

# **Introdução Procedimentos Armazenados – PL/SQL**

Baseado nos slides do curso de Oracle

# Objetivos

---

▶ Após completar esta lição, você deveria ser capaz de fazer o seguinte:

- ▶ Diferenciar entre blocos anônimos e subprogramas
- ▶ Criar um procedimento simples e invocá-lo desde um bloco anônimo
- ▶ Criar uma função simples
- ▶ Criar uma função simples que aceita um parâmetro
- ▶ Diferenciação entre procedimentos e funções



# Agenda

---

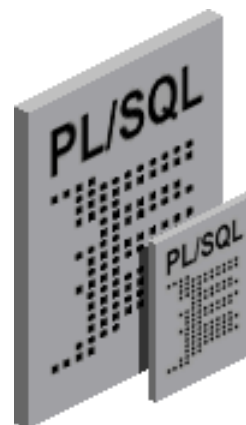
- ▶ Introduzindo procedimentos e funções
- ▶ Visualizando procedimentos
- ▶ Visualizando funções



# Procedimentos e Funções

---

- ▶ São blocos PL/SQL com nomes
- ▶ São subprogramas PL/SQL que podem ser chamados
- ▶ Tem estruturas de bloco similares aos blocos anônimos:
  - ▶ Seção declarativa opcional (sem a palavra chave `DECLARE` )
  - ▶ Seção executável Obrigatória
  - ▶ Seção opcional para tratar exceções



# Diferenças Entre Blocos Anônimos e Subprogramas

Blocos Anônimos	Subprogramas
Blocos PL/SQL sem nome	Blocos PL/SQL com nome
Sempre Compilado	Compilado somente uma vez
Não armazenado no BD	Armazenado no BD
Não podem ser invocados por outras aplicações	Nomeados e, além disso, podem ser invocados por outras aplicações
Não retorna valores	Se funcionarem, devem retornar valores
Não podem ter parâmetros	Podem ter parâmetros



# Procedimento: Sintaxe

```
CREATE [OR REPLACE] PROCEDURE
  nome_procedimento
  [(argumento1 [modo1] tipo_dado1,
   argumento2 [modo2] tipo_dado2,
   . . .)]
IS|AS
Corpo_procedimento;
```

**Atenção:** não é possível fazer qualquer restrição ao tamanho do tipo de dado neste ponto.

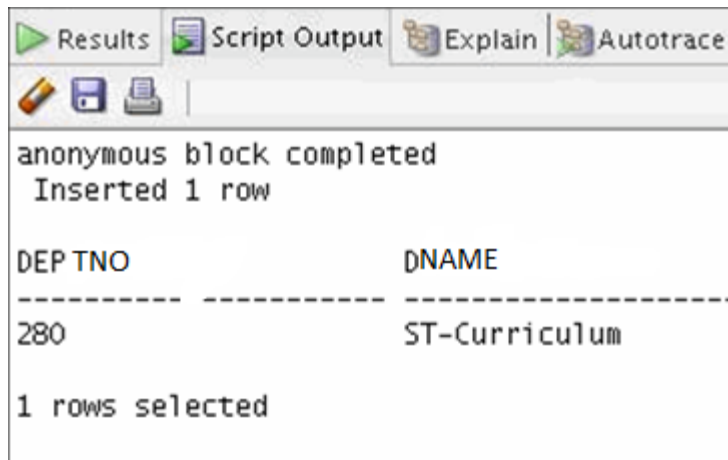
Read more: [http://www.linhadecodigo.com.br/artigo/335/pl\\_sql-procedures-e-funcoes.aspx#ixzz2Cd6gmfTV](http://www.linhadecodigo.com.br/artigo/335/pl_sql-procedures-e-funcoes.aspx#ixzz2Cd6gmfTV)

# Criando um procedimento

```
...  
CREATE TABLE dept AS SELECT * FROM departamentos;  
CREATE PROCEDURE adicione_dept IS  
  v_dept_id dept.deptno%TYPE;  
  v_dept_nome dept.dname%TYPE;  
BEGIN  
  v_dept_id:=280;  
  v_dept_nome:='ST-Curriculum';  
  INSERT INTO dept(deptno, dname)  
  VALUES(v_dept_id,v_dept_nome);  
  DBMS_OUTPUT.PUT_LINE(' Inseridas ' || SQL%ROWCOUNT  
    || ' linhas ');  
END;
```

# Invocando um Procedimento

```
...  
BEGIN  
  adicione_dept;  
END;  
/  
SELECT deptno, dname FROM dept WHERE deptno=280;
```



The screenshot shows a database client window with a toolbar at the top containing icons for Results, Script Output, Explain, and Autotrace. Below the toolbar, the text "anonymous block completed" and "Inserted 1 row" is displayed. A table with two columns, DEPTNO and DNAME, is shown with a single row of data: 280 and ST-Curriculum. At the bottom, it says "1 rows selected".

DEPTNO	DNAME
280	ST-Curriculum



# Função: Sintaxe

```
CREATE [OR REPLACE] FUNCTION nome_função
  [(argumento1 [modo1] tipo_dado1,
    argumento2 [modo2] tipo_dado2,
    . . .)]
RETURN tipo_dado
IS|AS
Corpo_função;
```

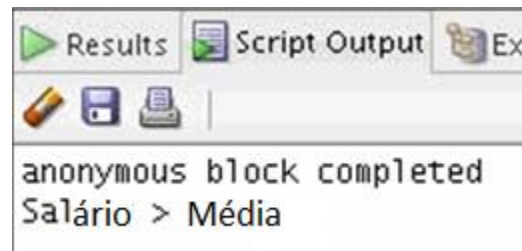


# Criando uma função

```
CREATE FUNCTION verifica_sal RETURN Boolean IS
  v_dept_id empregados.deptno%TYPE;
  v_empno    empregados.empno%TYPE;
  v_sal      empregados.sal%TYPE;
  v_avg_sal  empregados.sal%TYPE;
BEGIN
  v_empno:=205;
  SELECT sal, deptno INTO v_sal,v_dept_id FROM empregados
  WHERE empno = v_empno;
  SELECT avg(sal) INTO v_avg_sal FROM empregados WHERE
    deptno = v_dept_id;
  IF v_sal > v_avg_sal THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN NULL;
END;
```

# Invocando uma Função

```
BEGIN
  IF (verifica_sal IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('A função retornou
      NULL devido a exceção');
  ELSIF (verifica_sal) THEN
    DBMS_OUTPUT.PUT_LINE('Salário > Média');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Salário < Média');
  END IF;
END;
/
```



# Passando um Parâmetro à Função

```
DROP FUNCTION verifica_sal;
CREATE FUNCTION verifica_sal(p_empno empregados.empno%TYPE)
RETURN Boolean IS
    v_dept_id empregados.deptno%TYPE;
    v_sal      empregados.sal%TYPE;
    v_avg_sal  empregados.sal%TYPE;
BEGIN
    SELECT sal, deptno INTO v_sal, v_dept_id FROM empregados
        WHERE empno = p_empno;
    SELECT avg(sal) INTO v_avg_sal FROM emp
        WHERE deptno = v_dept_id;
    IF v_sal > v_avg_sal THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
EXCEPTION
    ...
```

