

Academia ABAP With

Hana & Fiori

**Programas de carga/Interface (Upload/Download)**

Muitas das interfaces de comunicação no SAP, tem como sua principal base a troca de arquivo entre sistemas, para isso, os arquivos podem ser recebidos no SAP através de um TXT, CSV, Excel, dentre outros formatos conhecidos e mais utilizados, o SAP também pode gerar arquivos nesse formato, para fornecer para sistemas terceiros (legados) as informações que forem necessárias para essa transferência de dados.

**Formato TXT (Upload/Download)**

Um dos formatos mais conhecidos para geração de arquivos é o TXT, por ser mais leve e de fácil comunicação entre vários sistemas, vejamos abaixo como funciona um programa de upload e download e informações no SAP:

**Upload de Arquivos – File Local:**

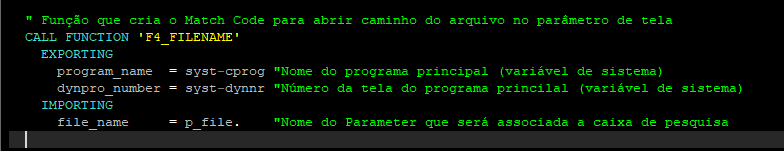
Podemos subir arquivos diretamente de um caminho local no computador, para isso, criamos um PARAMETER associado a uma ajuda de pesquisa para encontrar esse arquivo, o tipo de objeto padrão associado a esse PARAMETER é o RLGRAP-FILENAME, pois este tipo de objeto já está associado a um tamanho de 128 caracteres padrão aos caminhos de arquivo, sua declaração ficaria como na imagem abaixo:



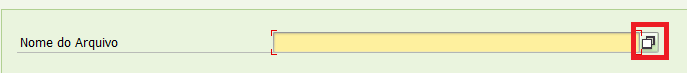
Ao criar o objeto acima, o campo na tela será criado com o tamanho indicado, porém, a caixa de pesquisa para o caminho no Windows não será exibida ao menos que associarmos um evento para que essa pesquisa seja adicionada, veremos mais adiante no curso como os eventos funcionam, mas para não perdemos esse ponto nessa aula, vamos criar o evento abaixo para associar uma ação ao PARAMETER P\_FILE criado, conforme na imagem abaixo:



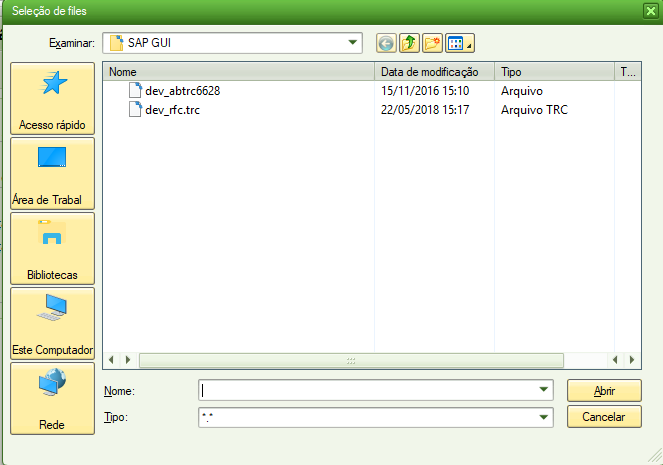
O evento acima indica que uma ação será adicionada ao campo P\_FILE, no caso, chamaremos uma função do SAP que atribui automaticamente a caixa de diálogo do Windows, permitindo a pesquisa pelo arquivo, para isso, basta informar após esse evento o código, conforme abaixo:



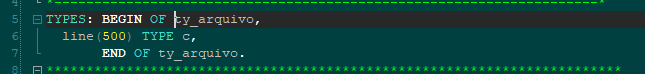
Ao criar a função com os parâmetros corretos, o PARAMETER terá uma opção de pesquisa associado, permitindo ao usuário buscar o caminho do arquivo, conforme as imagens abaixo:



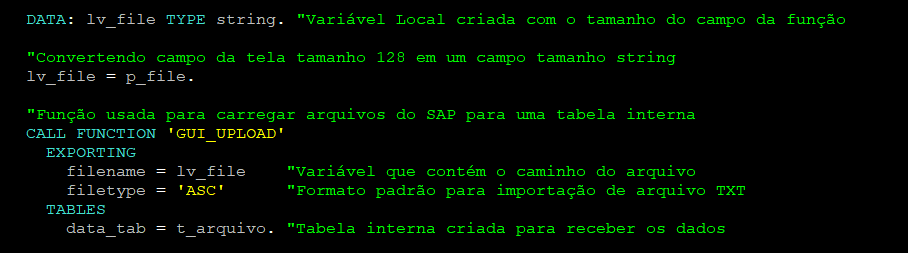
Caixa de diálogo exibida:



Para que o arquivo seja importado para o SAP, precisamos usar uma outra função do SAP, chamada GUI\_UPLOAD, essa função tem diversas utilizações, utilizaremos nesse exemplo o seu conceito mais simples, onde ela lê o nome do arquivo e o converte para uma tabela interna, para isso, antes de mais nada, vamos criar uma tabela interna que possa importar os dados do arquivo de forma completa, nesse exemplo declaramos nossa tabela interna com o tipo abaixo:



O tamanho 500 foi utilizado de modo genérico, em programas de ambientes produtivos, teremos o tamanho exato do que esses arquivos podem conter, então, sempre vamos definir o seu tipo com o tamanho máximo da linha, para que nenhum dado seja perdido durante a importação, após a criação da tabela interna, vamos chamar a função GUI\_UPLOAD, passando os parâmetros obrigatórios para a conversão do arquivo em tabela interna, conforme demonstrado abaixo:



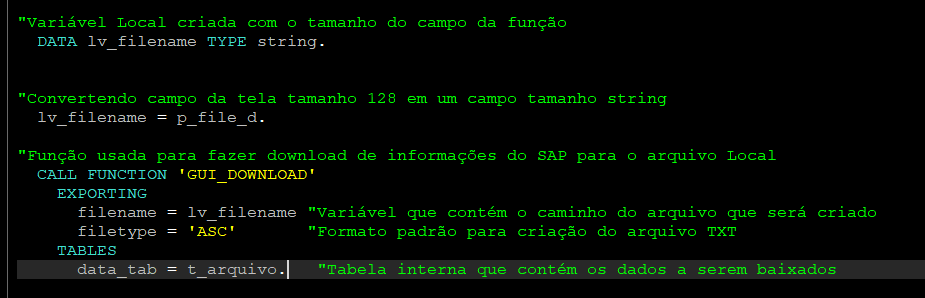
Todas as funções do SAP só podem receber campos definidos exatamente com o seu tipo correto, no caso o campo P\_FILE foi definido com tamanho de 128 para a tela, e o tamanho do campo da função é de formato STRING, devido a isso, fizemos a passagem do campo para uma variável criada localmente, caso contrário a função não interpretará o campo correto causando DUMP durante a importação das informações necessárias.

**Download de Arquivos – File Local:**

Seguindo o mesmo conceito do Upload, precisamos criar um PARAMETER com o tipo RLGRAP-FILENAME para que o usuário possa informar o caminho e o arquivo a ser criado, também é possível associar um evento para encontrar um arquivo, mas a função é um pouco diferente da que usamos para encontrar um arquivo local, já que na criação do arquivo o ideal é que o usuário sugira o local já de modo Default e o arquivo seja gerado via programa, ou seja, dentro da lógica o nome do arquivo seja gerado através de algumas concatenações de data, siglas etc, por isso, no exemplo abaixo, criamos apenas um PARAMETER simples associado ao tipo mencionado:



Após informado na tela o caminho e o nome do arquivo, vamos chamar a função GUI\_DOWNLOAD, que se parece muito com a função GUI\_UPLOAD, conforme demonstrada abaixo:



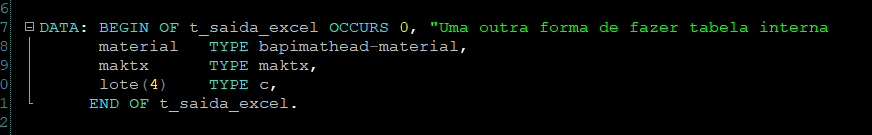
**Formato Excel (Upload/Download)**

Além do formato TXT, também muitas vezes usamos o formato Excel para carregar os dados para o SAP, para isso, seguimos com a mesma lógica para a criação do PARAMETER de importação.

