20. Query Optimization

Este capítulo é uma introdução ao otimizador de consultas do DBMaker. O otimizador de consultas torna uma consulta em comandos SQL muito mais rápida e eficiente, escolhendo o melhor método de execução interna.

Os conteúdos deste capítulo abordam os seguintes tópicos:

- O que é otimização de consultas e por que precisamos dela? Ao entender o objetivo da otimização de consultas, você perceberá o papel que ela desempenha em uma consulta SQL.
- O que é o Plano de Execução de Consulta (QEP) e como ler um QEP? Quando você conhece o QEP, aprenderá como o DBMaker executa um comando de consulta SQL.
- Como o otimizador de consultas opera? Quando você entender como o otimizador de consultas busca um QEP, poderá ajudá-lo a encontrar um QEP mais eficiente reescrevendo uma consulta SQL equivalente.
- O que é a função de custo? Quando você souber quanto tempo leva para uma operação no QEP, aprenderá como o otimizador de consultas escolhe uma operação adequada. Use alguns comandos fornecidos pelo DBMaker para ajudar o otimizador de consultas a encontrar uma operação melhor.
- Quais são os valores estatísticos e onde esses valores têm sido usados? Quando você entender o uso dos valores estatísticos na otimização de consultas, verá a razão pela qual o otimizador de consultas escolhe um plano de execução específico.

Como a velocidade de execução de uma consulta pode ser acelerada? Quando você sabe como escrever uma consulta eficiente, pode aumentar a eficiência da execução reescrevendo o comando da consulta.

20.1 What is Query Optimization

Os comandos de Linguagem de Manipulação de Dados (DML), como SELECT, INSERT, DELETE e UPDATE, são uma etapa muito importante na otimização de consultas. O DBMaker pode ter vários métodos de execução para uma única consulta SQL. O objetivo da otimização de consultas é encontrar o plano de execução mais eficiente. O principal trabalho do otimizador de consultas é decidir cada operação e a ordem em que ela será executada.

 Para encontrar a operação mais eficiente:

1. Leia os dados de uma tabela — os dados podem ser lidos por meio de uma varredura sequencial ou de índice.
2. Junte tabelas — as tabelas podem ser unidas com um loop aninhado ou uma junção por mesclagem.
3. Classifique — quando uma ordenação é necessária, antes ou depois de uma operação? A ordenação pode ser evitada usando uma solução alternativa?

O otimizador de consultas deve estimar o número de linhas afetadas por junções externas para otimizar a sequência das tabelas a serem unidas. Alguns usuários de banco de dados se familiarizam tanto com as características dos dados que se tornam mais eficientes do que o otimizador de consultas na execução de uma consulta.

O otimizador de consultas do DBMaker estimará todos os possíveis planos de execução para cada plano, calculará o número de linhas, quantas entradas e saídas de páginas de disco são necessárias e o tempo de CPU que leva para uma única tabela. A partir dos fatores mencionados acima, ele encontrará um plano com o menor custo.

Quando o DBMaker procura um plano de execução de consulta, ele considerará algumas operações principais:

- Varredura de tabela — ou varredura sequencial, que significa receber cada linha das páginas de dados de um banco de dados em ordem sequencial.
- Varreduras de índice — a ordem de recuperação dos dados é referenciada pelo endereço da página de dados indicado pela página de índice.
- Junção aninhada — compara duas ou mais tabelas, linha por linha, para alcançar o objetivo da mesclagem.
- Junção por mesclagem — classifica duas tabelas e compara linha por linha para realizar a mesclagem.
- Classificação — executa a ordenação.
- Tabela temporária — no processo de execução da consulta, estabelece uma tabela temporária.

20.2 How Does the Optimizer Operat

Quando o otimizador do DBMaker realiza o processo de otimização de uma consulta, ele usará as seguintes regras:

- Analisar a consulta e, em seguida, dividir o predicado WHERE em vários fatores.
- Pesquisar todas as sequências de execução possíveis e a sequência de junção.

- Decidir se deve usar uma junção aninhada ou uma junção de mesclagem ordenada.
- Decidir se deve usar uma varredura de tabela ou de índice.
- Decidir a ordem de classificação.

Input of Optimizer

A precisão da estimativa é o fator crítico que decide se o otimizador será bem-sucedido ou não. No entanto, há apenas informações limitadas para o otimizador estimar o tempo necessário para uma operação, uma pequena parte em comparação com o tempo real de execução. Todas as informações necessárias para o otimizador vêm das tabelas de catálogo do sistema. Para garantir que essas informações sejam úteis e não estejam desatualizadas, use o comando UPDATE STATISTICS. Consulte a seção 19.4, Estatísticas, para mais informações.

Os dados listados nas tabelas de catálogo do sistema incluem:

- Número de linhas em uma tabela
- Número de páginas de dados usadas por uma tabela
- Bytes médios por linha em uma tabela
- Bytes médios usados por uma coluna
- O valor distinto de cada coluna de índice
- O segundo maior e menor valor de cada coluna; a razão pela qual não escolhemos o valor máximo e mínimo é para evitar que valores especialmente grandes e pequenos afetem a precisão
- Número de páginas de varredura de índice ocupadas pelo índice B-tree
- Número de níveis (altura) no índice B-tree
- Número de páginas de folha no índice B-tree
- Contagem de clusters para o índice B-tree

A premissa para o otimizador usar essas informações é que os valores dos dados estejam distribuídos uniformemente. Se a distribuição dos dados estiver distorcida, não uniforme, o otimizador escolherá um plano inadequado.

Factors

A primeira tarefa do otimizador é examinar todas as expressões no predicado WHERE. Podemos decompor essas expressões em várias expressões menores e independentes chamadas fatores

Exemplo 1:
O otimizador decomporá o predicado WHERE em dois fatores: `tb_staff.id = tb_salary.id` e `tb_salary.basepay = 3000`.

```
dmSQL> SELECT * FROM tb_staff, tb_salary WHERE tb_staff.id =
tb_salary.id AND
tb_salary.basepay=3000;
```

Exemplo 2:

Usando o predicado WHERE para um fator: `tb_staff.id = tb_salary.id` ou `tb_salary.basepay = 3000`.

```
dmSQL> SELECT * FROM tb_staff, tb_salary WHERE tb_staff.id =
tb_salary.id OR
tb_salary.basepay=3000;
```

Exemplo 3:

Usando o predicado WHERE para dois fatores: `tb_staff.id = tb_salary.id` e `tb_salary.basepay = 3000` ou `tb_staff.name = 'joy'`.

```
dmSQL> SELECT * FROM tb_staff, tb_salary WHERE tb_staff.id =
tb_salary.id AND
(tb_salary.basepay=3000 OR tb_staff.name='joy');
```

Exemplo 4:

Usando o predicado WHERE para um fator: `tb_staff.id = tb_salary.id` ou `tb_staff.name = 'joy'`:

```
dmSQL> SELECT * FROM tb_staff, tb_salary WHERE tb_staff.id =
tb_salary.id AND
tb_salary.basepay=3000 OR tb_staff.name='joy';
```

No exemplo acima, percebemos que quando a expressão contém a operação binária "e" (AND), ela pode ser dividida em diferentes fatores. No entanto, quando contém a operação binária "ou" (OR), a decomposição não é permitida.

Para encontrar os fatores, o otimizador precisa estimar a seletividade de cada fator. A seletividade é a proporção de dados filtrados por cada fator; seu valor está entre 0 e 1. A Tabela 1 contém 100 linhas.

Exemplo 5: Usando 5 linhas para a consulta na tabela tb_staff, tb_staff.id = 3 é 5 / 100, ou seja, 0,05.

```
dmSQL> SELECT * FROM tb_staff WHERE tb_staff.id=3;
```

Se houver mais de um fator em uma expressão, a seletividade dessa expressão é o produto desses fatores, pois são independentes entre si.

