FIAP GRADUAÇÃO

FIAP

Tecnologia em Análise e Desenvolvimento de Sistemas

Application Development For Databases

PROF. MILTON

Versão 1 – <fevereiro de 24>

FIAP

Criando Funções

Objetivos

Ao concluir esta lição, você será capaz de:

- Fazer a distinção entre procedures e funções
- Descrever os usos de funções
- Criar funções armazenadas
- Chamar uma função
- Remover uma função

4

Objetivo da Lição

Nesta lição, você aprenderá a criar, chamar e manter funções.

Agenda de Lições

- Trabalhando com funções:
  - Diferenciando entre procedures e funções
  - Descrevendo os usos de funções
  - Criando, chamando e removendo funções armazenadas

5

Visão Geral de Funções Armazenadas

Uma função:

- É um bloco PL/SQL nomeado que retorna um valor
- Pode ser armazenada no banco de dados como um objeto de esquema para execuções repetidas
- É chamada como parte de uma expressão ou usada para fornecer um valor de parâmetro para outro subprograma
- Pode ser agrupada em pacotes PL/SQL

6

Visão Geral de Funções Armazenadas

Uma função é um bloco PL/SQL nomeado que pode aceitar parâmetros, ser chamado e retornar um valor. Em geral, as funções são usadas para calcular um valor. As funções e os procedures têm estruturas semelhantes. As funções devem retornar um valor para o ambiente de chamada, enquanto os procedures retornam zero ou mais valores para esse ambiente. Assim como os procedures, as funções têm um cabeçalho, uma seção declarativa, uma seção executável e uma seção opcional de tratamento de exceções. Elas também devem ter uma cláusula `RETURN` no cabeçalho e, no mínimo, uma instrução `RETURN` na seção executável.

As funções podem ser armazenadas no banco de dados como objetos de esquema para execuções repetidas. Essas funções são denominadas "funções armazenadas". Também é possível criar funções nas aplicações do cliente.

As funções promovem a capacidade de manutenção e reutilização. Quando validadas, elas podem ser usadas em várias aplicações. Se as necessidades de processamento mudarem, somente a função precisará ser atualizada.

Além disso, é possível chamar uma função como parte de uma expressão SQL ou de uma expressão PL/SQL. No contexto de uma expressão SQL, as funções devem obedecer a regras específicas para controlar os efeitos colaterais. Em uma expressão PL/SQL, o identificador da função funciona como uma variável cujo valor depende dos parâmetros especificados para ela.

As funções (e procedures) pode ser agrupadas em pacotes PL/SQL. Os pacotes tornam o código ainda mais reutilizável. Eles são abordados nas lições intituladas “Criando Pacotes” e “Trabalhando com Pacotes.”

Criando Funções

O bloco PL/SQL deve ter, no mínimo, uma instrução `RETURN`.

```
CREATE [OR REPLACE] FUNCTION function_name
 [(parameter1 [mode1] datatype1, . . .)]
RETURN datatype IS|AS
 [local_variable_declarations;
  . . .]
BEGIN
  -- actions;
  RETURN expression;
END [function_name];
```

Bloco PL/SQL

7

Sintaxe para a Criação de Funções

Uma função é um bloco PL/SQL que retorna um valor. É necessário fornecer uma instrução `RETURN` para que seja retornado um valor com um tipo de dados consistente com a declaração da função.

Para criar novas funções, use a instrução `CREATE FUNCTION` . Essa instrução pode declarar uma lista de parâmetros, deve retornar um valor e definir as ações a serem executadas pelo bloco PL/SQL padrão.

Considere os seguintes pontos sobre a instrução `CREATE FUNCTION`:

- A opção `REPLACE` indica que, se a função existir, ela será eliminada e substituída pela nova versão criada pela instrução.
- O tipo de dados `RETURN` não deve incluir uma especificação de tamanho.
- O bloco PL/SQL começa com uma instrução `BEGIN` após a declaração de qualquer variável local e termina com uma instrução `END`, seguida opcionalmente de *`function_name`*.
- Deve haver no mínimo uma instrução `RETURN` *`expression`*.
- Não é possível fazer referência a variáveis de host ou de bind no bloco PL/SQL de uma função armazenada.

Observação: Embora seja possível usar os modos de parâmetro `OUT` e `IN OUT` com funções, essa não é uma prática de programação recomendada. Entretanto, caso uma função deva retornar mais de um valor, considere retornar os valores em uma estrutura de dados composta, como um registro ou uma tabela PL/SQL.

FIAP

A Diferença entre Procedures e Funções

Procedures	Funções
São executados como uma instrução PL/SQL	São chamadas como parte de uma expressão
Não contêm a cláusula `RETURN` no cabeçalho	Devem conter uma cláusula `RETURN`no cabeçalho
Podem retornar valores (se houver) em parâmetros de saída	Devem retornar um único valor
Podem conter uma instrução `RETURN` sem um valor	Devem conter pelo menos uma instrução `RETURN`