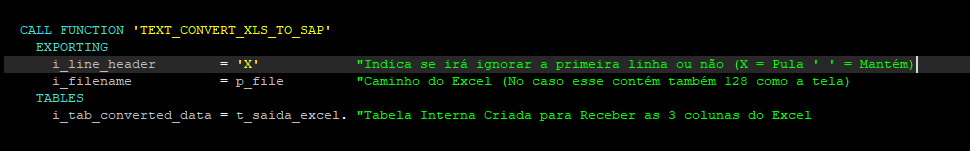
# Upload de Arquivos Excel – File Local:

Para importar um arquivo Excel para o SAP, criamos um PARAMETER da mesma forma que criamos para os formatos TXT, sem nenhuma alteração a única diferença para esse tipo de importação está na declaração da tabela interna que receberá as informações e na função que é chamada para fazer a conversão, vejamos abaixo em um exemplo como isso é realizado.

Foi criada uma tabela interna com 3 campos, esses 3 campos representam as 3 colunas em um Excel, ou seja, caso o Excel tenha 15 colunas, vamos precisar criar 15 campos dentro do nosso TYPES, usamos um exemplo de declaração de tabelas internas mais antigo, embora obsoleto ele fica como um exemplo diferente de como uma tabela interna pode ser criada, segue abaixo:



Após a criação da tabela interna e o número de campos equivalente as colunas do Excel, usaremos a função TEXT\_CONVERT\_XLS\_TO\_SAP para converter os dados do Excel para uma tabela interna do SAP, vejamos abaixo como o preenchimento dessa função deve ocorrer:

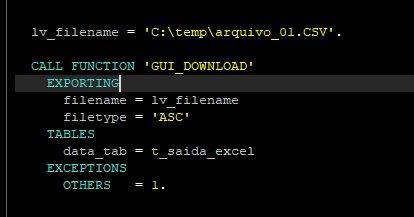


Existe uma diferença em relação a função GUI\_UPLOAD que recebe o caminho do arquivo apenas por um campo do tipo STRING, a função TEXT\_CONVERT\_XLS\_TO\_SAP pode receber diretamente do campo da tela de seleção, desde que o mesmo tenha o

formato de 128 caracteres, pois a função contem o mesmo parâmetro, não causando problemas. O parâmetro i\_line\_header indica se será importada a primeira linha ou apenas a partir da segunda, isso para o caso de arquivos Excel que possam conter linhas de cabeçalho.

### Download de Arquivos Excel – File Local:

Para gerar um arquivo em CSV, o formato padrão em arquivos Excel usados em comunicação entre sistemas, usamos a mesma função GUI\_DOWNLOAD, porém, com a regra que ao invés de informar um arquivo com extensão TXT usamos o formato CSV, e a função fará a conversão automaticamente, no exemplo abaixo, usamos os mesmos dados que foram importados do Excel no exemplo acima para gerar um download em CSV das mesmas informações, segue abaixo o exemplo:



**Diferentes tipos de ALVs e seus conceitos**

Os ALVs são os relatórios gerados para exibir as informações diversas do SAP, muito requisitados em todos os clientes, pois podem fornecer quaisquer tipo de informações programadas além de carregar automaticamente uma série de ferramentas para manuseio desse relatório, abaixo veremos como montar um ALV simples passo a passo e em seguida alguns conceitos de diferentes tipos de ALVs que o SAP pode conter.

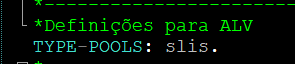
No exemplo abaixo, temos um ALV que não possui tela de seleção, quando executado, o programa faz uma busca dos dados de uma tabela e apresenta diretamente o relatório dos dados encontrados nesta tabela.

A função do ALV, como as demais funções do SAP, precisa de algumas configurações pré-definidas para que possa ser executada, embora a função REUSE\_ALV\_GRID\_DISPLAY tenha muitas ferramentas que possam ser utilizadas a que

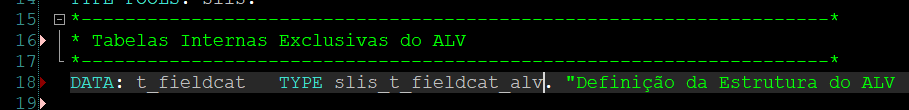
veremos logo abaixo são as essenciais para que o relatório seja exibido de forma simples:

**Tabela de configuração de Fieldcat:**

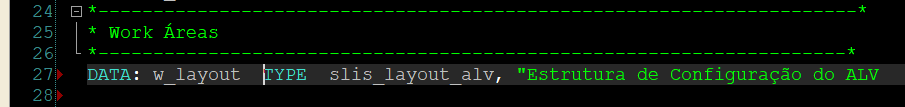
O Fieldcat pode ser comparado o TYPES de uma tabela interna, o ALV precisa saber quais campos e qual o tipo dos campos que serão exibidos em seu relatório, bem como sua descrição, tamanho, etc. Essa estrutura é obrigatória e sem ela não é possível exibir o relatório, para definir essa estrutura, sempre devemos declarar uma tabela interna com o tipo SLIS\_T\_FIELDCAT\_ALV, que pertence a um grupo de tipos do ALV. O grupo de tipos é um conjunto de TYPES já definidos dentro do SAP, ele pode ser acessado pela sintaxe TYPE-POOLS, dentro desse grupo de tipos definimos diversos TYPES que podem ser reutilizados em um programa ABAP, é possível criar um grupo de tipos Z, veremos isso mais adiante no curso, mas é essencial lembrar que para todos os ALVs, vamos precisar definir logo no início do programa a chamada da sintaxe conforme abaixo:



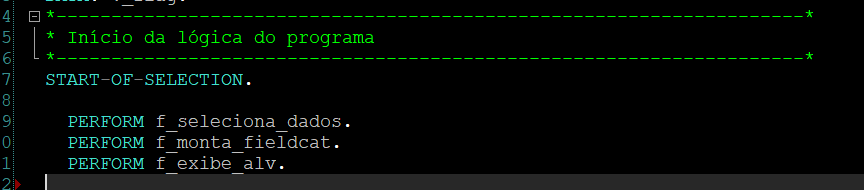
O grupo de tipo SLIS contém diversos TYPES do ALV, inclusive o SLIS\_T\_FIELDCAT\_ALV que vamos usar para referenciar nossa tabela interna do fieldcat, para isso, vamos criar uma tabela interna, conforme a imagem abaixo:



Ainda se tratando da configuração do ALV, podemos definir algumas opções para a exibição do relatório, como colunas otimizadas automaticamente, campos de valores já somados de modo default, o zebra, que faz as linhas ficarem tracejadas para melhor visualização, e para isso usamos uma outra referencia associada ao grupo de tipos SLIS que é SLIS\_LAYOUT\_ALV, nesse caso esse objeto não precisa e nem deve ser criado como tabela, pois possui apenas uma linha, sendo assim, criamos apenas como uma work área, conforme a imagem abaixo:

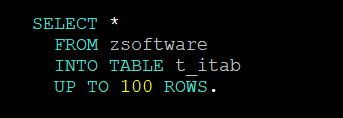


Em nosso programa de exemplo, criamos 3 rotinas de modularização para agrupar os passos do programa, uma para a seleção dos dados que serão exibidos, a outra para montar o fieldcat e a última para exibir o ALV, conforme a imagem abaixo:



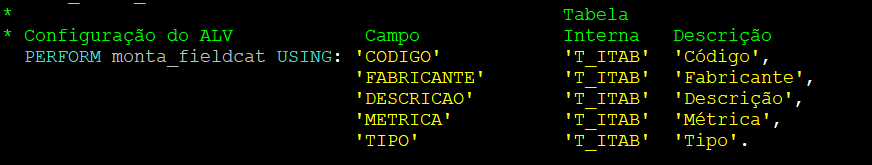
**Seleção dos dados:**

Foi utilizada uma seleção simples em uma tabela Z já existente, para pegar todo o seu conteúdo e jogar em uma tabela interna que foi criada com o nome de t\_itab, ela faz referência a tabela ZSOFTWARE.



