**1: Criando Tabelas e Índices > Objetivos**

Após a conclusão deste módulo, você deverá ser capaz de:

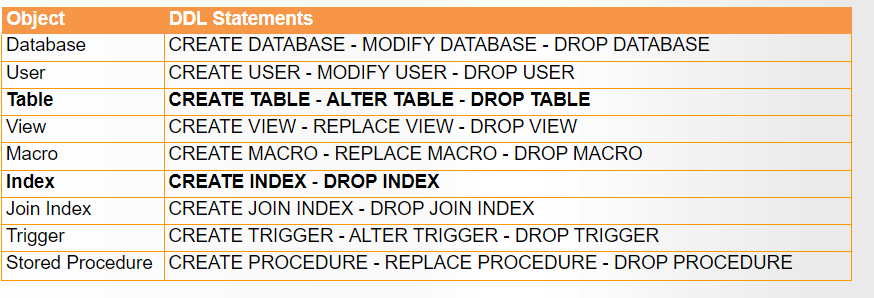
CREATE, DROP ou ALTER uma tabela.

CREATE ou DROP um índice.

Atributos da tabela ALTER.

A linguagem de definição de dados (DDL) é usada pelo SQL para criar, modificar e remover definições de objetos:

O banco de dados Teradata implementa instruções DDL separadas para objetos diferentes. Abaixo está um gráfico de alguns usados ​​com frequência. Neste módulo veremos apenas a criação de tabelas e índices.



Quando executada, a instrução CREATE TABLE cria e armazena a definição da estrutura da tabela no Data Dictionary (DD).

A instrução CREATE TABLE permite criar:

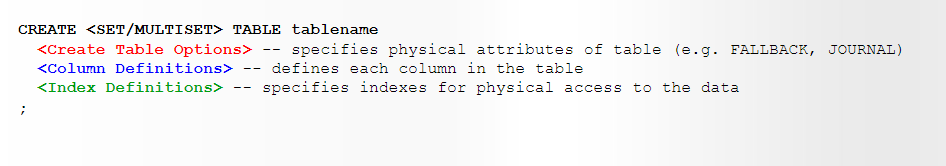
Opções da TABELA.

COLUNA definições.

Restrições de nível TABLE.

Definições de ÍNDICE.

Nossa discussão aqui se concentrará nas opções de tabela, definições de coluna e definições de índice. As restrições de nível de tabela são abordadas em outro curso. Consulte Referências na página de Ajuda para obter mais informações.



A instrução CREATE TABLE permite que você especifique se deseja ou não permitir linhas duplicadas em sua tabela.

Uma tabela SET não permite linhas duplicadas. Este é o padrão no modo Teradata.

Exemplo

CREATE SET TABLE tabela1

(c1 INTEIRO )

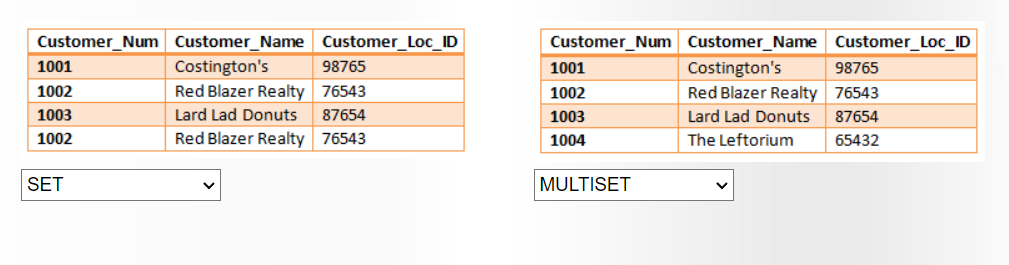
ÍNDICE PRIMÁRIO ÚNICO (c1);

Uma tabela MULTISET permite linhas duplicadas. Este é o padrão no modo ANSI.

Exemplo:



Tabelas:



A instrução CREATE TABLE permite especificar opções de implementação física, como fallback e proteção de diário permanente, tamanho do bloco de dados e fator de espaço livre do cilindro. Você pode definir um índice primário (diferente da chave primária), bem como índices secundários para melhorar o acesso físico aos dados na tabela.

Opções de proteção de tabela (sublinhados indicam padrões)

FALLBACK ou SEM FALLBACK

BEFORE JOURNAL – (NO, SINGLE OR DUAL) (NÃO, SIMPLES ou DUPLO)

AFTER JOURNAL (NO, SINGLE (LOCAL or NOT LOCAL) or DUAL) -APÓS DIÁRIO (NÃO, ÚNICO (LOCAL ou NÃO LOCAL) ou DUPLO)

WITH JOURNAL TABLE [tablename] - COM TABELA DE DIÁRIO [nome da tabela]

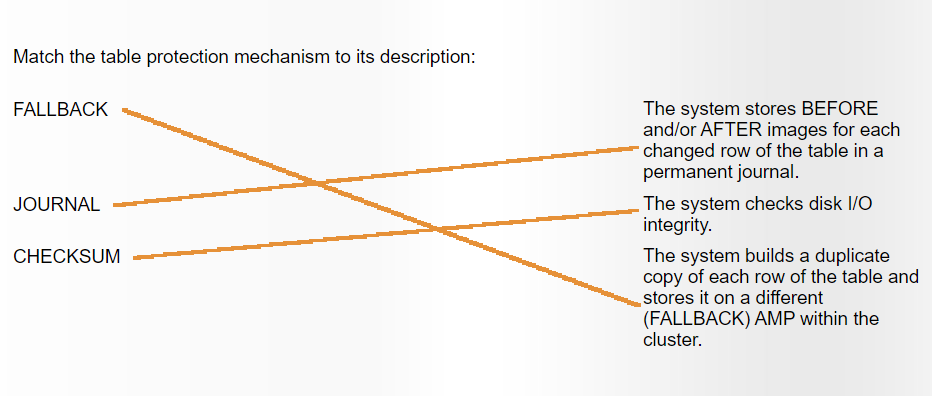
CHECKSUM - SOMA DE VERIFICAÇÃO

FALLBACK especifica que o sistema cria uma cópia duplicada de cada linha da tabela e a armazena em um AMP diferente (FALLBACK) dentro do cluster.

JOURNAL especifica que o sistema armazena imagens ANTES e/ou DEPOIS de cada linha alterada da tabela no diário permanente especificado, fornecendo capacidade de recuperação de desastres.

A opção CHECKSUM implementa a verificação de integridade de E/S de disco de linhas de dados da tabela primária, linhas de dados da tabela de fallback e linhas de subtabela de índice secundário em vários níveis especificados pelo usuário ou em todo o sistema.

Tente agora



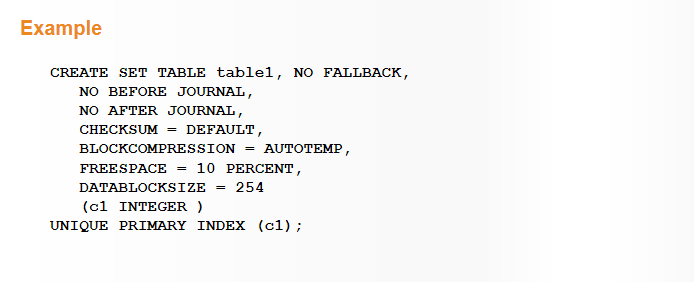
Opções de gerenciamento de espaço (os padrões são fornecidos se não especificados)

FREESPACE = n [PERCENT] - especifica a porcentagem de espaço para permanecer livre durante uma operação de carregamento

DATABLOCKSIZE - especifica o tamanho máximo do bloco para blocos de dados de várias linhas. O bloco de dados é a unidade de E/S no banco de dados Teradata.

MERGEBLOCKRATIO = n [PERCENT]- fornece uma maneira de combinar pequenos blocos de dados existentes em um único bloco de dados maior. n especifica o tamanho do único bloco de dados maior, como uma porcentagem do tamanho máximo do bloco da tabela. O padrão do sistema é 60.

BLOCKCOMPRESSION - define o estado de compressão do bloco com base na temperatura.



Até 2.048 colunas podem ser definidas em uma única instrução CREATE TABLE.

Você pode definir os seguintes aspectos de cada coluna:

Nome da coluna

Tipo de dados (INTEGER, CHAR, etc.)

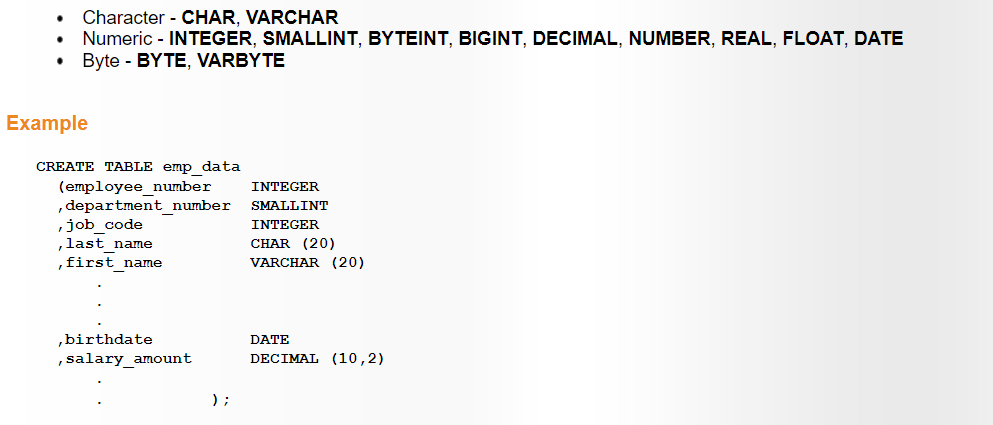
Atributos de tipo de dados (atributos de coluna como DEFAULT, FORMAT, TITLE, NULL, CASESPECIFIC, UPPERCASE, etc)

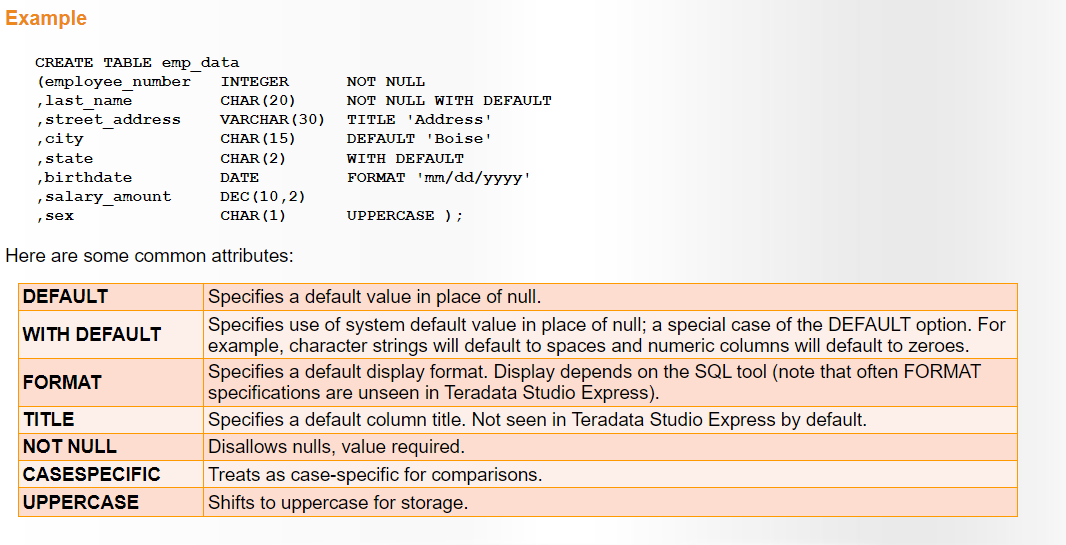
Atributos de armazenamento de coluna (compressão para nulos ou para valores específicos)