Join Sequence

A sequência de junção é a ordem de acesso das tabelas originais a serem mescladas. Diferentes sequências de junção produzirão diferentes sequências de execução e diferentes tempos de execução. No entanto, não importa como executemos, sempre recuperaremos o resultado correto.

Exemplo 1:

```
dmSQL> SELECT * FROM tb_staff, tb_salary WHERE tb_staff.id =
tb_salary.id;
```

Plano de Execução da Consulta 1:

```
nested join
 table scan tb_staff
 table scan tb_salary, filter tb_staff.id = tb_salary.id;
```

Plano de Execução da Consulta 2:

```
nested join
 table scan tb_salary
 index scan tb_staff on tb_staff (id), filter tb_staff.id =
tb_salary.id;
```

Exemplo 2:

```
dmSQL> SELECT * FROM tb_staff, tb_salary,tb_dept WHERE tb_staff.id =
tb_salary.id
```

```
AND tb_salary.id=tb_dept.id;
```

Resultado da Execução da Consulta: Haverá 3! (= 6) sequências de junção:

```
(tb_staff, tb_salary), tb_dept
(tb_staff, tb_dept), tb_salary
(tb_salary, tb_staff), tb_dept
(tb_salary, tb_dept), tb_staff
(tb_dept, tb_staff), tb_salary
(tb_dept, tb_salary), tb_staff
```

O DBMaker pesquisará todas essas sequências de junção, calculará o custo de cada uma e escolherá a melhor.

Nested Join and Merge Join

Os métodos de junção suportados pelo DBMaker são junções aninhadas e junções de mesclagem.

- A junção aninhada usa um loop aninhado sobre duas camadas para realizar a junção. O algoritmo de análise para sua complexidade de tempo é $O(n^2)$.
- A junção de mesclagem ordenará duas tabelas previamente e, em seguida, mesclará essas duas tabelas na ordem classificada, linha por linha. A complexidade de tempo para ordenar é $O(n \times \log(n))$. Os dados que já foram ordenados têm a complexidade de tempo para realizar uma junção de $O(n)$. A junção de mesclagem só pode ser usada para junções de igualdade.

Do ponto de vista da complexidade de tempo, a junção de mesclagem é melhor do que a junção aninhada. No entanto, ainda existem exceções; por exemplo, quando há uma grande diferença no número de linhas entre duas tabelas. Independentemente disso, o otimizador decidirá a melhor maneira de realizar a junção com funções de custo e valores estatísticos.

Table Scan and Index scan

As varreduras de tabela adquirem linhas de uma tabela sequencialmente, linha por linha. Por exemplo, se um usuário quiser encontrar todas as linhas em uma tabela que correspondam à condição idade > 50, ele receberá cada linha de cada página de dados e, em seguida, comparará cada linha com a condição para recuperar a desejada.

Outro tipo de varredura é a varredura de índice, que constrói o índice nas colunas de uma tabela e depois encontra todos os dados necessários referenciando o índice. O método de índice usado pelo DBMaker é uma árvore B (B-tree), e a condição para usar uma varredura de índice é construir um índice na coluna para uso com o predicado.

Sort

Outra operação importante do otimizador de consultas é determinar como ordenar, antes ou depois de uma junção, ou tentar evitar uma ordenação. Exemplo:

Criando uma ordenação:

```
dmSQL> SELECT * FROM tb_staff, tb_salary WHERE
tb_staff.id=tb_salary.id ORDER BY
tb_staff.basepay;
```

Plano de Execução de Consulta 1, o otimizador realiza a ordenação após a junção:

```
sort tb_staff.basepay
merge join tb_staff.id=tb_salary.id
index scan tb_staff on idx_id(id)
sort tb_salary.id
table scan tb_salary;
```

Plano de Execução de Consulta 2, o otimizador realiza a ordenação antes da junção:

```
nested join
index scan tb_staff on idx_base(basepay)
table scan tb_salary, filter tb_staff.id=tb_salary.id;
```

20.3 Time Cost of a Query

O tempo para ler dados do disco e o tempo para comparar os valores das colunas são duas partes principais na execução de uma consulta.

CPU Cost

O servidor de banco de dados deve processar os dados na memória. Ele precisa ler uma linha na memória e, em seguida, usar uma expressão de filtro para testar. É necessário carregar dados de duas tabelas na memória primeiro e, em seguida, testar

a condição de junção. Além disso, o servidor de banco de dados deve coletar dados para as colunas selecionadas de cada linha. Mais tempo é necessário para ordenação quando curingas, como "like" e "match", são usados em instruções SQL.

I/O Cost

Leva muito mais tempo para ler uma linha do disco do que para verificar uma linha na memória, portanto, um dos principais objetivos do otimizador é reduzir a entrada/saída (I/O) do disco. A unidade básica de armazenamento em disco que um servidor de banco de dados processa é chamada de página. Uma página é composta por blocos agrupados, e o tamanho de uma página está relacionado ao servidor de banco de dados. O DBMaker suporta tamanhos de 4 KB, 8 KB, 16 KB ou 32 KB. A capacidade de linhas de uma página está relacionada ao tamanho de uma linha. No caso de o tamanho da página ser 4 KB, geralmente há de 10 a 100 linhas em uma página de dados. A entidade de uma página de índice contém um valor de chave e um ponteiro de quatro bytes. Normalmente, há de 100 a 1000 entradas em uma página de índice.

O servidor de banco de dados precisa de memória para armazenar cópias das leituras de páginas do disco para processamento. Devido a limitações de memória, algumas páginas podem ser relidas. A memória é chamada de buffer de página. Se a página necessária estiver no buffer de página, o servidor não lerá mais a linha do disco, o que aumentará o desempenho. O servidor de banco de dados e o sistema operacional decidem o tamanho de uma página de disco e o número de buffers de página. O custo real para ler uma página é variável e difícil de estimar.

Os buffers são combinações dos seguintes fatores:

• **Buffers** – É possível que a página alvo esteja no buffer de página. O custo de acesso pode ser praticamente ignorado nesta condição.

• **Contenção** – Se houver mais de um programa de aplicativo tentando usar dispositivos de hardware, como um disco, as solicitações do servidor de banco de dados serão atrasadas.

• **Tempo de busca** – Esta é a operação mais demorada para um disco. Refere-se ao tempo decorrido para mover a cabeça de leitura/gravação até o local dos dados desejados. É afetado pela velocidade do disco e pela posição inicial da cabeça de leitura/gravação do disco. A variação também depende em grande parte do tempo de busca.

• **Tempo de latência** – Também conhecido como tempo de atraso de rotação, está relacionado à velocidade do disco e à localização da cabeça de leitura/gravação.

Cost of Table Scan

O tempo gasto para escanear todos os dados em uma tabela. Independentemente de haver ou não predicados na consulta, é necessário comparar todos os dados contidos nas páginas. O custo de uma varredura de tabela é igual ao número de páginas de dados.

Cost of Index Scan

As varreduras de índice leem dados através das páginas de índice B-tree. Existem dois tipos de varreduras de índice: uma lê as páginas de dados referenciadas pela B-tree, e a outra lê os dados diretamente da folha do índice, conhecida como varredura de folha (leaf scan).

Exemplo

Usando a tabela **tb_staff** com as colunas **id** e **name**, e realizando uma varredura em um índice construído sobre a coluna **id**:

```
dmSQL> SELECT * FROM tb_staff WHERE id > 0;
```

Forma alternativa:

```
dmSQL> SELECT id FROM tb_staff WHERE id > 0;
```

Podemos usar uma varredura de folha e não será necessário ler dados das páginas de dados, pois as páginas de folha contêm todos os dados desejados.