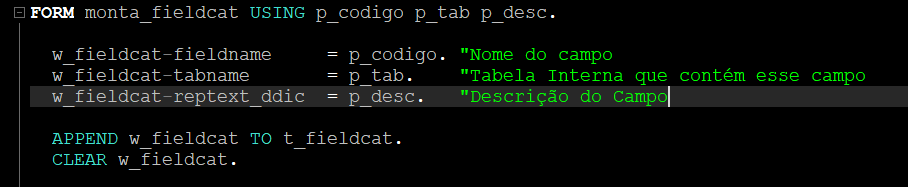
**Montagem do Fieldcat:**

A tabela do fieldcat contém inúmeras configurações que podem ser levadas ao ALV, usamos apenas 3 dos diversos campos que ela pode receber, esses 3 campos informam para o ALV qual é o campo, de qual tabela interna e qual sua descrição a ser exibida no ALV:



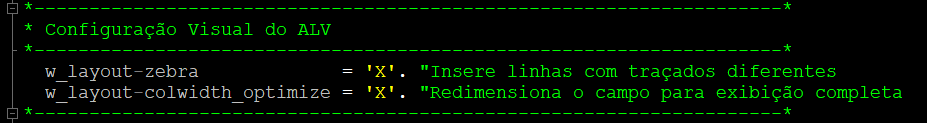
Cada um dos campos exibidos em literais acima, são convertidos em variáveis criadas dentro do form, na mesma ordem em que foram criadas no comando USING, ou seja, P\_CODIGO recebe o valor do nome do campo declarado na chamada do PERFORM, P\_TAB recebe o nome da tabela interna e P\_DESC recebe o nome da descrição do campo que foi informado anteriormente.

Os dados são passados para a work área da estrutura do fieldcat em seguida são inseridos na tabela interna do fieldcat, que neste exemplo terá 3 linhas, contendo a configuração dos campos que serão exibidos no ALV:



### Exibição do ALV:

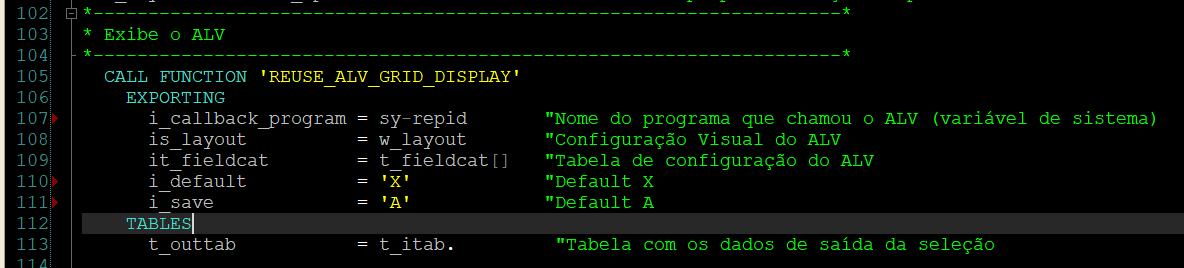
Antes de chamar a função do ALV, definimos algumas configurações para sua exibição, essa estrutura abaixo contém diversas configurações que podem ser personalizadas, usamos apenas o ZEBRA mencionado acima e a otimização das colunas, para que sejam redimensionadas de acordo com o tamanho do texto e do campo:



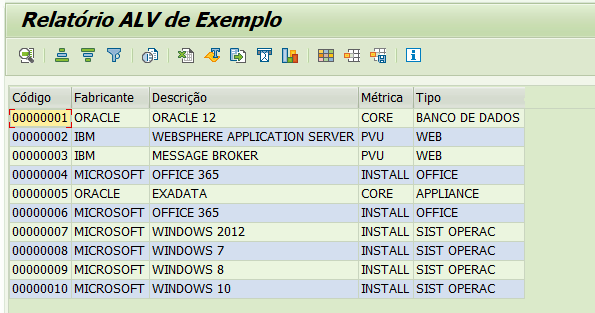
Após feita a configuração de exibição, passamos os parâmetros para a função do alv US, informando os parâmetros básicos essenciais para que o relatório seja exibido.

O parâmetro i\_callback\_program deve sempre receber a variável de sistema com o nome do programa, assim quando o usuário clicar em voltar, o SAP irá retornar para a tela de seleção do programa que chamou a função.

Os parâmetros i\_default e i\_save normalmente são preenchidos como estão abaixo, salvo raras exceções para alvs específicos.



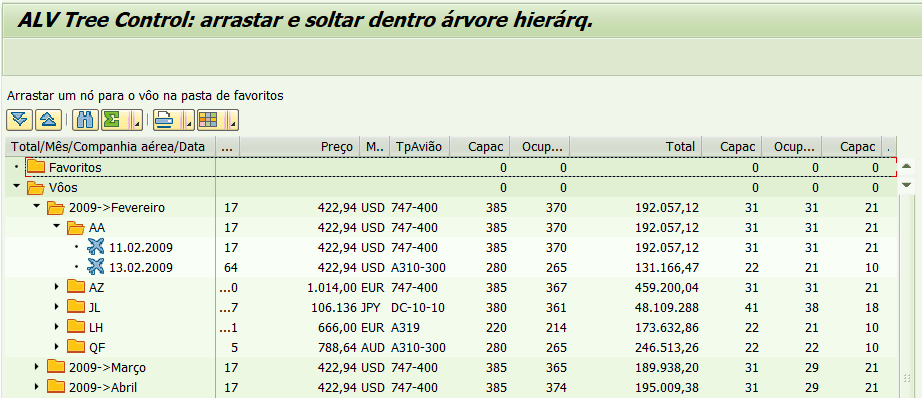
### Resultado do ALV:



As ferramentas exibidas acima do relatório já são configuradas automaticamente pela função, é possível editar essas ferramentas, seja para adicionar ou remover botões, porém as que já vem configuradas não precisam de nenhuma programação ABAP para funcionar, pois a função já tem sua programação realizada.