Atributos de restrição em nível de coluna (intervalo de valores possíveis, exclusividade, uso como chave primária ou estrangeira

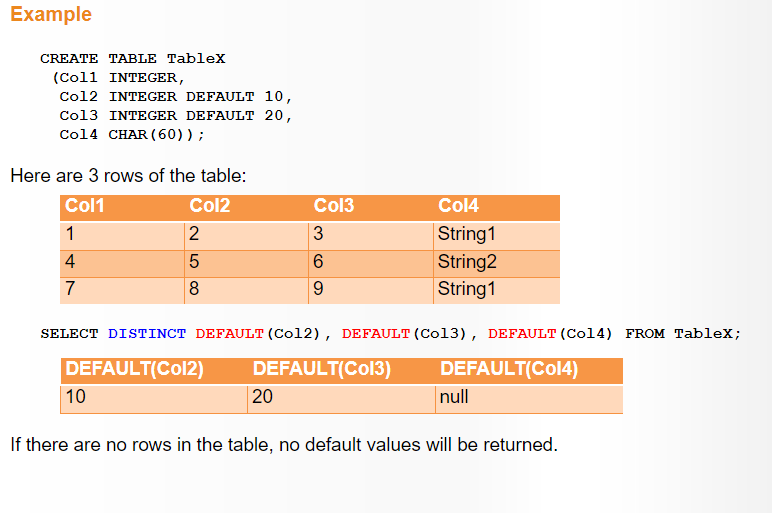
Discutiremos cada um deles em detalhes nas páginas a seguir.

Você deve especificar um nome e tipo de dados para cada coluna definida para uma tabela. Aqui estão alguns tipos de dados comumente usados:



Alguns (column attributes)atributos de coluna são específicos do tipo de dados, como (UPPERCASE)MAIÚSCULAS (que é apenas para dados de caracteres). Outros atributos podem ser para qualquer coluna, como NOT NULL.

Se você deseja verificar qual é o valor padrão de uma coluna, pode usar SHOW TABLE ou pode usar a função DEFAULT. A função DEFAULT recupera o valor padrão atribuído a uma coluna. Se nenhum valor padrão for atribuído à coluna especificada, um valor nulo será retornado.



A frase COMPRESS permite que valores em uma ou mais colunas de uma tabela sejam compactados, reduzindo assim o espaço de armazenamento físico necessário para uma tabela.

O banco de dados Teradata oferece 3 opções de compactação que podem ser usadas separadamente ou em conjunto:

1. Compactação de vários valores (ou compactação de lista de valores)

Um recurso que permite que até 255 valores distintos (mais NULL) sejam compactados por coluna. Isso reduz o custo de armazenamento e melhora o desempenho porque há menos dados físicos a serem recuperados durante consultas orientadas a varredura.

2. Compressão Algorítmica (ALC) ou Compressão em Nível de Coluna

Permite que os usuários apliquem um algoritmo de compactação aos dados no nível da coluna em uma linha. A compactação/descompactação é feita especificando uma função UDF.

3. Compressão em nível de bloco (BLC)

A compactação é executada pelo banco de dados Teradata no nível do sistema de arquivos em blocos de dados inteiros antes que os blocos de dados sejam realmente gravados/lidos nos dispositivos de armazenamento.

A compressão é abordada com mais detalhes em outro curso. Consulte Referências na página de Ajuda para obter mais informações.

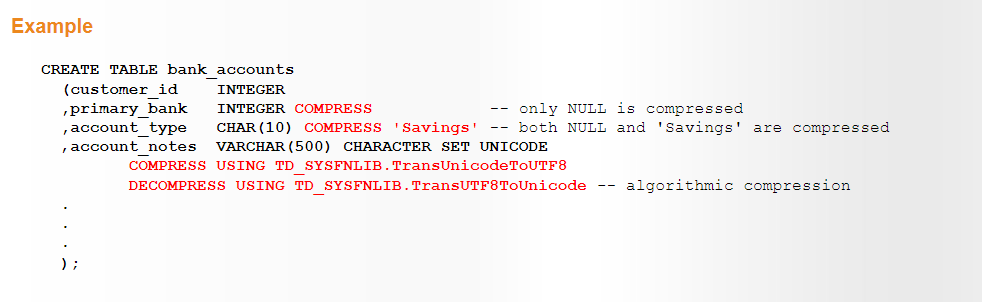
A compactação de vários valores tem três variações:

COMPRESS - Nulos são compactados.

COMPRESS NULL - Nulos são compactados. (COMPRESS & COMPRESS NULL significam a mesma coisa)

COMPRESS <constante> - Nulos e o valor <constante> especificado são compactados.

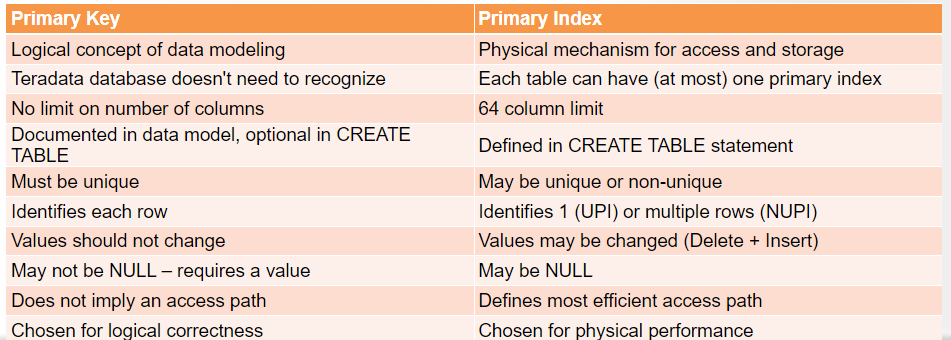
Com compactação algorítmica, você deve especificar um UDF para compactação e um UDF para descompactação.

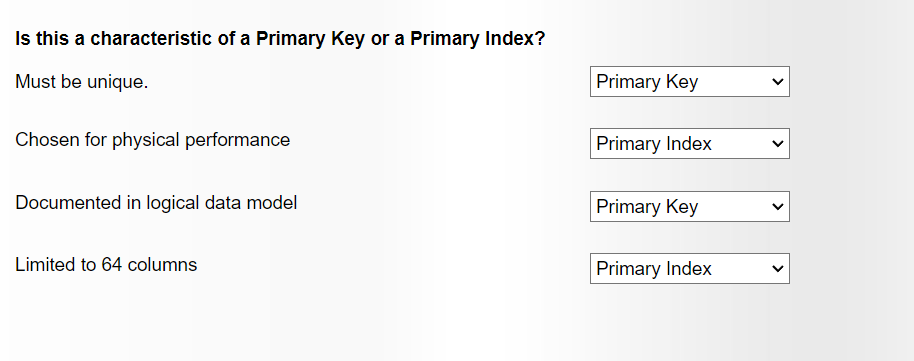


Os índices são conceitualmente diferentes das chaves. Uma chave primária é uma convenção de modelagem relacional que permite que cada linha seja identificada exclusivamente. Um índice primário é uma convenção de banco de dados Teradata que determina como a linha será armazenada e acessada.

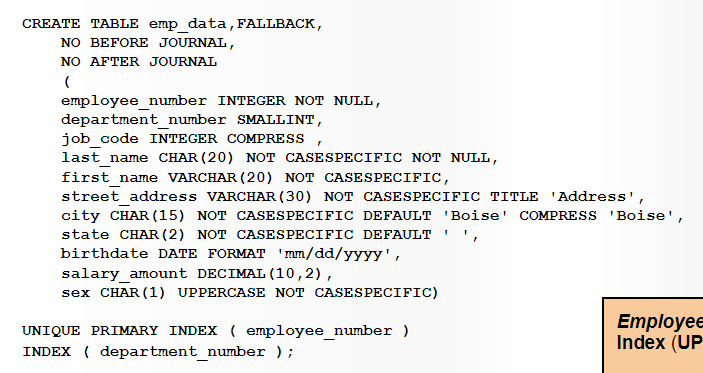
Chave primária (PK) - é definida como uma ou mais colunas usadas para identificar exclusivamente cada linha em uma tabela. PKs são usados ​​em conjunto com chaves estrangeiras para definir os relacionamentos de colunas importantes em um banco de dados. PKs são sempre únicos e não podem ser nulos. Se você definir um PK com CREATE TABLE, o Teradata RDBMS o implementará como um índice exclusivo.

Índice primário (PI) - é definido como uma ou mais colunas usadas para distribuir e localizar linhas em uma tabela. A escolha do índice primário afetará a distribuição, o acesso e o desempenho. Freqüentemente, mas nem sempre, o Índice Primário e a Chave Primária de uma tabela são os mesmos. Os índices (primários ou secundários) podem ser usados ​​para reforçar a exclusividade (como em um PK) ou para melhorar o acesso. Eles podem ser únicos ou não únicos.





Aqui está um exemplo demonstrando muitos dos conceitos que discutimos até este ponto sobre a instrução CREATE TABLE.



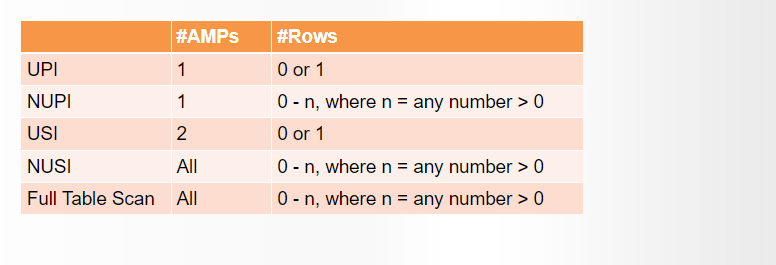
Employee\_number é um índice primário exclusivo (UPI).

Department\_number é um índice secundário não exclusivo (NUSI

Vimos que podemos ter um índice primário único (UPI) ou um índice primário não único (NUPI).

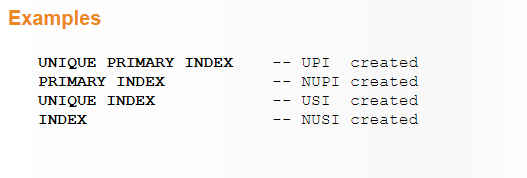
Da mesma forma, as tabelas podem ter um índice secundário exclusivo (USI) e um índice secundário não exclusivo (NUSI).

