracle 12c

Aula 01: Introdução

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br



Quem sou eu?



Analista de Sistemas
Certificada **OCE** SQL Expert
Certificada **OCP** DBA 11g
Certificada **OCP** PL/SQL
Autora de Artigos na OTN
<https://www.linkedin.com/in/petruzalek>

ORACLE®
Certified Expert
Oracle Database SQL

ORACLE®
Certified Professional
Oracle Database 11g Administrator

ORACLE®
Certified Professional
Advanced PL/SQL Developer

Objetivos do Curso

- Arquitetura Oracle Database
- Metodologia para Performance Tuning
- Estatísticas
- Planos de Execução
- Ferramentas de Diagnóstico
- Performance PL/SQL
- Melhores Práticas
- Modelagem Física

Referências para o Curso

- Oracle Database Concepts
 - <https://docs.oracle.com/database/121/CNCPT/toc.htm>
- Oracle Database Performance Tuning Guide
 - <http://docs.oracle.com/database/121/TGDBA/toc.htm>

Referências para o Curso – Blogs

- AskTom [Tom Kyte]
 - <http://asktom.oracle.com>
- Oracle Developer [Adrian Billington]
 - <http://www.oracle-developer.net>
- Oracle Optimizer Blog [Maria Colgan]
 - <https://blogs.oracle.com/optimizer/>
- Oracle Scratchpad [Jonathan Lewis]
 - <https://jonathanlewis.wordpress.com/>
- The Tom Kyte Blog
 - <http://tkyte.blogspot.com.br/>

Cronograma

- Aula 01: Introdução
 - Introdução ao Curso
 - Preparação do Ambiente do Curso
 - Arquitetura do Ambiente do Curso
- Aula 02: Arquitetura I
 - Arquitetura Oracle
 - Arquitetura de Rede
 - Estruturas de Memória: SGA
 - Estruturas de Memória: PGA
- Aula 03: Arquitetura II
 - Arquitetura de Processos
 - Estruturas de Disco
 - Tipos de Leitura e Inserção
 - Transações e SCN
- Aula 04: Buffer Cache
 - Estatísticas de Sessão
 - Tipos de I/O
 - *Cache Hit Ratio*

Cronograma

- Aula 05: Metodologia de *Tuning*
 - Metodologia Oracle de *Tuning*
 - Padrões de Codificação
 - SQL Dinâmico
- Aula 06: SQL e Otimização
 - Etapas do Processamento SQL
 - Otimizador
 - Tipos de *Parse*
 - Otimização Baseada em Custo
- Aula 07: Planos de Execução
 - Captura de Planos
 - Interpretação
 - Ferramentas de Visualização
- Aula 08: Otimizador I
 - *Hints*
 - Caminhos de Acesso: Tabelas
 - Caminhos de Acesso: Índices

Cronograma

- Aula 09: Otimizador II
 - Caminhos de Acesso: *Cluster*
 - Operadores Binários
 - Operadores n-Ários
- Aula 10: Estatísticas I
 - Estatísticas de Tabelas
 - Estatísticas de Índices
 - Estatísticas de Colunas
 - Estatísticas Estendidas
- Aula 11: Estatísticas II
 - Estatísticas de Sistema
 - Gerenciamento de Estatísticas
 - Estatísticas de Tabelas Temporárias
- Aula 12: Cursores
 - *Cursor Sharing*
 - *Bind Peeking*
 - *Adaptive Cursor Sharing*

Cronograma

- Aula 13: *Trace* em Aplicações
 - Tipos de *Trace*
 - Ferramentas de Análise
 - Instrumentação de Aplicações
- Aula 14: PL/SQL I
 - Máquina Virtual PL/SQL
 - Compilação Nativa
 - *Optimize Level*
- Aula 15: PL/SQL II
 - Trocas de Contexto
 - BULK COLLECT
 - FORALL
- Aula 16: Técnicas de Cache
 - *Result Cache*
 - *Scalar Subquery Cache*
 - Funções Determinísticas
 - *User Defined Functions*

Cronograma

- Aula 17: Views Materializadas
 - Definição
 - *Materialized View Query Rewrite*
 - Condições para Reescrita
- Aula 18: Modelagem Física
 - Compactação de Índices
 - Compactação de Dados
 - Particionamento
- Aula 19: In-Memory Database
 - Formato Colunar x Linear
 - Configuração e Uso

O Ambiente de Testes

- Pré-Requisitos:
 - Oracle VM VirtualBox (versão atual: 5.1)
 - Oracle SQL Developer (versão atual: 4.1.5)
 - Processador 2GHz (i5 ou superior, 2 cores ou mais)
 - Mínimo 4 GB RAM. A VM usa por padrão 2G RAM (ideal 4 GB+ para a VM)
 - 15GB de espaço em disco
 - Oracle Client 12c [opcional]
- Componentes da VM:
 - Oracle Linux 7
 - Oracle Database 12c Release 1 Enterprise Edition (12.1.0.2 with In-Memory Option)
 - Oracle SQL Developer 4.1.3

Arquivos Necessários

Welcome to VirtualBox.org!

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. See "About VirtualBox" for an introduction.

Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris and OpenSolaris, OS/2, and OpenBSD.

VirtualBox is being actively developed with frequent releases and has an ever growing list of features, supported guest operating systems and platforms it runs on. VirtualBox is a community effort backed by a dedicated company: everyone is encouraged to contribute while Oracle ensures the product always meets professional quality criteria.

Download VirtualBox 5.1

Hot picks:

- Pre-built virtual machines for developers at [Oracle Tech Network](#)
- [Hyperbox](#) Open-source Virtual Infrastructure Manager [project site](#)
- [phpVirtualBox](#) AJAX web interface [project site](#)
- [IQEmu](#) automated Windows VM creation, application integration <http://mirage335-site.member.hacd.org:6380/wiki/Category:IQEmu>

Oracle VM VirtualBox

<https://www.virtualbox.org/>

Sign In/Register Help Country ▾ Communities ▾ I am a... ▾ I want to... ▾ Search Search

Products Solutions Downloads Store Support Training Partners About OTN

Oracle Technology Network > Database > Database 12c

Oracle Technology Network Developer Day

Database Virtual Box Appliance / Virtual Machine

*Set-Up Instructions: Database Application Development Hands On Labs
Updated: 6/22/2016*

Welcome to the Oracle Technology Network Developer Day - Hands-on Database Application Development lab installation instructions. This is a bring-your-own-laptop workshop; this document describes how to install a virtual guest appliance that provides pre-configured Oracle software for your use while working on the labs. (Note: this appliance is not for the .NET track.)

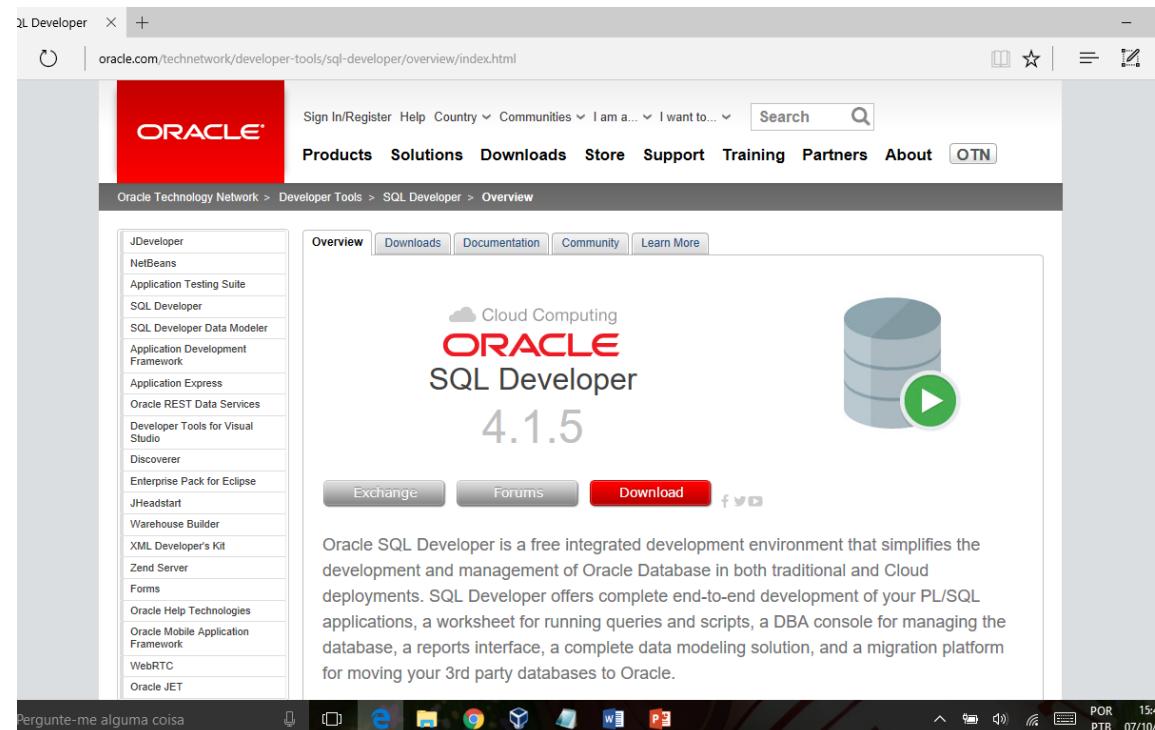
Please note that this appliance is for testing purposes only, as such it is unsupported and should not be used in production environment. This virtual machine contains:

- Oracle Linux 7
- Oracle Database 12c Release 1 Enterprise Edition (12.1.0.2 with In-Memory Option)
- Oracle XML DB
- Oracle SQL Developer 4.1.3
- Oracle Application Express 5.0
- Oracle REST Data Services 3.0.5
- Hands-On-Labs (accessed via the Toolbar Menu in Firefox)

OTN Developer Day Database VirtualBox Appliance

<http://www.oracle.com/technetwork/database/enterprise-edition/databaseappdev-vm-161299.html>

Arquivos Necessários



Oracle SQL Developer

<http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>

Importante:

Se ao tentar executar o sqldeveloper.exe (Windows) ocorrer o erro “unable to launch the java virtual machine located at path”:

Copiar o arquivo MSVCR100.dll do diretório de instalação do Virtual Box, exemplo:

C:\Program Files\Oracle\VirtualBox\msvcr100.dll

Para o diretório:

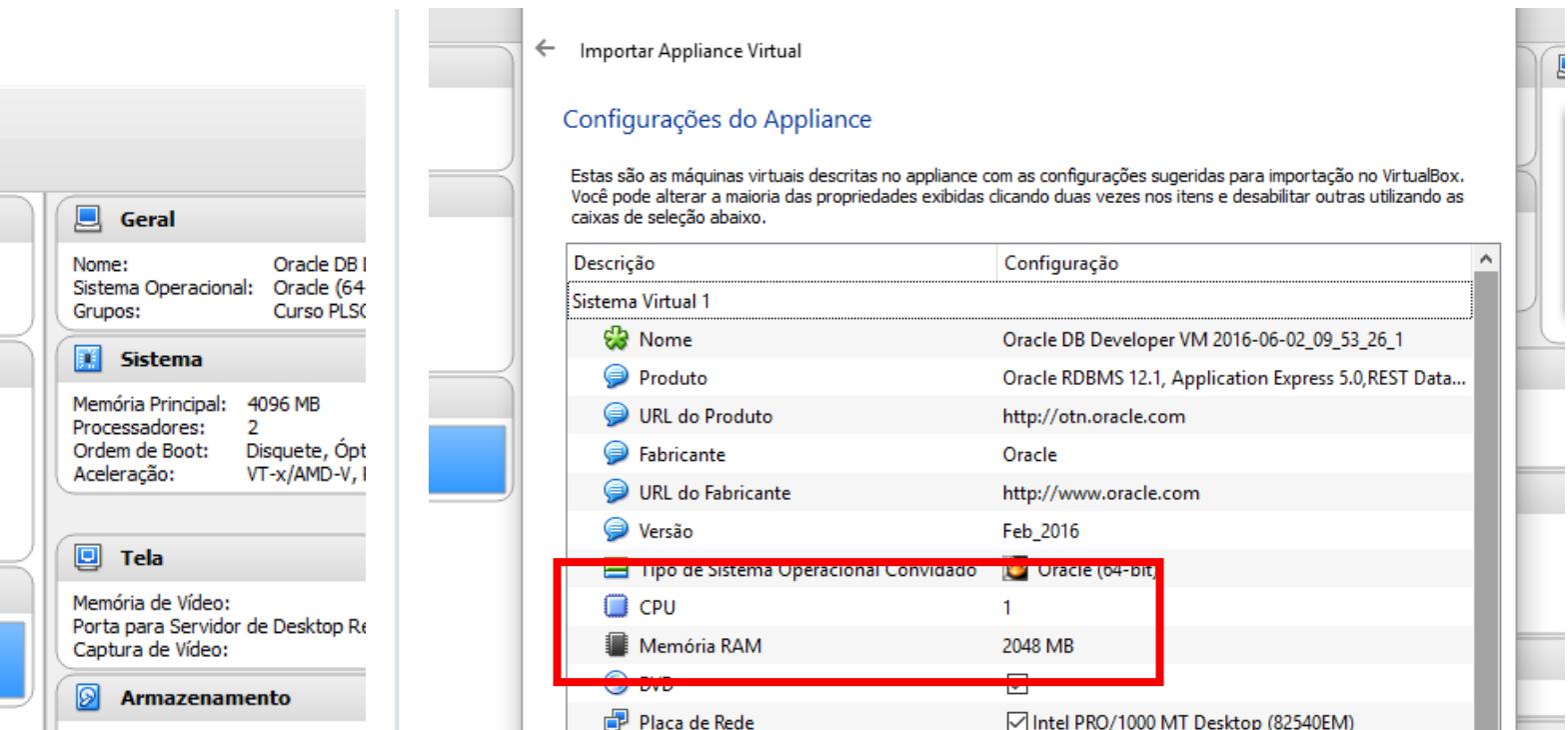
<seu caminho>\sqldeveloper\sqldeveloper\bin

Configuração do Ambiente: Importação da VM



Passo 1: Importar Appliance

Arquivo -> Importar Appliance...



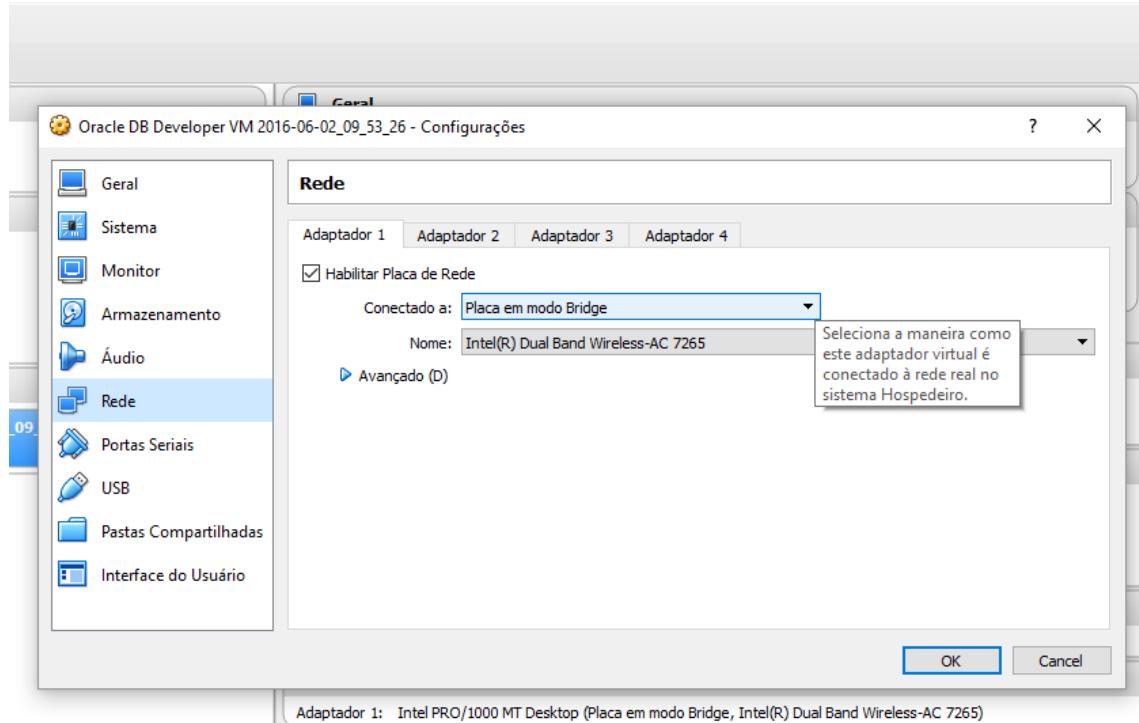
The screenshot shows the 'Importar Appliance Virtual' configuration dialog. It includes a 'Configurações do Appliance' section with a note about suggested configurations for importing the appliance. A table lists various settings with their current values:

Descrição	Configuração
Sistema Virtual 1	
Nome	Oracle DB Developer VM 2016-06-02_09_53_26_1
Produto	Oracle RDBMS 12.1, Application Express 5.0, REST Data...
URL do Produto	http://otn.oracle.com
Fabricante	Oracle
URL do Fabricante	http://www.oracle.com
Versão	Feb_2016
Tipo de Sistema Operacional Convidado	Oracle (64-bit)
CPU	1
Memória RAM	2048 MB
DVD	
Placa de Rede	<input checked="" type="checkbox"/> Intel PRO/1000 MT Desktop (82540EM)

Passo 2: Configuração de CPU e Memória

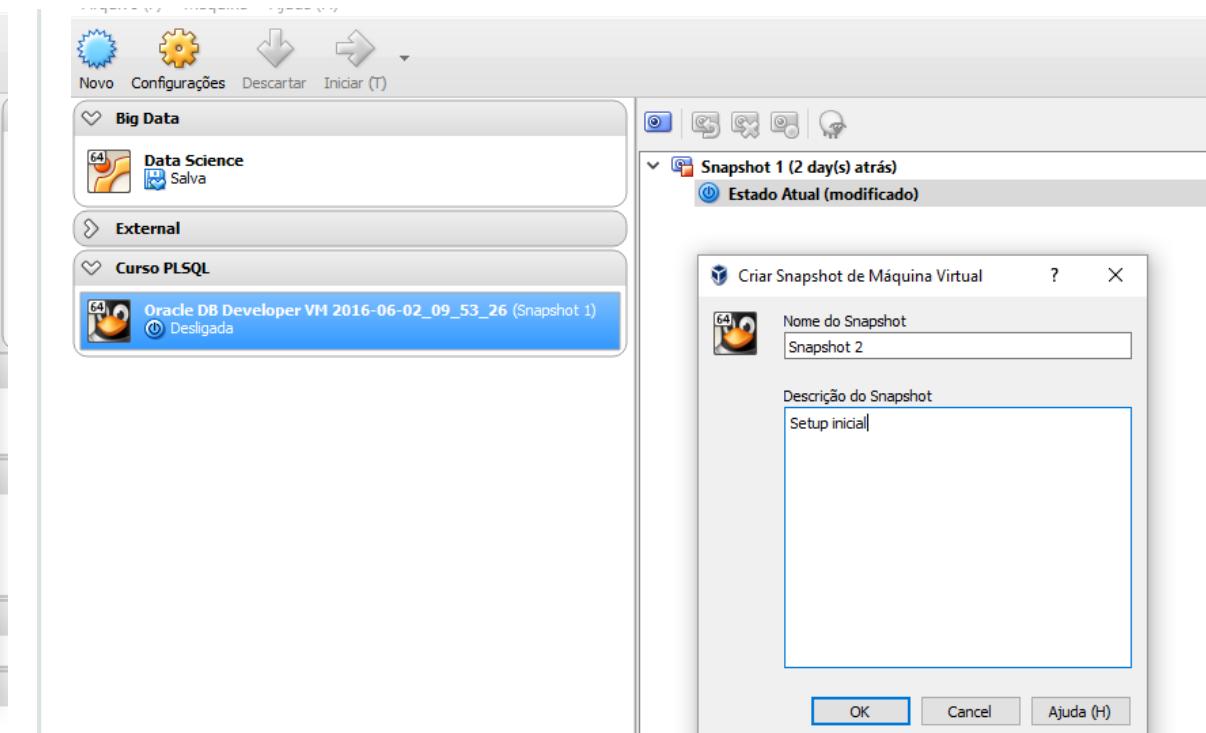
Deixar pelo menos 1 GB para o SO hospedeiro e 1 core (físico) livre.

Configuração do Ambiente: Rede e Snapshot



Passo 3: Alterar placa de rede para modo bridge

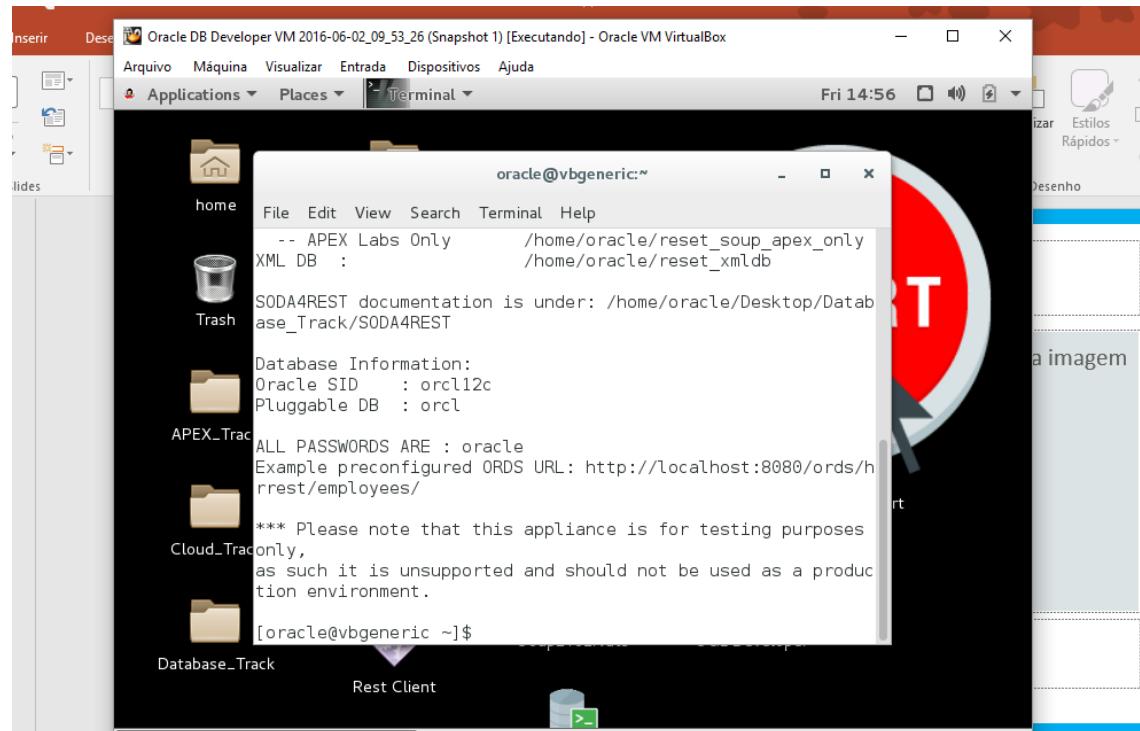
Configurações -> Rede



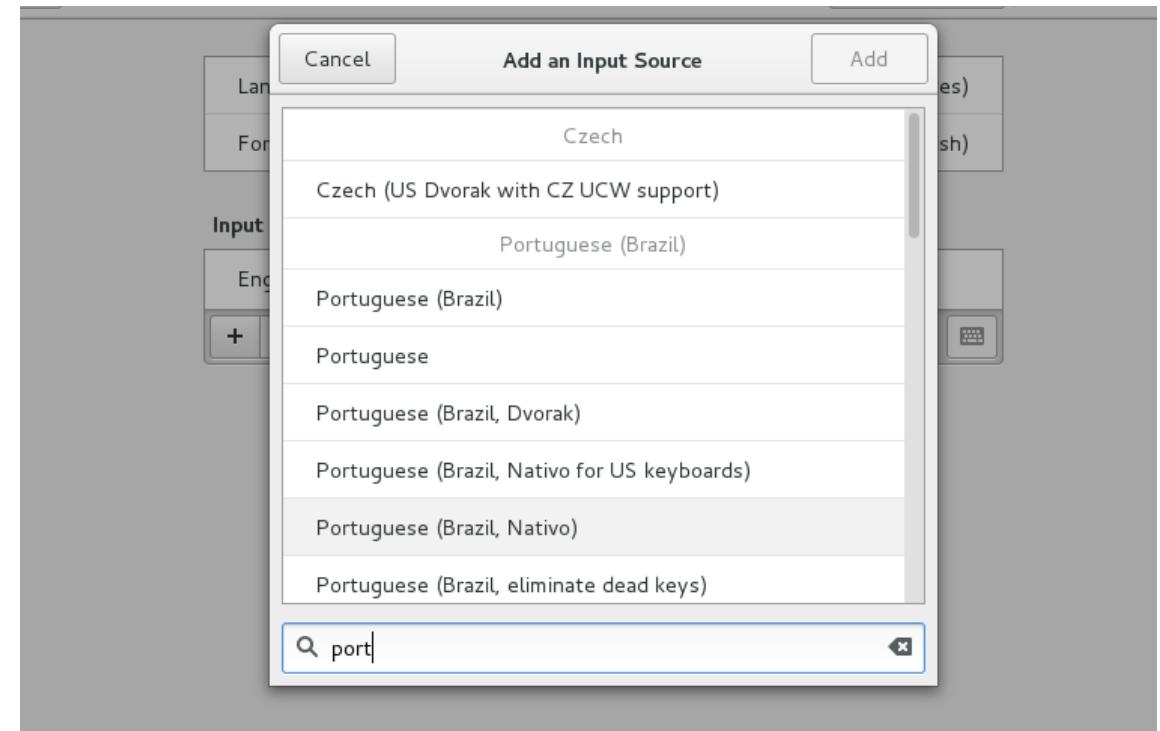
Passo 4: Criar Snapshot

Snapshots -> Criar... Ou Ctrl+Shift+S

Configuração do Ambiente: Teclado



Passo 5: Iniciar a VM



Passo 6: Configuração do Teclado

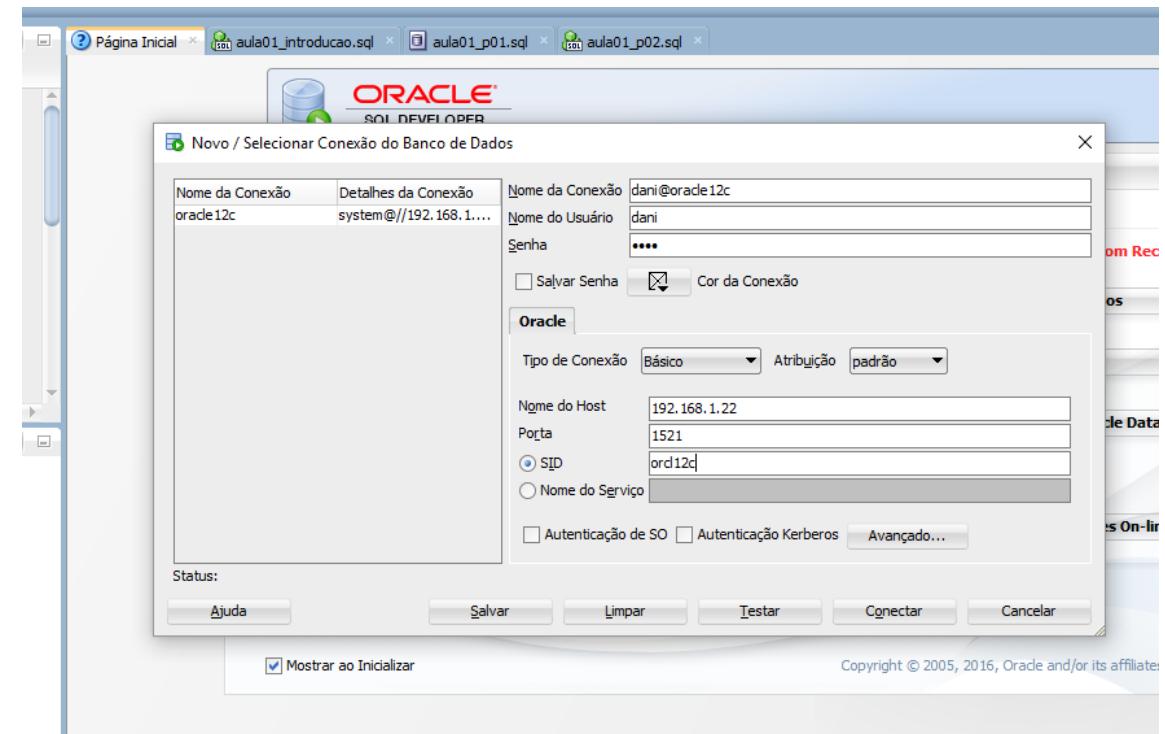
Applications -> System Tools -> Settings -> Keyboard -> Input Sources

Configuração do Ambiente: Cliente

```
[oracle@vbgeneric ~]$ ifconfig enp0s3
enp0s3: flags=4162<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
          inet 192.168.1.19 netmask 255.255.255.0 broadcast 192.168.1.255
              brd 192.168.1.255 scope 1:0:1:c0 txqueuelen 1000 (Ethernet)
                  RX packets 20666 bytes 2074553 (1.9 MiB)
                  RX errors 0 dropped 0 overruns 0 frame 0
                  TX packets 1039 bytes 190191 (185.7 KiB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
[oracle@vbgeneric ~]$
```

Passo 7: Localizar o endereço IP

Digitar no terminal da VM: ifconfig enp0s3



Passo 7: SQL Developer

No SQL Developer criar conexões para o serviço **orcl** e para o SID **orcl12c**, usuários **SYS** e **SYSTEM**

Todos os passwords são **oracle**

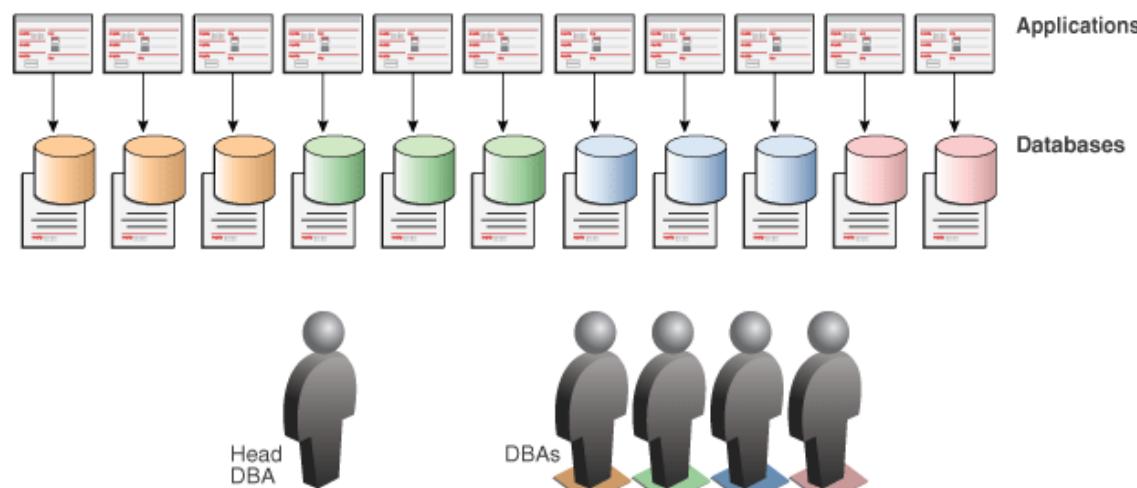
Scripts das Aulas Práticas

- Todos os scripts estão disponíveis no GitHub:
 - https://www.github.com/danicat/curso_tuning_v2
- Vocês podem:
 - Instalar o ‘git’ na máquina virtual:

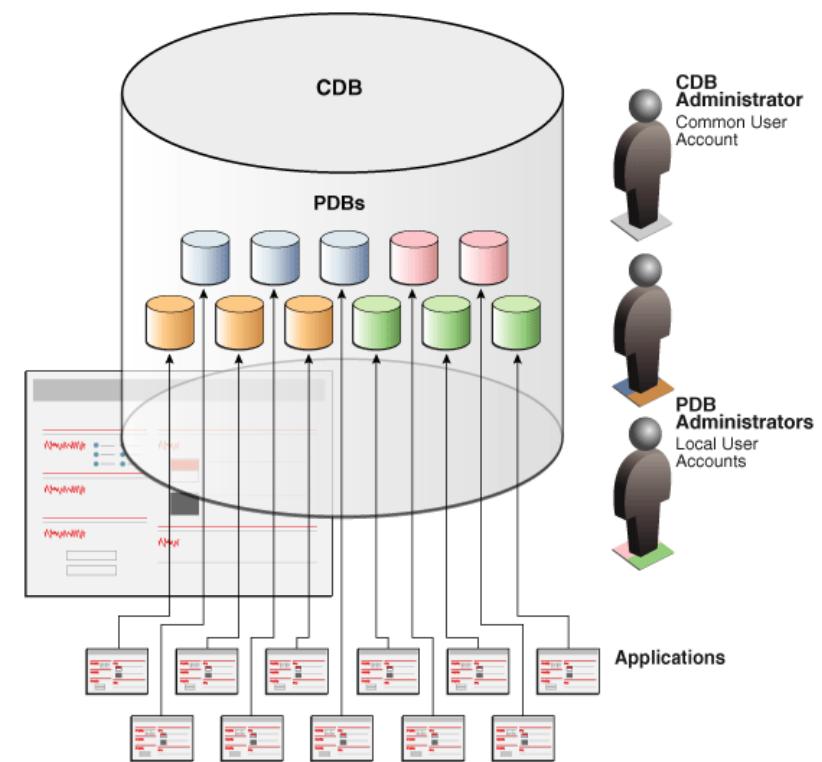
```
sudo yum install -y git  
git clone https://www.github.com/danicat/curso_tuning_v2
```
 - Instalar o ‘git’ no host (depende do sistema operacional)
 - Ou apenas baixar os arquivos (botão “*Clone or Download*”)

Entendendo a VM: Arquitetura *Multitenant*

Single Tenant (Tradicional)

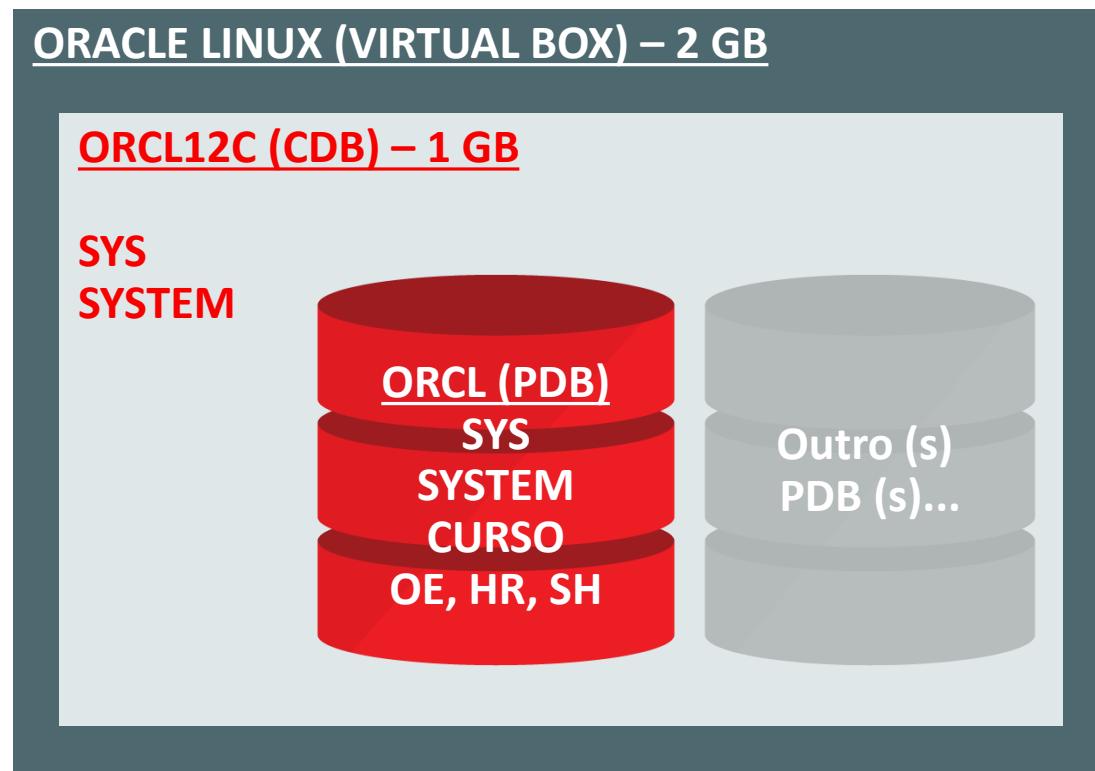


Multitenant



Arquitetura do Ambiente do Curso

- Arquitetura *Multitenant*
- Container (CDB)
 - Parâmetros globais
 - SID = orcl12c
 - SYS, SYSTEM
- Banco de Dados (PDB)
 - Parâmetros locais (instância)
 - service_name = orcl
 - SYS, SYSTEM
 - Usuários locais: CURSO, OE, HR, SH



Prática 01: Configuração do Ambiente

- Tarefas
 - Baixar os arquivos indicados
 - Instalar o VirtualBox
 - Importar a VM
 - Instalar e Configurar o SQL Developer (ou outro client)
 - Testar a conexão
 - Criar usuário CURSO
- Notas:
 - As senhas de SYS e SYSTEM são todas **oracle**.

Performance Tuning com Oracle 12c

Aula 02: Arquitetura I

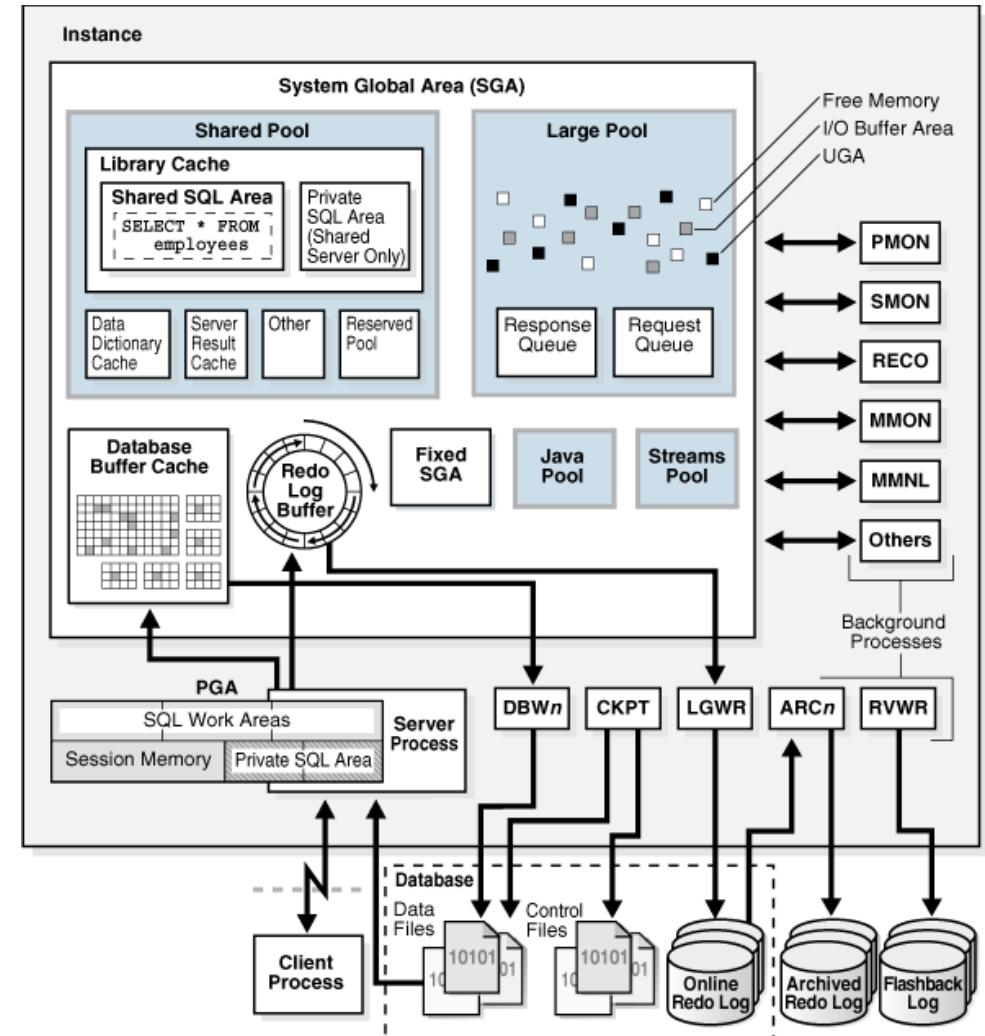
Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

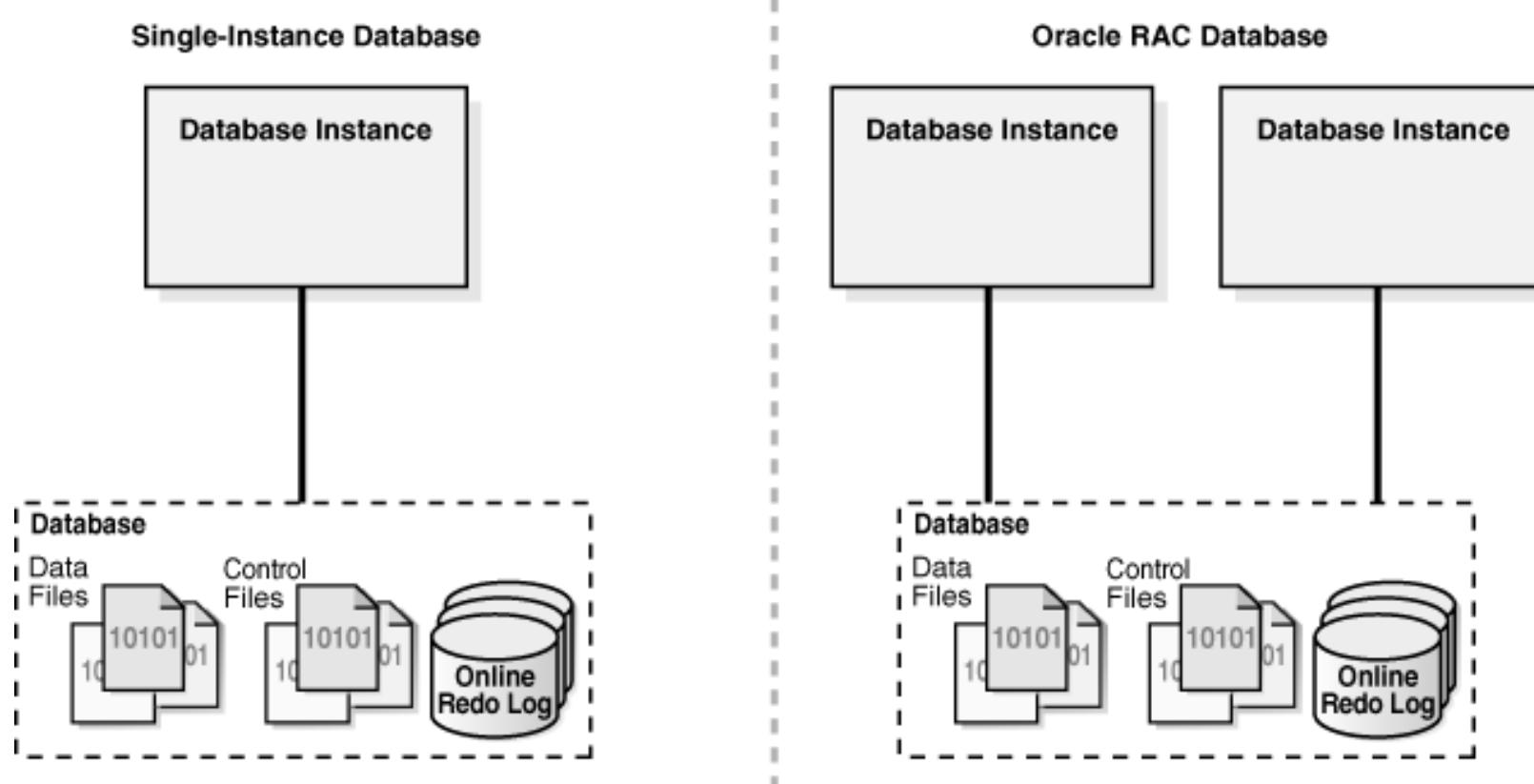
- Arquitetura Oracle
- Arquitetura de Rede
- Estruturas de Memória: SGA
- Estruturas de Memória: PGA

Arquitetura Oracle Database

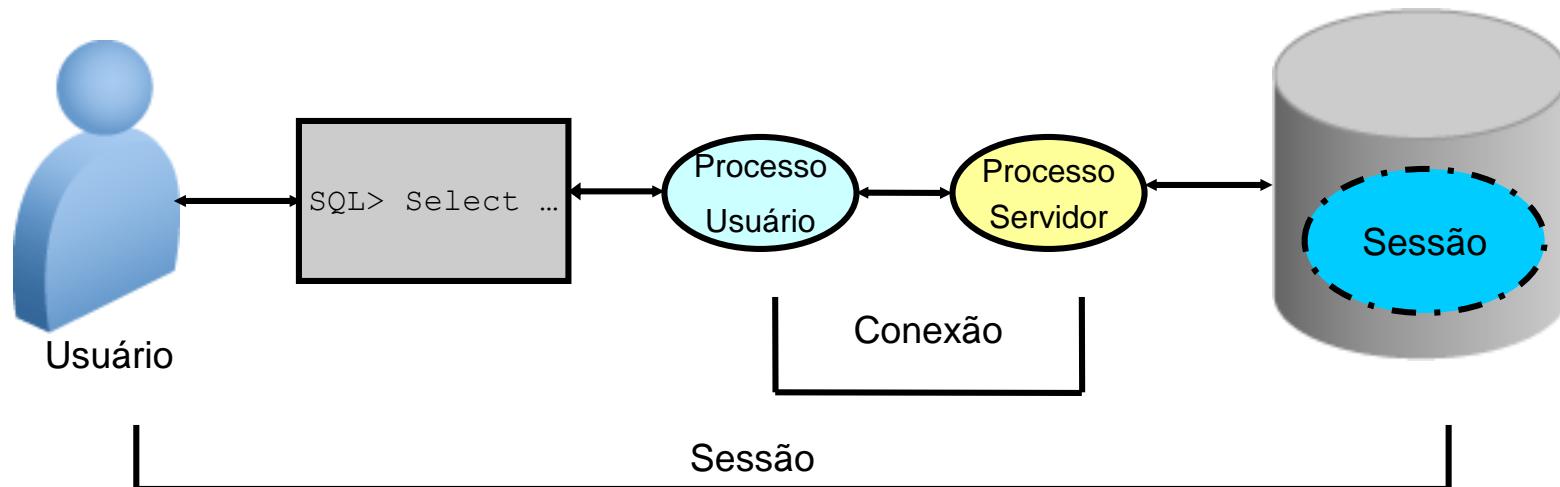
- Principais Componentes:
 - Estruturas de Memória
 - Estruturas de Disco (Arquivos)
 - Processos
- Instância
 - Conjunto de processos e estruturas de memória
- Banco de Dados:
 - Estruturas de Disco (Arquivos)



