# 13

# Manipulando Grandes Conjuntos de Dados

# Objetivos deste Capítulo

- Ao concluir este capítulo, você poderá:
  - Manipular dados usando subconsultas
  - Descrever os recursos de inserções em várias tabelas
  - Usar os seguintes tipos de inserções em várias tabelas:
    - `INSERT` Incondicional
    - `INSERT` de Criação de Pivô
    - `ALL INSERT` Condicional
    - `FIRST INSERT` Condicional
  - Intercalar linhas em uma tabela

13-2

**Objetivos deste Capítulo**

Neste capítulo, você aprenderá a manipular os dados do banco de dados Oracle usando subconsultas. Você também conhecerá as comandos de inserção em várias tabelas e o comando `MERGE`, além de aprender a controlar as alterações feitas no banco de dados.

# Subconsultas para Manipular Dados

- É possível usar subconsultas em comandos DML para:
  - Copiar dados de uma tabela para outra
  - Recuperar dados de uma visão inline
  - Atualizar dados em uma tabela com base nos valores de outra tabela
  - Remover linhas de uma tabela com base nas linhas de outra tabela

13-3

**Subconsultas para Manipular Dados**

As subconsultas podem ser usadas para recuperar dados a partir de uma tabela usada como entrada para fazer um `INSERT` em uma tabela diferentf. Desse modo, você pode copiar facilmente grandes volumes de dados de uma tabela para outra com uma único comando `SELECT`. Da mesma forma, você pode usar subconsultas para fazer atualizações e exclusões em massa, incluindo-as na cláusula `WHERE` das comandos `UPDATE` e `DELETE`. Também é possível usar subconsultas na cláusula `FROM` de um comando `SELECT`. Esse processo se chama visão inlinf.

# Copiando Linhas de Outra Tabela

- Crie o comando INSERT com uma subconsulta.

```
INSERT INTO repr_venda(codigo, nome, salario, comissao)
  SELECT cod_funcionario, sobrenome, salario, comissao
  FROM   funcionario
  WHERE  cod_cargo LIKE '%REP%';

4 linhas criadas.
```

- Não use a cláusula VALUES.
- Estabeleça uma correspondência entre o número de colunas na cláusula INSERT e o número de colunas na subconsulta.

13-4

## Copiando Linhas de Outra Tabela

É possível usar o comando INSERT para adicionar linhas a uma tabela cujos valores são provenientes de tabelas existentes. No lugar da cláusula VALUES, use uma subconsulta.

**Sintaxe**

```
INSERT INTO tabela [ coluna (, coluna) ] subconsulta;
```

Na sintaxe:

| | |
|---|---|
| *tabela* | é o nome da tabela |
| *coluna* | é o nome da coluna da tabela a ser preenchida |
| *subconsulta* | é a subconsulta que retorna linhas para a tabela |

O número de colunas e os respectivos tipos de dados na lista de colunas da cláusula INSERT devem corresponder ao número de valores e aos respectivos tipos de dados na subconsulta. Para criar uma cópia das linhas de uma tabela, use SELECT * na subconsulta.

```
INSERT INTO FUNC3
  SELECT *
  FROM   funcionario;
```

# Inserção Usando uma Subconsulta como

```
INSERT INTO
        (SELECT cod_funcionario, sobrenome,
                email, data_admissao, cod_cargo,
                salario, cod_departamento
         FROM   funcionario
         WHERE  cod_departamento = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ES_AUX', 5000, 50);

1 linha criada.
```

13-5

**Inserção Usando uma Subconsulta como Destino**

É possível usar uma subconsulta no lugar do nome da tabela na cláusula `INTO` do comando `INSERT`.

A lista de seleção da subconsulta deve ter o mesmo número de colunas que a lista de colunas da cláusula `VALUES`. Para a execução bem-sucedida do comando `INSERT`, todas as regras nas colunas da tabela base devem ser cumpridas. Por exemplo, não é possível especificar um código de funcionário duplicado nem omitir um valor de uma coluna `NOT NULL` obrigatória.

Essa aplicação de subconsultas ajuda a evitar que seja necessário criar uma visão apenas para executar uma inserção.

# Inserção Usando uma Subconsulta como

- Verifique os resultados.

```
SELECT   cod_funcionario, sobrenome, email,
         data_admissao,
         cod_cargo, salario, cod_departamento
FROM     funcionario
WHERE    cod_departamento = 50;
```

| COD_FUNCIONARIO | SOBRENOME | EMAIL | DATA_ADMISSAO | COD_CARGO | SALARIO | COD_DEPARTAMENTO |
|---|---|---|---|---|---|---|
| 124 | Brunni | MBRUNNI | 16/11/08 | ES_GER | 5800 | 50 |
| 141 | Lopes | TLOPES | 17/10/05 | ES_AUX | 3500 | 50 |
| 142 | Cabral | CCABRAL | 29/01/07 | ES_AUX | 3100 | 50 |
| 143 | Miranda | RMIRANDA | 15/03/08 | ES_AUX | 2600 | 50 |
| 144 | Chaves | PCHAVES | 09/07/08 | ES_AUX | 2500 | 50 |
| 99999 | Taylor | DTAYLOR | 07/06/99 | ES_AUX | 5000 | 50 |

13-6

**Inserção Usando uma Subconsulta como Destino (continuação)**

O exemplo mostra os resultados da subconsulta usada para identificar a tabela para o comando `INSERT`.

# Recuperando Dados com uma Subconsulta como Origem

```
SELECT  a.sobrenome, a.salario,
        a.cod_departamento, b.med_sal
FROM    funcionario a, (SELECT   cod_departamento,
                        AVG(salario) med_sal
                        FROM     funcionario
                        GROUP BY cod_departamento) b
WHERE   a.cod_departamento = b.cod_departamento
AND     a.salario > b.med_sal;
```

| SOBRENOME | SALARIO | COD_DEPARTAMENTO | MED_SAL |
|---|---|---|---|
| Carlos | 24000 | 90 | 19333,3333333333333333333333333333333333 |
| Thacker | 13000 | 20 | 9500 |
| Almeida | 12000 | 110 | 10150 |
| Brunni | 5800 | 50 | 3750 |
| Taylor | 5000 | 50 | 3750 |
| Schultz | 10500 | 80 | 10033,3333333333333333333333333333333333 |
| Sue | 11000 | 80 | 10033,3333333333333333333333333333333333 |
| Honorato | 9000 | 60 | 6400 |

13-7

**Recuperando Dados Usando uma Subconsulta como Origem**

Você pode usar uma subconsulta na cláusula `FROM` do comando `SELECT`, que é muito semelhante à forma como as visões são usadas. Uma subconsulta na cláusula `FROM` de umo comando `SELECT` também é chamada de visão *inlinf*. Uma subconsulta na cláusula `FROM` de um comando `SELECT` define uma origem de dados apenas para esse comando `SELECT` específico. O exemplo do slide exibe os sobrenomes dos funcionários, os salários, os números dos departamentos e os salários médios de todos os funcionários que recebem mais que o salário médio dos respectivos departamentos. A subconsulta na cláusula `FROM` é denominada `b`, e a consulta exterior faz referência à coluna `MED_SAL` usando esse apelido.