8

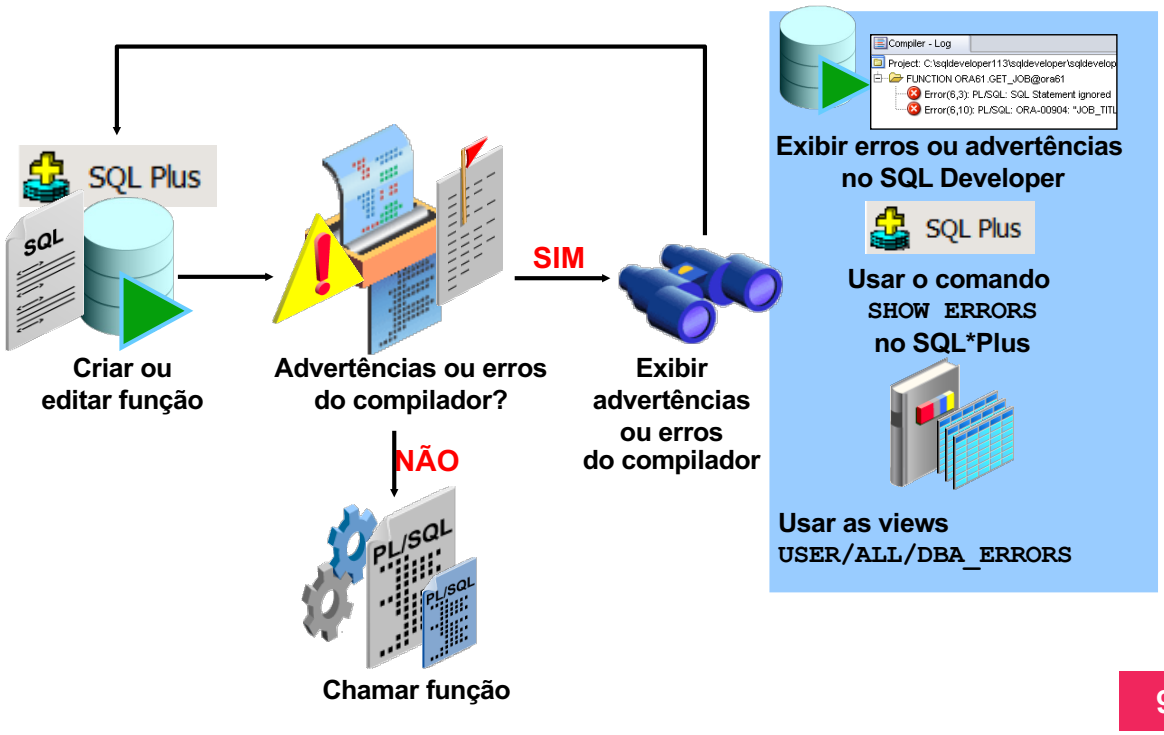
A Diferença entre Procedures e Funções

Os procedures são criados para armazenar uma série de ações para execução posterior. Eles podem conter zero ou mais parâmetros que podem ser transferidos de/para o ambiente de chamada; contudo, eles não precisam retornar um valor. Um procedure pode chamar uma função para ajudar suas ações.

Observação: Um procedure que contém apenas um parâmetro `OUT` seria mais eficiente se fosse reescrito como uma função que retorna um valor.

As funções são criadas quando se deseja calcular um valor que deve ser retornado para o ambiente de chamada. Elas podem conter zero ou mais parâmetros que são transferidos do ambiente de chamada. Em geral, as funções retornam apenas um valor, por meio de uma instrução `RETURN`. As funções usadas em instruções SQL não devem usar parâmetros `OUT` ou `IN OUT`. Embora seja possível usar funções com parâmetros de saída em um bloco ou procedure PL/SQL, elas não podem ser usadas em instruções SQL.

Criando e Executando Funções: Visão Geral



Criando e Executando Funções: Visão Geral

O diagrama no slide ilustra as etapas básicas envolvidas na criação e execução de uma função:

1. Crie a função usando a árvore Object Navigation do SQL Developer ou a área SQL Worksheet.
2. Compile a função. A função é criada no banco de dados. A instrução `CREATE FUNCTION` compila e armazena no banco de dados o código-fonte e o *m-code* compilado. Para compilar a função, clique com o botão direito do mouse no nome da função na árvore Object Navigator e depois clique em Compile.
3. Se houver advertências ou erros de compilação, será possível exibir (e corrigir) as advertências ou erros usando um dos seguintes métodos:
   a. Usando a interface do SQL Developer (Compiler – tab Log)
   b. Usando o comando `SHOW ERRORS` no SQL*Plus
   c. Usando as views `USER/ALL/DBA_ERRORS`
4. Após o término bem-sucedido da compilação, chame a função para retornar o valor desejado.

FIAP

Criando e Chamando uma Função Armazenada Usando a Instrução CREATE FUNCTION: Exemplo

```
CREATE OR REPLACE FUNCTION get_sal
 (p_id employees.employee_id%TYPE) RETURN NUMBER IS
  v_sal employees.salary%TYPE := 0;
  BEGIN
    SELECT salary
    INTO   v_sal
    FROM    employees
    WHERE  employee_id = p_id;
    RETURN v_sal;
  END get_sal;
/
```

```
FUNCTION get_sal Compiled.
```

```
-- Invoke the function as an expression or as
-- a parameter value.

EXECUTE dbms_output.put_line(get_sal(100))
```

```
anonymous block completed
24000
```

10

Função Armazenada: Exemplo

A função `get_sal` é criada com apenas um parâmetro de entrada e retorna o salário como um número. Execute o comando conforme mostrado ou salve-o em um arquivo de script e execute esse script para criar a função `get_sal`.