# Invocando a Função com um Parâmetro

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Verificando por empregado com id 205');
  IF (verifica_sal(205) IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('A função retornou
      NULL devido a exceção');
  ELSIF (verifica_sal(205)) THEN
    DBMS_OUTPUT.PUT_LINE('Salário > Média');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Salário < Média');
  END IF;
DBMS_OUTPUT.PUT_LINE('Verificando por empregado com id 70');
  IF (verifica_sal(70) IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('A função retornou
      NULL devido a exceção');
  ELSIF (verifica_sal(70)) THEN
    ...
  END IF;
END;
/
```

# Criando Procedimentos

---

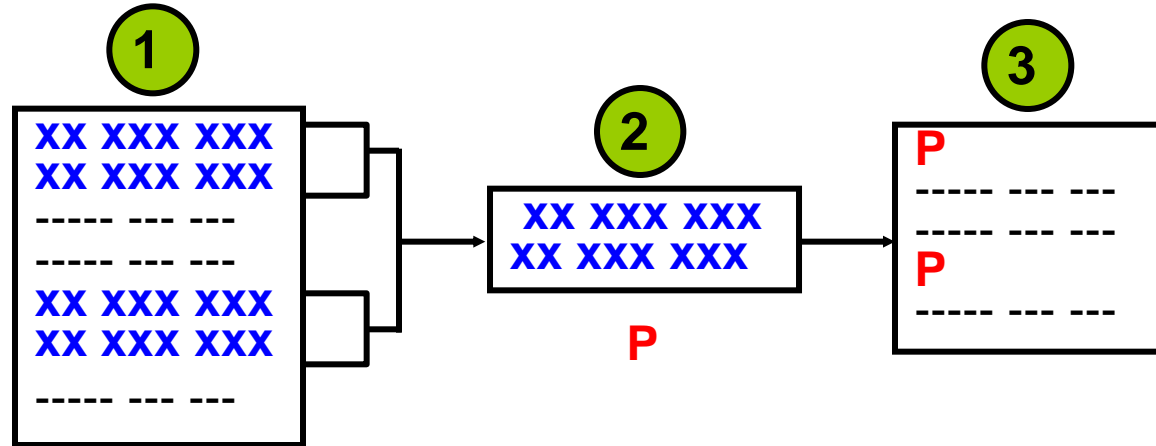
## ▶ Objetivos

▶ Depois de completar esta lição, você deveria ser capaz de fazer o seguinte:

- ▶ Identificar os benefícios do projeto de subprogramas modularizados e em camadas
- ▶ Criar e chamar procedimentos
- ▶ Usar parâmetros formais e reais
- ▶ Usar notação posicional, nomeada, ou misturada para a passagem de parâmetros
- ▶ Identificar as formas disponíveis de passagem de parâmetros
- ▶ Tratar exceções em procedimentos
- ▶ Remover um procedimento
- ▶ Apresentar a informação dos procedimentos

# Criando um Projeto de subprogramas modularizado

---



Modularizar código em subprogramas.

1. Localizar sequências de código repetido.
  2. Criar um subprograma **P** que contem o código repetido
  3. Modificar o código original para invocar o novo subprograma.
-

# Criando um projeto de subprogramas em camadas

---

- ▶ Criar camadas de subprogramas para a sua aplicação.
  - ▶ Camada de subprogramas de acesso aos dados com a lógica SQL
  - ▶ Camada de subprogramas da lógica de negócios, as quais podem, ou não, usar a camada de acesso aos dados





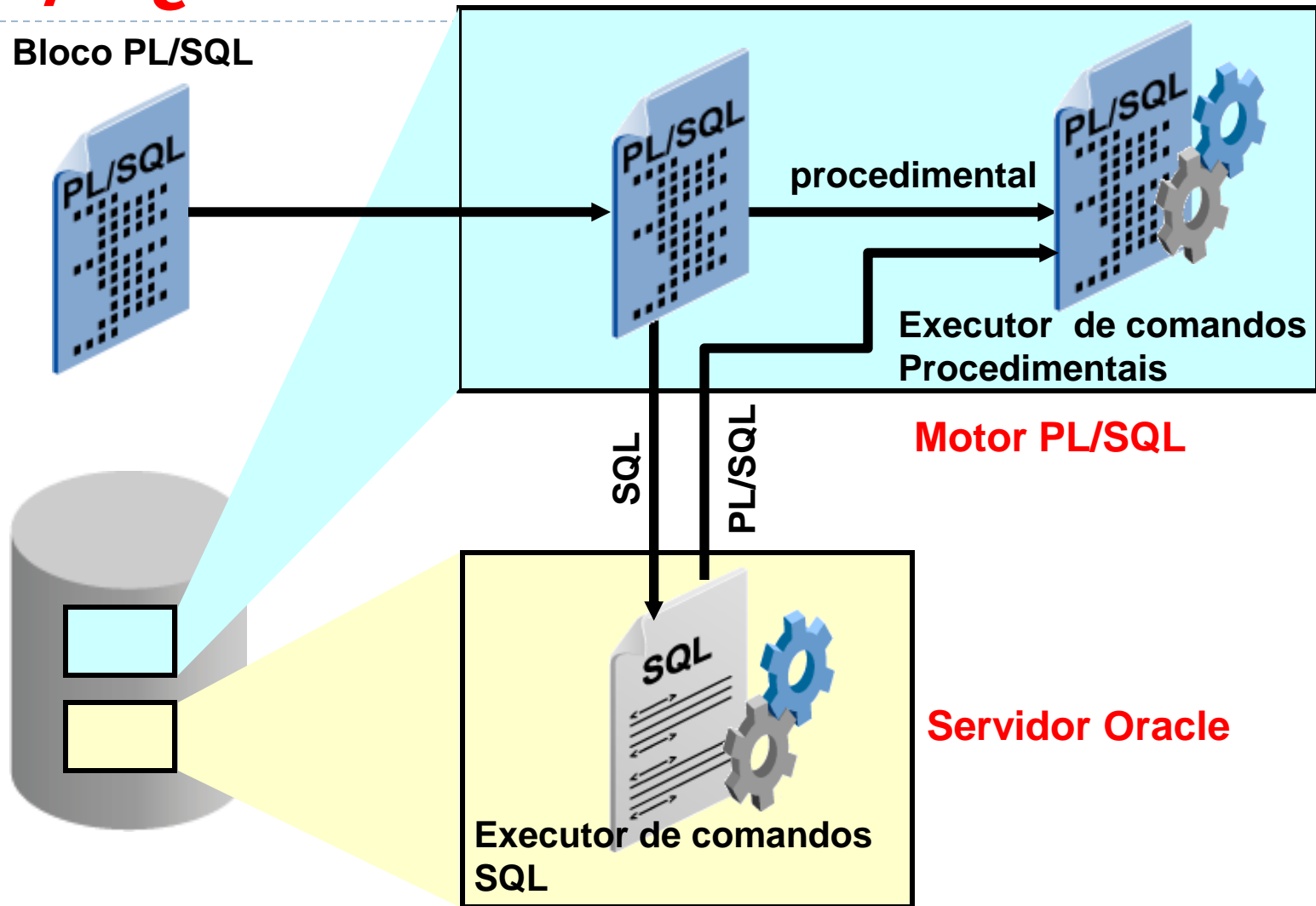
# Modularizando o Desenvolvimento com Blocos PL/SQL

---

- ▶ PL/SQL é uma linguagem estruturada em blocos. O bloco de código PL/SQL ajuda modularizar o código usando:
  - ▶ Blocos Anônimos
  - ▶ Procedimentos e funções
  - ▶ Packages