# Atualizando Duas Colunas com uma Subconsulta

- Atualize o cargo e o salário do funcionário 114 para corresponder ao cargo e ao salário do funcionário 205.

```
UPDATE    funcionario
SET       cod_cargo  = (SELECT  cod_cargo
                    FROM    funcionario
                    WHERE   cod_funcionario = 205),
          salario  = (SELECT  salario
                    FROM    funcionario
                    WHERE   cod_funcionario = 205)
WHERE     cod_funcionario    =  114;
1 linha atualizada.
```

13-8

**Atualizando Duas Colunas com uma Subconsulta**

É possível atualizar diversas colunas na cláusula `SET` de um comando `UPDATE` criando várias subconsultas.

**Sintaxe**

```
UPDATE tabela
SET     coluna  =         (SELECT    coluna
                           FROM tabela
                           WHERE condição)
       [ ,
        coluna  =         (SELECT    coluna
                           FROM tabela
                           WHERE condição)]
[WHERE  condição ];
```

# Atualizando Linhas com Base em Outra Tabela

- Use subconsultas nos comandos `UPDATE` para atualizar linhas de uma tabela com base em valores de outra tabela.

```
UPDATE   func3
SET      cod_departamento  = (SELECT cod_departamento
                             FROM funcionario
                             WHERE cod_funcionario = 100)
WHERE    cod_cargo         = (SELECT cod_cargo
                             FROM funcionario
                             WHERE cod_funcionario = 200);
1 linha atualizada.
```

13-9

**Atualizando Linhas com Base em Outra Tabela**

É possível usar subconsultas em comandos `UPDATE` para atualizar as linhas de uma tabela. O exemplo do slide atualiza a tabela `FUNC3` com base nos valores da tabela `FUNCIONARIO`. Ele altera o número do departamento de todos os funcionários com o código de cargo do funcionário 200 para o número do departamento atual do funcionário 100.

# Removendo Linhas com Base em Outra Tabela

- Use subconsultas em comandos `DELETE` para remover linhas de uma tabela com base nos valores de outra tabela.

```
DELETE FROM func3
WHERE  cod_departamento =
                      (SELECT cod_departamento
                       FROM   departamento
                       WHERE  nome_departamento
                              LIKE '%Público%');
1 linha deletada.
```

13-10

**Removendo Linhas com Base em Outra Tabela**

É possível usar subconsultas para remover linhas de uma tabela com base nos valores de outra tabela. O exemplo do slide remove todos os funcionários que trabalham em um departamento cujo nome contém a string “Público". A subconsulta pesquisa a tabela `DEPARTAMENTO` para localizar o número do departamento com base no nome do departamento que contém a string “Público". Em seguida, a subconsulta informa o número do departamento para a consulta principal, que remove as linhas de dados da tabela `FUNCIONARIO` com base nesse número de departamento.

## Cláusula WITH CHECK OPTION em DML

- Uma subconsulta é usada para identificar a tabela e as colunas do comando DML.
- A palavra-chave WITH CHECK OPTION impede a alteração de linhas que não estão na subconsulta.

```
INSERT INTO  (SELECT cod_funcionario, sobrenome, email,
                  data_admissao, cod_cargo, salario
              FROM   func3
              WHERE  cod_departamento = 50
                  WITH CHECK OPTION)
VALUES (99998, 'Smith', 'JSMITH',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ES_AUX', 5000);
INSERT INTO
         *
ERRO na linha 1:
ORA-01402: violação da cláusula where da view WITH CHECK OPTION
```

13-11

### Usando a Cláusula WITH CHECK OPTION em Comandos DML

Especifique WITH CHECK OPTION para indicar que, se a subconsulta for usada no lugar de uma tabela em um comando INSERT, UPDATE ou DELETE, não serão permitidas alterações nessa tabela que produzam linhas não incluídas na subconsulta.

No exemplo mostrado, a palavra-chave WITH CHECK OPTION é usada. A subconsulta identifica linhas que estão no departamento 50, mas o código do departamento não está na lista SELECT e não tem um valor especificado na lista VALUES. A inserção dessa linha resulta em um código de departamento nulo, que não está na subconsulta.

# Recurso de Default Explícito

- Com o recurso de default explícito, é possível usar a palavra-chave `DEFAULT` como um valor de coluna onde se deseja especificar o valor padrão de coluna.
- Esse recurso é existe para manter a compatibilidade com o padrão SQL:1999.
- O recurso permite ao usuário controlar onde e quando o valor padrão deve ser aplicado aos dados.
- É possível usar defaults explícitos em comandos `INSERT` e `UPDATE`.

13-12

**Recurso de Default Explícitos**

É possível usar a palavra-chave `DEFAULT` em comandos `INSERT` e `UPDATE` para identificar um valor de coluna padrão. Se não houver um valor padrão, será usado um valor nulo.

A opção `DEFAULT` dispensa a codificação do valor padrão nos programas e a consulta ao dicionário para encontrá-lo, como se fazia antes da introdução deste recurso. A codificação do valor padrão é um problema quando ele se altera porque o código consequentemente precisa ser alterado.

# Usando Valores Default Explícitos

- DEFAULT com INSERT:

```
INSERT INTO deptm3
  (cod_departamento, nome_departamento, cod_gerente)
VALUES (300, 'Engenharia', DEFAULT);
```

- DEFAULT com UPDATE:

```
UPDATE deptm3
SET cod_gerente = DEFAULT
WHERE cod_departamento = 10;
```

