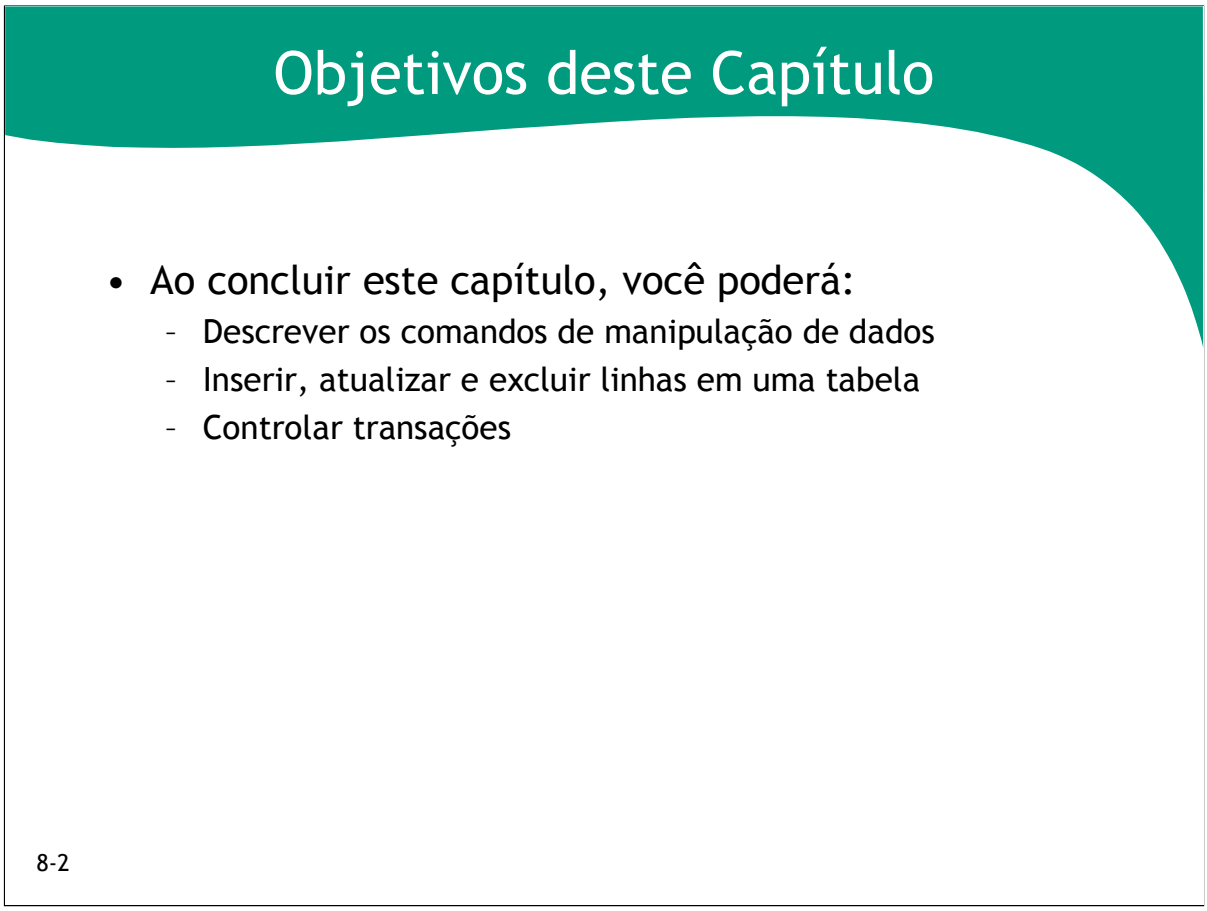
8

Manipulação de Dados



Objetivos deste Capítulo

Neste capítulo, você verá como usar comandos de manipulação de dados para inserir, atualizar e excluir linhas em uma tabela. Você também aprenderá a controlar transações com as instruções `COMMIT`, `SAVEPOINT` e `ROLLBACK`.

Linguagem de Manipulação de Dados

- Os comandos de manipulação de dados são conhecidos pela sigla DML (Data Manipulation Language)
- Um comando DML é executado quando você adiciona, modifica ou remove linhas de uma tabela
- Uma transação consiste em um conjunto de comandos DML que formam uma unidade lógica de trabalho.

8-3

Linguagem de Manipulação de Dados

Os comandos de manipulação de dados são parte essencial do SQL, eles são conhecidos pela sigla DML (Data Manipulation Language). Para adicionar, atualizar ou excluir dados no banco de dados, você deve executar um comando DML.

Um conjunto de comandos DML que formam uma unidade lógica de trabalho é chamado de *transação*.

Considere um banco de dados de uma instituição bancária. Quando um cliente do banco transfere dinheiro da poupança para a conta corrente, a transação pode consistir em três operações distintas: diminuição da poupança, aumento da conta corrente e registro da transação no diário de transações. O Oracle deve garantir a execução de todos os três comandos SQL para manter as contas com os saldos corretos. Quando algo impedir a execução de um dos comandos da transação, será necessário desfazer todas as outras.

Comando INSERT

- Você adiciona novas linhas a uma tabela com o comando INSERT:

```
INSERT INTO tabela [(coluna [, coluna...])]
VALUES             (valor  [, valor...]);
```

- A sintaxe acima insere apenas uma linha por vez.

8-4

Comando INSERT

O comando INSERT é usado para inserir novas linhas a uma tabela.

Na sintaxe:

tabela	é o nome da tabela
coluna	é o nome da coluna da tabela a ser preenchida
valor	é o valor correspondente da coluna

O comando INSERT quando usado com a cláusula VALUES, como na sintaxe do slide, insere somente uma linha por vez a uma tabela.

Comando INSERT

- Se você especificar um valor para cada coluna na ordem padrão das colunas, não é necessário especificar a lista de colunas no comando INSERT.
- Você pode listar as colunas na cláusula INSERT e especificar os valores correspondentes dessas colunas.

```
INSERT INTO departamento(cod_departamento,
       nome_departamento, cod_gerente, cod_localidade)
VALUES (70, 'Relações Públicas', 100, 1700);
1 linha criada.
```

- Delimite os valores de caractere e data com aspas simples.