  - ▶ Triggers de BDs
- ▶ Os benefícios do uso da construção modular de programas são:
  - ▶ Fácil manutenção
  - ▶ Melhora na segurança e integridade dos dados
  - ▶ Melhora no desempenho
  - ▶ Melhora na clareza do código



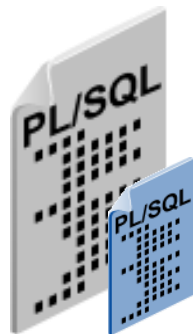
# Arquitetura em tempo de Execução PL/SQL



# O que são subprogramas PL/SQL?

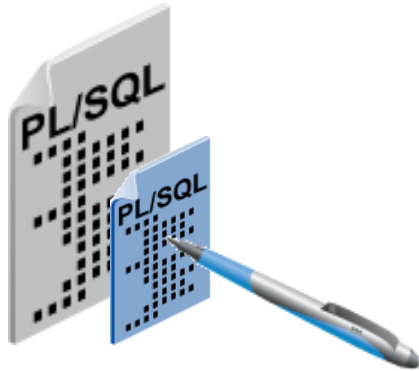
---

- ▶ Um subprograma PL/SQL é um bloco PL/SQL com nome que pode ser chamado com um conjunto de parâmetros.
- ▶ Você pode declarar e definir um subprograma dentro de um bloco PL/SQL ou outro subprograma.
- ▶ Um subprograma consiste de uma especificação e um corpo.
- ▶ Um subprograma pode ser um procedimento ou uma função.
- ▶ Tipicamente, você usa um procedimento para executar uma ação e uma função para calcular e retornar um valor.
- ▶ Subprogramas podem ser agrupados em pacotes PL/SQL (packages) .



# Os Benefícios de Usar Subprogramas PL/SQL

---



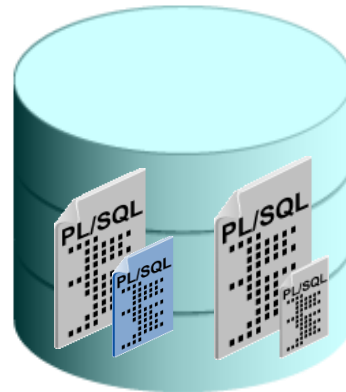
**Fácil manutenção**



**Dados com segurança e integridade melhorada**



**Clareza de código melhorada**



**Subprogramas:  
Procedimentos e  
funções armazenados**



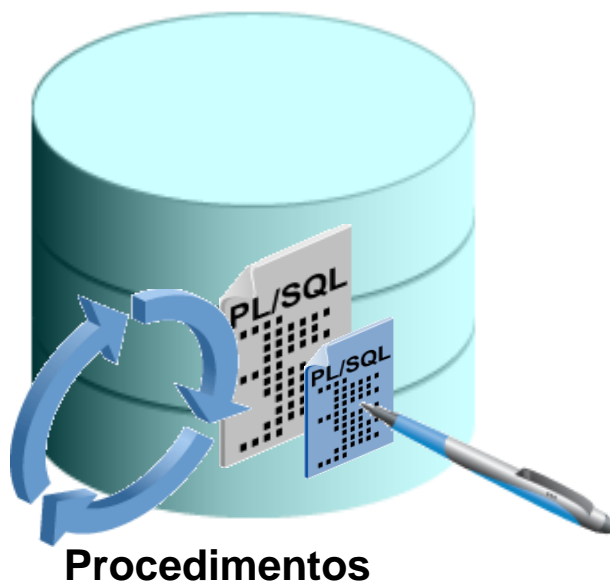
**Desempenho melhorado**



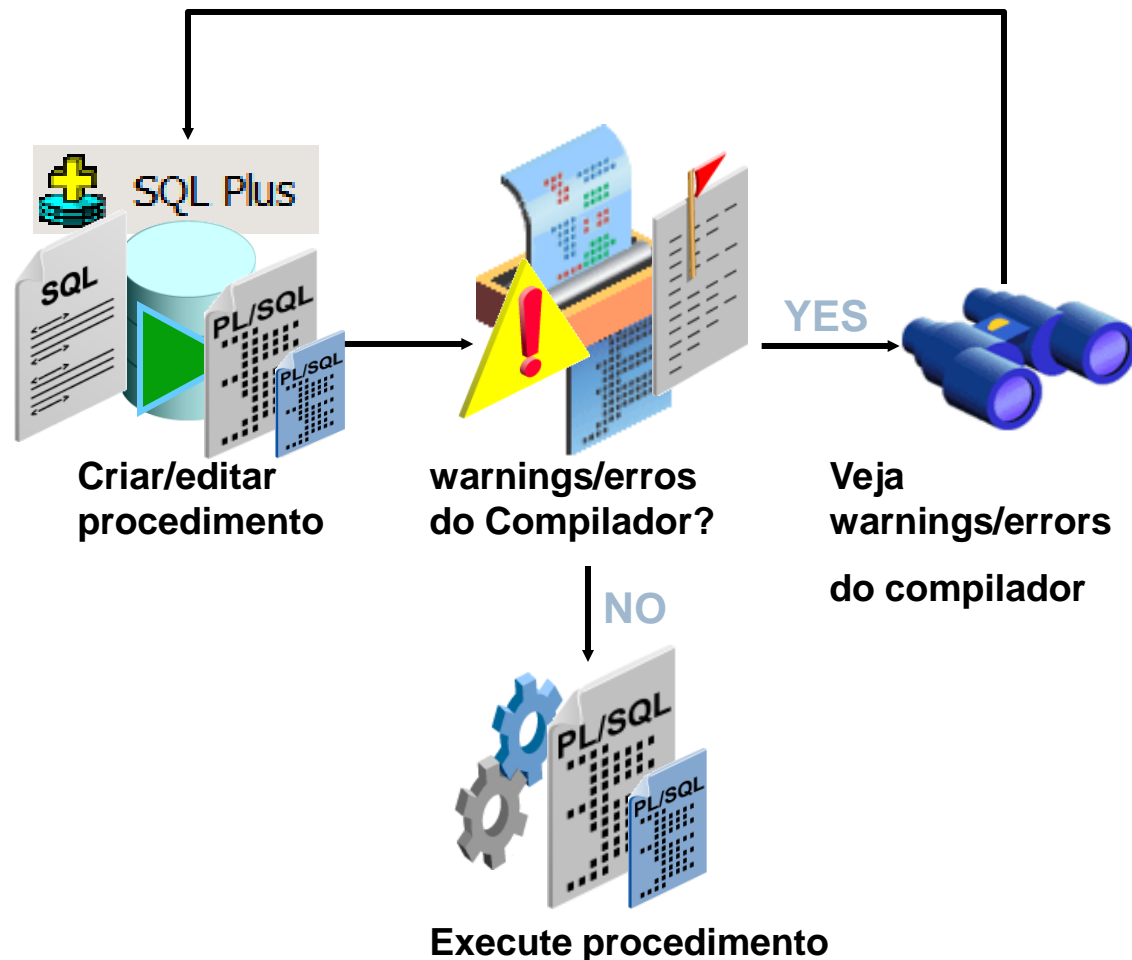
# O que são procedimentos?

---

- ▶ São um tipo de subprogramas que executa uma ação
- ▶ Podem ser armazenados no BD como um objeto do esquema
- ▶ Promove reuso e manutenção



# Criando procedimentos: Resumo



Project: C:\Program Files\SQL Developer 1.1\sql  
PROCEDURE ORA41.ADD\_JOB\_HISTORY@  
Error(8,5): PLS-00103: Encountered the

**Veja errors/warnings em SQL Developer**

**Use SHOW ERRORS command em SQL\*Plus**

**Use USER/ALL/DBA\_ERRORS views**

# Criando Procedimentos com o comando SQL CREATE OR REPLACE

---

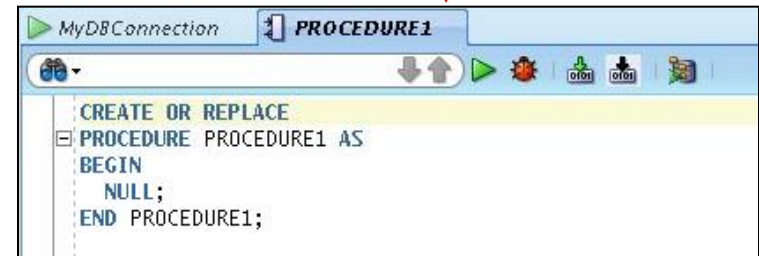
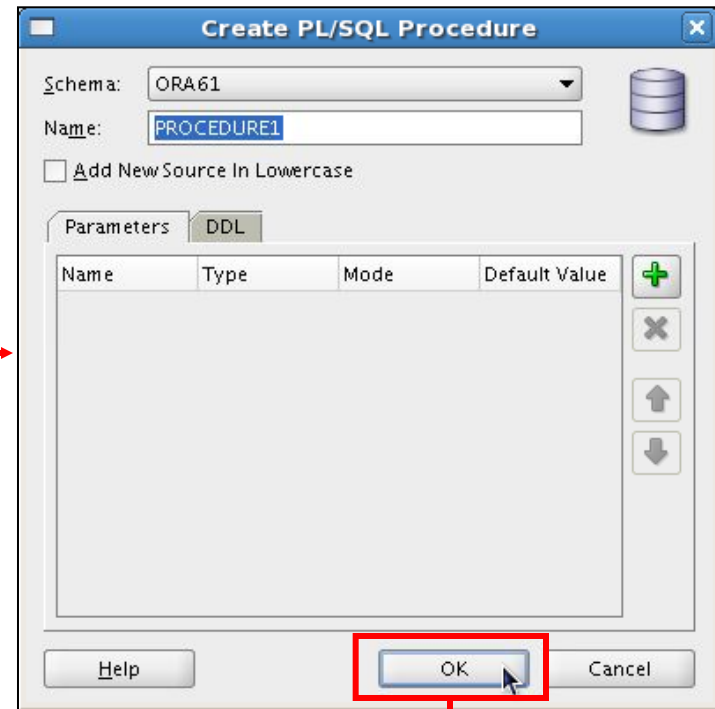
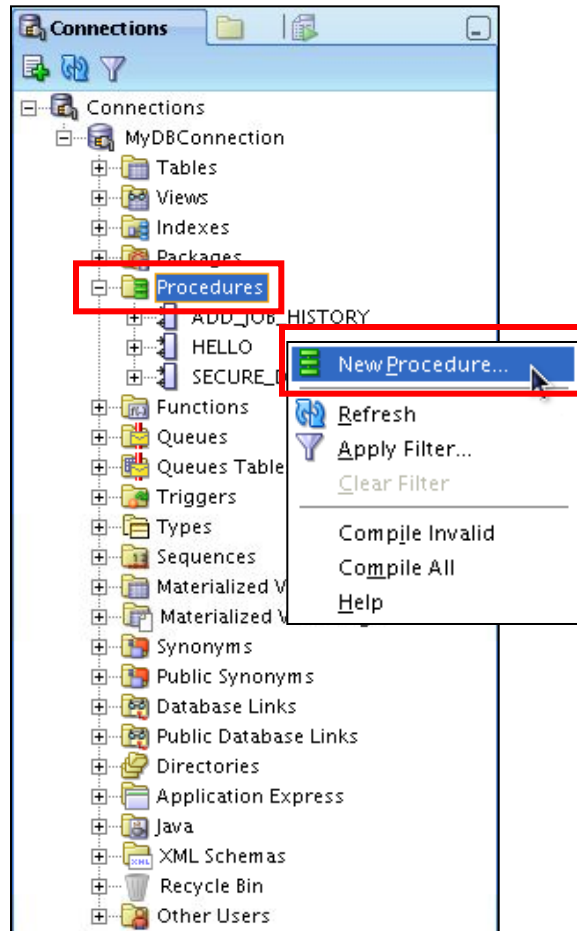
- ▶ Use a cláusula CREATE para criar um procedimento standalone que é armazenado no BD Oracle database.
- ▶ Use o OR REPLACE opção para sobrescrever um procedimento existente.