• Quando todos os dados são lidos, o custo de uma varredura de índice é:

custo = I/O de nível B-tree + número de I/Os das páginas de folha + contagem de clusters

• Quando todos os dados são lidos, mas apenas uma varredura de folha é necessária, o custo da varredura de índice é:

custo = I/O de nível B-tree + número de I/Os das páginas de folha

• Quando uma linha é lida, o custo de uma varredura de índice é:

custo = I/O de nível B-tree + I/O de uma página de folha + I/O de uma página de dados

• Quando uma linha é lida, mas apenas uma varredura de folha é necessária, o custo da varredura de índice é:

custo = I/O de nível B-tree + I/O de uma página de folha

• Quando dados parciais são lidos, o custo de uma varredura de índice é:

custo = I/O de nível B-tree + (número de páginas de folha × S) + (contagem de clusters × S), onde S é a seletividade

Cost of Sort

Ler dados do disco para a memória é a única operação que leva mais tempo. O custo de computação é proporcional a **c × w × n × log2(n)**, onde:

- **c** é o número de colunas sendo ordenadas,
- **w** é o número de bytes da chave ordenada,
- **n** é o número de linhas sendo ordenadas.

Cost of Nested Join

São necessários mais de dois loops para acessar as páginas de dados em uma junção aninhada. Em uma junção aninhada, a tabela externa é diferente da tabela interna. Geralmente, o custo de uma junção aninhada é:

custo = I/O da tabela externa + I/O da tabela interna × número de linhas na tabela externa.

Cost of Merge Join

É necessário ordenar as tabelas antes de realizar uma junção por mesclagem (merge join). Suponha que duas tabelas já tenham sido ordenadas para mesclar com as chaves de mesclagem. O custo da junção por mesclagem é a soma dos I/Os dessas duas tabelas. Se a ordenação não for realizada nas chaves de mesclagem, o custo da ordenação ainda precisará ser adicionado.

20.4 Statistics

Estatísticas representam a quantidade e distribuição dos dados de uma tabela. Elas fornecem as informações para as funções de custo encontrarem o melhor plano de acesso. No entanto, as estatísticas podem ficar desatualizadas se dados na tabela forem inseridos, excluídos ou atualizados. O comando **UPDATE STATISTICS** deve ser utilizado para atualizar os valores estatísticos e obter estatísticas em tempo real para melhorar a eficiência de uma consulta.

Types of Statistics

O DBMaker coletará as seguintes estatísticas:

FOR TABLE

- **nPg** – número de páginas de dados
- **nRow** – número de linhas de dados
- **avLen** – média de bytes por linha

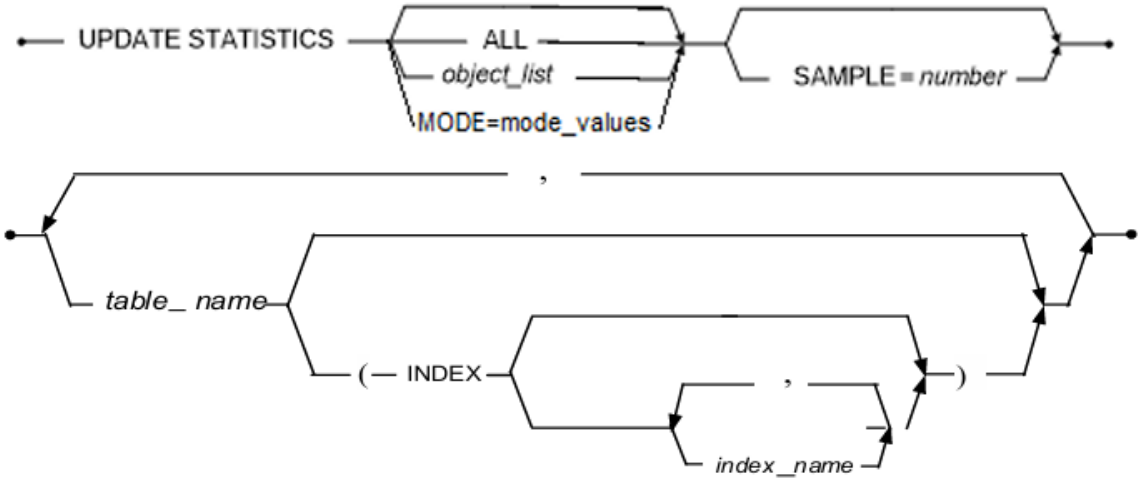
FOR COLUMN

- **distVal** – número de valores distintos
- **avLen** – média de bytes por cada coluna
- **loVal** – o segundo valor mínimo de uma coluna
- **hiVal** – o segundo valor máximo de uma coluna

FOR INDEX

- **nPg** – número de páginas de índice
- **nLevel** – número de níveis em uma árvore de índice
- **nLeaf** – número de folhas em uma árvore de índice
- **distKey** – número de chaves distintas
- **distC1** – número de chaves distintas para a primeira coluna do índice
- **distC2** – número de chaves distintas para as duas primeiras colunas do índice
- **distC3** – número de chaves distintas para as três primeiras colunas do índice
- **nPgKey** – número de páginas de índice para cada chave
- **cCount** – número de clusters contados; o número de páginas de dados acessadas através de um índice

UPDATE STATISTICS Syntax



The *object_list* Clause

- **ALL** – significa atualizar forçadamente os valores das estatísticas para todos os objetos do esquema.
- **SAMPLE** – significa a taxa de amostragem expressa como uma porcentagem do total, um número inteiro entre 1 e 100.
- **MODE** – significa o modo de configuração da amostra ao executar o comando **UPDATE STATISTICS**. O valor padrão é 0, o que significa usar a taxa de amostragem definida no banco de dados (DB_StsSp); definido como 1 significa usar a taxa de amostragem da configuração da tabela (comando **UPDATE STATISTICS SET**); definido como 2 significa que o comando **UPDATE STATISTICS** decidirá automaticamente a taxa de amostragem para cada tabela.

Exemplo 1:

Para atualizar os valores das estatísticas para todos os objetos do esquema:

```
dmSQL> UPDATE STATISTICS;
```

Se o banco de dados for extremamente grande, levará muito tempo para atualizar os valores das estatísticas de todos os objetos do esquema. Uma alternativa é atualizar as estatísticas de objetos específicos do esquema que foram modificados desde a última atualização e definir a taxa de amostragem.

Exemplo 2:

Para atualizar forçadamente os valores das estatísticas para todos os objetos do esquema:

```
dmSQL> UPDATE STATISTICS ALL;
```

Exemplo 3:

Para atualizar as estatísticas de todas as tabelas, incluindo colunas, índices e tabelas do sistema, com o valor padrão da taxa de amostragem de 30:

```
dmSQL> UPDATE STATISTICS SAMPLE=30;
```

Exemplo 4:

Para atualizar as estatísticas da tabela **jeff.tb_staff**:

```
dmSQL> UPDATE STATISTICS jeff.tb_staff;
```

Exemplo 5:

Para atualizar as estatísticas das tabelas **jeff.tb_staff** e **jeff.tb_dept**:

```
dmSQL> UPDATE STATISTICS jeff.tb_staff, jeff.tb_dept;
```

Exemplo 6:
Para atualizar as estatísticas no banco de dados com **mode=0**:

```
dmSQL> UPDATE STATISTICS mode=0;
```

Exemplo 7:
Para atualizar as estatísticas no banco de dados com taxa de amostragem decidida automaticamente:

```
dmSQL> UPDATE STATISTICS mode=2;
```

Auto Update Statistics Daemon

O DBMaker fornece um daemon para atualizar automaticamente as estatísticas de todo o banco de dados. Nem todas as estatísticas de todas as tabelas são regeneradas, mas sim uma taxa de amostragem otimizada com base na última vez em que as estatísticas foram atualizadas para uma tabela e em quanto a tabela foi alterada desde a última atualização. Os usuários podem selecionar o modo de atualização de estatísticas por conta própria, definir taxas de amostragem diferentes para diferentes tabelas e alterar o tempo inicial e o intervalo de atualização das estatísticas. Os usuários podem consultar a tabela do sistema **SYSUSER** para obter o status de atualização das estatísticas. O status é uma string armazenada na coluna **SQL_CMD**. Além disso, se o comando de atualização de estatísticas estiver sendo executado, os usuários podem interrompê-lo com o procedimento armazenado **setSystemOption()**, e a conexão não será interrompida.