8-5

Comando INSERT (continuação)

Como é possível inserir uma nova linha com valores para cada coluna, a lista de colunas não é necessária na cláusula INSERT. No entanto, se você não usar a lista de colunas, os valores deverão ser listados de acordo com a ordem padrão das colunas na tabela e um valor deverá ser fornecido para cada coluna.

```
   DESCRIBE  departamento
 Nome               Nulo?    Tipo
 ------------------ -------- ----------------
 COD_DEPARTAMENTO   NOT NULL NUMBER(4)
 NOME_DEPARTAMENTO  NOT NULL VARCHAR2(30)
 COD_GERENTE                 NUMBER(6)
 COD_LOCALIDADE              NUMBER(4)
```

Para manter a clareza no comando, use a lista de colunas na cláusula INSERT.

Delimite os valores de caractere e data com aspas simples.

Os valores numéricos não devem ser delimitados com aspas simples, pois ocorrerá conversão implícita dos valores especificados para as colunas para o tipo de dados NUMBER.

Comando INSERT com Valores Nulos

- Omita a coluna da lista de colunas

```
INSERT INTO   departamento (cod_departamento,
                            nome_departamento )
VALUES        (30, 'Compras');
1 linha criada.
```

ou

- Especifique a palavra-chave NULL na cláusula VALUES.

```
INSERT INTO   departamento
VALUES        (100, 'Finanças', NULL, NULL);
1 linha criada.
```

8-6

Comando INSERT com Valores Nulos

Método	Descrição
Implícito	Omita a coluna da lista de colunas.
Explícito	Especifique a palavra-chave NULL na lista VALUES; especifique a string vazia ('') na lista VALUES para strings de caracteres e datas.

Antes de inserir um valor NULL para uma coluna, verifique se a coluna permite valores nulos (DESCRIBE do SQL*Plus).

O Oracle impõe os tipos de dados, faixas de dados e constraints de integridade de dados. Todas as colunas não listadas explicitamente obtêm um valor nulo na nova linha.

Erros comuns que podem ocorrer durante uma inserção:

- Valor obrigatório ausente para uma coluna NOT NULL
- Constraint de unicidade violada por valor duplicado
- Constraint de chave estrangeira violada
- Constraint CHECK violada
- Incompatibilidade de tipo de dados
- Valor muito extenso para caber na coluna

Comando INSERT com Valores Especiais

- A função SYSDATE registra a data e o horário atuais.

```
INSERT INTO funcionario (cod_funcionario,
                         nome, sobrenome,
                         email, telefone,
                         data_admissao, cod_cargo, salario,
                         comissao, cod_gerente,
                         cod_departamento)
VALUES                   (113,
                         'Luis', 'de Carvalho',
                         'LCARVALHO', '051.124.4567',
                         SYSDATE, 'AD_ASST', 6900,
                         NULL, 205, 100);
1 linha criada.
```

8-7

Comando INSERT com Valores Especiais

É possível usar funções para especificar valores especiais em uma tabela.

O exemplo do slide registra informações sobre o funcionário Luis na tabela FUNCIONARIO. São fornecidos a data e o horário atuais na coluna DATA_ADMISSAO. A função SYSDATE é usada para obter a data e o horário atuais.

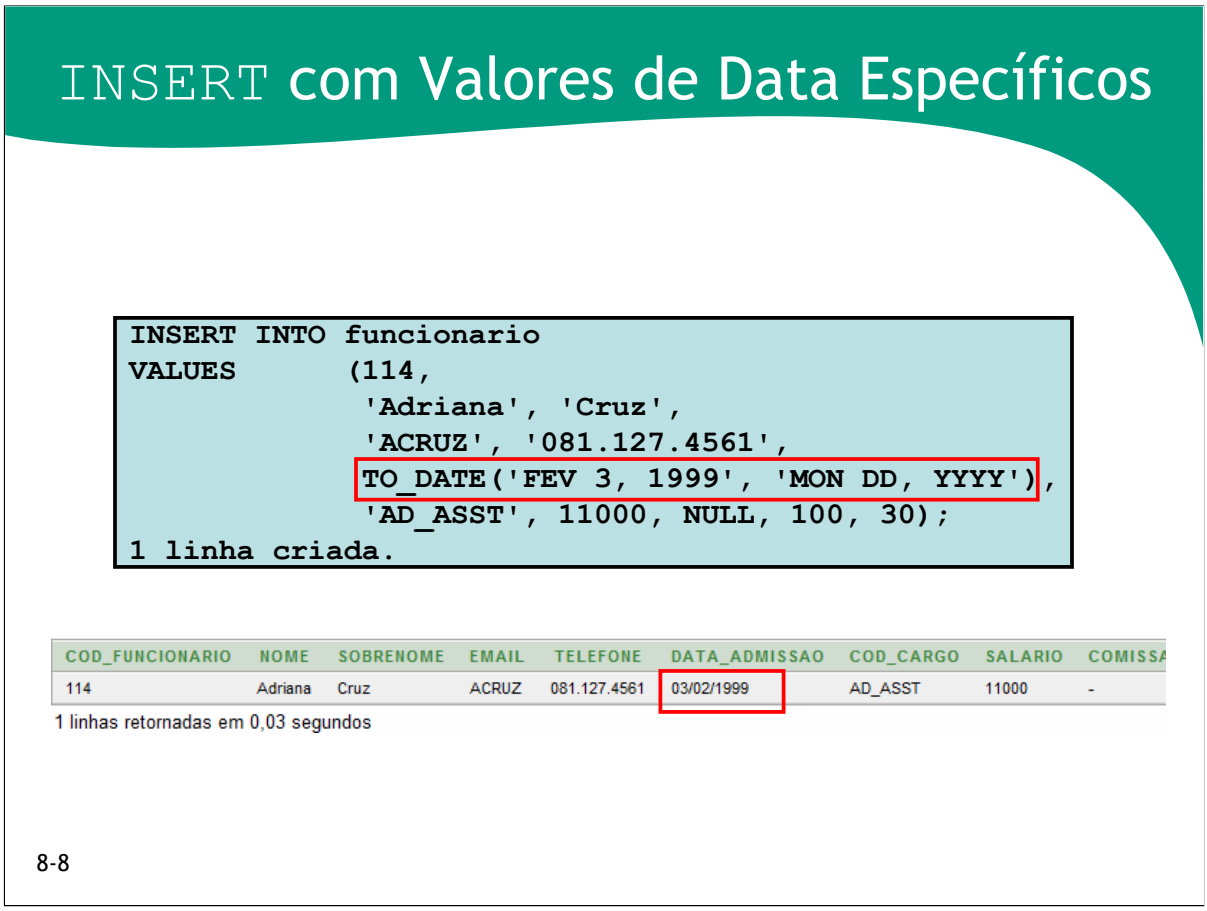
Você também pode usar a função USER ao inserir linhas em uma tabela. Essa função registra o nome do usuário atual.