13-13

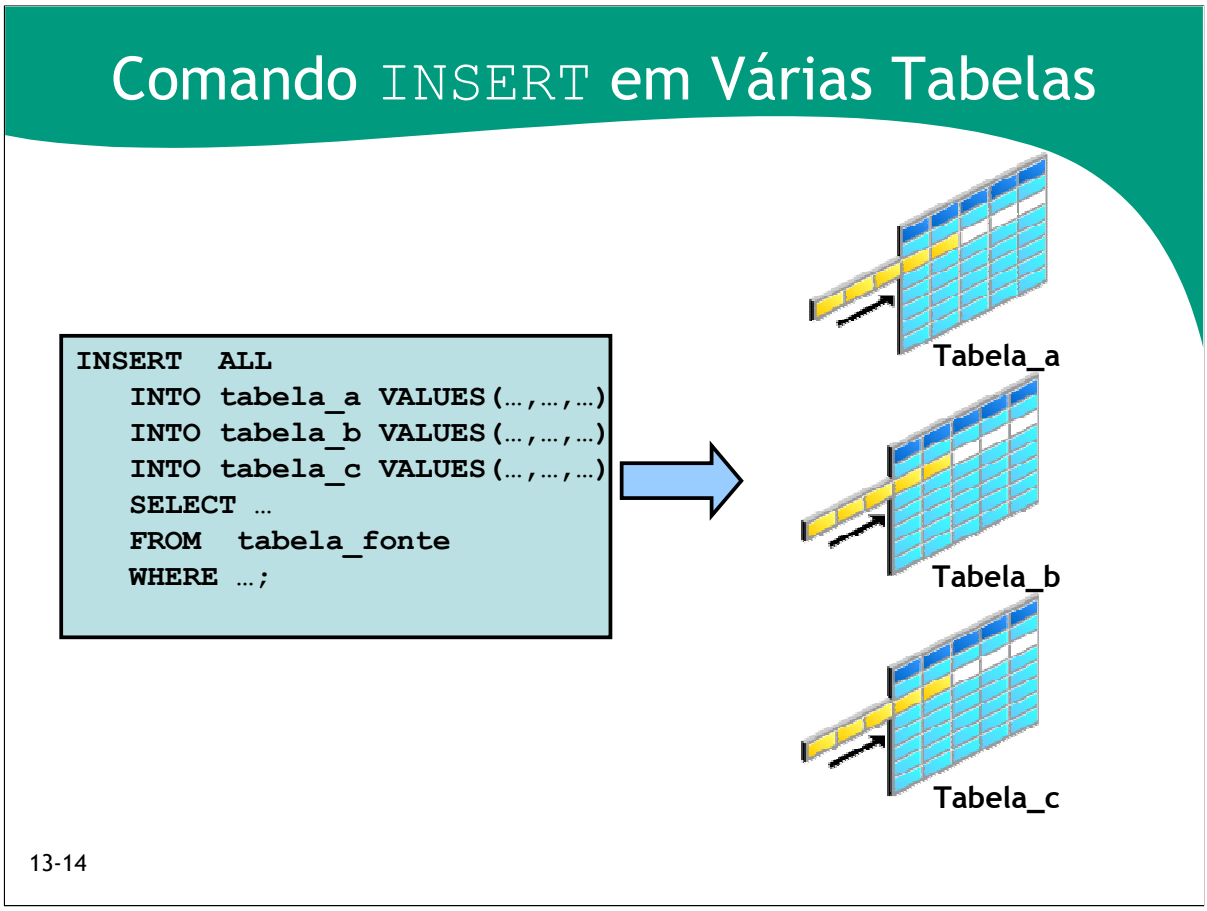
**Usando Valores Default Explícitos**

Especifique DEFAULT para definir a coluna para o valor especificado anteriormente como o seu valor padrão. Se não tiver sido especificado um valor padrão para a coluna correspondente, o Oracle definirá a coluna como nula.

No primeiro exemplo do slide, o comando INSERT usa um valor padrão para a coluna COD_GERENTE. Se não houver um valor padrão definido para a coluna, um valor nulo será inserido.

O segundo exemplo usa o comando UPDATE para definir a coluna COD_GERENTE com um valor padrão para o departamento 10. Se nenhum valor padrão for definido para a coluna, o valor será alterado para nulo.

Ao criar uma tabela, você poderá especificar um valor padrão para uma coluna. Esse assunto já foi abordado em um capítulo anterior deste curso.




### Comando INSERT em Várias Tabelas

Em um comando INSERT em várias tabelas, são inseridas as linhas calculadas derivadas das linhas retornadas da avaliação de uma subconsulta em uma ou mais tabelas.

Os comandos INSERT em várias tabelas podem desempenhar um papel muito útil em um cenário de data warehousf. Carregue o data warehouse regularmente para que ele atenda ao propósito de facilitar a análise de negócios.

## Comando INSERT em Várias Tabelas

- O comando INSERT…SELECT pode ser usado para inserir linhas em várias tabelas como parte de um único comando DML.
- Vários comandos INSERT podem ser usadas em sistemas de data warehouse para transferir dados de uma ou mais origens operacionais para um conjunto de tabelas de destino.
- Eles permitem uma melhoria significativa no desempenho em relação a:
  - Um único comando DML x vários comandos INSERT…SELECT
  - Um único comando DML x um procedimento para executar várias inserções usando a sintaxe IF...THEN

13-15

**Comando INSERT em Várias Tabelas (Continuação)**

As comandos INSERT em várias tabelas oferecem os benefícios do comando INSERT ... SELECT quando há várias tabelas como destino. Antes do Banco de Dados Oracle9*i*, para usar essa funcionalidade, era necessário lidar com *n* comandos INSERT ... SELECT independentes, processando, assim, os mesmos dados-fonte *n* vezes e aumentando a carga de trabalho de transformação *n* vezes.

Como ocorre com o comando INSERT ... SELECT existente, a novo comando pode ser paralelizado e usado com o mecanismo de carga direta para garantir um melhor desempenho.

Agora os registros de qualquer fluxo de entrada, como, por exemplo, uma tabela de banco de dados não relacional, podem ser convertidos em vários registros para um ambiente de tabela de banco de dados relacional.

## Comando INSERT em Várias Tabelas

- Os diversos tipos de comandos INSERT em várias tabelas são:
  - INSERT Incondicional
  - ALL INSERT Condicional
  - FIRST INSERT Condicional
  - INSERT de Criação de Pivô

13-16

**Tipos de Comandos INSERT em Várias Tabelas**

Os tipos de comandos INSERT em várias tabelas são:

- INSERT Incondicional
- ALL INSERT Condicional
- FIRST INSERT Condicional
- INSERT de Criação de Pivô

Use cláusulas distintas para indicar o tipo de comando INSERT a ser executado.

# Comando INSERT em Várias Tabelas

- Sintaxe

```
INSERT [ALL] [cláusula_de_condição_de_insert]
[cláusula_insert_into cláusula_values] (subconsulta)
```

- cláusula_de_condição_de_insert

```
[ALL] [FIRST]
[WHEN condição THEN] [cláusula_insert_into cláusula_values]
[ELSE] [cláusula_insert_into cláusula_values]
```

13-17

### Comandos INSERT em Várias Tabelas

O slide exibe o formato genérico para as comandos INSERT em várias tabelas.

**INSERT Incondicional: ALL cláusula_into**

Especifique ALL seguido por diversas cláusula_insert_intos para executar uma inserção incondicional em várias tabelas. O Oracle executa cada cláusula_insert_into uma vez por linha retornada pela subconsulta.

**INSERT Condicional: cláusula_de_condição_de_insert**

Especifique cláusula_de_condição_de_insert para executar um comando INSERT condicional em várias tabelas. O Oracle filtra cada cláusula_insert_into pela condição WHEN correspondente, que determina se essa cláusula_insert_into será executada. Uma único comando INSERT em várias tabelas pode conter até 127 cláusulas WHEN.

**INSERT Condicional: ALL**

Se você especificar ALL, o Oracle avaliará cada cláusula WHEN independentemente dos resultados da avaliação de qualquer outra cláusula WHEN. Para cada cláusula WHEN cuja condição é avaliada como verdadeira, o Oracle executa a lista de cláusulas INTO correspondentf.

## Comandos INSERT em Várias Tabelas (continuação)

### INSERT Condicional: FIRST

Se você especificar FIRST, o Oracle avaliará cada cláusula WHEN na ordem em que aparece no comando. Se a primeira cláusula WHEN for avaliada como verdadeira, o Oracle executará a cláusula INTO correspondente e ignorará as cláusulas WHEN subsequentes relativas a essa linha.

### INSERT Condicional: Cláusula ELSE

Para uma linha específica, se nenhuma cláusula WHEN for avaliada como verdadeira:

- Caso você tenha especificado uma cláusula ELSE, o Oracle executará a lista de cláusulas INTO associada à cláusula ELSE.
- Caso você não especifique uma cláusula ELSE, o Oracle não executará nenhuma ação para essa linha.

### Restrições às Comandos INSERT em Várias Tabelas

- Você pode executar comandos INSERT em várias tabelas apenas em tabelas, mas não em visões nem em visões materializadas.
- Não é possível executar um comando INSERT em várias tabelas em uma tabela remota.
- Não é possível especificar uma expressão de coleta de tabelas ao executar um comando INSERT em várias tabelas.
- Em um comando INSERT em várias tabelas, não é possível combinar todas as cláusula_insert_intos para especificar mais de 999 colunas de destino.