A função `get_sal` segue uma prática de programação comum, ou seja, ela usa apenas uma instrução `RETURN` para retornar um valor designado a uma variável local. Caso a função tenha uma seção de exceções, ela também poderá conter uma instrução `RETURN`.

Chame uma função como parte de uma expressão PL/SQL porque a função retornará um valor para o ambiente de chamada. A segunda caixa de código usa o comando `EXECUTE` no SQL*Plus para chamar o procedure `DBMS_OUTPUT.PUT_LINE`, cujo argumento é o valor retornado da função `get_sal`. Nesse caso, primeiro é chamada a função `get_sal` para calcular o salário do funcionário com ID 100. O valor do salário retornado é fornecido como o valor do parâmetro `DBMS_OUTPUT.PUT_LINE`, que exibe o resultado (se você tiver executado o comando `SET SERVEROUTPUT ON`).

Observação: A função sempre deve retornar um valor. O exemplo não retorna um valor se não for encontrada uma linha para um determinado `id`. O ideal é criar um handler de exceção para retornar também um valor.

FIAP

Usando Diferentes Métodos para Executar Funções

```
-- As a PL/SQL expression, get the results using host variables

VARIABLE b_salary NUMBER
EXECUTE :b_salary := get_sal(100)
```

```
anonymous block completed
b_salary
-----
24000
```

```
-- As a PL/SQL expression, get the results using a local
-- variable
SET SERVEROUTPUT ON
DECLARE
  sal employees.salary%type;
BEGIN
  sal := get_sal(100);
  DBMS_OUTPUT.PUT_LINE('The salary is: '|| sal);
END;
/
```

```
anonymous block completed
The salary is: 24000
```

11

Usando Diferentes Métodos para Executar Funções

Se forem criadas de forma criteriosa, as funções poderão ser estruturas eficientes. Elas podem ser chamadas das seguintes maneiras:

- **Como parte de expressões PL/SQL:** Você pode usar variáveis de host ou locais para armazenar o valor retornado de uma função. O primeiro exemplo do slide usa uma variável de host, e o segundo usa uma variável local em um bloco anônimo.

Observação: As vantagens e as restrições que se aplicam às funções usadas em instruções SQL são abordadas nas próximas páginas.

Usando Diferentes Métodos para Executar Funções

```
-- Use as a parameter to another subprogram

EXECUTE dbms_output.put_line(get_sal(100))
```

```
anonymous block completed
24000
```

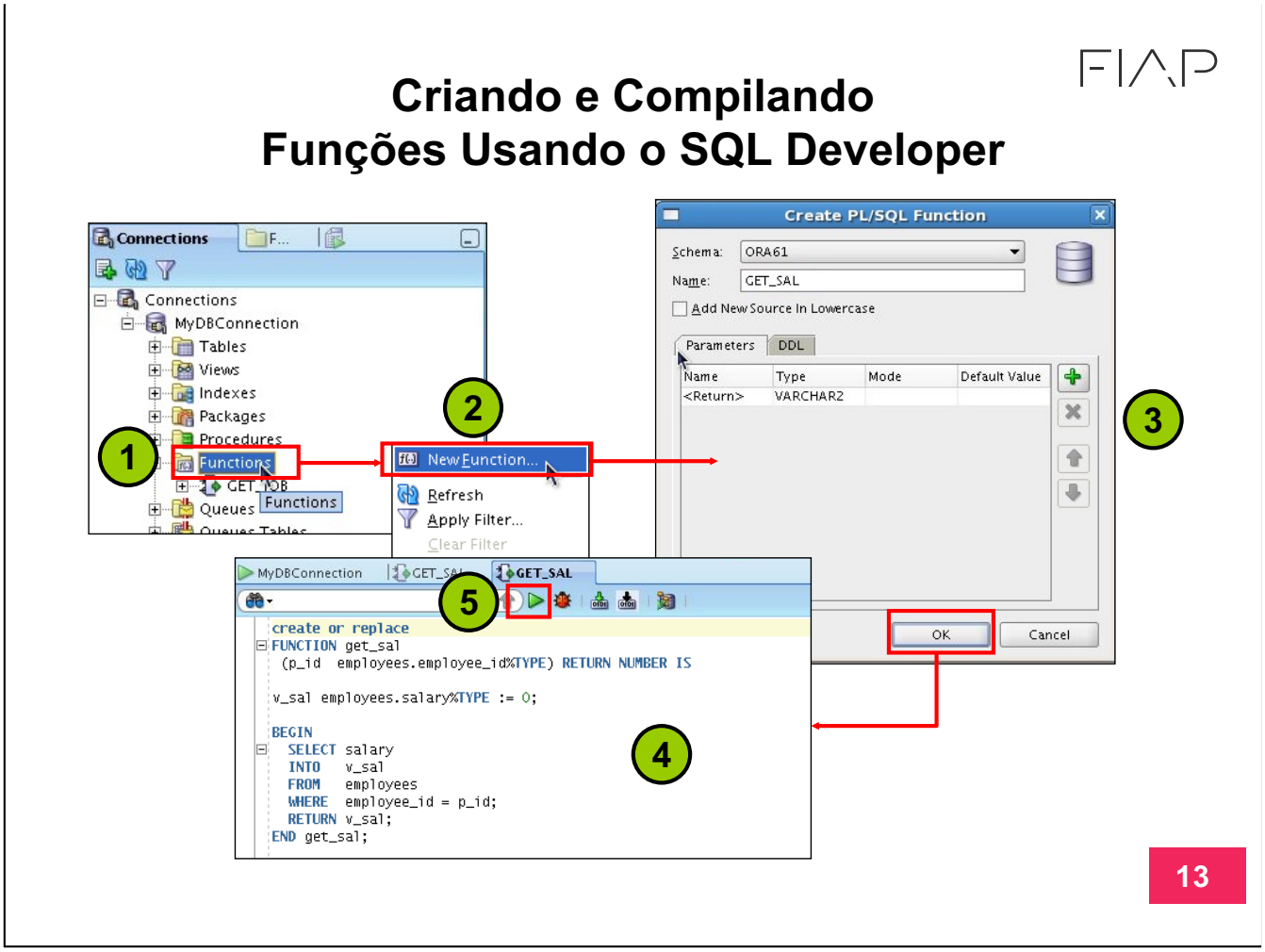
```
-- Use in a SQL statement (subject to restrictions)

SELECT job_id, get_sal(employee_id)
FROM employees;
```

```
JOB_ID     GET_SAL(EMPLOYEE_ID)
---------- --------------------
SH_CLERK   2600
SH_CLERK   2600
AD_ASST    4400
MK_MAN     13000
   . . .
SH_CLERK   3100
SH_CLERK   3000

107 rows selected
```