Relação: Instâncias x Databases



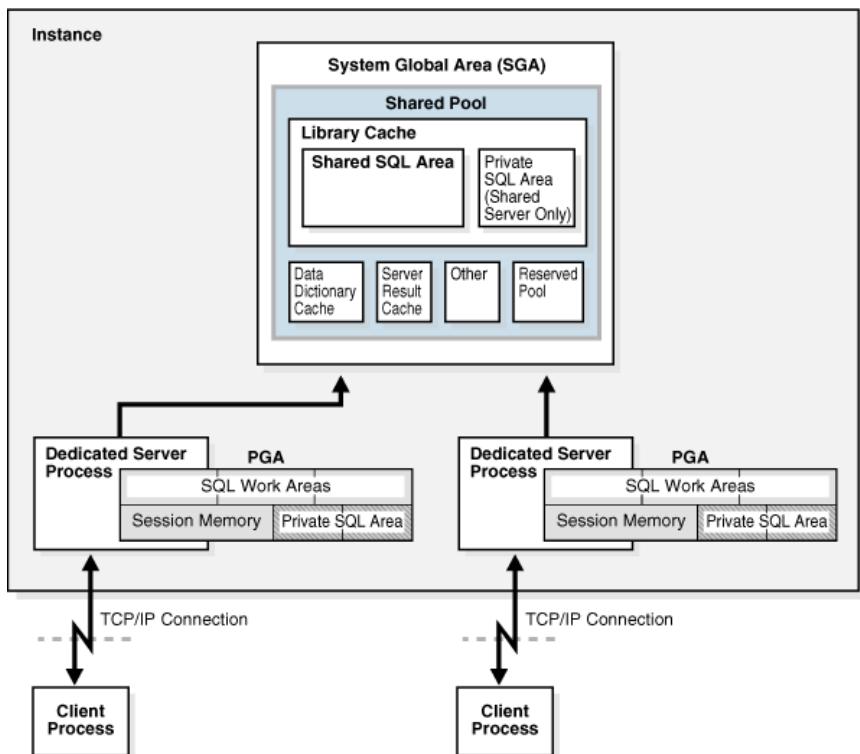
Arquitetura de Rede



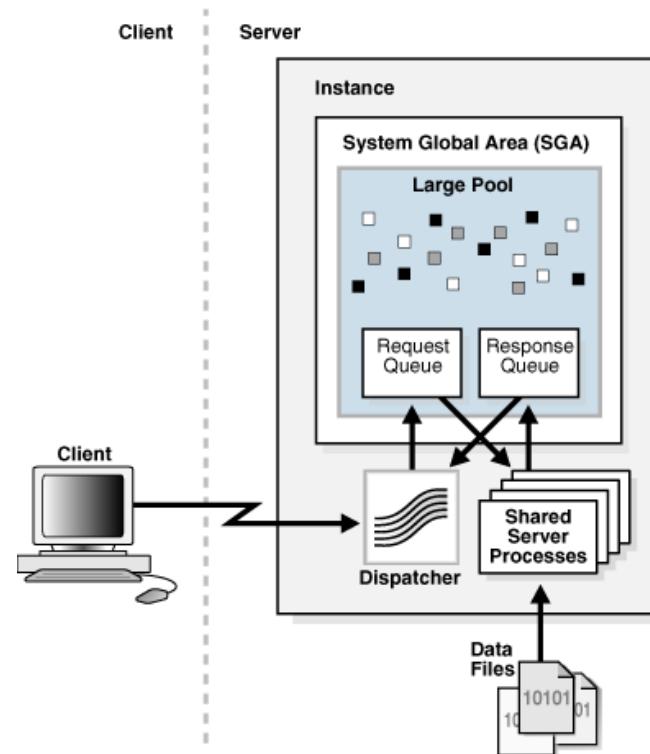
- **Conexão:** canal de comunicação entre o processo do usuário e o processo do servidor.
 - IPC: mesma máquina
 - Rede (TCP/IP): máquinas diferentes
- **Sessão:** estado do usuário durante uma conexão

Arquitetura de Rede: Modelos

Dedicated Server

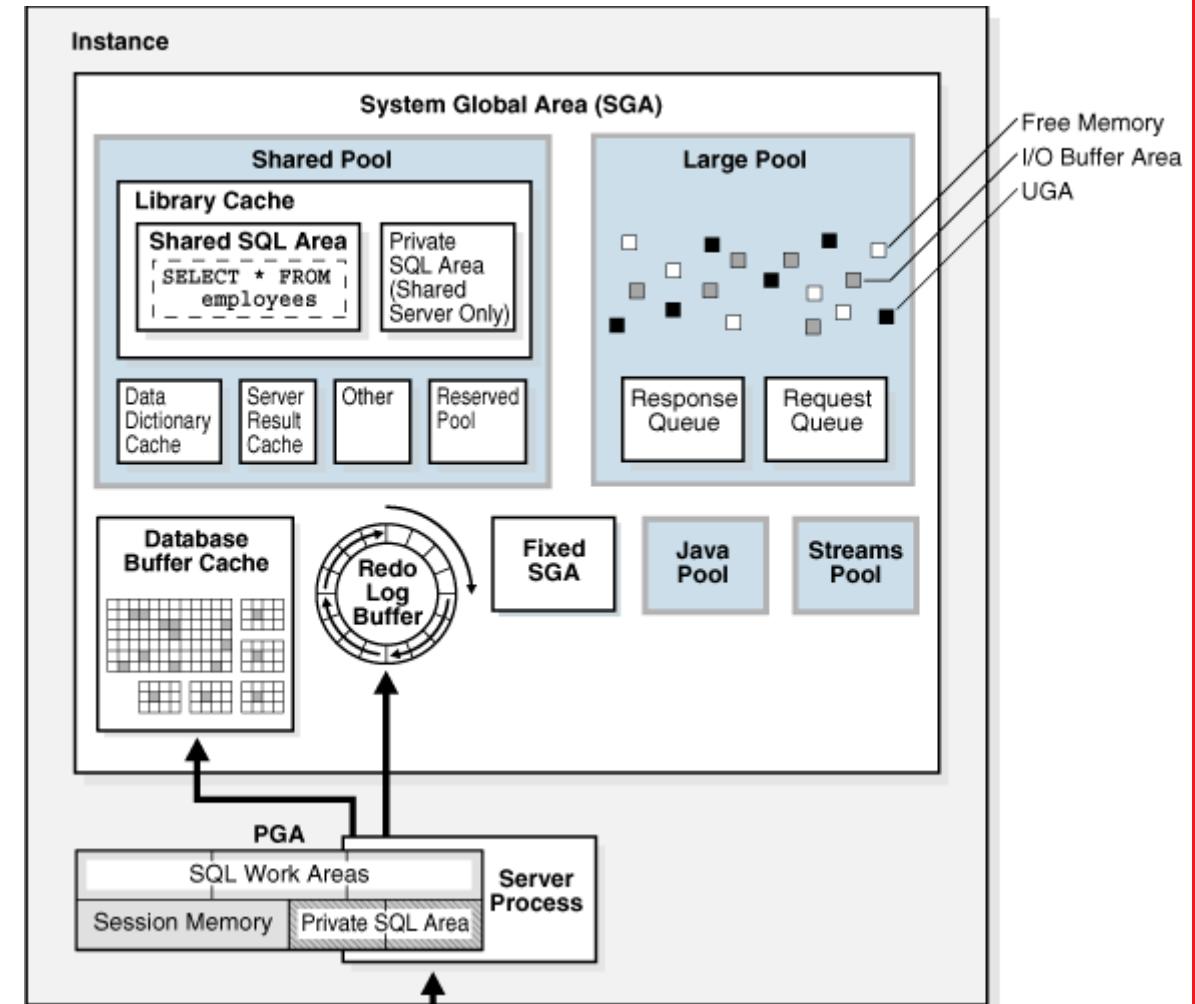


Shared Server



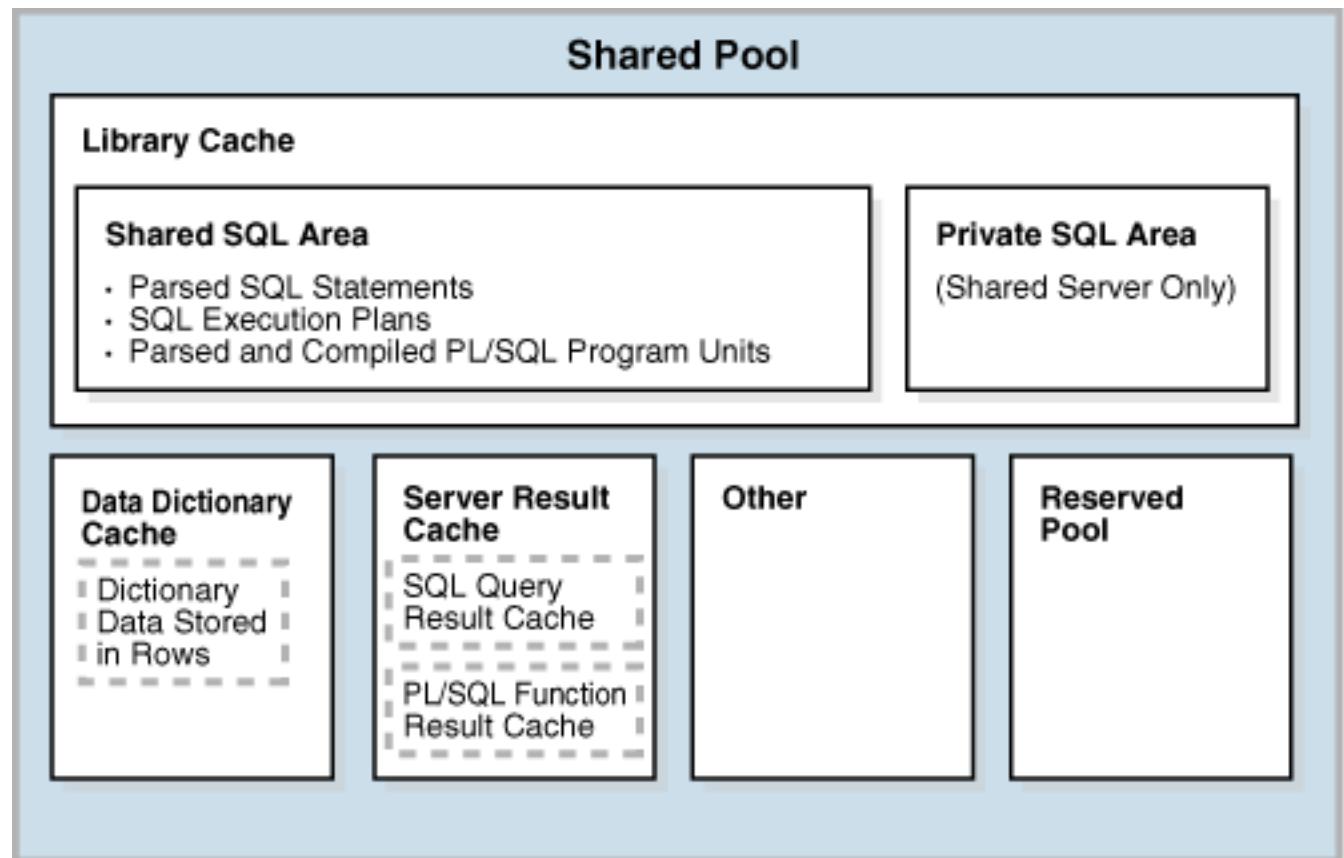
Estruturas de Memória

- System Global Area (SGA)
 - Compartilhada por todos os processos
- Program Global Area (PGA)
 - Conjunto de todas as áreas de memória privada dos processos



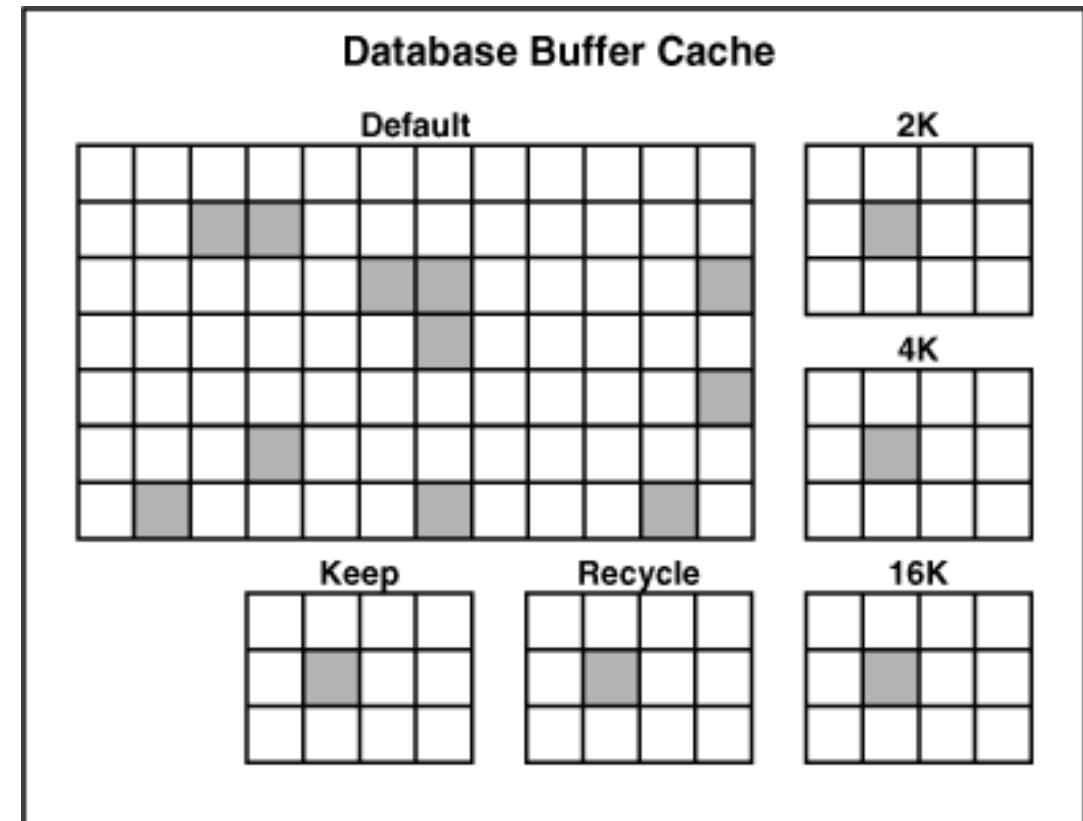
SGA: Shared Pool

- Library Cache
 - Shared SQL Area
 - Private SQL Area (Shared Server)
- Data Dictionary Cache
- SQL Query Result Cache
- PL/SQL Query Result Cache



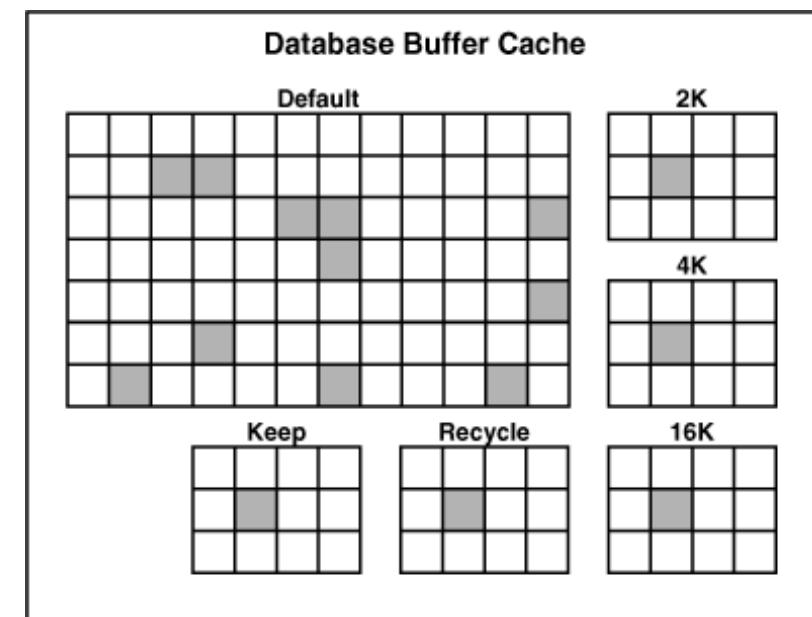
SGA: Buffer Cache

- Guarda cópias dos blocos lidos do disco
 - Blocos padrão: DB_BLOCK_SIZE [8k]
 - Tamanho: DB_CACHE_SIZE
 - Blocos não-padrão são salvos em buffers independentes (nK buffer cache)
 - Nota: as transações correntes gravam direto no buffer cache
 - Consistent Read (CR): reconstrução de uma imagem consistente de um bloco que está sendo atualizado por uma transação.



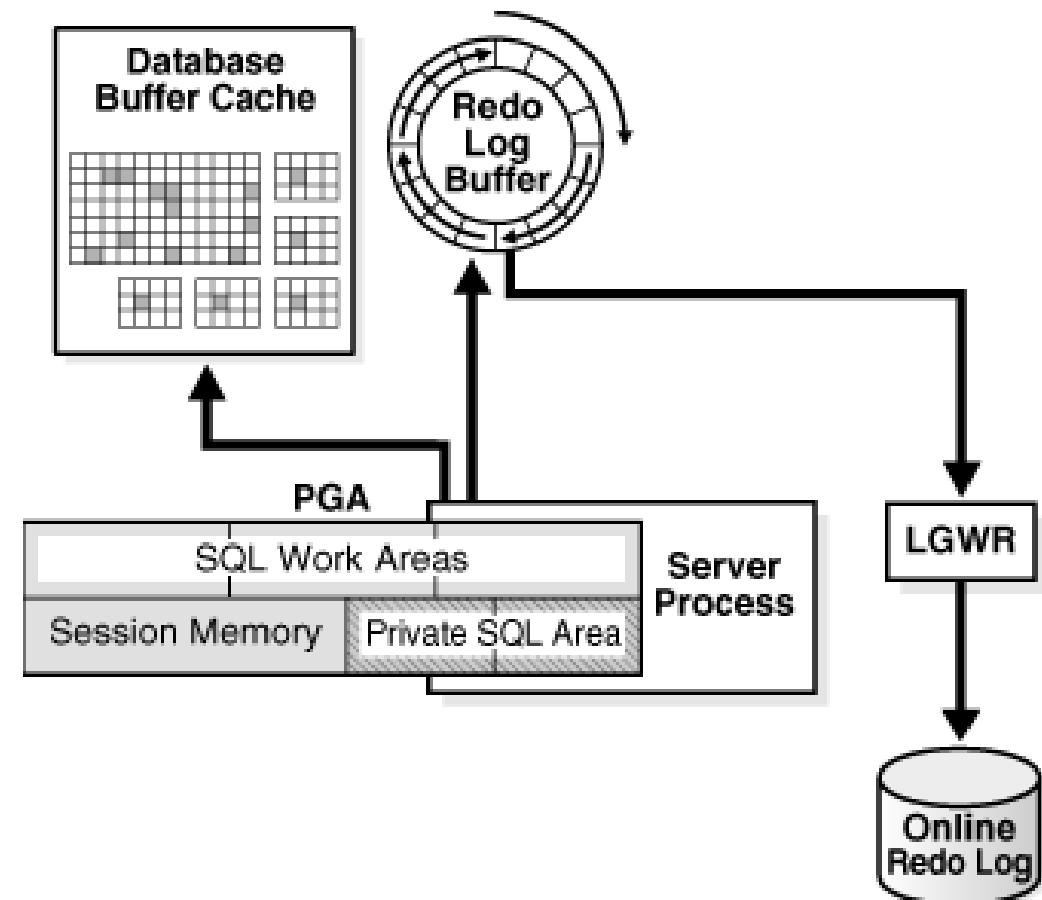
Buffers Keep e Recycle

- Keep: mantém dados “in-memory” (!!!)
- Recycle: destinados para leituras “descartáveis”, para evitar que processos “destruam” o buffer pool default.
- Todos os buffers tem o mesmo algoritmo: LRU
- Como fazer:
 - ALTER TABLE T STORAGE (BUFFER_POOL KEEP)
 - ALTER TABLE T STORAGE (BUFFER_POOL RECYCLE)
- Não confundir KEEP com CACHE
 - CACHE dá maior prioridade no LRU (“fura fila”)
 - ALTER TABLE T CACHE;



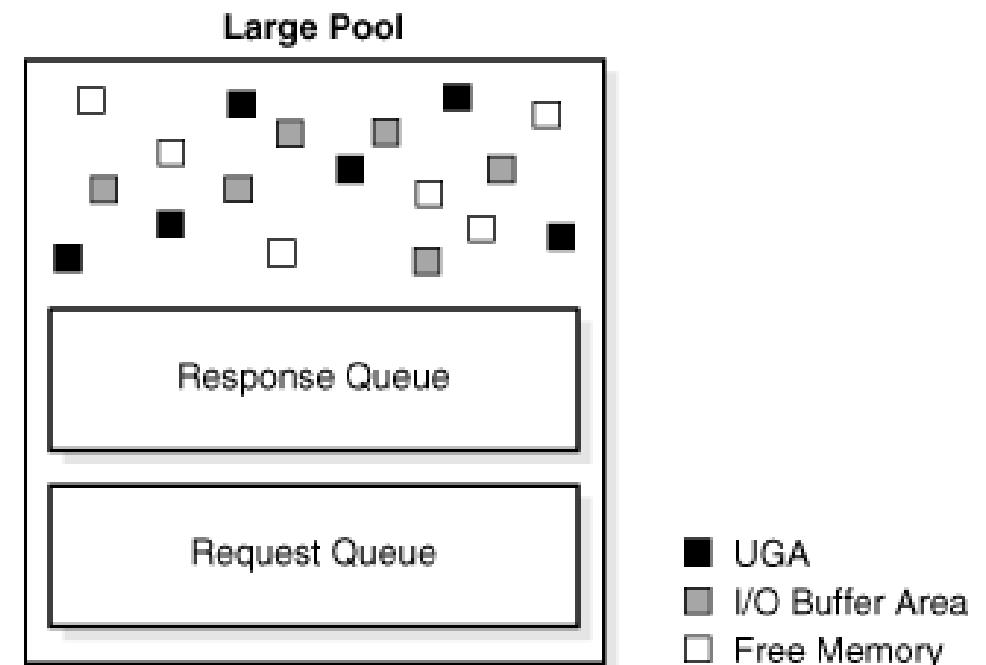
SGA: Redo Log Buffer

- Buffer circular
 - Least Recently Used (LRU)
- Contém informações sobre alterações realizadas no banco de dados
- É gravado periodicamente para o disco pelo processo LGWR
- Crítico para processos de Recovery



SGA: Large Pool

- Área com múltiplas finalidades, importante para:
 - Shared server (modelo de conexão)
 - Transações distribuídas (XA)
 - **Parallel Query Buffers**
 - Backup e Restore
 - AQ (Advanced Queuing)



SGA: Java e Streams pool

- Java pool
 - JVM interna do banco
 - Java Stored Procedures
 - Linha de comando: **loadjava**
 - Direto no banco: CREATE JAVA
- Streams pool
 - Oracle Streams (*deprecated* no 12c!)
 - Golden Gate
 - Xstream
 - AQ
 - Data Pump

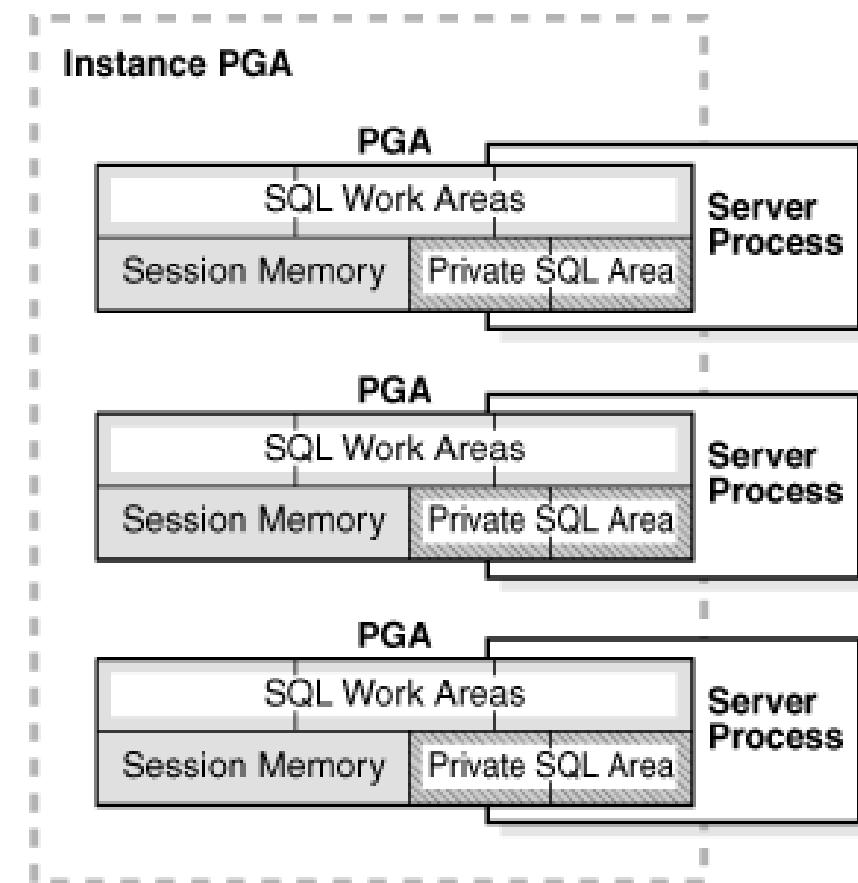
```
$ loadjava -user HR Oscar.class

SQL> CREATE FUNCTION oscar_quote RETURN VARCHAR2
2 AS LANGUAGE JAVA
3 NAME 'Oscar.quote() return java.lang.String';

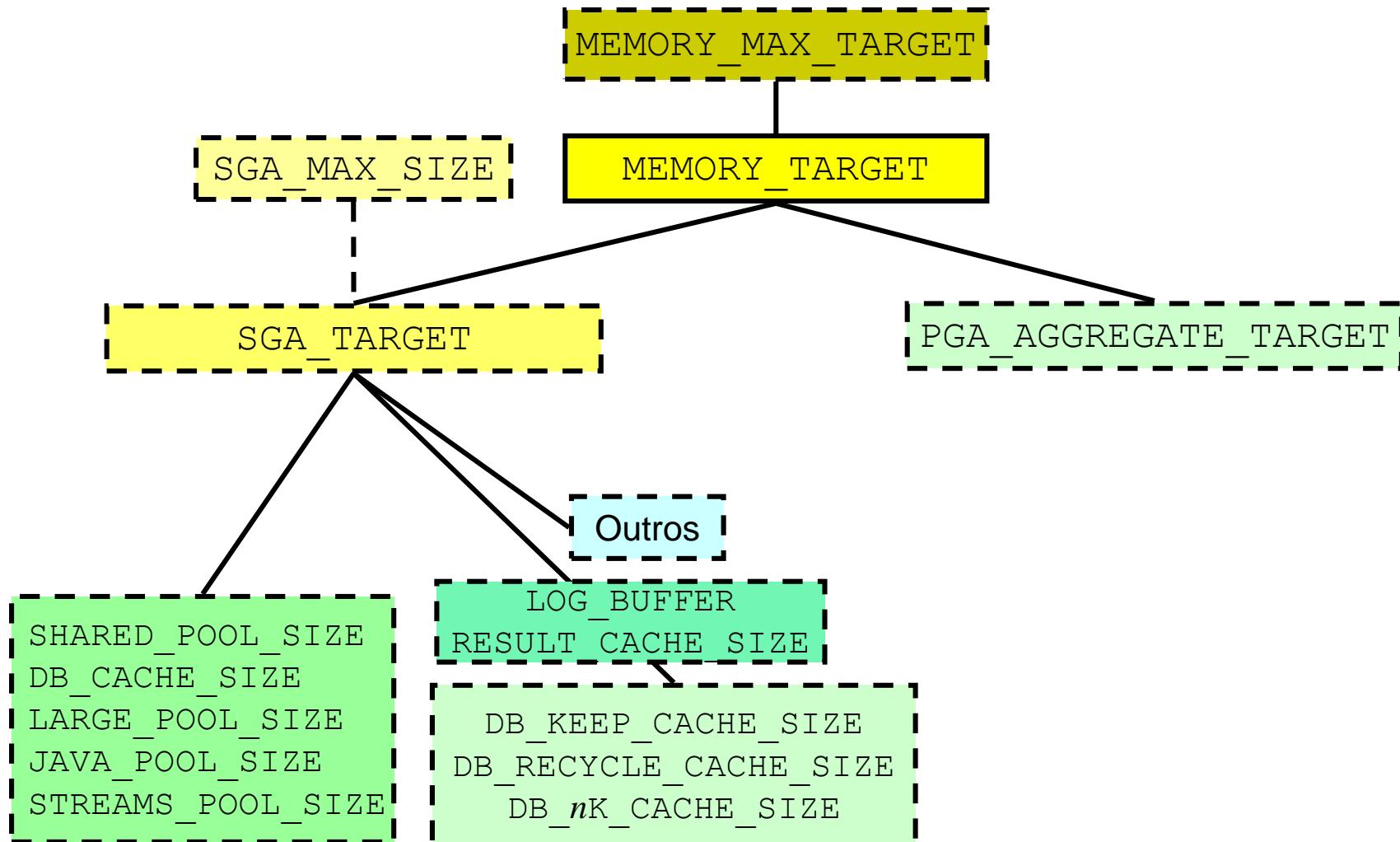
CREATE JAVA SOURCE NAMED "Welcome" AS
public class Welcome {
    public static String welcome() {
        return "Welcome World";    } }
/
```

PGA: Program Global Area

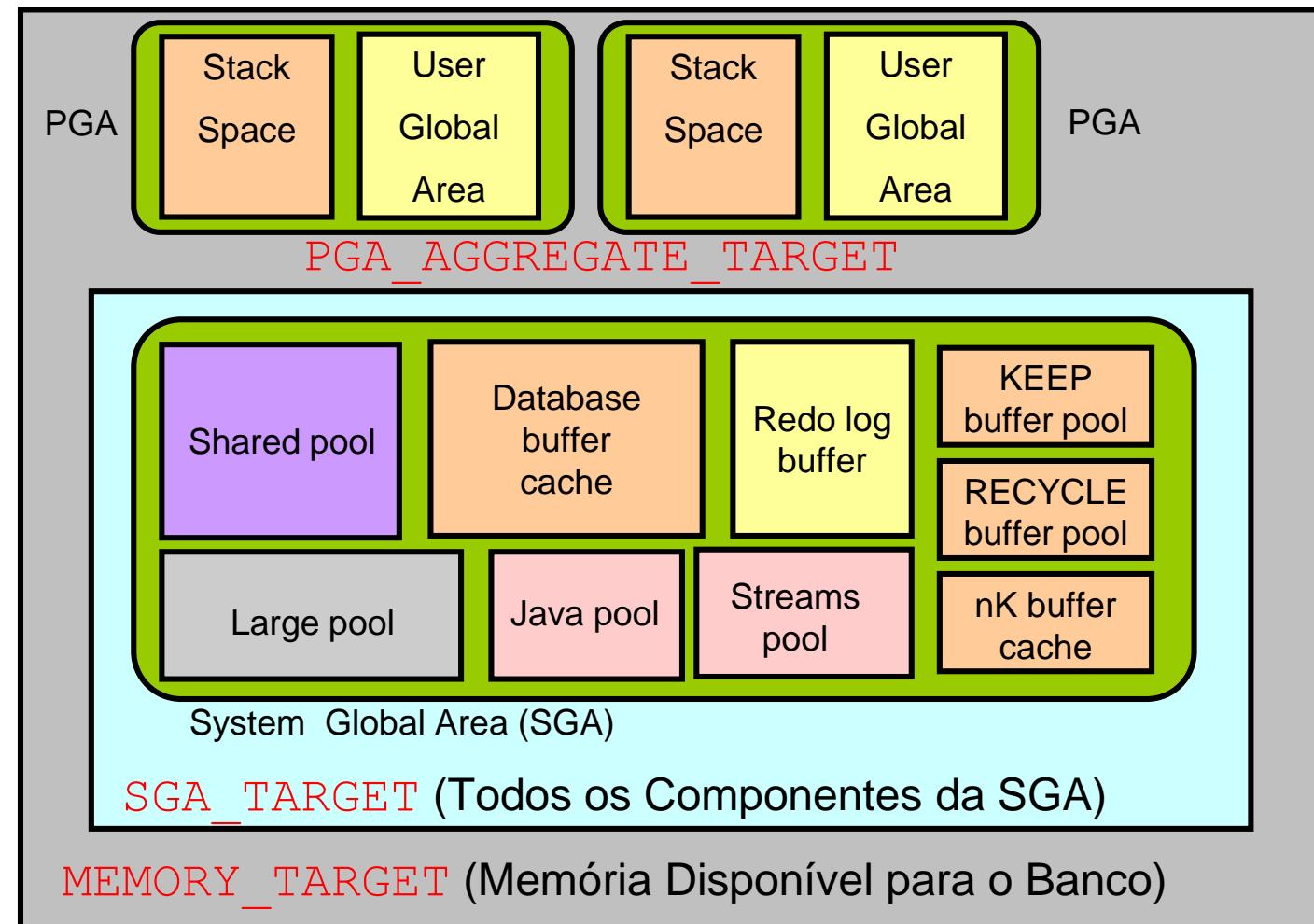
- Área de memória privada para os *server processes*
 - PGA total = PGA_AGGREGATE_TARGET
- Conexão Dedicada
 - User Global Area (UGA)
 - SQL working area:
 - Sort area: Order by, Group By
 - Hash area: Hash join
- Conexão Compartilhada
 - Private SQL Area -> Library Cache
 - UGA -> Shared Pool ou Large Pool



Principais Parâmetros de Memória



Gerenciamento Automático de Memória



Prática 02: Estruturas de Memória

- Objetivo: evidenciar as diferentes estruturas de memória e explorar o seu acesso pelo dicionário de dados.
- Views:
 - v\$parameter
 - v\$sga, v\$sgainfo, v\$sgastat
 - v\$sga_dynamic_components, v\$sga_dynamic_free_memory
 - v\$sga_resize_ops, v\$sga_current_resize_ops
 - v\$process
 - v\$pgastat
 - v\$buffer_pool, v\$buffer_pool_statistics

Performance Tuning com Oracle 12c

Aula 03: Arquitetura II

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

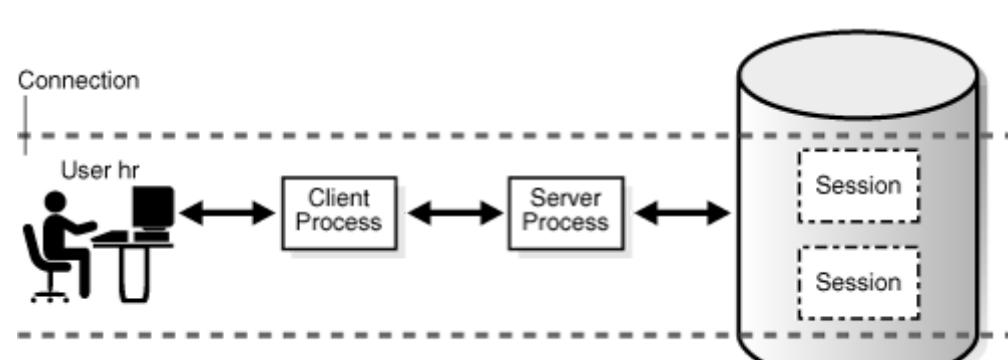
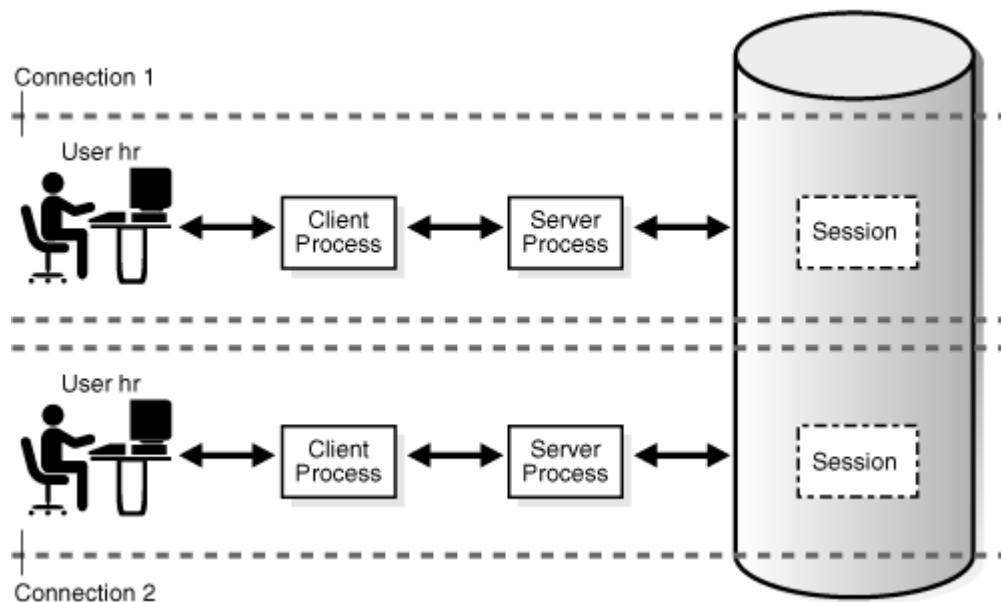
Objetivos

- Arquitetura de Processos
- Estruturas de Disco
- Tipos de Inserção
- Tipos de Leitura
- Transações e SCN

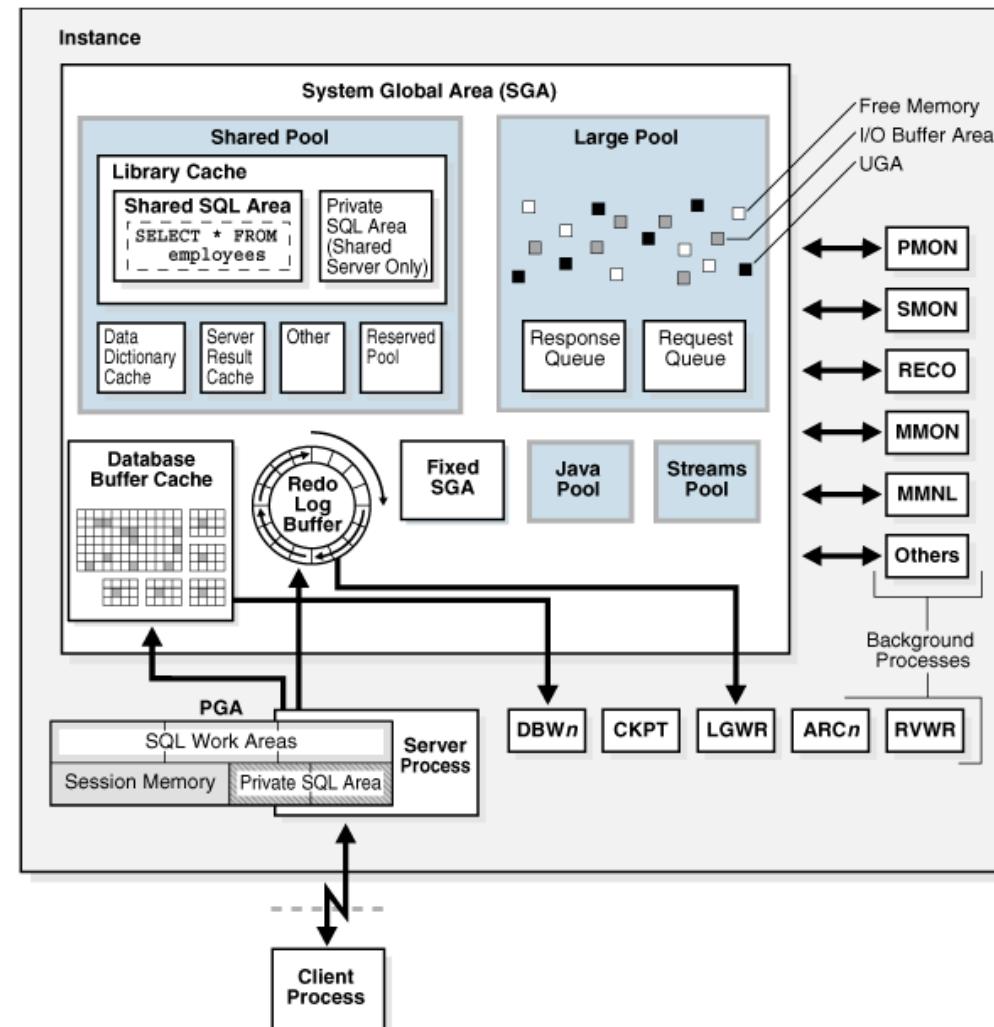
Arquitetura de Processos

- Processo do Usuário
 - Aplicação ou ferramenta que conecta no banco de dados (cliente)
- Processos do Banco de Dados
 - Processos do Servidor (*Server Process*): contraparte do processo do usuário no servidor/instância de banco
 - Processos em *Background*: operações internas
- Daemons (Serviços no Windows)
 - Listeners
 - Grid infrastructure

Arquitetura de Processos

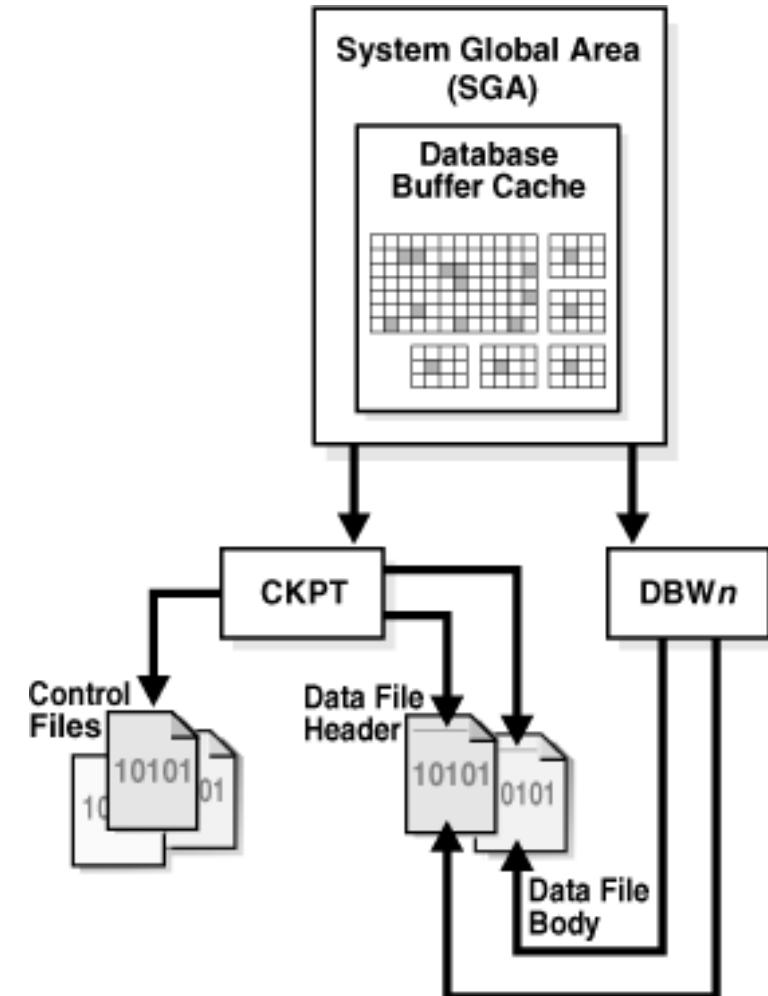


Arquitetura de Processos: Visão Geral



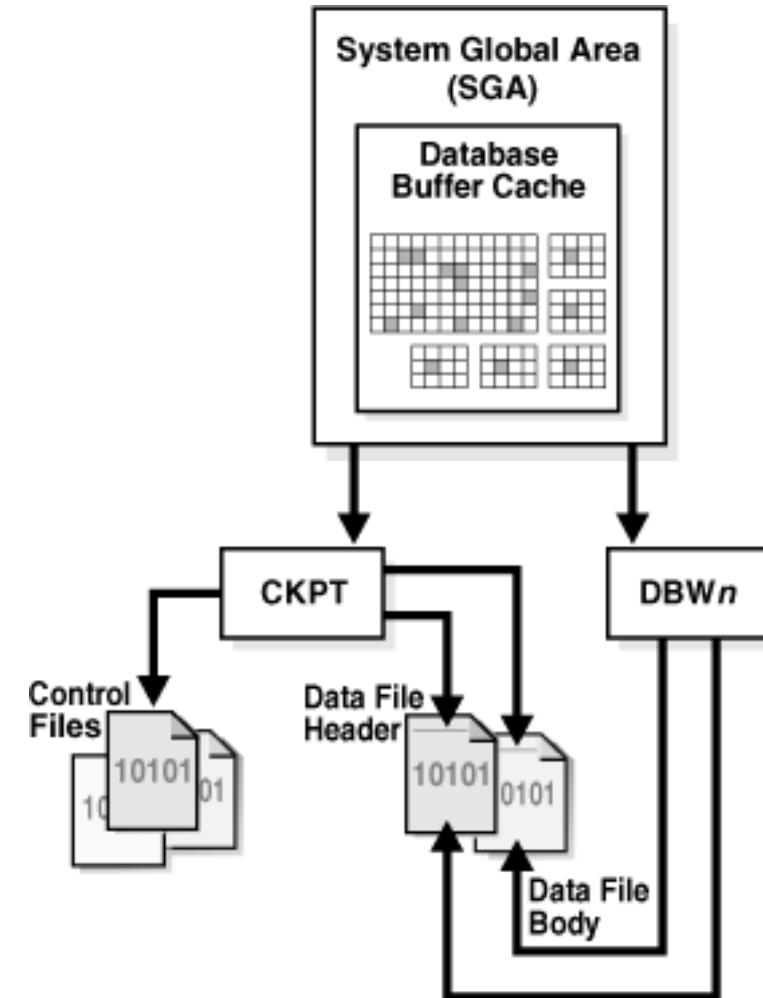
Processos: Checkpoint (CKPT)

