



# Sistemas de Informação

Prof. Me. Alexandre Barcelos  
profalexandre.barcelos@fiap.com.br



## Criação de Procedimentos

# Objetivos

- Descrever e criar um procedimento
- Create procedures with parameters
- Diferenciar parâmetros formais e reais
- Utilizar diferentes modos de passagem de parâmetros
- Invocar um procedimento
- Tratar exceções em um procedimento
- Remover um procedimento

## Objetivo da lição

Criar, executar e remover procedimentos com ou sem parâmetros. Os procedimentos são a base da programação modular em PL/SQL. Para tornar os procedimentos mais flexíveis, é importante que dados variáveis sejam calculados ou passados para um procedimento usando parâmetros de entrada. Os resultados calculados podem ser retornados ao ambiente de chamada de um procedimento utilizando os parâmetros OUT.

Para tornar seus programas robustos, você sempre deve gerenciar as condições de exceção usando os recursos de gerenciamento de exceção do PL/SQL.

## O que é um Procedimento?

- Um procedimento é:
  - É um tipo de subprograma que realiza uma ação
  - Pode ser armazenado no banco de dados como um objeto de esquema
  - Promove a reutilização e facilidade de manutenção

### Definição de um procedimento

Um procedimento é um bloco PL/SQL nomeado que pode aceitar parâmetros (também chamados de argumentos). Geralmente, você usa um procedimento para executar uma ação. Ele é composto por um cabeçalho, uma seção de declaração, uma seção executável e uma seção opcional de tratamento de exceções.

Um procedimento é invocado usando o nome do procedimento na seção de execução de outro bloco PL/SQL.

## Sintaxe de criação de procedimentos

- Utilize o comando CREATE PROCEDURE seguido do nome, parâmetros opcionais e a palavra-chave IS ou AS.
- Adicione a opção OR REPLACE para substituir um procedimento existente.
- Escreva um bloco PL/SQL contendo variáveis locais, BEGIN e END (ou END procedure\_name).