12

Usando Diferentes Métodos para Executar Funções (continuação)

- **Como parâmetro de outro subprograma:** O primeiro exemplo do slide mostra essa utilização. A função `get_sal` com todos os argumentos é aninhada no parâmetro necessário ao procedure `DBMS_OUTPUT.PUT_LINE`. Isso está relacionado ao conceito de aninhamento de funções, conforme abordado no curso *Oracle Database: Fundamentos de SQL I*.
- **Como uma expressão em uma instrução SQL:** O segundo exemplo no slide mostra como utilizar uma função de apenas uma linha em uma instrução SQL.



Criando e Compilando Funções Usando o SQL Developer

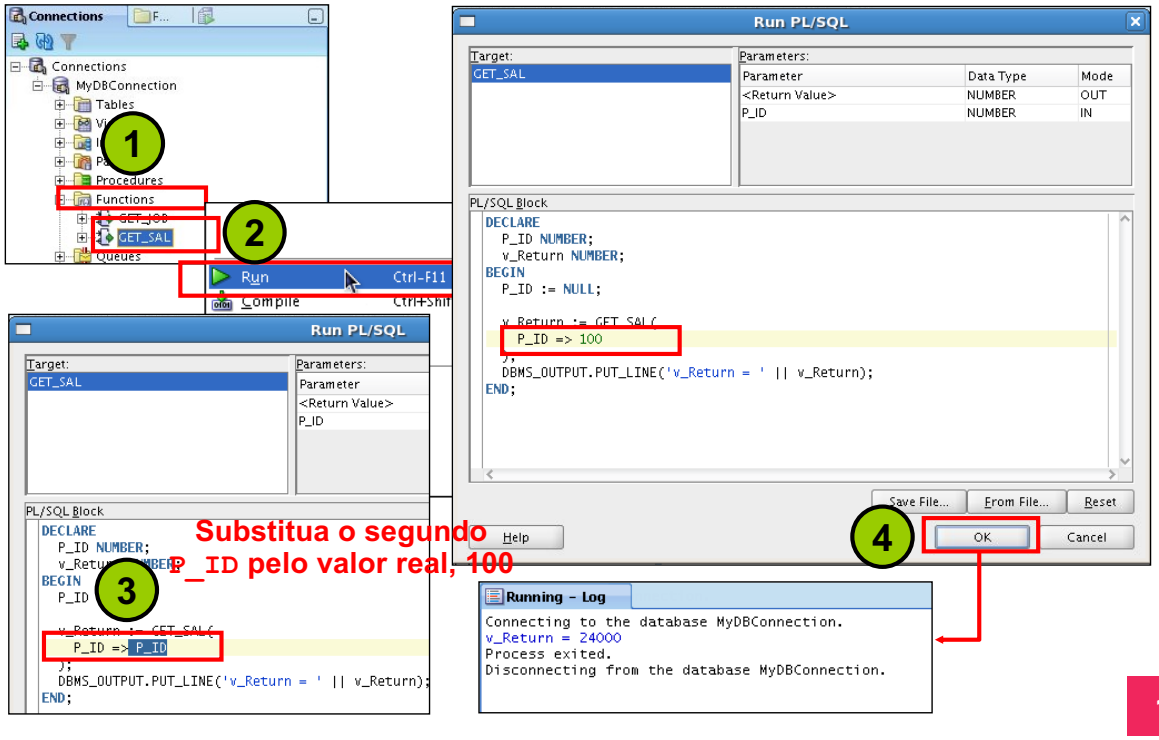
Você pode criar uma nova função no SQL Developer segundo as próximas etapas:

1. Clique com o botão direito do mouse no nó**Functions** .
2. Selecione **New Function** no menu de atalho. A caixa de diálogo **Create PL/SQL Function** é exibida.
3. Selecione o esquema, nome de função e a lista de parâmetros (usando o ícone+), e depois clique em **OK**. O editor de código para a função é exibido.
4. Informe o código da função.
5. Para compilar a função, clique no ícone **Compile**.