- Grava as informações de *checkpoint* – *system change number (SCN)*:
 - Nos *control files*
 - Nos *headers* de cada *data file*
- Pelo menos um *checkpoint* a cada *log file switch*
- Importante ponto de *tuning* de instância



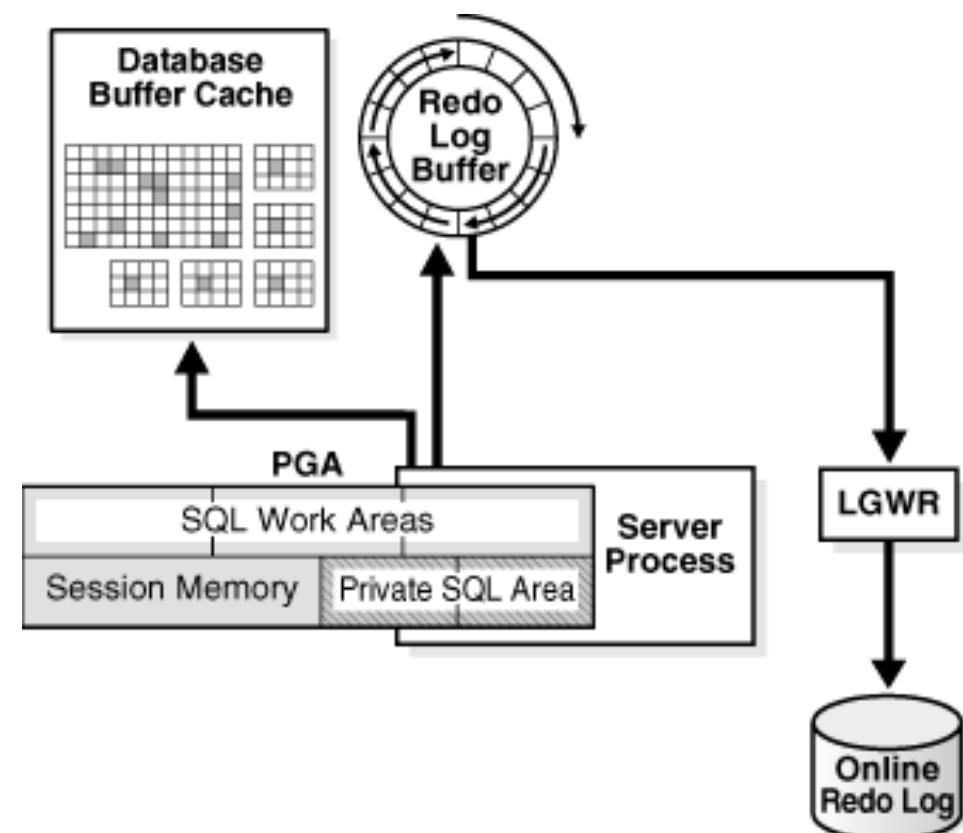
Processos: Database Writer (DBWn)

- Responsável por gravar os blocos “sujos” do *buffer cache* para o disco.
 - DBW0
 - Opcional: DBWn: $1 \leq N \leq 9$ ou ‘a’ $\leq N \leq 'z'$
 - Opcional: BW36 a BW99
- Assíncrono (*commit* ocorre em memória!)
- LRU: descarrega blocos menos usados para o disco



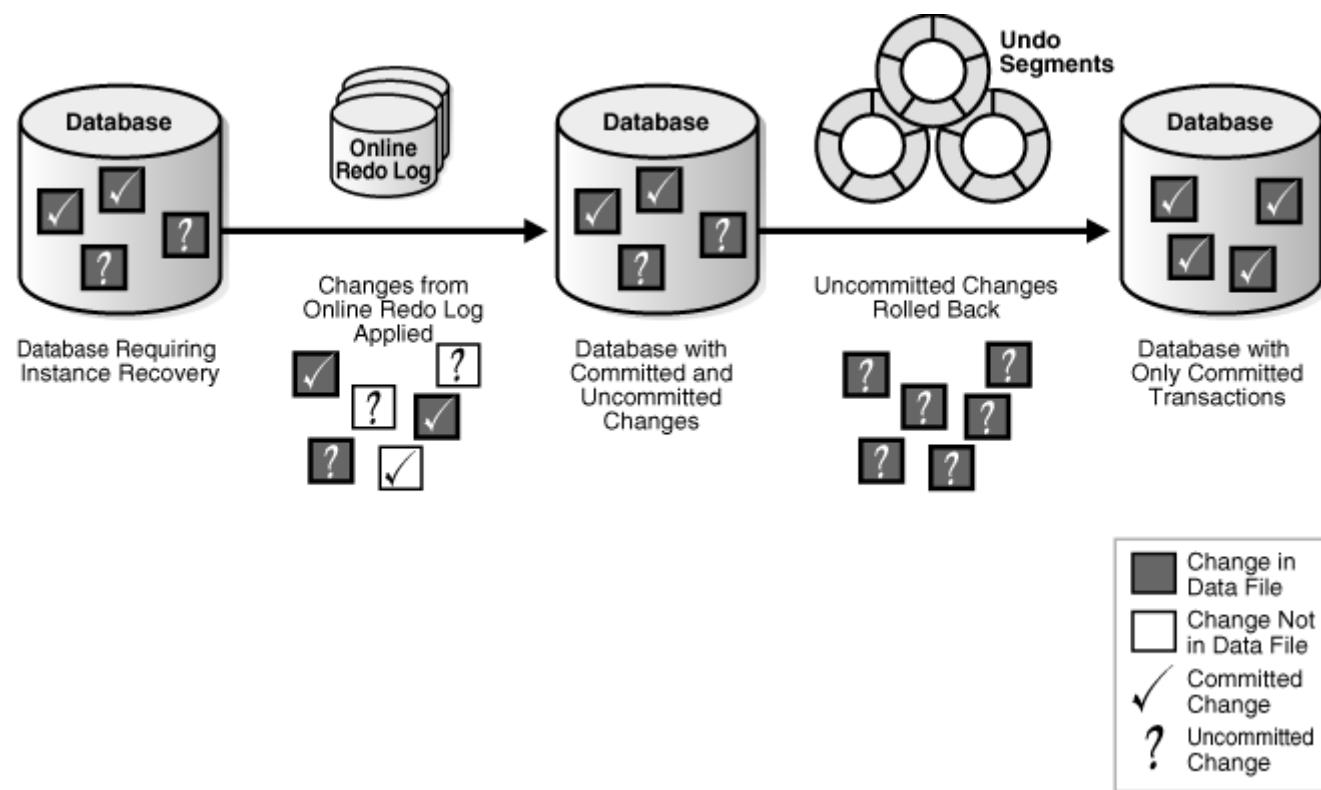
Processos: Log Writer (LGWR)

- Grava o *redo log buffer* para o *redo log file*
- Condições – a primeira de:
 - Quando um processo do usuário faz *commit*
 - Quando o redo log buffer está 1/3 cheio
 - Antes que um processo DBWn grave no disco
 - A cada 3 segundos



Processos: System Monitor (SMON)

- Realiza o processo de *recovery* na inicialização do banco (se necessário)
- Faz a limpeza de segmentos temporários



Processos: Process Monitor (PMON)

- Realiza recuperação de processos quando processos do usuário falham
 - Limpa o database *buffer cache*
 - Libera recursos utilizados pelo processo
- Monitora as sessões por timeouts
- Registra dinamicamente os serviços do banco de dados no listener (pre-12c)
 - ALTER SYSTEM REGISTER;
 - Pós 12c: processo LREG

Processos: Outros

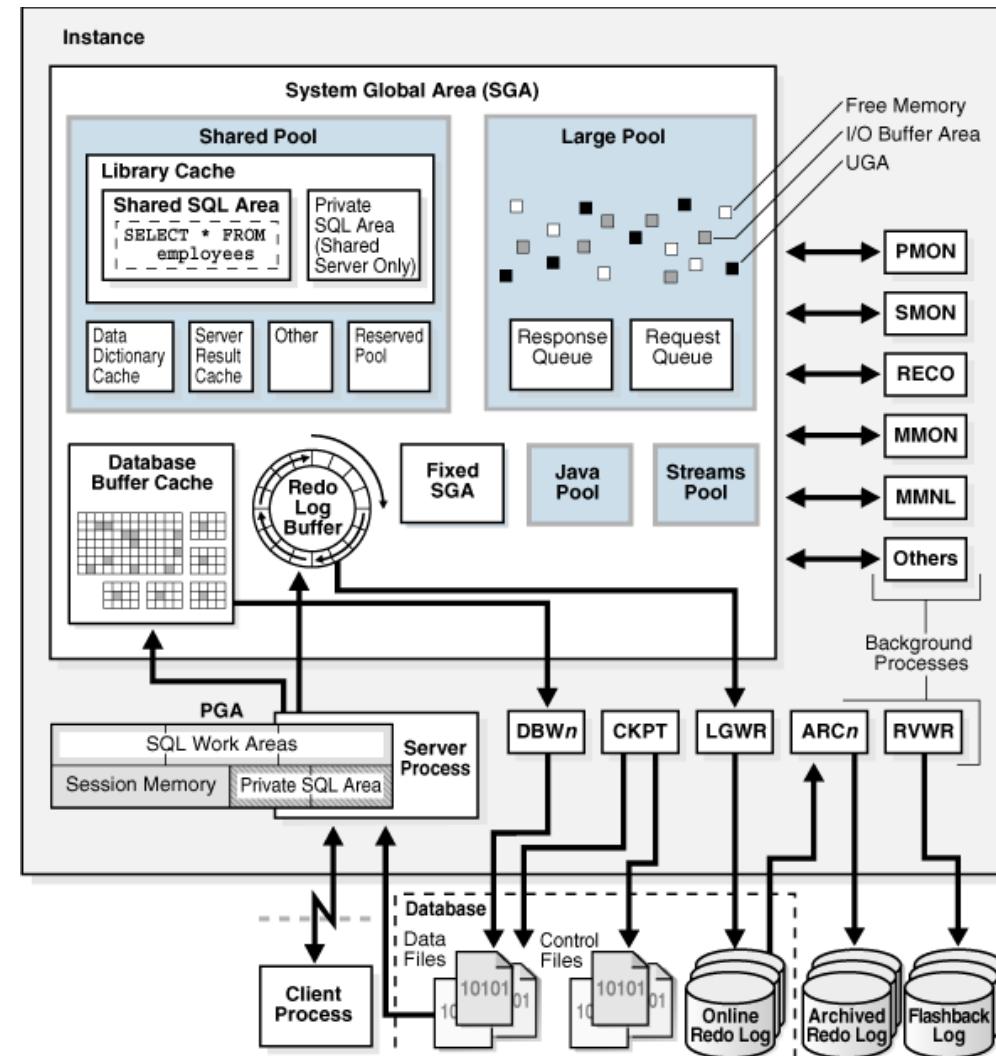
- Recoverer Process (RECO): resolve conflitos em transações distribuídas
- Archiver Process (ARCn):
 - copia *archive logs* para *recovery area* após um *log switch*
 - Transmite dados de *redo* para *stand by* (*Data Guard*)
- Shared Server Process (Snnn) e Dispatcher Process (Dnnn)
 - Arquitetura Shared Server
- Parallel Execution / Query Slaves (Pnnn)
- Memory Manager (MMAN)
 - Gerenciamento automático de memória

Prática 03a: Processos

- Objetivos: explorar as principais *views* de processos e saber vincular um processo no banco à sua contraparte no sistema operacional.
- Views:
 - v\$process: ADDR
 - v\$session: PADDR
- No terminal:
 - ps -ef | grep SPID | grep -v grep
- Matando uma sessão: kill -9 SPID
 - Equivalente a: ALTER SYSTEM KILL SESSION ‘sid,serial’;
 - **Atenção:** matar qualquer um dos processos DBWn, CKPT, LGWR, SMON, PMON ou RECO irá abortar a instância!

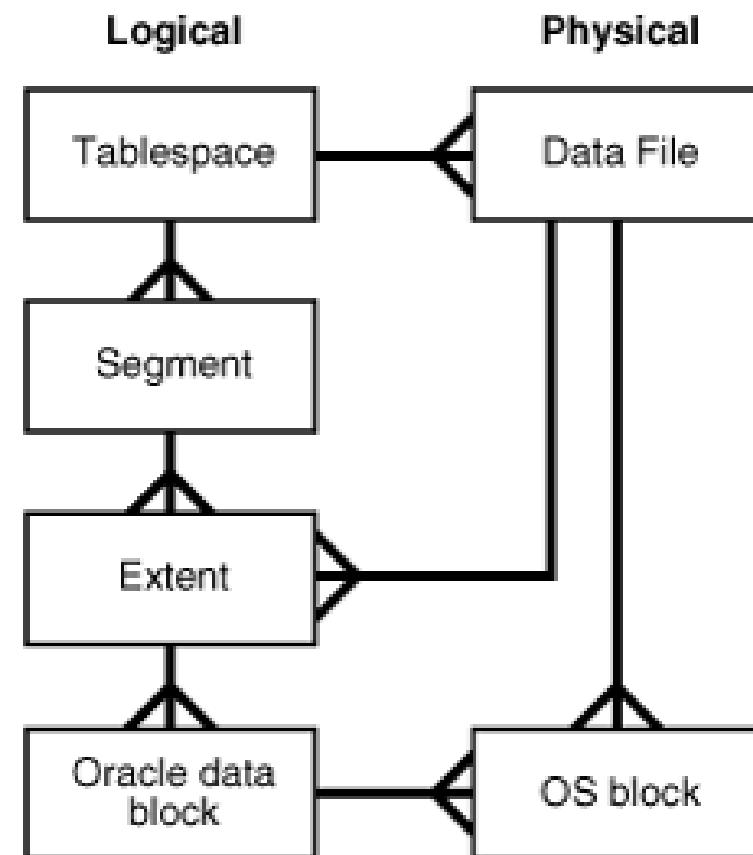
Estruturas de Disco

- Database:
 - Data Files
 - Control Files
 - Online Redo Log
- Outros:
 - Archived Redo Log
 - Flashback Log
 - SPFILE
 - Password File
 - Backups
 - Logs (alert, trace, etc)

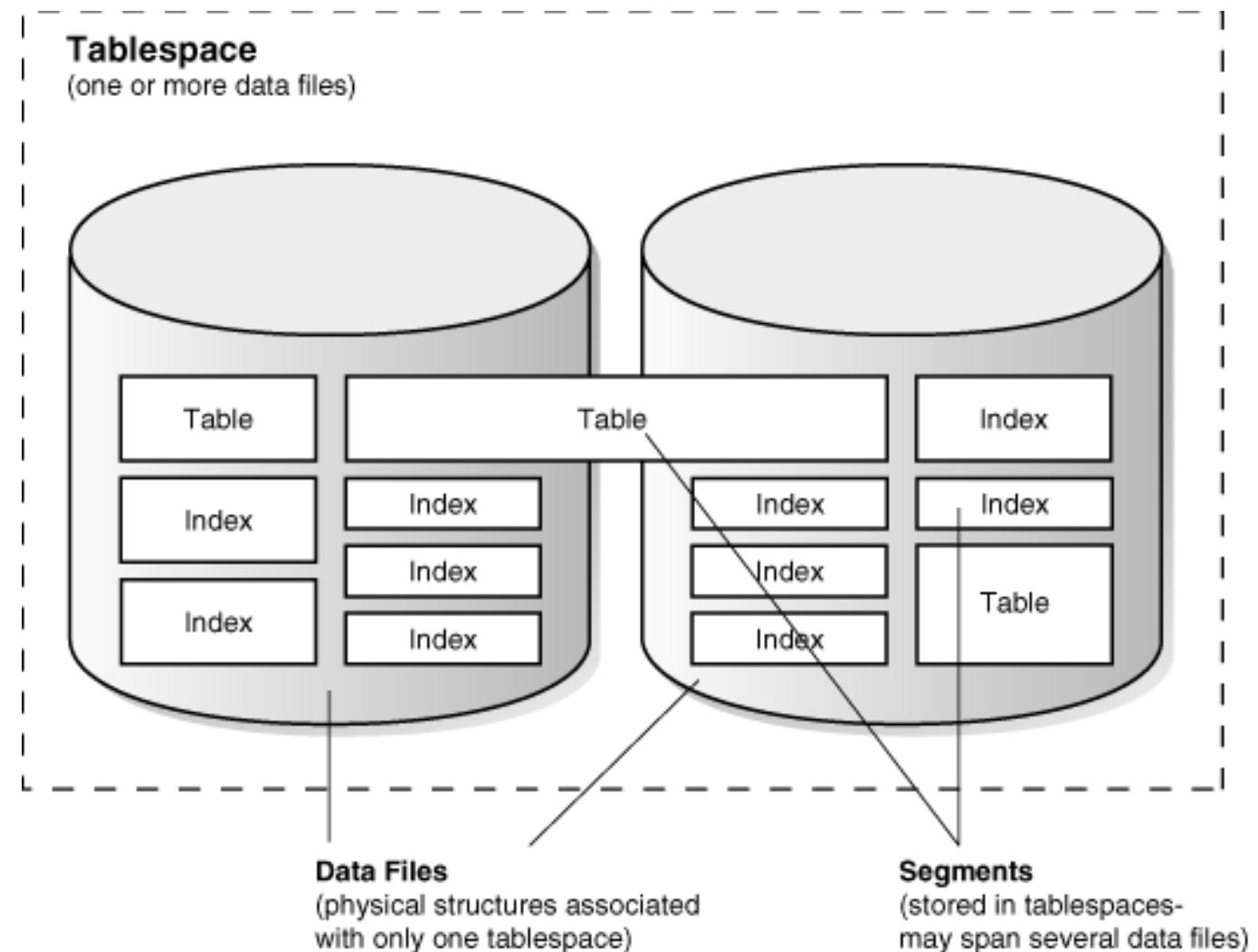


Estruturas Físicas e Lógicas

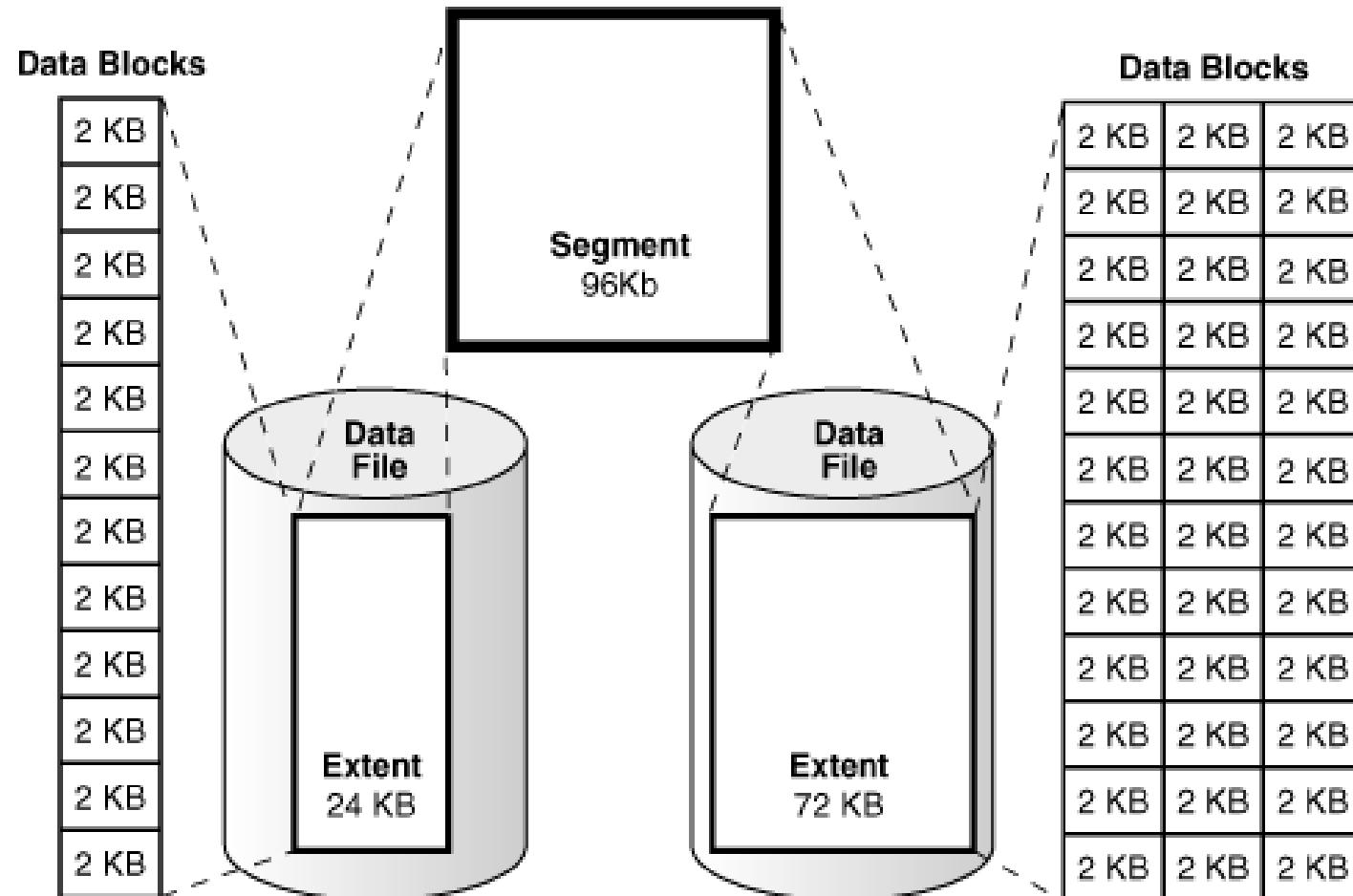
- Físicas:
 - Data Files
 - Blocos do dispositivo (ex.: filesystem)
- Lógicas
 - Database Blocks
 - Segments
 - Extents
 - Tablespace



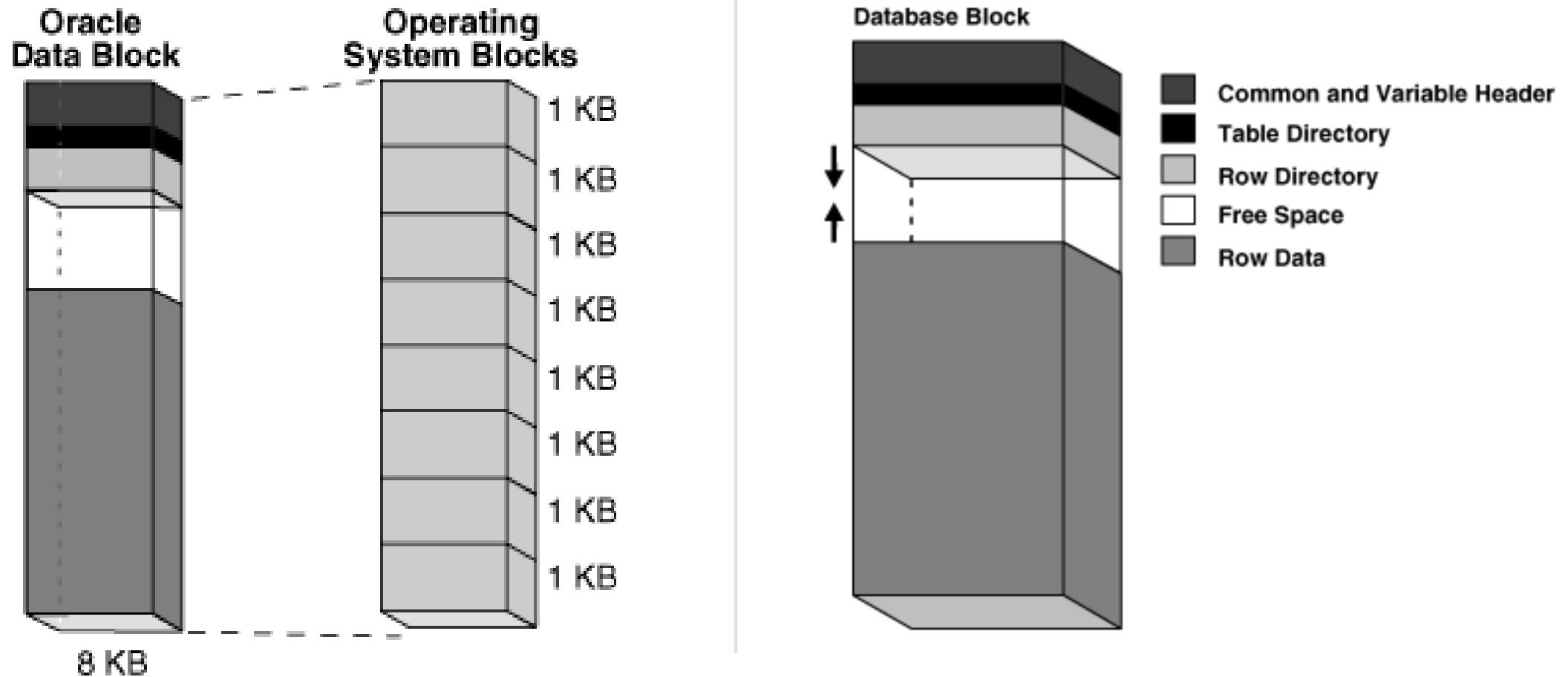
Estruturas Físicas e Lógicas



Estruturas Lógicas de um Tablespace

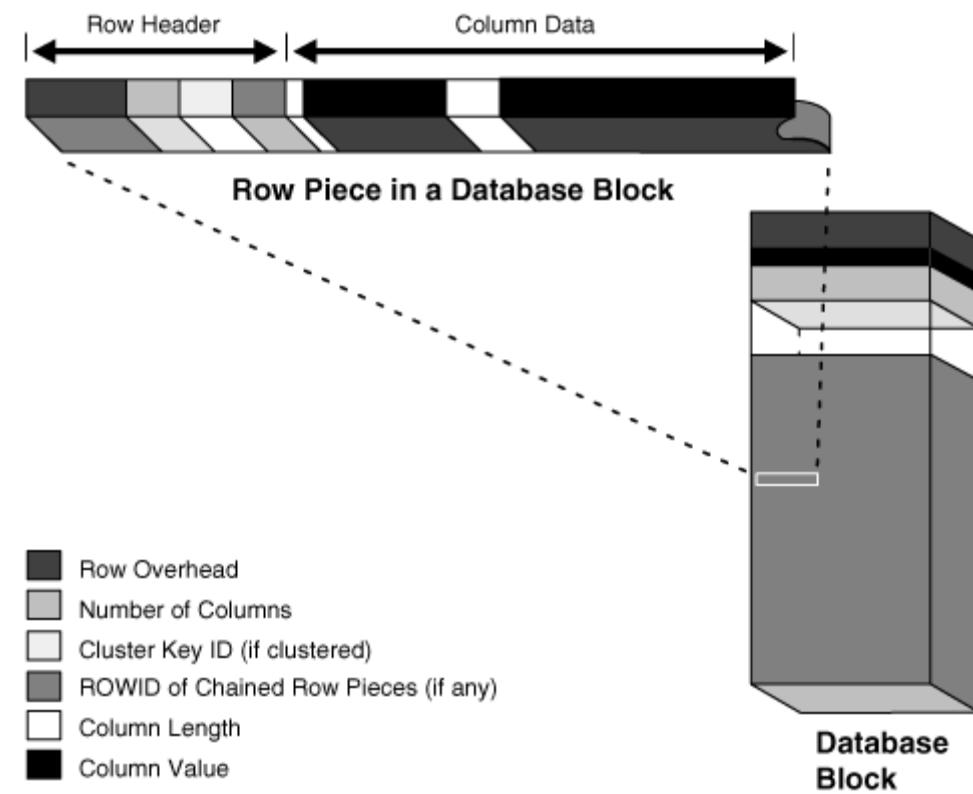
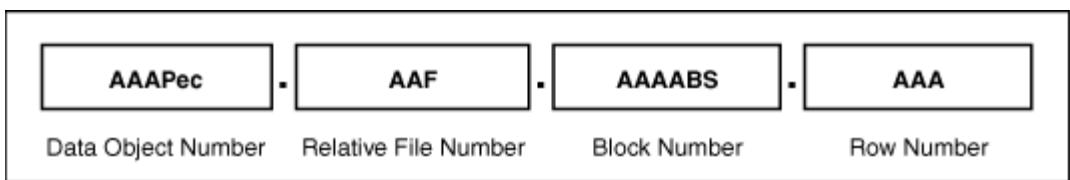


Blocos: Sistema Operacional x Banco de Dados

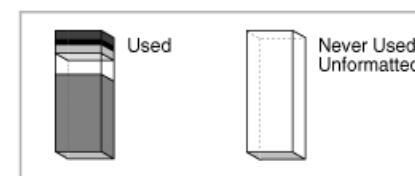
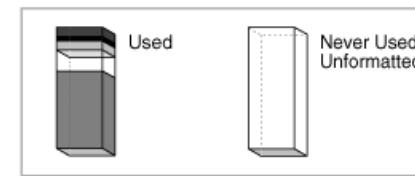
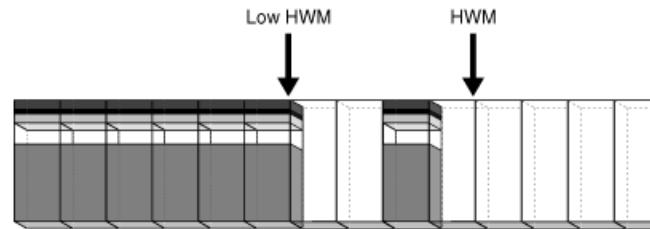
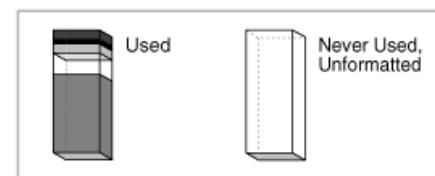
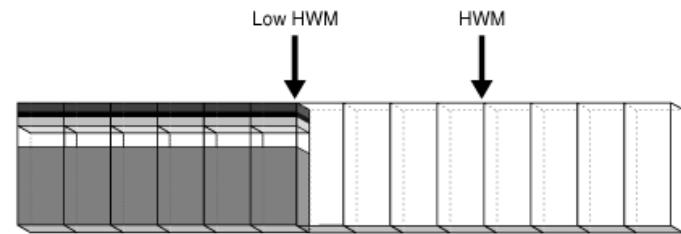
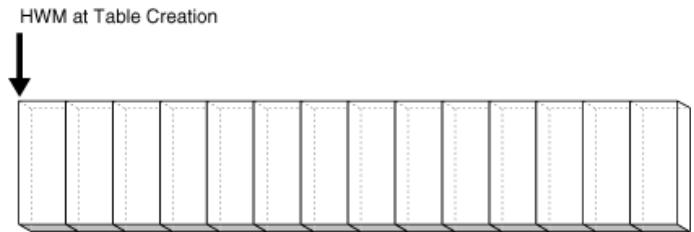


Estrutura de Uma Linha

ROWID: Identificador Único da Linha



High Water Mark

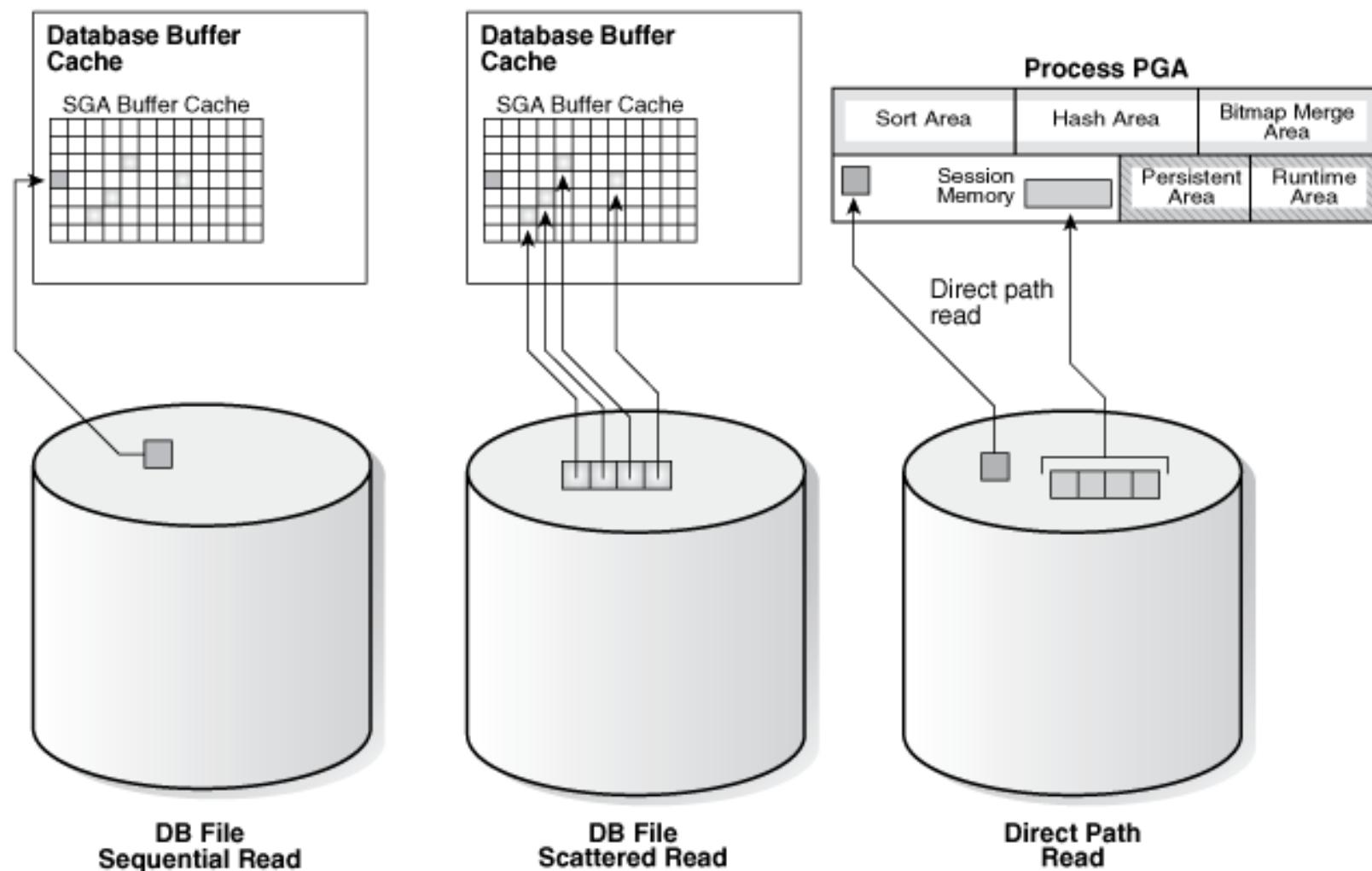


Tipos de Inserção

- INSERT convencional: procura blocos livres abaixo da HWM, pode elevar a HWM caso não haja espaço suficiente disponível.
- Direct-Path INSERT: insere as linhas diretamente acima da HWM, sem se preocupar em buscar blocos com espaço livre.
 - Melhor performance, melhor compactação (quando ativado)
 - Pode desperdiçar espaço
 - Controlado por HINTS: APPEND ou APPEND_VALUES

```
INSERT /*+ APPEND */ INTO sales_hist SELECT * FROM sales WHERE cust_id=8890;  
  
FORALL i IN 1..numrecords  
    INSERT /*+ APPEND_VALUES */ INTO orderdata  
        VALUES (ordernum(i), custid(i), orderdate(i), shipmode(i), paymentid(i));  
COMMIT;
```

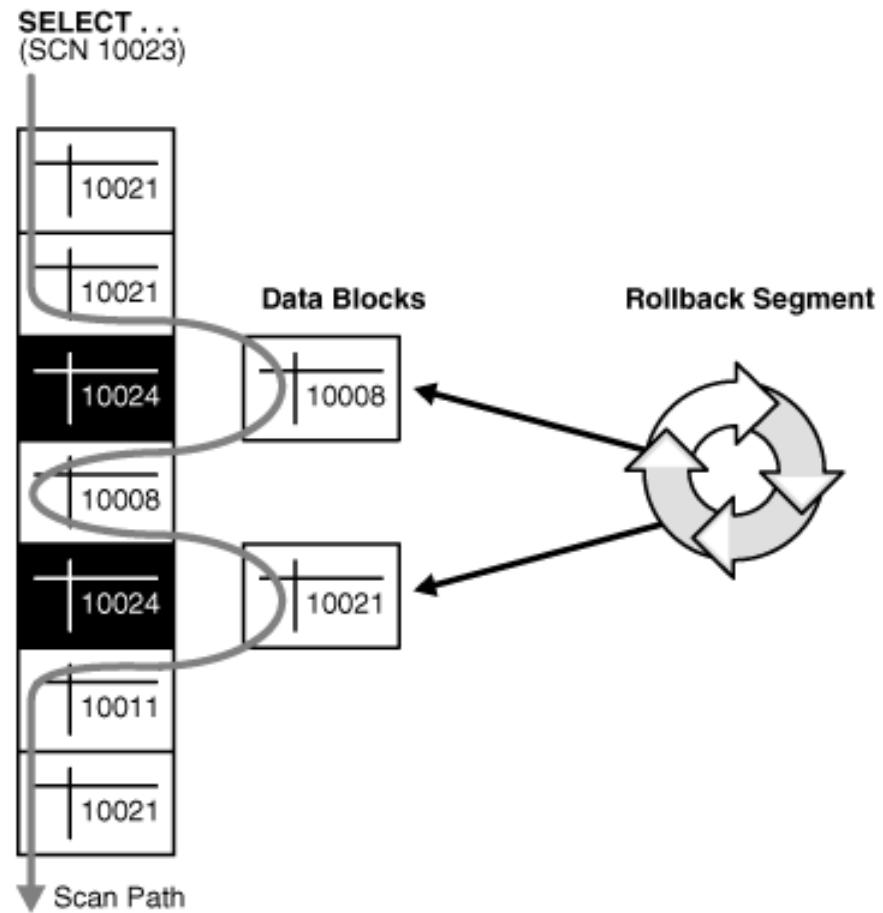
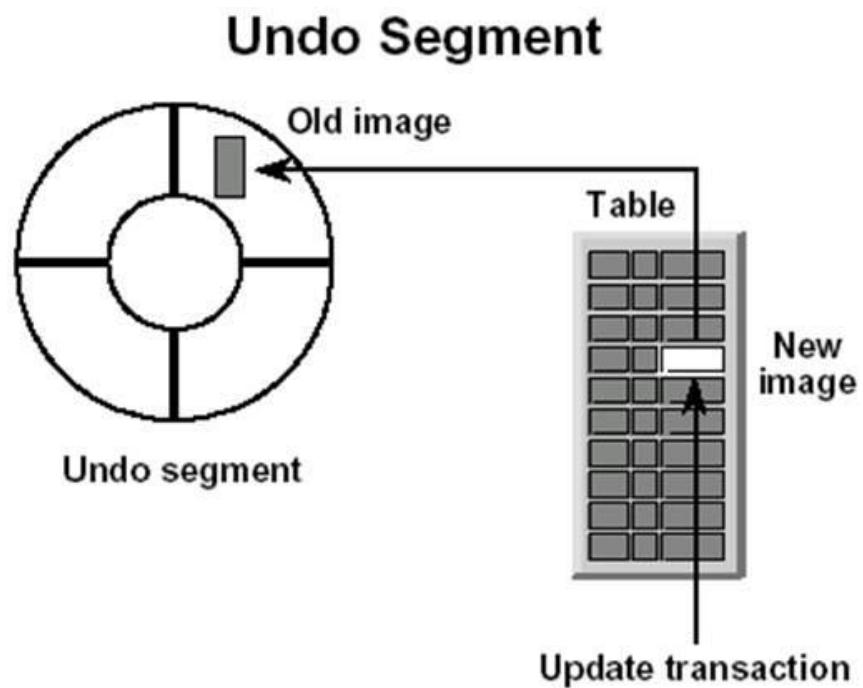
Tipos de Leitura



Transações

- O System Change Number (SCN) é um identificador numérico que indica o momento em que uma operação (transação) ocorreu no banco de dados.
 - Indica um ponto no tempo, usado para leituras consistentes e recuperação de backups
 - Pode ser convertido para um timestamp: SCN_TO_TIMESTAMP
 - Pseudo-coluna ORA_ROWSCN
- Relembrando: as transações ocorrem em memória e são descarregadas de modo assíncrono pelo DBWn para o disco.
- Leituras consistentes dependem da reconstrução do bloco a partir da área de UNDO (*rollback segment*).

Transações



Prática 03b: Estruturas de Disco

- Objetivos: explorar as estruturas de disco, físicas e lógicas, e os seus relacionamentos.
- Views
 - dba_segments (all_segments, user_segments)
 - v\$tablespace, v\$datafile
- HWM e *direct-path insert*
- Transações, SCN e leituras consistentes
 - ORA_ROWSCN
 - *Flashback Query*

Performance Tuning com Oracle 12c

Aula 04: Buffer Cache

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- Estatísticas de Sessão
- Tipos de I/O
- Cache Hit Ratio

Estatísticas

- Base de todo o diagnóstico de performance em Oracle
- Estatísticas podem ser “sistêmicas” ou de “sessão”
- Sistêmicas são utilizadas pelo otimizador para tomada de decisões:
 - Hardware (CPU, disco, memória, rede)
 - Objetos (Tabelas, Índices, etc)
- Estatísticas de sessão tem finalidade diagnóstica
 - Reads (I/O), wait events, context switches, parsing, etc.