```
CREATE [OR REPLACE] PROCEDURE nome_procedimento
  [(parametro1 [modo] tipo_dados1,
    parametro2 [modo] tipo_dados2, ...)]
IS|AS
  [declarações_variáveis_locais; ...]
BEGIN
  -- ações;
END [nome_procedimento];
```

**bloco PL/SQL**



# Criando Procedimentos Usando SQL Developer





# Compilando Procedimentos e Apresentando Erros de Compilação no SQL Developer

The diagram illustrates two methods to compile a procedure in SQL Developer, both leading to a compilation error.

**Method 1 (Left):** A tree view on the left shows a procedure named 'HELLO' under the 'Procedures' folder. A context menu is open, and the 'Compile' option (Ctrl+Shift-F9) is highlighted. A green circle with the number '1' is next to the menu.

**Method 2 (Right):** A code editor window shows the SQL code for the 'HELLO' procedure. The 'Compile' button in the toolbar (Ctrl+Shift-F9) is highlighted. A green circle with the number '2' is next to the toolbar.

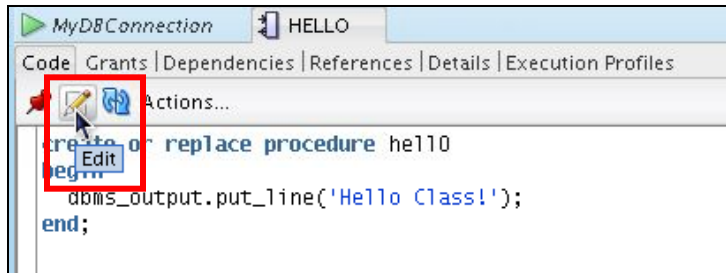
**OR**

Both methods lead to the 'Compiler - Log' window, which displays the following error:

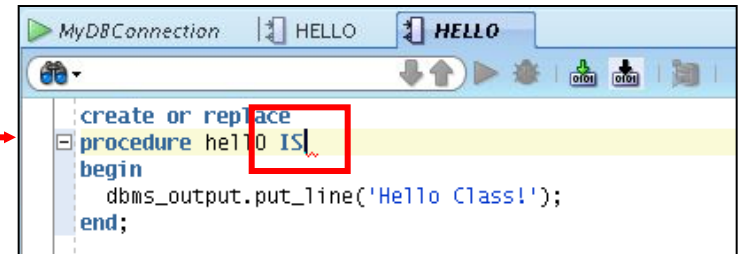
```
Project: /home/oracle/.sqldeveloper/system1.5.4.59.41/o.sqldeveloper.11.1.1.59.41/DefaultWorkspace/Project1.jpr
PROCEDURE ORA61.HELLO@MyDBConnection
Error(3,1): PLS-00103: Encountered the symbol "BEGIN" when expecting one of the following:  (; is with authid as cluster co
```

The error message indicates a syntax error: "Error(3,1): PLS-00103: Encountered the symbol 'BEGIN' when expecting one of the following: (; is with authid as cluster co".

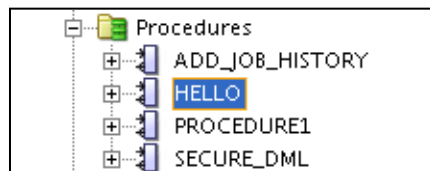
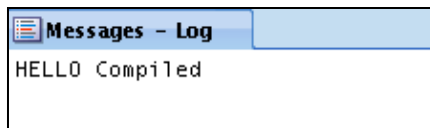
# Corrigindo Erros de Compilação no SQL Developer



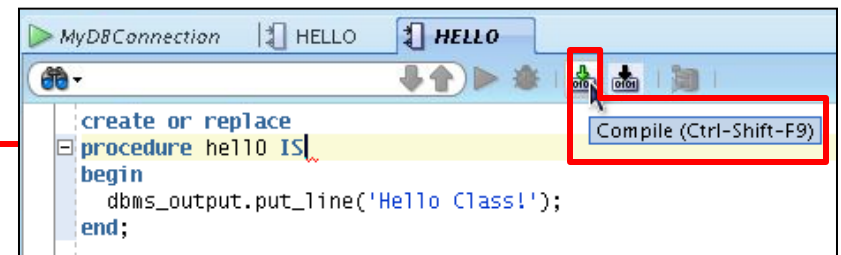
1. Editar procedimento



2. Corrigir erro (adicione keyword IS)



4. Recompilação com sucesso



3. Recompilar procedimento

# O que são Parâmetros e tipos de Parâmetros?

---

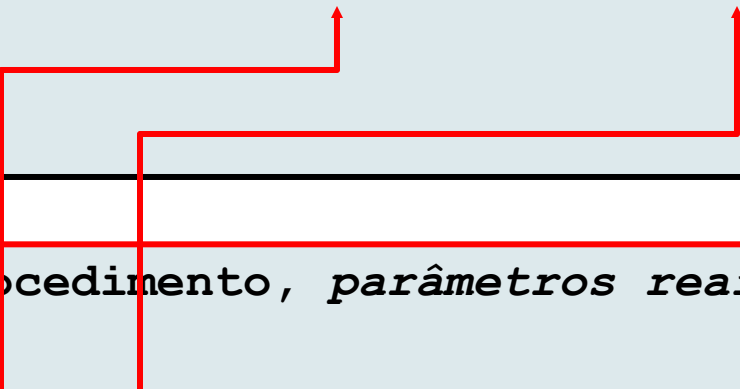
- ▶ São declarados após o nome do subprogram no cabeçalho PL/SQL
- ▶ Passar ou comunicar dados entre o ambiente de chamado e o subprograma
- ▶ São usados como variáveis locais mas são dependentes da sua forma de passagem:
  - ▶ Um parâmetro `IN` (por definição) fornece valores para que um subprograma os processe
  - ▶ Um parâmetro `OUT` retorna um value para quem chama
  - ▶ Um parâmetro `IN OUT` fornece um valor de entrada, que pode ser retornado como um valor modificado



# Parâmetros Formais e Reais

- ▶ Parâmetros Formais: variáveis locais declaradas na lista de parâmetros da especificação do subprograma
- ▶ Parâmetros Reais (ou argumentos): Valores Literais, variáveis, e expressões usadas na lista de parâmetros do subprograma que chama