Observação

- Para criar uma nova função no SQL Developer, você também pode inserir o código no SQL Worksheet, e depois clicar no ícone Run Script.
- Para obter informações adicionais sobre a criação de funções no SQL Developer, acesse o tópico de ajuda on-line apropriado intitulado “Criar Função ou Procedure de Subprograma PL/SQL.”

Executando Funções Usando o SQL Developer



Executando Funções Usando o SQL Developer

Você pode executar uma função no SQL Developer segundo as próximas etapas:

1. Clique no nó **Functions** .
2. Clique com o botão direito do mouse no nome da função e selecione **Run**. A caixa de diálogo **Run PL/SQL** é exibida.
3. Substitua o segundo nome de parâmetro pelo valor real do parâmetro conforme mostrado no exemplo do slide.
4. Clique em OK.

Observação: Para obter informações adicionais sobre a execução de funções no SQL Developer, acesse o tópico de ajuda on-line intitulado “Executando e Depurando Funções e Procedures.”

FIAP

Vantagens das Funções Definidas pelo Usuário em Instruções SQL

- Ampliam o alcance da linguagem SQL quando as atividades são muito complexas ou não estão disponíveis em SQL
- Aumentam a eficiência quando usadas na cláusula `WHERE` para filtrar os dados, em vez de filtrá-los na aplicação
- Manipulam valores de dados

15

Vantagens das Funções Definidas pelo Usuário em Instruções SQL

As instruções SQL podem fazer referência a funções PL/SQL definidas pelo usuário em qualquer local em que uma expressão SQL seja permitida. Por exemplo, você pode usar uma função definida pelo usuário em qualquer local em que possa ser inserida uma função SQL incorporada, como `UPPER()`.

Vantagens

- Permitem fazer cálculos que são muito complexos ou que não estão disponíveis na linguagem SQL Aumentam a independência dos dados, executando análises complexas no servidor Oracle, em vez de recuperar os dados para uma aplicação
- Aumentam a eficiência das consultas uma vez que são executadas na consulta em vez da aplicação
- Manipulam novos tipos de dados (por exemplo, latitude e longitude), codificando strings de caracteres e usando funções para operar nas strings

FIAP

Usando uma Função em uma Expressão SQL: Exemplo

```
CREATE OR REPLACE FUNCTION tax(p_value IN NUMBER)
 RETURN NUMBER IS
BEGIN
   RETURN (p_value * 0.08);
END tax;
/
SELECT employee_id, last_name, salary, tax(salary)
FROM    employees
WHERE department_id =100;
```

```
FUNCTION tax(value Compiled.
EMPLOYEE_ID           LAST_NAME                 SALARY               TAX(SALARY)
--------------------- ------------------------- -------------------- ----------------------
108                   Greenberg                 12000                960
109                   Faviet                    9000                 720
110                   Chen                      8200                 656
111                   Sciarra                   7700                 616
112                   Urman                     7800                 624
113                   Popp                      6900                 552

6 rows selected
```

16

Função em Expressões SQL: Exemplo

O exemplo do slide mostra como criar uma função `tax` para calcular o imposto de renda. A função aceita um parâmetro `NUMBER` e retorna o imposto de renda calculado com base em uma alíquota fixa simples de 8%.

Para executar o código exibido no exemplo do slide no SQL Developer, insira o código no SQL Worksheet e clique no ícone **Run Script**. A função `tax` é chamada como uma expressão na cláusula `SELECT` junto com o ID, o sobrenome e o salário dos funcionários de um departamento com ID `100`. O resultado retornado da função `tax` é exibido com a saída normal da consulta.

FIAP

Chamando Funções Definidas pelo Usuário em Instruções SQL

As funções definidas pelo usuário atuam como funções incorporadas de apenas uma linha e podem ser usadas nos seguintes locais:

- Na lista ou na cláusula `SELECT` de uma consulta
- Em expressões condicionais das cláusulas `WHERE` e `HAVING`
- Nas cláusulas `CONNECT BY`, `START WITH`, `ORDER BY`, e `GROUP BY` de uma cláusula
- Na cláusula `VALUES` da instrução `INSERT`
- Na cláusula `SET` da instrução `UPDATE`

17

Chamando Funções Definidas pelo Usuário em Instruções SQL

Uma função PL/SQL definida pelo usuário pode ser chamada a partir de qualquer expressão SQL em que seja possível chamar uma função incorporada de apenas uma linha, conforme é mostrado no seguinte exemplo:

```
SELECT employee_id, tax(salary)
FROM   employees
WHERE  tax(salary) > (SELECT MAX(tax(salary))
                      FROM employees
                      WHERE department_id = 30)
ORDER BY tax(salary) DESC;
```

```
EMPLOYEE_ID          TAX(SALARY)
-------------------- --------------------
100                  1920
101                  1360
102                  1360
145                  1120
146                  1080
201                  1040
205                  960
147                  960
108                  960
168                  920

10 rows selected
```

