1.1 Em que consiste a optimização de SQL?

A linguagem SQL define o que queremos que seja feito e não como vai ser feito. Perante uma instrução SQL o Oracle tem vários caminhos possíveis para lhe dar resposta. Os recursos envolvidos nessas respostas podem ser insignificantes ou significativos, levando a que a obtenção da resposta possa ser instantânea ou demorada. O programador dispõe de alguns mecanismos que podem ajudar a base de dados Oracle a escolher o melhor caminho.

No contexto deste documento "Optimização de SQL" refere-se a um processo que começa com a análise da aplicação, continua com o desenho e prolonga-se com a escrita de instruções SQL. O objectivo deste processo é garantir que a base de dados Oracle escolhe o caminho óptimo para resolver uma instrução SQL. O caminho óptimo é aquele que a resolve consumindo o mínimo de recursos da máquina e demorando um mínimo de tempo.

Vamos reflectir sobre eficiência, pois assumimos que a eficácia está garantida.

1.2 Porquê optimizar as instruções SQL?

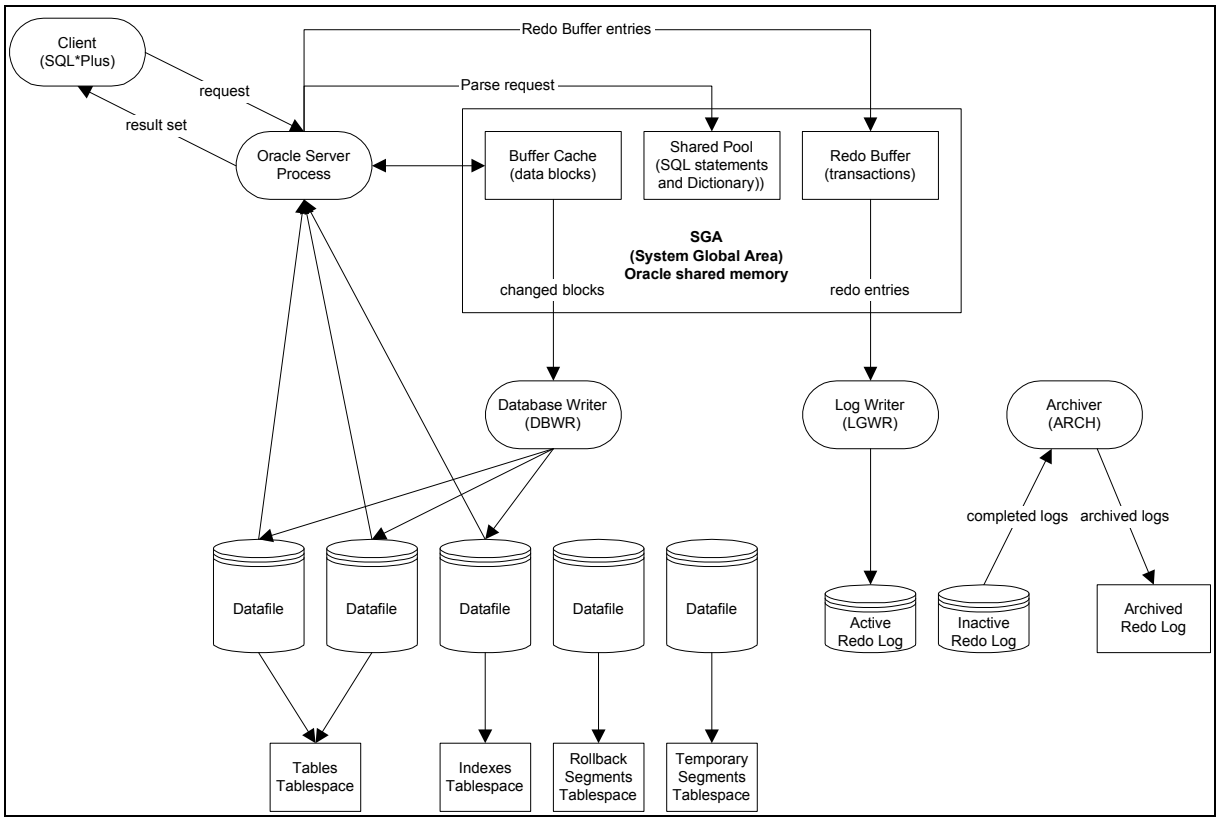
Fazer optimização de SQL não é fácil. Requer um esforço adicional sobre o trabalho de encontrar um comando que satisfaça a nossa necessidade. Normalmente exige a comparação entre várias alternativas e o tempo envolvido é inversamente proporcional à experiência adquirida. Ao fazê-lo podemos obter os seguintes benefícios:

- Melhorar o tempo de resposta das aplicações transaccionais que usam a base de dados. A maior fatia de tempo é gasta pela base de dados na procura e devolução das linhas pretendidas pelo "query" ou pelo "update". A optimização do processo de pesquisa reduz o esforço envolvido, poupa tempo e recursos do servidor, melhorando a resposta global.
- Melhorar o tempo de resposta de processos "batch". Só nos apercebemos que a produção dos relatórios de vendas diárias está com problemas porque o processo deveria correr durante a noite e ainda não terminou na manhã seguinte. A optimização dos "queries" envolvidos pode ser o caminho a seguir.
- Evitar aumento do tempo de resposta. Com o crescimento do volume de dados pode dar-se um aumento exponencial do tempo de resposta da aplicação. Mais uma vez, uma estrutura de dados bem definida e "queries" apropriados minimizam este impacto.
- Evitar a compra de mais hardware. Com a degradação do tempo de resposta será inevitável a compra de mais hardware, seja para dividir o processamento por várias máquinas, seja para aumentar a capacidade de uma única máquina. O processo de optimização pode adiar a necessidade da compra.

Comentários dos utilizadores sobre optimização de SQL:

- O "optimizer" faz isso por mim. O optimizer está cada vez mais inteligente, mas não é capaz de corrigir erros de concepção ou estrutura. Por exemplo quando as relações entre as tabelas não são as mais adequadas, quando falta de um índice, ou mesmo quando o tipo de índice não é o mais adequado.
- Não sou especialista em SQL. Sou especialista em JAVA. Se este problema se coloca é porque o técnico recebeu a responsabilidade de interagir com a base de dados, pelo que deve faze-lo da melhor maneira possível. Os erros cometidos nesta etapa comprometem o sucesso de toda a aplicação, e como consequência de toda a equipa.
- Farei a optimização mais tarde. Um factor crítico de sucesso é "fazer bem à primeira". No processo de desenvolvimento, quanto mais depressa se detectam e corrigem os erros, menores são os custos. Estes crescem exponencialmente com o avançar das etapas.
- Eu escrevo o SQL, os especialistas que o optimizem. Os conceitos envolvidos na optimização não são complexos e podem ser assimilados durante a aprendizagem do SQL.
- Não podemos gastar tempo e recursos no processo de optimização. Qual o custo de fazer as coisas mal? Que prejuízo teremos quando o cliente verificar que a aplicação é lenta, ou se tornou exponencialmente lenta à medida que o volume de dados cresceu linearmente? Qual o custo de corrigir o erro depois? Qual o custo de obrigarmos o cliente a comprar mais hardware? Qual o custo para o cliente pelo tempo de resposta insatisfatório? Qual o custo para o cliente por ter escolhido um mau fornecedor?

1.3 Qual a arquitectura da base de dados Oracle?



1.4 O que é um cursor?

Um **cursor** é um contexto de execução de uma instrução SQL. É uma área de memória no "server process" onde o Oracle armazena os seguintes dados da instrução SQL:

- O texto do comando;
- O código compilado do comando;
- O plano de execução;
- Um apontador para a linha do resultado que está a ser processada;