```
-- Definição do Procedimento, parâmetros formais  
CREATE PROCEDURE eleva_sal(p_id NUMBER, p_sal NUMBER) IS  
BEGIN  
  . . .  
END eleva_sal;
```

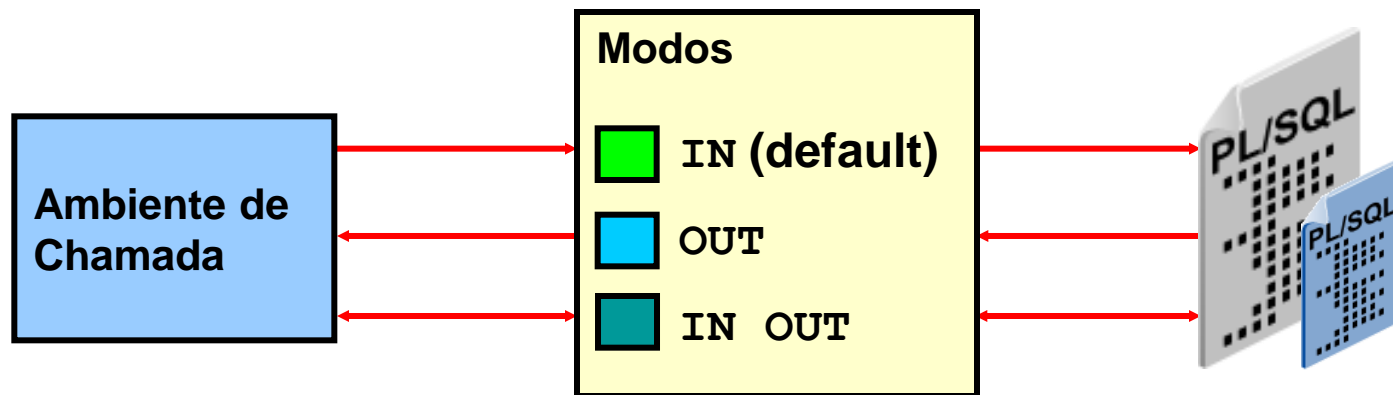


```
-- Chamada do Procedimento, parâmetros reais  
(argumentos)  
v_emp_id := 100;  
eleva_sal(v_emp_id, 2000)
```

# Formas de Parâmetros Procedimental

- ▶ Formas dos Parâmetros são especificados na declaração dos parâmetros formais, após o nome do parâmetro e antes do seu tipo de dados.
- ▶ A forma `IN` é a default se não ha uma definição explícita.