### ALV Tree

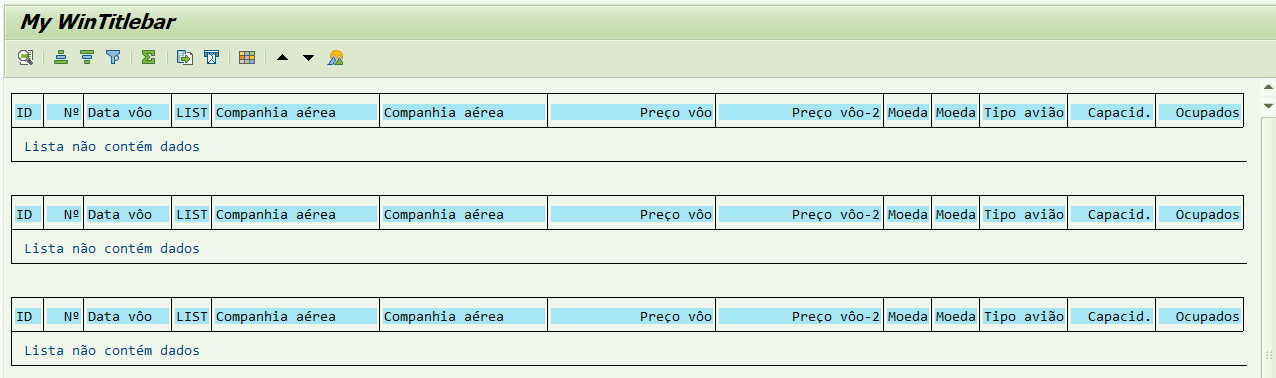
O SAP possui diversos tipos de ALVs mais complexos, a imagem abaixo exibe um ALV com pastas para seus itens, com um conceito de cabeçalho, item e sub itens de registros, embora sua codificação seja um pouco mais ampla, a SAP disponibiliza exemplos desses ALVs, são os chamados BCALV\*, esse abaixo se chama: BCALV\_TREE\_DND\_MULTIPLE



**ALV Múltiplo na mesma tela**

Também é possível exibir mais de um ALV na mesma tela, como o

BCALV\_TEST\_BLOCK\_LIST, esse conceito é chamado de ALV SPLIT, não é muito utilizado, mas é importante que seja conhecido, pois em alguma etapa de algum processo pode ser necessária a criação de algum ALV desse tipo, mesmo que os exemplos do SAP não ajudem, na internet ao usar o termo ALV SPLIT várias referências serão demonstradas sobre esse ALV.



o **Instruções para códigos mais performáticos**

**Aula de Performance Avançada**

### Objetivo

* Troca de informações sobre os pontos mais comuns no mercado de programação ABAP que tem causado problemas graves de performance (clientes e parceiros)
* Recomendações SAP
* Demonstrar alguns casos verídicos
* Demonstrar estatísticas sobre o uso de algumas técnicas de conversão (Batch- Input, Call Transaction, Direct Input)

### Arquitetura Client-Server

O R/3 trabalha com uma filosofia de cliente/servidor de 3 níveis:

Database server : responsável pelo acesso e pela atualização dos dados Application server : responsável pelo processamento da aplicação (tempo

despendido com a interpretação de comandos ABAP)

Frontend : responsável pelo processamento dos gráficos (tempo despendido pelo sistema R/3, ou seja, o middleware e Kernel). Kernel = conjunto de programas utilitários que existe dentro do R/3, servindo ao cumprimento de inúmeras tarefas

A alocação dos servidores de aplicação é definida automaticamente no momento de logon (é checado qual servidor está menos sobrecarregado).

Dica: Para se obter uma otimização da performance em programas ABAP, deve-se minimizar o tempo de acesso ao database.

**Grandes vilões no que se refere à performance**

* Ninhos de select
* Select …. Endselect ao invés de comandos que recuperem do banco de dados todos

os registros de uma única vez

* Select \* ao invés de select com as colunas necessárias ao processamento
* Select single sem chave completa especificada ao invés de Select up to 1 row
* Selects genéricos, ou seja, onde a cláusula where não foi fortemente especificada, com várias condições, visando restringir a seleção
* Índices não utilizados
* Falta de índices
* Select em tabelas com alto número de registros utilizando cláusula where baseada em tela de seleção onde o preenchimento não é obrigatório
* Definição funcional falha
* Grandes tabelas do sistema: BKPF, BSEG, EKKO, EKPO, VBAK, VBAB, MKPF, MSEG, J\_1BNFDOC, J\_1BNFLIN
* Funções genéricas, como por exemplo CLAF\_CLASSIFICATION\_OF\_OBJECTS

**Database Server - Transmissão de pacotes**

**Comunicação entre o Application Server e**

### Premissa:

VBAK contém 1000 registros, sendo que cada registro contém 575 bytes.

### Fato:

A comunicação entre o DB server e o Application server acontece em pacotes de 32.000 bytes, dependendo da rede e equipamentos de comunicação utilizados.

🡺 Select \* from VBAK.

### Portanto:

1000 registros x 575 bytes = 575.000 bytes

575.000 bytes / 32.000 = 20 pacotes

20 pacotes: na verdade este número é um pouco mais alto (mais ou menos 24), uma vez que informações administrativas são transferidas em cada transmissão, juntamente com os dados.

O exemplo acima tenta ilustrar que, se forem selecionadas apenas as colunas necessárias, a transferência de dados será menor e, consequentemente, haverá uma redução significativa no tempo de resposta.

### Select e Logical Database

🡺 Get VBAK

🡺 Get VBAK fields vbeln auart bname kunnr

O segundo comando é muito melhor que o primeiro. Porém, nem todos os logical database suportam select em colunas. Para verificar se isso é possível, acesse a transação SE36, coloque o nome do database no qual se deseja efetuar uma pesquisa e escolha a opção: extras -> field selection.

**Cursor Caching**

Os seguintes comandos SQL produzem o mesmo resultado, mas cada um deles requer seu próprio cursor uma vez que os comandos não são idênticos (verifique a ordem das colunas após a palavra Select e na cláusula WHERE):

Select vbeln auart into (vbak-vbeln, vbak-auart) from vbak where vbeln = nnn and auart = yyyy

Select vbeln auart into (vbak-vbeln, vbak-auart) from vbak where auart = yyyy and vbeln = nnn

Select auart vbeln into (vbak-auart, vbak- vbeln) from vbak where vbeln = nnn and auart = yyyy

Select auart vbeln into (vbak-auart, vbak- vbeln) from vbak where auart = yyyy and vbeln = nnn

Desnecessários comandos Declare e Prepare podem ser evitados se os comandos SQL são mantidos consistentes nos programas que estão sendo desenvolvidos (quando se usar select com colunas ou declarando condições WHERE sempre use a sequência de campos conforme definido no Data Dictionary)

**Logical database e Tables statements**

Report ZAZ\_PERFORMANCE.

Tables: vbak, vbkd, vbpa.

Get vbak fields vbeln auart bname kunnr. Write: vbak-vbeln, vbak-auart, …. .

Get vbpa fields parvw kunnr.

Write: vbpa-parvw, vbpa-kunnr, vbkd-zterm.

Ao invés use:

Get vbak fields vbeln auart bname kunnr. Write: vbak-vbeln, vbak-auart, …. .

Get vbkd field zterm.

Get vbpa fields parvw kunnr.

Write: vbpa-parvw, vbpa-kunnr, vbkd-zterm.

OBS.: A estrutura do logical database VAV (utilizado anteriormente) é VBAK – VBUK –

VBKD – VBPA.

Se uma tabela não é especificada via um cartão TABLES então o sistema automaticamente recupera somente os campos chave para aquela tabela (aqui VBUK não está presente no cartão tables, assim somente os campos chaves são recuperados pelo sistema).

Se uma tabela é especificada via TABLES, todas as colunas são recuperadas, mesmo se nenhum campo daquela tabela é usado no programa (veja a tabela VBKD – somente o campo ZTERM deveria ser impresso).

Para evitar ter todas as colunas selecionadas da tabela, insira um comando GET para aquela tabela, especificando os campos que serão utilizados pelo programa posteriormente.

### Select … exit versus Select …. Up to 1 rows.

**Vantagem em performance:**

Select … exit. Endselect. = 150.000 ms

Select … up to 1 rows = 1.500 ms

### Check inside Select …endselect.

Comparação de Runtime é baseada numa varredura de 57.000 registros na VBAK

|  |  |  |
| --- | --- | --- |
| Select & Check | = | 27.958.000 ms |
| Select com cláusula WHERE | = | 3.065.000 ms |

Em caso de condições de checagens muito complexas poderia ser mais rápido obter dados do banco de dados primeiramente. SQL Trace precisa ser usado para verificar qual método é o mais rápido.