## INSERT ALL Incondicional

- Selecione os valores de COD_FUNCIONARIO, DT_ADMISSAO, SALARIO e COD_GERENTE na tabela FUNCIONARIO para os funcionários cujo COD_FUNCIONARIO é maior que 200.
- Insira esses valores nas tabelas HISTORICO_SAL e HISTORICO_GER usando um comando INSERT em várias tabelas.

```
INSERT  ALL
   INTO historico_sal VALUES(CODIGO,DT_ADMISSAO,SAL)
   INTO historico_ger VALUES(CODIGO,GER,SAL)
   SELECT cod_funcionario CODIGO, data_admissao DT_ADMISSAO,
          salario SAL, cod_gerente GER
   FROM  funcionario
   WHERE cod_funcionario > 200;
8 linhas criadas.
```

13-19

### INSERT ALL Incondicional

O exemplo do slide insere linhas nas tabelas HISTORICO_SAL e HISTORICO_GER.

O comando SELECT recupera, na tabela FUNCIONARIO, os detalhes sobre o código do funcionário, a data de admissão, o salário e o código do gerente dos funcionários cujo código é maior que 200. Os detalhes sobre o código do funcionário, a data de admissão e o salário são inseridos na tabela HISTORICO_SAL. Os detalhes sobre o código do funcionário, o código do gerente e o salário são inseridos na tabela HISTORICO_GER.

Esso comando INSERT é denominado INSERT incondicional, pois não são aplicadas outras restrições às linhas recuperadas pelo comando SELECT. Todas as linhas recuperadas pelo comando SELECT são inseridas nas duas tabelas, HISTORICO_SAL e HISTORICO_GER. A cláusula VALUES nos comandos INSERT especifica as colunas do comando SELECT que precisam ser inseridas em cada uma das tabelas. Cada linha retornada pelo comando SELECT resulta em duas inserções, uma na tabela HISTORICO_SAL e outra na tabela HISTORICO_GER.

É possível interpretar as 8 linhas criadas e retornadas como um total de oito inserções executadas nas tabelas-base, HISTORICO_SAL e HISTORICO_GER.

# INSERT ALL Condicional

- Selecione os valores de COD_FUNCIONARIO, DT_ADMISSAO, SALARIO e COD_GERENTE na tabela FUNCIONARIO para os funcionários cujo COD_FUNCIONARIO é maior que 200.
- Se o valor de SALARIO for maior que R$10.000, insira esse valor na tabela HISTORICO_SAL usando um comando INSERT condicional em várias tabelas.
- Se o valor de COD_GERENTE for maior que 200, insira esse valor na tabela HISTORICO_GER usando um comando INSERT condicional em várias tabelas.

13-20

**INSERT ALL Condicional**

As orientações para criar umo comando INSERT ALL condicional estão especificadas no slidf. A solução para esse problema está indicada na próxima página.

## INSERT ALL Condicional

```
INSERT ALL
  WHEN SAL > 10000 THEN
    INTO historico_sal VALUES(CODIGO,DT_ADMISSAO,SAL)
  WHEN GER > 200   THEN
    INTO historico_ger VALUES(CODIGO,MGR,SAL)
    SELECT cod_funcionario CODIGO,data_admissao DT_ADMISSAO,
           salario SAL, cod_gerente GER
    FROM   funcionario
    WHERE  cod_funcionario > 200;
4 linhas criadas.
```

13-21

### INSERT ALL Condicional (continuação)

O exemplo do slide é semelhante ao exemplo anterior, pois ele insere linhas nas tabelas HISTORICO_SAL e HISTORICO_GER. O comando SELECT recupera, na tabela FUNCIONARIO, os detalhes sobre o código do funcionário, a data de admissão, o salário e o código do gerente dos funcionários cujo código é maior que 200. Os detalhes sobre o código do funcionário, a data de admissão e o salário são inseridos na tabela HISTORICO_SAL. Os detalhes sobre o código do funcionário, o código do gerente e o salário são inseridos na tabela HISTORICO_GER.

Esso comando INSERT é denominado ALL INSERT condicional, pois são aplicadas outras restrições às linhas recuperadas pelo comando SELECT. Das linhas recuperadas pelo comando SELECT, apenas aquelas cujo valor na coluna SAL é maior que 10.000 são inseridas na tabela HISTORICO_SAL. Da mesma forma, apenas as linhas cujo valor na coluna GER é maior que 200 são inseridas na tabela HISTORICO_GER.

Observe que, diferentemente do exemplo anterior, no qual oito linhas foram inseridas nas tabelas, neste exemplo, apenas quatro linhas são inseridas.

É possível interpretar as 4 linhas criadas e retornadas como um total de quatro operações INSERT executadas nas tabelas-base, HISTORICO_SAL e HISTORICO_GER.