Restrições à Chamada de Funções em Expressões SQL

- As funções definidas pelo usuário que podem ser chamadas em expressões SQL devem:
  - Ser armazenadas no banco de dados
  - Aceitar apenas parâmetros `IN` com tipos de dados SQL válidos, e não tipos específicos PL/SQL-
  - Retornar tipos de dados SQL válidos, e não tipos específicos PL/SQL
- Ao chamar funções em instruções SQL:
  - Você deve ser o proprietário da função ou ter o privilégio `EXECUTE`.
  - Você pode precisar ativar a palavra-chave `PARALLEL_ENABLE` para permitir uma execução paralela da instrução SQL

18

Restrições à Chamada de Funções em Expressões SQL

As funções PL/SQL definidas pelo usuário que podem ser chamadas em expressões SQL devem atender aos seguintes requisitos:

- Devem ser armazenadas no banco de dados.
- Os parâmetros de função devem ser `IN` e de tipos de dados válidos SQL.
- Devem retornar tipos de dados SQL válidos. Não podem ser tipos de dados específicos PL/SQL, como `BOOLEAN`, `RECORD` ou `TABLE`. A mesma restrição se aplica aos parâmetros da função.

As seguintes restrições se aplicam à chamada de funções em uma instrução SQL:

- Os parâmetros devem usar a notação posicional. A notação nomeada não é suportada.
- Você deve ser o proprietário da função ou ter o privilégio `EXECUTE`.
- Você pode precisar ativar a palavra-chave `PARALLEL_ENABLE` para permitir uma execução paralela da instrução SQL usando a função. Cada escravo paralelo terá cópias privadas das variáveis locais da função.

As seguintes restrições também se aplicam às funções definidas pelo usuário: Não podem ser chamadas a partir da cláusula de constraint `CHECK` de uma instrução `CREATE TABLE` ou `ALTER TABLE`. Além disso, não podem ser usadas para especificar um valor default para uma coluna. Somente as funções armazenadas podem ser chamadas em instruções SQL. Os procedures armazenados só podem ser chamados a partir de uma função que atenda aos requisitos anteriores.

Controlando os Efeitos Colaterais ao Chamar Funções em Expressões SQL

FIAP

As funções chamadas em:

- Uma instrução `SELECT` não podem conter instruções DML
- Uma instrução `UPDATE` ou `DELETE` em uma tabela `T` não podem consultar nem conter instruções DML na mesma tabela `T`
- As instruções SQL não podem encerrar transações (ou seja, não podem executar operações `COMMIT` ou `ROLLBACK`)

Observação: As chamadas a subprogramas que não respeitam essas restrições também não são permitidas na função.

19

Controlando os Efeitos Colaterais ao Chamar Funções em Expressões SQL

Para executar uma instrução SQL que chame uma função armazenada, o servidor Oracle precisará saber se a função não está sujeita a efeitos colaterais específicos. Os efeitos colaterais são alterações inaceitáveis nas tabelas de banco de dados.

Restrições adicionais se aplicam quando uma função é chamada em expressões de instruções SQL.

- Quando uma função é chamada a partir de uma instrução `SELECT` ou uma instrução paralela `UPDATE` ou `DELETE`, a função não pode modificar tabelas de banco de dados.
- Quando uma função é chamada a partir de uma instrução `UPDATE` ou `DELETE`, a função não pode consultar ou modificar tabelas de banco de dados alteradas por essa instrução.
- Quando uma função é chamada a partir de uma instrução`SELECT`, `INSERT`, `UPDATE` ou `DELETE`, a função não pode ser executada diretamente ou indiretamente por outro subprograma ou instrução de controle de transação SQL como:
  - Uma instrução `COMMIT` ou `ROLLBACK` é emitida
  - Uma instrução de controle de sessão (como `SET ROLE`)
  - Uma instrução de controle do sistema (como `ALTER SYSTEM`)
  - Instruções DDL (como `CREATE`) porque elas são seguidas de um commit automático

Restrições à Chamada de Funções em SQL: Exemplo

```
CREATE OR REPLACE FUNCTION dml_call_sql(p_sal NUMBER)
   RETURN NUMBER IS
BEGIN
  INSERT INTO employees(employee_id, last_name,
                  email, hire_date, job_id, salary)
  VALUES(1, 'Frost', 'jfrost@company.com',
         SYSDATE, 'SA_MAN', p_sal);
  RETURN (p_sal + 100);
END;
```

```
UPDATE employees
  SET salary = dml_call_sql(2000)
WHERE employee_id = 170;
```

```
FUNCTION dml_call_sql(p_sal Compiled.

Error starting at line 1 in command:
UPDATE employees
  SET salary = dml_call_sql(2000)
WHERE employee_id = 170
Error report:
SQL Error: ORA-04091: table ORA62.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA62.DML_CALL_SQL", line 4
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:    A trigger (or a user defined plsql function that is referenced in
           this statement) attempted to look at (or modify) a table that was
           in the middle of being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read that table.
```

20

Restrições à Chamada de Funções em SQL: Exemplo

A função `dml_call_sql` do slide contém uma instrução `INSERT` que insere um novo registro na tabela `EMPLOYEES` e retorna o valor de salário de entrada incrementado em 100. Essa função é chamada na instrução `UPDATE` que modifica o salário do funcionário 170 para a quantia retornada da função. Ocorre uma falha na instrução `UPDATE`, cujo erro indica que a tabela está em mutação (ou seja, as alterações já estão em andamento na mesma tabela). No exemplo a seguir, a função `query_call_sql` consulta a coluna `SALARY` da tabela `EMPLOYEES`:

```
CREATE OR REPLACE FUNCTION query_call_sql(p_a NUMBER)
   RETURN NUMBER IS
   v_s NUMBER;
 BEGIN
   SELECT salary INTO v_s FROM employees
   WHERE employee_id = 170;
   RETURN (v_s + p_a);
 END;
```