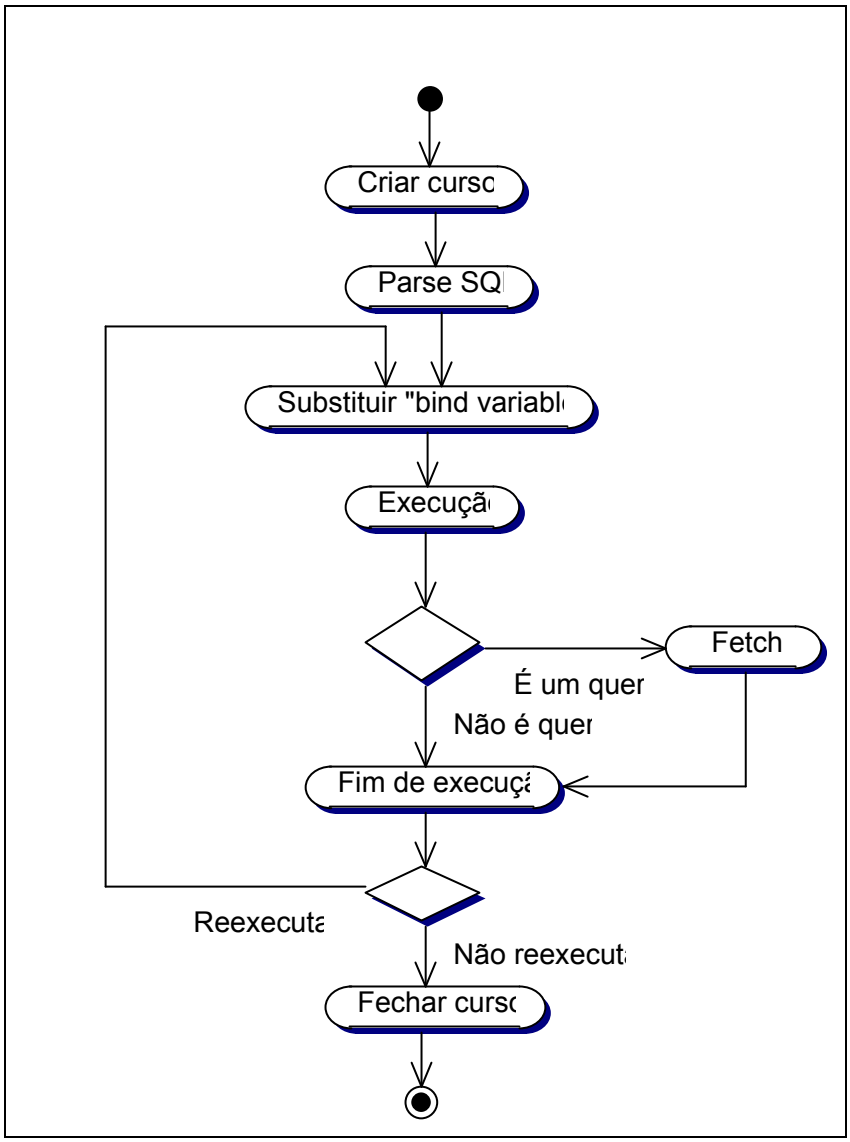
Quando a execução da instrução termina o cursor é removido, sendo libertada a memória por ele consumida.

1.5 O que é "parsing" de uma instrução SQL?

Quando uma instrução SQL é submetida ao SGBD é feito o seu "parsing". Este consiste nas seguintes operações:

- Verificação sintáctica.
- Verificação semântica - os objectos referenciados são/estão válidos.
- Verificação de segurança – o utilizador que pretende executar a instrução tem as permissões requeridas sobre todos os objectos.
- Determinar um plano de execução.

1.6 Como a base de dados executa uma instrução SQL?



1.7 Como o Oracle procura uma linha de uma tabela?

O Oracle possui 4 métodos para recuperar uma linha de uma tabela:

- Full table scan – É o modo de acesso mais simples. Todos os blocos de dados da tabela são consultados à procura das linhas que satisfazem o critério.
- Acesso por rowid – Todas as linhas da tabela têm uma pseudo coluna chamada rowid. Este valor não só identifica univocamente todas as linhas, como também informa o Oracle da localização exacta do bloco que armazena a linha. Não faz parte das colunas da tabela, mas pode ser consultada como se fizesse. É o modo mais rápido para retirar os dados de uma linha.
- "index lookup" – O Oracle começa por procurar no índice o identificador pretendido. Do índice retira o rowid da linha que satisfaz o critério de pesquisa. Finalmente faz o acesso aos dados da tabela usando o rowid.
- Consulta por hash-key – Um "Hash Cluster" é uma tabela cujas linhas são armazenadas usando funções hash. Estas funções matemáticas devolvem números que permitem ao Oracle identificar univocamente as linhas da tabela e a respectiva posição física onde estão armazenadas.

1.8 Quais as características dos mecanismos de "locking" do Oracle?

Os mecanismos de locking do Oracle têm as seguintes características:

- Só as linhas que estão a ser alteradas ou apagadas têm um "lock". As tabelas que têm índices "unique" sofrem um lock no índice da linha que está a ser actualizada.
- Uma linha que esteja "locked" continua a ser visível por instruções select. Estas vêm os dados antes da actualização. Os dados depois da actualização só se tornam visíveis às outras sessões depois do commit. As actualizações não prejudicam as leituras.
- Os locks são libertados com um commit ou rollback.
- O comando LOCK TABLE permite bloquear todas as linhas de uma tabela. A sua utilização deve estar muito bem justificada, pois pode por em perigo a funcionalidade da aplicação.
- Uma linha pode ser bloqueada pelo programador através do comando SELECT .... FOR UPDATE.
- As regras de integridade referencial podem gerar "locks". Ver na secção de índices quais e como os evitar.