Observe que o daemon de atualização de estatísticas não funcionará se o servidor de atualização de estatísticas não existir ou tiver sido encerrado. E os usuários não podem definir estatísticas para tabelas temporárias.

Os usuários podem melhorar o desempenho da atualização das estatísticas pelos seguintes métodos: configuração do **dmconfig.ini**, configuração de cada tabela, **setSystemOption**, obtenção do status de atualização das estatísticas e interrupção do comando de atualização das estatísticas.

DMCONFIG.INI SETTING

O DBMaker possui algumas palavras-chave de configuração para ativar o daemon de atualização de estatísticas. Configurar o parâmetro **DB_StSvr** como 1 no arquivo **dmconfig.ini** ativa o Daemon de Atualização Automática de Estatísticas. A palavra-

chave **DB_StMod** especifica o modo de atualização de estatísticas (modo geral ou modo de configuração para cada tabela) para o banco de dados. A palavra-chave **DB_StsTm** especifica o horário de início para a atualização das estatísticas, a palavra-chave **DB_StsTv** especifica o intervalo do daemon de atualização de estatísticas e a palavra-chave **DB_StsSp** é usada para definir a taxa de amostragem da atualização das estatísticas.

Exemplo 1:

Antes de iniciar o banco de dados, configure algumas palavras-chave relacionadas ao daemon de atualização de estatísticas no arquivo **dmconfig.ini**:

```
[DBNAME]
; Here omit other keywords
DB_StSvr = 0
DB_StMod = 1
DB_StsTm = 2010-10-10 10:00:00
DB_StsTv = 12:00:00
DB_StsSp = 70
```

Agora, inicie o banco de dados, e o daemon de atualização de estatísticas atualizará automaticamente as estatísticas de acordo com o cronograma.

Exemplo 2:

Permitir que o usuário defina todos os parâmetros sobre o daemon de atualização de estatísticas durante o tempo de execução:

```
dmSQL> CALL SETSYSTEMOPTION('STSVR','1'); //activate automatic update statistic server

dmSQL> CALL SETSYSTEMOPTION('STMOD','1'); //reset DB_StMod

dmSQL> CALL SETSYSTEMOPTION('STSTM','2009/6/6 20:30:00'); //reset DB_StsTm
dmSQL> CALL SETSYSTEMOPTION('STSTV','7-00:00:00'); //reset DB_StsTv

dmSQL> CALL SETSYSTEMOPTION('STSSP','60'); //reset DB_StsSp
```

EVERY TABLE SETTING

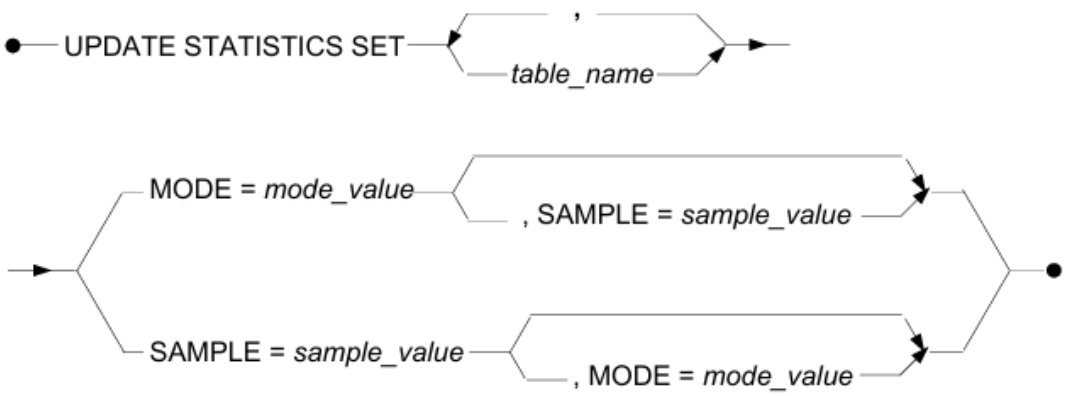
Se os usuários configurarem a opção de atualização de estatísticas para cada tabela executando o comando SQL **UPDATE STATISTICS SET**, há quatro condições de filtro conforme descrito abaixo:

- **Se for uma tabela nova**, ou seja, a tabela nunca passou por uma atualização de estatísticas, então será executada a atualização automática de estatísticas.
- **Se o número total de páginas na tabela for menor que 20 páginas**, então será executada a atualização automática de estatísticas.
- **Se o número total de páginas na tabela for maior que 20 páginas** e o número de novas páginas for superior a 2 páginas desde a última atualização automática de estatísticas, então será executada a atualização automática de estatísticas.
- **Se a tabela não atualizar estatísticas por mais de 10 dias**, será executada a atualização automática de estatísticas.

Se o daemon de atualização de estatísticas for iniciado no modo de configuração para cada tabela (ou seja, o valor de **DB_StMod** é 1), os usuários podem configurar a opção de atualização de estatísticas para cada tabela executando o comando SQL **UPDATE STATISTICS SET**.

- Se o valor de **MODE** no comando **UPDATE STATISTICS SET** for **0**, a taxa de atualização das estatísticas da tabela será decidida pelo valor de **DB_StsSp** no arquivo **dmconfig.ini**, mas considerando as quatro condições de filtro mencionadas acima.
- Se o valor de **MODE** no comando **UPDATE STATISTICS SET** for **1**, a taxa de atualização das estatísticas da tabela será decidida pelo valor de **SAMPLE** no comando **UPDATE STATISTICS SET**, mas considerando as quatro condições de filtro mencionadas acima.
- Se o valor de **MODE** no comando **UPDATE STATISTICS SET** for **2**, a taxa de atualização das estatísticas da tabela será decidida pelo valor de **SAMPLE** no comando **UPDATE STATISTICS SET**, **sem considerar** as quatro condições de filtro mencionadas acima.

As informações de configuração de cada tabela serão armazenadas na tabela de sistema **SYSTABLE**. A coluna **UPD_STS_MODE** armazenará o modo de atualização de estatísticas da tabela, e a coluna **UPD_STS_SAMPLE** armazenará a taxa de amostragem de atualização de estatísticas da tabela.