Quando chamada na instrução `UPDATE`a seguir, ela retorna uma mensagem de erro semelhante à mostrada no slide:

```
UPDATE employees SET salary = query_call_sql(100)
WHERE employee_id = 170;
```

Notação Nomeada e Combinada em SQL

- O código PL/SQL permite que os argumentos em uma chamada de sub-rotina sejam especificados usando a notação posicional, nomeada ou combinada.
- Antes do Oracle Database 11*g*, somente a notação posicional era suportada em chamadas do SQL.
- A partir do Oracle Database 11*g*, a notação nomeada e combinada pode ser usada para especificar argumentos em chamadas para sub-rotinas PL/SQL a partir de instruções SQL.
- Em listas de parâmetros longos, com a maioria tendo valores default, você pode omitir valores originados cos parâmetros opcionais.
- Pode evitar a duplicação do valor default do parâmetro opcional em cada instalação de chamada.

21

Notação Nomeada e Combinada em SQL: Exemplo

```
CREATE OR REPLACE FUNCTION f(
  p_parameter_1 IN NUMBER DEFAULT 1,
  p_parameter_5 IN NUMBER DEFAULT 5)
RETURN NUMBER
IS
 v_var number;
BEGIN
  v_var := p_parameter_1 + (p_parameter_5 * 2);
  RETURN v_var;
END f;
/
```

```
FUNCTION f( Compiled.
```

```
SELECT f(p_parameter_5 => 10) FROM DUAL;
```

```
F(P_PARAMETER_5=>10)
----------------------
21

1 rows selected
```

22

Exemplo de Uso da Notação Nomeada e Combinada em uma Instrução SQL

No exemplo do slide, a chamada para a função `f`na instrução SQL `SELECT`utiliza a notação nomeada. Antes do Oracle Database 11*g*, não era possível usar a notação nomeada ou combinada ao especificar parâmetros para uma função a partir de uma instrução SQL. Antes do Oracle Database 11*g*, ocorria o seguinte erro:

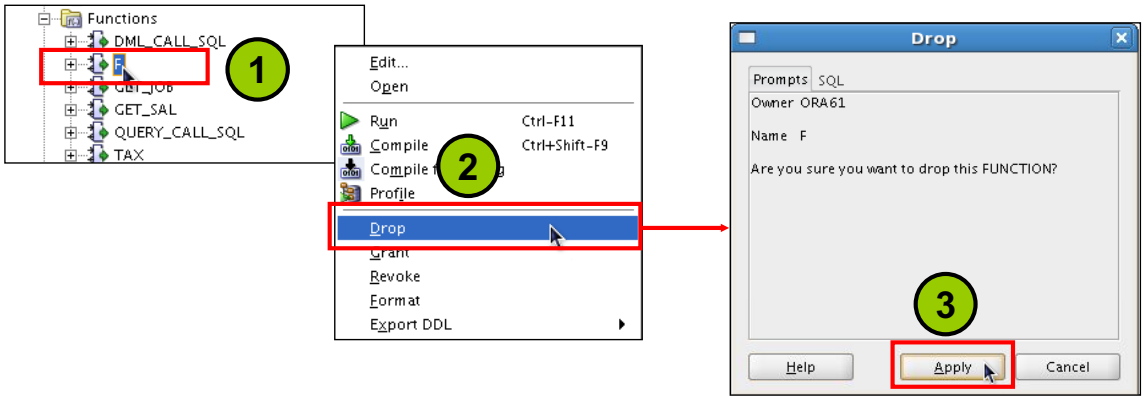
```
SELECT f(p_parameter_5 => 10) FROM DUAL;

ORA-00907: missing right parenthesis
```

Removendo Funções: Usando a Instrução DROP do SQL ou SQL Developer

- **Usando a instrução DROP:**

```
DROP FUNCTION f;
```

- **Usando o SQL Developer:**



23

Removendo Funções

Usando a instrução DROP

Quando uma função armazenada não for mais necessária, você poderá usar uma instrução SQL no SQL*Plus para eliminá-la. Para remover uma função armazenada usando o SQL*Plus, execute o comando SQL `DROP FUNCTION`.

Usando CREATE OR REPLACE Versus DROP e CREATE

A cláusula `REPLACE` na sintaxe`CREATE OR REPLACE` equivale a eliminar uma função e depois recriá-la. Quando você usa a sintaxe `CREATE OR REPLACE`, os privilégios concedidos nesse objeto a outros usuários permanecem os mesmos. Quando você usa o comando `DROP` para eliminar uma função e depois a recria, todos os privilégios concedidos nessa função são automaticamente revogados.

Usando o SQL Developer

Para eliminar uma função no SQL Developer, clique com o botão direito do mouse no nome da função no nó **Functions** , e selecione **Drop**. A caixa de diálogo **Drop**será exibida. Para eliminar a função, clique em **Apply**.