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter1 [mode] datatype1,
  parameter2 [mode] datatype2, ...)]
IS|AS
  [local_variable_declarations; ...]
BEGIN
  -- actions;
END [procedure_name];
```

→ Bloco PL/SQL

### Sintaxe para criar procedimentos

Você pode criar novos procedimentos com a instrução CREATE PROCEDURE, que pode declarar uma lista de parâmetros e deve definir as ações a serem executadas pelo bloco PL/SQL padrão. A cláusula CREATE permite que você crie procedimentos autônomos armazenados em um banco de dados Oracle.

- Os blocos PL/SQL começam com o BEGIN, opcionalmente precedido pela declaração de variáveis locais. Os blocos PL/SQL terminam com END ou END procedure\_name.
- A opção REPLACE indica que, se o procedimento existir, ele é descartado e substituído pela nova versão criada pela declaração.

### Outros elementos sintáticos

- O parâmetro 1 representa o nome de um parâmetro.
- A opção de modo define como um parâmetro é usado: IN (padrão), OUT ou IN OUT.
- Tipo de dados especifica o tipo de dados do parâmetro, sem qualquer precisão.

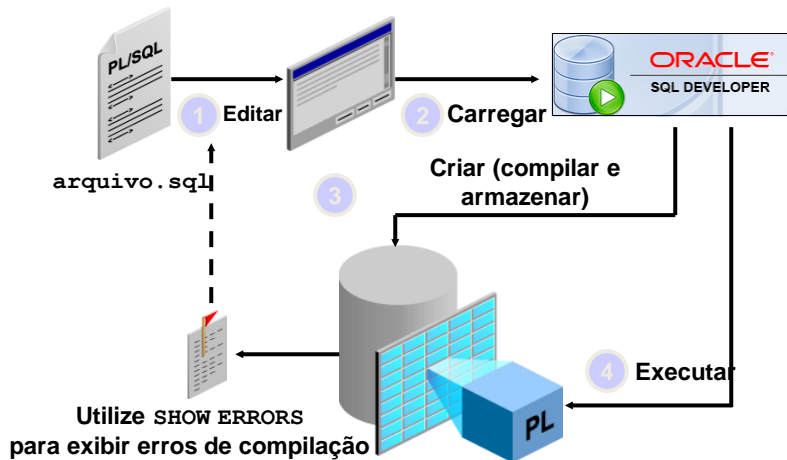
**Nota:** Os parâmetros podem ser considerados como variáveis locais. As variáveis substituição e host (bind) não podem ser referenciadas em qualquer lugar na definição de um procedimento PL/SQL armazenado.

A opção OR REPLACE não requer nenhuma alteração na segurança do objeto, desde que você possua o objeto e tenha o privilégio CREATE

[ANY] PROCEDURE.

.

# Desenvolvendo um Procedimento



## Procedimentos de desenvolvimento

Para desenvolver um procedimento armazenado, execute as seguintes etapas:

1. Escreva o código para criar um procedimento em um editor ou um processador de texto e, em seguida, guarde-o como um arquivo de script SQL (normalmente com uma extensão .sql).

2. Carregue o código em uma das ferramentas de desenvolvimento, como SQL\*Plus ou SQL Developer

3. Crie o procedimento no banco de dados. A instrução `CREATE PROCEDURE` compila e armazena o código-fonte e o código compilado no banco de dados. Se existem erros de compilação, o meta código não está armazenado e você deve editar o código-fonte para fazer correções. Não é possível invocar um procedimento que contenha erros de compilação. Para visualizar os erros de compilação no SQL\*Plus ou SQL DEVELOPER, use:  
`SHOW ERRORS` para o procedimento compilado mais recente (último)  
`SHOW ERRORS PROCEDURE procedure_name` para qualquer procedimento compilado anteriormente

4. Após a compilação bem-sucedida, execute o procedimento para executar a ação desejada. Use o comando `EXECUTE` ou um bloco PL/SQL anônimo a partir de ambientes que suportem PL/SQL.

Nota: Se ocorrerem erros de compilação, use uma instrução `CREATE OR REPLACE PROCEDURE` para substituir o código existente se você já usou uma instrução `CREATE PROCEDURE`. Caso contrário, `DROP` primeiro o procedimento e, em seguida, execute



a instrução CREATE PROCEDURE.

# O que são parâmetros?

## Parâmetros:

- São declarados após o nome do subprograma no cabeçalho PL/SQL
- Permite a passagem ou a comunicação de dados entre o ambiente de chamada e o subprograma
- São usados como variáveis locais, mas dependem do modo de passagem de parâmetros:
  - Um parâmetro IN (o padrão) fornece valores para um subprograma a ser processado.
  - Um parâmetro OUT retorna um valor ao chamador.
  - Um parâmetro IN OUT fornece um valor de entrada, que pode ser retornado (output) como um valor modificado

## O que são parâmetros?

Os parâmetros são usados para transferir valores de dados para e do ambiente de chamada e do procedimento (ou subprograma). Os parâmetros são declarados no cabeçalho do subprograma, após o nome e antes da seção de declaração para variáveis locais.

Os parâmetros estão sujeitos a um dos três modos de passagem de parâmetros: IN, OUT ou IN OUT.

- Um parâmetro IN passa um valor constante do ambiente de chamada para o procedimento.
- Um parâmetro OUT passa um valor do procedimento para o ambiente de chamada.
- Um parâmetro IN OUT passa um valor do ambiente de chamada para o procedimento e um valor possivelmente diferente do procedimento de volta para o ambiente de chamada usando o mesmo parâmetro.

Os parâmetros podem ser pensados como uma forma especial de variável local, cujos valores de entrada são inicializados pelo ambiente de chamada quando o subprograma é chamado e cujos valores de saída são retornados ao ambiente de chamada quando o subprograma retorna o controle ao chamador.

## Parâmetros formais e reais

- Parâmetros formais: variáveis locais declaradas na lista de parâmetros de uma especificação de subprograma
- Exemplo

```
CREATE PROCEDURE raise_sal(id NUMBER,sal NUMBER) IS  
BEGIN ...  
END raise_sal;
```

- Parâmetros reais: valores literais, variáveis ou expressões usadas na lista de parâmetros do subprograma chamado
- Exemplo:

```
emp_id := 100;  
raise_sal(emp_id, 2000)
```

### Parâmetros formais e reais

Parâmetros formais são variáveis locais que são declaradas na lista de parâmetros de uma especificação de subprograma. No primeiro exemplo, no procedimento `raise_sal`, os identificadores representam os parâmetros formais.

Os parâmetros reais podem ser valores literais, variáveis ou expressões que são fornecidos na lista de parâmetros de um subprograma. No segundo exemplo, uma chamada é feita para `raise_sal`, onde a variável `emp_id` fornece o valor do parâmetro real para o parâmetro `id` formal e 2000 é fornecido como o valor real do parâmetro para `sal`.

Parâmetros reais:

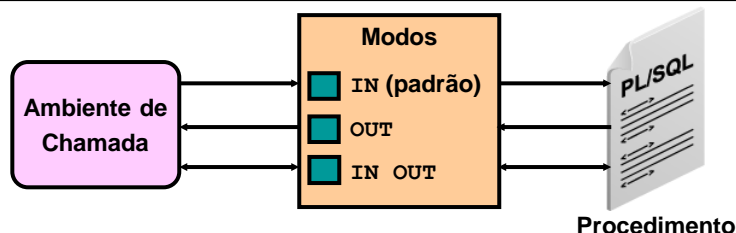
Estão associados a parâmetros formais durante a chamada do subprograma

Também pode ser expressões, como no exemplo a seguir: `Raise_sal(emp_id, raise + 100);`

## Modos de Parâmetros

- Os modos de parâmetro são especificados na declaração formal do parâmetro, após o nome do parâmetro e antes do seu tipo de dados.
- O modo IN é o padrão se nenhum modo for especificado

```
CREATE PROCEDURE procedure(param [mode] datatype)
...
```



### Modos de parâmetros processuais

Quando você cria o procedimento, o parâmetro formal define um nome de variável cujo valor é usado na seção executável do bloco PL/SQL. O parâmetro real é usado ao invocar o procedimento para fornecer valores de entrada ou receber resultados de saída.

O modo de parâmetro IN é o modo de passagem padrão. Ou seja, se nenhum modo for especificado com uma declaração de parâmetro, o parâmetro é considerado um parâmetro IN. Os modos de parâmetro OUT e IN OUT devem ser explicitamente especificados nas suas declarações de parâmetros.

O parâmetro do tipo de dados é especificado sem uma especificação de tamanho. Pode ser especificado:

- Como um tipo de dados explícito
- Usando a definição % TYPE
- Usando a definição de % ROWTYPE

**Nota:** Um ou mais parâmetros formais podem ser declarados, cada um separado por uma vírgula.

## Utilizando o Parâmetro IN

```
CREATE OR REPLACE PROCEDURE raise_salary
(id      IN employees.employee_id%TYPE,
 percent IN NUMBER)
IS
BEGIN
    UPDATE employees
    SET    salary = salary * (1 + percent/100)
    WHERE employee_id = id;
END raise_salary;
/
```

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

### Usando Parâmetros IN: Exemplo

O exemplo mostra um procedimento com dois parâmetros IN. Será criado o procedimento `raise_salary` no banco de dados. O segundo exemplo invoca `raise_salary` e fornece o primeiro valor de parâmetro de 176 para o ID do empregado e um aumento de salário percentual de 10% para o segundo valor do parâmetro.

Para invocar um procedimento, use o seguinte comando EXECUTE:

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

Para invocar um procedimento de outro procedimento, use uma chamada direta dentro de uma seção executável do bloco de chamada. No local de chamada do novo procedimento, digite o nome do procedimento e os parâmetros reais. Por exemplo:

```
...
BEGIN
    raise_salary (176, 10);
END;
```

**Nota:** os parâmetros IN são passados como valores somente leitura do ambiente de chamada no procedimento. Tentativas de alterar o valor de um resultado IN em um erro de compilação.

## Utilizando o Parâmetro OUT

```
CREATE OR REPLACE PROCEDURE query_emp
(id      IN  employees.employee_id%TYPE,
 name    OUT employees.last_name%TYPE,
 salary  OUT employees.salary%TYPE) IS
BEGIN
  SELECT  last_name, salary INTO name, salary
  FROM    employees
  WHERE   employee_id = id;
END query_emp;
```

```
DECLARE
  emp_name employees.last_name%TYPE;
  emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, emp_name, emp_sal); ...
END;
```

### Usando Parâmetros OUT: Exemplo

Neste exemplo, você cria um procedimento com parâmetros OUT para recuperar informações sobre um funcionário. O procedimento aceita o valor 171 para o ID do empregado e recupera o nome e o salário do empregado com ID 171 nos dois parâmetros OUT. O procedimento `query_emp` possui três parâmetros formais. Dois deles são parâmetros OUT que retornam valores para o ambiente de chamada, mostrado na caixa de código na parte inferior do slide. O procedimento aceita o valor de ID de um empregado através do parâmetro `id`. As variáveis `emp_name` e `emp_salary` são preenchidas com a informação obtida da consulta em seus dois parâmetros OUT correspondentes.

Se você imprimir os valores retornados nas variáveis P /SQL do bloco de chamada mostrado no segundo bloco de código, as variáveis contêm o seguinte:

- `Emp_name` mantém o valor Smith.
- `Emp_salary` possui o valor 7600.

**Nota:** Certifique-se de que o tipo de dados para as variáveis de parâmetro reais usadas para recuperar valores dos parâmetros OUT tenha um tamanho suficiente para manter os valores de dados retornados.

A tentativa de usar ou ler os parâmetros OUT dentro do procedimento que os declara resulta em um erro de compilação. Os parâmetros OUT podem ser atribuídos apenas valores no corpo do procedimento em que são declarados.

## Exibindo o conteúdo dos Parametros OUT

- Utilize o procedimento

DBMS\_OUTPUT.PUT\_LINE(DBMS\_OUTPUT.PUT\_LINE.