Confirmando Inserções à Tabela

```
SELECT cod_funcionario, sobrenome, cod_cargo,
data_admissao, comissao
FROM   funcionario
WHERE  cod_funcionario = 113;
```

COD_FUNCIONARIO	SOBRENOME	COD_CARGO	DATA_ADMISSAO	COMISSAO
113	de Carvalho	AD_ASST	10/05/09	-

1 linhas retornadas em 0,05 segundos



Comando INSERT com Valores de Data Específicos

O formato DD-MON-YY é normalmente usado para inserir um valor de data. Com esse formato, lembre-se de que o século usado como padrão é o atual. Como a data também contém informações sobre horário, o horário padrão é meia-noite (00:00:00).

Se for necessário informar uma data em um formato diferente do padrão (por exemplo, com outro século ou um horário específico), use a função TO_DATE.

O exemplo do slide registra informações sobre a funcionária Adriana na tabela FUNCIONARIO. Ele define a coluna DATA_ADMISSAO como Fevereiro 3, 2009. Se você usar o comando a seguir em vez do mostrado no slide, o ano de admissão será interpretado como 2099.

```
INSERT INTO funcionario
VALUES      (114,
             'Adriana', 'Cruz',
             'ACRUA', '081.127.4561',
             TO_DATE('03-FEV-99','DD-MON-YY'),
             'AD_ASST', 11000, NULL, 100, 30);
```

Se o formato RR for usado, o sistema fornecerá o século correto automaticamente, mesmo que não seja o atual.

Criação de Scripts

- Use a variável de substituição `&` em um comando SQL para solicitar valores de entrada na execução do script.

```
INSERT INTO departamento
           (cod_departamento, nome_departamento, cod_localidade)
VALUES     (&cod_departamento, '&nome_departamento',&localidade);

Informe o valor para cod_departamento: 40
Informe o valor para nome_departamento: Recursos Humanos
Informe o valor para localidade: 2500
antigo   3: VALUES     (&cod_departamento, '&nome_departamento',
                                                    &localidade)
novo   3: VALUES     (40, 'Recursos Humanos',2500)

1 linha criada.
```

8-9

Criação de Scripts para Manipular Dados

É possível salvar comandos com variáveis de substituição em um arquivo e executar os comandos no arquivo. O exemplo do slide registra informações de um departamento na tabela `DEPARTAMENTO`.

Ao executar o arquivo de script, você será solicitado a especificar informações para as variáveis de substituição (identificadores iniciados por `&`). Os valores especificados serão aplicados ao comando. Isso permite executar várias vezes o mesmo arquivo de script, mas fornecer um conjunto de valores diferente a cada execução.

Copia de Linhas de Outra Tabela

- Você pode consultar várias linhas de outras tabelas e usar essa consulta num comando INSERT para inserir o resultado em outra tabela.

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

4 linhas criadas.
```

- Com essa sintaxe, não use a cláusula VALUES.
- Estabeleça uma correspondência entre o número de colunas na cláusula INSERT e o número de colunas na subconsulta.

8-10

Copia de Linhas de Outra Tabela

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

Sintaxe

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

onde:

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 inserção

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 copia_func
   SELECT *
   FROM   funcionario;
```

Comando UPDATE

- Você altera linhas existentes em uma tabela com o comando UPDATE:

```
UPDATE      tabela
SET         coluna = valor [, coluna = valor, ...]
[WHERE      condição];
```

- Se for necessário, você pode atualizar mais de uma linha por vez.

8-11

Comando UPDATE

É possível modificar linhas existentes em uma tabela com o comando UPDATE.

Na sintaxe:

tabela	é o nome da tabela
coluna	é o nome da coluna da tabela a ser preenchida
valor	é o valor correspondente ou a subconsulta da coluna
condição	identifica as linhas a serem atualizadas e é composta de nomes de colunas, expressões, constantes, subconsultas e operadores de comparação

Para confirmar a operação de atualização, consulte a tabela para exibir as linhas atualizadas.

Em geral, usa-se a chave primária para identificar uma única linha. O uso de outras colunas pode resultar na atualização inesperada de várias linhas. Por exemplo, identificar uma única linha da tabela FUNCIONARIO pelo nome é perigoso, pois mais de um funcionário pode ter o mesmo nome.

Comando UPDATE

- Quando a cláusula WHERE é especificada, uma ou mais linhas são modificadas:

```
UPDATE funcionario
SET    cod_departamento = 70
WHERE  cod_funcionario = 113;
1 linha atualizada.
```

- A omissão da cláusula WHERE modifica todas as linhas da tabela:

```
UPDATE   copia_func
SET      cod_departamento = 110;
22 linhas atualizadas.
```

8-12

Comando UPDATE (continuação)

O comando UPDATE modifica uma ou mais linhas quando a cláusula WHERE é especificada. O exemplo do slide transfere o funcionário 113 (Luis) para o departamento 70.

Se você omitir a cláusula WHERE, todas as linhas da tabela serão modificadas.

```
SELECT sobrenome, cod_departamento
FROM   copia_func;
```

SOBRENOME	COD_DEPARTAMENTO
Carlos	110
Martins	110
da Silva	110
Honorato	110
Almeida	110
Nascimento	110
de Carvalho	110
Cruz	110

22 linhas retornadas em 0,00 segundos

Comando UPDATE com Subconsulta

- Altere o cargo e o salário do funcionário 114 para que correspondam aos 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.
```

8-13

Comando UPDATE com 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];
```

Se nenhuma linha for atualizada, a mensagem "0 linhas atualizadas." será exibida.