```
CREATE PROCEDURE nome_proc (nome_param [modo] tipodado)  
... 
```



# Comparando os modos dos the Parâmetros

---

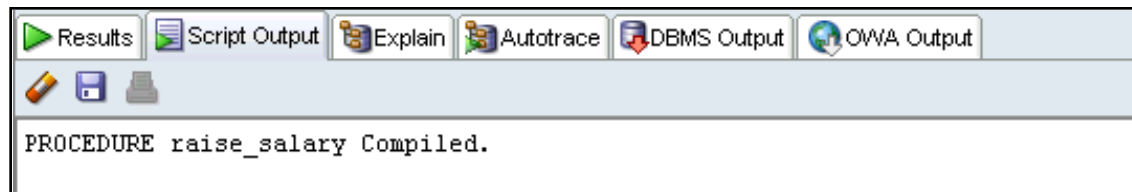
IN	OUT	IN OUT
Modo pré-definido	Deve ser especificado	Deve ser especificado
Valor é passado ao subprograma	O valor é retornado para o ambiente de chamada	O valor passado no subprograma; valor retornado para o ambiente de chamada
Parâmetro formal atua como uma constante	Variável não inicializada	Variável inicializada
Parâmetro real pode ser um a literal, expressão, constante, ou variável inicializada	Deve ser uma variável	Deve ser uma variável
Pode ser atribuído um valor default	Não pode ser atribuído um valor default	Não pode ser atribuído um valor default



# Usando o Parâmetro do modo IN :

## Exemplo

```
CREATE OR REPLACE PROCEDURE raise_salary
(p_id      IN emp.empno%TYPE,
 p_percentagem IN NUMBER)
IS
BEGIN
    UPDATE emp
    SET     sal = sal * (1 + p_percentagem/100)
    WHERE  empno = p_id;
END raise_salary;
/
```



```
EXECUTE raise_salary(176, 10)
```

# Usando o Parâmetro do modo OUT :

## Exemplo

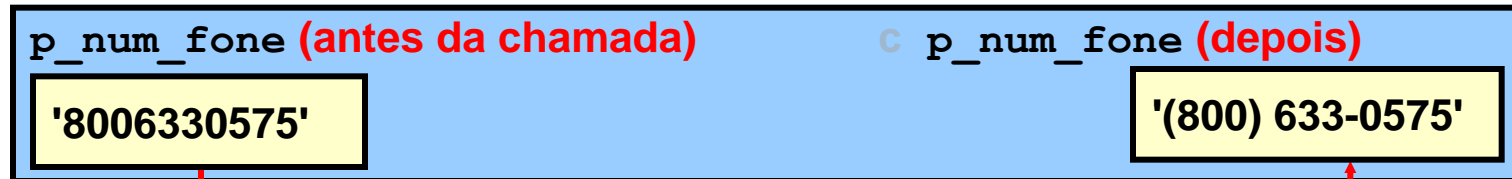
```
CREATE OR REPLACE PROCEDURE consulta_emp
  (p_id      IN  emp.empno%TYPE,
   p_name    OUT emp.ename%TYPE,
   p_salario OUT emp.sal%TYPE) IS
BEGIN
  SELECT  ename, sal INTO p_name, p_salario
  FROM    emp
  WHERE   empno = p_id;
END consulta_emp;
/
```

```
SET SERVEROUTPUT ON
DECLARE
  v_emp_name emp.ename%TYPE;
  v_emp_sal  emp.sal%TYPE;
BEGIN
  consulta_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE(v_emp_name||' ganha '||
    to_char(v_emp_sal, '$999,999.00'));
END;
/
```



# Usando o Parâmetro do modo IN OUT : Exemplo

Ambiente de Chamada



```
CREATE OR REPLACE PROCEDURE formato_fone
  (p_num_fone IN OUT VARCHAR2) IS
BEGIN
  p_num_fone := '(' || SUBSTR(p_num_fone,1,3) ||
                ')' || SUBSTR(p_num_fone,4,3) ||
                '-' || SUBSTR(p_num_fone,7);
END formato_fone;
/
```

```
anonymous block completed
b_phone_no
-----
8006330575

anonymous block completed
b_phone_no
-----
(800) 633-0575
```

# Visualizando os parâmetros OUT: Usando a subrotina DBMS\_OUTPUT.PUT\_LINE

---

- ▶ Usando variáveis PL/SQL que são impressas com chamadas ao procedimento DBMS\_OUTPUT.PUT\_LINE

```
SET SERVEROUTPUT ON

DECLARE
  v_emp_name emp.ename%TYPE;
  v_emp_sal   emp.sal%TYPE;
BEGIN
  consulta_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE('Nome: ' || v_emp_name);
  DBMS_OUTPUT.PUT_LINE('Salario: ' || v_emp_sal);
END;
```

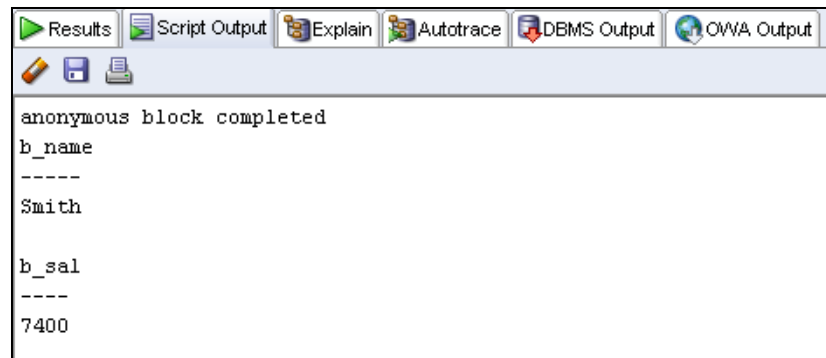
```
anonymous block completed
Name: Smith
Salary: 7400
```

# Visualizando Parâmetros OUT: Usando variáveis hospedeiras SQL\*Plus

---

1. Use variáveis hospedeiras SQL\*Plus.
2. Execute CONSULTA\_EMP usando variáveis hospedeiras.
3. Imprima as variáveis hospedeiras.