1.9 Quais os parâmetros da base de dados que influenciam a execução de uma instrução SQL?

O Oracle possui um ficheiro de configuração que determina a forma como vai ser alocada memória na SGA. Durante o arranque da base de dados o Oracle lê este ficheiro e configura a SGA de acordo com as opções nele especificadas. O ficheiro chama-se INIT<SID>.ORA e está armazenado em ORACLE_HOME/database. Em Unix e NT normalmente encontra-se nesta directoria um link para o ficheiro original que está em ORACLE_BASE/admin/SID/pfile.

Os valores destes parâmetros num determinado instante podem ser consultados usando os métodos a seguir indicados, que requerem permissões de leitura sobre a tabela V$PARAMETER:

- No SQL*Plus:

```
SELECT * FROM V$PARAMETER;
```

- No Toad:
  - Aceder ao menu View → Oracle Parameters.

Alguns dos parâmetros do INIT<SID>.ORA influenciam a execução das instruções SQL e serão abordados ao longo deste manual, pelo que se descrevem a seguir, agrupados por categorias.

1.9.1 O processamento de SQL

Nome	Valores possíveis	Descrição
OPTIMIZER_MODE	CHOOSE, RULE, FIRST_ROWS, LAST_ROWS	Define qual o "optimizer" que o Oracle vai usar nesta instância.
HASH_AREA_SIZE	Numero em bytes	Define a área de memória que ficará reservada para operações de "hash-join".Não deve ser maior que a maior tabela que sofre um "hash-join, pois nesse caso toda a tabela cabe em memória.

1.9.2 A ordenação de linhas

Nome	Valores possíveis	Descrição
SORT_ÁREA_SIZE	Valor em bytes	Define o espaço de memória RAM que será alocado em cada "server process" para fazer ordenações de dados temporários. Se esse espaço não for suficiente, a ordenação usa o disco para swaping

1.9.3 O SQL tracing

Nome	Valores possíveis	Descrição
SQL_TRACE	True False	Se True todas as sessões Oracle vão gerar ficheiros de trace, nos quais estão os dados relativos aos comandos nelas executados. Isto

		gera um volume de dados em disco grande. O valor por omissão é False.
TIMED_STATISTICS	True False	Se True nas sessões Oracle é contabilizado o tempo que cada operação demora a ser executada e o respectivo tempo de CPU. Isto activa uma série de métricas da base de dados e permite ao tkprof calcular medidas de tempo. O valor por omissão é False. Pode ser alterado dinamicamente usando os comandos ALTER SYSTEM SET TIMED_STATISTICS=TRUE ou ALTER SESSION SET TIMED_-STATISTICS=TRUE.
MAX_DUMP_FILE_SIZE	Blocos Oracle	Define a maior dimensão que pode assumir um ficheiro de trace. Se uma sessão executar muitos comandos, pode exceder o valor máximo aqui definido, não sendo registados os dados adicionais.
USER_DUMP_DEST	Directoria	Define a directoria onde serão armazenados os ficheiros de trace do utilizador.

2 Ferramentas de diagnóstico

Neste capítulo vamos estudar as ferramentas que nos permitem analisar como uma instrução SQL vai ser executada e quais os recursos que vai consumir. Sem recorrer a estes métodos o programador sabe se a instrução demora muito ou pouco tempo, mas não tem ideia de como está a ser executada e quais os recursos envolvidos. Este conhecimento ajuda a definir caminhos alternativos no processo de optimização.

2.1 Como utilizar o "Explain plan"

O comando Oracle "EXPLAIN PLAN" mostra como o Oracle vai executar uma instrução SQL. Os passos seguidos pela base de dados na execução da instrução são armazenados numa tabela cujo nome, por omissão, será "PLAN_TABLE". A cada passo corresponde uma linha, e estas irão possuir relações hierárquicas entre si. O plano de execução será obtido por uma instrução SELECT à "PLAN_TABLE".

Suponha que queremos saber que clientes compraram produtos para bicicletas. As duas instruções SQL a seguir apresentadas respondem a esse pedido:

```
select c.customer_id,c.name,c.phone_number
  from customer c
 where exists (select 1
                 from sales_order so, item i, product p
                where so.customer_id=c.customer_id
                  and so.order_id=i.order_id
                  and i.product_id=p.product_id
                  and p.description like '%BICYCLE%'
               )
order by customer_id;


select customer.customer_id,customer.name, customer.phone_number
  from customer, sales_order, item, product
 where customer.customer_id=sales_order.customer_id
   and sales_order.order_id=item.order_id
   and item.product_id=product.product_id
   and product.description like '%BICYCLE%'
order by customer.customer_id;
```

2.1.1 Como ver o plano de execução em SQL*Plus?

A Oracle fornece um script SQL que permite criar a tabela que será usada pelo comando "explain plan". O script chama-se "UTLXPLAN.SQL" e está armazenado na directoria ORACLE_HOME/rdbms/admin. Deve ser executado pelo mesmo utilizador cujos comandos SQL queremos optimizar (user00 no nosso exemplo). Da sua execução resulta uma tabela de nome PLAN_TABLE. Editando o script podemos mudar o nome da tabela e adicionar colunas. Não devemos mudar o nome das colunas pré-definidas pois o comando "explain plan" espera encontrar esses nomes.