Isso também é válido para select single.

### Selects em campos sem índices:

Select-options: bname\_so for vbak-bname.

Select vbeln bname …. Into (vbak-vbeln …) from vbak

Where bname in bname\_so order by <field> descending.

Endselect.

O sistema standard é distribuído sem um índice para o campo bname. A seleção acima irá resultar numa varredura sequencial da tabela vbak. Isto não é aceitável num sistema em produção. Criando um índice na tabela vbak para o campo bname irá resolver o problema neste caso. Exceção: tabelas cuja buferização é full ou generic (neste caso não tem sentido pois o primeiro acesso irá trazer todos os dados para a memória. Das próximas vezes, o sistema busca do buffer e não do database – até que o dado mude novamente).

### Select sum, avg, min, max versus ABAP calculations

Select matnr kwmeng meins into <internal table>

From vbap where …. .

Collect <internal table>.

Endselect.

### Ao invés use:

Select matnr sum( kwmeng ) meins into table <internal table>

From vbap where ….

Group by matnr meins.

O ganho em performance para o processamento de 10.000 registros: Select + collect = 2.370.000 ms

Select sum …. Into table = 1.574.000 ms

OBS.: o group by não pode ser usado para tabelas pool e cluster.

### Select sem cláusula where

Select col1 col2 …. Into …. from vbak.

Perform calculate\_stuff.

Endselect.

Um select sem cláusula WHERE idnica um erro no design do programa, especialmente se o select é usado em tabelas SAP com grande crescimento/tamanho em pouco tempo, em geral, BKPF, BSEG, COBK, COEP, LIPK, LIPS, MKPF, MSEG, VBAK, VBAP, VBPA, VBFA.

Programas irão rodar de forma correta e bem na entrada em produção, todavia a performance irá decrescer à medida que dados são adicionados diariamente.

### Table buffering

Para analisar a buferização de tabelas use função de chamadas estatísticas: Transaction ST02 -> details analysis menu -> call statistics (tables). Esta função ajuda você a determinar se tabelas desenvolvidas pelo cliente são buferizadas corretamente.

Para analisar a qualidade da buferização de tabelas use a transação ST02, dê duplo click em TABLES (generic key) e clique no pushbutton “buffered objects”.

Por que buferizar ?

Usando buferização pode-se reduzir consideravelmente o tempo que se leva para recuperar um registro.

### Tipos de buferização:

1. full: resident buffer (100 %): o conteúdo de toda a tabela é carregado no buffer no primeiro acesso realizado na tabela
2. generic: uma chave genérica (primeiros “n” campo chaves) é especificado quando da criação da tabela, quando se mantém technical settings. Esta chave genérica divide o conteúdo da tabela em chamadas áreas genéricas. Quando acessando qualquer dado com uma deteminada/especificada chave genérica, toda a área genérica é carregada no buffer.
3. Partial (single record): somente registros únicos são lidos do banco de dados e armazenados no buffer.

### Quando bufferizar uma tabela ?

Uma tabela deveria ser buferizada quando:

* Menor em tamanho (poucas linhas ou poucas colunas com tamanho de campos pequenos)
* acessada muito mais para leitura
* alterações não ocorrem frequentemente

### Tabelas que são boas candidatas a buferização:

* tabelas de controle/tabelas de customização
* “small” master data tables (ex.: 100 material master recores -> few changes)

### Como buferizar uma tabela ?

a) Como ativar/desativar uma buferização:

* dictionary maintenance (SE11 -> technical settings)
* technical settings for a table (transaction SE13)

a) Decidindo sobre o tipo de buferização:

* full (resident = 100 %)
* generic (with number of key fields for the generic key)
* single records (partial buffering)

### Comandos SQL “bypassing” a buferização:

* select … bypassing buffer
* select from database views (projection views are ok)
* select … distinct
* select … count, sum, avg, min, max
* select … order by (outros além da chave primária)
* select … for update
* Cláusula where contém o comando IS NULL
* Comandos natives do SQL (EXEC SQL … ENDEXEC) – Evite esses commandos quando estiver trabalhando com tabelas buferizadas!!!

### Outros responsáveis pela degradação da performance

* Move corresponding é interessante para pequenas tabelas ou quando alguns campos (mas não todos) precisam ser movimentados.
* Não use sort - appends. Ele consome muito tempo de CPU e os resultados são frequentemente imprevisíveis.
* Sempre especifique ASCENDING ou DESCENDING em processamentos com SORT. Qualificar todos os comados SORT através da cláusula BY <data fields> melhora a performance e a leitura do código, evitando assim que todos os campos do registro sofram classificação

∙

* Use COMPUTE ao invés de ADD, SUB, MULTI, DIV desde que performance não seja um grande problema. O comando COMPUTE é mais fácil de implementar e com ele a leitura do programa se torna mais fácil. A diferença em performance é minuto, todavia se o objetivo é conseguir tempo de resposta de subsegundos para transações complexas, então deve-se usar comandos separados
* Quando da definição de uma tabela no dicionário de dados do R/3 os campos chave sempre devem ser colocados nas primeiras posições do registro da tabela (primeiras colunas). O banco de dados comprime os dados para todos os campos da tabela, mas a compressão não ocorre para campos chave. A compressão também só é possível para as colunas após o último campo chave: isso explica o porque de agrupar todos os campos chaves nas primeiras posições
* Transparent e Pool Tables: sempre que for feito um select para buscar dados de tabelas deste tipo pode-se qualificar a cláusula WHERE com campos chave e não

chave, ou seja, quanto mais condições puderem ser definidas, melhor para a obtenção de performance. Isto permite que o banco de dados avalie os registros e retorne somente aqueles que combinem com o critério de seleção definido

* Cluster table: ocorre exatamente o inverso: devido a forma de armazenamento destas tabelas, na cláusula WHERE deve constar apenas os campos chaves e os demais campos devem ser checados através do comando CHECK. O banco de dados não consegue processar cluster tables como ele processa transparent tables. Forçando o banco de dados a descompactar e checar campos (em caso de Select com campos não chaves na cláusula WHERE) é menos eficiente, na maioria dos casos, que qualificando somente com campos chaves e deixando o CHECK para os campos não chave após eles terem sido retornados. Na verdade ambos modos irão funcionar, porém um deles será ineficiente pois irá requerer grande parte da memória disponível, espaço em buffer e tempo do banco de dados para descompactar campos não chave, o que certamente irá refletir em performance. Em algumas tabelas Cluster, mesmo a seleção com restrições através das chaves primárias torna-se lenta, compensando, nestes casos, a seleção da tabela inteira e jogando-a para uma tabela interna e a partir desta fazer as restrições.
* Campos que serão comparados na cláusula WHERE devem ter atributos similares. Se isso não ocorre, o sistema tem que converter o dado toda vez que a comparação é feita. Quando isso não é possível, mova o dado a ser comparado para uma área

auxiliar, antes de efetuar a comparação (esta área auxiliar deve ser definida com o mesmo tipo e tamanho da coluna da tabela)

* Evitar classificações em várias tabelas e, posteriormente, a realização de loops. Ou seja, evitar construções do tipo SORT tab1, SORT tab2, …, e depois o LOOP tab1, LOOP tab2 …. . Isso funciona, porém irá requerer mais memória e recursos para possuir todos os dados resultantes da classificação até que eles sejam usados. Ao contrário, quando se faz SORT seguido de LOOP, o espaço é liberado ao término do loop

### Quando usar internal table ou Dictionary structure

Para tabelas que são usadas repetidamente ou por vários programas, considere a possibilidade de criar estruturas de dados no Data Dictionary (e abrí-las através do comando TABLES) ao invés de definí-las dentro de programas individuais (através do comando DATA …. ENDDATA).