A tabela abaixo ilustra as características de desempenho para operações SQL que usam índices. As entradas são listadas em ordem do mais desejável ao menos desejável em termos de uso de recursos do sistema. Eles mostram o número de AMPs envolvidos em cada operação e o número de linhas que podem ser retornadas.

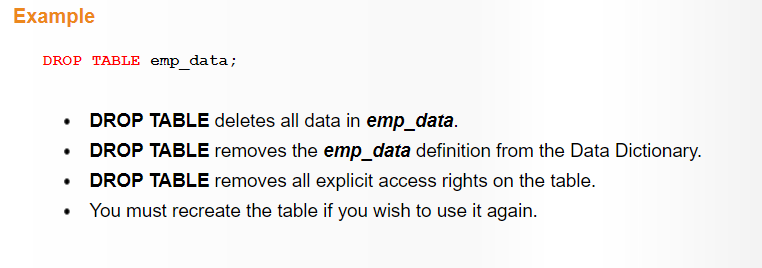


Qualquer usuário pode criar uma tabela em um determinado banco de dados, desde que tenha o privilégio CREATE TABLE para esse banco de dados.

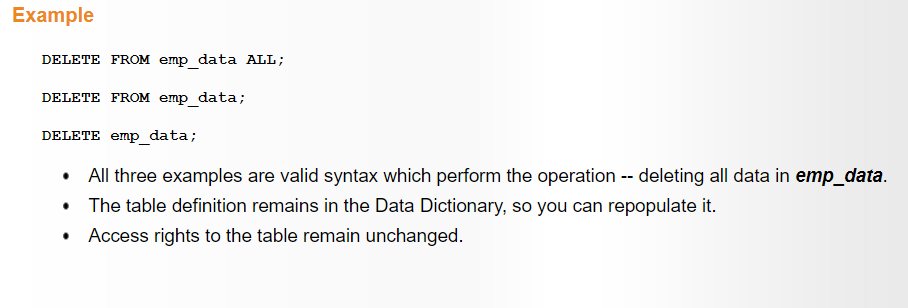
Para um índice secundário, se UNIQUE não for explícito, o padrão será um índice não exclusivo. Da mesma forma, se PRIMARY não for explícito, o padrão será um índice secundário.



Para remover todos os dados associados a uma tabela, bem como a definição da estrutura da tabela do Dicionário de Dados, use a instrução DROP TABLE.

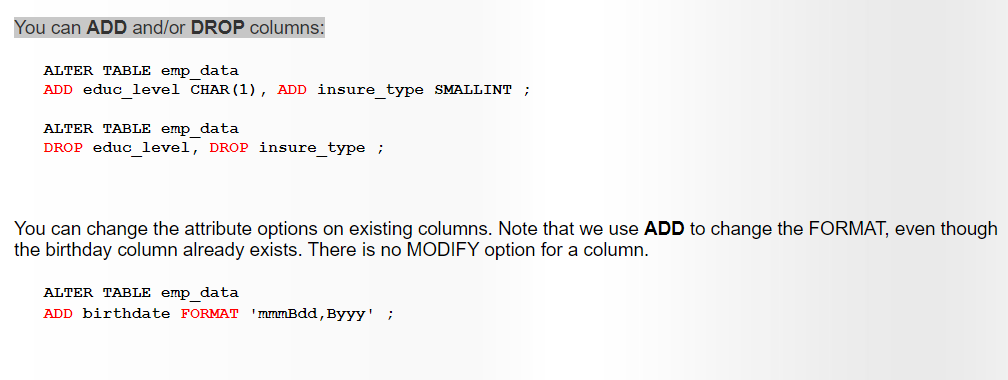


Para remover todos os dados associados a uma tabela, mas sem descartar a definição da tabela do Dicionário de Dados, use a instrução DELETE.

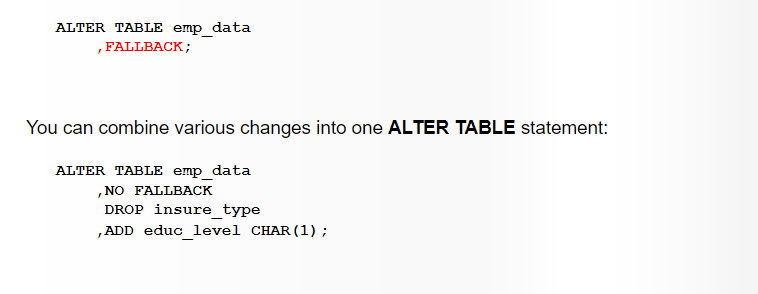


Depois que uma tabela é criada, certas características não podem ser alteradas, como a escolha do índice primário. Para alterá-las você deve DROP e recriar a tabela com as novas características.

Outras características podem ser alteradas. Você pode usar a instrução ALTER TABLE para modificar essas características.



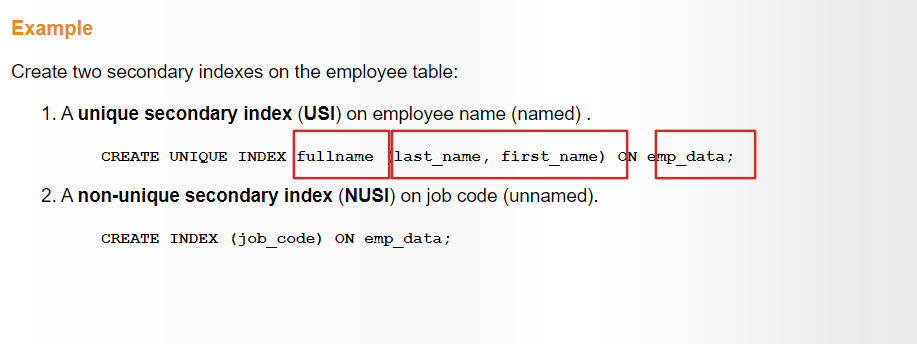
Você pode transformar uma tabela NO FALLBACK em uma tabela FALLBACK ou vice-versa. Mudar uma tabela de FALLBACK para NO FALLBACK pode ser usado para recuperar espaço no banco de dados quando necessário:



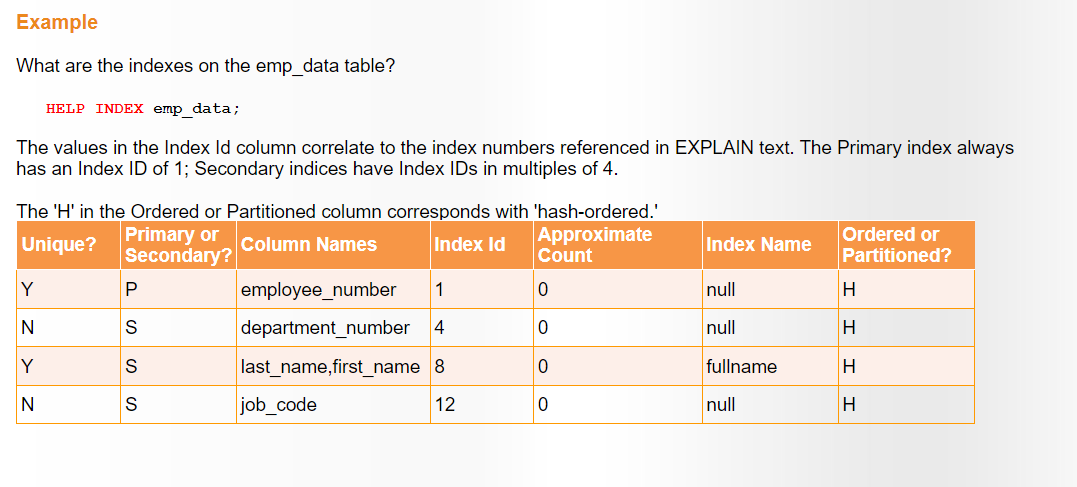
Os índices primários são sempre criados na criação da tabela com a instrução CREATE TABLE. Eles não podem ser alterados sem descartar e recriar a tabela (com um novo índice primário).

Os índices secundários podem ser criados na criação da tabela na instrução CREATE TABLE ou posteriormente usando a instrução CREATE INDEX. A instrução CREATE INDEX é usada para criar índices secundários em uma tabela existente.

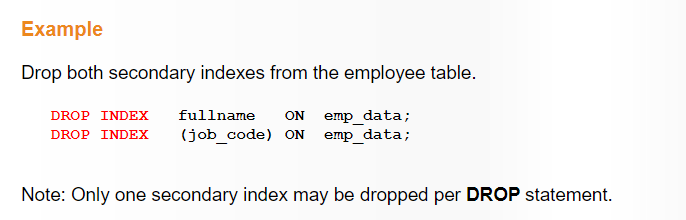
Os índices secundários podem ser criados com ou sem um nome de identificação. A nomenclatura tem algumas vantagens em termos de simplificação da sintaxe usada para identificar o índice nos comandos SQL subsequentes.



HELP INDEX <tablename> exibe definições de índice para a tabela especificada. Se o índice não tiver nome, o valor na coluna 'Index Name' será exibido como NULL. A nomeação de índices é opcional, no entanto, os índices nomeados fornecem mais facilidade de uso.



Se você estiver descartando um índice nomeado, poderá especificar o índice pelo nome do índice ou por sua definição de coluna.



Para este conjunto de exercícios de laboratório, você pode precisar de informações do documento Database Info. Antes de fazer esses laboratórios, será útil definir seu banco de dados padrão para seu banco de dados (ou seja, DATABASE NNXXXXXX;).

Laboratorio.

Clique no botão Avançar na parte inferior da página para ver as soluções.

1.) CRIE uma tabela de funcionários em seu próprio banco de dados idêntica à tabela de funcionários no banco de dados CustomerService. Nomeie-o emp\_new. Use o seguinte comando para ajudá-lo a criar a nova tabela em seu próprio banco de dados.Observação: esta tabela será usada em muitos exercícios de laboratório subseqüentes.

**database** pollyana.ferreira@capgemini.com**;**

**show** **table** employee**;**

**CREATE** **SET** **TABLE** "pollyana.ferreira@capgemini.com".emp\_new ,**FALLBACK** ,

**NO** **BEFORE** **JOURNAL**,

**NO** **AFTER** **JOURNAL**,

**CHECKSUM** = **DEFAULT**,

**DEFAULT** **MERGEBLOCKRATIO**

(

employee\_number INTEGER,

manager\_employee\_number INTEGER,

department\_number INTEGER,

job\_code INTEGER,

last\_name CHAR(20) CHARACTER **SET** LATIN **NOT** CASESPECIFIC **NOT** **NULL**,

first\_name VARCHAR(30) CHARACTER **SET** LATIN **NOT** CASESPECIFIC **NOT** **NULL**,

hire\_date DATE **FORMAT** 'YY/MM/DD' **NOT** **NULL**,

birthdate DATE **FORMAT** 'YY/MM/DD' **NOT** **NULL**,

salary\_amount DECIMAL(10,2) **NOT** **NULL**)

**UNIQUE** **PRIMARY** **INDEX** ( employee\_number )**;**

Renomear a tabela

**Rename** **table** "pollyana.ferreira@capgemini.com".employee **to** "pollyana.ferreira@capgemini.com".emp\_new**;**

2.) Modifique a tabela emp\_new criada no Laboratório nº 1 para:

Remova a proteção contra fallback.