- Limpar a "PLAN_TABLE":

```
truncate table plan_table;
```

- Determinar os passos de execução e armazenar em "PLAN_TABLE":

```
Explain plan for
select *
  from customer c
 where exists (select 1
                 from sales_order, item, product
                where sales_order.customer_id=c.customer_id
                  and sales_order.order_id=item.order_id
                  and item.product_id=product.product_id
                  and product.description like '%BICYCLE%')
order by customer_id;
```

- Para ver os passos de execução corremos o query:

```
select rtrim(lpad(' ',2*level)||
       rtrim(operation)||' '||
       rtrim(options)||' '||
       object_name) query_plan
```

```
from plan_table
connect by prior id=parent_id start with id=0
```

- O resultado do 1º query será o seguinte:

```
SELECT STATEMENT
  FILTER
    TABLE ACCESS BY INDEX ROWID CUSTOMER
      INDEX FULL SCAN I_CUSTOMER$CUSTOMER_ID
    NESTED LOOPS
      NESTED LOOPS
        TABLE ACCESS FULL ITEM
        TABLE ACCESS BY INDEX ROWID PRODUCT
          INDEX UNIQUE SCAN I_PRODUCT$PRODUCT_ID
      TABLE ACCESS BY INDEX ROWID SALES_ORDER
        INDEX UNIQUE SCAN I_SALES_ORDER$ORDER_ID
```

- Limpar a tabela e repetir o processo para o segundo query:

```
truncate table plan_table;

Explain plan for
select distinct customer.customer_id,customer.name, customer.phone_number
  from customer, sales_order, item, product
 where customer.customer_id=sales_order.customer_id
   and sales_order.order_id=item.order_id
   and item.product_id=product.product_id
   and product.description like '%BICYCLE%'
order by customer.customer_id;

select rtrim(lpad(' ',2*level)||
       rtrim(operation)||' '||
       rtrim(options)||' '||
       object_name) query_plan
from plan_table
connect by prior id=parent_id start with id=0

  SELECT STATEMENT
    SORT (UNIQUE)
      NESTED LOOPS
        NESTED LOOPS
          NESTED LOOPS
            TABLE ACCESS FULL ITEM
            TABLE ACCESS BY INDEX ROWID PRODUCT
              INDEX UNIQUE SCAN I_PRODUCT$PRODUCT_ID
          TABLE ACCESS BY INDEX ROWID SALES_ORDER
            INDEX UNIQUE SCAN I_SALES_ORDER$ORDER_ID
        TABLE ACCESS BY INDEX ROWID CUSTOMER
          INDEX UNIQUE SCAN I_CUSTOMER$CUSTOMER_ID
```

Nos exemplos anteriores foi necessário apagar as linhas da tabela PLAN_TABLE quando pretendíamos obter o plano do segundo comando. Consulte a documentação da Oracle para saber como armazenar os dois planos de execução em simultâneo.

2.1.2 Como ver o plano de execução no Toad?

O primeiro passo é instalar as tabelas auxiliares que o Toad usa para gerar os planos de execução.

- Abrir uma conexão como System
- Abrir o script TOADPREP.SQL, que está na subdirectoria TEMPS da directoria de instalação dos binários do Toad. Este script cria o utilizador Toad, e dentro dele cria um conjunto de tabelas, sequências e sinónimos, que serão usados pelo programa.
- Alterar a "connect string" usada pelo script quando se liga como utilizador Toad.
- Executar o script.
- Ir à opção de menu View→ Options → Oracle e verificar que a opção "Explain Plan Table name" está com o valor "TOAD_PLAN_TABLE" e a opção "Save previous explain plan results" está activa.

Para ver o plano de execução de uma instrução procedemos da seguinte forma:

- Abrir uma janela de edição de SQL.
- Colocar a instrução SQL na janela.
- Executar a opção "Explain Plan current SQL" do menu SQL-Window.