Exemplo 1:

Configurando o modo de atualização de estatísticas e a taxa de amostragem para a tabela **jeff.tb_staff**:

```
dmSQL> UPDATE STATISTICS SET MODE=1, SAMPLE=50 FOR jeff.tb_staff;
dmSQL> SELECT TABLE_NAME, TABLE_OWNER, UPD_STS_MODE, UPD_STS_SAMPLE
FROM
SYSTABLE;


TABLE_NAME    TABLE_OWNER    UPD_STS_MODE   UPD_STS_SAMPLE
========== ============== ============= ================
TB_STAFF         JEFF             1               80

1 rows selected
```

Exemplo 2:

Configurando o modo de atualização de estatísticas e a taxa de amostragem para as tabelas jeff.tb_staff e jim.tb_salary:

```
dmSQL> UPDATE STATISTICS SET jeff.tb_staff, jim.tb_salary MODE = 1, SAMPLE = 60;
dmSQL> SELECT TABLE_NAME, TABLE_OWNER, UPD_STS_MODE, UPD_STS_SAMPLE
FROM SYSTABLE;

TABLE_NAME    TABLE_OWNER   UPD_STS_MODE    UPD_STS_SAMPLE

========== ==============  =============== ===============

TB_STAFF           JEFF               1                60

TB_SALARY          JIM                1                60
```

```
2 row selected
```

Exemplo 3:

Se uma tabela for muito grande, o comando **UPDATE STATISTICS** pode levar muito tempo, atrasando comandos subsequentes. Nesse caso, é possível usar o comando **UPDATE STATISTICS SET** para definir a taxa de amostragem de tabelas específicas. Por exemplo: pular a tabela **t1** (definindo a taxa de amostragem como 0) e atualizar as estatísticas na tabela **t2** com uma taxa de amostragem de 80%.

```
dmSQL> UPDATE STATISTICS SET t1 mode=1 sample=0;

dmSQL> UPDATE STATISTICS SET t2 mode=1 sample=80;

dmSQL> UPDATE STATISTICS mode=1;
```

SETSYSTEMOPTION

A opção de sistema para atualização automática de estatísticas inclui STSVR, STMOD, STSTM, STSTV e STSSP. O DBMaker oferece suporte para configurar essas opções de sistema em tempo de execução, chamando o procedimento armazenado do sistema **setSystemOption()**, e agora adiciona um novo procedimento armazenado do sistema, **setSystemOptionW()**, para permitir a configuração dessas opções de sistema em tempo de execução e salvar as configurações no arquivo **dmconfig.ini**. Além disso, os usuários podem obter informações sobre essas opções de sistema chamando o procedimento armazenado do sistema **getSystemOption()**. Para mais informações sobre como usar os três procedimentos armazenados do sistema **setSystemOption()**, **setSystemOptionW()** e **getSystemOption()**, consulte o **SQL Command and Function Reference** e o **ODBC Programmer's Guide**.

Exemplo 1:
Configurar a opção de sistema **STSVR** para 0 enquanto o banco de dados está em execução:

```
dmSQL> CALL SETSYSTEMOPTION('STSVR','0');
```

Configurar a opção de sistema **STSVR** para 1 e gravar **DB_StSvr = 1** na seção atual do banco de dados no arquivo **dmconfig.ini** enquanto o banco de dados está em execução:

```
dmSQL> CALL SETSYSTEMOPTIONW('STSVR','1');
```

Obter o valor da opção de sistema **STSVR** enquanto o banco de dados está em execução:

```
dmSQL> CALL GETSYSTEMOPTION('STSVR',?);
```

Exemplo 2

Configurar a opção de sistema **STSSP** para 70 enquanto o banco de dados está em execução:

```
dmSQL> CALL SETSYSTEMOPTION('STSSP','70');
```

Configurar a opção de sistema **STSSP** para 30 e gravar **DB_StsSp = 30** na seção atual do banco de dados no arquivo **dmconfig.ini** enquanto o banco de dados está em execução:

```
dmSQL> CALL SETSYSTEMOPTIONW('STSSP','30');
```

Obter o valor da opção de sistema **STSSP** enquanto o banco de dados está em execução:

```
dmSQL> CALL GETSYSTEMOPTION('STSSP',?);
```

NOTA: *Nem todas as palavras-chave de configuração podem ser alteradas durante o tempo de execução.*

UPDATE STATISTIC STATUS

O usuário pode consultar a tabela de sistema **SYSUSER** para obter o status de atualização de estatísticas, que é uma cadeia de caracteres armazenada na coluna **SQL_CMD**.

A seguir, encontra-se a informação sobre o status de atualização de estatísticas:

```
[EXEC] update statistics command // (Total, start_time, execute_time, remain_time, complete_percent) (table_name, start_time, execute_time, remain_time, complete_percent) (index_name, start_time, execute_time, remain_time, complete_percent)
```

O status de atualização de estatísticas inclui 3 partes: **total**, **tabela** e **índice**.

- **Total** significa todos os objetos de atualização de estatísticas.
- **Tabela** significa as estatísticas de uma tabela, incluindo os índices pertencentes a ela e a página de dados.

- **Índice** significa as estatísticas do índice ou as estatísticas da página de dados.

Essas três partes representam os diferentes níveis de estatísticas que podem ser atualizadas no banco de dados.

Exemplo:

```
dmSQL> SELECT CONNECTION_ID, SQL_CMD FROM SYSUSER;

CONNECTION_ID                                SQL_CMD
==================== ========================================

        3264 [EXEC] Update t1 set c1 = c1 + 1;

        3267 [EXEC] update statistics // for table SYSADM.HUNDRED index
HUNDRED_CODE start at 2011/02/21 09:19:54

2 rows selected
```

MONITOR AND ABORT UPDATE STATISTICS

O DBMaker oferece suporte para exibir o status da atualização de estatísticas. Você pode consultar a coluna **SQL_CMD** da tabela **SYSUSER** para monitorar o progresso da atualização de estatísticas. Além disso, você pode chamar o procedimento armazenado do sistema **setSystemOption('STS_ABORT', 'connection_id')** para abortar uma atualização de estatísticas em andamento.

Para obter informações sobre a coluna **SQL_CMD** da tabela de sistema **SYSUSER** e o uso do procedimento armazenado **setSystemOption()**, consulte o **JDBA Tool User's Guide** e o **SQL Command and Function Reference**.

Exemplo 1:

Consultar a coluna **SQL_CMD** da tabela **SYSUSER** para monitorar o progresso da atualização de estatísticas:

```
dmSQL> SELECT SQL_CMD FROM SYSUSER;
```

Exemplo 2:

Abortar o comando de atualização de estatísticas com o ID de conexão 14076:

```
dmSQL> CALL SETSYSTEMOPTION('STS_ABORT','14076');
```

Exemplo 3:

O valor 0 é um ID de conexão especial. Ele significa abortar todas as conexões relacionadas à atualização de estatísticas:

```
dmSQL> CALL SETSYSTEMOPTION('STS_ABORT','0');
```

Load and Unload Statistics

Os usuários podem usar o comando **UNLOAD STATISTICS** para despejar os valores das estatísticas em um arquivo de texto externo. Os usuários também podem usar o comando **LOAD STATISTICS** para copiar os valores das estatísticas para um banco de dados a partir de um arquivo de texto externo.

Exemplo 1
Para usar o **UNLOAD STATISTICS**:

```
dmSQL> UNLOAD STATISTICS TO file1;  // Despeja as estatísticas do
banco de dados para o arquivo file1
```

Exemplo 2
Para usar o **LOAD STATISTICS**:

```
dmSQL> LOAD STATISTICS FROM file1;  // Lê as estatísticas do arquivo
de texto externo file1
```

Um usuário experiente pode melhorar a eficiência da consulta modificando arquivos com estatísticas e inserindo-os no banco de dados.

Exemplo 3

Um arquivo de texto externo gerado pelo comando **UNLOAD STATISTICS** pode ter o seguinte formato:

```
DBname = TESTDB

TBowner = jeff

TBname = tb_staff

TBpage = 5

TBrows = 30

Tbavlen = 50
```

```
COname = age

COtype = INTEGER

COdist = 12

COavlen = 4

COlow = 25

COhigh = 42


IXname = idxage

IXpages = 5

IXlevel = 2

IXleaf = 3

IXdist = 12

IXdistC1 = 12

IXdistC2 = 12

IXdistC3 = 12

IXpgkey = 8

IXcount = 7
```

20.5 Accelerating Execution of Query

A execução de uma consulta pode ser acelerada ao realizar as seguintes modificações:

- **Ler menos linhas de dados**: Filtrar ou limitar os dados retornados pela consulta pode reduzir o tempo de execução.
- **Evitar ordenação ou ordenar menos linhas de dados e colunas**: A ordenação pode ser uma operação cara, por isso ordenar apenas as colunas e linhas necessárias ajuda a melhorar a performance.
- **Ler os dados sequencialmente**: A leitura sequencial de dados tende a ser mais eficiente do que acessar dados de forma aleatória, reduzindo a sobrecarga no acesso aos dados.