Adicione uma coluna numérica chamada nível\_salário como um BYTEINT.

**ALTER** **TABLE** emp\_new

**ADD** salary\_level BYTEINT**;**

Adicione um índice secundário exclusivo na coluna salario\_nível.

**CREATE** **UNIQUE** **INDEX** (salary\_level) **ON** emp\_new**;**

3.) Faça um HELP INDEX para exibir os índices de sua tabela.

**HELP** **INDEX** emp\_new**;**

4.) ALTERE a tabela para inverter o Lab #2. (ou seja, DROP o índice; DROP a coluna; Retornar a tabela para FALLBACK)

**DROP** **INDEX** (salary\_level) **ON** emp\_new**;**

**ALTER** **TABLE** emp\_new ,**FALLBACK**

**DROP** salary\_level**;**

**2 – Macros Simples > Objetos**

Após a conclusão deste módulo, você deverá ser capaz de:

CRIE e EXECUTE uma macro simples.

SUBSTITUIR uma macro.

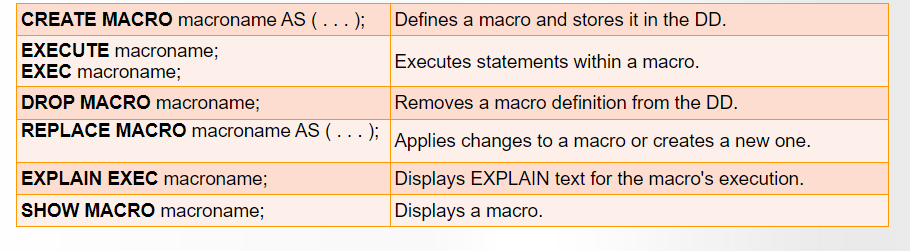
DROP uma macro.

Uma macro é uma extensão do Teradata para ANSI SQL que contém instruções SQL pré-escritas.

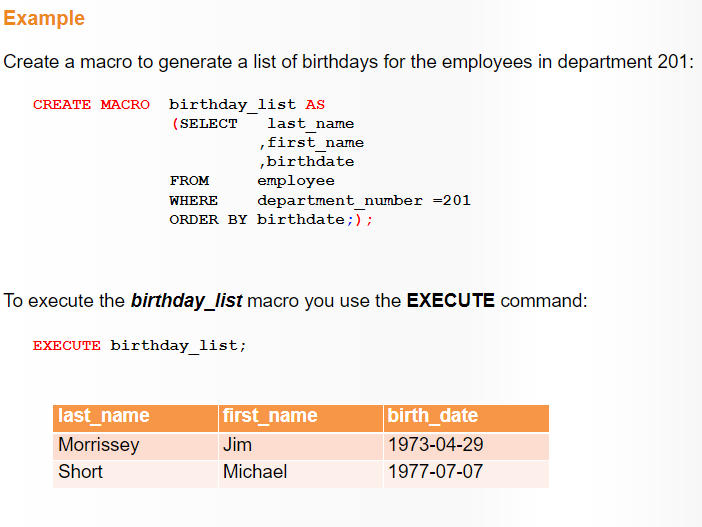
O texto real da macro é armazenado em um repositório global chamado Data Dictionary (DD). As macros são objetos de banco de dados e, portanto, pertencem a um usuário ou banco de dados especificado. Eles podem conter uma ou mais instruções SQL. As macros têm eficiências especiais no ambiente de banco de dados Teradata e são altamente desejáveis ​​para criar consultas reutilizáveis.

Uma macro permite nomear um conjunto de uma ou mais instruções. Quando você precisar executar essas instruções, simplesmente execute a macro nomeada. As macros fornecem um atalho conveniente para a execução de grupos de instruções SQL executadas com frequência.

A tabela abaixo é uma lista bastante completa de comandos que você pode usar para manipular macros.



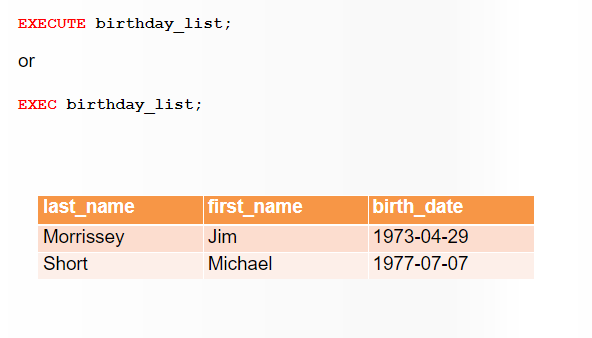
O comando CREATE MACRO permite definir um conjunto de instruções SQL e colocá-las em uma instrução executável.

Coisas a serem observadas

Há um ponto e vírgula (;) antes do parêntese de fechamento. Isso é necessário.

A instrução CREATE MACRO falhará se houver um erro de sintaxe em qualquer parte do corpo da instrução SQL. Se existir uma macro, é garantido que ela foi previamente verificada pela sintaxe. No entanto, isso não garante que funcionará sem erros porque os objetos referenciados na macro (por exemplo, a tabela de funcionários) podem não existir mais ou podem ter sido alterados.

Execute a macro com um simples comando chamado EXECUTE ou EXEC



Use o DRO MACRO pra deletar a macro



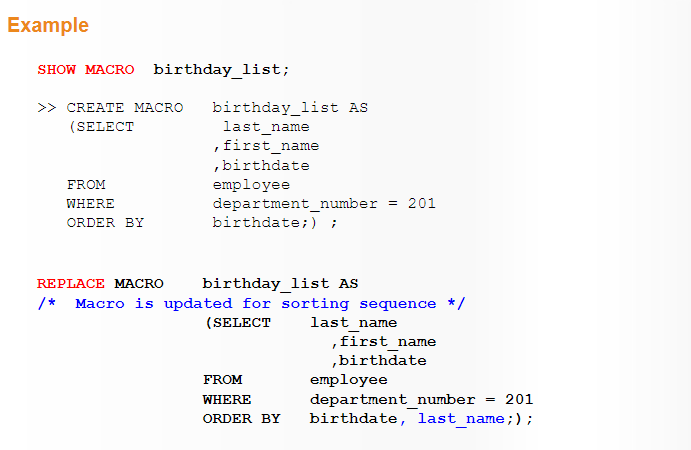
Para modificar uma macro, você precisa reenviar todo o conteúdo com um comando REPLACE MACRO. A maneira mais fácil de fazer isso é primeiro exibir a definição de macro atual e depois copiá-la/colá-la em um editor de texto. Você pode então modificar a macro sem redigitar o comando inteiro. Para exibir a definição de uma macro, use o comando SHOW.

Step 1: Exiba a definição de macro atual com SHOW MACRO.

Step 2: Copie/cole a definição de macro atual em um editor de texto.

Step 3: Faça as modificações necessárias, incluindo a alteração de 'CREATE' para 'REPLACE'. Você também pode adicionar comentários explicando a mudança (opcional).

Step 4: Copie/cole na sua ferramenta SQL e envie o SQL.



Ao enviar este SQL, a antiga macro lista\_aniversários será substituída pela nova.

Laboratorio:

1.) Escreva uma macro simples chamada LocationMacro1 para criar uma lista de distribuição geográfica do cliente a partir da tabela de localização no banco de dados CustomerService. Inclua o número do local, o número do cliente, a cidade e o estado (limite a cidade e o estado a 14 caracteres cada). Classifique por cidade dentro do estado. CRIE a macro. EXECUTAR a macro.Observação: CustomerService é um banco de dados somente leitura. Você precisa criar as macros em seu próprio espaço de usuário, mas fazer referência à tabela no banco de dados CustomerService. **execute** LocationMacro1**;**

**Create** **Macro** LocationMacro1 **as**

(

**Select** location\_number

,customer\_number

,Cast (city **as** char(14))

,Cast (state **as** char(14))

**FROM** CustomerService.location

**Order** **By** city, state**;** )**;**

2.) Modifique a macro do Laboratório nº 1 para remover o número do local. Adicione o código postal como a última coluna do relatório. O código postal deve ser o nível mais baixo de classificação. Limite sua seleção a clientes baseados no estado de Illinois. Salve a macro modificada como LocationMacro2. Envie e execute a nova macro.

**Create** **Macro** LocationMacro2 **as**

(

**Select** customer\_number

,Cast (city **as** char(14))

,Cast (state **as** char(14))

,zip\_code

**FROM** CustomerService.location

**Where** state = 'Illinois'

**Order** **By** state, city, zip\_code**;**

)**;**

**exec** LocationMacro2**;**

3.) Use o comando HELP apropriado para localizar os nomes de todas as macros em seu banco de dados.

**Help** **database** "pollyana.ferreira@capgemini.com"**;**

4.) SUBSTITUA a macro que você criou para o Laboratório nº 1, com estas alterações: Selecione apenas os clientes na cidade de Chicago. Inclua a primeira linha de endereço como a última coluna do relatório. Remova o estado do relatório. Classifique a saída por número de cliente em ordem decrescente.

**REPLACE** **MACRO** LocationMacro1 **AS**

(

**Select** location\_number

,customer\_number

,Cast (city **as** char(14))

,first\_address\_line

**FROM** CustomerService.location

**Where** state = 'Chicago'

**Order** **By** customer\_number **DESC;**

)**;**

5.) Solte ambas as macros.

**DROP** **MACRO** LocationMacro1**;**

**DROP** **MACRO** LocationMacro2**;**

**3 Macros Parametrizadas:**

Após a conclusão deste módulo, você deverá ser capaz de:

Crie e execute uma macro parametrizada.

As macros contêm uma ou mais instruções SQL pré-escritas que são armazenadas no dicionário de dados do banco de dados Teradata.

Outros pontos importantes a serem observados sobre macros:

As macros são uma extensão Teradata para SQL.

As macros só podem ser executadas com o privilégio EXECUTE.

Se um usuário tiver privilégios EXECUTE, ele/ela pode executar a macro. O usuário não precisa de privilégios para as tabelas ou exibições subjacentes usadas pela macro.

As macros podem fornecer segurança em nível de coluna.