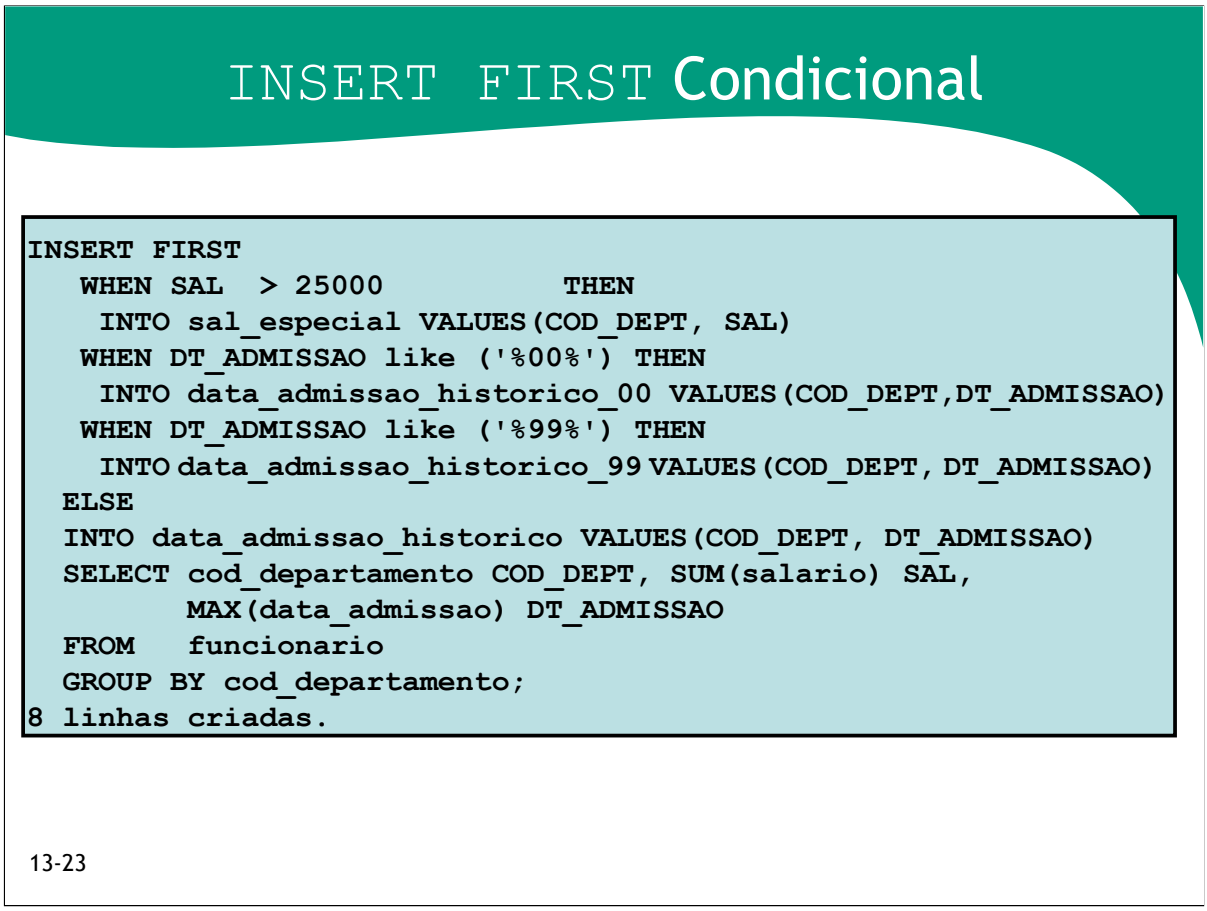
## FIRST INSERT Condicional

- Selecione COD_DEPARTAMENTO, SUM(SALARIO) e MAX(DATA_ADMISSAO) na tabela FUNCIONARIO.
- Se o valor de SUM(SALARIO) for maior que R$25.000, insira esse valor em SAL_ESPECIAL usando um comando FIRST INSERT condicional em várias tabelas.
- Se a primeira cláusula WHEN for avaliada como verdadeira, as cláusulas WHEN subsequentes relativas a essa linha deverão ser ignoradas.
- Insira as linhas que não atenderem à primeira condição WHEN na tabela DATA_ADMISSAO_HISTORICO_00, DATA_ADMISSAO_HISTORICO_99 ou DATA_ADMISSAO_HISTORICO, com base no valor da coluna DT_ADMISSAO usando um comando INSERT condicional em várias tabelas.

13-22

### FIRST INSERT Condicional

As orientações para criar umo comando FIRST INSERT condicional estão especificadas no slidf. A solução para esse problema está indicada na próxima página.




## INSERT FIRST Condicional (continuação)

O exemplo do slide insere linhas em mais de uma tabela, usando um único comando INSERT. O comando SELECT recupera os detalhes sobre o código, o salário total e a data de admissão máxima relativos a todos os departamentos da tabela FUNCIONARIO.

Esse comando INSERT é denominado FIRST INSERT condicional, pois é feita uma exceção para os departamentos cujo salário total é maior que R$25.000. A condição WHEN ALL > 25.000 é avaliada primeiro. Se o salário total de um departamento for maior que R$25.000, o registro será inserido na tabela SAL_ESPECIAL independentemente da data de admissão. Se a primeira cláusula WHEN for avaliada como verdadeira, o Oracle executará a cláusula INTO correspondente e ignorará as cláusulas WHEN subsequentes relativas a essa linha.

Quando as linhas não atendem à primeira condição WHEN (WHEN SAL > 25.000), as outras condições são avaliadas exatamente como o comando INSERT condicional, e os registros recuperados pelo comando SELECT são inseridos na tabela DATA_ADMISSAO_HISTORICO_00, DATA_ADMISSAO_HISTORICO_99 ou DATA_ADMISSAO_HISTORICO, com base no valor da coluna DT_ADMISSAO.

# INSERT de Criação de Pivô

- Suponha que você receba um conjunto de registros de vendas de uma tabela de banco de dados não relacional, FONTE_DADOS_VENDA, no seguinte formato:

  COD_FUNCIONARIO, COD_SEMANA, VENDAS_SEG, VENDAS_TER, VENDA_WED, VENDAS_QUI, VENDAS_SEX

- Você quer armazenar esses registros na tabela VENDAS_INFO em um formato relacional:

  COD_FUNCIONARIO, SEMANA, VENDA

- Com um comando INSERT de criação de pivô, converta o conjunto de registros de vendas da tabela de banco de dados não relacional em um formato relacional.

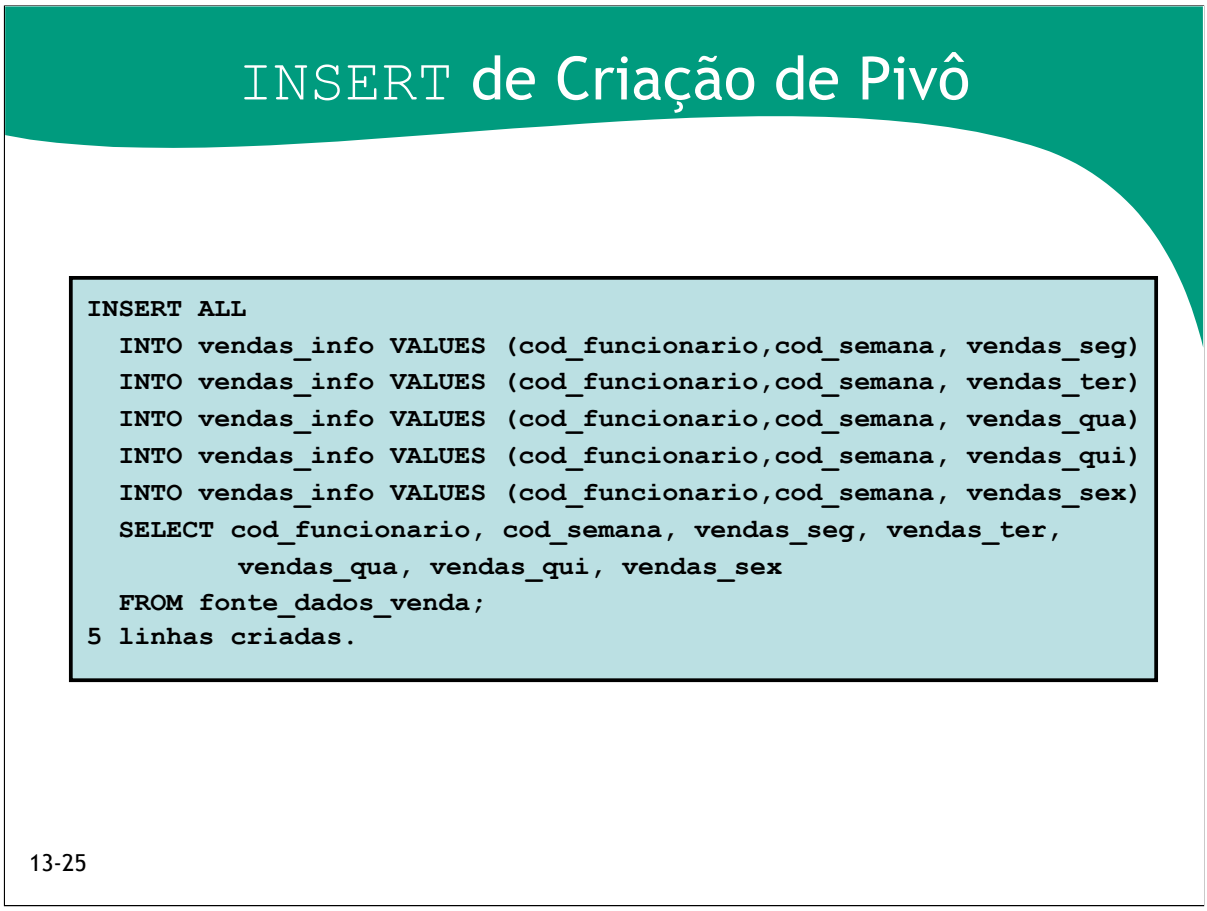
13-24

**INSERT de Criação de Pivô**

A criação de pivô é uma operação na qual você precisa criar uma transformação de forma que cada registro de qualquer fluxo de entrada, como uma tabela de banco de dados não relacional, seja convertido em vários registros para um ambiente de tabela de banco de dados relacional.