Tipos de I/O

- Physical I/O (PIO): acesso no disco
- Logical I/O (LIO): acesso na memória – buffer cache
 - Também chamada de “buffer I/O”

Logical I/O

- Consistent Read / Get (CR)
 - Leitura do bloco direto da memória (buffer cache)
 - Pode envolver reconstrução do bloco a partir do UNDO
- Current Get / DB Block Get
 - Leitura do bloco mais recente direto da memória
 - Geralmente em processos de modificação do bloco (atualiza a última cópia)

Physical I/O

- Leitura de data files para obter dados de índices e tabelas.
 - Lê primeiro para o buffer cache e depois faz um consistent get (LIO)
- Leitura direta (direct read) do tablespace temporário. Ocorre quando o Oracle usa o tablespace como área de swap – ex.: sort area ou hash area muito pequenos.
 - Direct read: physical I/O sem logical I/O

LIO x PIO

- “The difference between PIO and LIO time is not the difference between disk access and memory access, it’s much smaller (37x vs 10,000x as per their measurement), and therefore performance optimization needs to address LIO as well as physical I/O (PIO)” – Cary Millsap et al.
 - [https://orlandoolguin.files.wordpress.com/2010/03/cary millsap why you should focus on lios instead of pios.pdf](https://orlandoolguin.files.wordpress.com/2010/03/cary_millsap_why_you_should_focus_on_lios_instead_of_pios.pdf)
- Um artigo mais recente mostrou que o LIO é 8x mais rápido (em média) que um PIO
 - <https://savvinov.com/2015/02/17/logical-io/>
- Resumindo: o melhor I/O é aquele que você **não faz** ☺

Cache Hit Ratio

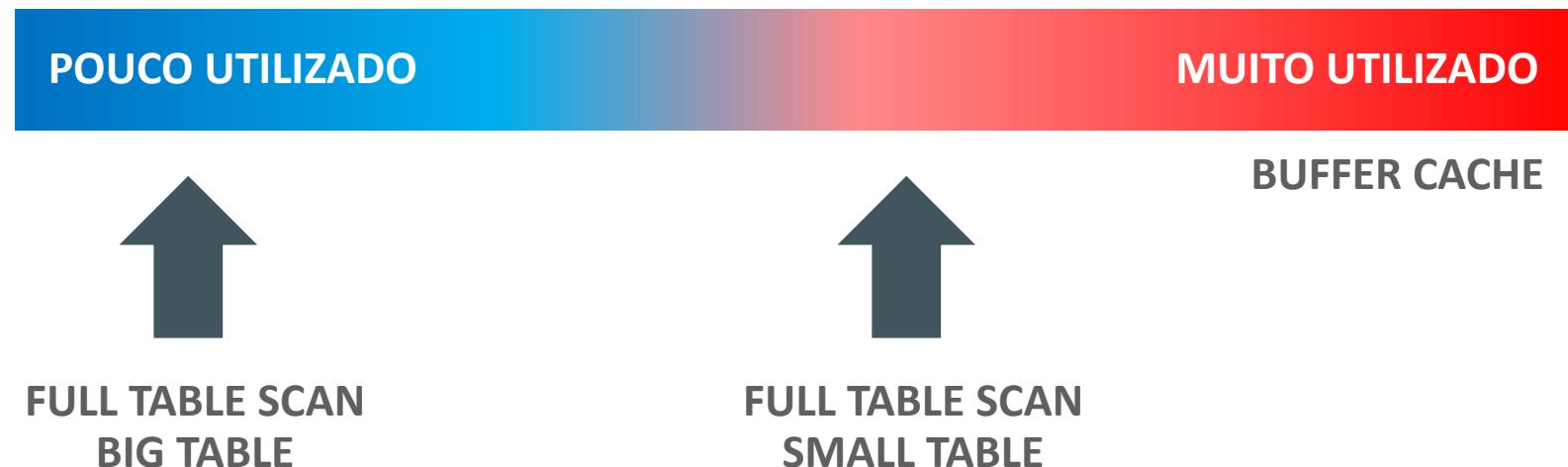
- # LIO >> # PIO
- Buscar um alto **Cache Hit Ratio**, porém:
 - Full Table Scans vão diminuí-lo, mas podem ser necessários
 - Ele pode ser artificialmente elevado por leituras repetidas de blocos desnecessariamente
- Fórmula: $1 - (\text{physical_reads} / (\text{consistent_gets} + \text{block_gets}))$

Fatores que Influenciam o Cache Hit Ratio

- Evitar ler dados repetidas vezes realizando o processo em uma única passada
- Evitar repetidas leituras fazendo cache na camada de aplicação (cliente ou mid-tier)
- Em aplicações OLTP muitas linhas são acessadas apenas uma vez (ou nunca)
- Não há necessidade de aumentar o buffer cache se o database realiza muitos full table scans (não tem influencia!)
- Full table scans diminuem naturalmente o Cache Hit Ratio, mas nem sempre isso é um problema

Buffer Cache e LRU

- Small Table Threshold (STT): aprox. 2% do buffer cache
 - Tabela maior que o STT: FTS vai usar **db_file_multiblock_read_count** blocos no fim da LRU (Pouco Utilizado). Este é um *table scan (long table)*
 - Tabela menor que o STT: ela vai ser lida para o **meio** do cache, mas sem registrar o Touch Count (x\$bh.tch). Este é um *table scan (short table)*



Prática 04: Buffer Cache

- Objetivo: aprofundar o entendimento sobre o funcionamento do Buffer Cache.
- Views:
 - v\$mystat, v\$sesstat, v\$sysstat
- Calcular o Cache Hit Ratio e observar o impacto de diferentes querys
- Mecanismo de LRU do Buffer Cache
 - v\$bh
 - x\$bh e “touch count” (TCH)

Performance Tuning com Oracle 12c

Aula 05: Metodologia de *Tuning*

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- Metodologia Oracle de *Tuning*
- Padrões de Codificação
- SQL Dinâmico

Performance

- O que é performance?
 - Conceito relativo, cuja principal métrica é a **percepção do usuário**
- Geralmente se resume a três pontos:
 - CPU
 - I/O
 - Memória
- Meta: busca de gargalos – achar o “culpado”
 - Fazer mais com menos, da melhor maneira possível
- Cuidado: o processo de tuning é infinito! (sempre tem como melhorar)

Exemplos de Querys Ineficientes

1

```
SELECT COUNT(*) FROM products p
WHERE prod_list_price <
    1.15 * (SELECT avg(unit_cost) FROM costs c
            WHERE c.prod_id = p.prod_id)
```

2

```
SELECT * FROM job_history jh, employees e
WHERE substr(to_char(e.employee_id),2) =
substr(to_char(jh.employee_id),2)
```

3

```
SELECT * FROM orders WHERE order_id_char = 1205
```

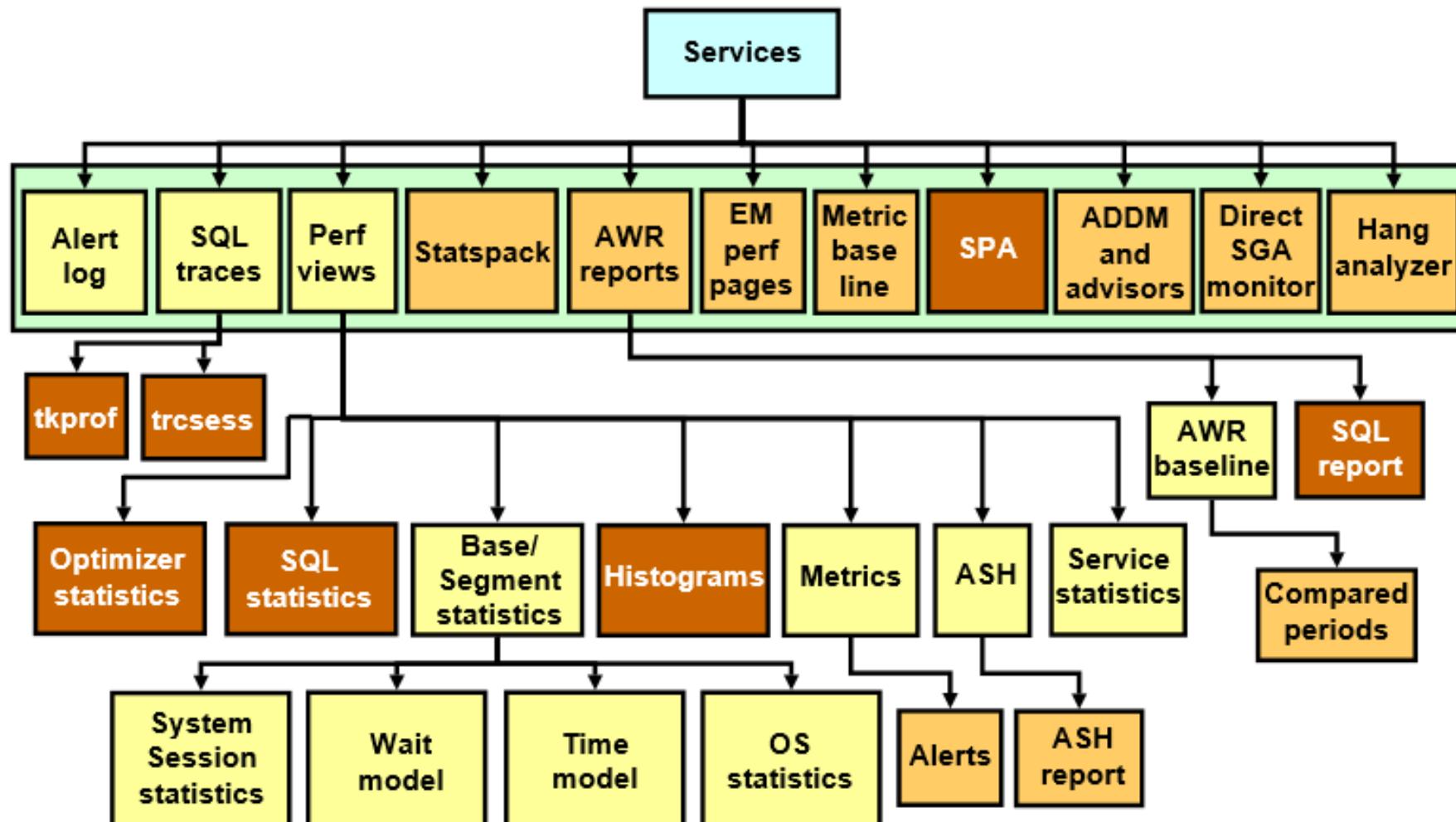
4

```
SELECT * FROM employees
WHERE to_char(salary) = :sal
```

5

```
SELECT * FROM parts_old
UNION
SELECT * FROM parts_new
```

Como Diagnosticar Problemas de Performance?



Pontos Críticos para Performance de SQL

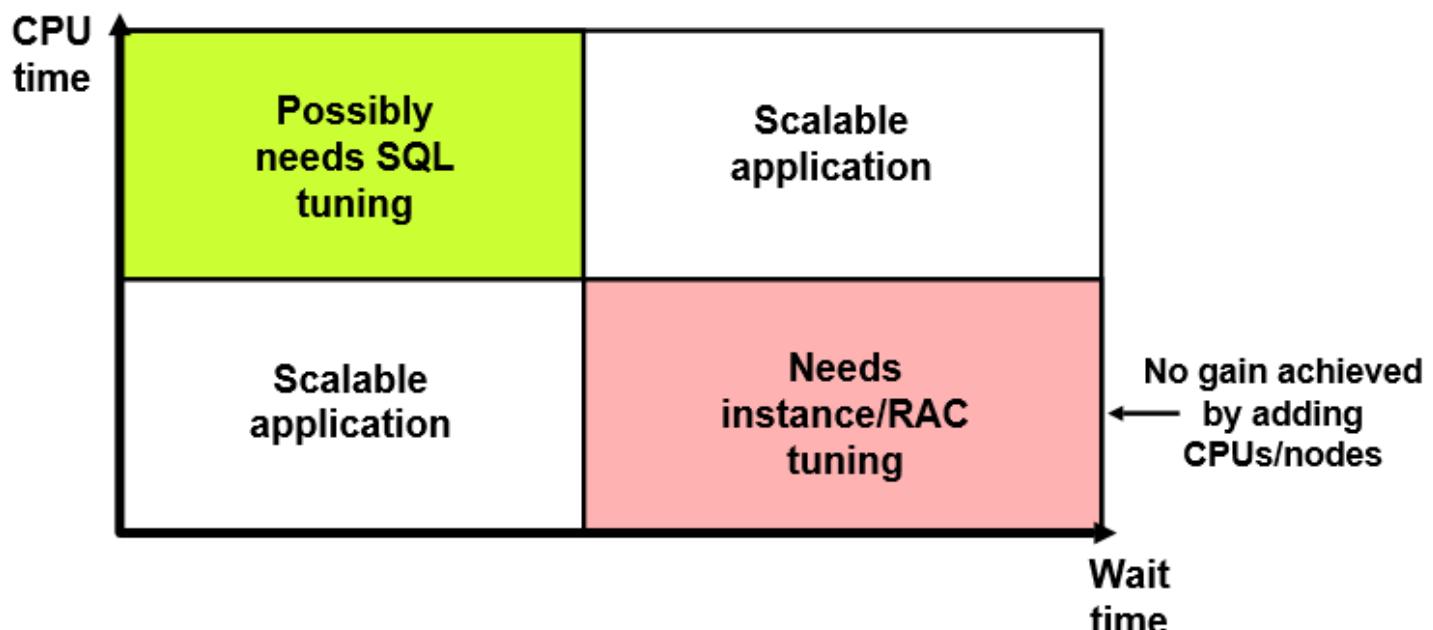
- Estatísticas do otimizador desatualizadas ou ausentes
- Estruturas de acesso inadequadas
 - Índices, Materialized Views, Partições
- Seleção do plano de execução inadequado
 - Estimativas incorretas: custo, cardinalidade e seletividade
- SQL mal escrito
- *Hardware* Sub-dimensionado ou Obsoleto

Principais Atividades no Processo de Tuning

- Identificar SQLs “pesados”
- Coletar estatísticas
- Gerar estatísticas de sistema
- Otimizar índices (rebuild, compress)
- Manutenção de planos de execução
- Criar novas estratégias de indexação
- **Remover índices**

Melhores Práticas para Escalabilidade

- Escalabilidade é a habilidade de um sistema em processar mais carga com um aumento proporcional do uso de recursos do sistema.
- Foco no desenho da aplicação:
 - Schema: SQL “pesado”
 - Transações: locks e contenções
 - Conexão: tempos de resposta



Erros Comuns em Aplicações

1. Gerenciamento de conexão ineficiente
2. Mal uso dos cursors e da shared pool
3. Excesso de SQLs que consomem muitos recursos
4. Uso de parâmetros de inicialização fora do padrão
5. Configuração de discos inadequada
6. Problemas na configuração do redo log
7. Serialização excessiva
8. *Full table scans* desnecessários

Tuning Pró-Ativo

- Começa pelo desenho da aplicação:
 - Design simples
 - Modelo de Dados
 - Tabelas, Índices e Views
- Escrever SQL da maneira correta
- Compartilhar cursores
- Usar *bind variables*

Desenho da Aplicação e do Modelo de Dados

- Aplicação
 - Mantenha o design simples
 - SQL bem-escrito
 - Indexar apenas quando necessário
 - Recuperar apenas informações relevantes (sem **select *** !)
- Modelo:
 - Objetivo: representar as regras de negócio
 - Foco nas transações mais importantes
 - Usar ferramentas de modelagem
 - Normalizar os dados de acordo com o tipo de aplicação (OLTP vs DW)

Tabelas, Índices e Views

- Flexibilidade x Performance:
 - Normalizar sempre que possível
 - De-normalizar quando necessário
- Foco em objetos críticos para o negócio
 - Sempre há oportunidades para tuning! Quando parar?
- Criar índices para PKs e FKs
- Usar o SQL como guia para criar índices
 - Dados que são frequentemente acessados (lista select)
- Views simplificam o desenho e facilitam a reusabilidade
 - Mas podem piorar a performance!!! (*views aninhadas*)

Escrevendo SQL Eficiente

- Boa estratégia de conexão no banco
 - Fazer o máximo possível em um menor número de tarefas
 - Pensar em “Teoria de Conjuntos”
- Minimizar o “*Parsing*”
- Compartilhar cursores
- Usar *bind variables*

Como Facilitar o Compartilhamento de Cursos

- Criar código genérico:
 - Packages
 - Stored Procedures
 - Functions
 - Triggers
- Padrão de codificação
 - Melhora a leitura
 - Capitalização, espaço em branco, binds
 - O Oracle faz comparação exata
 - **SELECT * FROM TABELA** é diferente de **select * from tabela**
 - **Cuidado com EXECUTE IMMEDIATE**

Solução de Problemas de Performance

- Verificar o uso de recursos pelas *querys*
 - Estatísticas de sessão (*query* específica)
 - SQL Trace (*query* ou processo específico)
 - Enterprise Manager (*tuning* de instância)
- Validar conexões pelo *middleware* / cliente
 - Dedicated vs Shared Server
 - Reuso de conexões
- Verificar compartilhamento de cursores
 - Parâmetro de inicialização CURSOR_SHARING
- Verificar validade e disponibilidade de estatísticas para o otimizador

SQL Dinâmico

- SQL construído em tempo de execução para realização de tarefas complexas.
- Três opções:
 - DBMS_SQL
 - EXECUTE IMMEDIATE
 - OPEN FOR, FETCH, CLOSE
- Todas as opções permitem o uso de *bind variables*. Exemplos:
 - DBMS_SQL.BIND_VARIABLE
 - EXECUTE IMMEDIATE v_sql USING var1, var2, ...;
 - OPEN v_cursor FOR v_sql USING var1, var2, ...;

Prática 05: Padrões de Codificação

- Objetivos: observar o impacto de pequenas modificações na forma como o banco interpreta as consultas.
- Parsing Soft x Hard
 - “select * from tabela” ≠ “select * from tabela”?
- Impactos de usar:
 - Espaços em branco, capitalização
 - Parâmetros literais
- SQL Dinâmico
 - EXECUTE IMMEDIATE
 - OPEN FOR

Performance Tuning com Oracle 12c

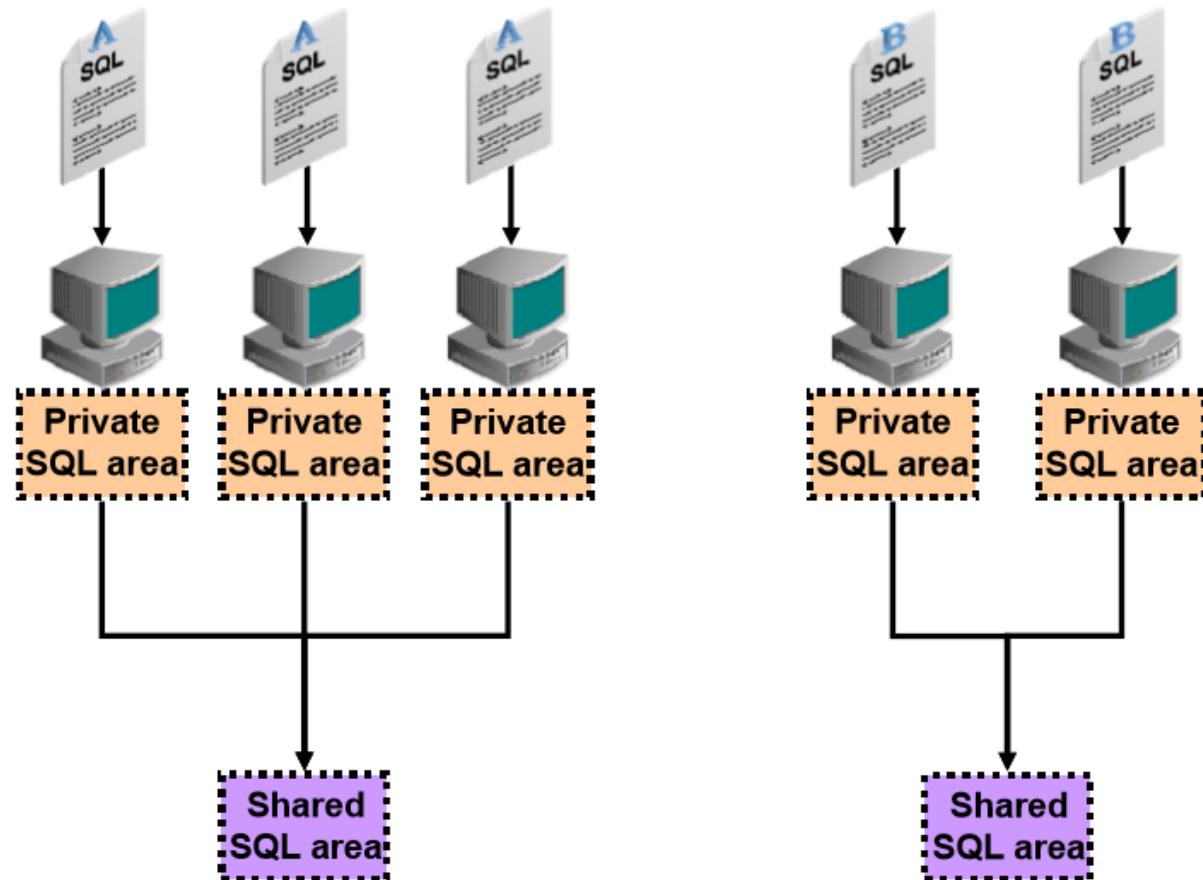
Aula 06: SQL e Otimização

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

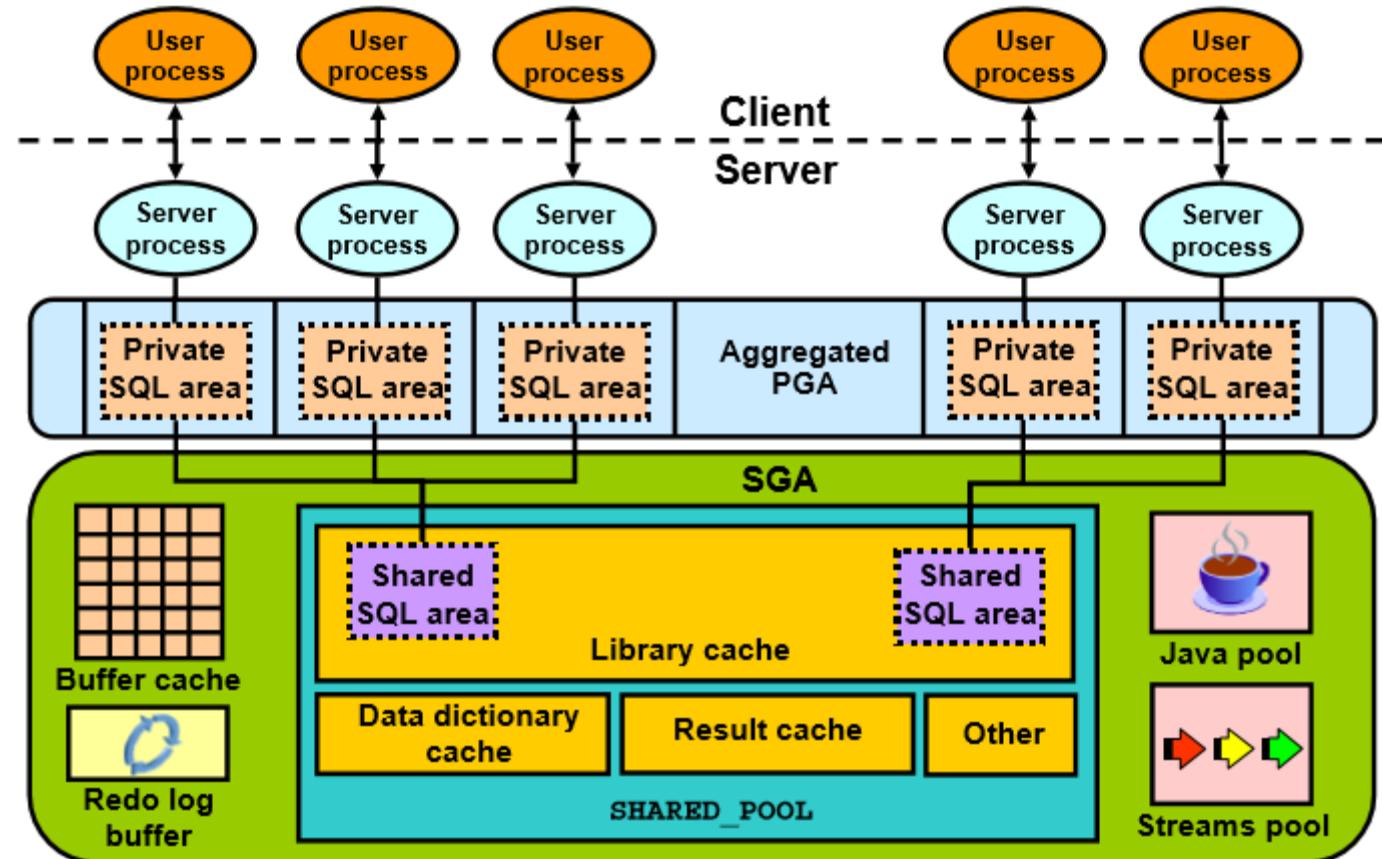
Objetivos

- Etapas do Processamento SQL
- Otimizador
- Tipos de Parse
- Otimização Baseada em Custo

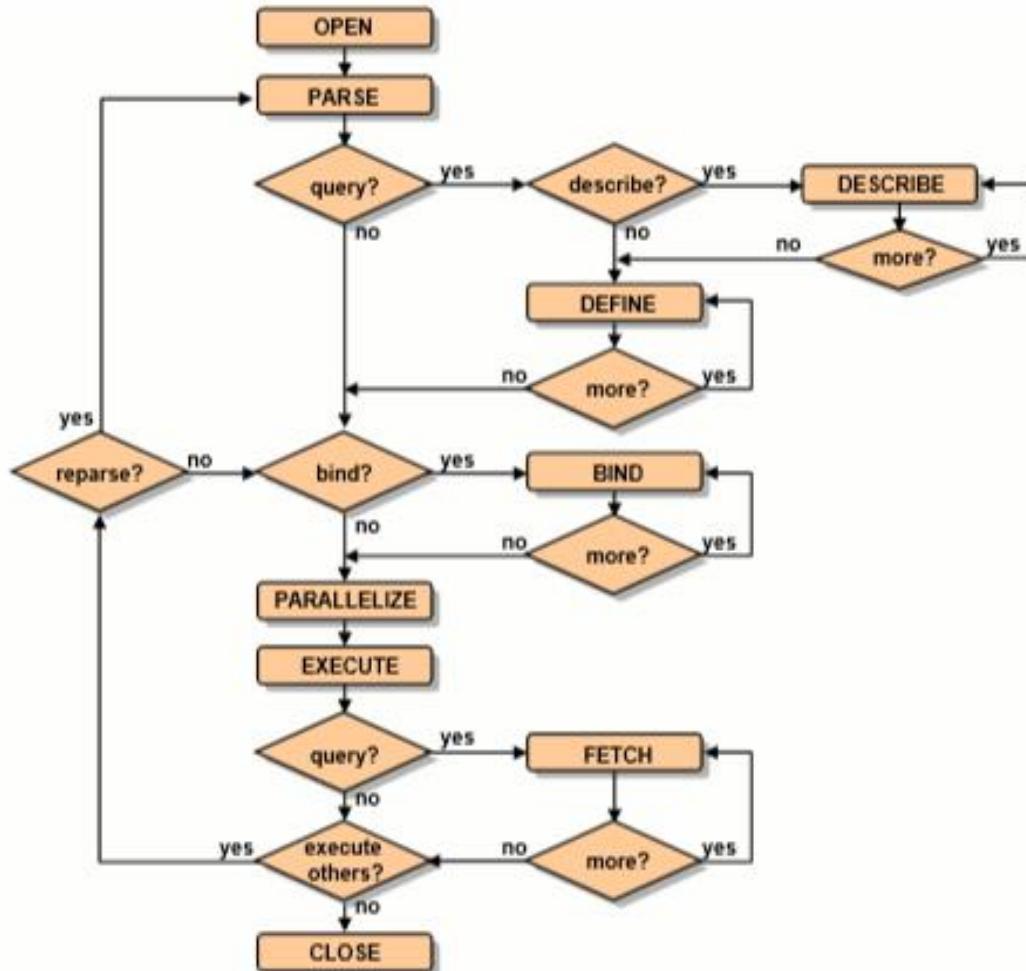
Como o SQL é Processado?



Como o SQL é Processado?



Etapas do Processamento SQL



1. Criar o cursor (*open*)
2. Analisar o comando (*parse*)
3. Descrever os resultados (*describe*)
4. Definir a saída (*define*)
5. Vincular variáveis (*bind*)
6. Paralelizar (*parallelize*)
7. Executar (*execute*)
8. Adquirir (*fetch*)
9. Fechar (*close*)

Etapas do Processamento SQL

- Passo 1: *Open* (Cria o Cursor)
 - Cursor é um identificador para uma SQL Area privada
- Passo 2: *Parse* (“Análise”)
 - Envia o comando para o processo do servidor (*server process*)
 - Cria uma representação interna do SQL e transfere-a para a Shared SQL Area
 - Pode ser reutilizado se existir outro SQL idêntico (*soft parse*)
- Passo 3: *Describe* (“Descrever”)
 - Cria/popula a lista de seleção (colunas do *select*)
- Passo 4: *Define* (“Definir”)
 - Aloca as estruturas necessárias para a leitura de dados (saída) da operação

Etapas do Processamento SQL

- Passo 5: *Bind* (“Vincular”)
 - Associa valores das variáveis
- Passo 6: *Parallelize* (“Paralelizar”)
 - Válido para: select, insert, update, merge, delete, create, alter
- Passo 7: *Execute* (“Executar”)
 - Prepara as estruturas para leitura
- Passo 8: *Fetch* (“Adquirir”)
 - Realiza a seleção/leitura para as estruturas de saída
- Passo 9: *Close* (“Fechar”)

Exemplo de Processamento Manual

```
declare
    cur1 number;
    num1 number := 10;
    num2 number := 0;
    ret1 number;
begin
    cur1 := dbms_sql.open_cursor;

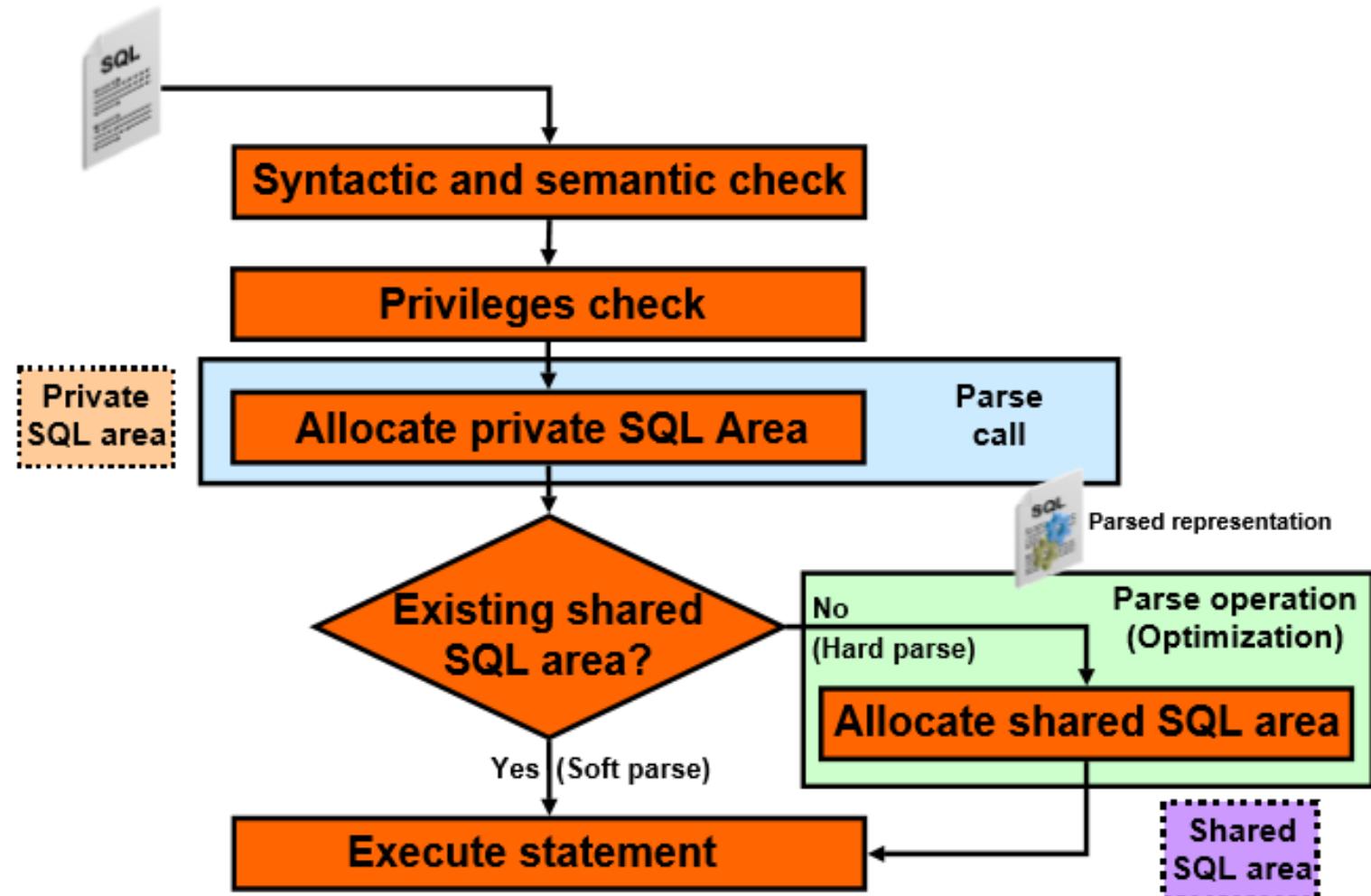
    dbms_sql.parse(cur1,
                    'select rownum n from dual connect by level <= :num',
                    dbms_sql.native);

    dbms_sql.bind_variable(cur1, ':num', num1);

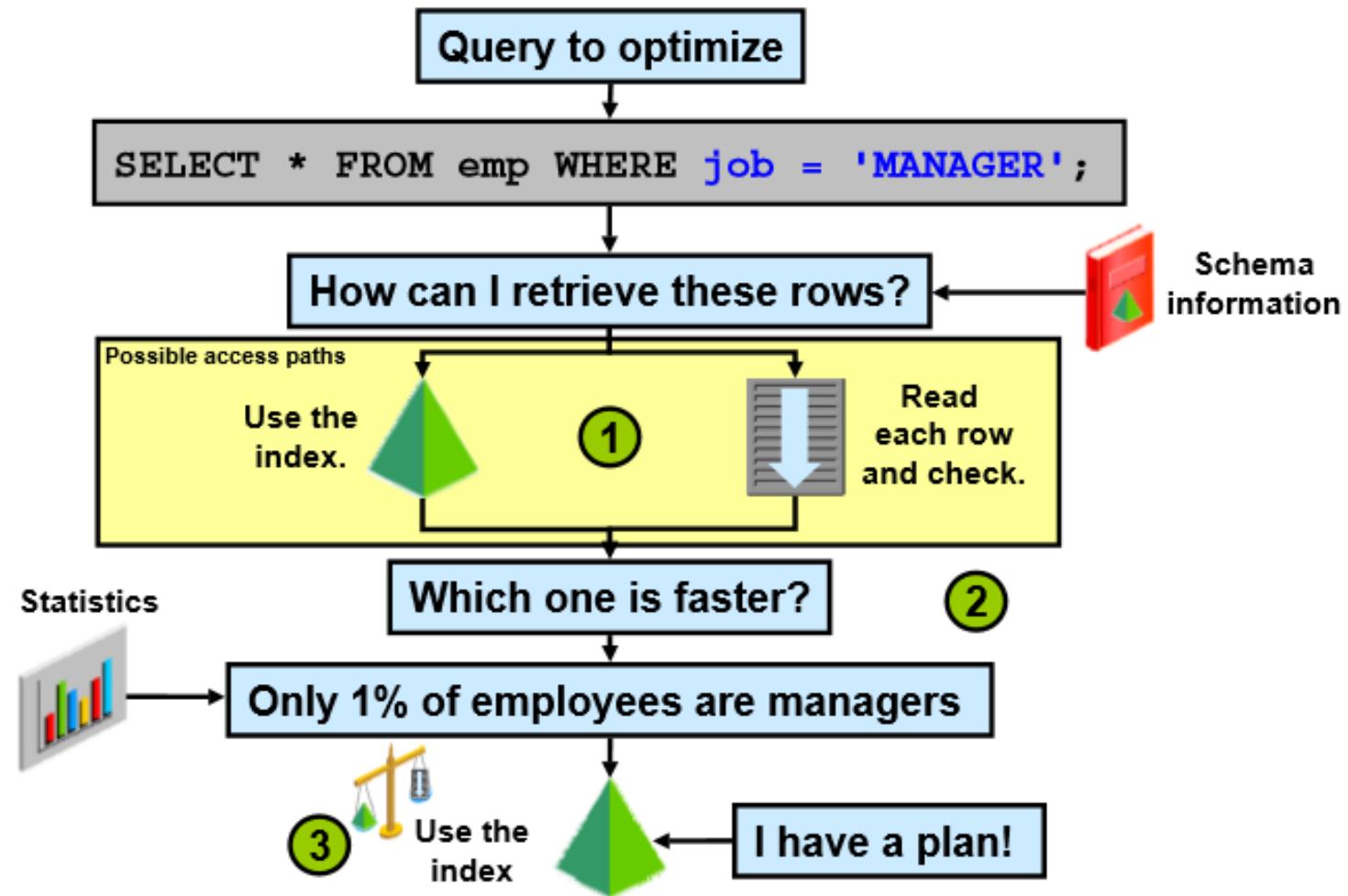
    ret1 := dbms_sql.execute(cur1);
loop
    exit when dbms_sql.fetch_rows(cur1) = 0;
    -- processa alguma coisa
    num2 := num2 + 1;
end loop;
dbms_sql.close_cursor(cur1);

    dbms_output.put_line('num2 = ' || num2);
end;
/
```

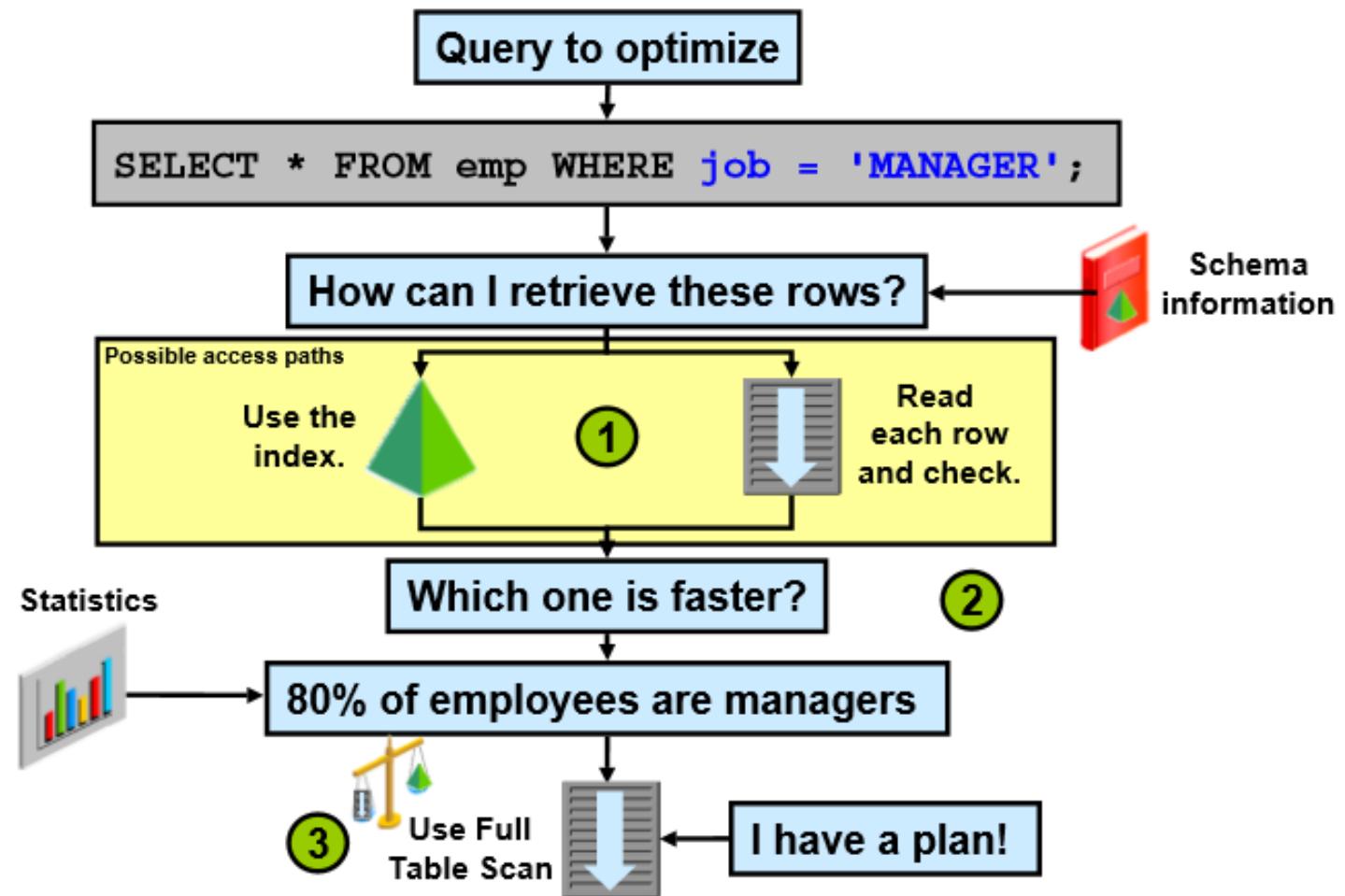
Parsing: Visão Geral



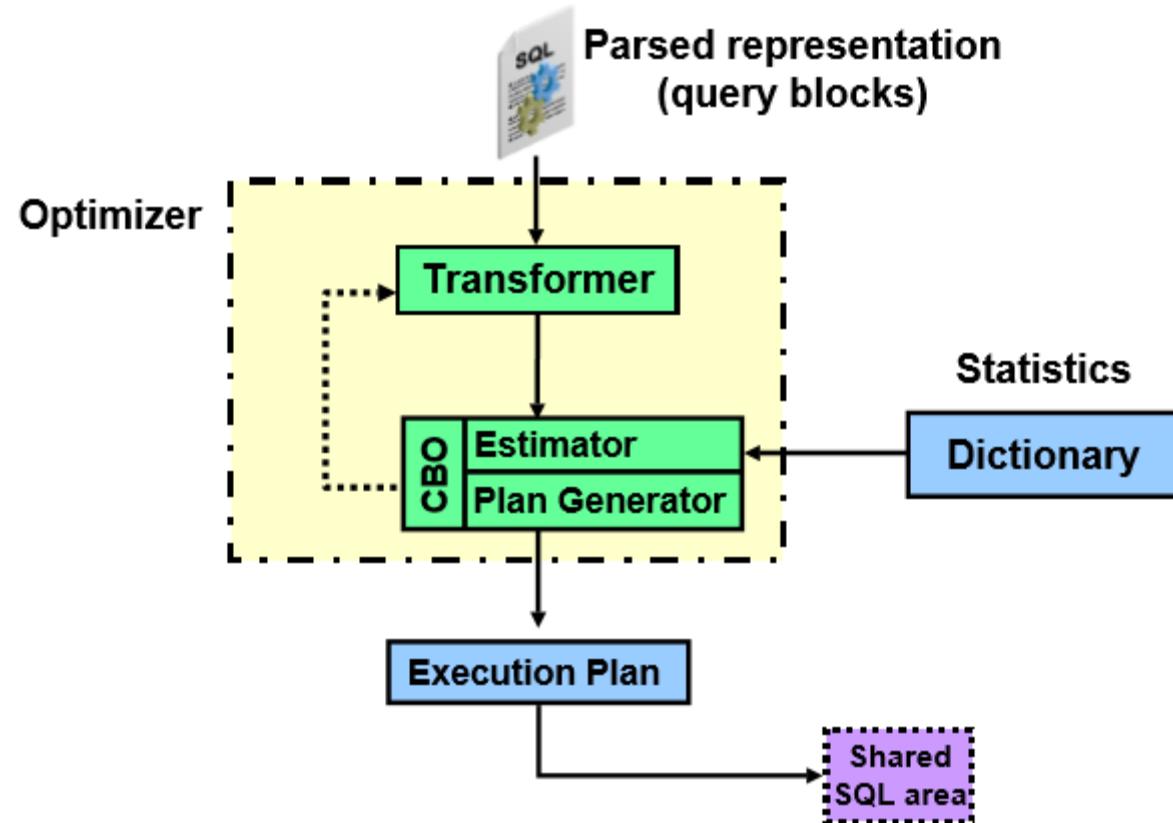
Por Que Precisamos do Otimizador?



Por Que Precisamos do Otimizador?

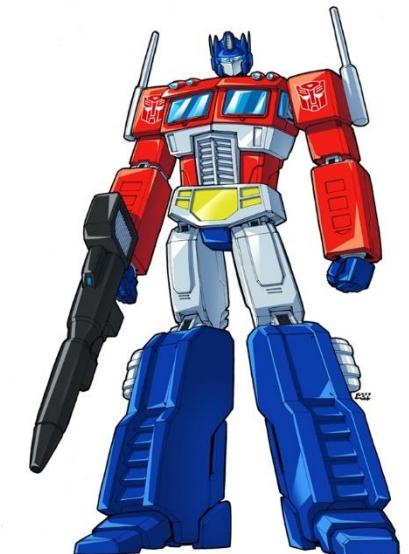


Otimização Durante o Hard Parse



Transformer

- Reescreve consultas.
- Exemplos de operações:
 - OR Expansion
 - Subquery Unnesting
 - View Merge
 - Predicate Push Down
 - Transitivity
 - Materialized View Query Rewrite
 - In-memory Aggregation



Transformer: OR Expansion

```
CREATE INDEX sales_prod_promo_ind
  ON sales(prod_id, promo_id);
```

```
SELECT *
  FROM sales
 WHERE promo_id=33
   OR prod_id=136;
```

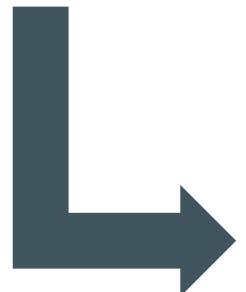


```
SELECT *
  FROM sales
 WHERE prod_id=136
UNION ALL
SELECT *
  FROM sales
 WHERE promo_id=33
 AND LNNVL(prod_id=136);
```

Id	Operation	Name	Rows
0	SELECT STATEMENT		
1	CONCATENATION		
2	TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED	SALES	710
3	INDEX RANGE SCAN	SALES_PROD_PROMO_IND	710
4	PARTITION RANGE ALL		229K
5	TABLE ACCESS FULL	SALES	229K

Transformer: Subquery Unnesting

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



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

Transformer: View Merge

```

SELECT e.first_name, e.last_name, dept_locs_v.street_address,
       dept_locs_v.postal_code
  FROM employees e,
       ( SELECT d.department_id, d.department_name,
                  l.street_address, l.postal_code
            FROM departments d, locations l
           WHERE d.location_id = l.location_id ) dept_locs_v
 WHERE dept_locs_v.department_id = e.department_id
   AND e.last_name = 'Smith';
  
```



```

SELECT e.first_name, e.last_name, l.street_address, l.postal_code
  FROM employees e, departments d, locations l
 WHERE d.location_id = l.location_id
   AND d.department_id = e.department_id
   AND e.last_name = 'Smith';
  
```

			Name	Cost (%CPU)
	Id	Operation		
	0	SELECT STATEMENT		7 (15)
*	1	HASH JOIN		7 (15)
	2	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	2 (0)
*	3	INDEX RANGE SCAN	EMP_NAME_IX	1 (0)
	4	VIEW		5 (20)
*	5	HASH JOIN		5 (20)
	6	TABLE ACCESS FULL	LOCATIONS	2 (0)
	7	TABLE ACCESS FULL	DEPARTMENTS	2 (0)

			Name	Cost (%CPU)
	Id	Operation		
	0	SELECT STATEMENT		4 (0)
	1	NESTED LOOPS		
	2	NESTED LOOPS		4 (0)
	3	NESTED LOOPS		3 (0)
	4	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	2 (0)
*	5	INDEX RANGE SCAN	EMP_NAME_IX	1 (0)
	6	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1 (0)
*	7	INDEX UNIQUE SCAN	DEPT_ID_PK	0 (0)
*	8	INDEX UNIQUE SCAN	LOC_ID_PK	0 (0)
	9	TABLE ACCESS BY INDEX ROWID	LOCATIONS	1 (0)

Transformer: Predicate Pushing

1) Definição da View:

```
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 );
```

2) Consulta:

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

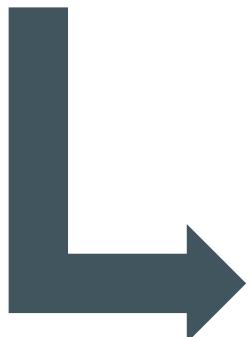


3) Consulta Transformada:

```
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 );
```

Transformer: Transitivity

```
SELECT *
  FROM hr.employees  emp,
       hr.departments dept
 WHERE emp.department_id = 20
   AND emp.department_id = dept.department_id;
```



```
SELECT *
  FROM hr.employees  emp,
       hr.departments dept
 WHERE emp.department_id = 20
   AND emp.department_id = dept.department_id
   AND dept.department_id = 20;
```

Otimização Baseada em Custo

- Cost-Based Optimizer (CBO)
- Componentes:
 - Estimator
 - Plan Generator
- Estimator: faz a estimativa de custo da operação
- Plan Generator: testa diferentes técnicas de otimização para gerar vários caminhos distintos
 - Utiliza o estimator para atribuir um custo a cada um dos planos
 - Escolhe o de menor custo
 - **Limitado por tempo**

Estimativas: Seletividade

- *Seletividade = $\frac{\text{Número de linhas que satisfazem a condição}}{\text{Total de Linhas}}$*
- É a proporção estimada entre as linhas selecionadas por um ou mais predicados (condições) e o total de linhas
- Pode assumir os valores de 0.0 a 1.0
 - Alta seletividade: pequena proporção de linhas
 - Baixa seletividade: grande proporção de linhas
- Cálculo:
 - Estatísticas da tabela. Se não tiver: *Dynamic Sampling*
 - Se não houver histogramas considera distribuição homogênea

Cardinalidade

- *Cardinalidade = Seletividade * Total de Linhas*
- Número de linhas esperado ao executar uma operação do plano de execução
- Utilizado para determinar o custo de *joins*, filtros e ordenações

Custo

- Uma forma padronizada de comparar querys
 - Estimativa de I/Os necessários para executar um procedimento
 - Unidade de custo = 1 SRds (single block random read)
- Fórmula do Custo
 - $$Custo = \frac{\#SRds*sreadtim + \#MRds*mreadtim + \#CPUCycles/cpuspeed}{sreadtim}$$
- Onde:
 - #SRds: Número de single block reads / #MRds: Número de multiblock reads
 - #CPUCycles: Número de ciclos de CPU
 - sreadtim: tempo de leitura de um bloco (single block read time)
 - mreadtim: tempo de leitura multibloco (multiblock read time)
 - cpuspeed: milhões de instruções por segundo

Gerador de Planos de Execução

- Plan Generator
 - Faz diversas simulações até chegar na opção de menor custo
 - Testa transformações, tipos de acesso, permutações de join, etc...

```

Join order[1]: DEPARTMENTS [D] #0 EMPLOYEES [E] #1
NL Join: Cost: 41.13 Resp: 41.13 Degree: 1
SM cost: 8.01
HA cost: 6.51
Best:: JoinMethod: Hash
Cost: 6.51 Degree: 1 Resp: 6.51 Card: 106.00
Join order[2]: EMPLOYEES [E] #1 DEPARTMENTS [D] #0
NL Join: Cost: 121.24 Resp: 121.24 Degree: 1
SM cost: 8.01
HA cost: 6.51
Join order aborted
Final cost for query block SEL$1 (#0)
All Rows Plan:
Best join order: 1
+-----+
| Id | Operation           | Name      | Rows | Bytes | Cost |
+-----+
| 0  | SELECT STATEMENT    |           |       |        | 7     |
| 1  | HASH JOIN            |           | 106  | 6042  | 7     |
| 2  | TABLE ACCESS FULL   | DEPARTMENTS | 27   | 810   | 3     |
| 3  | TABLE ACCESS FULL   | EMPLOYEES   | 107  | 2889  | 3     |
+-----+

```

Prática 06: SQL e Otimizador

- Processamento SQL Passo a Passo
- Views:
 - Dba_tables
 - Dba_tab_statistics
 - Dba_tab_col_statistics
- Estimativas do Otimizador (Cardinalidade, Seletividade, Custo)
- Gerador de Planos de Execução

Performance Tuning com Oracle 12c

Aula 07: Planos de Execução

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- Definição
- Captura de Planos
- Interpretação
- Ferramentas de Visualização

Plano de Execução

- Combinação de fontes de linhas (*rowsources*) e operadores em forma hierárquica (árvore) indicando as etapas executadas (ou planejadas) para obter um resultado de uma operação.