Eles podem ser executados a partir de qualquer front-end SQL viável, incluindo:

\* Teradata Studio or Studio Express

\*BTEQ

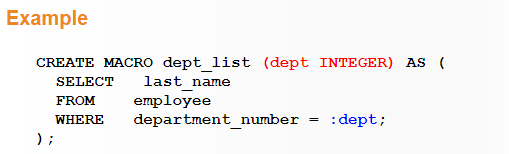
\*Preprocessor

\*CLI routines

\*LOGON Startup

\*Another macro

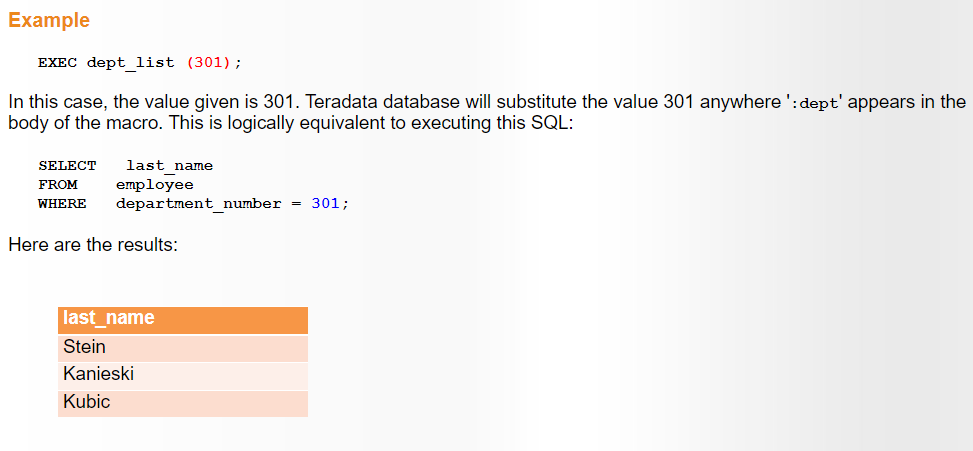
Macros parametrizadas permitem variáveis ​​substituíveis na macro. Os valores dessas variáveis ​​são fornecidos no tempo de execução.

**

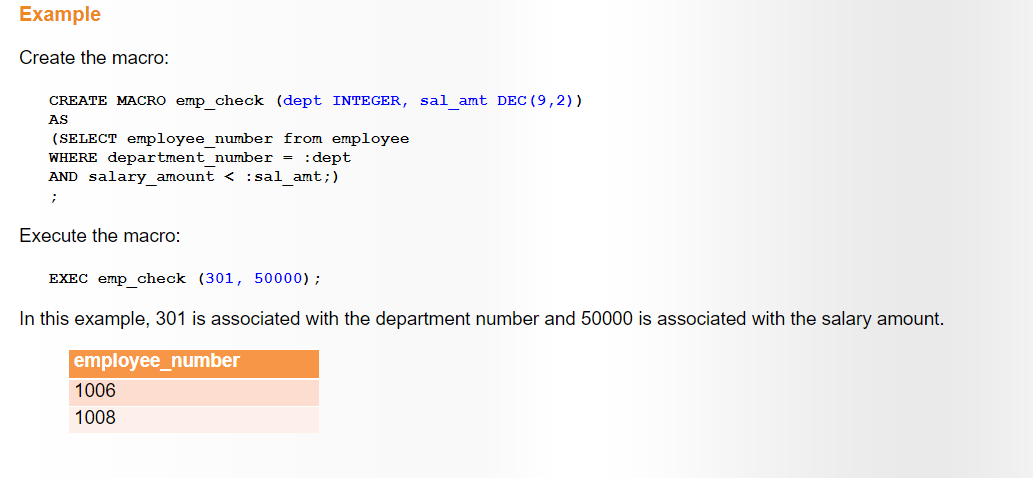
A lista de parâmetros está entre parênteses após o nome da macro. Cada nome de parâmetro é seguido por seu tipo de dados. Nesse caso, temos um parâmetro chamado dept e é um número inteiro. O valor fornecido para dept no tempo de execução deve ser um número inteiro.

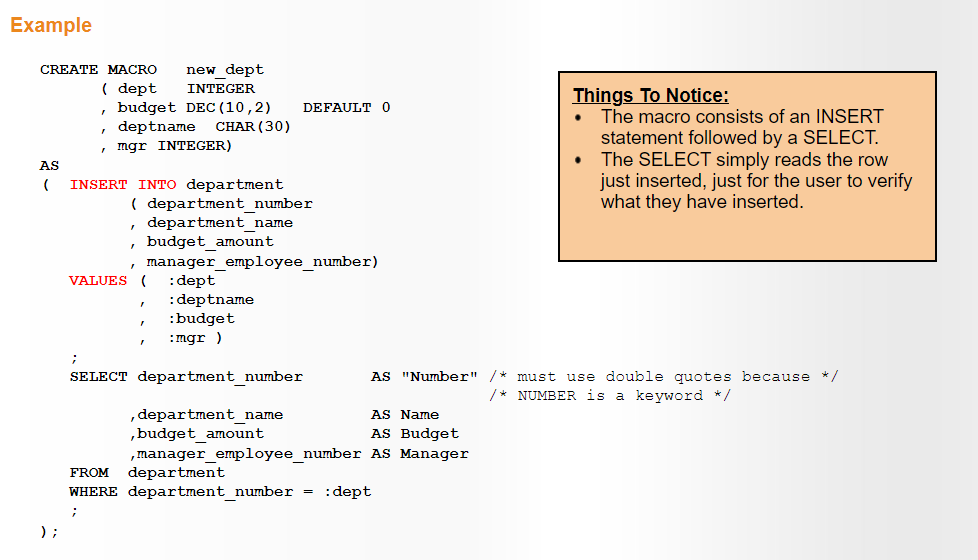
Se você tiver um parâmetro estabelecido, poderá usá-lo no SQL no corpo da macro. Normalmente, isso filtrará os dados de alguma forma. Quando um parâmetro é utilizado no corpo de uma macro, ele sempre é precedido por dois pontos (:).

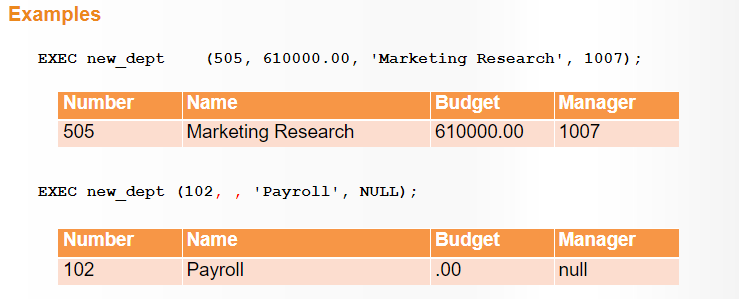
Ao executar uma macro parametrizada, você precisa fornecer um valor para o parâmetro.



Macros podem ter mais de um parâmetro. Cada nome e seu tipo associado são separados por uma vírgula do próximo nome e seu tipo associado. A ordem é importante. O primeiro valor no EXEC da macro será associado ao primeiro valor na lista de parâmetros. O segundo valor no EXEC está associado ao segundo valor na lista de parâmetros e assim por diante.



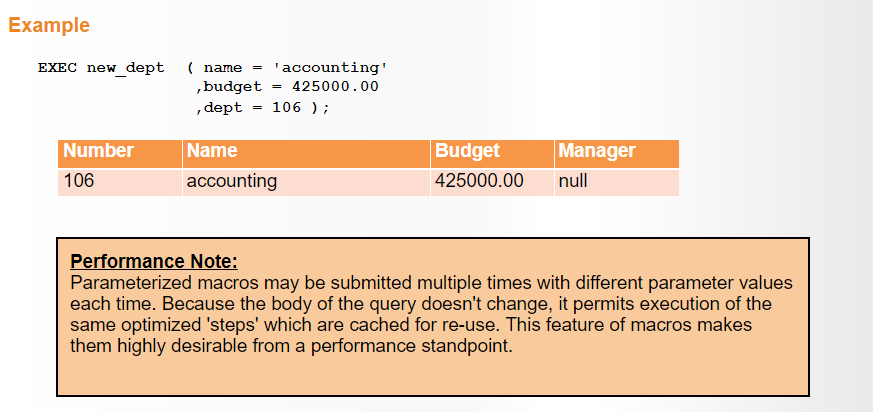
Você pode usar uma macro parametrizada para inserir dados. Esse é um uso comum, especialmente em aplicativos.  
Para EXECUTAR a macro, fornecemos valores para todos os parâmetros. Os dados de entrada devem estar na ordem especificada na lista de parâmetros da macro e devem corresponder ao número exato de parâmetros especificados na lista.



No segundo exemplo acima, o valor após 102 foi omitido, daí as duas vírgulas seguidas. Nesses casos, o valor do parâmetro será definido para o padrão especificado na instrução CREATE MACRO. Neste caso é 0.

Observe que, para este exemplo funcionar, as colunas budget e manager devem permitir valores nulos ou ter um valor padrão na definição da tabela de departamentos.

Outra opção disponível para passar parâmetros para uma macro é nomear explicitamente os parâmetros com um valor associado, conforme visto neste exemplo. Quando esta abordagem é selecionada, os parâmetros podem ser especificados em qualquer sequência. Observe no exemplo a seguir, parâmetros implicitamente default (como 'mgr') podem simplesmente ser omitidos sem contabilizar sua posição.



Para fazer os laboratórios a seguir, primeiro você precisará criar e preencher sua própria cópia da tabela de departamentos. Isso pode ser feito executando os seguintes comandos:

CREATE TABLE dept\_new, FALLBACK

(department\_number SMALLINT

,department\_name CHAR(30) NOT NULL

,budget\_amount DECIMAL(10,2)

,manager\_employee\_number INTEGER

)

UNIQUE PRIMARY INDEX (department\_number)

,UNIQUE INDEX (department\_name);

INSERT INTO dept\_new SELECT \*

FROM CustomerService.department;

1.) Crie uma macro parametrizada chamada dept\_macro que insere uma nova linha em sua tabela dept\_new e exibe a linha recém-inserida para verificação.

**CREATE MACRO dept\_macro**

**( dept   INTEGER**

**, name   CHAR(30)**

**, budget DEC(10,2)**

**, mgr    INTEGER)**

**AS**

**(   INSERT INTO   dept\_new**

**( department\_number**

**, department\_name**

**, budget\_amount**

**, manager\_employee\_number)**

**VALUES  (  :dept**

**,  :name**

**,  :budget**

**,  :mgr )**

**;**

**SELECT   department\_number       as deptnum**

**,department\_name         as deptname**

**,budget\_amount           as budget**

**,manager\_employee\_number as manager**

**FROM dept\_new**

**WHERE   department\_number = :dept**

**;**

**);**

**CREATE** **MACRO** dept\_macro

(

dept INTEGER

,**name** CHAR(30)

,budget DEC(10,2)

,mgr INTEGER))

**AS**

(

**INSERT** **INTO** dept\_new

(

department\_number

,department\_name

,budget\_amount

,manager\_employee\_number)

)

**Values** (

:dept

,:**name**

,:budget

,:mgr)

**;**

**SELECT** department\_number **as** DeptNumber

,department\_name **as** DeptName

,budget\_amount **as** Budget

, manager\_employee\_number **as** Manager

**From** dept\_new

**Where** department\_number = : dept**;**

)**;**

EXEC dept\_macro (700, ‘RH’, 80000, NULL);

Insira três linhas na tabela usando a macro. Use os valores abaixo para as colunas mostradas:

**EXEC** dept\_macro (700, 'RH', 80000, **NULL**)**;**

**EXEC** dept\_macro (701, 'Financeiro', 900000, 1005)**;**

**EXEC** dept\_macro (702, 'Contabilidade', 100000, 1011)**;**

número\_departamento nome\_departamento orçamento\_quantia gerente\_emprego

----------------- --------------- ------------- ----- --------

601 envio 800000 nulo

**EXEC** dept\_macro (601, 'Shipping', 800000.00, **NULL**)**;**

701 crédito 1200000 1018

**EXEC** dept\_macro (801, 'Credit', 1200000.00, 1018)**;**

905 pessoal do presidente 5000000 801

**EXEC** dept\_macro (905, 'President´s Staff', 5000000.00, 801)**;**

2.) EXPLAIN a execução da macro dept\_macro.