```
VARIABLE b_name    VARCHAR2(25)
VARIABLE b_sal      NUMBER
EXECUTE consulta_emp(171, :b_name, :b_sal)
PRINT b_name b_sal
```



# Notações Disponíveis para a passagem de parâmetros reais

---

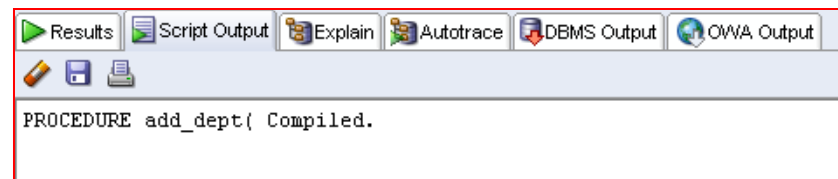
- ▶ Quando é chamado um subprograma, você pode escrever parâmetros reais usando as notações seguintes:
  - ▶ Posicional: Lista os parâmetros reais na mesma ordem que os parâmetros formais
  - ▶ Nomeados: Lista os parâmetros reais em ordem arbitrária e usa o operador associação ( $=>$ ) para associar um parâmetro formal nomeado com seu parâmetro real
  - ▶ Misturado: Lista alguns dos parâmetros reais como posicionais e alguns como nomeados.
- ▶ Antes do Oracle Database 11g, somente a notação posicional é suportado nas chamadas de SQL
- ▶ Iniciando em Oracle Database 11g, a notação nomeada e misturada podem ser usada para especificar argumentos em chamadas a sub rotinas PL/SQL de comandos SQL



# Passando Parâmetros Reais: Criando o Procedimento adiciona\_dept

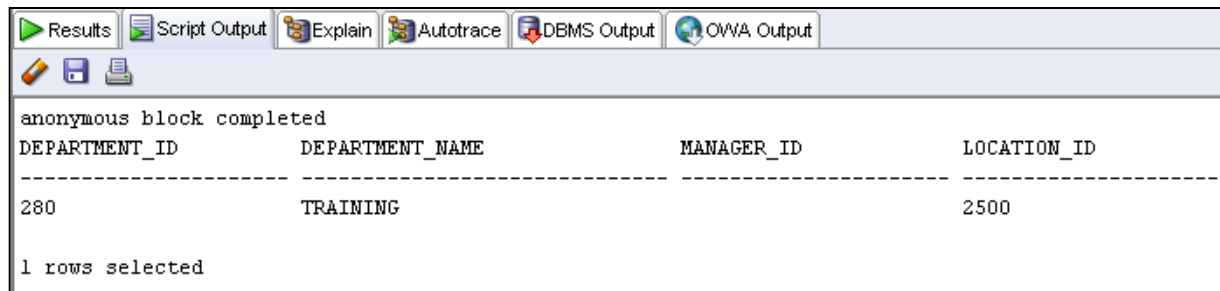
---

```
CREATE OR REPLACE PROCEDURE adiciona_dept(  
  p_name IN dept.dname%TYPE,  
  p_loc  IN deptloc%TYPE) IS  
BEGIN  
  INSERT INTO dept(deptno,  
                  dname, loc)  
  VALUES (dept_seq.NEXTVAL, p_name , p_loc );  
END adiciona_dept;  
/
```



# Passando Parâmetros Reais : Exemplos

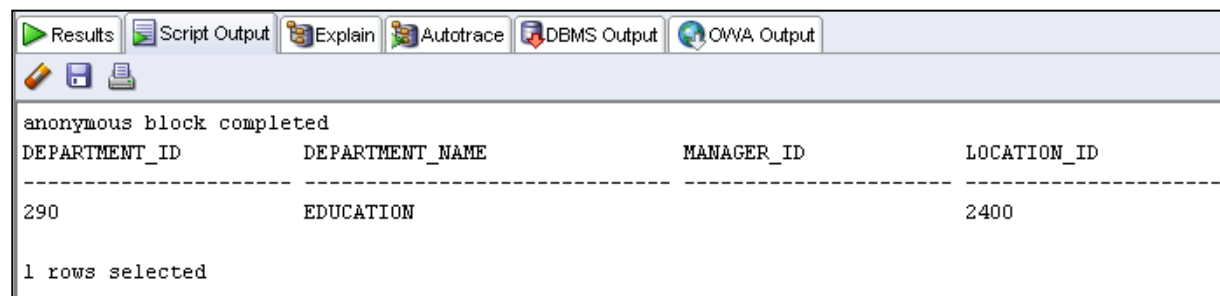
```
-- Passando parâmetros usando a notação posicional.  
EXECUTE adicione_dept ('TRAINING', 2500)
```



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING		2500

1 rows selected

```
-- Passando parâmetros usando a notação nomeada.  
EXECUTE adicione_dept (p_loc=>2400, p_name=>'EDUCATION')
```



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
290	EDUCATION		2400

1 rows selected

# Usando a Opção DEFAULT para os parâmetros

---

- ▶ Define valores default para os parâmetros
- ▶ Oferece flexibilidade combinando a sintaxe de passagem de parâmetros posicional e nomeada

```
CREATE OR REPLACE PROCEDURE add_dept(  
  p_name dept.dname%TYPE := 'Unknown',  
  p_loc  dept.loc%TYPE  DEFAULT 1700)  
IS  
BEGIN  
  INSERT INTO dept (deptno, dname, loc)  
  VALUES (dept_seq.NEXTVAL, p_name, p_loc);  
END add_dept;
```

```
EXECUTE add_dept  
EXECUTE add_dept ('ADVERTISING', p_loc => 1200)  
EXECUTE add_dept (p_loc => 1200)
```

# Chamando Procedimentos

- ▶ Você pode chamar procedimentos usando blocos anônimos, outro procedimento, ou pacotes.
- ▶ Você deve ser dono do procedimento ou ter o privilégio EXECUTE.

```
CREATE OR REPLACE PROCEDURE process_employees
IS
    CURSOR cur_emp_cursor IS
        SELECT empno
        FROM   emp;
BEGIN
    FOR emp_rec IN cur_emp_cursor
    LOOP
        raise_salary(emp_rec.empno, 10);
    END LOOP;
    COMMIT;
END process_employees;
/
```

PROCEDURE process\_employees Compiled.



# Tratamento de Exceções

---

Procedimento que chama

```
PROCEDURE  
  PROC1 ...  
IS  
  ...  
BEGIN  
  ...  
  PROC2 (arg1) ;  
  ...  
EXCEPTION  
  ...  
END PROC1;
```

Procedimento chamado