Para solucionar o problema mencionado no slide, é preciso criar uma transformação para que cada registro da tabela de banco de dados não relacional original, FONTE_DADOS_VENDA, seja convertido em cinco registros para a tabela VENDAS_INFO de data warehousf. Essa operação é geralmente chamada de *criação de pivô*.

As orientações para desenvolver um comando INSERT de criação de pivô estão especificadas no slidf. A solução para esse problema está indicada na próxima página.




## INSERT de Criação de Pivô (continuação)

No exemplo do slide, os dados de vendas, relativos aos detalhes das vendas realizadas por um representante de vendas em cada dia de uma semana com um código de semana específico, são recebidos da tabela de banco de dados não relacional FONTE_DADOS_VENDA.

```
DESC FONTE_DADOS_VENDA
```

| Table | Column | Tipo De Dados | Tamanho | Precisão | Escala | Chave Primária | Anulável |
|---|---|---|---|---|---|---|---|
| FONTE_DADOS_VENDA | COD_FUNCIONARIO | Number | - | 6 | 0 | - | ✓ |
| | COD_SEMANA | Number | - | 2 | 0 | - | ✓ |
| | VENDAS_SEG | Number | - | 8 | 2 | - | ✓ |
| | VENDAS_TER | Number | - | 8 | 2 | - | ✓ |
| | VENDAS_QUA | Number | - | 8 | 2 | - | ✓ |
| | VENDAS_QUI | Number | - | 8 | 2 | - | ✓ |
| | VENDAS_SEX | Number | - | 8 | 2 | - | ✓ |

## INSERT de Criação de Pivô (continuação)

```
SELECT * FROM FONTE_DADOS_VENDA;
```

| COD_FUNCIONARIO | COD_SEMANA | VENDAS_SEG | VENDAS_TER | VENDAS_QUA | VENDAS_QUI | VENDAS_SEX |
|---|---|---|---|---|---|---|
| 176 | 6 | 2000 | 3000 | 4000 | 5000 | 6000 |

```
DESC VENDAS_INFO
```

| Table | Column | Tipo De Dados | Tamanho | Precisão | Escala | Chave Primária | Anulável |
|---|---|---|---|---|---|---|---|
| VENDAS_INFO | COD_FUNCIONARIO | Number | - | 6 | 0 | - | ✓ |
| | SEMANA | Number | - | 2 | 0 | - | ✓ |
| | VENDA | Number | - | 8 | 2 | - | ✓ |

```
SELECT * FROM vendas_info;
```

| COD_FUNCIONARIO | SEMANA | VENDA |
|---|---|---|
| 176 | 6 | 2000 |
| 176 | 6 | 3000 |
| 176 | 6 | 4000 |
| 176 | 6 | 5000 |
| 176 | 6 | 6000 |

No exemplo anterior, observe que, ao usar um comando `INSERT` de criação de pivô, uma linha da tabela `FONTE_DADOS_VENDA` é convertida em cinco registros para a tabela relacional, `VENDAS_INFO`.

# O Comando MERGE

- Permite atualizar ou inserir dados de forma condicional em uma tabela de banco de dados
- Executa uma operação UPDATE se a linha existir e uma operação INSERT se a linha for nova
  - Evita atualizações separadas
  - Melhora o desempenho e facilita o uso
  - É útil nas aplicações de data warehouse

13-27

## O Comando MERGE

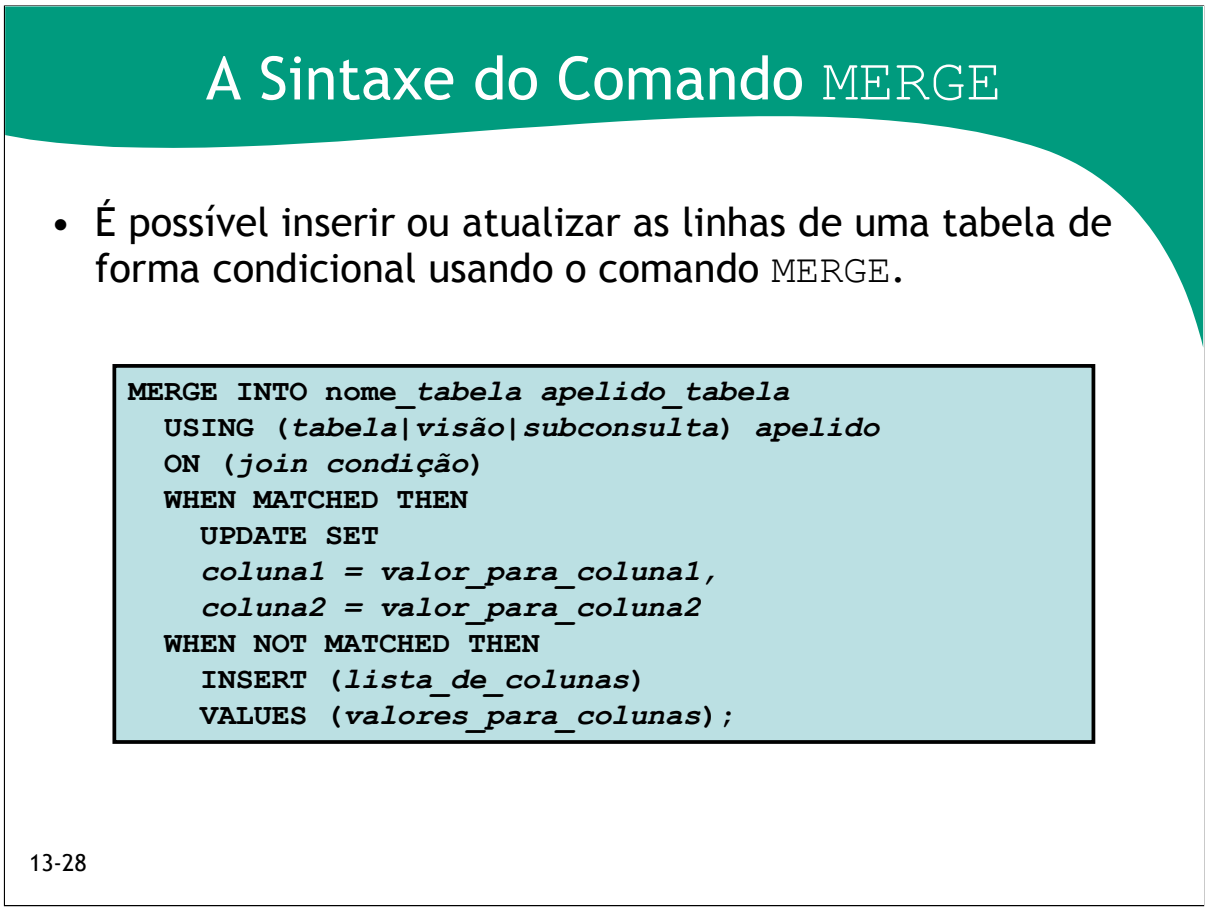
O Oracle suporta o comando MERGE para as operações INSERT, UPDATE e DELETE. Ao usar esse comando, você pode atualizar, inserir ou remover uma linha de forma condicional em uma tabela, evitando, assim, vários comandos DML. A decisão de efetuar uma atualização, inserção ou remoção na tabela de destino baseia-se na condição na cláusula ON.