Onde Encontrar o Plano de Execução?

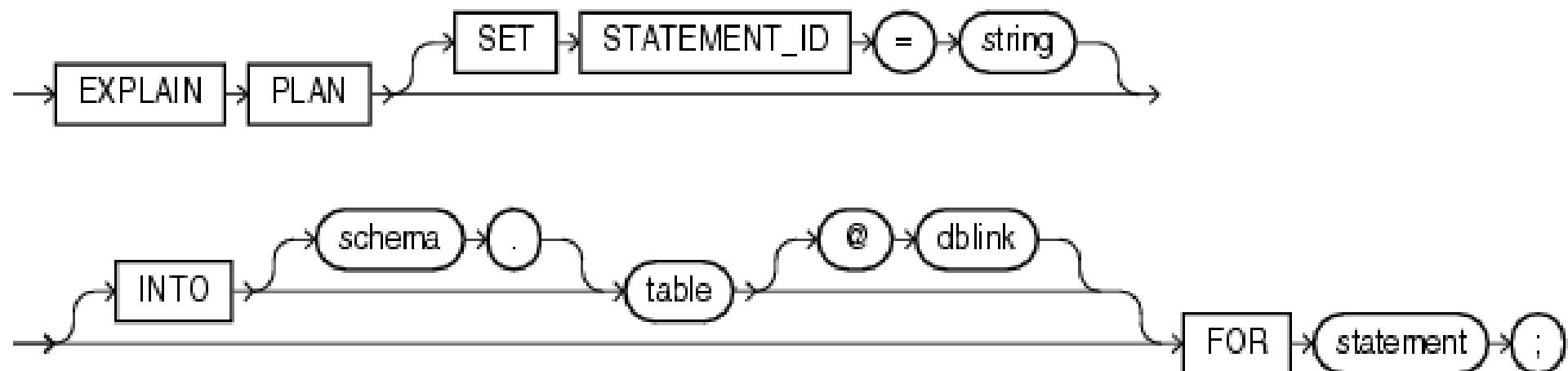
- Views:
 - V\$sql_plan
 - V\$sql_plan_monitor (11g+)
 - Dba_hist_sql_plan (AWR / Diagnostic Pack)
 - Stats\$sql_plan (statspack)
- Comando EXPLAIN PLAN e PLAN_TABLE
- Autotrace (SQL*Plus ou SQL Developer)
- Arquivos de trace
 - Evento 10053, dbms_monitor, etc

Visualizando os Planos

- EXPLAIN PLAN
 - SELECT * FROM PLAN_TABLE
 - DBMS_XPLAN.DISPLAY()
- Autotrace
 - SET AUTOTRACE ON [EXPLAIN]
- DBMS_XPLAN
 - DISPLAY_CURSOR()
 - DISPLAY_AWR()
 - DISPLAY_SQLSET()
 - DISPLAY_SQL_PLAN_BASELINE()

EXPLAIN PLAN

- Gera o plano de execução e grava na PLAN_TABLE
- Não executa o comando (SQL)
- Pode **não ser** o plano real executado



EXPLAIN PLAN

- *statement* pode ser:
 - SELECT
 - INSERT
 - UPDATE
 - DELETE
 - MERGE
 - CREATE TABLE [AS SELECT]
 - CREATE INDEX
 - ALTER INDEX ... REBUILD

PLAN_TABLE

```
EXPLAIN PLAN
  SET STATEMENT_ID = 'Raise in Tokyo'
  INTO plan_table
  FOR UPDATE employees
    SET salary = salary * 1.10
  WHERE department_id =
    (SELECT department_id FROM departments
     WHERE location_id = 1700);
```

```
SELECT id, LPAD(' ',2*(LEVEL-1))||operation operation, options,
       object_name, object_alias, position
  FROM plan_table
 START WITH id = 0 AND statement_id = 'Raise in Tokyo'
 CONNECT BY PRIOR id = parent_id AND statement_id = 'Raise in Tokyo'
 ORDER BY id;
```

ID	OPERATION	OPTIONS	OBJECT_NAME	OBJECT_ALIAS	POSITION
0	UPDATE STATEMENT				4
1	UPDATE		EMPLOYEES		1
2	INDEX	RANGE SCAN	EMP_DEPARTMENT_IX	EMPLOYEES@UPD\$1	1
3	TABLE ACCESS	BY INDEX ROWID	DEPARTMENTS	DEPARTMENTS@SEL\$1	1
4	INDEX	RANGE SCAN	DEPT_LOCATION_IX	DEPARTMENTS@SEL\$1	1

Análise de Planos de Execução

- Principais Objetivos:
 - Identificar “desperdício de recursos”
 - Estimativas incorretas

Rows	Execution Plan
12	SORT AGGREGATE
2	SORT GROUP BY
76563	NESTED LOOPS
76575	NESTED LOOPS
19	TABLE ACCESS FULL CN_PAYRUNS_ALL
76570	TABLE ACCESS BY INDEX ROWID CN_POSTING_DETAILS_ALL
76570	INDEX RANGE SCAN (object id 178321)
76563	TABLE ACCESS BY INDEX ROWID CN_PAYMENT_WORKSHEETS_ALL
11432983	INDEX RANGE SCAN (object id 186024)

DBMS_XPLAN

- DBMS_XPLAN.DISPLAY():
 - Formata o conteúdo da PLAN_TABLE
 - select * from table(DBMS_XPLAN.DISPLAY)
- DBMS_XPLAN.DISPLAY_CURSOR():
 - Formata o conteúdo e exibe o plano de execução de qualquer cursor carregado.
 - Por default exibe o último executado
 - Pode exibir o plano de qualquer SQL no cursor cache

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

```
DBMS_XPLAN.DISPLAY_CURSOR(
    sql_id         IN  VARCHAR2  DEFAULT NULL,
    child_number   IN  NUMBER     DEFAULT NULL,
    format          IN  VARCHAR2  DEFAULT 'TYPICAL');
```

DBMS_XPLAN: Exemplos

```
EXPLAIN PLAN
SET statement_id = 'ex_plan2' FOR
SELECT last_name
FROM employees
WHERE last_name LIKE 'Pe%';
```

```
SELECT PLAN_TABLE_OUTPUT
  FROM TABLE(DBMS_XPLAN.DISPLAY(NULL, 'ex_plan2','BASIC'));
```

Id	Operation	Name
0	SELECT STATEMENT	
1	INDEX RANGE SCAN	EMP_NAME_IX

```
SELECT /* TOTO */ ename, dname
  FROM dept d join emp e USING (deptno);
```

SQL_ID	CHILD_NUMBER
gwp663cqh5qbf	0

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR('gwp663cqh5qbf',0));
```

```
Plan hash value: 3693697075, SQL ID: gwp663cqh5qbf, child number: 0
```

```
-----
```

```
SELECT /* TOTO */ ename, dname
  FROM dept d JOIN emp e USING (deptno);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				7 (100)	
1	SORT GROUP BY		4	64	7 (43)	00:00:01
* 2	HASH JOIN		14	224	6 (34)	00:00:01
3	TABLE ACCESS FULL	DEPT	4	44	3 (34)	00:00:01
4	TABLE ACCESS FULL	EMP	14	70	3 (34)	00:00:01

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

```
2 - access("E"."DEPTNO"="D"."DEPTNO")
```

DBMS_XPLAN: Format

- Formatos:
 - BASIC: Mínimo de informação – operação, id (ordem) e opção (objeto)
 - TYPICAL: Default = BASIC + ROWS, BYTES e COST
 - SERIAL: = Typical, porém exibe planos paralelos como seriais
 - ALL: = Typical + PROJECTION, ALIAS e REMOTE
- Modificadores
 - PARTITION: se relevante, mostra informações de partition pruning
 - PARALLEL: se relevante, mostra informações de parallel execution
 - NOTE: notas de execução (dynamic sampling, adaptive cursor, etc)
 - ALLSTATS = IOSTATS + MEMSTATS
 - LAST: apenas considera a última execução

Estimativas x Realidade

- Hint: GATHER_PLAN_STATISTICS
- Format: ALLSTATS LAST

```
SELECT p.prod_name, SUM(s.quantity_sold)
FROM sales s, products p
WHERE s.prod_id = p.prod_id
AND p.prod_desc = 'Envoy Ambassador'
GROUP By p.prod_name ;
```

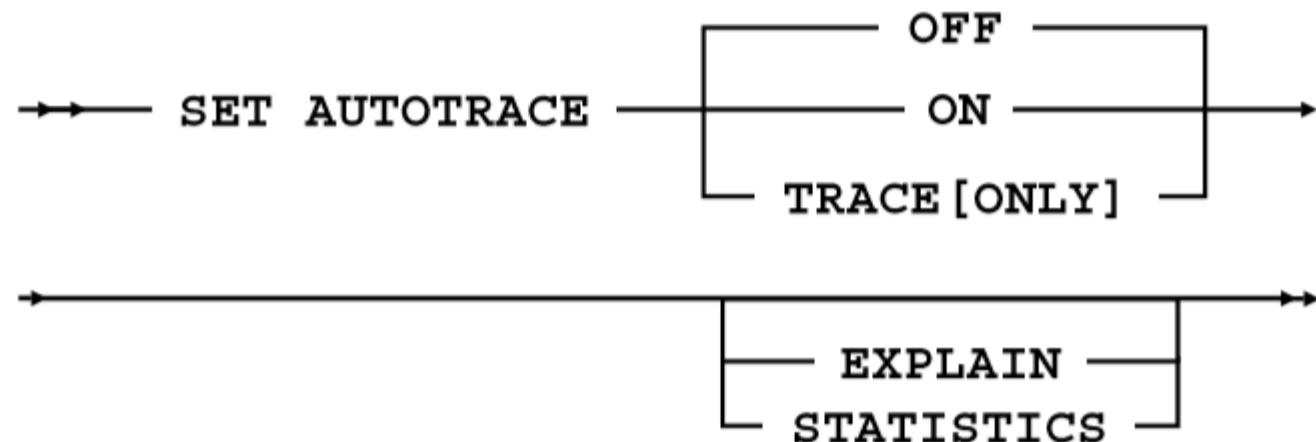
Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0 SELECT STATEMENT				539 (100)	
1 HASH GROUP BY		1	65	539 (4)	00:00:07
* 2 HASH JOIN		12941	821KI	537 (3)	00:00:07
* 3 TABLE ACCESS STORAGE FULL	PRODUCTS	1	58	3 (0)	00:00:01
4 PARTITION RANGE ALL		918KI	6281KI	530 (3)	00:00:07
5 TABLE ACCESS STORAGE FULL	SALES	918KI	6281KI	530 (3)	00:00:07

```
SELECT /*+ GATHER_PLAN_STATISTICS */ p.prod_name, SUM(s.quantity_sold)
FROM sales s, products p
WHERE s.prod_id = p.prod_id
AND p.prod_desc = 'Envoy Ambassador'
GROUP By p.prod_name ;
```

Id Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0 SELECT STATEMENT		1		1	00:00:00.35	1638
1 HASH GROUP BY		1	1	1	00:00:00.35	1638
* 2 HASH JOIN		1	12941	9591	00:00:00.01	1638
* 3 TABLE ACCESS STORAGE FULL	PRODUCTS	1	1	1	00:00:00.01	3
4 PARTITION RANGE ALL		1	918KI	918KI	00:00:00.41	1635
5 TABLE ACCESS STORAGE FULL	SALES	28	918KI	918KI	00:00:00.21	1635

Autotrace

- Ferramenta do SQL*Plus
- Útil para visualizar o plano e estatísticas básicas rapidamente
- Depende da PLAN_TABLE
- Pode não ser o plano de execução real



Autotrace

- Executa a query e mostra estatísticas + plano
 - set autotrace on
- Apenas mostra o plano de execução
 - set autotrace traceonly explain
- Executa a query e mostra estatísticas (sem plano)
 - set autotrace on statistics
- Estatísticas + plano (sem resultado da query)
 - set autotrace traceonly
- Mostra configuração atual:
 - show autotrace

Views Importantes

- V\$sql_plan: planos de execução que ainda estão no library cache, plano real utilizado (diferente da PLAN_TABLE que é o plano teórico)
 - Para juntar com a V\$sql: ADDRESS, HASH_VALUE e CHILD_NUMBER
- V\$sql_plan_statistics: estatísticas de execução
 - STATISTICS LEVEL = ALL
 - GATHER_PLAN_STATISTICS (hint)

Prática 07: Planos de Execução

- Objetivos: praticar as técnicas de captura e visualização de planos de execução.
- EXPLAIN PLAN e PLAN_TABLE
- DBMS_XPLAN
 - DISPLAY
 - DISPLAY_CURSOR
- Autotrace
- Principais Views
 - v\$sql_plan
 - v\$sql_plan_statistics

Performance Tuning com Oracle 12c

Aula 08: Otimizador I

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- *Hints*
- Caminhos de Acesso: Tabelas
- Caminhos de Acesso: Índices

Operações do Otimizador

- Operadores Unários: caminhos de acesso (*Access Path*)
- Operadores Binários: combinação de fontes de linhas (*Join*)
- Operadores n-ários: operações entre conjuntos

Hints do Otimizador

- São instruções que **podem** influenciar a decisão do otimizador.
 - Geralmente são respeitadas, a menos que sejam *hints* incoerentes, ex., tentar forçar um índice que não consegue responder a consulta
- Usando *hints* manuais:
 - Após o comando (SELECT, UPDATE, INSERT) adicionar um comentário com o sinal de '+':
`INSERT /*+ APPEND */ INTO ...`
`SELECT /*+ GATHER_PLAN_STATISTICS */ ...`
 - Também pode ser usada a sintaxe abaixo:
`INSERT --+ APPEND`
`INTO ...`

Access Path: Principais Operadores

- Tabelas:
 1. *Full Table Scan*
 2. *Rowid Scan*
 3. *Sample Table Scan*
 4. *In-Memory Table Scan*
- Índices
 1. *Index Scan (Unique)*
 2. *Index Scan (Range)*
 3. *Index Scan (Full)*
 4. *Index Scan (Fast Full)*
 5. *Index Scan (Skip)*
 6. *Index Scan (Index Join)*
 7. *Bitmap Index Single Value*
 8. *Bitmap Index Range Scan*
 9. *Bitmap Merge*

Full Table Scan

- Executa *multiblock reads*
 - DB_MULTIBLOCK_READ_COUNT
- Pode aplicar um filtro na leitura
- **Mais rápido que índices para grandes volumes de dados!**
- Usado quando:
 - Não há índice
 - Baixa seletividade do filtro (ou sem filtro) [density => 1]
 - Tabela pequena
 - Alto DOP

```

SQL_ID  54c20f3udfnws, child number 0
-----
select salary from hr.employees where salary > 4000

Plan hash value: 3476115102

-----



| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU) | Time      |
|---|---|---|---|---|---|---|
|  0  | SELECT STATEMENT   |             |       |       |     3 (100) |           |
| * 1  |  TABLE ACCESS FULL | EMPLOYEES  |    98 |  6762 |       3 (0)  | 00:00:01  |

-----


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


      1 - filter("SALARY">>4000)
  
```

ROWID Scan

- ROWID: endereço físico da linha nos data files e blocos
- Geralmente consequência de um acesso por índice

```
SELECT *
FROM   employees
WHERE  employee_id > 190;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				2 (100)	
1	TABLE ACCESS BY INDEX ROWID BATCHED	EMPLOYEES	16	1104	2 (0)	00:00:01
*2	INDEX RANGE SCAN	EMP_EMP_ID_PK	16	1	(0)	00:00:01

Predicate Information (identified by operation id):

2 - access ("EMPLOYEE_ID">>190)

Sample Table Scan

- Cláusula SAMPLE do SELECT.
- Faz leitura por amostragem
 - SAMPLE (percentual): amostragem de linhas
 - SAMPLE BLOCK (percentual): amostragem de blocos

```
SELECT * FROM hr.employees SAMPLE BLOCK (1);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		1	68	3 (34)
1	TABLE ACCESS SAMPLE	EMPLOYEES	1	68	3 (34)

In-Memory Table Scan

- Option: In-memory Database do 12c
- In-memory column store precisa estar ativada

```

SQL_ID  2mb4h57x8pabw, child number 0
-----
select * from oe.product_information where list_price > 10 order by product_id

Plan hash value: 2256295385

-----  

| Id| Operation           | Name          | Rows | Bytes | TempSpc|Cost (%CPU) |Time |
-----  

|  0| SELECT STATEMENT    |               |      |       |        |21 (100)|      |  

|  1|  SORT ORDER BY      |               | 285 | 62415 |82000|21   (5)|00:00:01|  

|*2|  TABLE ACCESS INMEMORY FULL| PRODUCT_INFORMATION | 285 | 62415 |       | 5   (0)|00:00:01|
-----  

Predicate Information (identified by operation id):  

-----  

  2 - inmemory("LIST_PRICE">>10)  

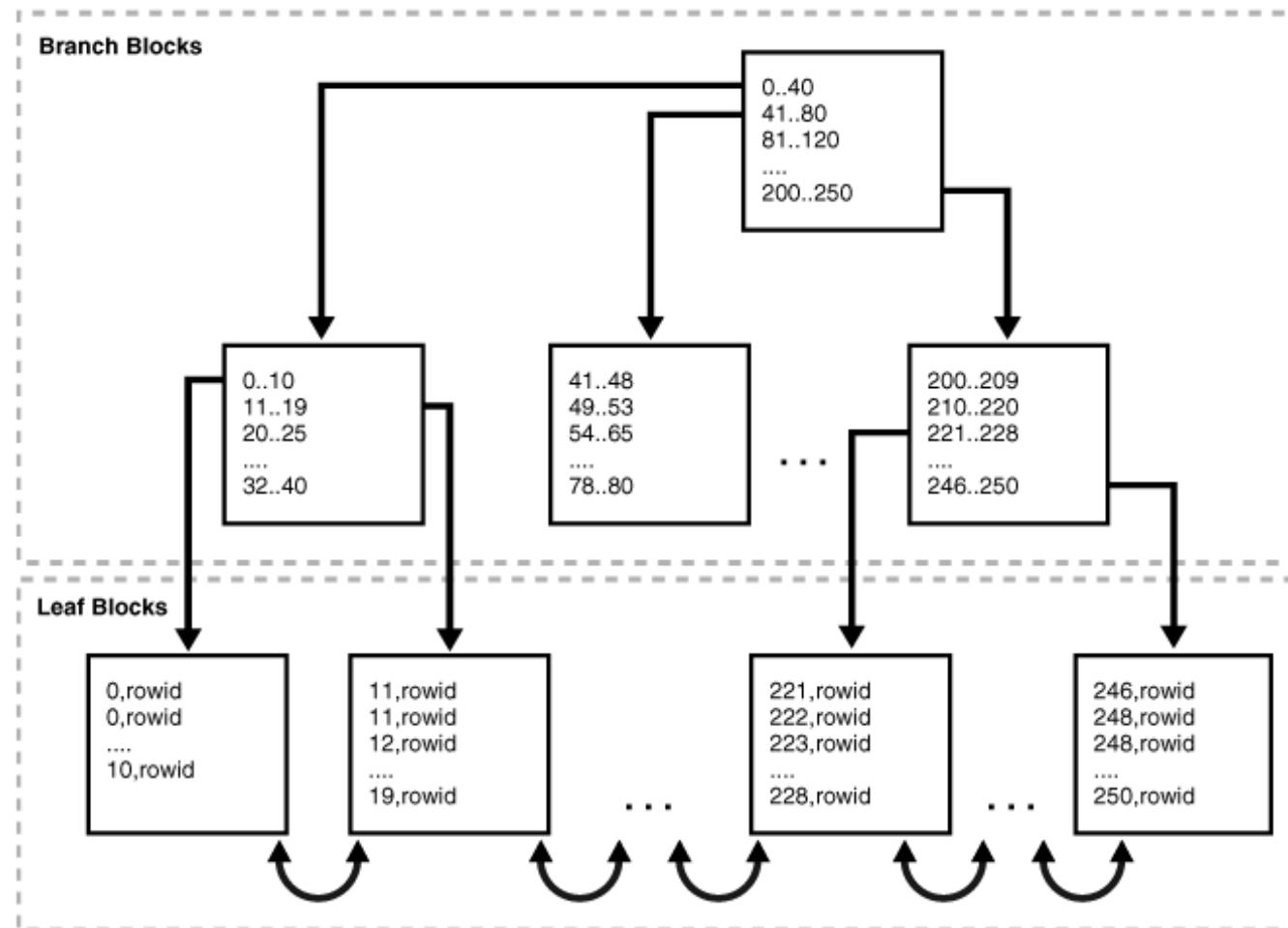
      filter("LIST_PRICE">>10)

```

Índices: Visão Geral

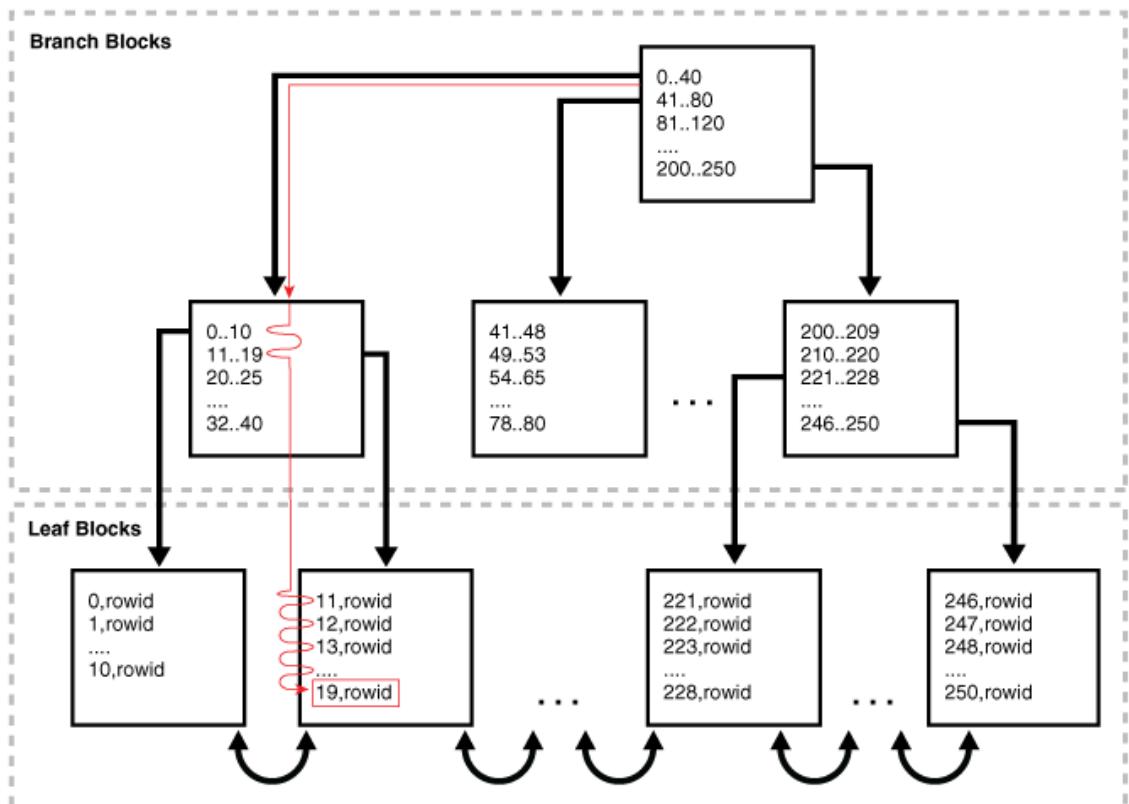
- Tipos de Índices:
 - Normal
 - Baseado em Função
 - Tabela organizada em índice (IOT)
 - *Bitmap*
- Atributos
 - Compressão
 - Chave normal ou reversa
 - Sentido: ascendente ou descendente

Índices B*-tree: Estrutura



Importante: Índices B*-tree não armazenam chaves nulas!

Index Unique Scan



```

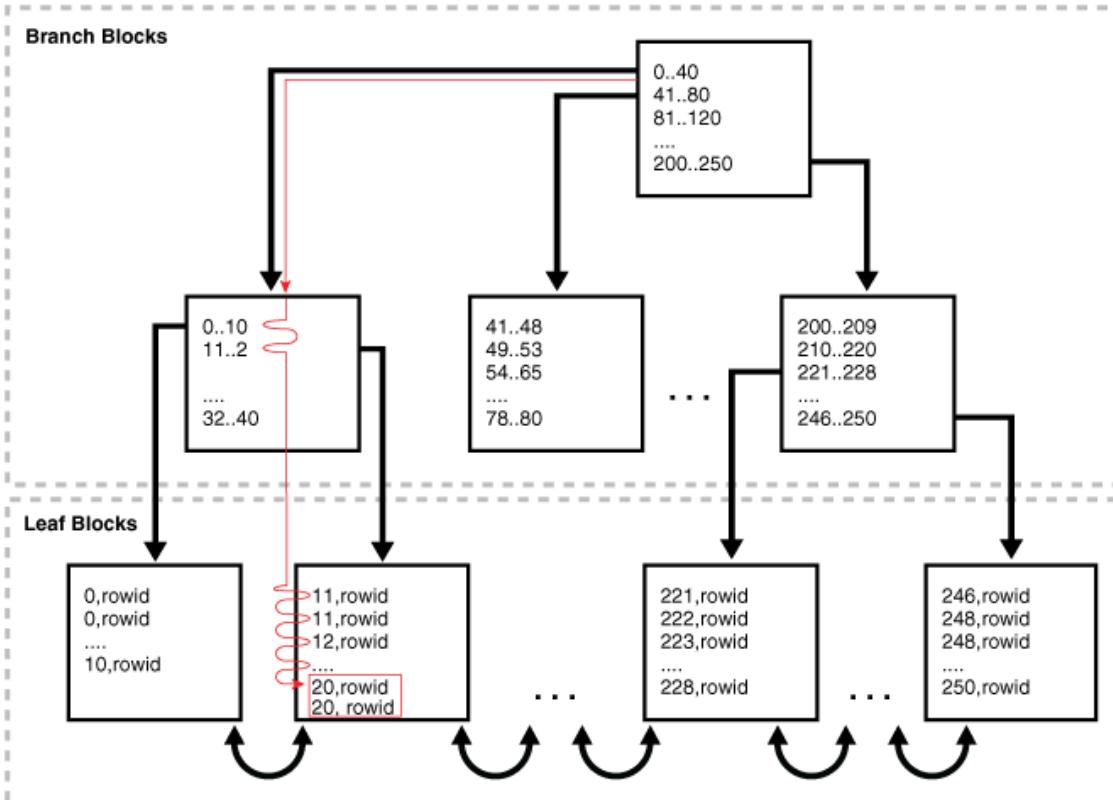
SQL_ID 3ptq5tsd5vb3d, child number 0
-----
select * from sh.products where prod_id = 19

Plan hash value: 4047888317

| Id | Operation           | Name      | Rows | Bytes | Cost (%CPU) | Time      |
| 0  | SELECT STATEMENT   |          | 1    | 173   | 1 (100)    | 00:00:01 |
| 1  |  TABLE ACCESS BY INDEX ROWID | PRODUCTS | 1    | 173   | 1 (0)      | 00:00:01 |
| *2 |   INDEX UNIQUE SCAN | PRODUCTS_PK | 1    | 1    | 0 (0)      |            |

Predicate Information (identified by operation id):
-----
2 - access ("PROD_ID"=19)
  
```

Index Range Scan



```

SQL_ID  brt5abvbxw9tq, child number 0
-----
SELECT * FROM employees WHERE department_id = 20 AND salary > 1000

Plan hash value: 2799965532
  
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	138	2 (100)	00:00:01
*1	TABLE ACCESS BY INDEX ROWID BATCHED	EMPLOYEES	2	138	2 (0)	00:00:01
*2	INDEX RANGE SCAN	EMP_DEPARTMENT_IX	2	1	1 (0)	00:00:01

Predicate Information (identified by operation id):

- 1 - filter("SALARY">>1000)
- 2 - access("DEPARTMENT_ID"=20)

Index Range Scan (desc)

```
SQL_ID  8182ndfj1ttj6, child number 0
-----
SELECT * FROM employees WHERE department_id < 20 ORDER BY department_id DESC

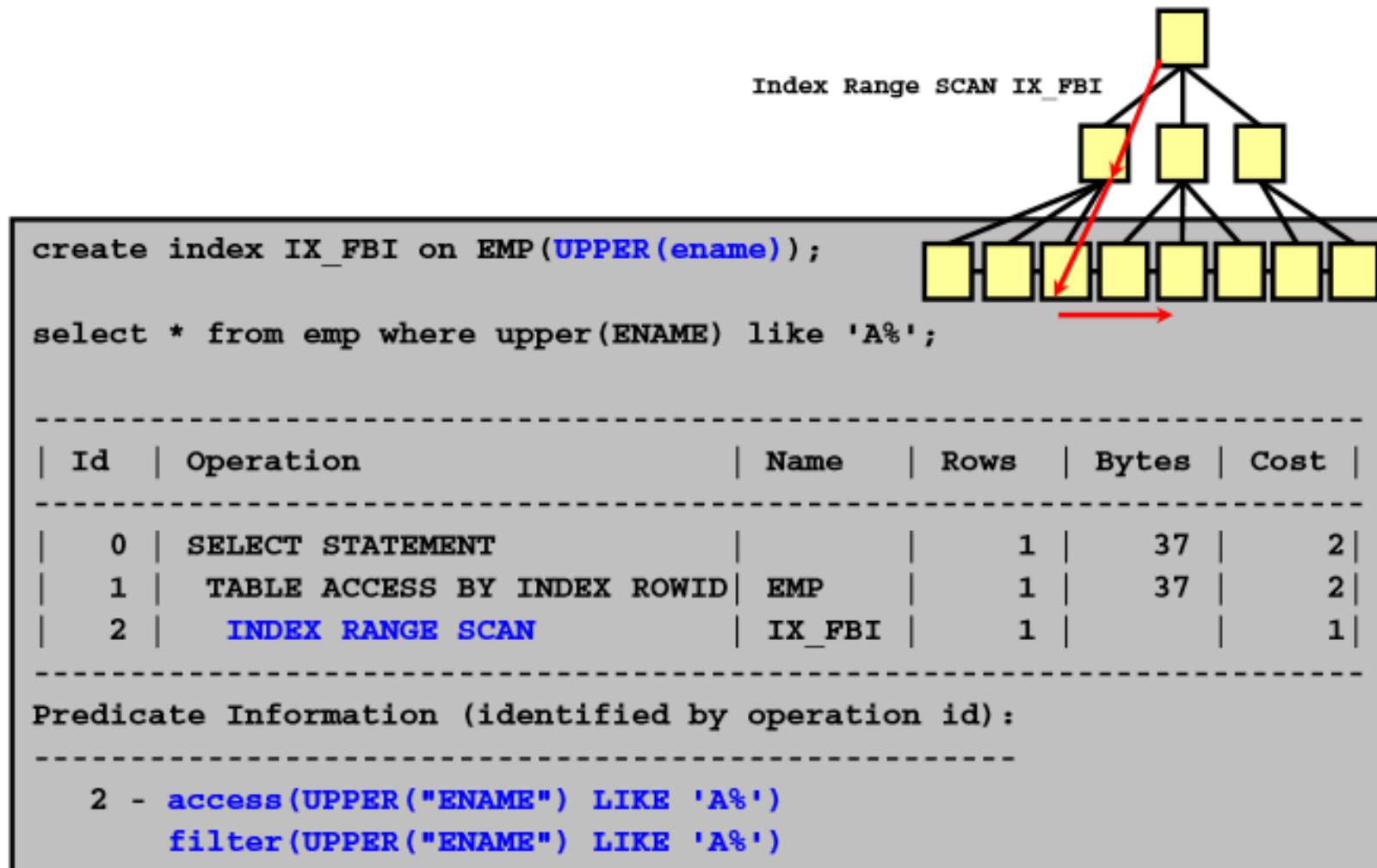
Plan hash value: 1681890450

-----

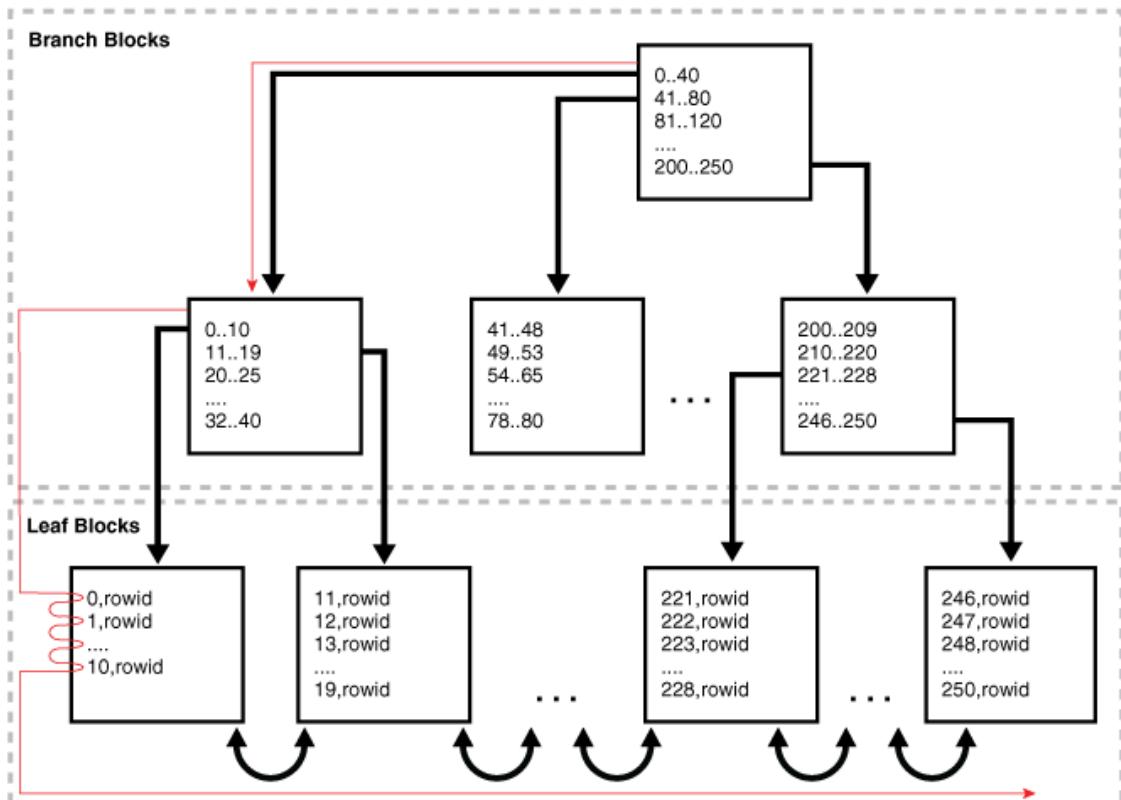
| Id | Operation                   | Name              | Rows | Bytes | Cost (%CPU) | Time     |
|----|-----------------------------|-------------------|------|-------|-------------|----------|
| 0  | SELECT STATEMENT            |                   | 2    | (100) |             |          |
| 1  | TABLE ACCESS BY INDEX ROWID | EMPLOYEES         | 2    | 138   | 2 (0)       | 00:00:01 |
| *2 | INDEX RANGE SCAN DESCENDING | EMP_DEPARTMENT_IX | 2    | 1     | (0)         | 00:00:01 |


-----
Predicate Information (identified by operation id):
-----
2 - access ("DEPARTMENT_ID"<20)
```

Index Range Scan: Function-Based



Index Full Scan



SQL_ID 94t4a20h8what, child number 0

select department_id, department_name from departments order by department_id

Plan hash value: 4179022242

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				2 (100)	
1	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	27	432	2 (0)	00:00:01
2	INDEX FULL SCAN	DEPT_ID_PK	27		1 (0)	00:00:01

Lê em ordem, acessa a tabela pela rowid

Index Fast Full Scan

- Consulta apenas acessa atributos do índice
 - Lê o índice como se fosse uma tabela (direto nas folhas)
 - Não se preocupa com ordem

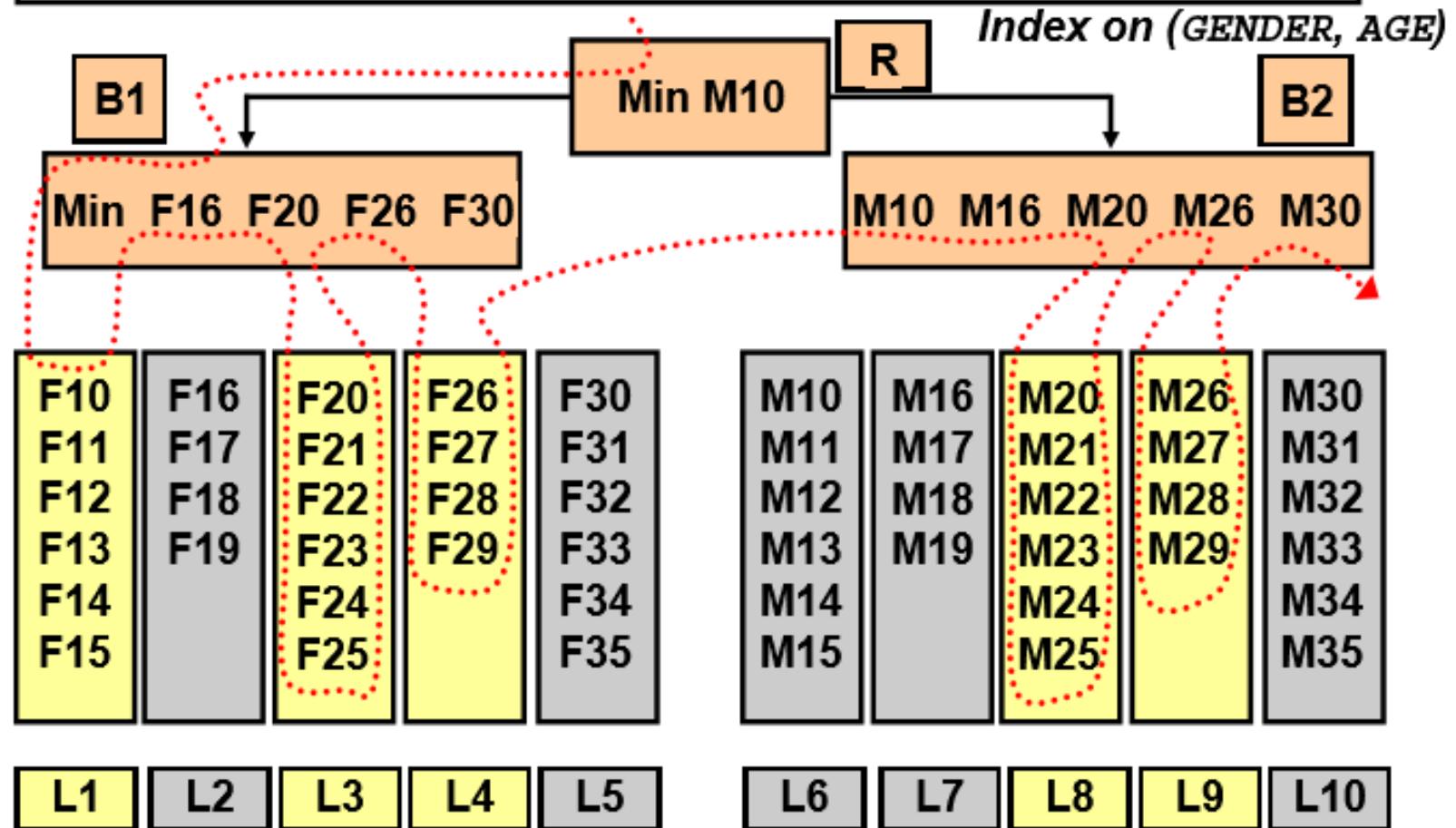
```
SQL_ID  fu0k5nvx7sftm, child number 0
-----
select /*+ index_ffs(departments dept_id_pk) */ count(*) from departments

Plan hash value: 3940160378

-----
| Id  | Operation           | Name      | Rows | Cost (%CPU) | Time      |
-----
|   0 | SELECT STATEMENT    |           |       | 2 (100) |           |
|   1 |  SORT AGGREGATE     |           |   1  |             |           |
|   2 | INDEX FAST FULL SCAN| DEPT_ID_PK|  27  | 2 (0)  | 00:00:01 |
```

Index Skip Scan

```
SELECT * FROM employees WHERE age BETWEEN 20 AND 29
```



Index Skip Scan

```
SQL_ID  d7a6xurcnx2dj, child number 0
-----
SELECT * FROM sh.customers WHERE cust_email = 'Abbey@company.example.com'

Plan hash value: 797907791

-----
| Id | Operation           | Name          | Rows | Bytes | Cost (%CPU) | Time |
| 0 | SELECT STATEMENT   |               | 1    | 10 (100) |           |       |
| 1 |  TABLE ACCESS BY INDEX ROWID BATCHED | CUSTOMERS   | 33  | 6237 | 10 (0) | 00:00:01 |
|*2|   INDEX SKIP SCAN    | CUST_GENDER_EMAIL_IX | 33 |     | 4 (0) | 00:00:01 |

-----
Predicate Information (identified by operation id):
-----
2 - access("CUST_EMAIL"='Abbey@company.example.com')
      filter("CUST_EMAIL"='Abbey@company.example.com')
```

Index Join Scan

```

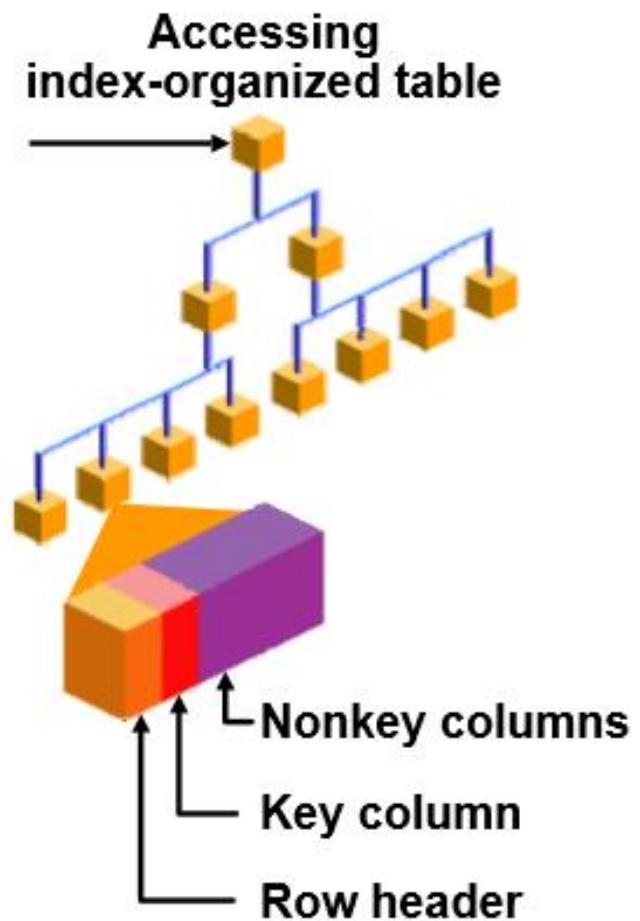
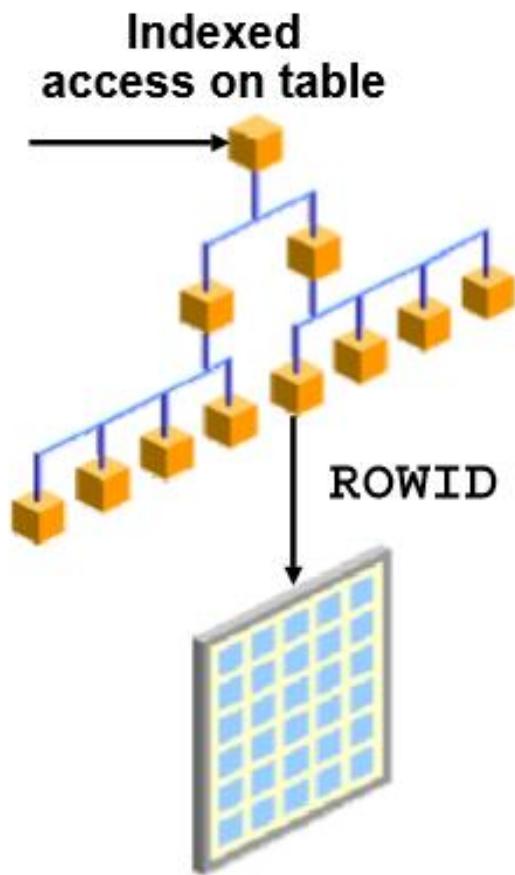
SQL_ID  d2djchyc9hmrz, child number 0
-----
SELECT /*+ INDEX_JOIN(employees) */ last_name, email FROM   employees
WHERE  last_name like 'A%'