```
SET SERVEROUTPUT ON
DECLARE
  emp_name employees.last_name%TYPE;
  emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, emp_name, emp_sal);
  DBMS_OUTPUT.PUT_LINE('Name: ' || emp_name);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || emp_sal);
END;
```

- Use host variables, execute QUERY\_EMP usando host variables e imprima as host variables

```
VARIABLE name VARCHAR2(25)
VARIABLE sal  NUMBER
EXECUTE query_emp(171, :name, :sal)
PRINT name sal
```

## Chamando PL/SQL utilizando Host Variables

- Uma variável de host (também conhecida como uma):
- É declarado e existe externamente ao subprograma PL/SQL.
- Deve ser precedida por dois pontos (:) quando referenciada no código PL/SQL
- Pode ser referenciada em um bloco anônimo, mas não em um subprograma armazenado
- Fornece um valor para um bloco PL/SQL e recebe um valor de um bloco PL/SQL

Uma variável de host (também conhecida como variável de ligação)

É declarada e existe externamente ao subprograma PL/SQL

É precedido por dois pontos (:) quando referenciado no código PL / SQL

Pode ser referenciado em um bloco anônimo, mas não em um subprograma armazenado

Fornecer um valor para um bloco PL / SQL e receber um valor de um bloco PL / SQL



## Utilizando o Parâmetro IN OUT

### Ambiente de Chamada

Número do telefone(antes)	Número do telefone(após)
'8006330575'	'(800)633-0575'

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

### Usando Parâmetros IN OUT: Exemplo

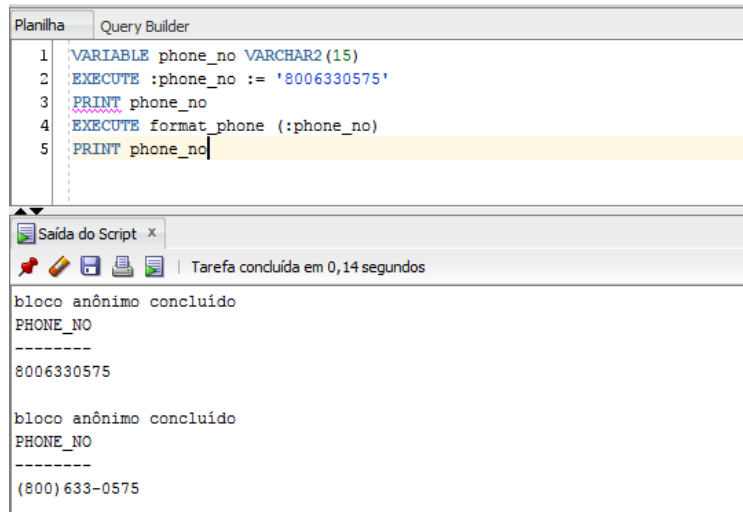
Usando um parâmetro IN OUT, você pode passar um valor para um procedimento que pode ser atualizado. O valor real do parâmetro fornecido a partir do ambiente de chamada pode retornar como um dos seguintes:

- O valor original inalterado

- Um novo valor que está definido no procedimento

**Nota:** Um parâmetro IN OUT atua como uma variável inicializada.

## Utilizando o Parâmetro IN OUT



The screenshot shows the SQL Developer interface. The top pane, titled 'Planilha Query Builder', contains a script with five lines: 1. `VARIABLE phone_no VARCHAR2(15)`, 2. `EXECUTE :phone_no := '8006330575'`, 3. `PRINT phone_no`, 4. `EXECUTE format_phone (:phone_no)`, and 5. `PRINT phone_no`. The bottom pane, titled 'Saída do Script', shows the execution results. It displays two blocks of output, each preceded by 'bloco anônimo concluído'. The first block shows 'PHONE\_NO' followed by a separator line and the value '8006330575'. The second block shows 'PHONE\_NO' followed by a separator line and the formatted value '(800) 633-0575'.