Data Model

A definição de um modelo de dados inclui todas as tabelas, visualizações (views) e índices no banco de dados, com ênfase na existência de índices. Ele descreve se um índice deve ser utilizado em condições como **join**, **ordenação** e **visualizações** (views).

Query Plan

Use o comando **Set Dump Plan ON** para verificar o plano de execução da consulta no DBMaker.

Características de um plano de execução:

- **Índice** — verifica os dados de saída para verificar se um índice foi usado ou não e, em caso afirmativo, como ele foi utilizado. Isso ajuda a entender se os índices estão sendo aproveitados corretamente para melhorar o desempenho da consulta.
- **Filtro** — verifica os fatores de predicado para ver quanto dos dados o predicado pode filtrar. Isso permite avaliar a eficácia das condições de filtragem e como elas afetam a quantidade de dados processados.
- **Consulta** — verifica a consulta após sua execução para ver se o plano de acesso é o melhor possível. Isso envolve a análise de como o banco de dados escolhe os métodos de acesso aos dados (por exemplo, leitura sequencial ou usando índices) e se há oportunidades para otimização adicional.

Exemplo:

Para visualizar o plano de execução, você pode usar o seguinte comando:

```
dmSQL> SET DUMP PLAN ON;
```

Index Check

Para verificar se os índices apropriados existem nas colunas de uma consulta, você pode seguir os métodos descritos a seguir para melhorar a eficiência da consulta:

Filter Columns

Basta uma pequena parte das informações da fonte para fazer uma consulta eficiente. Os usuários podem usar o predicado WHERE em um comando SELECT para controlar a quantidade de informações de saída, conhecido como filtro de dados. A seguir, estão alguns métodos para usar o predicado WHERE avançado:

AVOIDING CORRELATED SUB-QUERIES

Subconsultas correlacionadas existem quando colunas duplicadas aparecem na consulta principal e na subconsulta de um predicado WHERE. Um resultado diferente é retornado para uma subconsulta duplicada para cada linha de dados contida na consulta principal. Se os dados das colunas em cada linha forem diferentes dos da linha anterior em uma subconsulta, isso equivale a executar uma nova consulta para cada linha obtida da consulta principal. Se você encontrar uma subconsulta que consome muito tempo, primeiro verifique se ela é uma subconsulta correlacionada. Se for, reescreva a consulta para evitar a condição. Se não for fácil reescrever a consulta, tente outro método para reduzir o número de linhas de dados.

AVOIDING DIFFICULT REGULAR EXPRESSIONS

A palavra-chave LIKE fornece uma comparação com curingas, conhecida como expressão regular. Quando um curinga é usado no início de uma expressão, o servidor de banco de dados verificará cada linha, pois não pode usar um filtro de índice. Isso instruirá o DBMaker a acessar sequencialmente e verificar cada linha em uma tabela.

Exemplo:

Usando a palavra-chave LIKE com o curinga "*":

```
dmSQL> SELECT * FROM tb_salary WHERE name LIKE '*st';
```

Query Results

Quando você entende o que uma consulta realmente faz, pode encontrar outra consulta equivalente para obter o mesmo resultado. Apresentamos algumas sugestões para os usuários reescreverem consultas de forma mais eficiente.

• Reescrever junções usando visões

• Evitar ou reduzir a ordenação

• Evitar acessar uma tabela grande sequencialmente

• Usar uniões para evitar acesso sequencial

Temporary Tables

É útil criar uma tabela temporária e ordenada para acelerar uma consulta. Isso também pode ajudar a evitar operações de ordenação em várias colunas, simplificando a operação do otimizador.

• Usar uma tabela temporária para evitar ordenação em várias colunas

• Substituir a ordenação em acessos não sequenciais

20.6 Syntax-Based Query Optimizer

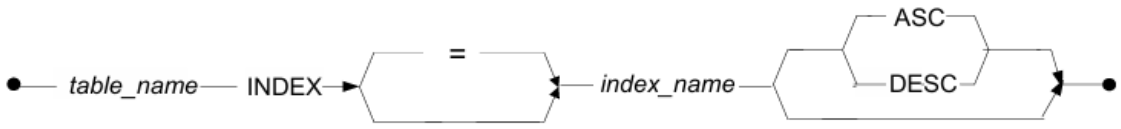
O otimizador escolhe um plano de execução de consulta com a função de custo e estatísticas automaticamente. Em alguns casos especiais, quando a distribuição dos dados é enviesada, o otimizador pode escolher um plano de execução de consulta ruim. Para resolver o problema, o DBMaker suporta um mecanismo de otimizador chamado otimizador de consulta baseado em sintaxe.

Agora, você pode especificar manualmente o tipo de varredura a ser usado em uma consulta e os índices a serem utilizados em uma varredura de índice. Além disso, o otimizador de consultas do DBMaker determina automaticamente o tipo de varredura mais eficiente a ser usado, mesmo que você não tenha atualizado recentemente as estatísticas do banco de dados. Existem cinco casos diferentes em que o tipo de índice a ser utilizado é determinado.

Forced Index Scans

Sintaxe geral usada para forçar uma varredura de índice:

```
tablename (INDEX [=] idxname [ASC|DESC])
```



Exemplo 1:

Para forçar uma varredura na tabela, especifique o valor 0:

```
dmSQL> SELECT * FROM tb_staff (INDEX=0);
```

Exemplo 2:

Para forçar uma varredura de índice em uma chave primária, especifique o valor 1:

```
dmSQL> SELECT * FROM tb_staff (INDEX=1);
```

Exemplo 3:

Para forçar uma varredura de índice no índice **idx_id**:

```
dmSQL> SELECT * FROM tb_staff (INDEX idx_id);
```

Exemplo 4

Permite que o otimizador de consultas decida o tipo de varredura a ser usado na tabela **tb_staff**, mas força uma varredura de índice no índice **idx_id** para a tabela **tb_salary**:

```
dmSQL> SELECT * FROM tb_staff, tb_salary (INDEX idx_id);
```

Forced Index Scan and "Alias"

Sintaxe geral usada para forçar uma varredura de índice e fornecer um alias para a tabela:

```
tablename (INDEX [=] idxname) aliasname
```

Exemplo:

Para forçar uma varredura de índice no índice **idx_id** e fornecer um alias para a tabela:

```
dmSQL> SELECT * FROM tb_staff (INDEX idx_id) a, tb_staff b WHERE
a.id = b.id;
```

Forced Index Scan and "Synonym"

Sintaxe geral usada para forçar uma varredura de índice usando um sinônimo:

```
synonymname (INDEX [=] idxname)
```

Exemplo:
Para forçar uma varredura de índice no índice **idx_id** usando o sinônimo **staff**:

```
dmSQL> SELECT * FROM staff (INDEX idx_id);
```

Forced Index Scan and "View"

Sintaxe geral usada para forçar uma varredura de índice ao criar uma view:

```
viewname (INDEX [=] idxname)
```

Exemplo 1:

Para forçar uma varredura de índice no índice **idx_id** ao criar a view **vi_staff**:

```
dmSQL> CREATE VIEW vi_staff AS SELECT * FROM tb_staff (INDEX
idx_id);
```

Não é possível forçar um índice ao selecionar de uma view.