**EXPLAIN** **EXEC** dept\_macro (601, 'Shipping', 800000.00, **NULL**)**;**

EXPLAIN EXEC dept\_macro (601, 'Shipping', 800000.00, NULL);

1) First, we execute the following steps in parallel.

1) We do an INSERT into

"POLLYANA.FERREIRA@CAPGEMINI.COM".dept\_new. The estimated

time for this step is 0.56 seconds.

2) We do a **single-AMP** **RETRIEVE** step from

"POLLYANA.FERREIRA@CAPGEMINI.COM".dept\_new by way of the

**unique** **primary** index

""**POLLYANA.FERREIRA**@**CAPGEMINI.COM**".dept\_new.department\_number

= 601" with no residual conditions. The estimated time for

this step is 0.01 seconds.

2) Finally, we send out an **END TRANSACTION** step to all AMPs involved

in processing the request.

-> No rows are returned to the user as the result of statement 1.

The row is sent directly back to the user as the result of

statement 2.

The total estimated time is 0.56 seconds.

3.) Use uma instrução INSERT SELECT para copiar todas as linhas da tabela de funcionários CustomerService para a tabela emp\_new criada no laboratório do primeiro módulo.Escreva uma macro parametrizada chamada emp\_macro para primeiro selecionar e, em seguida, atualize sua tabela de funcionários para dar a todos os funcionários de um determinado departamento um aumento de salário fixo. O número real do departamento e o aumento salarial serão fornecidos no momento da execução. A seleção deve preceder a atualização e deve incluir o número do funcionário, sobrenome e duas colunas salariais: uma intitulada "Salário atual" e a outra intitulada "Novo salário". Limite o sobrenome a 10 caracteres e classifique por sobrenome. Execute esta macro para dar aos funcionários do departamento 401 um aumento de US$ 1.000. Execute-o novamente para reverter os efeitos da execução anterior.

**CREATE** **MACRO** emp\_macro (dept INTEGER , raise DECIMAL (5))

**AS**

(**SELECT** employee\_number

,last\_name (char (10))

,salary\_amount **as** "Current Salary"

,salary\_amount + :raise **as** "New Salary"

**FROM** emp\_new

**WHERE** department\_number = :dept

**ORDER** **BY** 2

**;**

**UPDATE** emp\_new

**SET** salary\_amount = salary\_amount + :raise

**WHERE** department\_number = :dept

**;**

)**;**

**EXEC** emp\_macro (401,1000)**;**

**EXEC** emp\_macro (401, -1000)**;**

4.) Escreva uma macro parametrizada chamada salário\_macro para criar um relatório mostrando o sobrenome, departamento e salário dos funcionários dentro de uma faixa salarial que será fornecida em tempo de execução. Execute a macro para localizar todos os funcionários na faixa de renda acima de $ 50.000 abaixo de $ 100.000. Execute-o novamente para encontrar todos os funcionários na faixa de renda abaixo de $ 30.000.

**CREATE** **MACRO** salary\_macro(low DECIMAL (10,2), high DECIMAL (10,2))

**AS**

( **SELECT** last\_name

,department\_number

,salary\_amount

**FROM** emp\_new

**WHERE** salary\_amount BETWEEN :low and :high

**ORDER** **BY** 3

**;**

)**;**

**EXEC** salary\_macro (50001,99999)**;**

**EXEC** salary\_macro (high = 29999, low = 0)**;**

**MODULE 4: VIEWS**

Após a conclusão deste módulo, você deverá ser capaz de:

CRIE uma visualização.

Use uma exibição para fornecer acesso seguro aos dados.

DROP ou SUBSTITUIR uma visualização.

Liste vários motivos para usar exibições.

Uma visão funciona como uma 'janela' em uma tabela. Ele permite que os usuários personalizem seu acesso às tabelas base por:

Restringindo quais colunas são visíveis na tabela base.

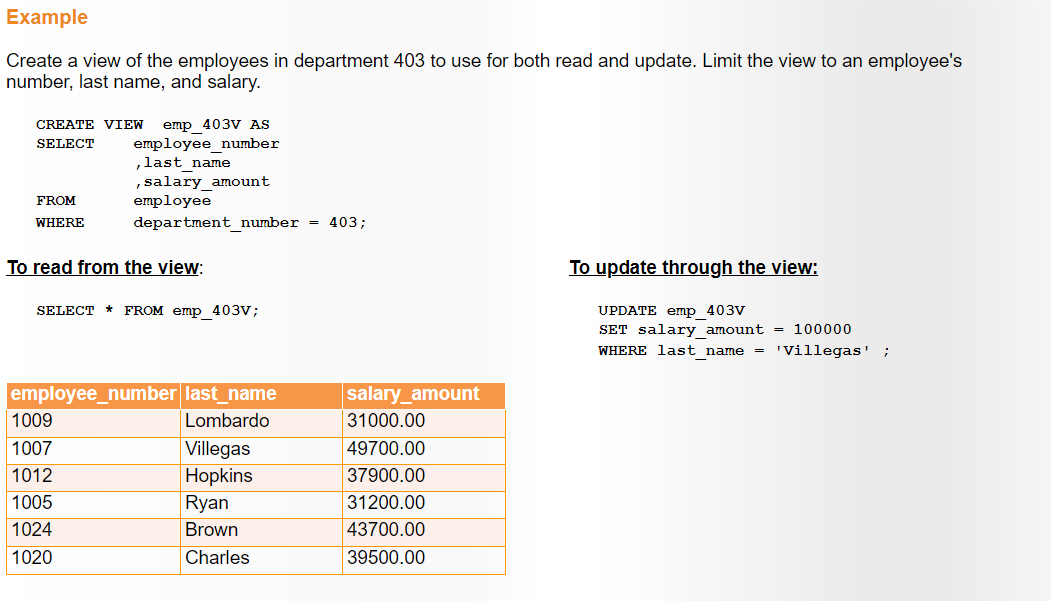
Restringindo quais linhas são visíveis na tabela base.

Combinando colunas e linhas de várias tabelas base.

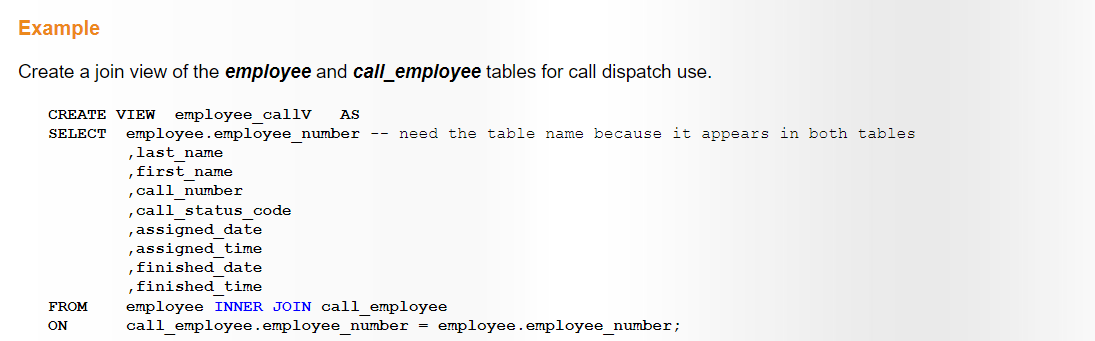
Você pode restringir as colunas da(s) tabela(s) base listando explicitamente os nomes de coluna desejados da(s) tabela(s) base na definição de visualização.

Você pode restringir quais linhas podem ser acessadas usando a cláusula WHERE na definição de visualização. Isso limitará as linhas retornadas de uma instrução SELECT referenciando a exibição.

Convenções de nomenclatura: muitas vezes você verá empresas nomeando suas visualizações com um "V" no nome para enfatizar que é uma visualização e não uma tabela. Isso não é necessário, mas pode ser útil.

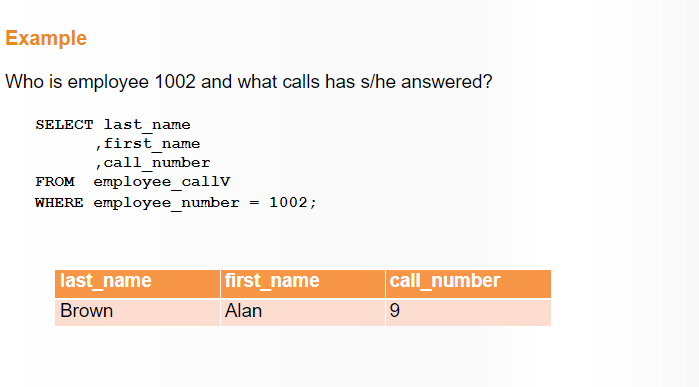


Uma 'visualização de junção' consiste em colunas de mais de uma tabela. Em outras palavras, é uma exibição com uma junção.