Você precisa ter privilégios de objeto INSERT e UPDATE na tabela de destino e o privilégio de objeto SELECT na tabela de origem. Para especificar a cláusula DELETE de cláusula_update_merge, é preciso ter o privilégio de objeto DELETE na tabela de destino.

O comando MERGE é determinantf. Não é possível atualizar a mesma linha da tabela de destino várias vezes na mesmo comando MERGE.

Uma abordagem alternativa é usar loops PL/SQL e várias comandos DML. No entanto, o comando MERGE é fácil de usar e é expressa de forma mais simples como uma único comando SQL.

O comando MERGE é apropriado para várias aplicações de data warehousf. Por exemplo, em uma aplicação de data warehouse, talvez seja necessário trabalhar com dados provenientes de várias origens, alguns dos quais podem ser duplicados. Com o comando MERGE, é possível adicionar ou modificar linhas de forma condicional.




### A Sintaxe do Comando MERGE

É possível atualizar linhas existentes e inserir novas linhas de forma condicional usando o comando `MERGE`.

Na sintaxe:

| | |
|---|---|
| `INTO` | especifica a tabela de destino para atualização ou inserção |
| `USING` | identifica a origem dos dados a serem atualizados ou inseridos; pode ser uma tabela, uma visão ou uma subconsulta |
| `ON` | a condição com base na qual a operação `MERGE` efetua a atualização ou inserção |
| `WHEN MATCHED \| WHEN NOT MATCHED` | instrui o servidor sobre como responder aos resultados da condição de join |

# Intercalação de Linhas

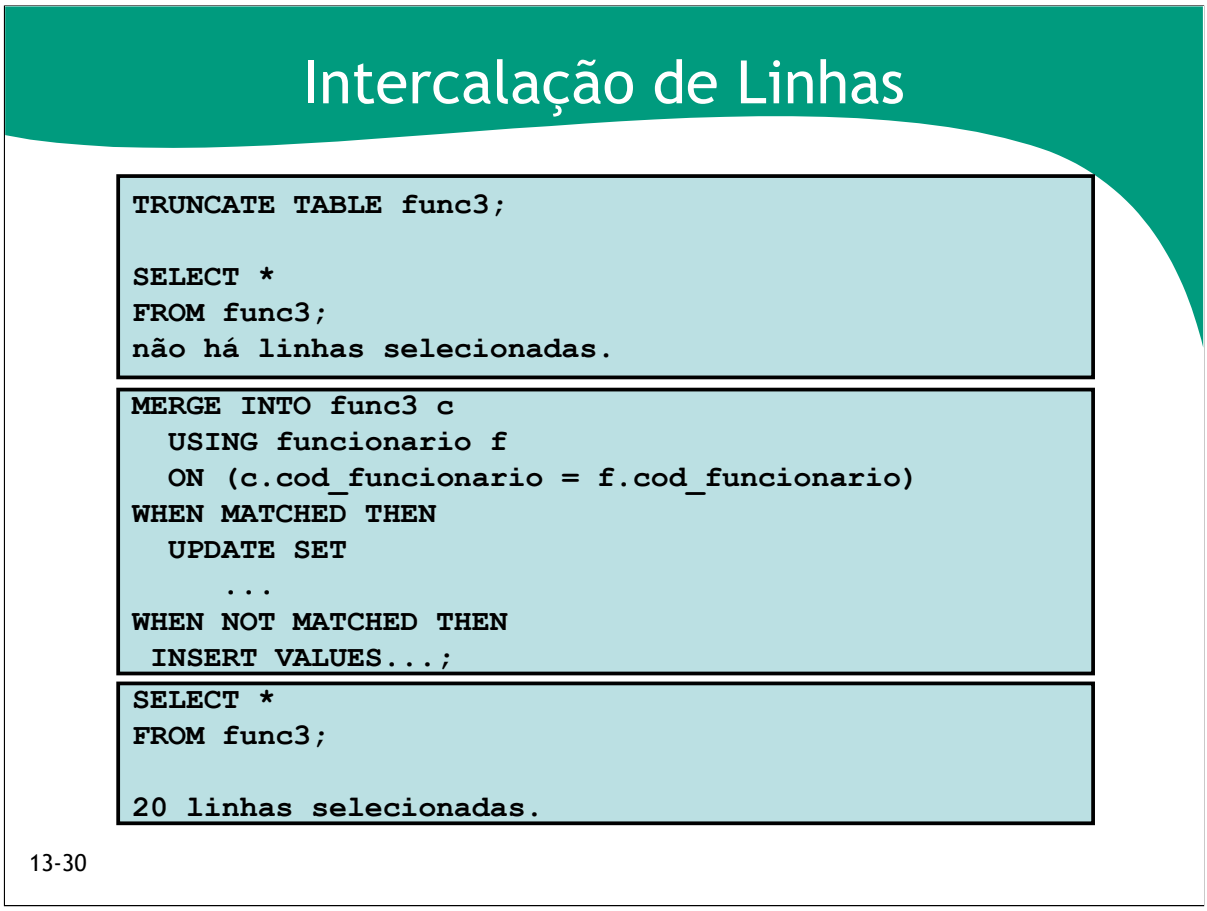
- Insira ou atualize linhas da tabela FUNC3 para corresponder à tabela FUNCIONARIO.

```
MERGE INTO func3  c
  USING funcionario f
  ON (c.cod_funcionario = f.cod_funcionario)
WHEN MATCHED THEN
  UPDATE SET
     c.nome             = f.nome,
     c.sobrenome        = f.sobrenome,
     ...
     c.cod_departamento = f.cod_departamento
WHEN NOT MATCHED THEN
 INSERT VALUES(f.cod_funcionario, f.nome, f.sobrenome,
          f.email, f.telefone, f.data_admissao, f.cod_cargo,
          f.salario, f.comissao, f.cod_gerente,
          f.cod_departamento);
```

13-29

**Exemplo de Intercalação de Linhas**

```
MERGE INTO func3 c
  USING funcionario f
  ON (c.cod_funcionario = f.cod_funcionario)
WHEN MATCHED THEN
  UPDATE SET
    c.nome                = f.nome,
    c.sobrenome           = f.sobrenome,
    c.email               = f.email,
    c.telefone            = f.telefone,
    c.data_admissao       = f.data_admissao,
    c.cod_cargo           = f.cod_cargo,
    c.salario             = f.salario,
    c.comissao            = f.comissao,
    c.cod_gerente         = f.cod_gerente,
    c.cod_departamento    = f.cod_departamento
WHEN NOT MATCHED THEN
  INSERT VALUES(f.cod_funcionario, f.nome, f.sobrenome,
       f.email, f.telefone, f.data_admissao, f.cod_cargo,
       f.salario, f.comissao, f.cod_gerente,
       f.cod_departamento);
```




**Exemplo de Intercalação de Linhas (continuação)**

O exemplo do slide estabelece a correspondência entre `COD_FUNCIONARIO` da tabela `FUNC3` e `COD_FUNCIONARIO` da tabela `FUNCIONARIO`. Caso seja encontrada uma correspondência, a linha da tabela `FUNC3` será atualizada para corresponder à linha da tabela `FUNCIONARIO`. Se não for encontrada, a linha será inserida na tabela `FUNC3`.