Sobre estas circunstâncias, o uso de tabela interna através do Data Dictionary é preferível:

* a tabela interna é grande
* a tabela é acessada da mesma forma toda vez
* a mesma tabela interna é usada em vários programas
* o programa irá processar várias tabelas internas ao mesmo tempo (e então precisa de muita memória)
* o uso de memória é um ponto chave para otimização de um programa em particular

Use o comando FREE para liberar memória alocada a tabelas internas. Este comando deve seguir o último comando para processamento do dado na tabela.

### Use o comando FREE sob as seguintes condições:

* a tabela interna é grande
* a tabela interna é classificada e reprocessada várias vezes
* o programa está processando várias tabelas internas (e portanto necessita de grande quantidade de memória)
* o uso de memória é importante para um programa em particular ser otimizado: um LOOP AT …. WHERE é preferível a um LOOP AT … CHECK … ENDLOOP porque estará sendo reduzido o número de comandos a serem interpretados

**Processamento de grandes tabelas**

Quando estiverem sendo manipuladas grandes tabelas é muito importante que seja processada a maior quantidade possível de informação num primeiro momento e, se possível, eliminar qualquer outra passagem nesta mesma tabela. Muitas vezes isso requer o uso de tabelas internas. Cada situação deve ser analisada e avalidada, para ver qual o procedimento correto a ser tomado: se armazenar o dado ou fazer novo acesso à tabela do Data Dictionary. Considere as seguintes questões:

qual o tamanho da tabela original comparado ao subset que seria armazenado em uma tabela interna ?

quanto espaço para armazenamento seria necessário para armazenar o dado em uma taela interna ?

Se o processamento precisar ocorrer somente uma vez (por exemplo, armazenar alguns dados para posterior comparação), esteja certo que isso acontecerá for a de uma estrutura em forma de looping (por exemplo, Select, Loop, Do), de forma que o processamento não seja repetido desnecessariamente.

Dica: o operador IN consome muito tempo de máquina e não deveria ser usado em lugar do operador EQ .

Expressões lógicas são avaliadas da esquerda para a direita. A avaliação termina quando o resultado final foi estabelecido (eliminação ou inclusão completa). Todavia, quando se utiliza os operadores AND ou OR (por exemplo, em IF, WHERE) o critério de eliminação mais comum deve ocorrer primeiramente.

Exemplo:

A seguinte tabela deve ser lida, imprimindo-se os empregados da companhia ABC na Georgia

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPLOYEE** | **NAME** | **COMPANY** | **STATE** |
| 001 | Doe, Jr. | ABC | TX |
| 002 | Doe, M. | ABC | OK |
| 003 | Jones, A. | XYZ | TX |
| 004 | Jones, B. | ABC | GA |
| 005 | Jones, C. | ABC | TX |
| 006 | Jones, D. | XYZ | GA |
| 007 | Jones, E. | ABC | TX |
| 008 | Smith, A. | ABC | GA |
| 009 | Smith, B. | ABC | TX |
| 010 | Smith, C. | ABC | OK |

if company = ‘ABC’ and state = ‘GA’

write …..

endif.

🡺 isto funciona, mas tanto os campos company quanto state precisam ser avaliados para oito de dez campos

if state = ‘GA’ and company = ‘ABC’

write …..

endif.

🡺 isto necessita menos tempo para ser processado, porque ele elimina todos os registros sem state = ‘GA’ e portanto tanto o campo company quanto state precisam ser avaliados para somente três registros.

**Índices secundários:**

### Sempre esteja certo que o seu índice está sendo usado:

“ A well-defined and properly implemented index is one of the best performance and tuning tool available. Because of the diversity of various database systems, however, and in particular various database optimizers, it is not possible to establish any hard-and-fast rules for creating and using database index. Additionally, it is impossible to guarantee that the database optimizers will use your index “.

Para saber se o índice definido está sendo usado:

\* use o SQL Trace (transação ST05) para ativar e desativar o trace

* clique em trace on
* execute a transação em questão em uma outra sessão
* retorne a ST05 e clique em trace off
* clique em list trace para visualizar os resultados de seu trace
* clique em prepare, open ou reopen para selecionar o comando SQL que será avaliado
* clique em EXPLAIN para obter o resultado do comando SQL

**secundários:**

**Regras gerais para criação e uso de índices**

Um índice suporta pesquisa a dados no banco de dados. Todas as tabelas standard SAP possuem um índice primário, o qual consiste de campos chaves que o usuário define quando da criação de uma tabela. Para os selects aonde a chave primária não pode ser utilizada na cláusula WHERE, ou quando selects não são qualificados, o banco de dados pesquisa a tabela inteira (executa uma varredura sequencial).

O Dicionário de Dados da SAP suporta até 16 índices para cada tabela. Geralmente, crie índices com menos que 5 campos.

Geralmente, se uma condição inclui OR, o otimizador pára o processamento (e chama uma varredura sequencial) tão logo o primeiro OR seja encontrado. A exceção possível é um OR que propõe uma condição separada e única para avaliação.

### Exemplo:

ZTABLE é definida com um índice secundário:

|  |  |  |
| --- | --- | --- |
| FIELD NAME | TYPE | LENGTH |
| FieldC | Char | 3 |
| FieldF | Char | 2 |

Select \* from ztable where

fieldc = ‘ABC’ and (fieldf = ‘12’ or ’13’)

🡺 isto é executado, mas não usa o índice conforme esperado

Select \* from ztable where

(fieldc = ‘ABC’ and fieldf = ‘12’) or (fieldc = ‘ABC’ and fieldf = ‘13’)

🡺 isto é executado usando o índice

### Informações gerais:

* Cláusulas IN são frequentemente interpretadas como condições OR e podem gerar alguns problemas
* Índices não são usados para condições IS (NOT) NULL
* A maior parte dos otimizadores tem problemas com condições OR, NEQ e LIKE

### Exemplo:

ZTABLE é definida com um índice secundário

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FIELD NAME | TYPE |  | LENGTH |  |
| FieldA | Char |  | 3 |  |
| FieldB | Char |  | 3 |  |
| FieldC | Char |  | 2 |  |
| FieldD |  | Char |  | 4 |

Select \* from ztable

Where fieldA = ‘ABC’ and field B = ‘XYZ’ and fieldC = ‘12’.

🡺 isto funciona bem.

Select \* from ztable

Where fieldA = ‘ABC’ and field B = ‘XYZ’ and fieldD = ‘DEFG’.

🡺 isto não usa o índice conforme esperado e provavelmente invoca uma varredura sequencial da tabela baseada na chave primária

### Considere a criação de um índice se um ou mais condições se aplicarem:

* campos não chave são repetidamente usados para fazer seleções
* somente uma pequena parte de uma grande tabela é selecionada (< 5%)
* a cláusula WHERE de um comando SELECT é simples
* os campos que compõe o índice reduzem significativamente o conjunto de registros em uma cláusula WHERE

### Não crie índices se uma das seguintes condições se aplicar:

* a redundância de dados de armazenamento do índice cria problemas devido ao tamanho da tabela ou índice
* atualizações constantes criam excessivo overhead ou perda de performance durante a atualização do índice
* manutenção de um índice (como a reorganização) causa perda dos benefícios
* estão sendo usados campos cujos valores para a maioria dos registros na tabela é um valor inicial ou tem um valor que é idêntico para a maioria dos registros

**Conceito de atualização em update task**

Para aproveitar ao máximo os recursos computacionais de cada nível, a SAP criou um conceito de atualização em update task. Se for feita uma pesquisa em todos os programas "on-line" da SAP, será verificado que a atualização é sempre em funções que só fazem o update, e que não retornam valores nem mesmo exceptions se estas ocorrerem. (CALL FUNCTION IN UPDATE TASK).