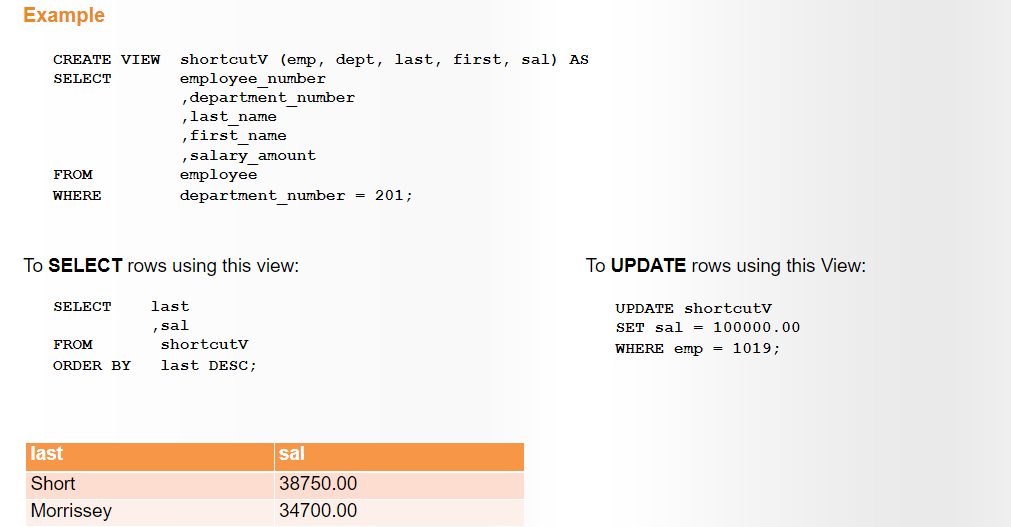


A utilização desta view permite SELECT das informações de ambas as tabelas, sem que o usuário precise saber como juntar as tabelas. Observe que os nomes das colunas que aparecem em ambas as tabelas (ou seja, Employees\_number) devem ser qualificados na definição

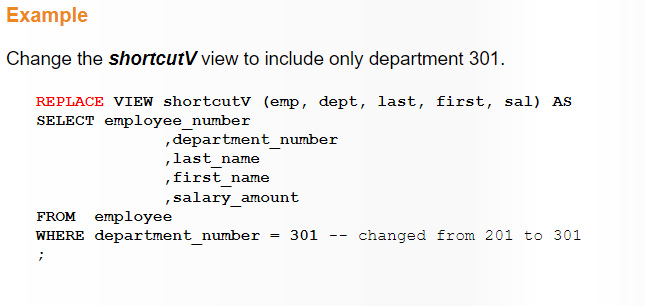
O uso de uma visualização de junção permite SELECT de informações de ambas as tabelas, sem que o usuário precise saber como unir as tabelas.



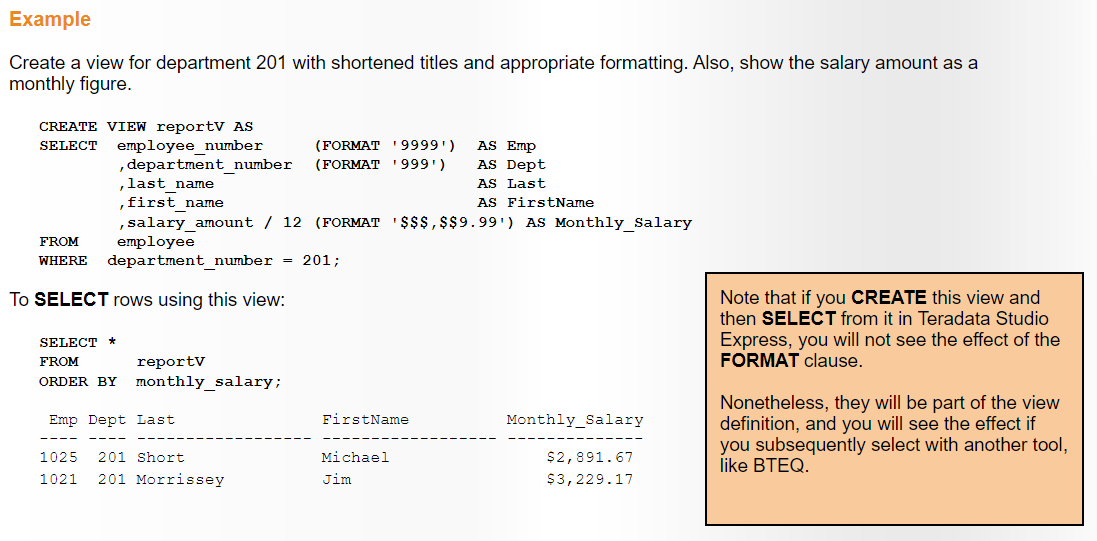
Você pode criar uma exibição que renomeia as colunas. Isso geralmente é usado para encurtar nomes de coluna longos, facilitando a criação de consultas.



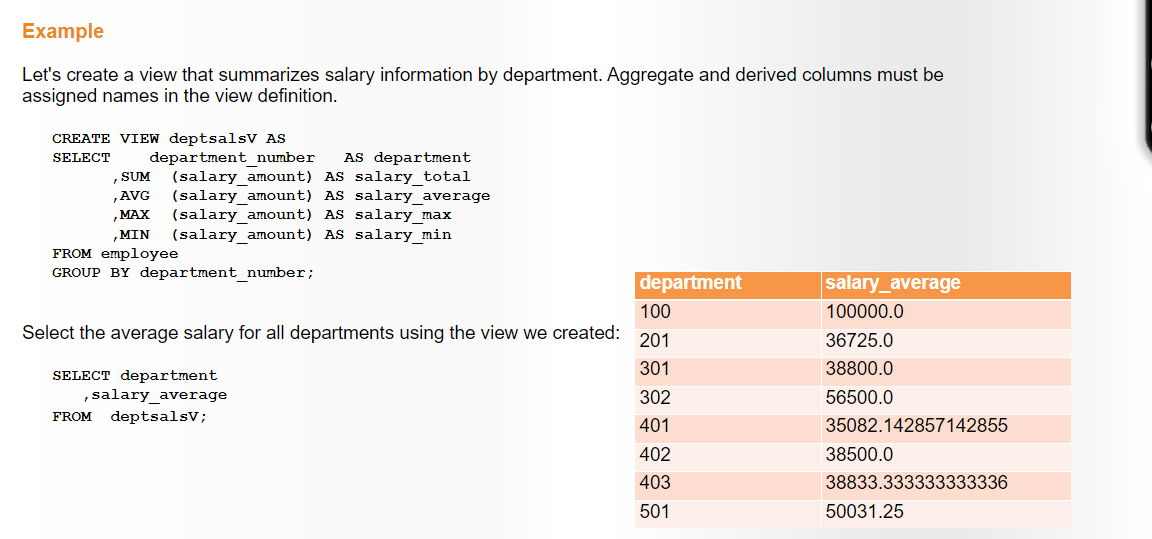
Você pode SUBSTITUIR (REPLACE)uma exibição para incluir ou remover colunas ou valores específicos. Isso significa que você não precisa descartá-lo e recriá-lo e, assim, manter todos os privilégios que foram concedidos na exibição.



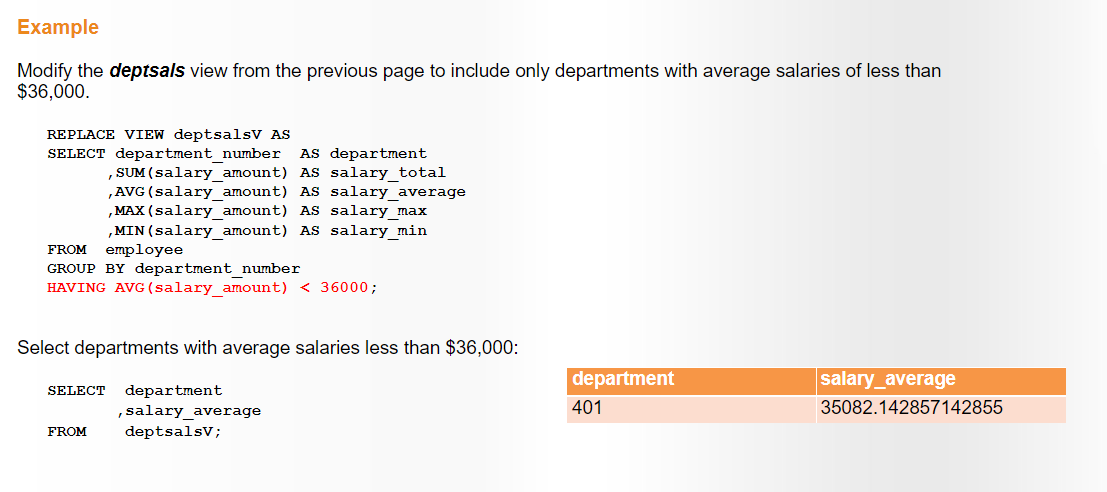
A cláusula FORMAT pode ser usada para definir formatos e a cláusula AS pode ser usada para atribuir novos nomes de colunas.



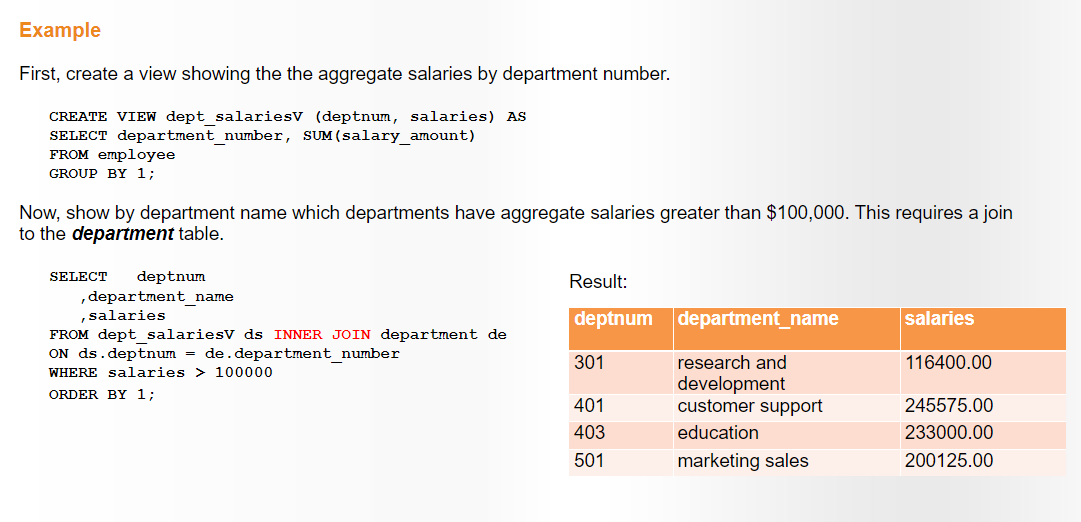
Você pode criar uma exibição que resume as informações usando agregações.



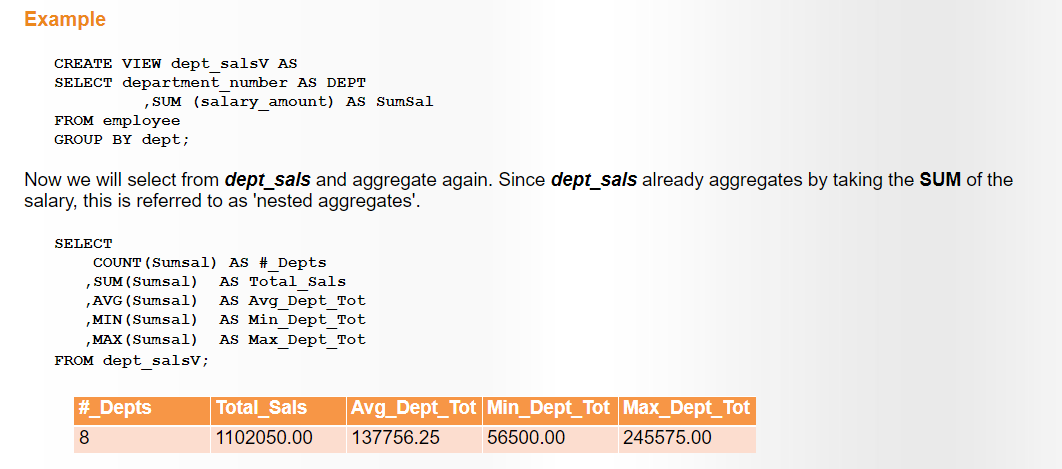
Você pode usar a cláusula HAVING em uma definição de exibição para restringir quais grupos participarão da exibição.