Plan hash value: 3719800892
-----
| Id  | Operation          | Name           | Rows | Bytes | Cost (%CPU) | Time      |
|---|---|---|---|---|---|---|
| 0  | SELECT STATEMENT  |                |       |       | 3 (100) |          |
| * 1 |  VIEW              | index$_join$_001 | 3    | 48   | 3 (34) | 00:00:01 |
| * 2 |  HASH JOIN          |                 |       |       |          |          |
| * 3 |  INDEX RANGE SCAN  | EMP_NAME_IX   | 3    | 48   | 1 (0)  | 00:00:01 |
| 4  |  INDEX FAST FULL SCAN| EMP_EMAIL_UK | 3    | 48   | 1 (0)  | 00:00:01 |

Predicate Information (identified by operation id):
-----
 1 - filter("LAST_NAME" LIKE 'A%')
 2 - access(ROWID=ROWID)
 3 - access("LAST_NAME" LIKE 'A%')

```

Tabelas Organizadas por Índice (IOT)



IOT Table Scan

```
select * from iotemp where empno=9999;
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	87	1
1	INDEX UNIQUE SCAN	SYS_IOT_TOP_75664	1	87	1

Predicate Information (identified by operation id):

```
1 - access("EMPNO"=9999)
```

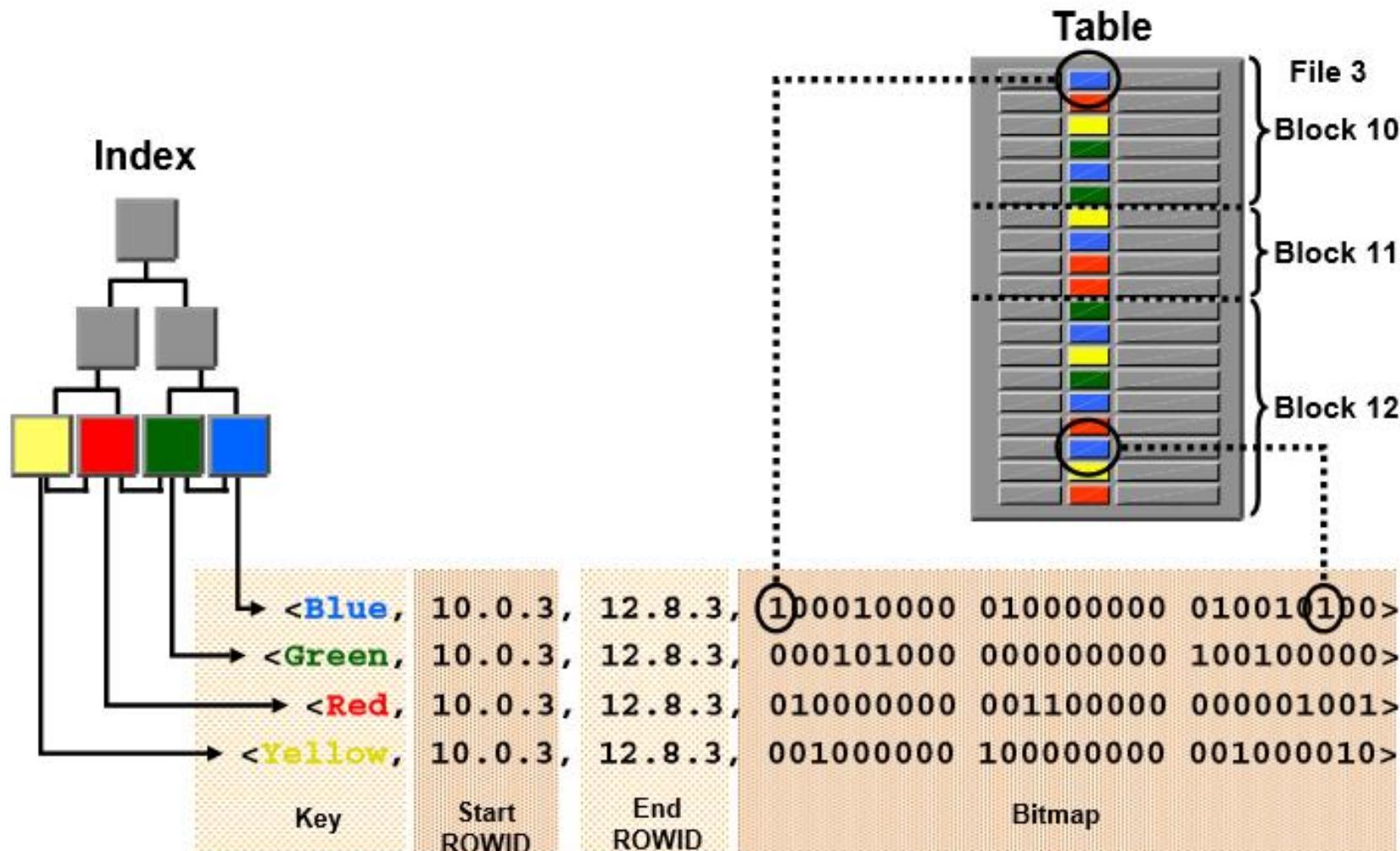
```
select * from iotemp where sal>1000;
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		12	1044
1	INDEX FAST FULL SCAN	SYS_IOT_TOP_75664	12	1044

Predicate Information (identified by operation id):

```
1 - filter("SAL">1000)
```

Índices Bitmap



Bitmap: Exemplos de Acesso

```
SELECT * FROM PERF_TEAM WHERE country='FR';
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	45
1	TABLE ACCESS BY INDEX ROWID	PERF_TEAM	1	45
2	BITMAP CONVERSION TO ROWIDS			
3	BITMAP INDEX SINGLE VALUE	IX_B2		

Predicate: 3 - access("COUNTRY"='FR')

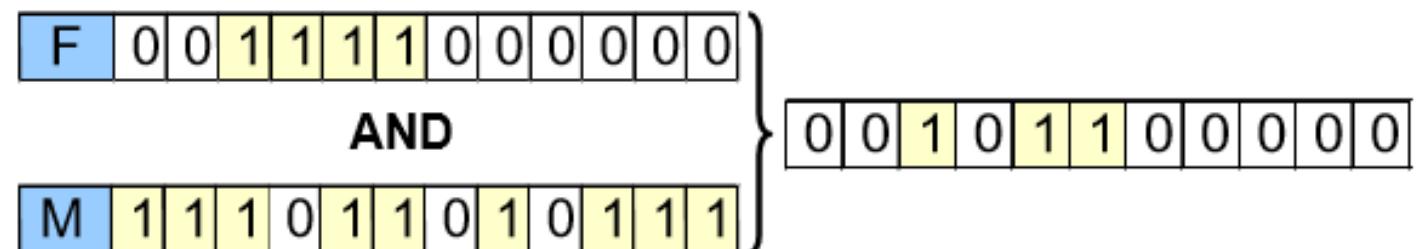
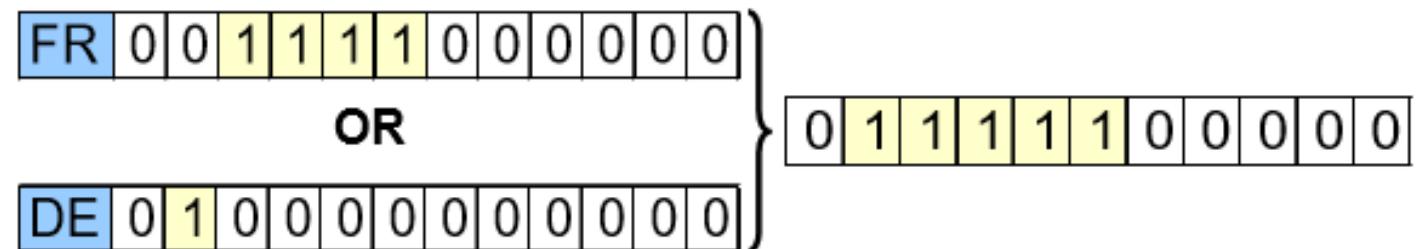
```
SELECT * FROM PERF_TEAM WHERE country>'FR';
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	45
1	TABLE ACCESS BY INDEX ROWID	PERF_TEAM	1	45
2	BITMAP CONVERSION TO ROWIDS			
3	BITMAP INDEX RANGE SCAN	IX_B2		

Predicate: 3 - access("COUNTRY">>'FR') filter("COUNTRY">>'FR')

Bitmap: Operações

```
SELECT * FROM PERF_TEAM WHERE country in('FR','DE');
```



```
SELECT * FROM EMEA_PERF_TEAM T WHERE country='FR' and gender='M';
```

Bitmap: Operações

```
SELECT * FROM PERF_TEAM WHERE country in ('FR','DE');
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	45
1	INLIST ITERATOR			
2	TABLE ACCESS BY INDEX ROWID	PERF_TEAM	1	45
3	BITMAP CONVERSION TO ROWIDS			
4	BITMAP INDEX SINGLE VALUE	IX_B2		

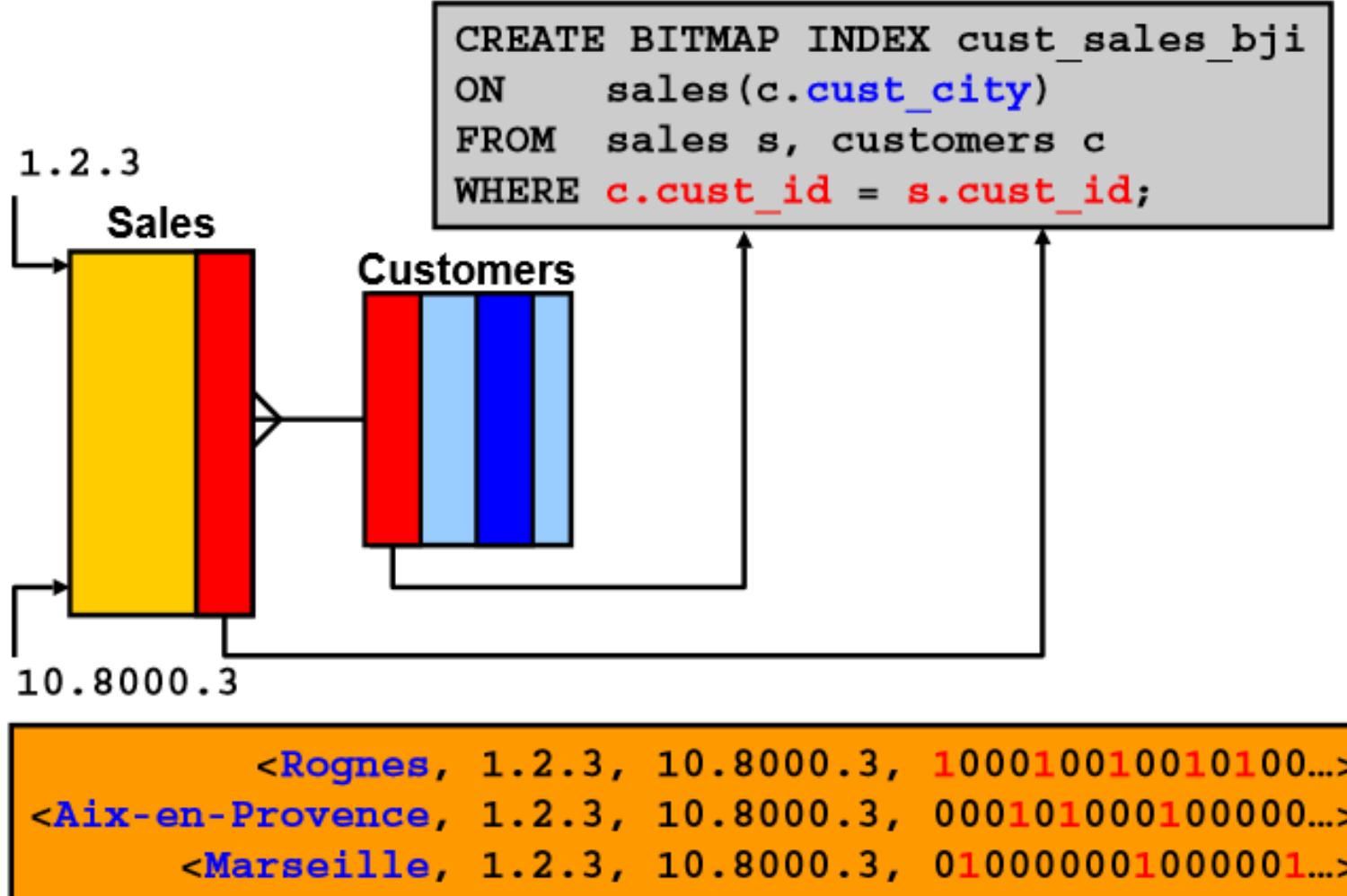
Predicate: 4 - access("COUNTRY"='DE' OR "COUNTRY"='FR')

```
SELECT * FROM PERF_TEAM WHERE country='FR' and gender='M';
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	45
1	TABLE ACCESS BY INDEX ROWID	PERF_TEAM	1	45
2	BITMAP CONVERSION TO ROWIDS			
3	BITMAP AND			
4	BITMAP INDEX SINGLE VALUE	IX_B1		
5	BITMAP INDEX SINGLE VALUE	IX_B2		

Predicate: 4 - access("GENDER"='M') 5 - access("COUNTRY"='FR')

Bitmap Join Index



Prática 08: Operadores do Otimizador II

- Objetivos: conhecer os principais caminhos de acesso para tabelas e índices.
- *Access Paths:*
 - Tabelas
 - Índices
 - IOT
 - Índices *Bitmap*

Performance Tuning com Oracle 12c

Aula 09: Otimizador II

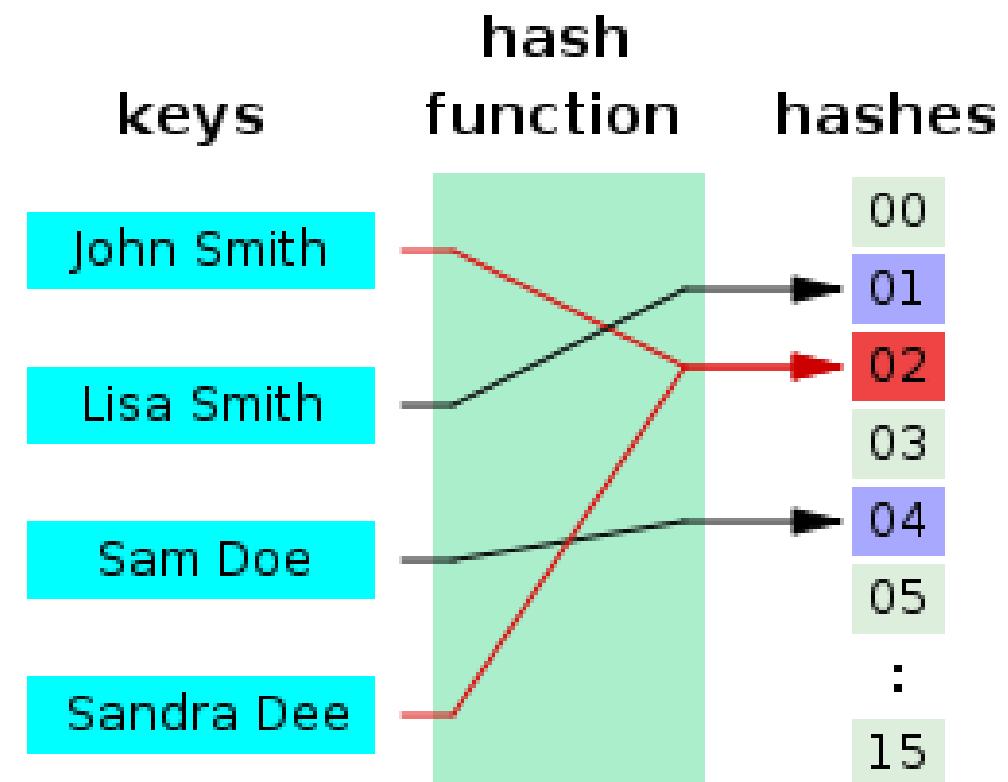
Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

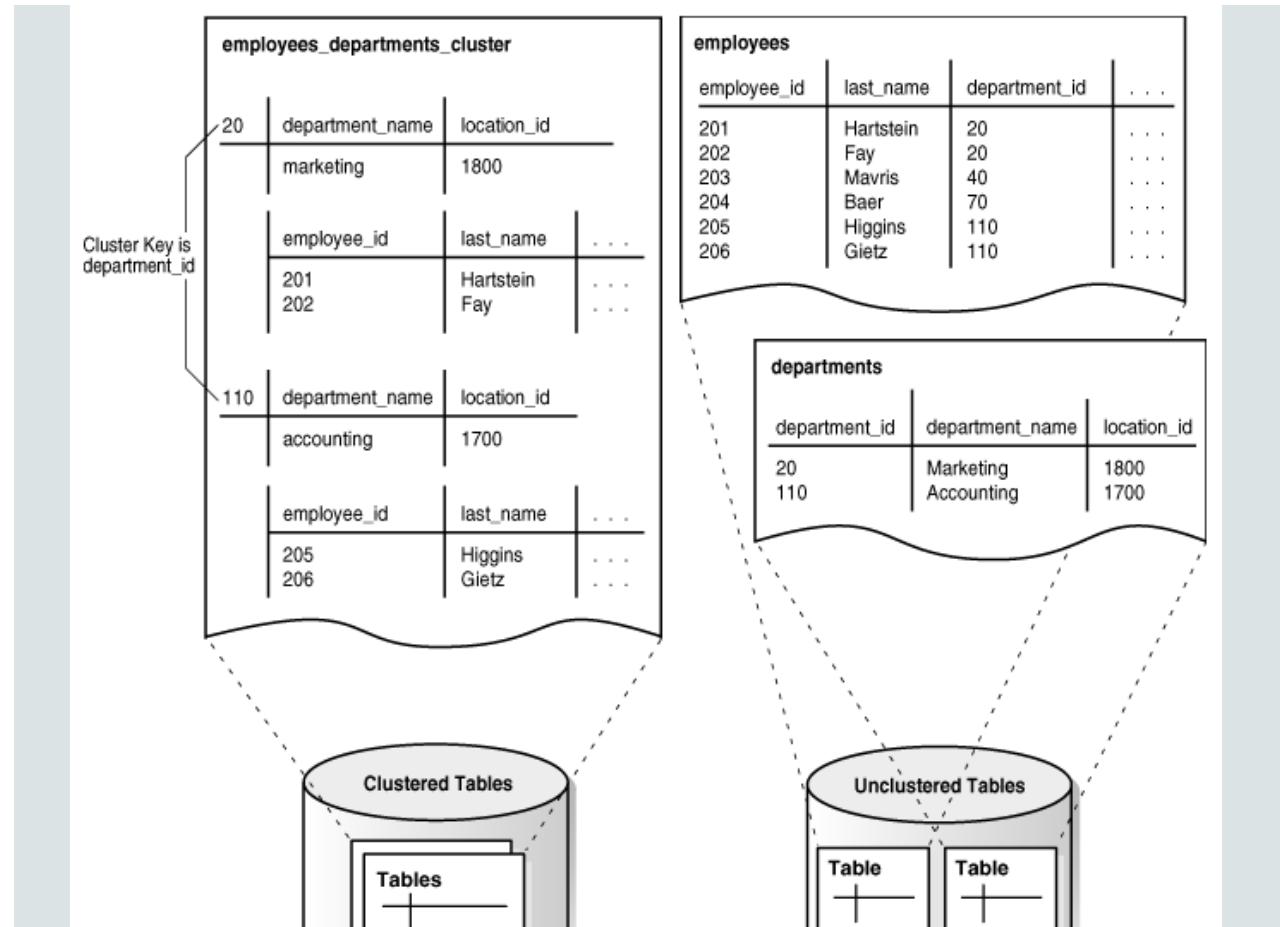
- Caminhos de Acesso: *Cluster*
- Operadores: Ordenação
- Operadores: *Join*
- Operadores: Conjuntos

Uma Breve Revisão Sobre Funções Hash

- Uma função hash é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo.
- Os valores retornados por uma função hash são chamados **valores hash, códigos hash, somas hash (hash sums), checksums ou simplesmente hashes**.
- Um *hash* pode ser entendido como uma assinatura de determinado dado. Ex.: MD4, MD5, SHA-1



Cluster Tables



Um *cluster* de tabelas é um grupo de tabelas que compartilham colunas e guardam dados correlacionados no mesmo espaço físico.

Podem ser de dois tipos:

1. *Indexed Cluster*
2. *Hash Cluster*

Benefícios:

1. Reduz I/O e melhora tempos de acesso em *joins* de tabelas agrupadas
2. Usa menos *storage* (eliminando repetições)

Indexed Cluster

- Quando usar:
 - Tabelas predominantemente lidas (poucos ou nenhum *update/delete*)
 - Tabelas acessadas juntas frequentemente
- Quando não usar:
 - Tabelas muito atualizadas
 - Tabelas que crescem demais
 - Tabelas que são acessadas isoladamente

Indexed Cluster

```
CREATE CLUSTER employees_departments_cluster
(department_id NUMBER(4)) SIZE 512;
```

```
CREATE INDEX idx_emp_dept_cluster
ON CLUSTER employees_departments_cluster;
```

```
CREATE TABLE employees2
CLUSTER employees_departments_cluster (department_id)
AS SELECT * FROM employees;
CREATE TABLE departments2
CLUSTER employees_departments_cluster (department_id)
AS SELECT * FROM departments;
```

```
SQL_ID b7xk1jzuwdc6t, child number 0
-----
```

```
SELECT * FROM employees2 WHERE department_id = 30
```

```
Plan hash value: 49826199
```

	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
	0	SELECT STATEMENT				2 (100)	
	1	TABLE ACCESS CLUSTER	EMPLOYEES2	6	798	2 (0)	00:00:01
	*2	INDEX UNIQUE SCAN	IDX_EMP_DEPT_CLUSTER	1		1 (0)	00:00:01

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

```
2 - access ("DEPARTMENT_ID"=30)
```

Hash Cluster

- Quando usar:
 - A maioria das consultas é por igualdade na chave:
 - SELECT ... WHERE cluster_key = ...
 - A tabela é estática em termos de tamanho
 - O cluster é dimensionado no momento da criação (tamanho físico no disco)
- Quando não usar:
 - Consultas por faixa (RANGE) são frequentes:
 - SELECT ... WHERE cluster_key > ...
 - Tabela que cresce continuamente
 - Necessidade de realizar muitos FULL TABLE SCAN

Hash Cluster

```

CREATE CLUSTER employees_departments_cluster
  (department_id NUMBER(4)) SIZE 8192 HASHKEYS 100;

CREATE TABLE employees2
  CLUSTER employees_departments_cluster (department_id)
  AS SELECT * FROM employees;

CREATE TABLE departments2
  CLUSTER employees_departments_cluster (department_id)
  AS SELECT * FROM departments;
  
```

SQL_ID 919x7hyyxr6p4, child number 0

SELECT * FROM employees2 WHERE department_id = 30

Plan hash value: 2399378016

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT				1
*	1 TABLE ACCESS HASH EMPLOYEES2		10	1330	

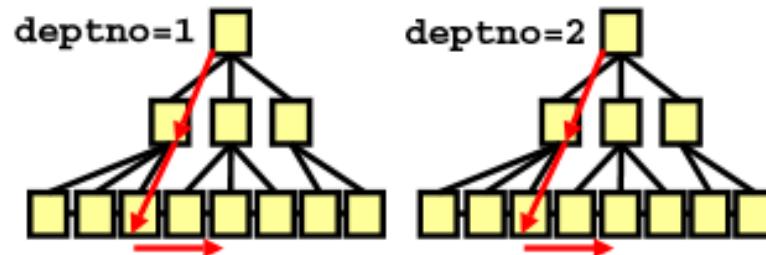
Predicate Information (identified by operation id):

1 - access ("DEPARTMENT_ID"=30)

Operadores do Otimizador: Ordenação

- Operador SORT:
 - AGGREGATE: Retorna uma linha em uma operação de grupo
 - UNIQUE: Elimina duplicidades
 - JOIN: Antecede um *merge join*
 - GROUP BY, ORDER BY
- Operador HASH:
 - GROUP BY
 - UNIQUE: Equivalente ao SORT UNIQUE
- Para resultados ordenados, sempre use ORDER BY.

Inlist Iterator



```
select * from emp where deptno in (1,2);
select * from emp where deptno = 1 or deptno =2 ;
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		2	78	2
1	INLIST ITERATOR				
2	TABLE ACCESS BY INDEX ROWID	EMP	2	78	2
3	INDEX RANGE SCAN	IX_SS	2		1

Predicate Information (identified by operation id):

```
3 - access("DEPTNO"=1 OR "DEPTNO"=2)
```

Min/Max e First Row

```
select min(id) FROM t WHERE id > 500000;
```

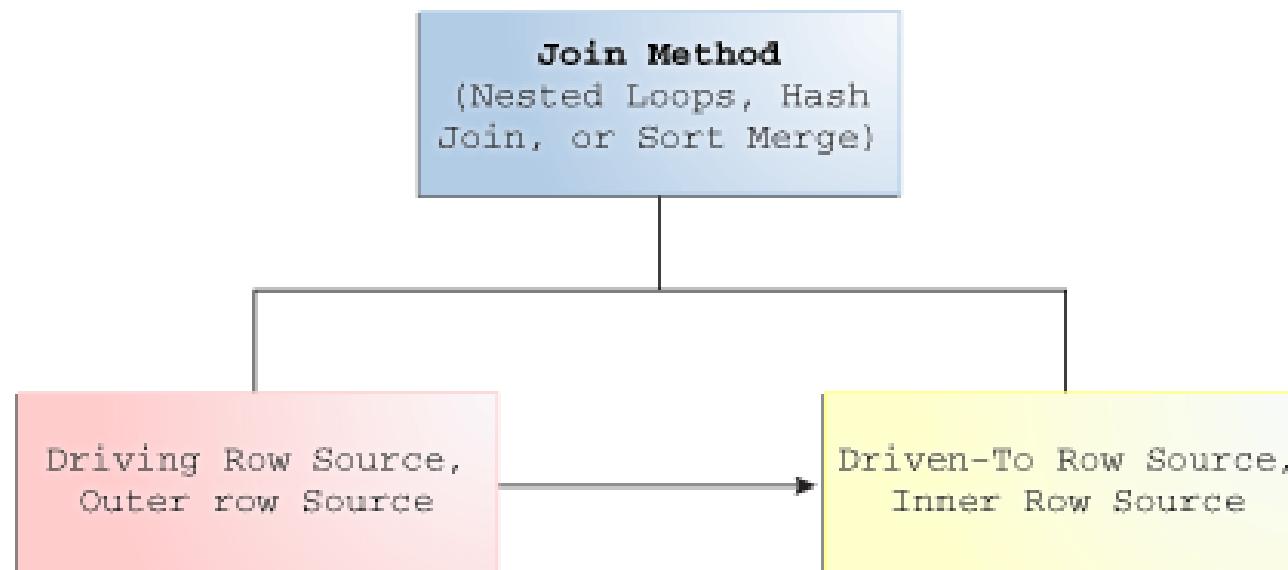
Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	13	3
1	SORT AGGREGATE		1	13	
2	FIRST ROW		717K	9113K	3
3	INDEX RANGE SCAN (MIN/MAX)	IXT	717K	9113K	3

Predicate Information (identified by operation id):

```
3 - access("ID">500000)
```

Métodos de *Join*

- Um *join* define um relacionamento entre duas fontes de linha
- É o método de combinar dados de duas fontes distintas
- É controlado por predicados que definem esta relação
- Métodos:
 - *Nested Loops*
 - *Hash Join*
 - *Sort-merge Join*
 - *Cartesian Join*



Exemplos de Sintaxe do *Join*

```
- select e.last_name, d.department_name
  from hr.employees e,
       hr.departments d
 where e.department_id = d.department_id;

- select e.last_name, d.department_name
  from hr.employees e
    natural join
      hr.departments d;

select e.last_name, d.department_name
  from hr.employees e join
       hr.departments d using(department_id);
```

Seleção do Método de *Join*

Método de Join	Custo do Join A, B	Custo do Join B,A
Nested Loops	39,480	6,187,540
Hash Join	187,528	194,909
Sort Merge	217,129	217,129

Nested Loops

- *Driving Row Source*: loop externo, comanda o loop interno
- Para cada linha externa faz a busca no loop interno
- Ideal para pequenos conjuntos de dados

```

FOR erow IN (select * from employees where X=Y) LOOP
    FOR drow IN (select * from departments where erow is matched) LOOP
        output values from erow and drow
    END LOOP
END LOOP

```

SELECT STATEMENT
 NESTED LOOPS 3
 NESTED LOOPS 2
 NESTED LOOPS 1
 OUTER LOOP 1.1
 INNER LOOP 1.2
 INNER LOOP 2.2
 INNER LOOP 3.2

- Row source becomes OUTER LOOP 3.1
 - Row source becomes OUTER LOOP 2.1



Nested Loops

```

SQL_ID  ahuavfcv4tnz4, child number 0

SELECT /*+ ORDERED USE_NL(d) */ e.last_name, d.department_name FROM
employees e, departments d WHERE  e.department_id=d.department_id AND
e.last_name like 'A%'

Plan hash value: 1667998133

-----



| Id | Operation          | Name   | Rows | Bytes | Cost (%CPU) | Time |
-----



| 0 | SELECT STATEMENT   |        |      |       | 5 (100) |       |
| 1 | NESTED LOOPS      |        |      |       |           |       |
| 2 |   NESTED LOOPS    |        |      |       |           |       |
| 3 |     TABLE ACCESS BY INDEX ROWID BATCHED | EMPLOYEES | 3 | 54 | 2 (0) | 00:00:01 |
|*4|     INDEX RANGE SCAN | EMP_NAME_IX | 3 | 1 | 1 (0) | 00:00:01 |
|*5|     INDEX UNIQUE SCAN | DEPT_ID_PK  | 1 | 10 | 0 (0) |       |
| 6 |     TABLE ACCESS BY INDEX ROWID  | DEPARTMENTS | 1 | 16 | 1 (0) | 00:00:01 |

-----



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



4 - access("E"."LAST_NAME" LIKE 'A%')
   filter("E"."LAST_NAME" LIKE 'A%')
5 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")

```

Hash Joins

- Uso: grandes volumes de dados, ou a maior parte de uma tabela pequena; condição de igualdade
- Algoritmo:
 1. Otimizador escolhe a menor tabela e monta uma tabela de hashes
 2. Para cada linha da tabela grande, calcula o *hash* e checa se existe um hash da tabela pequena igual
 3. Verifica a condição do join e retorna a linha se atender a condição

①

```
FOR small_table_row IN (SELECT * FROM small_table)
LOOP
  slot_number := HASH(small_table_row.join_key);
  INSERT_HASH_TABLE(slot_number, small_table_row);
END LOOP;
```

②

```
FOR large_table_row IN (SELECT * FROM large_table)
LOOP
  slot_number := HASH(large_table_row.join_key);
  small_table_row = LOOKUP_HASH_TABLE(slot_number, large_table_row.join_key);
  IF small_table_row FOUND
    THEN
      output small_table_row + large_table_row;
    END IF;
  END LOOP;
```

Hash Joins

```

SELECT o.customer_id, l.unit_price * l.quantity
FROM   orders o, order_items l
WHERE  l.order_id = o.order_id;

```

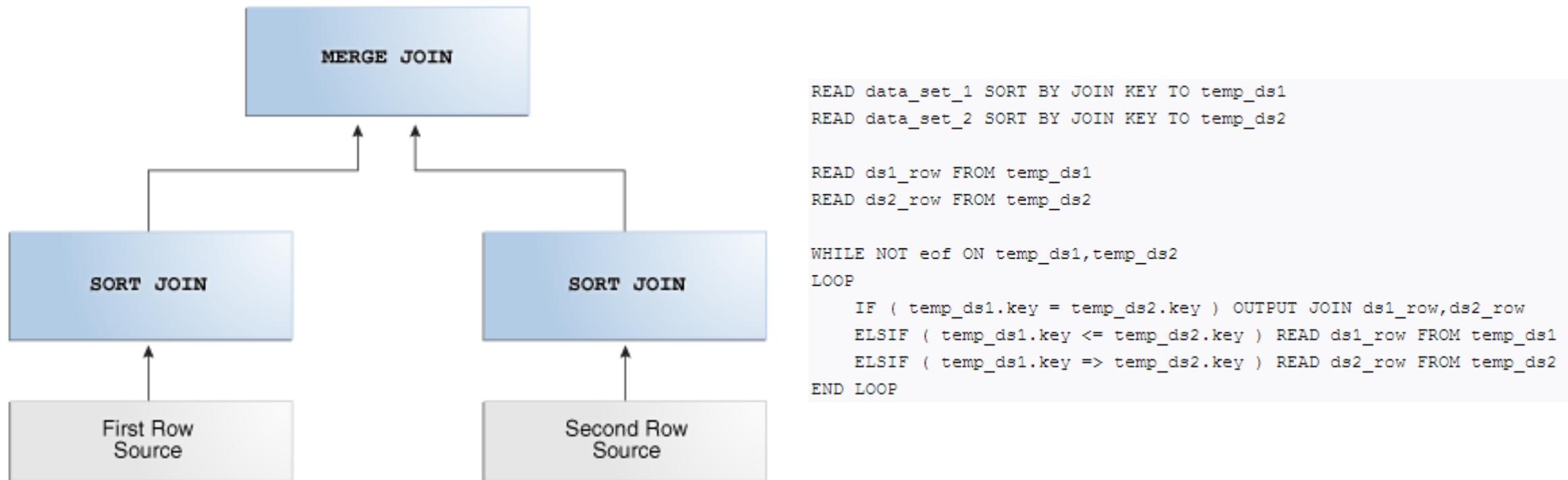
Id	Operation	Name	Rows	Bytes	Cost	(%CPU)
0	SELECT STATEMENT		665	13300	8	(25)
* 1	HASH JOIN		665	13300	8	(25)
2	TABLE ACCESS FULL	ORDERS	105	840	4	(25)
3	TABLE ACCESS FULL	ORDER_ITEMS	665	7980	4	(25)

Predicate Information (identified by operation id):

```
1 - access("L"."ORDER_ID"="O"."ORDER_ID")
```

Sort-Merge Join

- As duas fontes de dados são ordenadas pela mesma chave e depois unidas
- Condição de desigualdade ou necessidade de ordenação



Sort-Merge Join

```

SELECT /*+ USE_MERGE(d e) NO_INDEX(d) */ e.employee_id, e.last_name, e.first_name,
       e.department_id, d.department_name
  FROM employees e, departments d
 WHERE e.department_id = d.department_id
 ORDER BY department_id;

```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				6 (100)	
1	MERGE JOIN		106	9540	6 (34)	00:00:01
2	SORT JOIN		27	567	3 (34)	00:00:01
3	TABLE ACCESS FULL	DEPARTMENTS	27	567	2 (0)	00:00:01
*4	SORT JOIN		107	7383	3 (34)	00:00:01
5	TABLE ACCESS FULL	EMPLOYEES	107	7383	2 (0)	00:00:01

Predicate Information (identified by operation id):

```

4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
      filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")

```

Cartesian Join

- Uso: *join* sem predicado, decisão do otimizador [ex.: duas tabelas pequenas que vão se juntar a uma maior]

```

FOR ds1_row IN ds1 LOOP
    FOR ds2_row IN ds2 LOOP
        output ds1_row and ds2_row
    END LOOP
END LOOP

```

```

SELECT e.last_name, d.department_name
FROM employees e, departments d

```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	
0	SELECT STATEMENT				11 (100)		
1	MERGE JOIN CARTESIAN		2889	57780	11 (0)	00:00:01	
2	TABLE ACCESS FULL	DEPARTMENTS	27	324	2 (0)	00:00:01	
3	BUFFER SORT		107	856	9 (0)	00:00:01	
4	INDEX FAST FULL SCAN	EMP_NAME_IX	107	856	0 (0)		

Tipos de *Join*

- Inner Join (Equijoin/Natural – Nonequijoin)
 - Equijoin: igualdade
 - Nonequijoin: desigualdade $>$, $<$, \leq , \geq , between
- Outer join (Full, left, right)
 - Não precisa ter correspondência dos dois lados
- Semi join: EXISTS / IN subquery
 - Retorna apenas a primeira correspondência
- Anti join: NOT IN / NOT EXISTS subquery
 - Retorna o que não tem correspondência

Inner Joins: Equijoin x Non-equijoin

```
SELECT e.employee_id, e.last_name, d.department_name
FROM   employees e, departments d
WHERE  e.department_id=d.department_id;
```

```
SELECT e.employee_id, e.first_name, e.last_name, e.hire_date
FROM   employees e, job_history h
WHERE  h.employee_id = 176
AND    e.hire_date BETWEEN h.start_date AND h.end_date;
```

Outer Joins

```
SELECT employee_id, last_name, first_name
FROM   employees LEFT OUTER JOIN departments
ON     (employees.department_id=departments.departments_id);
```

```
SELECT /*+ USE_NL(c o) */ cust_last_name,
       SUM(NVL2(o.customer_id,0,1)) "Count"
  FROM customers c, orders o
 WHERE c.credit_limit > 1000
   AND c.customer_id = o.customer_id(+)
 GROUP BY cust_last_name;
```

```
SELECT cust_last_name, SUM(NVL2(o.customer_id,0,1)) "Count"
  FROM customers c, orders o
 WHERE c.credit_limit > 1000
   AND c.customer_id = o.customer_id(+)
 GROUP BY cust_last_name;
```

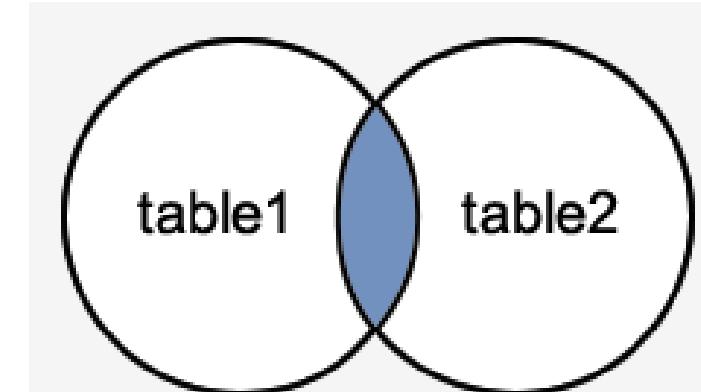
Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time
----------------	------	------	-------	-------------	------

PLAN_TABLE_OUTPUT

0 SELECT STATEMENT					7 (100)	
1 HASH GROUP BY		168	3192	7 (29)	00:00:01	
* 2 HASH JOIN OUTER		318	6042	6 (17)	00:00:01	
* 3 TABLE ACCESS FULL	CUSTOMERS	260	3900	3 (0)	00:00:01	
* 4 TABLE ACCESS FULL	ORDERS	105	420	2 (0)	00:00:01	

Inner x Outer Join

INNER JOIN



LEFT JOIN

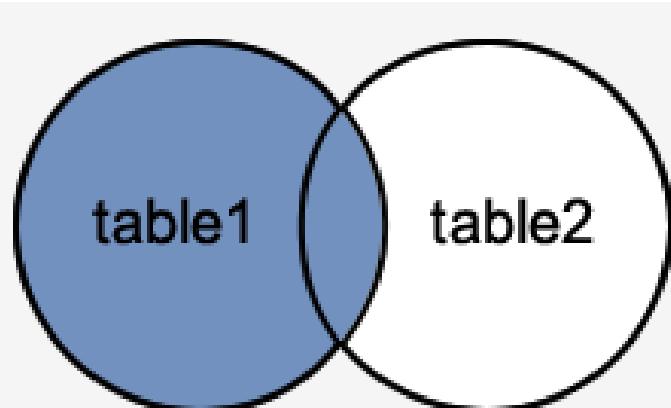


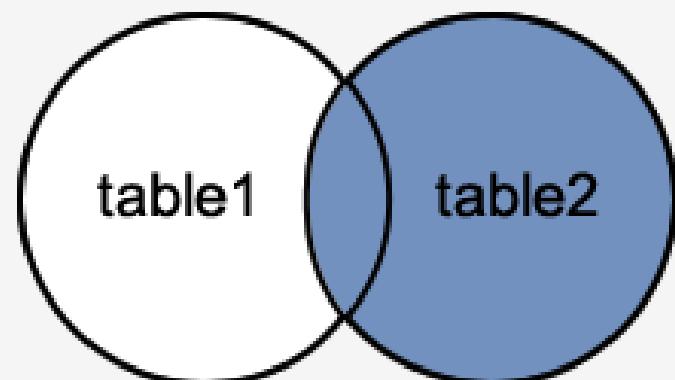
table1

table2

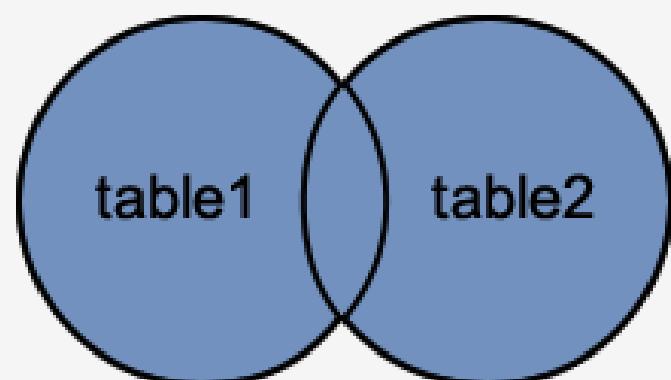
table1

table2

RIGHT JOIN



FULL OUTER JOIN



Semijoins

```
FOR ds1_row IN ds1 LOOP
    match := false;
    FOR ds2_row IN ds2_subquery LOOP
        IF (ds1_row matches ds2_row) THEN
            match := true;
            EXIT -- stop processing second data set when a match is found
        END IF
    END LOOP
    IF (match = true) THEN
        RETURN ds1_row
    END IF
END LOOP
```

Semijoins

```

SELECT department_id, department_name
FROM departments
WHERE EXISTS (SELECT 1
               FROM employees
              WHERE employees.department_id = departments.department_id)

```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				2 (100)	
1	NESTED LOOPS SEMI		11	209	2 (0)	00:00:01
2	TABLE ACCESS FULL	DEPARTMENTS	27	432	2 (0)	00:00:01
*3	INDEX RANGE SCAN	EMP_DEPARTMENT_IX	44	132	0 (0)	

Antijoin

```
FOR ds1_row IN ds1 LOOP
    match := true;
    FOR ds2_row IN ds2 LOOP
        IF (ds1_row matches ds2_row) THEN
            match := false;
            EXIT -- stop processing second data set when a match is found
        END IF
    END LOOP
    IF (match = true) THEN
        RETURN ds1_row
    END IF
END LOOP
```

Antijoin

```
SELECT emp.*  
FROM   emp, dept  
WHERE  emp.deptno = dept.deptno(+)  
AND    dept.deptno IS NULL
```

Execution Plan

Plan hash value: 1543991079

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		14	1400	5 (20)	00:00:01
*	HASH JOIN ANTI		14	1400	5 (20)	00:00:01
2	TABLE ACCESS FULL	EMP	14	1218	2 (0)	00:00:01
3	TABLE ACCESS FULL	DEPT	4	52	2 (0)	00:00:01

Operações N-árias

- *Filter*
 - Elimina linhas indesejadas
- *Concatenation*
 - *OR* ou *Union All*
- *Union All / Union*
 - Soma dos dois conjuntos
- *Intersect*
 - Existe em ambos os conjuntos
- *Minus*
 - Não existe no segundo conjunto

Prática 09: Operadores do Otimizador II

- Objetivos: concluir o estudo de caminhos de acesso (*clusters*) e observar os operadores binários e n-ários.
- *Access Paths:*
 - *Table Clusters*
- Operadores binários:
 - Métodos de *Join*
 - Tipos de *Join*
- Operadores n-ários
 - *Union / Union All*
 - *Intersect*
 - *Minus*

Performance Tuning com Oracle 12c

Aula 10: Estatísticas I

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- DBMS_STATS
- Estatísticas de Tabelas
- Estatísticas de Índices
- Estatísticas de Colunas

Estatísticas do Otimizador

- Descrevem o banco de dados (estrutura) e os objetos do banco de dados
- São utilizadas pelo otimizador para estimar:
 - Seletividade dos predicados
 - Custo de cada plano de execução
 - Método de acesso, ordem de *join*, método de *join*
 - Custos de CPU e I/O
- Manter as estatísticas atualizadas é extremamente importante
 - Coleta automática (*job*)
 - Manualmente com DBMS_STATS

DBMS_STATS

- Expõe a interface para a coleta de estatísticas do otimizador
- **Atenção:** o comando ANALYZE [TABLE | INDEX] também coleta estatísticas, mas **NÃO** deve mais ser utilizado
 - O ANALYZE não guarda histórico de coleta, não respeita parâmetros de coleta e não permite trabalhar com restauração de estatísticas e estatísticas pendentes.
- Métodos:
 - GATHER_*_STATS: coleta de estatísticas
 - SET_*_STATS: modifica estatísticas manualmente
 - GET_*_STATS: recupera estatísticas do objeto (também acessível via *views*)
 - IMPORT_*_STATS e EXPORT_*_STATS: importação e exportação de estatísticas
 - RESTORE_*_STATS: recupera versão anterior das estatísticas

DBMS_STATS: Preferências

- Preferências alteram o comportamento padrão da DBMS_STATS para TABLE, SCHEMA ou DATABASE)
 - SET_*_PREFS: alteram o comportamento padrão de coleta para objeto
 - GET_*_PREFS: recupera valor atual
 - Preferências disponíveis: CASCADE, DEGREE, ESTIMATE_PERCENT, METHOD_OPT, NO_INVALIDATE, GRANULARITY, **PUBLISH**, INCREMENTAL, STALE_PERCENT
- SET_GLOBAL_PREFS: altera comportamento global
 - **GLOBAL_TEMP_TABLE_STATS**: controla estatísticas para tabelas temporárias (SHARED ou SESSION)

Tipos de Estatísticas

- Tabela
 - Número de linhas
 - Número de blocos
 - Tamanho médio da linha
- Índice
 - Nível da árvore (B^* -tree)
 - Chaves distintas
 - Número de folhas
 - Fator de agrupamento (*clustering factor*)
- Sistema
 - Performance de I/O
 - Performance de CPU
- Colunas
 - Básicas: número de valores distintos, número de nulos, comprimento médio, mínimo, máximo
 - Histogramas (distribuição)
 - Estatísticas Extendidas

Estatísticas de Tabela: ALL_TAB_STATISTICS

- Usadas para determinar:
 - Custo de acesso
 - Cardinalidade do *join*
 - Ordem do *join*
- Exemplos:
 - Número de linhas [NUM_ROWS]
 - Número de blocos [BLOCKS]
 - Blocos vazios [EMPTY_BLOCKS]
 - Espaço livre médio [AVG_SPACE]
 - Número de linhas encadeadas (*chained*) [CHAIN_CNT]
 - Comprimento médio das linhas [AVG_ROW_LEN]

Estatísticas de Índice: ALL_IND_STATISTICS

- Usadas para decidir entre *table access full* vs índice
- Exemplos:
 - Nível da árvore (B*-tree level) [LEVEL]
 - Número de folhas [LEAF_BLOCKS]
 - Fator de agrupamento [CLUSTERING_FACTOR]
 - Chaves distintas [DISTINCT_KEYS]
 - Número médio de blocos por chave distinta [AVG_LEAF_BLOCKS_PER_KEY]
 - Número médio de blocos na tabela por chave [AVG_DATA_BLOCKS_PER_KEY]
 - Número de linhas no índice [NUM_ROWS]

Estatísticas de Índices: Clustering Factor

- Mede o grau de organização da tabela em relação a um índice
- Se a organização da tabela estiver próxima do índice, o *clustering factor* será baixo, próximo do número de blocos da tabela.
- Se a organização da tabela não tiver relação com o índice, o *clustering factor* será alto, próximo do número de linhas da tabela.