A configuração feita para o TOAD permite que este armazene os diferentes comandos sobre os quais é pedido um EXPLAIN PLAN. Isto permite comparar planos de execução para o mesmo query. Para ver esta opção aceda ao menu View → Explain Plan. Este formulário permite também apagar os "explain plans" referentes a instruções mais antigas.

2.1.3 Como interpretar o resultado do "explain plan"?

Para ler o resultado do "explain plan" devemos seguir os seguintes princípios:

1. Ler de cima para baixo e da direita para a esquerda (←).
2. Quanto mais identado mais cedo é executado.
3. Quando dois passos estão no mesmo nível de identação é executado primeiro o que estiver mais acima.

2.1.3.1 Exemplo 1

```
SELECT STATEMENT
10  FILTER
02    TABLE ACCESS BY INDEX ROWID CUSTOMER
01      INDEX FULL SCAN I_CUSTOMER$CUSTOMER_ID
09    NESTED LOOPS
06      NESTED LOOPS
03        TABLE ACCESS FULL ITEM
05        TABLE ACCESS BY INDEX ROWID PRODUCT
04          INDEX UNIQUE SCAN I_PRODUCT$PRODUCT_ID
08      TABLE ACCESS BY INDEX ROWID SALES_ORDER
07        INDEX UNIQUE SCAN I_SALES_ORDER$ORDER_ID
```

1. Percorrer todo o índice de CUSTOMER para obter todas as linhas da tabela. Para cada linha fazer:
2. Obter uma linha de CUSTOMER por rowid.
3. Percorrer todas as as linhas de ITEM com "full table scan".
4. Procurar no índice de PRODUTO o "rowid" do produto cujo identificador se relaciona com o item do passo anterior e são BICYCLE.
5. Acesso à tabela PRODUTO por rowid.
6. Junção de ITEM e PRODUTO usando o método "nested loops" que vai criar o "result set" temporário A.
7. Procurar no índice o rowid da linha de "SALES-ORDER" que se relaciona com o item escolhido e com o CUSTOMER em referência.
8. Aceder a SALES_ORDER por rowid.
9. Fazer a junção de A com SALES_ORDER produzindo o "result set" B.
10. Tendo obtido os vários clientes e respectivos produtos, vamos eliminar os repetidos, para obter apenas dados de cliente.
11. Uma linha de CUSTOMER só aparece no resultado se existirem linhas resultantes das operações 3 a 10.

2.1.3.2 Exemplo 2

```
  SELECT STATEMENT
11  SORT (UNIQUE)
10    NESTED LOOPS
07      NESTED LOOPS
04        NESTED LOOPS
01          TABLE ACCESS FULL ITEM
03          TABLE ACCESS BY INDEX ROWID PRODUCT
02            INDEX UNIQUE SCAN I_PRODUCT$PRODUCT_ID
06        TABLE ACCESS BY INDEX ROWID SALES_ORDER
05          INDEX UNIQUE SCAN I_SALES_ORDER$ORDER_ID
09      TABLE ACCESS BY INDEX ROWID CUSTOMER
08        INDEX UNIQUE SCAN I_CUSTOMER$CUSTOMER_ID
```

1. Acesso à tabela ITEM com "full table scan". Para cada linha de Item:
2. Acesso à tabela PRODUCT usando o índice para determinar o produto que corresponde ao Item. Determinar o respectivo rowid.
3. Obter a linha de PRODUCT via rowid para determinar a sua descrição e comparar com '%BICYCLE%'.

4. Junção de ITEM e PRODUTO usando o método "nested loops" que vai criar o "result set" temporário A.
5. Acesso a SALES_ORDER usando o índice a partir de Sales_order_id de item. Determinar o respectivo rowid.
6. Obter a linha de SALES_ORDER via rowid.
7. Junção de A e SALES_ORDER usando o método "nested loops" que vai criar o "result set" temporário B.
8. Acesso à tabela CUSTOMER usando índice para determinar o Customer que se relaciona com SALES_ORDER (no result set B). Obter o respectivo rowid.
9. Obter linha de CUSTOMER por rowid.
10. Junção do "result set" B com e CUSTOMER usando o método "nested loops" que vai criar o "result set" temporário C.
11. Ordenação de "result set" C eliminando as linhas repetidas.

Da análise anterior verificamos que a execução começou pela tabela ITEM. Provavelmente esta não é a melhor decisão, mas para podermos escolhermos entre outros caminhos alternativos necessitamos mais informação, que pode ser obtida com o SQL TRACE.