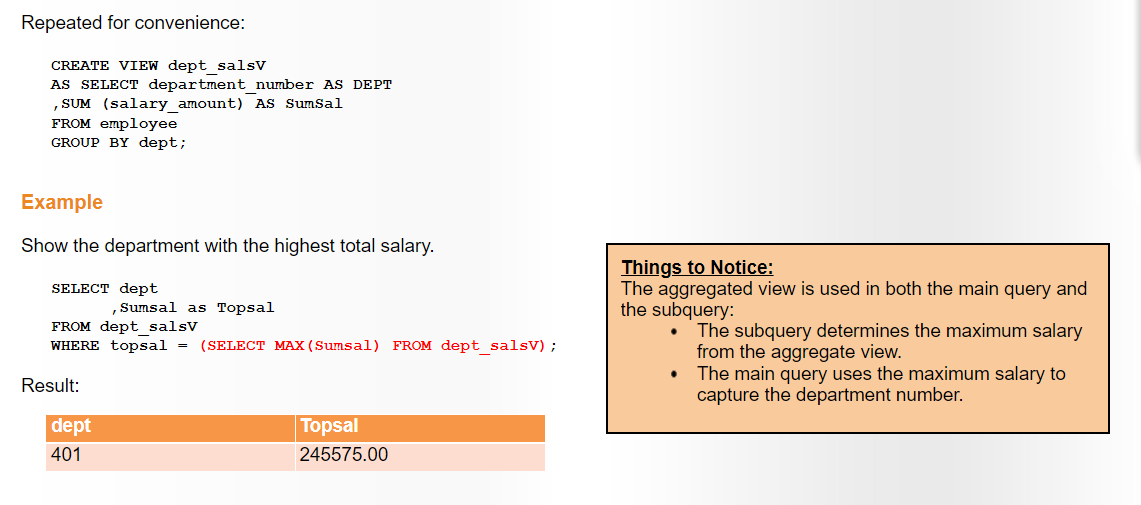
Visto que as visualizações são realmente tabelas virtuais, elas podem ser unidas a outras tabelas e/ou visualizações em uma consulta.

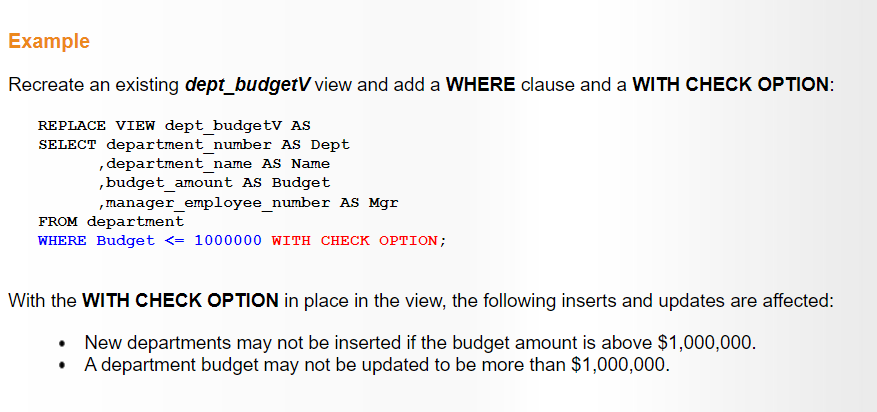


Você pode aninhar agregações agregando a partir de uma visualização que tenha agregação definida em sua criação.

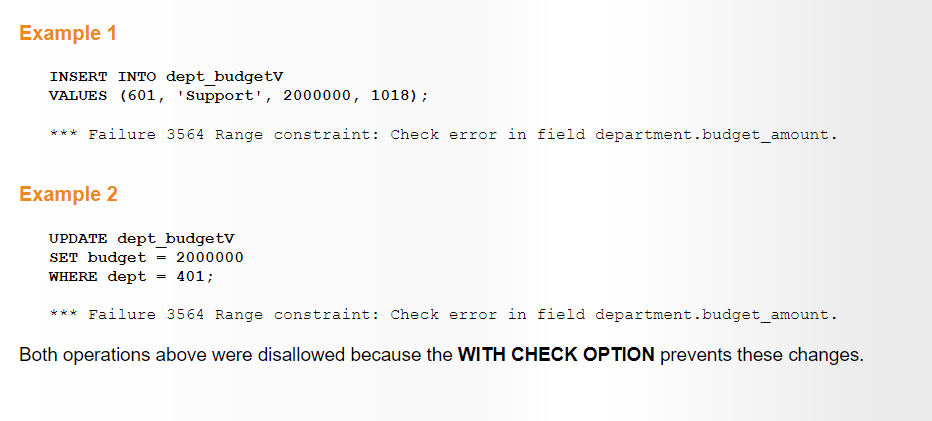


Agregados aninhados em uma cláusula WHERE podem ser usados ​​para limitar as linhas de resposta.

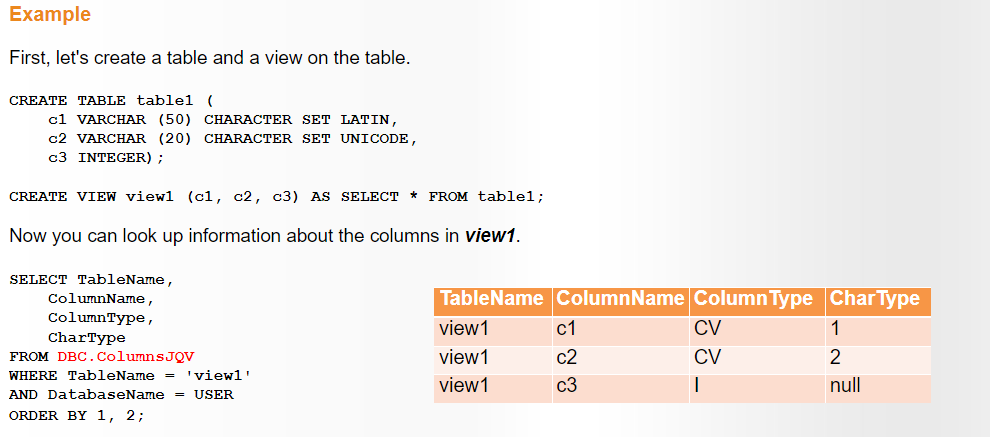


Em uma visualização, WITH CHECK OPTION controla a capacidade de atualização e inserção de usuários com esses privilégios na visualização. Um usuário não pode inserir ou atualizar uma linha se a linha resultante estiver fora da exibição. Dito de outra forma, uma inserção ou atualização será negada se a linha resultante violar as restrições da cláusula WHERE na definição da exibição.  


Os usuários não poderão atualizar ou inserir linhas cujos valores violem as restrições da cláusula WHERE na definição da exibição.



Se você deseja encontrar informações sobre as colunas subjacentes nas exibições, pode usar essas exibições no dicionário de dados: DBC.ColumnsQV[X] e DBC.ColumnsJQV[X] (o "X" é um filtro opcional, que filtra com base em os direitos de acesso do usuário). A exibição DBC.ColumnsQV[X] cobre todos os objetos de banco de dados, mas a exibição DBC.ColumnsJQV[X] cobre apenas tabelas e exibições. Por causa disso, DBC.ColumnsJQV[X] tem um desempenho um pouco melhor.



Restrições ao usar exibições

Um índice não pode ser criado em uma exibição.

Você só pode usar uma cláusula ORDER BY se também tiver uma cláusula TOP n.

A cláusula WHERE de um SELECT em uma exibição pode fazer referência a todas as colunas agregadas dessa exibição.

As colunas derivadas e agregadas devem receber um nome usando uma das seguintes técnicas:

Cláusula NAMED.

Cláusula AS (padrão ANSI).

Criando uma lista de nomes de colunas atribuídos na definição CREATE VIEW.

Uma visualização não pode ser usada para ATUALIZAR se contiver:

Dados de mais de uma tabela (Join View).

A mesma coluna duas vezes.

Colunas derivadas.

Expressões.

Agregados.

Uma cláusula DISTINCT.

Cláusula TOP n.

Uma cláusula GROUP BY, HAVING ou QUALIFY.

Vantagens ao usar visualizações

Um nível adicional de segurança.

Ajuda no controle de privilégios de leitura e atualização.

Simplifique o acesso do usuário final aos dados.

Não afetado se uma coluna for adicionada a uma tabela subjacente.

Não é afetado se uma coluna for removida de uma tabela subjacente, a menos que a coluna removida seja referenciada pela exibição.

Sugestões para usar visualizações

Crie pelo menos uma exibição para cada tabela base para restringir o acesso do usuário aos dados que eles não precisam ver.

Crie exibições de consulta conforme necessário para situações ad hoc.

Use bloqueios de acesso ao criar exibições para maximizar a disponibilidade de dados para os usuários.

Laboratorio:

1.) Crie uma exibição chamada sales\_territoryV. Inclua o número do cliente, o nome do cliente e o número do representante de vendas para clientes na Califórnia. SUGESTÃO: A tabela de localização contém o número do cliente e o estado. Use uma junção em vez de uma subconsulta.

**CREATE** **VIEW** sales\_territoryV **AS**

**SELECT** l.customer\_number

,c.customer\_name

,c.sales\_employee\_number

**FROM** CustomerService.customer c

,CustomerService.location l

**WHERE** c.customer\_number = l.customer\_number

**AND** l.state = 'California'**;**

Os dados nesta exibição são atualizáveis?

Não pode fazer atualização pois não tenho acesso ao banco.

2.) Selecione todas as colunas da exibição sales\_territoryV.

**SELECT** \* **FROM** sales\_territoryV **;**

3.) Crie uma exibição agregada chamada dept\_salV que acessa a tabela customerservice.employee e consiste em duas colunas: um número de departamento e a soma dos salários desse departamento.

**CREATE** **VIEW** dept\_saIV (dep, sumsal)

**AS**

**SELECT** department\_number

,SUM(salary\_amount)

**FROM** CustomerService.employee

**GROUP BY 1**;

Selecione todas as linhas desta visualização.

**SELECT** \* **FROM** dept\_saIV**;**

4.) Usando a visão dept\_salV, produza um relatório mostrando o nome do departamento e o salário total departamental para cada departamento cujo total

**SELECT** department\_name

,sumsal

**FROM** CustomerService.department d

**JOIN** dept\_saIV ds

**ON** ds.dep = d.department\_number

**Where** sumsal > 100000**;**