```
1 VARIABLE phone_no VARCHAR2(15)
2 EXECUTE :phone_no := '8006330575'
3 PRINT phone_no
4 EXECUTE format_phone (:phone_no)
5 PRINT phone_no
```

bloco anônimo concluído  
PHONE\_NO  
-----  
8006330575

bloco anônimo concluído  
PHONE\_NO  
-----  
(800) 633-0575

### Usando Parâmetros IN OUT: Exemplo

Usando um parâmetro IN OUT, você pode passar um valor para um procedimento que pode ser atualizado. O valor real do parâmetro fornecido a partir do ambiente de chamada pode retornar como um dos seguintes:

- O valor original inalterado

- Um novo valor que está definido no procedimento

**Nota:** Um parâmetro IN OUT atua como uma variável inicializada.

O exemplo no slide cria um procedimento com um parâmetro IN OUT para aceitar uma string de 10 caracteres contendo dígitos para um número de telefone. O procedimento retorna o número de telefone formatado com parênteses em torno dos três primeiros caracteres e um hífen após o sexto dígito - por exemplo, a sequência telefônica 8006330575 é retornada como (800) 633-0575.

O código a seguir usa a host variable `phone_no` para fornecer o valor de entrada passado para o procedimento `FORMAT_PHONE`. O procedimento é executado e retorna uma sequência atualizada na variável host do telefone.

```
VARIABLE phone_no VARCHAR2(15)
EXECUTE :phone_no := '8006330575'
PRINT phone_no
EXECUTE format_phone (:phone_no)
PRINT phone_no
```

# Sintaxe para a Passagem Parâmetros

## — Posicional:

- Lista os parâmetros reais na mesma ordem que os parâmetros formais

## — Nomeado:

- Lista os parâmetros reais em ordem arbitrária e usa o operador de associação (=>) para associar um parâmetro formal chamado com seu parâmetro real

## — Combinação:

- Lista alguns dos parâmetros reais como posicionais e alguns como nomeados

## Sintaxe para Passar Parâmetros

Para um procedimento que contém vários parâmetros, você pode usar uma série de métodos para especificar os valores dos parâmetros. Os métodos são:

Posicional, que lista os valores dos parâmetros reais na ordem em que os parâmetros formais são declarados

Nomeado, que lista os valores reais em ordem arbitrária e usa o operador de associação para associar cada parâmetro real com seu parâmetro formal por nome. O operador de associação PL / SQL é um sinal de igual seguido de um sinal maior do que o sinal, sem espaços: =>.

Combinação, que lista os primeiros valores dos parâmetros por sua posição, e o restante usando a sintaxe especial do método nomeado

# Passagem de Parâmetros

```
CREATE OR REPLACE PROCEDURE add_dept(  
    name IN departments.department_name%TYPE,  
    loc IN departments.location_id%TYPE) IS  
BEGIN  
    INSERT INTO departments(department_id,  
        department_name, location_id)  
    VALUES (departments_seq.NEXTVAL, name, loc);  
END add_dept;  
/
```

– Passagem por notação posicioal

```
EXECUTE add_dept ('TRAINING', 2500)
```

– Passagem por notação nomeada

```
EXECUTE add_dept (loc=>2400, name=>'EDUCATION')
```

Passagem de parâmetros: exemplos

No exemplo, o procedimento `add_dept` declara dois parâmetros IN: nome e loc. Os valores desses parâmetros são usados na instrução `INSERT` para definir as colunas `department_name` e `location_id`, respectivamente.

Passar parâmetros por posição é mostrado na primeira chamada para executar `add_dept` abaixo da definição do procedimento. O primeiro parâmetro real fornece o valor 'TREINAMENTO' para o parâmetro de nome. O segundo valor real do parâmetro de 2500 é atribuído por posição ao parâmetro loc.

Passar parâmetros usando a notação nomeada é mostrado no último exemplo. Aqui, o parâmetro loc, que é declarado como o segundo parâmetro formal, é referenciado por nome na chamada, onde está associado ao valor real de 2400. O parâmetro de nome está associado ao valor 'EDUCAÇÃO'. A ordem dos parâmetros reais é irrelevante se todos os valores dos parâmetros forem especificados.

## Utilizando a opção DEFAULT para Parâmetros

- Definir valores padrão para parâmetros