Block 1	Block 2	Block 3
-----	-----	-----
A A A	B B B	C C C

Block 1	Block 2	Block 3
-----	-----	-----
A B C	A C B	B A C

Estatísticas de Coluna: ALL_TAB_COL_STATISTICS

- Número de valores distintos de uma coluna [NUM_DISTINCT]
- Menor valor [LOW_VALUE]
- Maior valor [HIGH_VALUE]
- Número de nulos [NUM_NULLS]
- Estimativa de seletividade [DENSITY]
- Número de classes do histograma [NUM_BUCKETS]
- Tipo de histograma [HISTOGRAM]

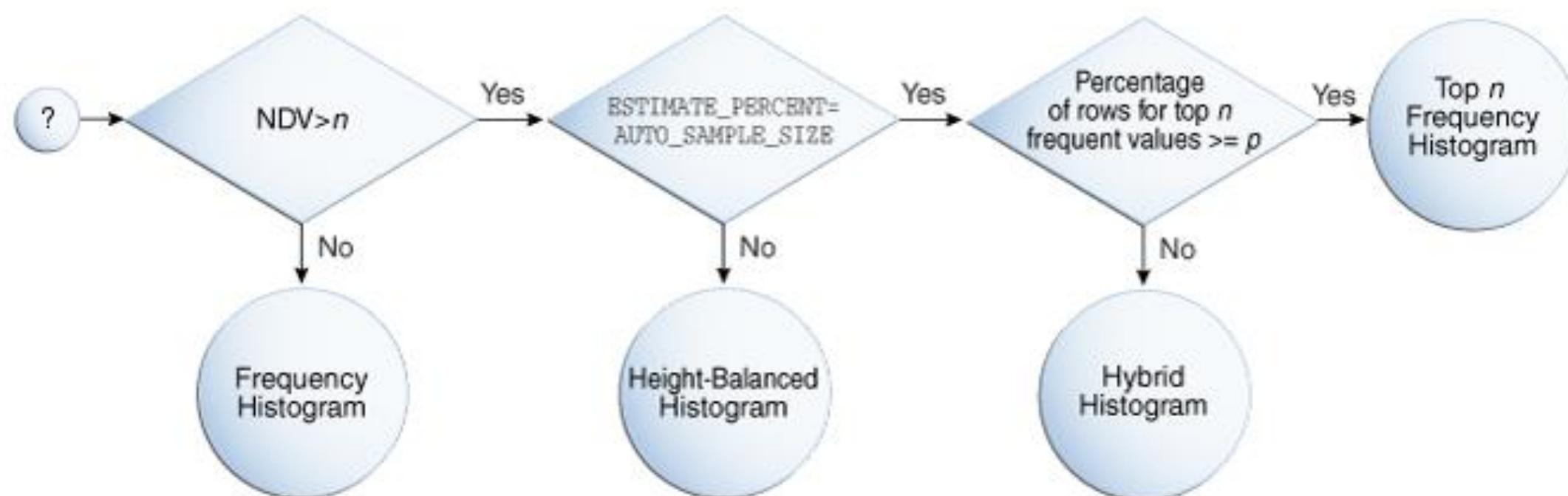
Histogramas

- Por padrão o otimizador assume distribuições uniformes, ou seja, todo valor distinto tem o mesmo número de ocorrências
- Este comportamento pode levar a planos sub-ótimos, exemplo:
 - Tabela empregados, dois tipos: colaborador e gerente
 - É esperado que haja muito mais colaboradores do que gerentes, logo:
 - *select * from empregados where tipo = 'colaborador'* precisa ter um plano diferente de *select * from empregados where tipo = 'gerente'*
- Para isto existem os histogramas

Histogramas

- Guardam informações de distribuição de valores nas colunas
- Ajudam a dar melhores estimativas de seletividade para distribuições não uniformes
- Tipos:
 - Frequency: cada valor distinto tem sua classe
 - Top Frequency [12c]: ignora valores estatisticamente insignificantes
 - Height-balanced [pre-12c]: cria um número de classes com limites variados para que cada classe tenha o mesmo número de valores distintos
 - Hybrid: combina as técnicas de height-balanced com frequency

Escolha do Tipo de Histograma



NDV = Number of distinct values
 n = Number of histogram buckets (default is 254)
 $p = (1 - (1/n)) * 100$

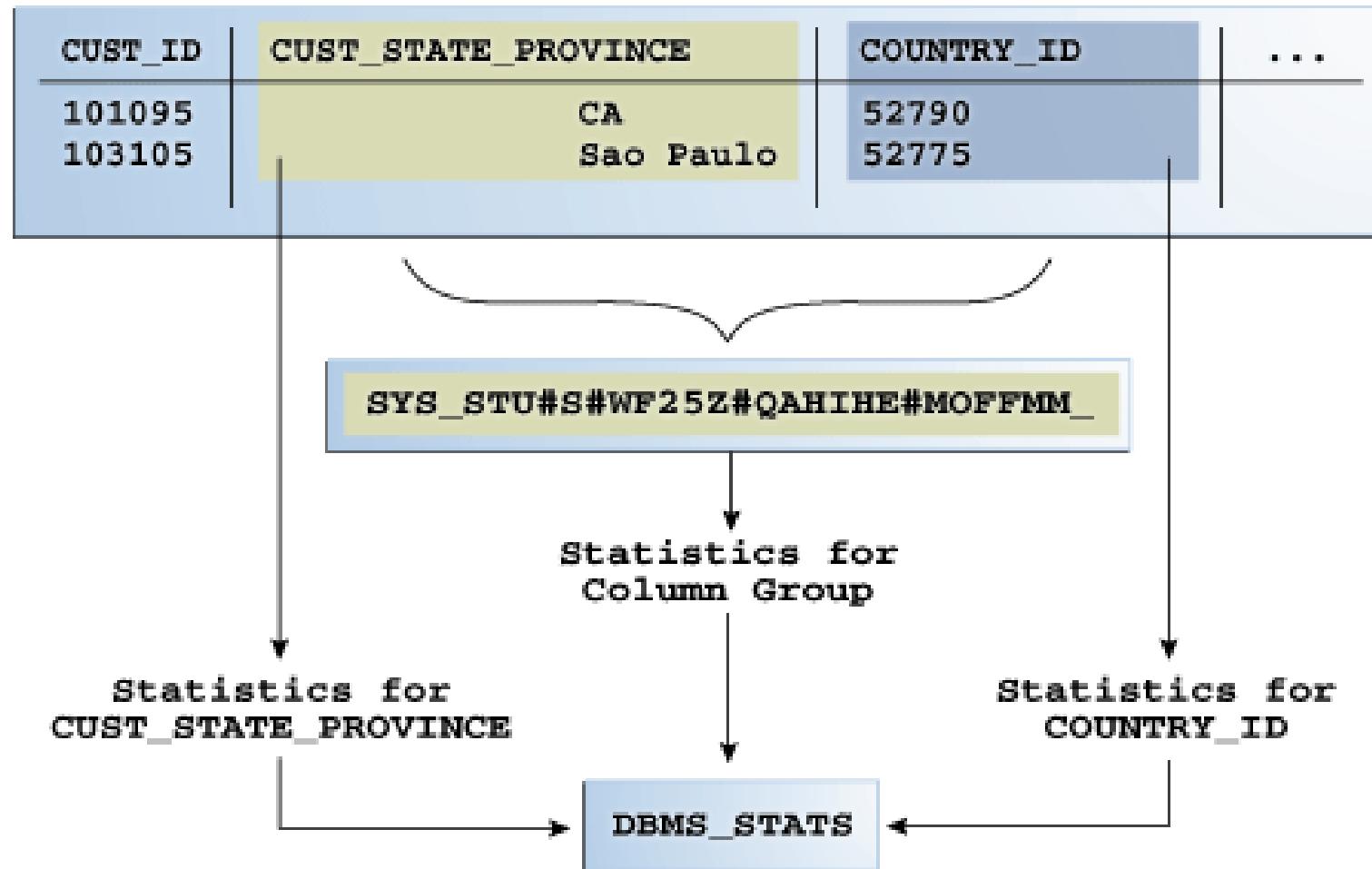
Na prática

- O Oracle decide automaticamente quando é melhor criar histogramas ou não durante a coleta de estatísticas
- Não use histogramas a não ser que eles melhorem substancialmente a performance
- Histogramas não são úteis nos seguintes casos:
 - Colunas que não aparecem na clausula WHERE ou JOIN
 - Colunas com distribuições uniformes
 - Predicados de igualdade com colunas UNIQUE

Estatísticas Estendidas

- Estatísticas de grupos de colunas
- Estatísticas de expressões (funções)

Estatísticas de Grupos de Colunas



Estatísticas de Grupos de Colunas

```

SQL> SELECT count(*)
  2  FROM Customers
  3 WHERE CUST_STATE_PROVINCE = 'CA'
  4 AND COUNTRY_ID = 52790;

COUNT(*)
-----
3341

```

```
SQL> select * from table(dbms_xplan.display_cursor());
```

```
PLAN_TABLE_OUTPUT
```

```
SQL_ID b8rcrw85nx32k, child number 0
```

```
SELECT count(*) FROM Customers WHERE CUST_STATE_PROVINCE = 'CA' AND
COUNTRY_ID = 52790
```

```
Plan hash value: 296924608
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				278 (100)	
1	SORT AGGREGATE		1	16		
* 2	TABLE ACCESS STORAGE FULL	CUSTOMERS	127	2032	278 (1)	00:00:01

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

```
2 - storage(("CUST_STATE_PROVINCE"='CA' AND "COUNTRY_ID"=52790))
      filter(("CUST_STATE_PROVINCE"='CA' AND "COUNTRY_ID"=52790))
```

Estatísticas de Grupos de Colunas

```
SQL> SELECT DBMS_STATS.CREATE_EXTENDED_STATS(null,'customers', '(country_id, cust_state_province)')  
2 FROM dual;  
  
DBMS_STATS.CREATE_EXTENDED_STATS(NULL,'CUSTOMERS','(COUNTRY_ID,CUST_STATE_PROVINCE)')  
-----  
SYS_STUJGVLRVH5USVIDU$XNV4_IR#4  
  
SQL>  
SQL> Exec DBMS_STATS.GATHER_TABLE_STATS(null,'customers');  
  
PL/SQL procedure successfully completed.
```

```
SQL> SELECT column_name, num_distinct, num_nulls, histogram  
2 FROM user_tab_col_statistics  
3 WHERE table_name='CUSTOMERS';
```

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	HISTOGRAM
SYS_STUJGVLRVH5USVIDU\$XNV4_IR#4	145	0	NONE
CUST_ID	55500	0	NONE
CUST_FIRST_NAME	1300	0	NONE
CUST_LAST_NAME	908	0	NONE
CUST_GENDER	2	0	NONE
CUST_YEAR_OF_BIRTH	75	0	NONE
CUST_MARITAL_STATUS	11	17428	NONE

Estatísticas de Grupos de Colunas

```

SQL> SELECT count(*)
  2  FROM Customers
  3 WHERE CUST_STATE_PROVINCE = 'CA'
  4 AND COUNTRY_ID = 52790;

          COUNT(*)
-----
          3341

SQL>
SQL> select * from table(dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
-----
SQL_ID b8rcrw85nx32k, child number 0
-----
SELECT count(*) FROM Customers WHERE CUST_STATE_PROVINCE = 'CA' AND
COUNTRY_ID = 52790

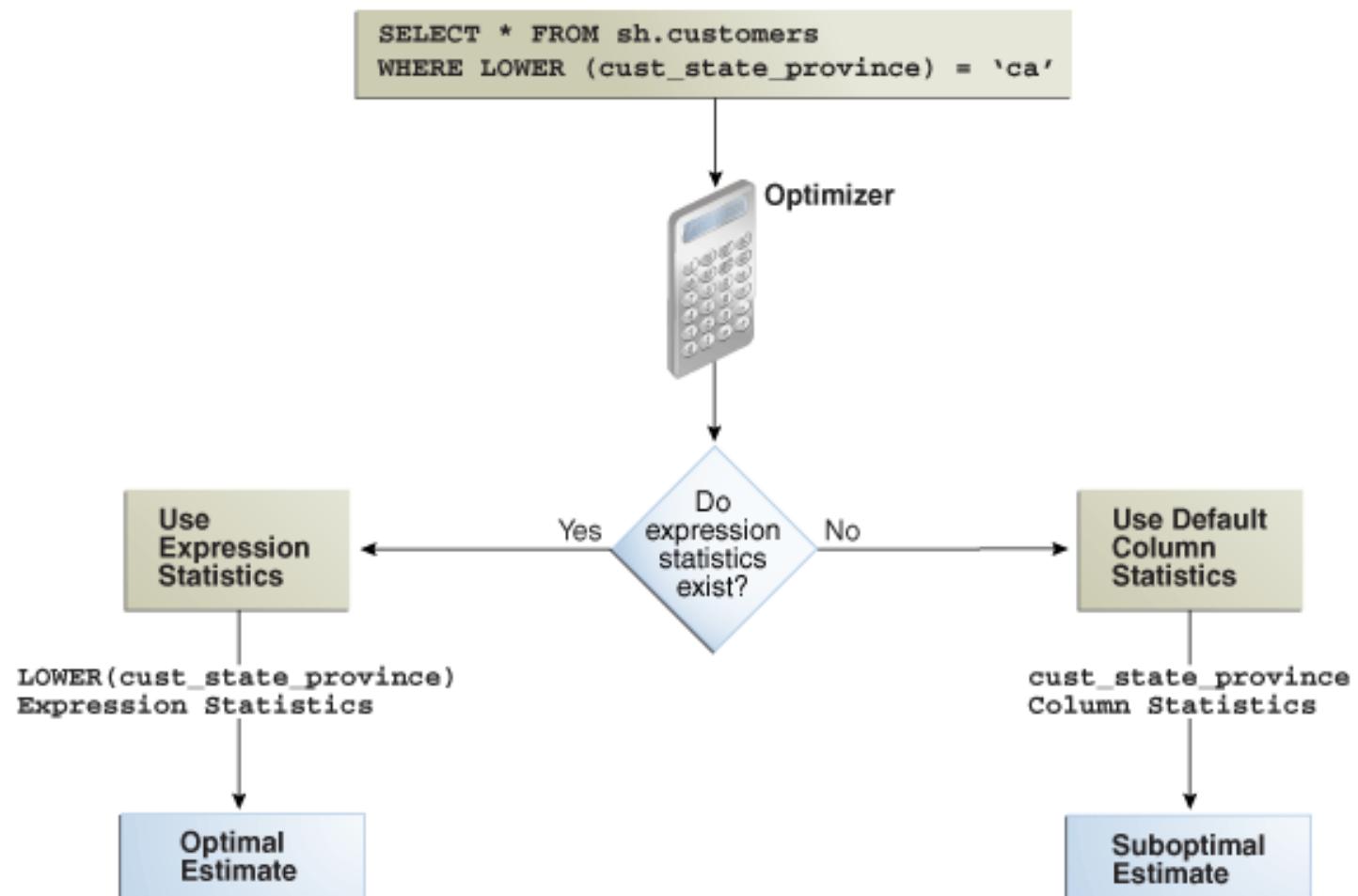
Plan hash value: 296924608

-----| Id | Operation           | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----| 0 | SELECT STATEMENT    |           |       |        | 278 (100)|          |
| 1 |  SORT AGGREGATE     |           |   1   |    16  |            |          |
|* 2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS | 3531  | 56496 | 278   (1)| 00:00:01 |

Predicate Information (identified by operation id):
-----
2 - storage(("CUST_STATE_PROVINCE"='CA' AND "COUNTRY_ID"=52790))
   filter(("CUST_STATE_PROVINCE"='CA' AND "COUNTRY_ID"=52790))

```

Estatísticas de Expressões



Estatísticas de Expressões

```
sys@PROD> EXPLAIN PLAN FOR
  2  SELECT * FROM sh.customers WHERE LOWER(cust_state_province)='ca';
Explained.
```

```
sys@PROD> select * from table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 2008213504

	Id	Operation		Name		Rows		Bytes		Cost	(%CPU)		Time	
	0	SELECT STATEMENT				555		108K		406	(1)		00:00:05	
*	1	TABLE ACCESS FULL	CUSTOMERS			555		108K		406	(1)		00:00:05	

Predicate Information (identified by operation id):

1 - filter(LOWER("CUST_STATE_PROVINCE")='ca')

Estatísticas de Expressões

```
BEGIN
    DBMS_STATS.GATHER_TABLE_STATS(
        'sh'
    ,   'customers'
    ,   method_opt => 'FOR ALL COLUMNS SIZE SKEWONLY FOR COLUMNS
                            (LOWER(cust_state_province)) SIZE SKEWONLY'
    );
END;
```

```
SELECT e.EXTENSION expression, t.NUM_DISTINCT, t.HISTOGRAM
FROM   USER_STAT_EXTENSIONS e, USER_TAB_COL_STATISTICS t
WHERE  e.EXTENSION_NAME=t.COLUMN_NAME
AND    e.TABLE_NAME=t.TABLE_NAME
AND    t.TABLE_NAME='CUSTOMERS';
```

EXPRESSION	NUM_DISTINCT	HISTOGRAM
<hr/>		
(LOWER ("CUST_STATE_PROVINCE"))	145	FREQUENCY

Prática 10: Estatísticas I

- Objetivos: explorar os principais tipos de estatísticas do otimizador e suas formas de coleta.
- Estatísticas de Tabelas
- Estatísticas de Índices
- Estatísticas de Colunas
- Histogramas
- Estatísticas Estendidas
 - Colunas
 - Expressões

Performance Tuning com Oracle 12c

Aula 11: Estatísticas II

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- Estatísticas de Sistema
- Gerenciamento de Estatísticas
- Estatísticas de Tabelas Temporárias

Estatísticas de Sistema

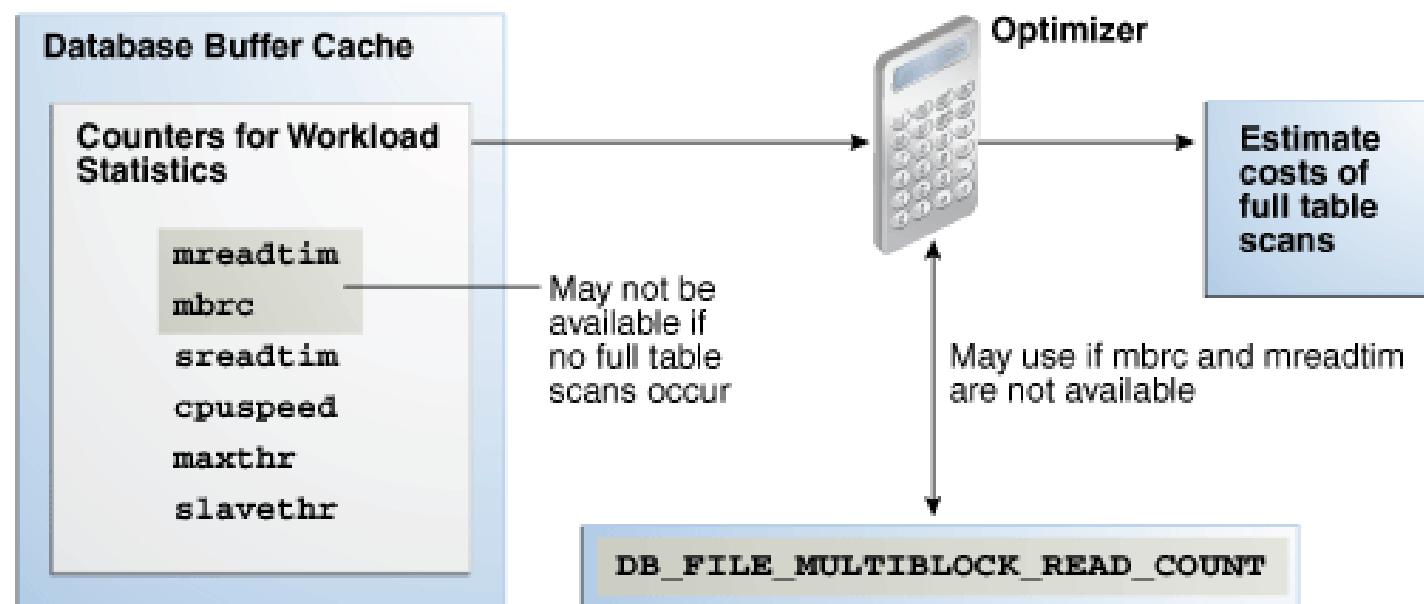
- Habilitam o CBO a utilizar características de CPU e I/O do banco de dados
- Se não coletadas o sistema trabalha com um valor default que geralmente não é adequado
- **Não são coletadas automaticamente**
- Como coletar / modificar:
 - DBMS_STATS.GATHER_SYSTEM_STATS
 - DBMS_STATS.SET_SYSTEM_STATS
 - SYS.AUX_STATS\$ [não recomendado]
- Modos de coleta:
 - NOWORKLOAD e INTERVAL [ou START e STOP]

Estatísticas de Sistema

- cpuspeed: média de ciclos de CPU por segundo
- cpuspeednw: igual a CPUSPEED, NW = no workload
- lseektim: tempo de posicionamento da cabeça de leitura dos discos
- lotfrspeed: taxa de transferência em uma requisição de leitura [single read]
- maxthr: máxima taxa de transferência de I/O [throughput]
- slavethr: taxa de transferência do parallel execution server
- sreadtim: single block read time – tempo médio para leitura de um bloco
- mreadtim: multiblock read time
- mbrc: multiblock read count – número de blocos lidos na leitura multiblock

Estatísticas de Sistema

- Quando coletar?
 - Logo após a instalação (NOWORKLOAD)
 - Pelo menos uma vez durante os períodos de pico (INTERVAL ou START/STOP)
 - Após toda e qualquer mudança significativa de hardware



O que Fazer Quando as Estatísticas Pioram a Performance

- Restaurar estatísticas anteriores:
 - dbms_stats.restore_*_stats
 - dba_optstat_operations: lista todas as coletas de estatística
 - dba_tab_stats_history: histórico de modificações de estatísticas
 - Período de retenção: 31 dias por padrão. [dbms_stats.alter_stats_history_retention]
- Setar manualmente as estatísticas desejadas:
 - dbms_stats.set_*_stats
 - dbms_stats.import_*_stats
- Travar as estatísticas:
 - dbms_stats.lock_*_stats

Estatísticas de Tabelas Temporárias

- A partir do Oracle 12c existem dois tipos de estatísticas de tabelas temporárias:
 - SHARED: Compartilhadas por todas as sessões (estatísticas globais)
 - SESSION: Cada sessão tem a sua
- Modo determinado pela preferência GLOBAL_TEMP_TABLE_STATS
 - dbms_stats.set_global_prefs(pname => 'GLOBAL_TEMP_TABLE_STATS', pvalue => 'SESSION');
- Como coletar: DBMS_STATS.GATHER_TABLE_STATS
 - **Cuidado com o COMMIT implícito! (ON COMMIT DELETE ROWS)**
- Alternativa: DBMS_STATS.SET_TABLE_STATS

O que o CBO faz quando não tem estatísticas?

- **Dynamic Sampling:** amostragem durante a elaboração do plano
- Se o *dynamic sampling* estiver desabilitado:

Selectivity:	
Equality	1%
Inequality	5%
Other predicates	5%
Table row length	20
# of index leaf blocks	25
# of distinct values	100
Table cardinality	100
Remote table cardinality	2000

Prática 11: Estatísticas II

- Objetivos: coletar estatísticas de sistema e observar o seu impacto.
Gerenciar o ciclo de vida das estatísticas do otimizador.
- Estatísticas de sistema
 - NOWORKLOAD
 - WORKLOAD (INTERVAL)
- Gerenciamento de Estatísticas
 - PUBLISH e PENDING STATS
- Estatísticas de Tabelas Temporárias
 - SESSION x SHARED

Performance Tuning com Oracle 12c

Aula 12: Curores

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- *Cursor Sharing*
- *Bind Peeking*
- *Adaptive Cursor Sharing*

Compartilhamento de Cursos

- Ao executar um cursor, o Oracle primeiro compara o cursor com aqueles que estão salvos na *library cache* para ver se há correspondência
- Se houver, ele reaproveita o plano de execução (*soft parse*)
- Se não, ele elabora um novo plano de execução (*hard parse*)
- A comparação é textual/literal:
 - Maiúsculas / minúsculas
 - Espaçamento (espaço, *tab*, *enter*)
 - Nomes de variáveis

Compartilhamento de Cursos

```
SELECT * FROM employees;  
SELECT * FROM Employees;  
SELECT * FROM employees;
```

```
SELECT count(1) FROM employees WHERE manager_id = 121;  
SELECT count(1) FROM employees WHERE manager_id = 247;
```

```
SELECT * FROM employees WHERE department_id = :department_id;  
SELECT * FROM employees WHERE department_id = :dept_id;
```

Compartilhamento de Cursos

- Para aumentar o compartilhamento:
 - Sempre usar *bind variables*
 - Estabelecer um padrão de codificação
 - Whitespace
 - CASE
 - Nomenclatura para *binds*
- No PL/SQL
 - Automaticamente todas as variáveis são transformadas em *bind variables*
 - Literais não são transformados (exceto com CURSOR_SHARING = FORCE)
 - Cuidado com SQL dinâmico concatenando valores (usar *binds*!!!)

Child Cursors

- Mesmo quando a comparação resulta em **igualdade**, pode ser que eles não sejam compartilhados:
 - Ex.: mesma tabela, esquemas diferentes
- Quando um SQL é igual mas não é possível compartilhá-lo, é criado um *child cursor*. O SQL original é chamado *parent cursor*.
- v\$sqlarea: contém uma linha para cada *parent cursor*
- v\$sql: contém uma coluna *child#*
- v\$sql_shared_cursor: contém o **motivo** pelo qual o cursor não foi compartilhado

Adaptive Cursor Sharing

- Aplicável em consultas que utilizam *bind variables* (BIND SENSITIVE)
- O otimizador gera um plano inicial (child# 0)
- Enquanto as consultas produzirem resultados consistentes com o plano nada muda
- No primeiro momento que o Oracle “errar” a estimativa ele marca o cursor para ser avaliado
- Na próxima execução o Oracle gera um novo plano para aquele valor do parâmetro cuja estimativa estava errada (child# 1, 2, 3, ...)
- A partir deste momento o cursor se torna BIND AWARE.

Prática 12: Compartilhamento de Cursos

- Objetivos: Observar na prática o mecanismo de geração de *child cursors* e identificar o motivo para a geração.
- Views
 - v\$sql
 - v\$sqlarea
 - v\$sql_shared_cursor
- Diagnosticando *hard parses*:
 - v\$mystats
- *Adaptive cursor sharing*

Performance Tuning com Oracle 12c

Aula 13: *Trace* em Aplicações

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- Tipos de *Trace*
- Ferramentas de Análise
- Instrumentação de Aplicações

Trace de Aplicações

- O banco Oracle é todo instrumentado para que seja possível diagnosticar todos os tipos de problemas
- O Trace é a ferramenta que permite capturar o funcionamento interno do banco de dados e explicar **porque** ele tomou as decisões que tomou
- Para diagnosticar aplicações é possível criar traces por:
 - Identificador de cliente [Client identifier]
 - Serviço [service]
 - Módulo [module]
 - Ação [action]
 - Sessão [session]
 - Instância [instance]

Serviços

- É uma forma de agrupar sessões que realizam o mesmo tipo de trabalho
- Encapsula o conceito de instância: o cliente conecta no serviço, não na instância
 - Permite alta disponibilidade através da migração do serviço.
- Alocação dinâmica de serviços para instâncias
 - Permite elasticidade de workloads.
- Pode ser utilizado como identificador para capturar traces
- No TNSNAMES.ORA: SID => SERVICE_NAME

Trace de Serviços

- Pode ser qualificado por:
 - MODULE: bloco funcional dentro de uma aplicação (serviço)
 - ACTION: ação ou operação dentro de um módulo
 - CLIENT_IDENTIFIER: usuário, baseado no *logon (trigger)*
- Para preencher estes campos:
 - DBMS_APPLICATION_INFO [SET_MODULE, SET_ACTION]
 - DBMS_SESSION [SET_IDENTIFIER]
- Para ativar o trace:
 - DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE / DISABLE

Outros tipos de *Trace*

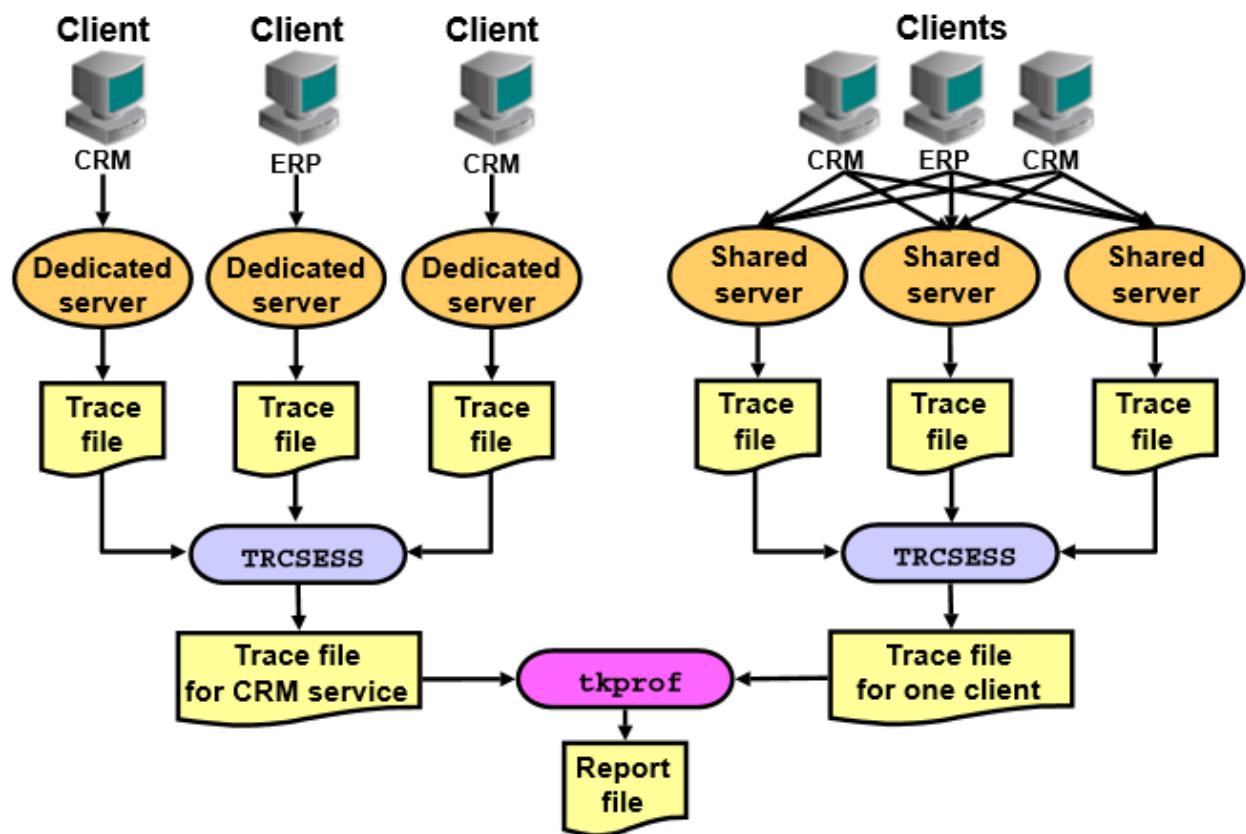
- *Trace* de cliente:
 - DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE / DISABLE
- *Trace* de sessão:
 - DBMS_MONITOR.SESSION_TRACE_ENABLE / DISABLE
- *Trace* de instância:
 - DBMS_MONITOR.DATABASE_TRACE_ENABLE / DISABLE

Coleta de Estatísticas de Execução

- Por *Client ID*:
 - DBMS_MONITOR.CLIENT_ID_STAT_ENABLE / DISABLE
 - v\$client_stats
- Por Serviço, Modulo e Ação:
 - DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE / DISABLE
 - v\$serv_mod_act_stats

Processando o Arquivo de *Trace*

- TKPROF: converte arquivos de *trace* em relatórios
- TRCSESS: consolida vários arquivos de *trace* em um só



Prática 13: *Trace* em Aplicações

- Objetivo: familiarizar-se com as ferramentas disponíveis para ativação e interpretação de arquivos de *trace*.
- DBMS_MONITOR
- DBMS_APPLICATION_INFO
- DBMS_SESSION
- Tkprof
- Trcsess

Performance Tuning com Oracle 12c

Aula 14: PL/SQL I

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

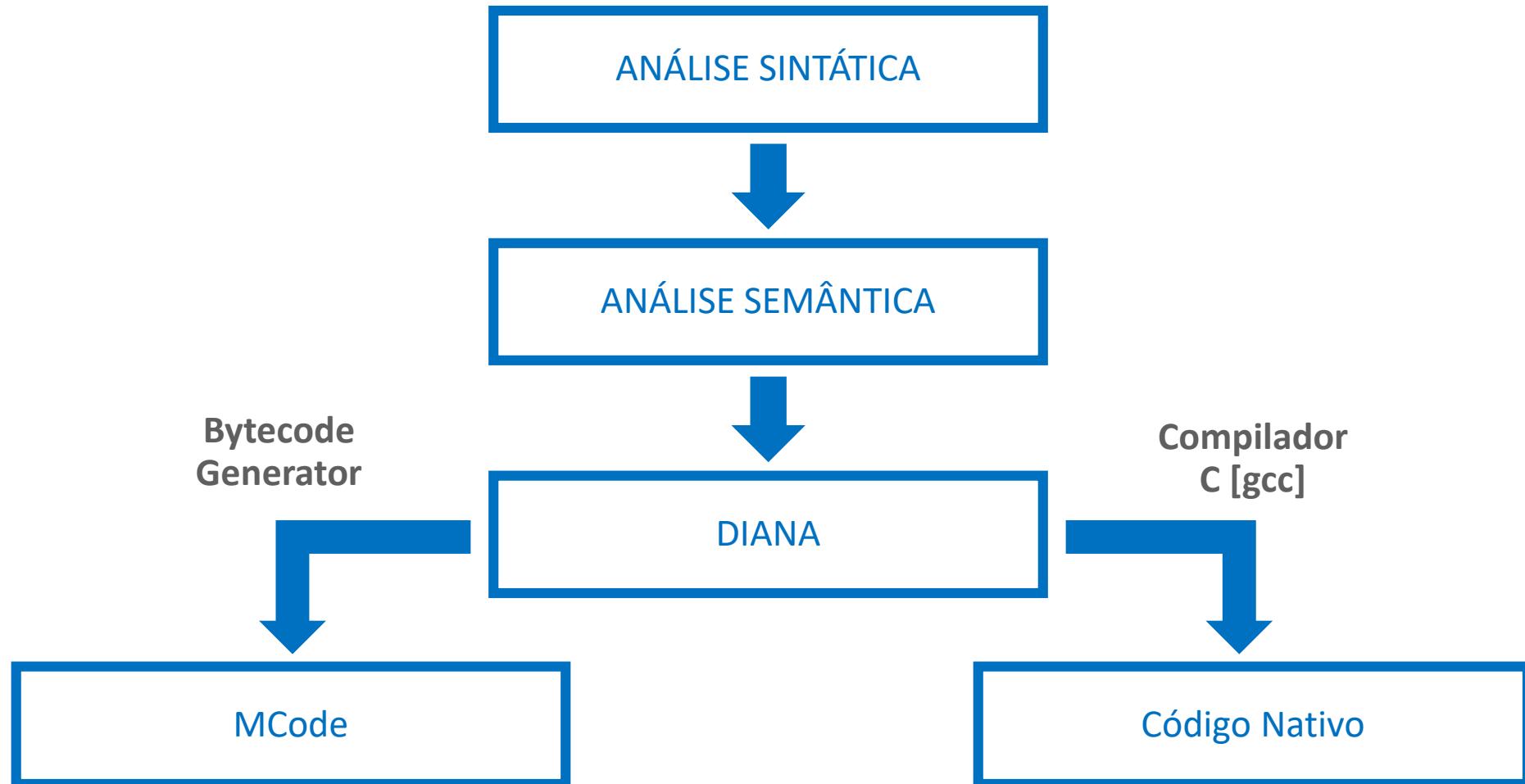
Objetivos

- Máquina Virtual PL/SQL
- Compilação Nativa
- PL/SQL Optimize Level

PL/SQL

- Extensão da linguagem SQL para incorporar características procedurais
- Derivada da linguagem ADA [por sua vez derivada do Pascal]
- Por padrão é uma linguagem interpretada
- PL/SQL Virtual Machine (PVM)
 - máquina virtual responsável por interpretar o Mcode [código intermediário]
- Porém, também é possível compilar o código PL/SQL em código de máquina com um compilador C
 - Compilação nativa [**não** é o *default!*]

Compilação do Código PL/SQL



Execução

- Mcode:
 - Código é carregado para a SGA [shared pool]
 - Executado pela PL/SQL runtime engine [PVM]
- Código nativo
 - Código é carregado para a PGA
 - Executado diretamente

Parâmetros do Compilador

- PLSQL_CODE_TYPE: Controla o tipo de código (nativo ou interpretado)
- PLSQL_OPTIMIZE_LEVEL: nível de otimização (0 a 3)
 - 0: sem otimização, padrão 9i
 - 1: otimização mínima – remove códigos desnecessários, ativa DEBUG
 - 2: remove invariantes de loops, converte FOR LOOPS em BULK COLLECT com LIMIT 100
 - 3: tenta transformar chamadas de função em código INLINE
- PLSQL_DEBUG: descontinuado no 12c, usar PLSQL_OPTIMIZE_LEVEL = 1
- View: USER_PLSQL_OBJECT_SETTINGS

Compilação Nativa

- Controlada por instância, sessão ou objeto:
- Instância:
 - `ALTER SYSTEM SET PLSQL_CODE_TYPE = 'NATIVE'`
 - `ALTER SYSTEM SET PLSQL_CODE_TYPE = 'INTERPRETED'`
- Sessão:
 - `ALTER SESSION SET PLSQL_CODE_TYPE='NATIVE';`
 - `ALTER PROCEDURE p_teste COMPILE;` (vale o *setup* da sessão)
- Objeto:
 - `ALTER PROCEDURE p_teste COMPILE PLSQL_CODE_TYPE = 'NATIVE';`

Compilação Nativa

- Alterar o parâmetro PLSQL_CODE_TYPE **não** recompila os objetos.
- É necessário recompilar cada um manualmente.
 - ALTER xxx COMPILEyyy
- Para alterar **todo o banco** para nativo por padrão:
 - STARTUP em UPGRADE mode
 - Seta todos os objetos para nativo: \$ORACLE_HOME/rdbms/admin/dbmsupgnv.sql
 - Recompila todos os objetos: \$ORACLE_HOME/rdbms/admin/utlrp.sql

Tipos de Dados Nativos

- Estes tipos de dados utilizam aritmética via *hardware*:
 - PLS_INTEGER: inteiro de 32 bits com sinal
 - Limites: -2.147.483.648 a 2.147.483.647
 - Sinônimo para BINARY_INTEGER (a partir da 10g)
 - Lança exceção quando ocorre *overflow*
 - SIMPLE_INTEGER: igual ao PLS_INTEGER mas...
 - não permite nulos
 - não checa por *overflow*.
 - Utiliza aritmética complemento de dois.
 - BINARY_FLOAT: ponto flutuante simples precisão (IEEE 754)
 - BINARY_DOUBLE: ponto flutuante dupla precisão (IEEE 754)
 - SIMPLE_FLOAT e SIMPLE_DOUBLE: não permitem nulos (melhor performance)

Processamento Nativo: Quando usar?

- Compilação nativa: sempre?
 - Maior benefício se recompilado o banco inteiro (Oracle recomenda)
 - Pode ser usado por procedure / function / package
- Tipos de dados nativos:
 - Com compilação nativa (benefícios não são tão grandes quando interpretados)
 - Processos com cálculos intensos
 - Processos que a precisão permite

Prática 14: Processamento Nativo

- Objetivos: observar os impactos da compilação nativa na performance de códigos PL/SQL.
- Compilação Nativa
- Native Data Types
- Nível de Otimização

Performance Tuning com Oracle 12c

Aula 15: PL/SQL II

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- Otimização de Trocas de Contexto
- Processamento em Massa
- BULK COLLECT
- FORALL

Trocas de Contexto

- Todo código SQL em Oracle é executado na SQL Execution Engine
- Sempre que um código PL/SQL chama um código SQL, ou vice-versa, ocorre a chamada **troca de contexto** (*context switch*).
- Exemplos:
 - Consulta SQL chama função PL/SQL
 - *Procedure* PL/SQL chama consulta SQL
- As trocas de contexto são processos caros para o banco e precisam ser evitadas:
 - Excessivas trocas de contexto podem causar sérios problemas de performance!

Otimizando Trocas de Contexto

- Reduzindo chamadas de função SQL dentro do PL/SQL
 - A maioria das funções SQL tem contrapartes em PL/SQL
 - Cuidado com **select into**
 - Exceção notável: REVERSE
- Utilizando o RESULT_CACHE
- Utilizando funções DETERMINISTIC
- Utilizando a técnica de *Scalar Subquery Caching*
- Índices baseados em função
- Processamento em massa com *arrays*

Processamento em Massa com *Arrays*

- Para leitura:
 - `SELECT ... BULK COLLECT INTO meu_array;`
- Para gravação:
 - `FORALL i IN meu_array.first .. meu_array.last [...]`
 - `INSERT, UPDATE, DELETE`
- O objetivo destas operações é reduzir o número de trocas de contexto entre SQL e PL/SQL.
- Ao trabalhar com *arrays*, estamos utilizando memória da PGA
- Sempre que possível tentar resolver tudo no SQL diretamente

Exemplo

```

1 CREATE OR REPLACE PROCEDURE increase_salary (
2     department_id_in    IN employees.department_id%TYPE,
3     increase_pct_in    IN NUMBER)
4 IS
5     TYPE employee_ids_t IS TABLE OF employees.employee_id%TYPE
6         INDEX BY PLS_INTEGER;
7     l_employee_ids    employee_ids_t;
8     l_eligible_ids    employee_ids_t;
9
10    l_eligible        BOOLEAN;
11
12    BEGIN
13        SELECT employee_id
14            BULK COLLECT INTO l_employee_ids
15            FROM employees
16            WHERE department_id = increase_salary.department_id_in;
17
18        FOR indx IN 1 .. l_employee_ids.COUNT
19            LOOP
20                check_eligibility (l_employee_ids (indx),
21                                    increase_pct_in,
22                                    l_eligible);
23
24                IF l_eligible
25                    THEN
26                        l_eligible_ids (l_eligible_ids.COUNT + 1) :=
27                            l_employee_ids (indx);
28                    END IF;
29
30        FORALL indx IN 1 .. l_eligible_ids.COUNT
31            UPDATE employees emp
32                SET emp.salary =
33                    emp.salary
34                    + emp.salary * increase_salary.increase_pct_in
35                    WHERE emp.employee_id = l_eligible_ids (indx);
36    END increase_salary;

```

Bulk Collect

- Basta usar as palavras-chave BULK COLLECT antes da palavra INTO do SELECT.
- Pode ser usado com três tipos de coleções:
 - *Associative Arrays*
 - *Nested Tables*
 - *VARRAYs*
- Pode inserir em uma coleção de *records* ou várias coleções individuais
- A coleção é populada de forma densa (começando do 1)
- Se a query resultar em nada, a coleção é limpa de todos os seus elementos

Bulk Collect: Múltiplas Colunas e Cláusula Limit

```

DECLARE
  TYPE two_cols_rt IS RECORD
  (
    employee_id  employees.employee_id%TYPE,
    salary        employees.salary%TYPE
  );
  TYPE employee_info_t IS TABLE OF two_cols_rt;
  l_employees   employee_info_t;
BEGIN
  SELECT employee_id, salary
    BULK COLLECT INTO l_employees
      FROM employees
     WHERE department_id = 10;
END;
  
```

```

DECLARE
  c_limit PLS_INTEGER := 100;
  CURSOR employees_cur
  IS
    SELECT employee_id
      FROM employees
     WHERE department_id = department_id_in;
  TYPE employee_ids_t IS TABLE OF
    employees.employee_id%TYPE;
  l_employee_ids   employee_ids_t;
BEGIN
  OPEN employees_cur;
  LOOP
    FETCH employees_cur
    BULK COLLECT INTO l_employee_ids
    LIMIT c_limit;
    EXIT WHEN l_employee_ids.COUNT = 0;
  END LOOP;
END;
  
```

Forall

- Sempre que houver um DML dentro de um *loop*, considerar o FORALL
- Cada FORALL comporta apenas uma operação.
 - Se o código dentro do *loop* tem mais de uma operação DML, criar um FORALL para cada uma delas
- FORALL por padrão opera com coleções densas (índice IN menor .. maior)
- Para coleções esparsas, utilizar a cláusula INDICES OF (ou VALUES OF)
- Exceções: utilizar a cláusula SAVE EXCEPTIONS e capturar o ORA-24381
 - O array SQL%BULK_EXCEPTIONS vai conter todas as exceções.