```
PROCEDURE  
  PROC2 ...  
IS  
  ...  
BEGIN  
  ...  
EXCEPTION  
  ...  
END PROC2;
```

← Exceção acontece

← Exceção tratada

O Controle retorna  
para o procedimento  
que chama



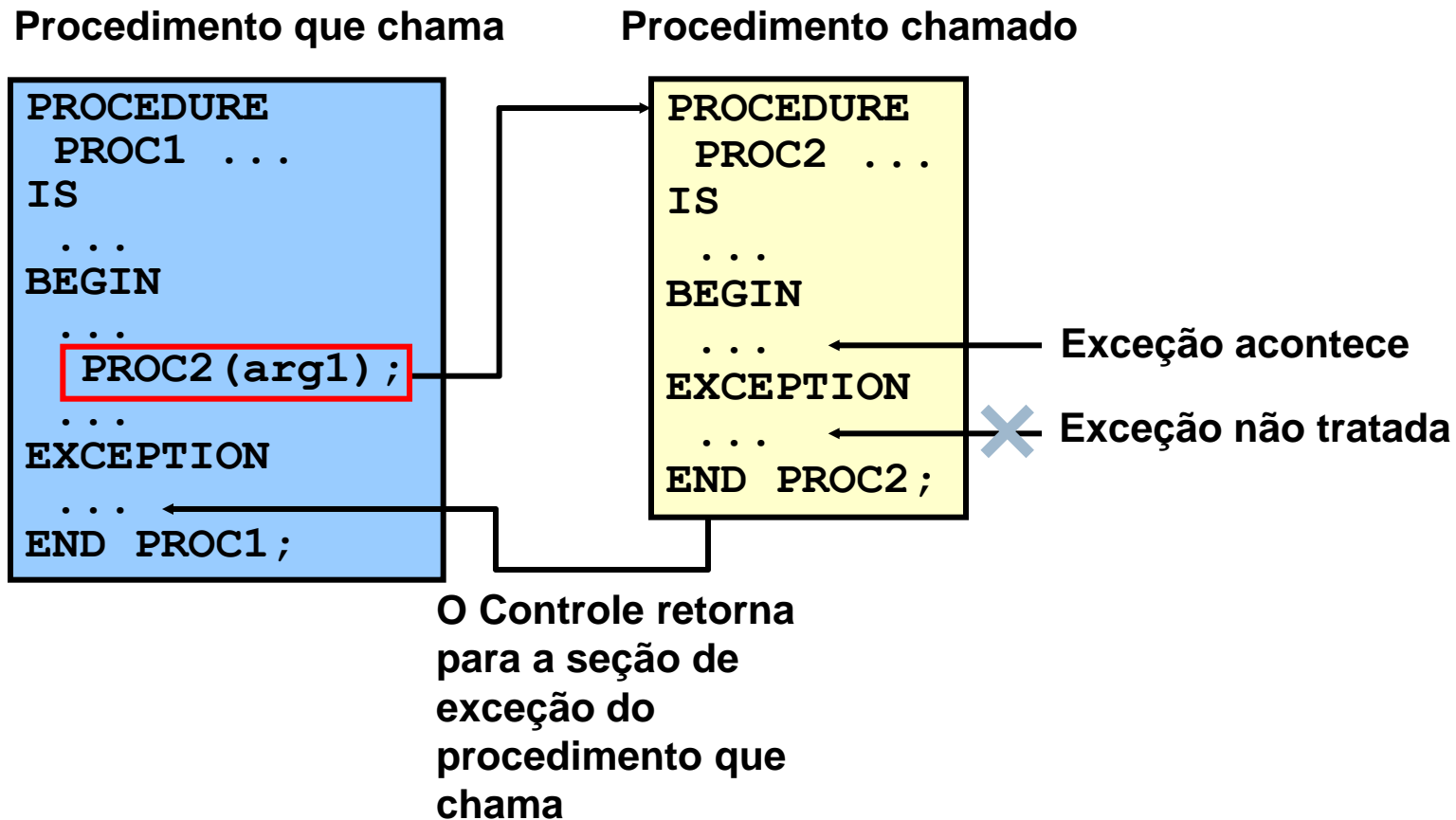
# Exceções Tratadas : Exemplo

```
CREATE PROCEDURE add_dept(  
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS  
BEGIN  
    INSERT INTO DEPT (deptno,dname, mgr, loc)  
    VALUES (DEPT_SEQ.NEXTVAL, p_name, p_mgr, p_loc);  
    DBMS_OUTPUT.PUT_LINE('Dept adicionado: '|| p_name);  
EXCEPTION  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('Err:adicionando dept:'|| p_name);  
END;
```

```
CREATE PROCEDURE cria_dept IS  
BEGIN  
    add_dept('Media', 100, 1800);  
    add_dept('Editing', 99, 1800);  
    add_dept('Advertising', 101, 1800);  
END;
```



# Exceções não tratadas



# Exceções Tratadas : Exemplo

---

```
SET SERVEROUTPUT ON
CREATE PROCEDURE add_dept(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
    INSERT INTO DEPT (deptno,dname, mgr, loc)
    VALUES (DEPT_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
    DBMS_OUTPUT.PUT_LINE('Dept adicionado: '|| p_name);
END;
```

```
CREATE PROCEDURE cria_dept IS
BEGIN
    add_dept('Media', 100, 1800);
    add_dept('Editing', 99, 1800);
    add_dept('Advertising', 101, 1800);
END;
```

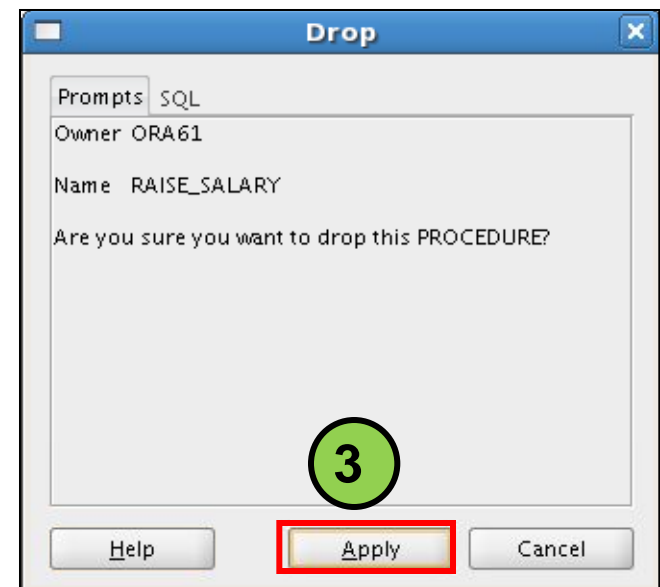
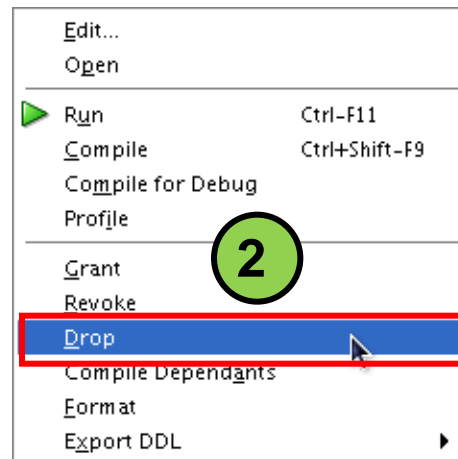
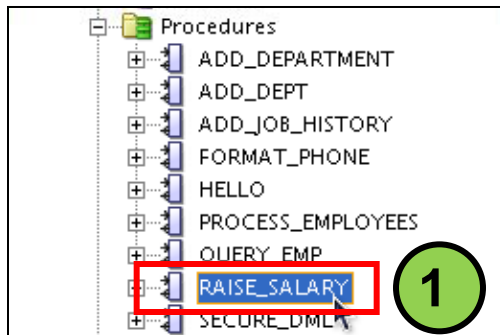


# Removendo Procedimentos: Usando o comando DROP ou SQL Developer

- ▶ Usando o comando DROP:

```
DROP PROCEDURE raise_salary;
```

- ▶ Usando SQL Developer:



# A Diferença Entre Procedimentos e Funções

Procedimentos	Funções
Executa como um comando PL/SQL	Chamada como parte de uma expressão
Não contem a cláusula <code>RETURN</code> no cabeçalho	Deve conter a cláusula <code>RETURN</code> no cabeçalho
Pode passar valores (se for necess.) usando parâmetros output	Deve retornar um valor simples
Pode conter um comando <code>RETURN</code> sem um valor	Deve conter no mínimo um comando <code>RETURN</code>



# Criando e chamando uma Função Armazenada

## Usando o Comando CREATE FUNCTION:

### Exemplo

---

```
CREATE OR REPLACE FUNCTION get_sal
(p_id emp.empno%TYPE) RETURN NUMBER IS
  v_sal emp.sal%TYPE := 0;
BEGIN
  SELECT sal
  INTO    v_sal
  FROM    emp
  WHERE   empno = 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
```

---

# 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 job_id, get_sal(empno)  
FROM emp;
```

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