```
CREATE OR REPLACE PROCEDURE add_dept(  
  name departments.department_name%TYPE := 'Unknown' ,  
  loc  departments.location_id%TYPE DEFAULT 1700)  
IS  
BEGIN  
  INSERT INTO departments (...)  
  VALUES (departments_seq.NEXTVAL, name, loc);  
END add_dept;
```

- Fornece flexibilidade ao combinar a sintaxe de passagem de parâmetro e nomeada:

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

### Utilizando a opção DEFAULT para parâmetros

Os exemplos de mostram duas maneiras de atribuir um valor padrão a um parâmetro IN. As duas formas mostradas usam:

O operador de atribuição (: =), como mostrado para o parâmetro de nome

A opção DEFAULT, conforme mostrado para o parâmetro loc

Quando os valores padrão são atribuídos a parâmetros formais, você pode chamar o procedimento sem fornecer um valor de parâmetro real para o parâmetro. Assim, você pode passar diferentes números de parâmetros reais para um subprograma, seja aceitando ou substituindo os valores padrão, conforme necessário. É recomendável que você declare parâmetros sem valores padrão primeiro. Então, você pode adicionar parâmetros formais com valores padrão sem ter que mudar todas as chamadas para o procedimento.

**Nota:** Você não pode atribuir valores padrão aos parâmetros OUT e IN OUT.

O slide mostra três maneiras de invocar o procedimento add\_dept:

O primeiro exemplo atribui os valores padrão para cada parâmetro.

O segundo exemplo ilustra uma combinação de posição e notação designada para atribuir valores. Nesse caso, o uso da notação nomeada é apresentado como um exemplo.

O último exemplo usa o valor padrão para o parâmetro nome e o valor fornecido para o parâmetro loc.

### Usando a opção DEFAULT para parâmetros (continuação)

Geralmente, você pode usar a notação nomeada para substituir os valores padrão dos parâmetros formais. No entanto, você não pode pular fornecendo um parâmetro real se não houver um valor padrão fornecido para um parâmetro formal.

**Nota:** Todos os parâmetros posicionais devem preceder os parâmetros nomeados em uma chamada de subprograma. Caso contrário, você receberá uma mensagem de erro, como mostrado no exemplo a seguir:

```
EXECUTE add_dept(name=>'new dept', 'new location')
```

A seguinte mensagem de erro é gerada:

```
ERROR at line 1:
```

```
ORA-06550: line 1, column 7:
```

```
PLS-00306: wrong number or types of arguments in call to  
'ADD_DEPT'
```

```
ORA-06550: line 1, column 7:
```

```
PL/SQL: Statement ignored
```

## Resumo dos modos de parâmetro

IN	OUT	IN OUT
Modo padrão	Deve ser especificado	Deve ser especificado
O valor é passado para o subprograma	Retornado ao ambiente de chamada	Passa para o subprograma; Retorna ao ambiente de chamada
O parâmetro formal atua como uma constante	Variável não inicializada	Variável inicializada
Parâmetro formal atua como uma constante	Deve ser uma variável	Deve ser uma variável
Pode ser atribuído um valor padrão	Não pode ser atribuído um valor padrão	Não pode ser atribuído um valor padrão

# Invocando Procedimentos

Você pode invocar parâmetros por:

- Usando blocos anônimos
- Usando outro procedimento, conforme o exemplo:

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

## Invocando Procedimentos

Você pode invocar procedimentos usando:

Blocos anônimos

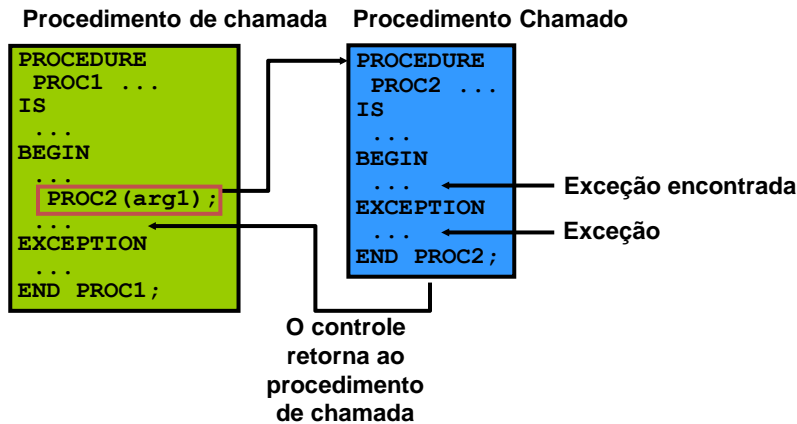
Outro procedimento ou subprograma PL / SQL

Exemplos nas páginas anteriores mostraram como usar blocos anônimos (ou o comando EXECUTE no iSQL \* Plus).

Este exemplo mostra como invocar um procedimento de outro procedimento armazenado. O procedimento armazenado PROCESS\_EMPS usa um cursor para processar todos os registros na tabela EMPREGADOS e passa a identificação de cada empregado para o procedimento RAISE\_SALARY, o que resulta em um aumento salarial de 10% em toda a empresa.



# Manipulando Exceções



## Manipulando Exceções

Quando você desenvolve procedimentos que são chamados de outros procedimentos, você deve estar ciente dos efeitos que as exceções manipuladas e não tratadas têm na transação e no procedimento de chamada.

Quando uma exceção é gerada em um procedimento chamado, o controle imediatamente vai para a seção de exceção desse bloco. Uma exceção é considerada tratada se a seção de exceção fornecer um manipulador para a exceção levantada.

Quando ocorre uma exceção e é tratada, ocorre o seguinte fluxo de código:

1. A exceção é aumentada.
2. O controle é transferido para o manipulador de exceções.
3. O bloco é encerrado.
4. O programa / bloco de chamada continua a executar como se nada acontecesse.

Se uma transação foi iniciada (ou seja, se qualquer declaração de linguagem de manipulação de dados [DML] executada antes de executar o procedimento no qual a exceção foi levantada), a transação não é afetada. Uma operação DML é revertida se for realizada dentro do procedimento antes da exceção.

**Nota:** Você pode terminar explicitamente uma transação executando uma operação COMMIT ou ROLLBACK na seção de exceção.

## Manipulando Exceções: Exemplo