Atualização com Base em Outra Tabela

- Você pode usar subconsultas em comandos `UPDATE` para atualizar as linhas de uma tabela com base nos valores de outra tabela:

```
UPDATE   copia_func
SET      cod_departamento  =  (SELECT cod_departamento
                              FROM funcionario
                              WHERE cod_funcionario = 100)
WHERE    cod_cargo         =  (SELECT cod_cargo
                              FROM funcionario
                              WHERE cod_funcionario = 200);
3 linhas atualizadas.
```

8-14

Atualização de 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 `COPIA_FUNC` com base nos valores da tabela `FUNCIONARIO`. Ele altera o número do departamento de todos os funcionários com o código do cargo do funcionário 200 para o número do departamento atual do funcionário 100.

Comando DELETE

- Para excluir linhas existentes de uma tabela, use o comando DELETE:

```
DELETE [FROM]   tabela
[WHERE          condição];
```

8-15

Comando DELETE

É possível remover linhas existentes de uma tabela com o comando DELETE.

Na sintaxe:

tabela	é o nome da tabela
condição	identifica as linhas a serem deletadas e é composta de nomes de colunas, expressões, constantes, subconsultas e operadores de comparação

Se nenhuma linha for removida, a mensagem "0 linhas deletadas." será exibida.

Comando DELETE

- Quando a cláusula WHERE é especificada, uma ou mais linhas são removidas:

```
DELETE FROM departamento
WHERE  nome_departamento = 'Finanças';
1 linha deletada.
```

- A omissão da cláusula WHERE remove todas as linhas da tabela!

```
DELETE FROM  copia_func;
22 linhas deletadas.
```

8-16

Comando DELETE (continuação)

É possível excluir linhas específicas usando a cláusula WHERE no comando DELETE. O exemplo do slide exclui o departamento Finanças da tabela DEPARTAMENTO. Para confirmar a operação de exclusão, exiba as linhas com o comando SELECT.

```
SELECT *
FROM   departamento
WHERE  nome_departamento = 'Finanças';
não há linhas selecionadas.
```

Se você omitir a cláusula WHERE, todas as linhas da tabela serão excluídas! O segundo exemplo do slide exclui todas as linhas da tabela COPIA_FUNC porque nenhuma cláusula WHERE foi especificada.

Exemplos:

```
DELETE FROM  funcionario WHERE cod_funcionario = 114;
1 linha deletada.

DELETE FROM  departamento WHERE cod_departamento IN (30,
40);
2 linhas deletadas.
```

Exclusão com Base em Outra Tabela

- Você pode usar subconsultas em comandos `DELETE` para remover as linhas de uma tabela com base nos valores de outra tabela:

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

8-17

Exclusão de Linhas com Base em Outra Tabela

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

Comando TRUNCATE

- Remove todas as linhas de uma tabela
- É um comando DDL (Data Definition Language), e não DML; não pode ser desfeita facilmente
- Sintaxe:

```
TRUNCATE TABLE nome_da_tabela;
```

- Exemplo:

```
TRUNCATE TABLE copia_func;
```

8-18

Comando TRUNCATE

Um método mais eficiente de esvaziar uma tabela é a utilização do comando TRUNCATE.
É possível usar esse comando para remover rapidamente todas as linhas de uma tabela. A remoção de linhas com o comando TRUNCATE é mais rápida do que com o comando DELETE pelos seguintes motivos:

- O comando TRUNCATE é um comando DDL (Data Definition Language) e não gera informações de rollback.
- Truncar uma tabela não dispara as triggers de deleção dessa tabela.
- Se a tabela for pai de uma constraint de integridade referencial, você não poderá truncá-la. É necessário desativar a constraint antes de executar o comando TRUNCATE. A desativação de constraints é abordada em um capítulo posterior.

Controle de Transações

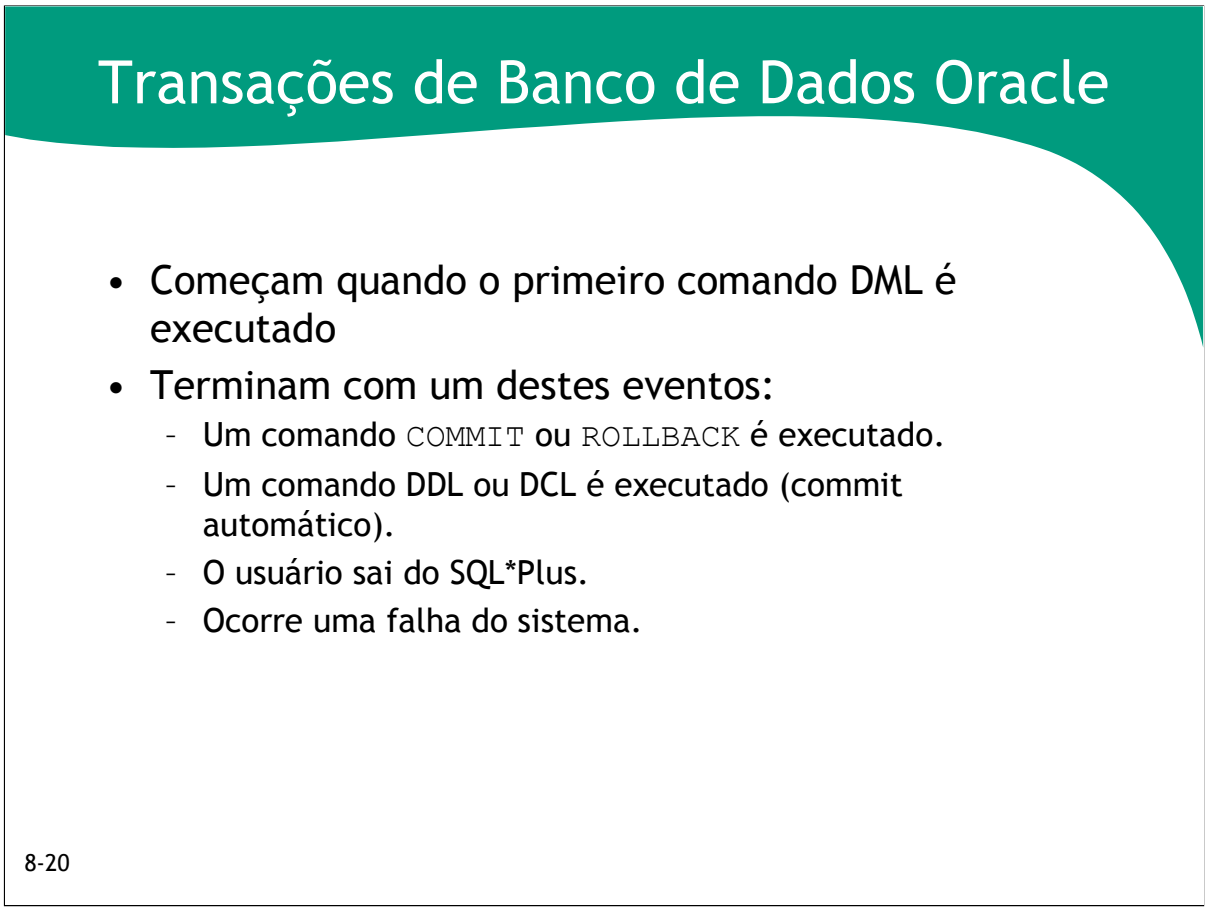
- Uma transação de banco dados é uma unidade lógica de trabalho.
- Uma transação é um conjunto de alterações (DML) que serem todas efetivadas ou todas canceladas.
- O banco de dados Oracle provê o controle de transações.