Logo todas as aplicações fazem todas as checagens possíveis e imagináveis antes de chamarem estas funções, ou seja, as exceptions que podem ocorrer são somentes as de BD (por exemplo table space, ou algo não previsto). Se uma aplicação faz "n" atualizações, esta irá chamar "n" funções (uma para cada update de um conjunto de tabelas) e, ao final, dará um commit work. O que o R/3 faz é simplesmente ir criando todos estes dados em uma espécie de log. No momento do commit work ele cria uma outra LUW (Logical Unit of Work) no servidor de BD e passa esta tarefa para este. Note que o programa (que esta sendo processado no servidor de aplicação), não fica esperando a resposta se o update ocorreu corretamente ou não, ele continua o processamento e termina a transação muitas vezes antes do servidor de DB ter terminado o processamento.

A vantagem óbvia deste processo é a diminuição de gargalos via este "tratamento de mensagens" entre as duas camadas (DB e aplicação).

Quando se utiliza o comando SET UPDATE TASK LOCAL o servidor de aplicação passa a executar o update (ou seja, não é criado uma outra LUW no servidor de DB) e este fica esperando a resposta se tudo ocorreu OK.

Um SET UPDATE TASK LOCAL é suficiente para todo o processamento do commit ser efetuado como local. Por isso, todos os updates da SAP procuram não utilizar o comando Set update task local. O default é o SET UPDATE (em funções). Isso evita que a performance seja degradada.

**Comandos Select**

Visando garantir a performance e evitar problemas futuros, cuidado com os comandos

empregados. Existem alguns tipos de select mais eficientes que outros. Analise a aplicação que será desenvolvida e faça a melhor escolha.

**ALGUNS TIPOS DE SELECT**

SELECT …FROM <table> INTO TABLE <INTERNAL TABLE> .

A estrutura da tabela interna deve corresponder à estrutura da tabela que está sendo acessada. O sistema lê os registros em conjunto, não individualmente, e os coloca dentro de uma internal table. Este processo é mais rápido que ler individualmente através de um LOOP e ir gravando os registros, um a um.

SELECT \* FROM <table> APPENDING TABLE <internal table>.

Lê os registros e os inclui - não sobrepõe - em uma internal table.

SELECT …. INTO CORRESPONDING FIELDS OF TABLE <itab>.

Neste caso a estrutura da tabela interna não precisa corresponder à estrutura da tabela que está sendo acessada. <itab> é o nome da internal table. Movimentará os registros para as colunas definidas na internal table que possuam nome igual ao da tabela acessada).

Obs.: corresponding ou appending corresponding não exigem o endselect.

SELECT ….. APPENDING CORRESPONDING FIELDS OF TABLE <itab>.

Lê e grava (não sobrepõe) os dados em uma internal table que possua nomes idênticos aos nomes da tabela que está sendo lida.

SELECT SINGLE \* FROM SPFLI WHERE …..<campo>….. EQ … <conteúdo>

Toda vez que se usa select single \* a chave primária completa deve ser especificada. Se a chave especificada não é qualificada, você receberá uma mensagem de warning e a performance ficará prejudicada.

No caso de haver a necessidade de acessar um único registro via select, as opções são: select \* ….. seguido de comando exit OU select \* … up to 1 rows. Neste caso não é necessário especificar a chave completa.

SELECT \* FROM …<tabela>

Quando não se impõe nenhum tipo de restrição, ocorre uma varredura sequencial dos registros da tabela. Quando se utiliza grandes tabelas, isso obviamente afeta o runtime.

Select \* seleciona todas as colunas de uma tabela. É melhor sempre especificar as colunas, pois em caso de tabelas com muitas colunas, prejudicará performance.

SELECT \* FROM <tabela> WHERE <campo> eq <conteúdo>.

Lê todos os registros da tabela especificada onde o campo é igual ao conteúdo especificado. O ideal é que se qualifique a cláusula WHERE tanto mais quanto seja possível. Atentar que isso é válido para tabelas do tipo Pool e Transparent. Para Cluster, seguir as recomendações dadas anteriormente.

SELECT <a1> <a2> … INTO (<f1>, <f2>, … ) FROM ….<tabela> WHERE …… .

Lê as colunas especificadas (a1, a2). Após INTO deverão ser especificadas as áreas de trabalho auxiliares (f1, f2). O número de colunas lidas deverá ser igual ao número de work-areas especificadas.

SELECT \* FROM <table> WHERE <table field> BETWEEN <field1> and <field2>.

Ex.: field1 = 100 e field2 = 500. Pega inclusive 100 e 500. Você trabalha com o range.

SELECT \* FROM <table> WHERE <table field> LIKE ….’\_R%’.

\_ = a primeira letra não importa o que virá a segunda deverá ser R (eu defini)

% = não importa a sequência de caracteres que virá. (Varredura para que possa realizar a comparação)

SELECT MAX(campo)

MIN(campo) AVG(campo)

COUNT(\*) FROM <table> INTO (…..,……,…..,….)

WHERE ………… .

AVG e SUM: somente para campos numéricos. Não se usa endselect.

Mais rápido fazer uma rotina “à mão” que utilizar este comando.

SELECT \* FROM <table> WHERE <table field> IN (…….,…….).

Exemplo: select \* from <table> where campo1 in (123,1000) - podem ser valores ou literais. É igual a perguntar se campo1 é 123 ou 1000.

SELECT \* FROM <table> WHERE <table field> IN <internal table>. Exemplo:

DATA : begin of ITAB occurs 10,

sign(1), option(2), low like sflight-price, high like sflight-price, end of ITAB.

\* RANGES: ITAB for sflight-table

Move: ’I’ to itab-sign, ‘bt’to itab-option, ‘500’ to itab-low, ‘1000’ to itab-high. Append itab. Move: ’I’ to itab-sign, ‘bt’to itab-option, ‘440’ to itab-low.

Append itab.

SELECT \* FROM (<table>) INTO <work area>. Exemplo: data: begin of WA,

line(100), end of WA.

Select \* from (tabname) into WA

Write ….

Endselect.

Parameters: tabname(10) default ‘SPFLI’.

Obs.: especificando o nome da tabela dinamicamente no select statement sempre consome mais tempo de CPU que especificando estaticamente no programa.

SELECT \* FROM <table> FOR ALL ENTRIES IN <internal table> WHERE campo1 = <conteúdo> and

campo2 = <conteúdo>

Defino uma tabela interna. Alimento os campos desta tabela interna. (move e append). No meu select campo1 e campo2 serão os campos definidos e alimentados na tabela interna.

Esta é uma excelente solução quando se trabalha com grandes tabelas. O select for all entries simula a funcionalidade join.

SELECT \* FROM <table> ORDER BY <field1> <field2> … PRIMARY KEY.

Obs.: Classifica a tabela interna numa área auxiliar, sem afetar a tabela original. Evitar o uso de sorts dentro de um select. Consome mais tempo que descarregar os dados em uma tabela interna e classificá-los.

SELECT carrid MIN( price ) max( price ) INTO (carrid, minimum, maximum) FROM sflight GROUP BY carrid.

(Todos os campos que eu quero que apareçam na minha lista eu preciso especificar após a cláusula GROUP BY. Carrid, maximum e minimum são campos auxiliares. Se o nome do database não é conhecido até runtime não se pode especificar a cláusula GROUP BY).

Performance não é tão boa (funções MIN, MAX, etc. geram uma varredura sequencial na tabela).

SELECT \* FROM <table> BYPASSING BUFFER.

(Usado para ler diretamente da tabela original, e não do buffer).

OBS.: Select single \* sempre com chave completa especificada. Particularidade do Abap/4.

Select \* - procurar evitar. Informar as colunas que serão necessárias, apenas.