Exibindo Funções
Usando Views de Dicionário de Dados

```
DESCRIBE USER_SOURCE
```

```
DESCRIBE user_source
Name                           Null     Type
------------------------------ -------- ----------------------------------------
NAME                                    VARCHAR2(30)
TYPE                                    VARCHAR2(12)
LINE                                    NUMBER
TEXT                                    VARCHAR2(4000)

4 rows selected
```

```
SELECT  text
FROM    user_source
WHERE   type = 'FUNCTION'
ORDER BY line;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA

Results:

	TEXT
1	FUNCTION tax(p_value IN NUMBER)
2	FUNCTION query_call_sql(p_a NUMBER) RETURN NUMBER IS
3	FUNCTION get_sal
4	FUNCTION dml_call_sql(p_sal NUMBER)
5	RETURN NUMBER IS
6	RETURN NUMBER IS
7	(p_id employees.employee_id%TYPE) RETURN NUMBER IS
8	v_s NUMBER;

. . .

24

Exibindo Funções Usando Views de Dicionário de Dados

O código-fonte das funções PL/SQL é armazenado nas tabelas do dicionário de dados. O código-fonte pode ser acessado pelas funções PL/SQL que são compiladas com sucesso ou não. Para exibir o código da função PL/SQL armazenado no dicionário de dados, execute uma instrução `SELECT` nas seguintes tabelas, em que o valor da coluna `TYPE` é `FUNCTION`:

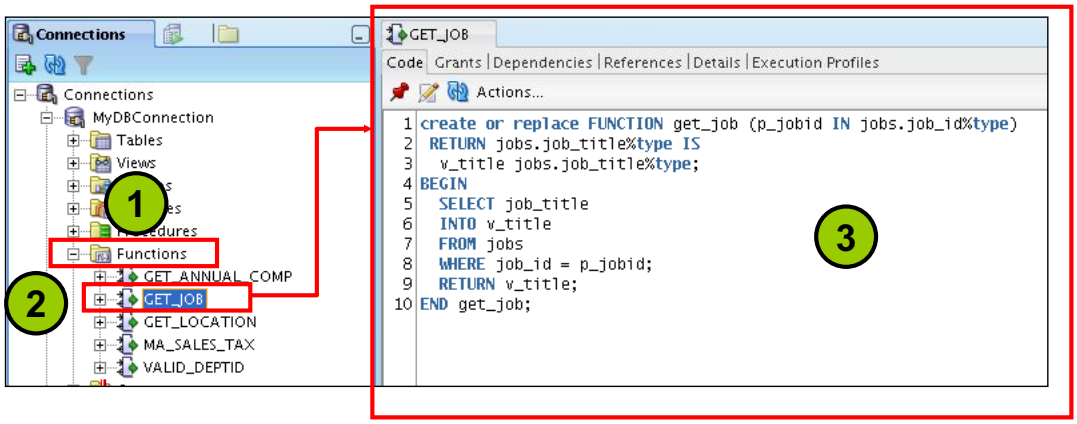
- Tabela `USER_SOURCE` para exibir o código PL/SQL pertencente a você
- Tabela `ALL_SOURCE` para exibir o código PL/SQL para o qual o proprietário do código desse subprograma tenha lhe concedido o direito `EXECUTE`

O segundo exemplo no slide usa a tabela `USER_SOURCE` para exibir o código-fonte para todas as funções no seu esquema.

Você também pode usar a view de dicionário de dados `USER_OBJECTS` para exibir uma lista dos seus nomes de função.

Observação: A saída do segundo exemplo de código no slide foi gerada usando o ícone Execute Statement (F9) na barra de ferramentas para fornecer saída mais bem formatada.

Exibindo as Informações das Funções Usando o SQL Developer



25

Exibindo as Informações das Funções Usando o SQL Developer

Para exibir o código de uma função no SQL Developer, faça o seguinte:

1. Clique no nó **Functions** na tab **Connections**
2. Clique no nome da função.
3. O código da função é exibido na tab **Code** conforme mostrado no slide.

Questionário

Uma função armazenada PL/SQL:

1. Pode ser chamada como parte de uma expressão
2. Deve conter uma cláusula `RETURN`no cabeçalho
3. Deve retornar um único valor
4. Deve conter pelo menos uma instrução `RETURN`
5. Não contém uma cláusula `RETURN`no cabeçalho

26

Resposta: 1, 2, 3, 4

Exercício 10: Visão Geral

FIAP

Este exercício aborda os seguintes tópicos:

- Criando funções armazenadas:
  - Para consultar uma tabela do banco de dados e retornar valores específicos
  - Para usá-las em uma instrução SQL
  - Para inserir uma nova linha, com valores de parâmetros especificados, em uma tabela do banco de dados
  - Usando valores de parâmetros default
- Chamando uma função armazenada a partir de uma instrução SQL
- Chamando uma função armazenada a partir de um procedure armazenado

27

Exercício 2-1: Visão Geral

Recomenda-se usar o SQL Developer neste exercício.

Sumário

Nesta lição, você aprendeu a:

- Fazer a distinção entre procedures e funções
- Descrever os usos de funções
- Criar funções armazenadas
- Chamar uma função
- Remover uma função

28

Sumário

Uma função é um bloco PL/SQL nomeado que deve retornar um valor. Em geral, as funções são criadas para calcular e retornar um valor, e os procedures, para executar uma ação.

É possível criar ou eliminar funções.

As funções são chamadas como parte de uma expressão.