O "explain plan" gerado depende de vários factores, como por exemplo: o "Optimizer" activo, o volume de dados e os parâmetros da base de dados. Uma variação nestes factores pode provocar diferentes planos de execução para o mesmo query. Se o Optimizer for "Rule" as variações são menores, sendo o resultado mais previsível. Para todos os outros casos as variações podem ser significativas.

3 O que é o "Optimizer"?

Na maior parte das situações um comando SQL pode ser executado de várias maneiras. Durante a operação de "parsing" o Oracle tem que escolher, de entre as várias alternativas, qual é o caminho mais eficiente para retirar as linhas que satisfazem o comando. O processo Oracle que determina o caminho óptimo chama-se "Optimizer" e tem dois modos de funcionamento:

- Rule – as decisões do "Optimizer" são baseadas num conjunto de regras que atribuem um peso a cada caminho alternativo. É utilizada uma função heurística para escolher o caminho com menor peso. Este mecanismo não tem em conta o volume de dados, o que significa que o "Optimizer" escolhe sempre o caminho que usa um índice em detrimento de um "full table scan", mesmo que a tabela só tenha duas linhas.
- Cost – as decisões do "Optimizer" têm em conta um conjunto de regras e a informação estatística retirada dos dados das tabelas e índices. Essa informação revela os volumes de dados envolvidos e a sua distribuição. Esta abordagem permite escolher caminhos diferentes quando a tabela tem duas linhas ou dois milhões de linhas.

O Rule surgiu com as primeiras versões de Oracle, enquanto que o Cost é mais recente e tem vindo a evoluir de forma significativa. O segundo normalmente escolhe um caminho que, se não for melhor, é igualmente bom, mas requer que seja gerada e mantida a informação estatística que o ajuda a decidir. O Rule tem um comportamento mais previsível, pois usa menos variáveis para tomar a decisão. O Cost é mais flexível e toma a decisão tendo em conta mais factores.

Por ser mais "inteligente" o optimizador Cost tende a tomar melhores decisões quando o SQL não foi optimizado. No entanto não se deve pensar que o seu uso dispensa o esforço de optimização, já que o Cost não consegue reescrever um comando SQL mal estruturado ou construir e usar um índice em falta, cuja criação nunca foi considerada.

3.1 Como decide o Optimizer Rule?

O optimizador analisa as tabelas usadas no query e as condições da cláusula where. Em seguida vai determinar os diferentes caminhos possíveis, atribuindo um peso a cada um baseado num conjunto de regras. A escolha recai sobre o caminho com menor peso. A tabela a seguir mostra os pesos atribuídos a cada operação usada no caminho:

Peso	Operação
1	Acesso a uma linha por rowid
2	Acesso a uma linha por "cluster join"
3	Acesso a uma linha por "hash cluster key" usando chave primária ou índice "unique"
4	Acesso a uma linha por chave primária ou índice "unique"
5	"Cluster join"
6	"Hash cluster key"
7	"Indexed cluster key"
8	Índice composto por várias colunas que têm que ser usadas na ordem em que foram criadas (todos ou parte)
9	Índice composto por uma única coluna
10	Pesquisa em colunas indexadas por um intervalo fechado de valores
11	Pesquisa em colunas indexadas por um conjunto não fechado de valores
12	"Sort merge join"
13	Pesquisa de valores máximo ou mínimo em colunas indexadas
14	Uso de Order by em colunas indexadas
15	Full table scan

- ***A pesquisa de uma linha é preferida sobre a pesquisa de um conjunto de linhas.***
- ***O uso dos índices é preferido a um "full table scan" e a "sort merge".***
- ***O uso do sinal = na pesquisa é preferido sobre um intervalo de linhas.***
- ***A pesquisa de um intervalo fechado é preferida a um intervalo aberto.***
- ***O uso de todas as colunas de um índice é preferido ao uso de apenas uma parte (mais selectivo).***
- ***A junção de tabelas com índices é preferida a tabelas sem índices.***
- ***A tabela usada como guia na junção é a que tem menor peso.***

3.2 Como decide o Optimizer Cost?

De forma semelhante ao optimizador Rule, o Cost analisa os caminhos possíveis e atribui-lhes um número. Este número que será proporcional aos recursos envolvidos para executar a instrução seguindo esse caminho, sendo interpretado como um custo. O custo tem em conta as regras e as estatísticas sobre os dados, tentando prever o número de blocos Oracle que serão manipulados, as necessidades de ordenação e o uso de "parallel query". Será escolhido o caminho com menor custo.

Esta aproximação tem resultados mais difíceis de prever que no caso anterior. Uma variação no volume de dados ou nos parâmetros da base de dados pode alterar o caminho escolhido.