Exmeplo 2:

Um uso incorreto que retornará erros:

```
dmSQL> SELECT * FROM vi_staff (INDEX idx_id);
```

Forced Text Index Scans

Sintaxe geral usada para forçar uma varredura de índice de texto:

```
tablename (TEXT INDEX [=] idxname)
```

Exemplo:

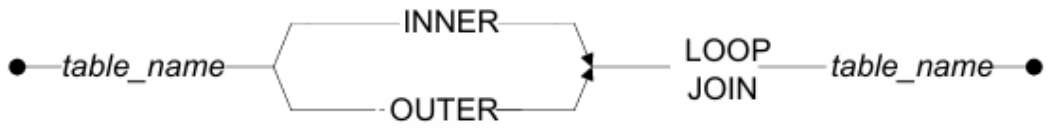
Para forçar uma varredura de índice de texto no índice **tidx1**:

```
dmSQL> SELECT * FROM tb_staff (TEXT INDEX tidx1);
```

Forced Loop Join (Nested Join)

Sintaxe geral usada para forçar uma junção aninhada entre duas tabelas:

```
tablename { INNER | OUTER } LOOP JOIN tablename
```



Nota: *Uma junção forçada desse tipo deve usar a sintaxe* ***INNER JOIN*** *ou* ***OUTER JOIN****.*

Exemplo 1:

```
dmSQL> SELECT * FROM tb_staff INNER LOOP JOIN tb_salary ON
tb_staff.id = tb_salary.id;
```

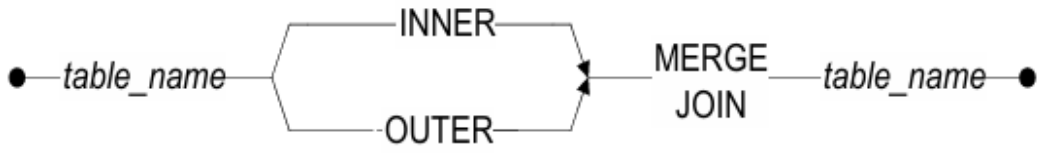
Exemplo 2:

```
dmSQL> SELECT * FROM tb_staff OUTER LOOP JOIN tb_salary ON
tb_staff.id = tb_salary.id;
```

Forced Merge Join

Sintaxe geral usada para forçar uma junção por mesclagem (Merge Join) entre duas tabelas:

```
tablename { INNER | OUTER } MERGE JOIN tablename
```



Nota: *Quando a junção não pode usar uma junção por mesclagem (Merge Join), forçar a junção por mesclagem é inútil, mas não retornará uma mensagem de erro.*

Exemplo 1:

```
dmSQL> SELECT * FROM tb_staff INNER MERGE JOIN tb_salary ON tb_staff.id=
tb_salary.id;
```

Exemplo 2:

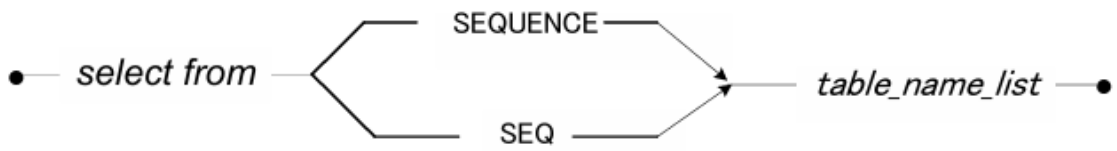
```
dmSQL> SELECT * FROM tb_staff OUTER MERGE JOIN tb_salary ON tb_staff.id=
tb_salary.id;
```

Forced Join Sequence

Force toda a sequência de junção das tabelas, e então a sequência de junção não pode ser alterada

Sintaxe geral usada para forçar a sequência de junção:

```
SELECT ..... FROM [SEQUENCE | SEQ] tablename_list;
```



Exemplo 1:

```
dmSQL> SELECT * FROM sequence tb_staff, tb_salary, tb_dept WHERE tb_staff.id=
tb_salary.id AND tb_salary.basepay=tb_dept.basepay;
```

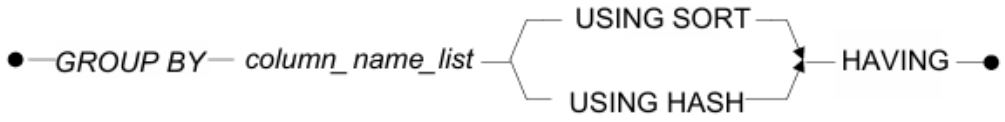
Exemplo 2:

```
dmSQL> SELECT * FROM seq tb_staff INNER JOIN tb_salary ON tb_staff.id= tb_salary.id
INNER JOIN tb_dept ON tb_staff.basepay=tb_detp.basepay;
```

Forced Group by Method

Sintaxe geral usada para forçar uma sequência de junção:

```
GROUP BY column_name_list [USING SORT | USING HASH] having .....
```



Exemplo 1:

```
dmSQL> SELECT id,name,count(*) FROM tb_staff GROUP BY id,name USING HASH;
```

Exemplo 2:

```
dmSQL> SELECT id,name,count(*) FROM tb_salary GROUP BY id,name USING SORT
HAVING sum(basepay)>0;
```

20.7 How to Read a Dump Plan

O primeiro passo para verificar uma consulta lenta é ler o plano de execução. O DBMaker suporta uma função de despejo e leitura de plano de execução.

Existem três comandos dmSQL para despejar planos:

```
dmSQL> SET DUMP PLAN ON;
```

Ativa a opção de despejo de plano. As consultas subsequentes irão despejar o plano e, em seguida, executar os comandos.

```
dmSQL> SET DUMP PLAN OFF;
```

Desativa a opção de despejo de plano. As consultas subsequentes irão apenas executar os comandos, sem despejar o plano. Esta é a opção padrão.

```
dmSQL> SET DUMP PLAN ONLY;
```

Ativa o despejo apenas do plano, mas não executa os comandos.

À primeira vista, parece que um plano de despejo é composto por vários blocos chamados ON. O otimizador de consultas divide uma consulta em vários blocos ON, e cada bloco é uma unidade lógica de otimização. O otimizador irá otimizar cada bloco ON. Consultas simples e unidas geralmente possuem apenas um bloco ON, mas uma consulta complexa, como uma subconsulta, pode gerar mais de um bloco ON, incluindo um bloco principal e subblocos.

O otimizador encontra o melhor método de execução baseado no custo para cada bloco ON. Ele irá dividir um bloco ON em vários blocos PL, onde cada bloco PL representa uma operação, como scan, join, etc.

Familiarize-se com os termos introduzidos nas seções anteriores deste capítulo: • varredura de tabela (table scan)

- Table scan
- Index scan
- Nested join
- Merge join
- Factor

Table Scan

Exemplo:

Para configurar o despejo de plano para uma varredura de tabela da tabela tb_staff:

```
dmSQL> SET DUMP PLAN ON;

dmSQL> SELECT * FROM tb_staff WHERE id>1;
```

Resultado, o plano de despejo para a varredura de tabela da tb_staff:

```
----- begin dump plan -----

{ON Block 0}

ON Type         : SCAN
```

```
[PL Block 0]
Method          : Scan
Table Name     : tb_staff
Type              : Table Scan
Order             : <none>
Factors          : (1) tb_staff.id > 1
I/O Cost         : 101.0
CPU Cost       : 25.3
Sub Cost        : 0.0
Result Rows      : 330.0
----- end dump plan -----
```

As duas primeiras linhas fornecem as informações para um bloco ON:

{ON Block 0} — Bloco ON com um ID de bloco 0

ON Type: SCAN — O tipo de bloco ON é uma varredura (scan)

O bloco ON contém um bloco PL:

[PL Block 0] — Um bloco PL com um ID de bloco 0

Method Scan — Este bloco PL executará uma operação de varredura

Table Name: tb_staff — A varredura é na tabela tb_staff

Type: Table Scan — O tipo de varredura é uma varredura de tabela

Order: <none> — Ordem da varredura, não há uso para uma varredura de tabela

Fatores: (1) tb_staff.id > 1 — A varredura utiliza o filtro tb_staff.id > 1

I/O Cost: 101.0 — O custo estimado de I/O é 101.0 páginas

CPU Cost: 25.3 — O custo estimado de CPU é 25.3 páginas

Sub Cost: 0.0 — Soma estimada dos custos para o subbloco do bloco PL. Neste exemplo, não há subbloco PL

Result Rows: 330.0 — Estimativa de linhas resultantes após a varredura e o filtro

Index Scan

Exemplo

Para configurar o despejo de plano para id e nome da tabela tb_salary usando WHERE:

```
dmSQL> SET DUMP PLAN ON;

dmSQL> SELECT id, name FROM tb_salary WHERE id>1 AND name='john';
```

Resultado, o plano de despejo para id e nome da tabela tb_salary usando WHERE:

```
----- begin dump plan -----

{ON Block 0}

ON Type : SCAN

[PL Block 0]

Method              : Scan

Table Name          : tb_salary

Scan Type           : Index Scan on idx21(name, id)

Order               : ASC

Index EQFA#         : 1

Index FA#           : 2

Index FACOL         : 1, 2

Index Cost          : 2

Factors             : (1) tb_salary.name = 'john'

                    : (2) tb_salary.id > 1

I/O Cost            : 2.0

CPU Cost            : 0.6

Sub Cost            : 0.0

Result Rows         : 13.0

----- end dump plan -----
```

As duas primeiras linhas fornecem as informações para um bloco ON:
{ON Block 0} — Bloco ON com um ID de bloco 0
ON Type: SCAN — O tipo de bloco ON é uma varredura (scan). O bloco ON também contém um bloco PL:
[PL Block 0] — Um bloco PL com um ID de bloco 0

Method Scan — O bloco executa uma varredura
Table Name: tb_salary — A varredura é na tabela tb_salary
Scan Type Index Scan on idx21(name, id) — O tipo de varredura é um índice, aplicando um índice em idx21 usando as colunas de índice name e id
Order: ASC — Ordem de varredura do índice em ordem ascendente
Index EQFA#: 1 — Número de fator de igualdade aplicado na varredura de índice, neste exemplo usando tb_salary.name = 'john'
Index FA#: 2 — Número de fator aplicado na varredura de índice, neste exemplo usando tb_salary.name = 'john' e tb_salary.id > 1
Index FACOL: 1, 2 — Mapeamento de ID de fator das colunas do índice. Neste exemplo, mapeia a primeira coluna de índice, name, para o fator (1) tb_salary.name = 'john', e a segunda coluna de índice, id, mapeia para o fator (2) tb_salary.id > 1.
Index Cost: 2 — Custo estimado da página do índice é 2

Factors: (1) tb_salary.name = 'john'

(2) tb_salary.id > 1 — Aplica os filtros tb_salary.name = 'john' e tb_salary.id > 1

I/O Cost: 2.0 — Custo estimado de I/O de 2.0 páginas

CPU Cost: 0.6 — Custo estimado de CPU de 0.6

Sub Cost: 0.0 — Soma estimada dos custos para o subbloco do bloco PL

Result Rows: 13.0 — Estimativa de linhas resultantes após a varredura e o filtro

Equal Join

Exemplo

Para configurar o despejo de plano de tb_staff e tb_salary usando WHERE:

```
dmSQL> SET DUMP PLAN ON;

dmSQL> SELECT * FROM tb_staff, tb_salary WHERE tb_staff.id=tb_salary.id;
```

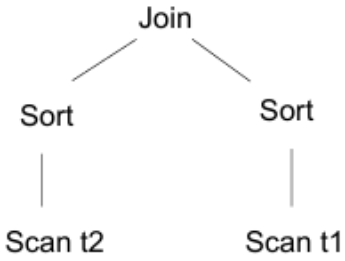
Resultado, o plano de despejo de tb_staff e tb_salary usando WHERE:

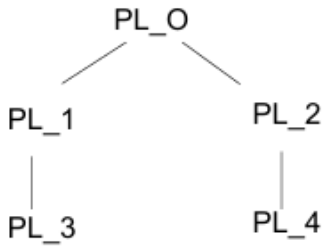
```
----- begin dump plan -----
{ON Block 0}
ON Type            : JOIN
[PL Block 0]
Method             : Join
Type               : Merge Join
Factors            : (1) tb_staff.id = tb_salary.id
I/O Cost           : 8.5
CPU Cost           : 573.8
Sub Cost           : 231.6
Result Rows        : 500.0
Sub Block 1        : [PL Block 1]
Sub Block 2        : [PL Block 2]
[PL Block 1]
Method             : Sort
I/O Cost           : 4.2
CPU Cost           : 274.4
Sub Cost           : 120.0
Result Rows        : 1000.0
SUB Block          : [PL Block 3]

[PL Block 3]
Method             : Scan
```

```
Table Name       : tb_salary
Type             : Table Scan
Order            : <none>
Factors          : <none>
I/O Cost         : 101.0
CPU Cost         : 25.3
Sub Cost         : 0.0
Result Rows      : 1000.0

[PL Block 2]
Method         : Sort
I/O Cost       : 4.2
CPU Cost       : 274.4
Sub Cost       : 120.0
Result Rows    : 1000.0
SUB Block      : [PL Block 4]

[PL Block 4]
Method           : Scan
Table Name       : tb_staff
Type             : Table Scan
Order            : <none>
Factors          : <none>
I/O Cost         : 101.0
```

```
CPU Cost          : 25.3

Sub Cost          : 0.0

Result Rows     : 1000.0

 ----- end dump plan -----
```

Neste exemplo, há mais de um bloco PL. A relação entre os blocos PL é combinada utilizando as informações de subbloco.





Uma árvore simples representando blocos

Substitua cada nó pelos nomes.

Descrições dos blocos de Join:

[PL Bloco 0] -- Um bloco PL com um ID de bloco igual a 0

Method: Join -- O bloco é um Join

Type: Merge Join -- O Join é do tipo merge

Factors: (1) tb_staff.id = tb_salary.id -- Aplica o filtro Join tb_staff.id = tb_salary.id usando um bloco Join

I/O Cost: 8,5 -- O custo estimado de I/O é de 8,5 páginas

CPU Cost: 573,8 -- O custo estimado de CPU é de 573,8 páginas

Sub Cost: 231,6 -- A soma estimada dos custos para o sub-bloco do bloco PL

Result Rows: 500,0 -- Estimativa das linhas resultantes após o bloco Join

Sub Block 1: [PL Bloco 1] -- O primeiro filho do bloco está vinculado ao [PL Bloco 1]

Sub Block 2: [PL Bloco 2] -- O segundo filho do bloco está vinculado ao [PL Bloco 2]

Descrição para um bloco de ordenação:

[PL Bloco 1] -- Um bloco PL com um ID de bloco igual a 1

Method: Sort -- Um bloco de ordenação

I/O Cost: 4,2 -- O custo estimado de I/O é de 4,2 unidades

CPU Cost: 274,4 -- O custo estimado de CPU é de 274,4 unidades

Sub Cost: 120,0 -- A soma estimada dos custos para o sub-bloco do bloco PL

Result Rows: 1000,0 -- Estimativa das linhas resultantes após o bloco de ordenação

SUB Block: [PL Bloco 3] -- O bloco filho deste está vinculado ao [PL Bloco 3]

Os casos de planos de despejo mais comuns que os usuários irão encontrar estão listados acima.

Muitas mudanças serão evidentes nos planos de despejo, mas todos eles consistem nos mesmos elementos: custo de I/O, custo de CPU e linhas resultantes. Se um plano de despejo for muito complexo, use o otimizador baseado em sintaxe discutido anteriormente para tentar outros métodos.