8-19

Controle de Transações de Banco de Dados

O Oracle garante a consistência de dados com base em transações. As transações permitem mais flexibilidade e controle durante a alteração de dados e garantem a consistência de dados em caso de falha de processo do usuário ou falha do sistema.

As transações consistem em comandos DML que formam uma alteração consistente dos dados. Por exemplo, uma transferência de fundos entre duas contas deve incluir o débito em uma conta e o crédito em outra conta no mesmo valor. As duas ações deverão apresentar falha ou ser bem-sucedidas; o crédito não deverá ser efetivado sem que o débito também o seja



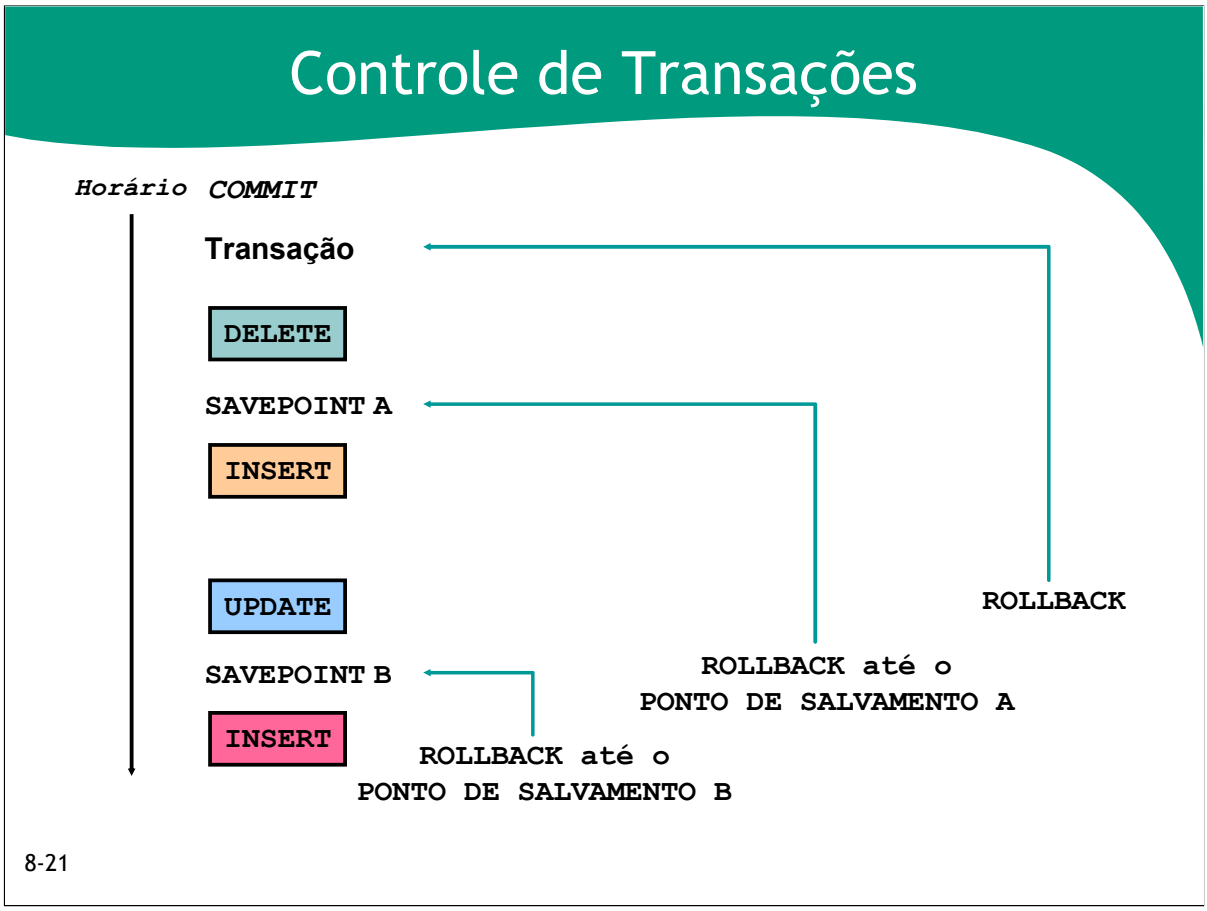
Transações de Banco de Dados Oracle

Uma transação começa quando o primeiro comando DML é encontrado e termina quando uma destas ações ocorre:

- Um comando `COMMIT` ou `ROLLBACK` é executado.
- Um comando `DDL`, como `CREATE`, é executado.
- Um comando `DCL` é executado.
- O usuário sai do SQL*Plus.
- Ocorre uma falha de máquina ou do sistema.