A condição `c.cod_funcionario = f.cod_funcionario` será avaliada. Como a tabela `FUNC3` está vazia, a condição retorna `FALSE`, indicando que não há correspondências. A lógica corresponde à cláusula `WHEN NOT MATCHED`, e o comando `MERGE` insere as linhas da tabela `FUNCIONARIO` na tabela `FUNC3`.

Se houver linhas na tabela `FUNC3`, e os códigos dos funcionários corresponderem nas duas tabelas (`FUNC3` e `FUNCIONARIO`), as linhas existentes na tabela `FUNC3` serão atualizadas para corresponderem à tabela `FUNCIONARIO`.

# Exercício 13



13-31

## Exercício 13

1. Execute o comando em `sol_13_01.sql` para criar a tabela `HISTORICO_SAL`.
2. Exiba a estrutura da tabela `HISTORICO_SAL`.

| Table | Column | Tipo De Dados | Tamanho | Precisão | Escala | Chave Primária | Anulável |
|---|---|---|---|---|---|---|---|
| HISTORICO_SAL | COD_FUNCIONARIO | Number | - | 6 | 0 | - | ✓ |
| | DATA_ADMISSAO | Date | 7 | - | - | - | ✓ |
| | SALARIO | Number | - | 8 | 2 | - | ✓ |

3. Execute o comando em `sol_13_03.sql` para criar a tabela `HISTORICO_GER`.
4. Exiba a estrutura da tabela `HISTORICO_GER`.

| Table | Column | Tipo De Dados | Tamanho | Precisão | Escala | Chave Primária | Anulável |
|---|---|---|---|---|---|---|---|
| HISTORICO_GER | COD_FUNCIONARIO | Number | - | 6 | 0 | - | ✓ |
| | COD_GERENTE | Number | - | 6 | 0 | - | ✓ |
| | SALARIO | Number | - | 8 | 2 | - | ✓ |

5. Execute o comando em `lab_03_05.sql` da pasta lab para criar a tabela `SAL_ESPECIAL`.
6. Exiba a estrutura da tabela `SAL_ESPECIAL`.

| Table | Column | Tipo De Dados | Tamanho | Precisão | Escala | Chave Primária | Anulável |
|---|---|---|---|---|---|---|---|
| SAL_ESPECIAL | COD_FUNCIONARIO | Number | - | 6 | 0 | - | ✓ |
| | SALARIO | Number | - | 8 | 2 | - | ✓ |

7. a) Crie uma consulta que faça o seguinte:

   Recupere na tabela `FUNCIONARIO` os detalhes de código do funcionário, data de admissão, salário e o código do gerente desses funcionários cujo código é inferior a 125.

   Se o salário for superior a R$20.000, insira os detalhes sobre o código do funcionário e o salário na tabela `SAL_ESPECIAL`.

   Insira o código do funcionário, a data de admissão e o salário na tabela `HISTORICO_SAL`.

   Insira os detalhes sobre o código do funcionário, o código do gerente e o salário na tabela `HISTORICO_GER`.

**Exercício 13 (continuação)**

b) Exiba os registros da tabela `SAL_ESPECIAL`.

| COD_FUNCIONARIO | SALARIO |
|---|---|
| 100 | 24000 |

c) Exiba os registros da tabela `HISTORICO_SAL`.

| COD_FUNCIONARIO | DATA_ADMISSAO | SALARIO |
|---|---|---|
| 101 | 21/09/99 | 17000 |
| 102 | 13/01/03 | 17000 |
| 103 | 03/01/00 | 9000 |
| 104 | 21/05/01 | 6000 |
| 107 | 07/02/09 | 4200 |
| 124 | 16/11/08 | 5800 |

d) Exiba os registros da tabela `HISTORICO_GER`.

| COD_FUNCIONARIO | COD_GERENTE | SALARIO |
|---|---|---|
| 101 | 100 | 17000 |
| 102 | 100 | 17000 |
| 103 | 102 | 9000 |
| 104 | 103 | 6000 |
| 107 | 103 | 4200 |
| 124 | 100 | 5800 |

8. a) Execute o comando em `sol_13_08a.sql` para criar a tabela `FONTE_DADOS_VENDA`.

b) Execute o comando em `sol_13_08b.sql` para inserir registros na tabela `FONTE_DADOS_VENDA`.

**Exercício 13 (continuação)**

c) Exiba a estrutura da tabela `FONTE_DADOS_VENDA`.

| Table | Column | Tipo De Dados | Tamanho | Precisão | Escala | Chave Primária | Anulável |
|---|---|---|---|---|---|---|---|
| FONTE_DADOS_VENDA | COD_FUNCIONARIO | Number | - | 6 | 0 | - | ✓ |
| | COD_SEMANA | Number | - | 2 | 0 | - | ✓ |
| | VENDAS_SEG | Number | - | 8 | 2 | - | ✓ |
| | VENDAS_TER | Number | - | 8 | 2 | - | ✓ |
| | VENDAS_QUA | Number | - | 8 | 2 | - | ✓ |
| | VENDAS_QUI | Number | - | 8 | 2 | - | ✓ |
| | VENDAS_SEX | Number | - | 8 | 2 | - | ✓ |

d) Exiba os registros da tabela `FONTE_DADOS_VENDA`.

| COD_FUNCIONARIO | COD_SEMANA | VENDAS_SEG | VENDAS_TER | VENDAS_QUA | VENDAS_QUI | VENDAS_SEX |
|---|---|---|---|---|---|---|
| 178 | 6 | 1750 | 2200 | 1500 | 1500 | 3000 |

e) Execute o comando em `sol_13_08e.sql` para criar a tabela `VENDAS_INFO`.

f) Exiba a estrutura da tabela `VENDAS_INFO`.

| Table | Column | Tipo De Dados | Tamanho | Precisão | Escala | Chave Primária | Anulável |
|---|---|---|---|---|---|---|---|
| VENDAS_INFO | COD_FUNCIONARIO | Number | - | 6 | 0 | - | ✓ |
| | SEMANA | Number | - | 2 | 0 | - | ✓ |
| | VENDAS | Number | - | 8 | 2 | - | ✓ |

g) Crie uma consulta que faça o seguinte:

Recupere da tabela `FONTE_DADOS_VENDA` os detalhes sobre o código do funcionário, o código da semana, vendas na segunda-feira, vendas na terça-feira, vendas na quarta-feira, vendas na quinta-feira e vendas na sexta-feira.

Crie uma transformação de modo que cada registro recuperado da tabela `FONTE_DADOS_VENDA` seja convertido em vários registros para a tabela `VENDAS_INFO`.

h) Exiba os registros da tabela `VENDAS_INFO`.

| COD_FUNCIONARIO | SEMANA | VENDAS |
|---|---|---|
| 178 | 6 | 1750 |
| 178 | 6 | 2200 |
| 178 | 6 | 1500 |
| 178 | 6 | 1500 |
| 178 | 6 | 3000 |