```
CREATE PROCEDURE add_department(  
    name VARCHAR2, mgr NUMBER, loc NUMBER) IS  
BEGIN  
    INSERT INTO DEPARTMENTS (department_id,  
        department_name, manager_id, location_id)  
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, name, mgr, loc);  
    DBMS_OUTPUT.PUT_LINE('Added Dept: ' || name);  
EXCEPTION  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('Err: adding dept: ' || name);  
END;
```

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

### Exceções Manipuladas: Exemplo

Os dois procedimentos no exemplo são os seguintes:

O procedimento `add_department` cria um novo registro de departamento alocando um novo número de departamento de uma seqüência Oracle e define os valores de coluna `department_name`, `manager_id` e `location_id` usando os parâmetros `name`, `mgr` e `loc`, respectivamente.

O procedimento `create_departments` cria mais de um departamento usando chamadas para o procedimento `add_department`.

O procedimento `add_department` captura todas as exceções levantadas em seu próprio manipulador. Quando `create_departments` é executado, a seguinte saída é gerada:

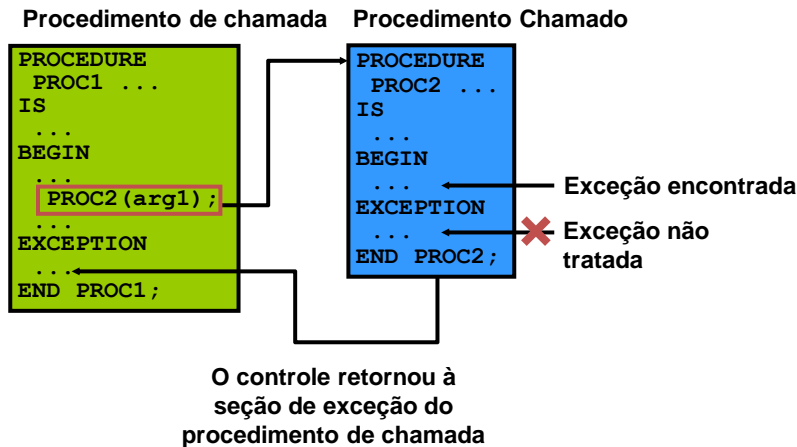
Departamento Adicional: Mídia

Err: Adicionando departamento: edição

Departamento Adicional: Publicidade

O departamento de edição com `manager_id` de 99 não está inserido por causa de uma violação de restrição de integridade de chave estrangeira no `administrador_id`. Como a exceção foi tratada no procedimento `add_department`, o procedimento `create_department` continua a ser executado. Uma consulta na tabela `DEPARTMENTS` onde o `location_id` é 1800 mostra que `Media` e `Advertising` são adicionados, mas não o registro de edição.

## Exceções não tratadas



### Exceptions Not Handled

As discussed, when an exception is raised in a called procedure, control immediately goes to the exception section of that block. If the exception section does not provide a handler for the raised exception, then it is not handled. The following code flow occurs:

1. The exception is raised.
2. The block terminates because no exception handler exists; any DML operations performed within the procedure are rolled back.
3. The exception propagates to the exception section of the calling procedure. That is, control is returned to the exception section of the calling block, if one exists.

If an exception is not handled, then all the DML statements in the calling procedure and the called procedure are rolled back along with any changes to any host variables. The DML statements that are not affected are statements that were executed before calling the PL/SQL code whose exceptions are not handled.

### Exceções não tratadas

Conforme discutido, quando uma exceção é gerada em um procedimento, o controle imediatamente vai para a seção de exceção desse bloco. Se a seção de exceção não fornecer um manipulador para a exceção levantada, então não é

tratada. O seguinte fluxo de código ocorre:

1. A exceção é encontrada.
2. O bloco termina porque nenhum manipulador de exceção existe; Todas as operações DML realizadas dentro do procedimento são revertidas.
3. A exceção se propaga para a seção de exceção do procedimento de chamada. Ou seja, o controle é retornado para a seção de exceção do bloco de chamada, se existir.

Se uma exceção não for tratada, todas as instruções DML no procedimento de chamada e o procedimento chamado são revertidas, juntamente com qualquer alteração em qualquer variável do host. As instruções DML que não são afetadas são instruções que foram executadas antes de chamar o código PL / SQL cujas exceções não são tratadas.

## Exceções não tratadas