Após o término de uma transação, o próximo comando DML inicia automaticamente a transação seguinte.

Um comando DDL ou DCL é submetido a commit automaticamente ao seu final e, portanto, encerra uma transação de forma implícita.



Comandos de Controle de Transação

É possível controlar a lógica de transações com os comandos `COMMIT`, `SAVEPOINT` e `ROLLBACK`.

Comando	Descrição
`COMMIT`	Termina a transação atual, tornando permanentes todas as alterações de dados pendentes
`SAVEPOINT` *nome*	Marca um ponto de salvamento na transação atual
`ROLLBACK`	ROLLBACK termina a transação atual e descarta todas as alterações de dados pendentes.
`ROLLBACK TO` *SAVEPOINT nome*	ROLLBACK TO SAVEPOINT efetua rollback da transação atual para o ponto de salvamento especificado e, como consequência, descarta todas as alterações e/ou pontos de salvamento criados após o ponto de salvamento do rollback atual. Se você omitir a cláusula TO SAVEPOINT, a instrução ROLLBACK efetuará rollback de toda a transação. Como os pontos de salvamento são lógicos, não há como listar os pontos de salvamento criados.

Observação: `SAVEPOINT` não é um comando SQL padrão ANSI.

Rollback até um Marcador

- Crie um marcador em uma transação atual usando o comando `SAVEPOINT`.
- Faça rollback até esse marcador usando o comando `ROLLBACK TO SAVEPOINT`.

```
UPDATE...
SAVEPOINT ponto_apos_alteracao;
Ponto de salvamento criado.

INSERT...
ROLLBACK TO ponto_apos_alteracao;
Rollback concluído.
```

8-22

Rollback até um Marcador

É possível criar um marcador na transação atual usando o comando `SAVEPOINT`, que divide a transação em seções menores. Depois, é possível descartar as alterações pendentes até esse marcador usando o comando `ROLLBACK TO` ponto de salvamento.

Se você criar um segundo ponto de salvamento com o mesmo nome do ponto de salvamento anterior, o ponto de salvamento anterior será descartado.

Processamento Implícito de Transação

- Um commit automático ocorre nas seguintes circunstâncias:
  - Um comando DDL é executado
  - Um comando DCL é executado
  - Saída normal do SQL*Plus, sem a execução implícita de comandos `COMMIT` ou `ROLLBACK`
- Um rollback automático ocorre em decorrência do encerramento anormal do SQL*Plus ou de uma falha do sistema.

Processamento Implícito de Transação

Status	Circunstâncias
Commit automatico	O comando DDL ou DCL é executado. O SQL*Plus foi encerrado normalmente, sem a execução explícita dos comandos `COMMIT` ou `ROLLBACK`.
Rollback automatic	Encerramento anormal do SQL*Plus ou falha do sistema.

Observação: Um terceiro comando está disponível no SQL*Plus. É possível alternar o comando `AUTOCOMMIT` entre os estados ativado e desativado. Se ele for *ativado*, cada comando DML será submetido a commit logo após sua execução. Não é possível fazer rollback das alterações. Se ele for *desativado*, o comando `COMMIT` ainda poderá ser executado explicitamente. Além disso, o comando `COMMIT` será executado quando um comando DDL for executado ou quando você sair do SQL*Plus.

Processamento Implícito de Transação (continuação)

Falhas do Sistema

Quando uma transação for interrompida por uma falha do sistema, será feito rollback automaticamente de toda a transação. Isso impede que o erro cause alterações indesejadas nos dados e retorna as tabelas para o estado em que se encontravam no momento do último commit. Dessa forma, o Oracle protege a integridade das tabelas.

No SQL*Plus, para sair normalmente da sessão, você deve digitar o comando `EXIT` no prompt. O fechamento da janela é interpretado como uma saída anormal.

Estado dos Dados antes de COMMIT

- É possível recuperar o estado anterior dos dados.
- O usuário atual pode verificar os resultados das operações DML usando o comando SELECT.
- Outros usuários *não podem* visualizar os resultados das operações DML executadas pelo usuário atual.
- As linhas afetadas são *bloqueadas (lock)*; outros usuários não podem alterar os dados nessas linhas.

8-25

Estado dos Dados antes de COMMIT ou ROLLBACK

Todas as alterações de dados feitas durante a transação serão temporárias até que ela seja submetida a commit.

O estado dos dados antes da execução do comando COMMIT ou ROLLBACK pode ser descrito da seguinte forma:

- É possível recuperar o estado anterior dos dados.
- O usuário atual pode visualizar os resultados das operações de manipulação de dados consultando as tabelas.
- Outros usuários não podem visualizar os resultados das operações de manipulação de dados executadas pelo usuário atual. O Oracle institui a consistência de leitura para garantir que cada usuário veja os dados da forma como se encontravam no último commit.
- As linhas afetadas são bloqueadas; outros usuários não podem alterar os dados nessas linhas.