3.2.1 Como gerar estatísticas sobre os dados?

O optimizador Cost necessita de informação sobre os dados para decidir melhor. Essa informação é obtida com o comando ANALYZE TABLE que retira dados das tabelas e respectivos índices. Alguns desses dados são:

- Tabelas – número de linhas, número de blocos Oracle usados e vazios, dimensão média de cada linha e quantidade média de espaço ocupado em cada bloco Oracle.
- Índices – número de chaves diferentes (selectividade do índice), número de folhas e profundidade da árvore.

Se as tabelas e índices forem de grande dimensão, a operação para obter as estatísticas pode ser impraticável por falta de espaço temporário ou tempo. Nesta situação o comando permite estimar as estatísticas em vez de as determinar com exactidão. A estimativa pode usar uma amostra de dimensão definida pelo programador.

```
Analyze table COSTUMER compute estatistics;
Analyze table COSTUMER estimate estatistics sample 20 percent;
```

3.2.2 Porquê usar "histograms"?

Normalmente o Optimizer sabe qual o número de chaves diferentes num índice, mas não conhece qual a sua distribuição. Considere que a nossa tabela CUSTOMER tem uma coluna que nos informa sobre a mão usada pelo cliente para escrever. O seu conteúdo teria 3 valores possíveis: esquerda, direita e ambas. As estatísticas indicam que 90% das pessoas escreve com a direita e apenas 1% com ambas. Se criarmos um índice sobre essa coluna, para os queries que pesquisam os clientes que escrevem com a direita, o índice não é selectivo. Se pelo contrário procurarmos quem escreve com ambas, já o índice seria selectivo. Os histograms armazenam informação sobre a frequência de valores nos índices permitindo ao Optimizer não usar o índice quando a pesquisa é sobre a mão direita (pouco selectivo) e usá-lo quando é sobre ambas (muito selectivo).

```
Analyze table COSTUMER compute estatistics for all indexed columns;
Analyze table COSTUMER compute estatistics for columns nome1,nome2;
```

O Optimizer não pode usar histograms se o valor a procurar no query estiver contido numa "bind variable". Isto porque as instruções que as usam são compiladas uma vez e executadas várias vezes com diferentes valores nas variáveis. O plano de execução é determinado na fase de parsing não sendo depois alterado em cada execução.

Se pretende tirar partido dos histograms não use "bind variables".

3.2.3 Que cuidados devemos ter ao usar o Optimizer Cost?

- Analisar **todas** as tabelas.
- Analisar as tabelas periodicamente. A frequência de análise deve ser proporcional às alterações e inserções nos dados.
- Criar "histograms" quando as colunas têm frequências de distribuição díspares. Neste caso não usar "bind variable".
- Garantir que os dados de desenvolvimento têm dimensão proporcional aos de produção, garantindo que a informação recolhida pelas estatísticas é representativa.

3.3 Como definir o objectivo do Optimizer?

O Optimizer pode ser configurado para 4 objectivos:

- Rule – o Optimizer vai basear-se nas regras Rule.
- Choose – o Optimizer vai basear-se nas regras Rule se alguma das tabelas não tiver informação estatística e em Cost caso todas tenham essa informação.
- All_rows – independentemente da existência de informação estatística o Optimizer vai usar Cost tendo o cuidado de escolher o caminho que permita a devolução mais rápida de todas as linhas do query. Este é o comportamento por omissão de Cost e permite obter melhores tempos de resposta em processamento "batch" e na execução de relatórios.
- First_roows - independentemente da existência de informação estatística o Optimizer vai usar Cost tendo o cuidado de escolher o caminho que permita a devolução mais rápida da primeira linha que satisfaz o query. Permite obter melhores tempos nas aplicações interactivas (OLTP – On-Line Transaction Processing).

Há três formas de definir estes objectivos:

- Definir o parâmetro OPTIMIZER_MODE no ficheiro INIT<SID>.ora. Esta definição será usada por todas as sessões da instância. Por omissão o parâmetro é CHOOSE.

```
OPTIMIZER_MODE=FIRST_ROWS
```

- Alterar o parâmetro OPTIMIZER_MODE para uma única sessão com o comando:

```
Alter session set OPTIMIZER_MODE=RULE;
```

- Alterar o parâmetro para uma instrução específica usando um HINT:

```
select /*+ FIRST_ROWS */
  distinct c.customer_id,c.name, c.phone_number
  from customer c, sales_order so, item i, product p
 where c.customer_id=so.customer_id
   and so.order_id=i.order_id
   and i.product_id=p.product_id
   and p.description like '%BICYCLE%'
order by c.customer_id;
```