Forall: Exceções e Coleções Esparsas

```

BEGIN
  FORALL indx IN 1 .. l_eligible_ids.COUNT SAVE EXCEPTIONS
    UPDATE employees emp
      SET emp.salary =
        emp.salary + emp.salary * increase_pct_in
      WHERE emp.employee_id = l_eligible_ids (indx);
EXCEPTION
  WHEN OTHERS
  THEN
    IF SQLCODE = -24381
    THEN
      FOR indx IN 1 .. SQL%BULK_EXCEPTIONS.COUNT
      LOOP
        DBMS_OUTPUT.put_line (
          SQL%BULK_EXCEPTIONS (indx).ERROR_INDEX
          || ': '
          || SQL%BULK_EXCEPTIONS (indx).ERROR_CODE);
      END LOOP;
    ELSE
      RAISE;
    END IF;
END increase_salary;

FOR indx IN 1 .. l_employee_ids.COUNT
LOOP
  check_eligibility (l_employee_ids (indx),
                      increase_pct_in,
                      l_eligible);

  IF NOT l_eligible
  THEN
    l_employee_ids.delete (indx);
  END IF;
END LOOP;

FORALL indx IN INDICES OF l_employee_ids
  UPDATE employees emp
    SET emp.salary =
      emp.salary +
      emp.salary *
      increase_salary.increase_pct_in
    WHERE emp.employee_id =
      l_employee_ids (indx);

```

Prática 15: *Bulk Collect e Forall*

- Objetivo: aprender a utilizar o processamento com BULK COLLECT e FORALL e observar o seu impacto na performance de processos.
- *Select Bulk Collect Into*
- *Fetch Bulk Collect*
- *Forall*
- *Forall Indices Of*
- *Forall Save Exceptions*

Performance Tuning com Oracle 12c

Aula 16: Técnicas de *Cache*

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- *Result Cache*
- *Scalar Subquery Cache*
- Funções Determinísticas
- *User Defined Functions*

Result Cache

- Armazena o resultado de operações custosas
- Três tipos:
 - *Query result cache*
 - *PL/SQL result cache*
 - *Client result cache*
- Ideal para operações longas que retornam resultados pequenos
 - Operações de agregação (SUM, COUNT, ...)
 - *Joins* entre tabelas grandes mas que retornam poucas linhas

Configuração

- **result_cache_mode:** [ALTER SYSTEM | ALTER SESSION]
 - MANUAL: cache ativado por *hint* [*hint*: RESULT_CACHE]
 - FORCE: todo SQL é automaticamente cacheado [*hint*: NO_RESULT_CACHE]
- **result_cache_max_size:** tamanho do *cache* em *bytes*
 - Alocado na *shared pool*
 - *Flush* da *shared pool* não limpa o *cache* (DBMS_RESULT_CACHE.FLUSH)
- **result_cache_max_result:** percentual máximo do *cache* que uma *query* pode ocupar (*default* 5%)
- **result_cache_remote_expiration:** tempo (minutos) para expirar *caches* remotos (via DB_LINK). 0 = desativado

SQL Result Cache

```
SQL> SELECT /*+ RESULT_CACHE */
  2      p.prod_name
  3 ,    SUM(s.amount_sold)  AS total_revenue
  4 ,    SUM(s.quantity_sold) AS total_sales
  5 FROM sales s
  6 ,  products p
  7 WHERE s.prod_id = p.prod_id
  8 GROUP BY
  9      p.prod_name;
```

Execution Plan

```
Plan hash value: 504757596
```

Id	Operation	Name	Rows	... Pstart Pstop
0	SELECT STATEMENT		71	...
1	RESULT CACHE	091zc7mvn8ums36mbd2ggac4h0	1	...
2	HASH GROUP BY		71	...
3	HASH JOIN		72	...
4	VIEW	VW_GBC_5	72	...
5	HASH GROUP BY		72	...
6	PARTITION RANGE ALL		918K	... 1 28
7	TABLE ACCESS FULL	SALES	918K	... 1 28
8	TABLE ACCESS FULL	PRODUCTS	72	...

Predicate Information (identified by operation id):

```
3 - access("ITEM_1"="P"."PROD_ID")
```

Result Cache Information (identified by operation id):

```
1 - column-count=3; dependencies=(SH.SALES, SH.PRODUCTS); parameters=(nls); name="SELECT /*+ RESULT_CACHE */ p.prod_name , SUM(s.amount_sold) AS total_revenue , SUM(s.quantity_sold) AS total_"
```

Controlando o *Result Cache*

- DBMS_RESULT_CACHE
 - FLUSH: limpa todo o *cache*
 - INVALIDATE: seletivamente invalida objetos e dependentes
 - MEMORY_REPORT: informações sobre o uso
 - STATUS: ativado ou não
- Views:
 - V\$result_cache_dependency
 - V\$result_cache_memory
 - V\$result_cache_objects
 - V\$result_cache_statistics

PL/SQL Result Cache

- Compartilha as estruturas de memória
- Não aparece no plano de execução

```
SQL> CREATE FUNCTION format_customer_name (
  2          p_first_name IN VARCHAR2,
  3          p_last_name  IN VARCHAR2
  4      ) RETURN VARCHAR2 RESULT_CACHE IS
  5 BEGIN
  6     counter.increment();
  7     RETURN p_first_name || ' ' || p_last_name;
  8 END format_customer_name;
  9 /
```

```
SQL> EXPLAIN PLAN SET STATEMENT_ID = 'PLSQL_CACHE'
  2 FOR
  3   SELECT c.cust_id
  4 ,      format_customer_name(
  5           c.cust_first_name, c.cust_last_name
  6       ) AS cust_name
  7   FROM   customers c;

Explained.

SQL> SELECT *
  2 FROM   TABLE(DBMS_XPLAN.DISPLAY(null,'PLSQL_CACHE'));
  PLAN_TABLE_OUTPUT
-----
Plan hash value: 2008213504

-----
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU) | Time   |
|---|---|---|---|---|---|---|
| 0  | SELECT STATEMENT |             | 55500 | 1083K| 405  (1)| 00:00:05 |
| 1  | TABLE ACCESS FULL| CUSTOMERS | 55500 | 1083K| 405  (1)| 00:00:05 |
-----
```

Outras Técnicas de *Cache*

- Funções determinísticas
- *Scalar subquery cache*
- *Cache manual*:
 - Carregar os dados em um *array* e encapsular o código numa *package*
 - Variável global = “*Package State*”

Funções Determinísticas

- Uma função determinística é uma função que, **sempre, para a mesma entrada, retorna a mesma saída.**
- Exemplo:
 - `SELECT UPPER('a') FROM DUAL;`
 - Sempre que o parâmetro de entrada for 'a' a saída vai ser 'A'.
- Exemplo de função **não-determinística**:
 - `SELECT SYSDATE FROM DUAL;`
 - Duas chamadas consecutivas podem retornar valores diferentes, mesmo sem mudar nenhum parâmetro da função (mesma entrada)

Funções Determinísticas

- Para criar um índice baseado em função com uma função do usuário é obrigatório que ela seja determinística.
- Como fazer: palavra-chave DETERMINISTIC
CREATE FUNCTION funcao(parametro IN NUMBER)
AS RETURN NUMBER **DETERMINISTIC**
BEGIN RETURN parametro; END;
- Este fato **pode** ser usado pelo Oracle para otimizar o número de chamadas de uma função.
 - Dentro da mesma sessão e do mesmo ciclo de FETCH
 - Altamente dependente do ARRAY SIZE / TAMANHO do FETCH do client

Scalar Subquery Caching

- Técnica que se aproveita do mecanismo interno do Oracle para *caching* de *subquerys*.
- Consultas na forma:
 - `SELECT A.* , F(X) FROM T`
 - Executam uma vez por linha
- Consultas na forma:
 - `SELECT A.* , (SELECT F(X) FROM DUAL) FROM T`
 - Executam uma vez por valor distinto de X

User Defined Functions (UDF)

- No **Oracle 12c** é possível sinalizar funções que são predominantemente chamadas por consultas SQL.
- Com base nesta sinalização o Oracle pode tomar algumas decisões para otimizar o número de chamadas de função.
 - Até o momento não foi esclarecido como ele faz isso.
- Para ativar este recurso: PRAGMA UDF

```
CREATE OR REPLACE FUNCTION normal_function(p_id IN NUMBER) RETURN NUMBER IS
    PRAGMA UDF;
BEGIN
    RETURN p_id;
END;
/
```

Inline PL/SQL

- Declaração de código PL/SQL na cláusula WITH (novo no Oracle 12c)
- Benefícios similares ao PRAGMA UDF
- Se o SQL não começar com WITH usar a *hint* WITH_PLSQL

```
WITH
  FUNCTION with_function(p_id IN NUMBER) RETURN NUMBER IS
BEGIN
  RETURN p_id;
END;
SELECT with_function(id)
FROM   t1
WHERE  rownum = 1
```

```
UPDATE /*+ WITH_PLSQL */ t1 a
SET a.id = (WITH
              FUNCTION with_function(p_id IN NUMBER) RETURN NUMBER IS
              BEGIN
                RETURN p_id;
              END;
              SELECT with_function(a.id)
              FROM   dual);
```

Prática 16: Técnicas de *Cache*

- Objetivos: explorar estratégias de otimização empregando diferentes técnicas de *cache*.
- *Result Cache*
- Funções Determinísticas
- *Scalar Subquery Cache*
- PRAGMA UDF
- *Inline PL/SQL*

Performance Tuning com Oracle 12c

Aula 17: Materialized Views

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- Definição
- *Materialized View Query Rewrite*
- Condições para Reescrita

Materialized Views

- São objetos que contém o resultado de uma consulta “materializados” em uma tabela.
- Aplicações:
 - *Data Warehouse*: sumários / agregações, *joins* pré-calculados
 - Computação distribuída: replicação de dados em *sites* distribuídos, fornecendo acesso local a dados remotos
 - Computação móvel: replicação de um subconjunto de dados para um dispositivo móvel.
- Antigamente chamadas de SNAPSHOTs.
- Na literatura para DW, são frequentemente chamadas de sumários.

Atualizando MVs

- A partir da sua criação, os dados são carregados para uma tabela a partir da consulta de definição
- A periodicidade da atualização pode ser:
 - Manual
 - Periódica (agendada)
 - Automática (*fast refresh*)
- Para utilizar a atualização automática é preciso criar objetos auxiliares, os MATERIALIZED VIEW LOGs, para registrar alterações que acontecem nos objetos de origem.

Criando uma *Materialized View*

```
CREATE MATERIALIZED VIEW mv1 AS SELECT * FROM hr.employees;
```

```
CREATE MATERIALIZED VIEW all_customers
  PCTFREE 5 PCTUSED 60
  TABLESPACE example
  STORAGE (INITIAL 50K)
  USING INDEX STORAGE (INITIAL 25K)
  REFRESH START WITH ROUND(SYSDATE + 1) + 11/24
  NEXT NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 15/24
  AS SELECT * FROM sh.customers@remote
    UNION
    SELECT * FROM sh.customers@local;
```

Fast Refresh

```
CREATE MATERIALIZED VIEW LOG ON times
    WITH ROWID, SEQUENCE (time_id, calendar_year)
    INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON products
    WITH ROWID, SEQUENCE (prod_id)
    INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW sales_mv
    BUILD IMMEDIATE
    REFRESH FAST ON COMMIT
    AS SELECT t.calendar_year, p.prod_id,
        SUM(s.amount_sold) AS sum_sales
        FROM times t, products p, sales s
        WHERE t.time_id = s.time_id AND p.prod_id = s.prod_id
        GROUP BY t.calendar_year, p.prod_id;
```

Principais Critérios para *Fast Refresh*

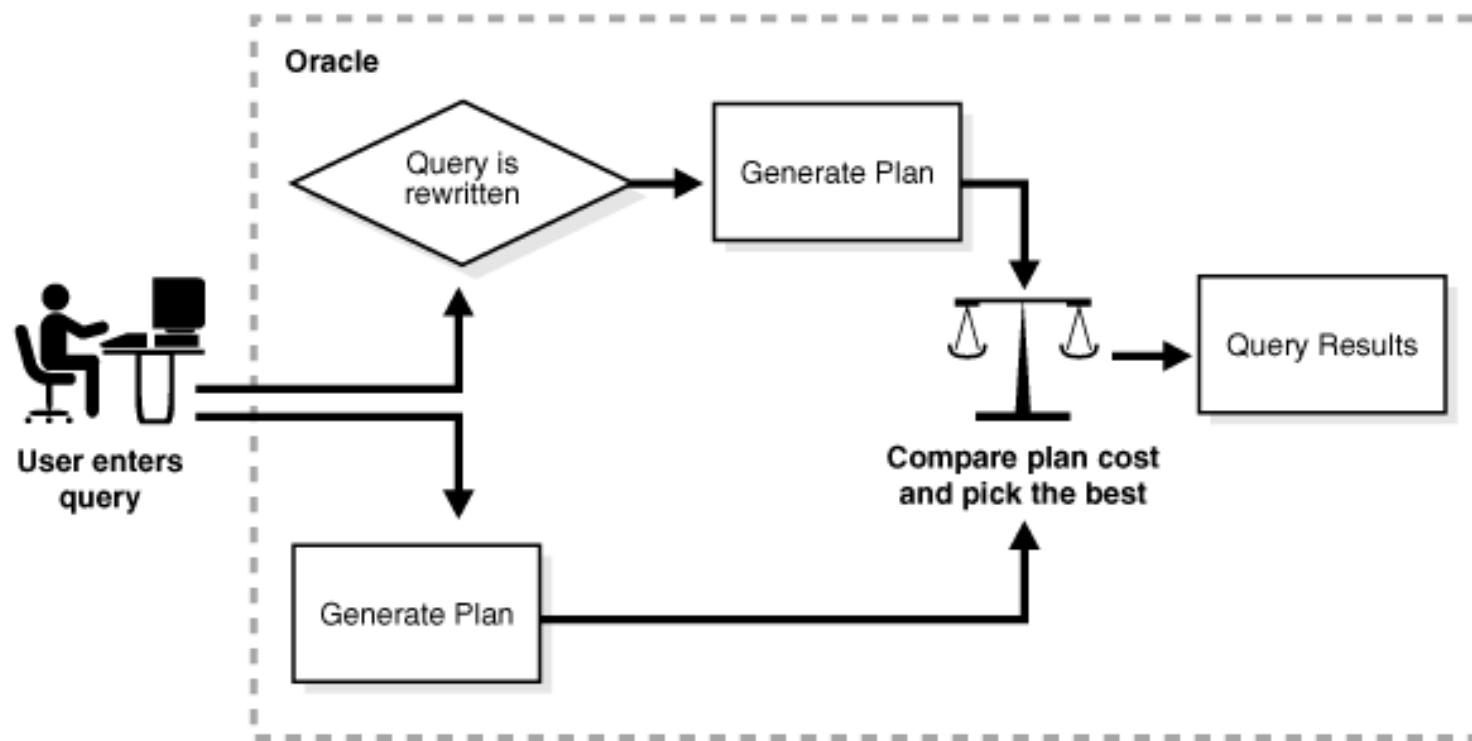
- Deve haver um MATERIALIZED VIEW LOG para cada tabela
 - Contendo todas as colunas referenciadas
- Agregações:
 - Suportadas: SUM, COUNT, AVG, STDDEV, VARIANCE, MIN e MAX
 - Precisa ter COUNT(*)
 - *Mview log*: especificar ROWID e INCLUDING NEW VALUES
 - *Mview log*: Especificar SEQUENCE para habilitar *direct-loads, deletes e updates*
- Joins:
 - Não pode ter agregações
 - Tem que ter o ROWID no *mview log*

Desenhando MVs para Performance

- O emprego de *Materialized Views* para *tuning* geralmente está associado a ambientes de cargas mistas ou DW que exigem o processamento de grandes volumes de informação.
- O objetivo é pré-calcular operações custosas:
 - Agregações: SUM, COUNT(x), COUNT(*), COUNT(DISTINCT x), AVG, VARIANCE, STDDEV, MIN, e MAX
 - *Joins*
- As MVs podem ser utilizadas direta ou indiretamente pelas aplicações
 - Direta: SELECT * FROM mview;
 - Indiretamente: *transparent query rewrite*

Materialized View Query Rewrite

- *Query Rewrite* é uma técnica de otimização que permite que o Oracle decida utilizar uma *materialized view* ao invés de realizar a operação em tempo real



Exemplo: Query Rewrite

```
CREATE MATERIALIZED VIEW cal_month_sales_mv
ENABLE QUERY REWRITE AS
SELECT t.calendar_month_desc, SUM(s.amount_sold) AS dollars
FROM sales s, times t WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

```
EXPLAIN PLAN FOR
SELECT t.calendar_month_desc, SUM(s.amount_sold)
FROM sales s, times t WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

SELECT OPERATION, OBJECT_NAME FROM PLAN_TABLE;	
OPERATION	OBJECT_NAME
-----	-----
SELECT STATEMENT	
MAT_VIEW REWRITE ACCESS	CALENDAR_MONTH_SALES_MV

Ferramentas para MVs

- SQL Access Advisor: parte do *Tuning Pack*, pode sugerir criação de *materialized views* e *materialized view logs*.
 - **Cuidado:** esta e algumas outras rotinas do DBMS ADVISED fazem parte do Tuning Pack
- DBMS ADVISED.TUNE_MVIEW: auxilia na modificação de *materialized views* para habilitar *fast refresh* e *query rewrite*.
 - Não é licenciada pelo Tuning Pack
- DBMS_MVIEW.EXPLAIN_MVIEW: lista capacidades de uma MVIEW
- DBMS_MVIEW.EXPLAIN_REWRITE: explica se o *query rewrite* vai ser utilizado ou não para uma consulta.

Prática 17: Materialized Views

- Criação de *Materialized Views*
- *Transparent Query Rewrite*

Performance Tuning com Oracle 12c

Aula 18: Modelagem Física

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

- Compactação de Índices
- Compactação de Dados
- Particionamento

Modelagem Física

- Lembrando: o melhor I/O é aquele que não fazemos!
- Estratégias de Modelagem Física para redução de I/Os:
 - Compressão de Índices
 - Compressão de Dados
 - Particionamento
 - *Materialized View Query Rewrite*

Compressão de Índice B*Tree

A	ROWID01
A	ROWID02
A	ROWID03
A	ROWID04
A	ROWID05
A	ROWID06
A	ROWID07
B	ROWID08
B	ROWID09
B	ROWID10
B	ROWID11
B	ROWID12



A	ROWID01
	ROWID02
	ROWID03
	ROWID04
	ROWID05
	ROWID06
	ROWID07
B	ROWID08
	ROWID09
	ROWID10
	ROWID11
	ROWID12

Tabela e Índice Como Objetos Separados

Root/Branch		
January	Customer1	➡
January	Customer1	➡
January	Customer1	➡
January	Customer2	➡
January	Customer2	➡
January	Customer3	➡
January	Customer3	➡
February	Customer1	➡
February	Customer1	➡
February	Customer2	➡
February	Customer3	➡
February	Customer3	➡

January	Customer1	500.00
January	Customer2	480.00
January	Customer3	360.00
January	Customer1	475.00
January	Customer2	600.00
January	Customer3	555.00
January	Customer1	200.00
February	Customer1	365.00
February	Customer3	510.00
February	Customer3	185.90
February	Customer2	800.00
February	Customer1	120.00

Index-Organized Table (IOT)

Root/Branch		
January	Customer1	500.00
January	Customer1	475.00
January	Customer1	200.00
January	Customer2	480.00
January	Customer2	600.00
January	Customer3	360.00
January	Customer3	555.00
February	Customer1	365.00
February	Customer1	120.00
February	Customer2	800.00
February	Customer3	510.00
February	Customer3	185.90

- IOTs economizam espaço em disco combinando os dados de tabela e índice em um único segmento (sem necessidade de duplicar dados)

Compressed Index-Organized Table (CIOT)

Root/Branch		
January	Customer1	500.00
		475.00
		200.00
January	Customer2	480.00
		600.00
		555.00
February	Customer1	365.00
		120.00
		800.00
February	Customer2	510.00
		185.90

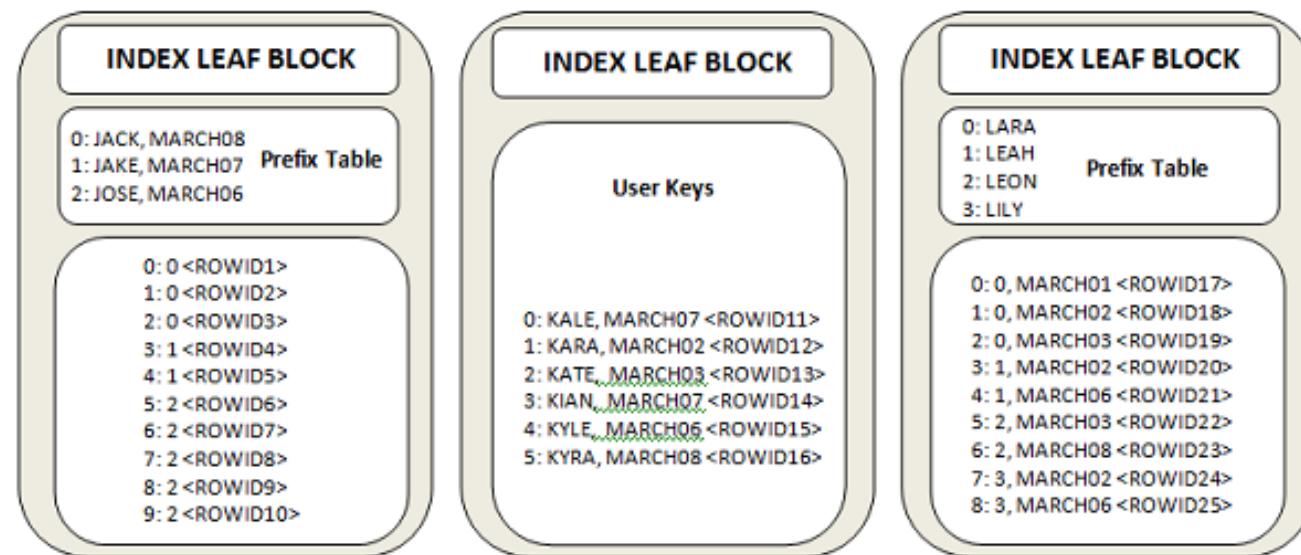
- IOTs economizam espaço em disco combinando os dados de tabela e índice em um único segmento (sem necessidade de duplicar dados)
- CIOTs economizam mais espaço eliminando valores repetidos (compressão de índice)

Compressão de Índices

- `ALTER INDEX ... REBUILD COMPRESS n`
 - Onde n é o tamanho da chave (número de colunas) a compactar
- O comando `ANALYZE INDEX ... VALIDATE STRUCTURE` pode indicar qual a melhor taxa de compressão e quantas colunas incluir na chave

Advanced Index Compression

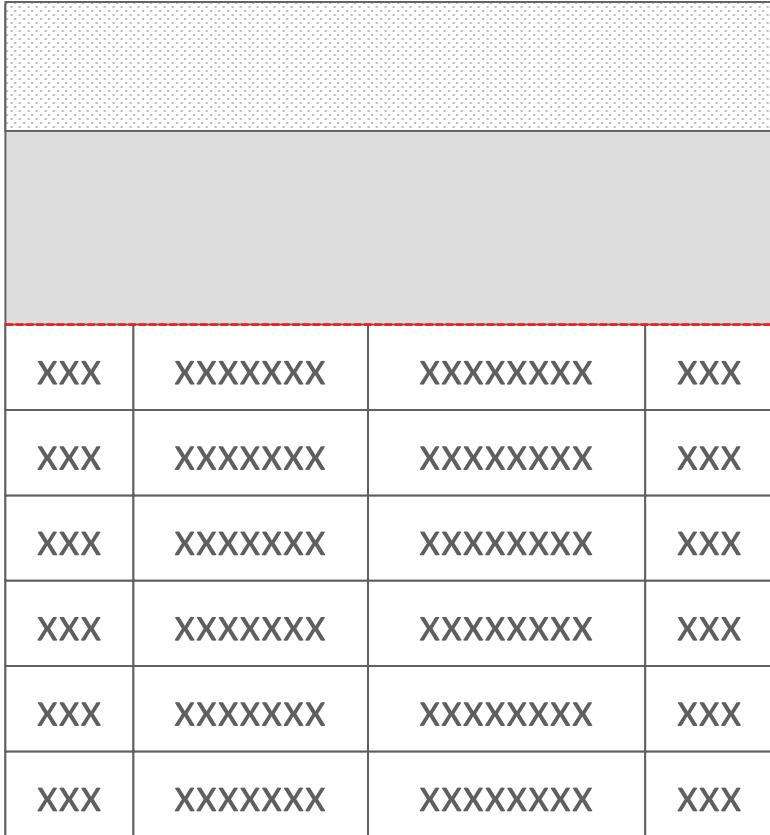
- Inclusa na *option Advanced Compression*
- Permite o cálculo automático do prefixo
- O prefixo é variável por bloco do índice



Advanced Index Compression

- Sintaxe:
 - CREATE INDEX idxname ON tablename(col1, col2, col3) **COMPRESS ADVANCED LOW**;
- Limitações:
 - Não é suportado em índices *bitmap*
 - Não é suportado em IOTs
 - Não é possível compactar índices baseados em função

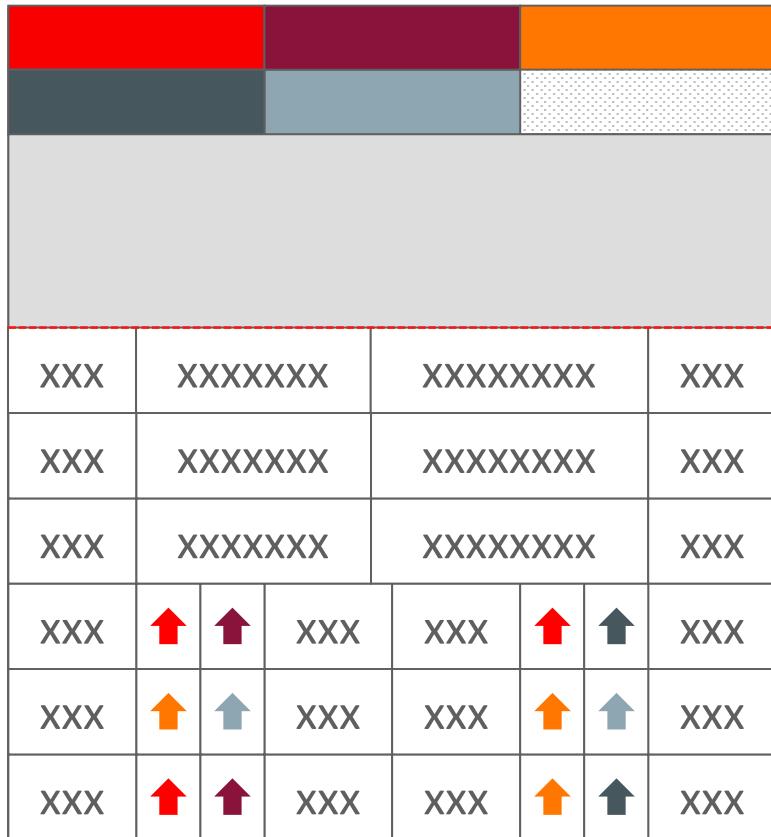
Compressão de Dados (*OLTP Compression*)



xxx	XXXXXX	XXXXXX	xxx
xxx	XXXXXX	XXXXXX	xxx
xxx	XXXXXX	XXXXXX	xxx
xxx	XXXXXX	XXXXXX	xxx
xxx	XXXXXX	XXXXXX	xxx
xxx	XXXXXX	XXXXXX	xxx

- Usuários inserem novos registros em um bloco do banco de dados
- Novos registros são gravados no formato normal (descompactados)
- Isto acontece até um **limite pré-definido (PCTFREE)**

Compressão de Dados (*OLTP Compression*)



- Quando o limite é atingido, os dados são comprimidos em *background*
- O processo
 - Varre os blocos e constrói uma tabela de simbolos no cabeçalho do bloco (*block header*)
 - Comprime os registros no bloco em questão
- Em seguida, o bloco está disponível para novos *inserts*

Compressão de Dados

- *Basic Compression* (não é OLTP!)
 - ALTER TABLE ... COMPRESS [BASIC]: sintaxe clássica
 - ALTER TABLE ... ROW STORE COMPRESS BASIC: sintaxe nova (12c)
 - É ativado somente através de um **direct-path insert** ou ALTER TABLE ... MOVE
- *Advanced Compression*
 - COMPRESS FOR OLTP: sintaxe clássica
 - ROW STORE COMPRESS ADVANCED: sintaxe nova (12c)
- *Hybrid Columnar Compression* (Exadata ou ZFS Appliance)
 - COMPRESS FOR QUERY | COMPRESS FOR ARCHIVE: sintaxe clássica
 - COLUMN STORE COMPRESS FOR [QUERY | ARCHIVE] [HIGH | LOW]

Particionamento

- Permite quebrar tabelas grandes em unidades físicas menores (partições)
- É possível controlar as propriedades de uma partição como se fosse uma tabela independente
 - *Tablespace*
 - *In-memory*
 - *Compression*
- O otimizador pode decidir por realizar o *partition pruning* e ler apenas a(s) partição(ões) relevante(s) caso a chave de particionamento apareça em uma cláusa WHERE.

Static Partition Pruning

```
SQL> explain plan for select * from sales where time_id = to_date('01-jan-2001', 'dd-mon-yyyy');
Explained.
```

```
SQL> select * from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
```

```
-----
```

```
Plan hash value: 3971874201
```

```
-----
```

Id Operation	Name	Rows	Bytes	Cost	(%CPU)	Time	Pstart	Pstop	
0 SELECT STATEMENT		673	19517	27	(8)	00:00:01			
1 PARTITION RANGE SINGLE		673	19517	27	(8)	00:00:01	17	17	
* 2 TABLE ACCESS FULL	SALES	673	19517	27	(8)	00:00:01	17	17	

```
-----
```

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

```
-----
```

```
2 - filter("TIME_ID"=TO_DATE('2001-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
```

Dynamic Partition Pruning

```
SQL> explain plan for select * from sales s where time_id in ( :a, :b, :c, :d);
Explained.
```

```
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
```

Plan hash value: 513834092

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		2517	72993	292 (0)	00:00:04		
1	INLIST ITERATOR							
2	PARTITION RANGE ITERATOR		2517	72993	292 (0)	00:00:04	KEY(I)	KEY(I)
3	TABLE ACCESS BY LOCAL INDEX ROWID	SALES	2517	72993	292 (0)	00:00:04	KEY(I)	KEY(I)
4	BITMAP CONVERSION TO ROWIDS							
* 5	BITMAP INDEX SINGLE VALUE	SALES_TIME_BIX					KEY(I)	KEY(I)

Predicate Information (identified by operation id):

5 - access("TIME_ID"=:A OR "TIME_ID"=:B OR "TIME_ID"=:C OR "TIME_ID"=:D)

Prática 18: Modelagem Física

- Compressão de Índices
- *Compressed IOT*
- Compressão de Dados
- Particionamento

Performance Tuning com Oracle 12c

Aula 19: In-Memory Database

Daniela Petruzalek
OCP DBA, OCP PL/SQL
daniela@svig.com.br

Objetivos

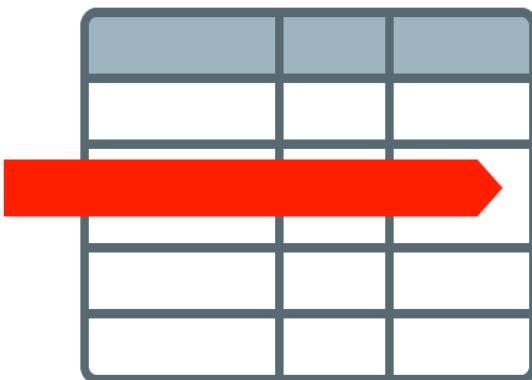
- Formato Colunar x Linear
- Configuração e Uso

In-Memory Database Option

- Não se trata de ser “apenas” in-memory (tudo no Oracle é in memory), mas sim no formato como os dados são salvos na memória
- Existem dois formatos clássicos de banco de dados:
 - Linear: os blocos de dados armazenam linhas de uma tabela
 - Colunar: os blocos de dados armazenam colunas de uma tabela
- Exemplo: tabela vendas
 - Linear: cada bloco contém a linha completa – preço, quantidade, id do produto, id do cliente, id da loja
 - Colunar: cada bloco contém uma coluna – tem o bloco dos preços, bloco das quantidades, etc

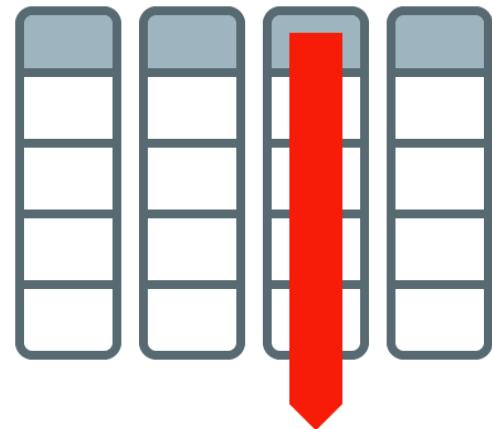
Formatos de Banco de Dados

- O formato linear é ideal para aplicações OLTP
 - Ao carregar dados de um cliente todos os dados do cliente são carregados na leitura de um bloco (idealmente)
 - `select * from clientes where cliente_id = 1`

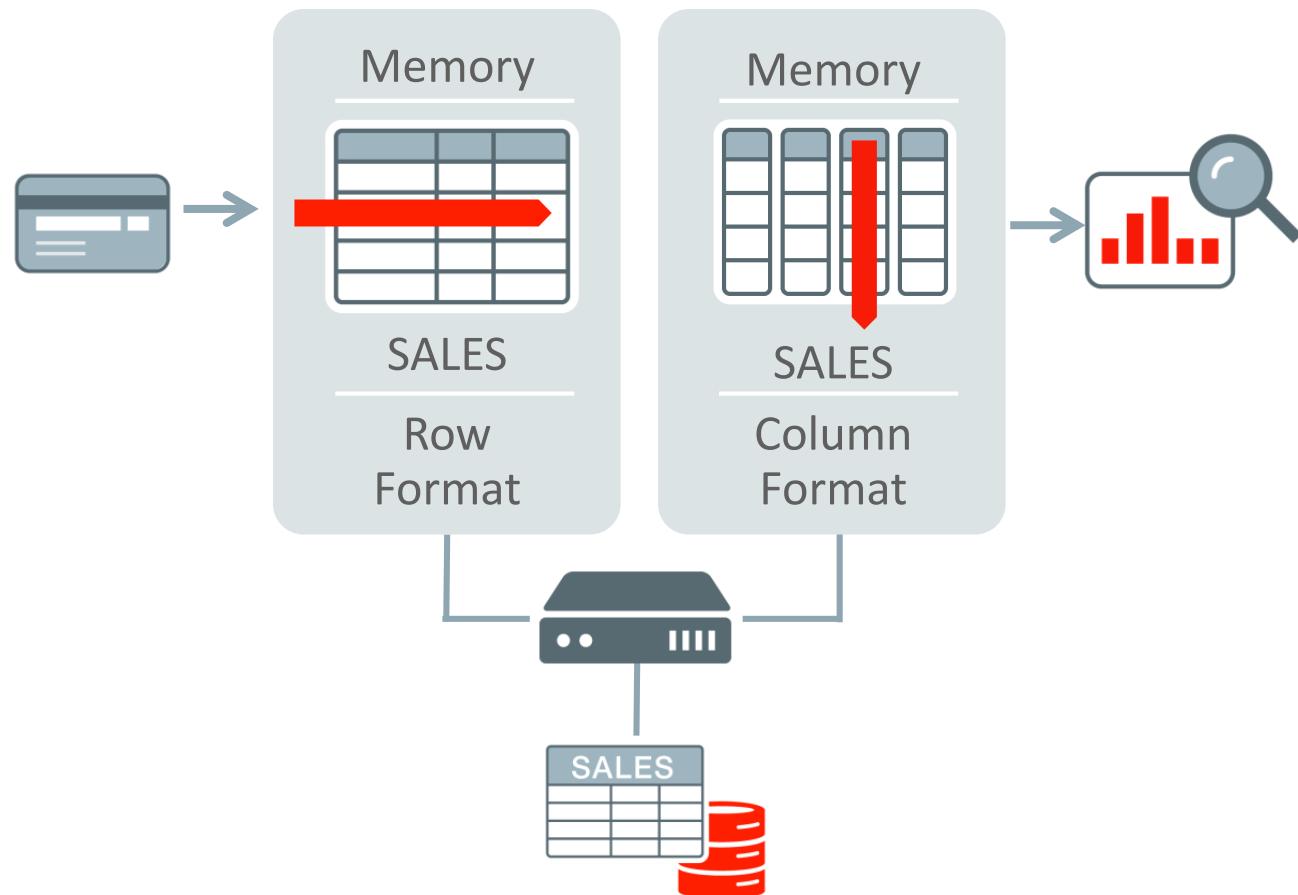


Formatos de Banco de Dados

- O formato colunar é ideal para agregações
 - Ao fazer uma query analítica, o banco de dados lê somente o dado que precisa (coluna valor_venda) sem perder I/Os lendo campos desnecessários
 - `select sum(valor_venda) from vendas where mes = 1`



In-Memory Database Option: Formato Híbrido

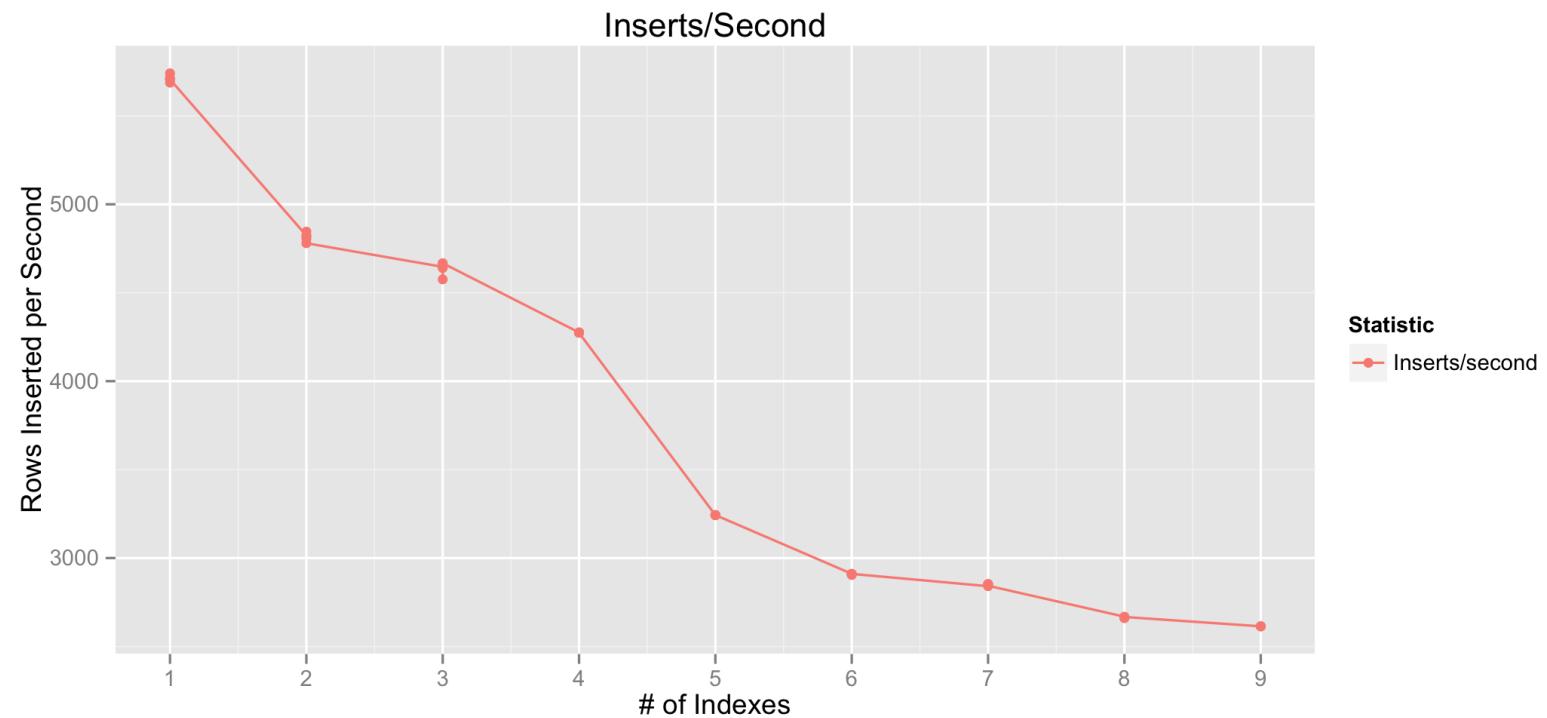


In-Memory Database Option: Benefícios

- Flexibilidade de formato
 - Melhor dos dois mundos no mesmo banco
- O formato colunar só existe em memória [não duplica uso de disco]
- O Oracle se encarrega de manter a consistência dos dados
- Porém:
 - Depende de memória física disponível [não é recomendado “substituir” memória de um uso para o outro]
 - Nem todas as querys vão se beneficiar do in-memory
 - Não resolve problemas de performance da aplicação

In-Memory: Oportunidades de Otimização

- Aumento de velocidade de querys analíticas
- Aumento de velocidade OLTP (**INSERT, UPDATE**)
 - Reduzindo o número de índices!



Configuração

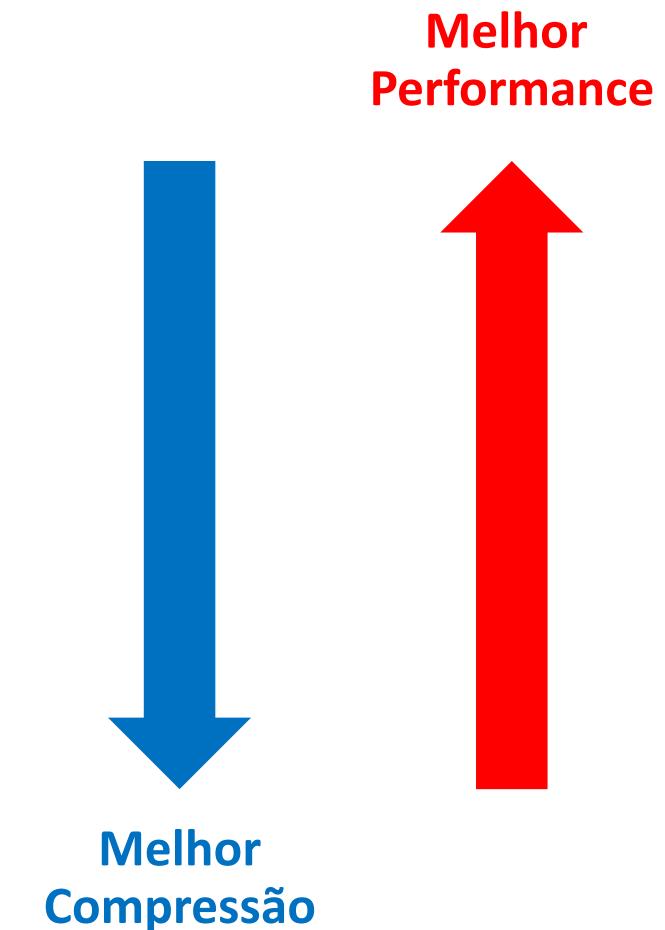
- ALTER SYSTEM SET INMEMORY_SIZE = XXX M|G SCOPE=SPFILE;
 - Precisa baixar/subir instância
- Ativando o in memory para objetos:
 - [ALTER | CREATE] TABLE ... INMEMORY;
 - ALTER PARTITION ... INMEMORY;
 - ALTER MATERIALIZED VIEW ... INMEMORY;
- Também pode ser setado por tablespace:
 - ALTER TABLESPACE ... DEFAULT INMEMORY MEMCOMPRESS FOR CAPACITY HIGH;

Principais Views

- [all | dba | user] _tables
 - inmemory
 - inmemory_priority
 - inmemory_distribute
 - inmemory_compression
 - inmemory_duplicate
- V\$im_column_level
- V\$im_segments
- V\$im_user_segments

Compressão

- Aplicado apenas na memória
- Tipos:
 - NO MEMCOMPRESS
 - MEMCOMPRESS FOR DML
 - MEMCOMPRESS FOR QUERY LOW
 - MEMCOMPRESS FOR QUERY HIGH
 - MEMCOMPRESS FOR CAPACITY LOW
 - MEMCOMPRESS FOR CAPACITY HIGH



Prioridade

- Controla quais objetos vão ocupar a *column store* quando não há espaço suficiente para todos.
 - PRIORITY NONE
 - PRIORITY LOW
 - PRIORITY MEDIUM
 - PRIORITY HIGH
 - PRIORITY CRITICAL

```
ALTER TABLE oe.product_information INMEMORY  
MEMCOMPRESS FOR CAPACITY HIGH  
PRIORITY LOW;
```

```
ALTER TABLE oe.product_information INMEMORY PRIORITY HIGH;
```

Granularidade de Colunas

```
ALTER TABLE oe.product_information
  INMEMORY MEMCOMPRESS FOR QUERY (
    product_id, product_name, category_id, supplier_id, min_price)
  INMEMORY MEMCOMPRESS FOR CAPACITY HIGH (
    product_description, warranty_period, product_status, list_price)
  NO INMEMORY (
    weight_class, catalog_url);
```

Prática 19: In-Memory

- Configuração
- Criação / Alteração
- Views
 - V\$im_segments
 - V\$im_user_segments
 - V\$im_column_level

Palavras Finais

- O objetivo deste curso era prepará-los com todas as ferramentas necessárias para encarar as necessidades de *tuning* do dia a dia.
- Cada situação demanda um tipo de estratégia diferente, não se contentem com a teoria, testem sempre!
- *Tuning* é um processo infinito, sempre existem oportunidades de melhora. Porém, o esforço para extrair qualquer percentual de melhora é cada vez maior... Logo, saiba quando parar!

MUITO OBRIGADA!!!

Dúvidas, críticas e sugestões: daniela@svig.com.br