Estado dos Dados após COMMIT

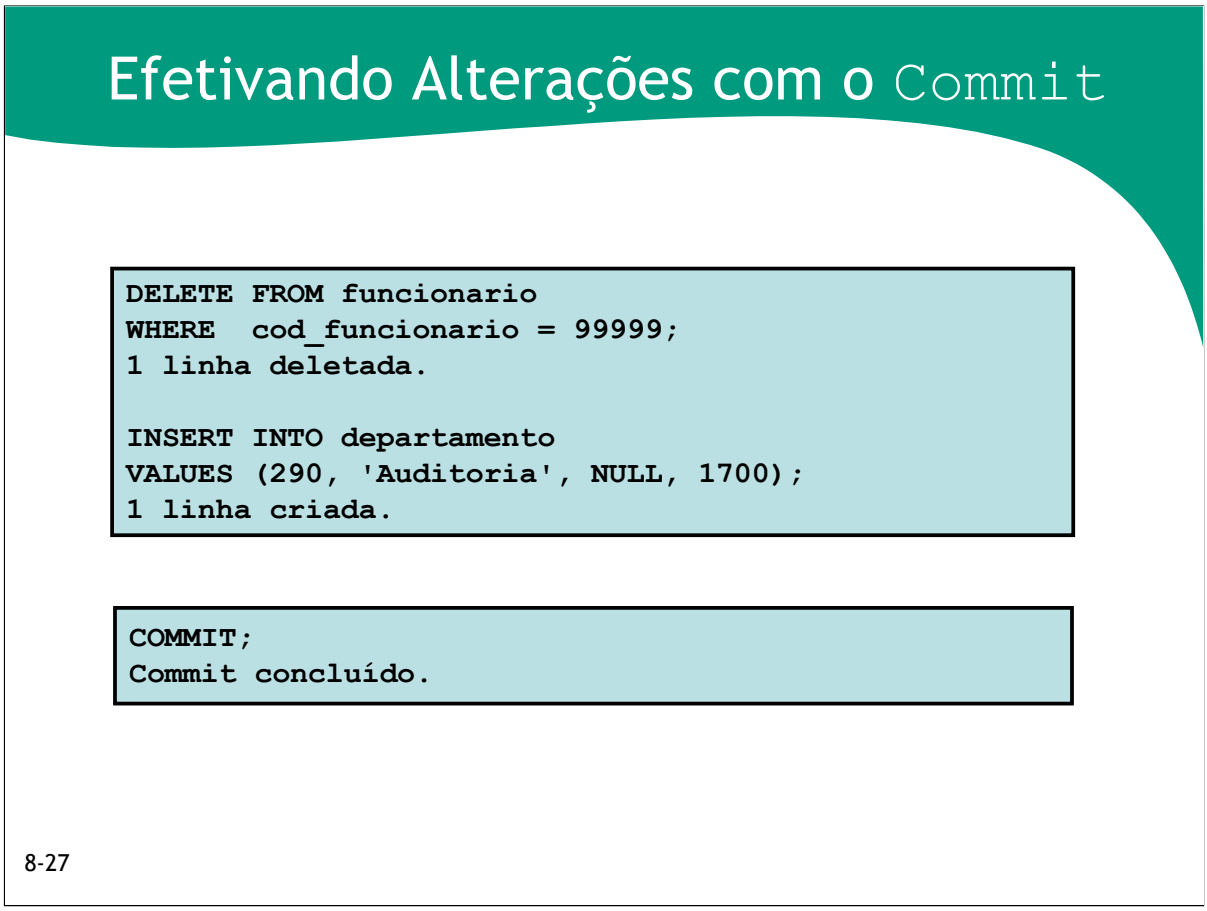
- As alterações de dados tornam-se permanentes no banco de dados.
- O estado anterior dos dados é perdido permanentemente.
- Todos os usuários podem visualizar os resultados.
- Os bloqueios nas linhas afetadas são liberados; essas linhas estão disponíveis para manipulação por outros usuários.
- Todos os pontos de salvamento são apagados.

8-26

Estado dos Dados após COMMIT

Use o comando COMMIT para tornar permanentes todas as alterações pendentes. Veja o que acontece após um comando COMMIT:

- As alterações de dados são gravadas no banco de dados.
- O estado anterior dos dados não está mais disponível nas consultas SQL comuns.
- Todos os usuários podem visualizar os resultados da transação.
- Os bloqueios nas linhas afetadas são liberados; agora, as linhas estão disponíveis para que outros usuários efetuem novas alterações de dados.
- Todos os pontos de salvamento são apagados.



Efetivando Alterações com o `Commit`

O exemplo do slide exclui uma linha da tabela `FUNCIONARIO` e insere uma nova linha na tabela `DEPARTAMENTO`. Em seguida, ele torna a alteração permanente executando o comando `COMMIT`.

Exemplo

Remova os departamentos 290 e 300 da tabela `DEPARTAMENTO` e atualize uma linha da tabela `COPIA_FUNC`. Torne a alteração de dados permanente.

```
DELETE FROM departamento
WHERE  cod_departamento IN (290, 300);
1 linha deletada.

UPDATE funcionario
  SET cod_departamento = 80
  WHERE cod_funcionario = 206;
1 linha atualizada.

COMMIT;
Commit Concluído.
```

Estado dos Dados após ROLLBACK

- Todas as alterações pendentes são descartadas com o comando ROLLBACK:
  - As alterações de dados são desfeitas.
  - O estado anterior dos dados é restaurado.
  - Os bloqueios nas linhas afetadas são liberados.

```
DELETE FROM copia_func;
22 linhas deletadas.
ROLLBACK ;
Rollback concluído.
```

8-28

Estado dos Dados após ROLLBACK

Você descarta todas as alterações pendentes quando usa o comando ROLLBACK.

O resultado será:

- As alterações de dados são desfeitas.
- O estado anterior dos dados é restaurado.
- Os bloqueios nas linhas afetadas são liberados.

Correção de Erros com o ROLLBACK

```
DELETE FROM teste;
25000 linhas deletadas.

ROLLBACK;
Rollback concluído.

DELETE FROM teste WHERE cod = 100;
1 linha deletada.

SELECT * FROM teste WHERE cod = 100;
No rows selected.

COMMIT;
Commit concluído.
```

8-29

Correção de Erros com o ROLLBACK

Ao tentar remover um registro da tabela TESTE, você poderá esvaziar a tabela acidentalmente. É possível corrigir o erro, reexecutar o comando correto e tornar a alteração de dados permanente.

Rollback de Comando

- Se houver falha de um único comando DML durante a execução, será feito rollback somente desso comando.
- O Oracle implementa um ponto de salvamento implícito.
- Todas as outras alterações são retidas.
- O usuário deve encerrar as transações explicitamente executando um comando `COMMIT` ou `ROLLBACK`.

8-30

Rollback de Comando

Parte de uma transação poderá ser descartada por um rollback implícito se for detectado um erro de execução de comando. Se houver falha de um único comando DML durante a execução de uma transação, seu efeito será anulado por um rollback de comando, mas as alterações feitas pelos comandos DML anteriores na transação não serão descartadas. Elas poderão ser submetidas a commit ou rollback explicitamente pelo usuário.

O Oracle executa um commit implícito antes e depois de qualquer comando DDL. Portanto, mesmo que o comando DDL não seja executado com êxito, você não poderá fazer rollback do comando anterior porque o servidor executou um commit.