**Dicas para otimização do código**

* Use o comando FREE para liberar espaço em internal tables; Sempre usar os comandos Clear / Refresh após o fim de um LOOP;
* Evite comparações num SELECT com campos numéricos versus campos alfanuméricos; o sistema perde tempo para conversão;
* Testar SY-SUBRC após cada acesso ao banco de dados;
* O comando MOVE-CORRESPONDING é bom para tabelas pequenas. É interessante que a tabela interna contenha os campos na sequência em que serão movimentados;
* Ao utilizar o comando CASE, codificar sempre a cláusula WHEN OTHERS;
* Sempre identifique se um SORT é ascending ou descending e especifique a cláusula BY <fields>. Caso contrário, todos os campos serão classificados.
* Evitar lógicas do tipo IF not CPOA = CPOB. É mais claro codificar IF CPOA ne CPOB.
* Evitar construções do tipo:

-

SORT tabela1, SORT tabela2, SORT tabela3. LOOP tabela1, LOOP tabela2, LOOP tabela3

Para cada SORT fazer o LOOP correspondente. Aí então iniciar novo SORT e LOOP, e assim por diante.

* Campos chave devem ser sempre os primeiros campos da tabela. Assim, todos os demais campos serão comprimidos;

-

* SELECT (para Transparent e Pool Tables): a cláusula WHERE deve conter, preferencialmente, os campos chaves e demais campos que possam restringir a pesquisa;

-

* SELECT (para Cluster Tables): só os campos chaves devem ser especificados na cláusula WHERE. Os demais devem ser checados através do comando CHECK;
* O conhecimento do conteúdo dos dados de uma tabela pode auxiliar no momento da codificação do comando select. O campo que ocorrer em número menor de vezes deverá constar na cláusula where antes daquele que ocorre um número maior de vezes, caso seja necessário satisfazer a ambas condições. Isso faz com que o processamento seja mais ágil.
* Manuseio de tabelas: Estudar a possibilidade de manuseio em tabelas internas para agilizar o processo. Analisar também o uso de comando select, sendo o que melhor se adapte a situação em questão.

**Ferramentas para auxliar os desenvolvedores**

Existem algumas ferramentas que auxiliam os desenvolvedores a descobrir erros (errors, warnings, etc), bem como avaliar a performance de seus programas.

Path: Tools – Abap/4 Workbench – Test - Runtime Analysis

Existe um pushbutton chamado Tips and Tricks – Clicando-se nele poderá ser feita comparação entre comandos Select (diferentes tipos). Dando double-click sobre um dos exemplos você passará para outra tela, onde você visualizará a medida do tempo em microsegundos de ambos, servindo de base de comparação.

Neste mesmo path, pode-se utilizar as facilidades do Runtime Analysis para verificar a performance de seu programa. Ao final da execução você poderá acessar informações sobre o seu programa (gráficos, acesso a tabelas, etc). Para tanto clique o pushbutton ANALYSE, que somente aparecerá depois que for informado o nome do programa e/ou transação e clicado execute.

O SQL Trace (transação ST05) é outra facilidade que pode ser utilizada para trilhar a lógica de sua aplicação e verificar possíveis pontos de correção. Você poderá visualizar os comandos de acesso a banco de dados utilizados, obter informações sobre um comando específico, visualizar os índices que estão sendo utilizados (ver tópico 5.3 sobre como utilizar o SQL Trace).

O Extended Program Check é outra facilidade que deve ser empregada visando manter o seu código o mais correto possível. Nesta opção você poderá selecionar os ítens que você deseja que sejam checados e o sistema apontará o seu parecer. Para tanto basta clicar os ítens e o pushbutton PERFORM CHECK Selecionando uma linha você poderá ver os detalhes, bem como posicionando o cursor em uma linha e clicando o pushbutton DISPLAY ALL CHANGES.

**Interfaces Batch ou Conversões:**

Segundo a documentação da SAP, existem três métodos para conversão: Direct input, Call transaction e o Batch Input.

* Direct Input: é um dos métodos para transferência de dados do sistema legado para o sistema R/3. É considerado o método mais rápido. Um arquivo sequencial com dados é gerado como um arquivo texto para processamento por alguns function modules especiais. Estas funções executam todas as checagens normais para garantir a integridade de dados. Quando os registros são processados com sucesso, eles são gravados diretamente nas correspondentes tabelas do banco de dados da aplicação. Na ocorrência de erros, os dados errados são passados para uma rotina de manuseio de exceção. O gargalo associado com o processamento do dialog e update são eliminados neste caso. Para todas as necessidades de transferência de dados, especialmente com transações de alto volume, o direct input é o método mais indicado. Nota: este método deveria ser usado em todas as situações nas quais funções de aplicação utilizando esta tecnologia existam.
* Call Transaction: é o próximo método mais rápido para processamento de dados do sistema legado para o sistema R/3. Neste caso, o programa de transferência de dados processa os dados do arquivo sequencial e chama a transação desejada usando um comando Abap. Dados de um arquivo sequencial são processados via telas de aplicação para uma única transação. A lógica de aplicação executa todos os checks e a validação dos dados. Este é um processamento síncrono. Quando o processamento do dialog é realizado com sucesso, o processamento do update é chamado pela aplicação correspondente para executar todas as solicitações do banco de dados. A escolha pode ser feita entre update síncrono ou assíncrono. Nenhum protocolo de erros ou saída para lidar com os erros é fornecida por esta técnica. É responsabilidade do programador da aplicação incorporar rotinas de execução e funções dos protocolos desejados como parte do programa de transferência de dados. A nossa recomendação é usar o método Call Transaction em todos os casos em que não existe um programa Direct Input. Somente em situações de lidar com erros nós recomendamos que o batch-input tradicional seja realizado para posterior processamento. Resumo: em contraste com o Batch- Input, o Call Transaction permite que sejam passados dados diretamente ao dialog interface sem usar uma fila. Para armazenar estes dados temporariamente, você usa uma internal table (uma tabela BDC, a qual tem a mesma estrutura daquela utilizada no batch-input). Antes de entrar dados na BDC table é necessário fazer uma verificação nos dados. Call transaction: pode ser executado imediatamente.
* Batch-Input: o batch-input tem sido tradicionalmente selecionado como um método de implementação de programas de transferência de dados. Um benefício que este método traz sobre o Call Transaction é que o batch-input tem um utilitário responsável pela administração e gerenciamento das funções do batch-input. Além do mais, utilitários existem para monitorar e dar manutenção em tarefas associadas com este método. Assim, não há necessidade de nenhuma programação adicional para análise de exceções e funções de protocolo. Dados vindos de um arquivo sequencial são processados via telas de aplicação e armazenados numa sessão batch-input. Por definição, uma sessão é uma coleção de dados de transação para uma ou mais transações. As sessões batch-input são fisicamente armazenadas pelo sistema num banco de dados como uma fila. Estas sessões podem conter tanto registros de dados corretos quanto incorretos. O batch-input trata tanto o update síncrono como o assíncrono. O método batch-input, em contraste com o método Call Transaction, pode transferir dados do sistema legado para o sistema R/3 para múltiplas transações da aplicação. Todavia, nenhuma nova transação é iniciada até que a transação anterior tenha sido gravada no correspondente banco de dados durante o processamento das sessões de batch-input. Também, sessões de batch- input não podem ser geradas em paralelo. Ele pode ser processado de três modos: foreground, display erros only ou background.

Este método oferece excelente capacidade para gerenciamento de erros e é detalhado para análise do protocolo da transação.

Resumo: é gerado um log e pode ser programada a execução. Em caso de erro, o registro pode ser corrigido e inserido novamente. Através deste recurso, todos os dados são entrados na transação original SAP, ou seja, seguindo todas as consistências necessárias, evitando assim que erros de integridade sejam cometidos.