Encerre as transações explicitamente executando um comando `COMMIT` ou `ROLLBACK`.

Consistência de Leitura

- A consistência de leitura garante uma visão consistente dos dados em todos os momentos.
- As alterações feitas por um usuário não entram em conflito com as alterações feitas por outro usuário.
- A consistência de leitura garante que nos mesmos dados:
  - Os usuários que efetuam operações de leitura não precisem aguardar os usuários que efetuam operações de alteração
  - Os usuários que efetuam operações de alteração não precisem aguardar os usuários que efetuam operações de leitura

8-31

Consistência de Leitura

Os usuários acessam o banco de dados de duas maneiras:

- Operações de leitura (comando `SELECT`)
- Operações de gravação (comando `INSERT`, `UPDATE`, `DELETE`)

A consistência de leitura é necessária para permitir que:

- Os usuários que efetuam operações de leitura e gravação no banco de dados tenham uma visão consistente dos dados.
- Os usuários que efetuam operações de leitura não vejam dados que estão sendo alterados.
- Os usuários que efetuam operações de alteração tenham certeza de que as alterações no banco de dados são feitas de maneira consistente.
- As alterações feitas por usuários que efetuam operações de gravação não interrompam umas às outras nem sejam conflitantes.

O objetivo da consistência de leitura é garantir que cada usuário veja os dados no estado em que se encontravam no momento do último commit, antes de ser iniciada uma operação DML.

Exercício 8

8-32

Exercício 8

1. Execute o comando no script `sol_08_01.sql` para criar a tabela `FUNCIONARIO_TEMP` a ser usada no exercício.
2. Descreva a estrutura da tabela `FUNCIONARIO_TEMP` para identificar os nomes de colunas.

Table	Column	Tipo De Dados	Tamanho	Precisão	Escala	Chave Primária	Anulável
FUNCIONARIO_TEMP	CODIGO	Number	-	4	0	-	-
	NOME	Varchar2	25	-	-	-	✓
	SOBRENOME	Varchar2	25	-	-	-	✓
	EMAIL	Varchar2	10	-	-	-	✓
	SALARIO	Number	-	9	2	-	✓

3. Crie um comando `INSERT` para *adicionar a primeira linha* de dados à tabela `FUNCIONARIO_TEMP` usando estes dados de amostra. Não liste as colunas na cláusula `INSERT`. *Não informe todas as linhas ainda*.

CODIGO	NOME	SOBRENOME	EMAIL	SALARIO
1	Ralf	Patrício	rpatricio	895
2	Beatriz	Dantas	bdantas	860
3	Benito	Santos	bsantos	1100
4	Cello	Newman	cnewman	750
5	Alma	Becker	abecker	1550

4. Preencha a tabela `FUNCIONARIO_TEMP` com a segunda linha de dados de amostra da lista anterior. Desta vez, liste as colunas explicitamente na cláusula `INSERT`.
5. Confirme a adição à tabela.

CODIGO	NOME	SOBRENOME	EMAIL	SALARIO
1	Ralf	Patrício	rpatricio	895
2	Beatriz	Dantas	bdantas	860

2 linhas retornadas em 0,05 segundos

Exercício 8 (continuação)

6. Crie um comando insert num arquivo de script para carregar linhas na tabela `FUNCIONARIO_TEMP`. O script deve solicitar o código, o nome, sobrenome e salário do funcionário a ser adicionado. Concatene a primeira letra do nome com o sobrenome para gerar o email do funcionário em letras minúsculas. Salve esse script no arquivo `cap_08_06.sql`.
7. Para preencher a tabela com as próximas duas linhas dos dados de exemplo, execute o script `cap_08_06.sql`.
8. Confirme as adições à tabela.

CODIGO	NOME	SOBRENOME	EMAIL	SALARIO
1	Ralf	Patrício	rpatricio	895
2	Beatriz	Dantas	bdantas	860
3	Benito	Santos	bsantos	1100
4	Cello	Newman	cnewman	750

4 linhas retornadas em 0,00 segundos

9. Torne as adições de dados permanentes.
10. Altere o sobrenome do funcionário 3 para Drexler.
11. Altere o salário de todos os funcionários com salário inferior a R$ 900,00 para R$ 1.000,00.
12. Verifique as alterações na tabela.

CODIGO	NOME	SOBRENOME	EMAIL	SALARIO
1	Ralf	Patrício	rpatricio	1000
2	Beatriz	Dantas	bdantas	1000
3	Benito	Drexler	bsantos	1100
4	Cello	Newman	cnewman	1000

4 linhas retornadas em 0,00 segundos

13. Exclua Beatriz Dantas da tabela `FUNCIONARIO_TEMP`.
14. Confirme as alterações na tabela.

CODIGO	NOME	SOBRENOME	EMAIL	SALARIO
1	Ralf	Patrício	rpatricio	1000
3	Benito	Drexler	bsantos	1100
4	Cello	Newman	cnewman	1000

3 linhas retornadas em 0,00 segundos

Exercício 8 (continuação)

15. Submeta todas as alterações pendentes a commit.
16. Preencha a tabela com a última linha dos dados de amostra usando o script `cap_08_06.sql`.
17. Confirme a adição à tabela.

CODIGO	NOME	SOBRENOME	EMAIL	SALARIO
1	Ralf	Patrício	rpatricio	1000
3	Benito	Drexler	bsantos	1100
4	Cello	Newman	cnewman	1000
5	Alma	Becker	abecker	1550

4 linhas retornadas em 0,00 segundos

18. Marque um ponto intermediário no processamento da transação.
19. Esvazie a tabela inteira.
20. Confirme se a tabela está vazia.
21. Descarte a operação `DELETE` mais recente sem descartar a operação `INSERT` anterior.
22. Confirme se a nova linha permanece intacta.

CODIGO	NOME	SOBRENOME	EMAIL	SALARIO
1	Ralf	Patrício	rpatricio	1000
3	Benito	Drexler	bsantos	1100
4	Cello	Newman	cnewman	1000
5	Alma	Becker	abecker	1550

4 linhas retornadas em 0,00 segundos

23. Torne a adição de dados permanente.