```
CREATE PROCEDURE add_department_noex(  
    name VARCHAR2, mgr NUMBER, loc NUMBER) IS  
BEGIN  
    INSERT INTO DEPARTMENTS (department_id,  
        department_name, manager_id, location_id)  
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, name, mgr, loc);  
    DBMS_OUTPUT.PUT_LINE('Added Dept: '||name);  
END;
```

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

### Exceções não tratadas: Exemplo

O exemplo de código no slide mostra `add_department_noex`, que não possui uma seção de exceção. Nesse caso, a exceção ocorre quando o departamento de Edição é adicionado. Devido à falta de manipulação de exceção em qualquer um dos subprogramas, nenhum novo registro de departamento é adicionado à tabela `DEPARTMENTS`. Executar o procedimento `create_departments_noex` produz um resultado semelhante ao seguinte:

Added Dept: Media

BEGIN create\_departments\_noex; END;

\*

ERROR at line 1:

ORA-02291: integrity constraint (ORA1.DEPT\_MGR\_FK)  
violated - parent key not

found

ORA-06512: at "ORA1.ADD\_DEPARTMENT\_NOEX", line  
4

ORA-06512: at "ORA1.CREATE\_DEPARTMENTS\_NOEX",  
line 4

ORA-06512: at line 1

Embora os resultados mostrem que o departamento de mídia foi adicionado, sua operação é revertida porque a exceção não foi tratada em nenhum dos

subprogramas invocados.

## Removendo Procedimento

- Você pode remover um procedimento do banco de dados com a instrução DROP

– Sintaxe:

```
DROP PROCEDURE procedure_name
```

– Exemplo:

```
DROP PROCEDURE raise_salary;
```

### Removendo Procedimento

Quando um procedimento armazenado não é mais necessário, você pode usar a instrução SQL DROP PROCEDURE para removê-lo.

## Benefícios dos subprogramas

- Fácil manutenção
- Melhoria da segurança e integridade dos dados
- Performance melhorada
- Melhoria da clareza do código

### Benefícios de subprogramas

Procedimentos e funções têm muitos benefícios devido à modularização do código:

É realizada uma fácil manutenção porque os subprogramas estão localizados em um só lugar. As modificações só precisam ser feitas em um único lugar para afetar múltiplas aplicações e minimizar testes excessivos.

A segurança aprimorada dos dados pode ser conseguida controlando o acesso indireto a objetos de banco de dados de usuários não privilegiados com privilégios de segurança. Os subprogramas executados são com o direito do definidor por padrão. O privilégio de execução não permite que um usuário de chamada tenha acesso direto a objetos acessíveis ao subprograma.

A integridade dos dados é gerenciada por ações relacionadas realizadas juntas ou não.

O desempenho aprimorado pode ser realizado a partir da reutilização do código PL / SQL analisado que fica disponível na área SQL compartilhada do servidor. Chamadas subsequentes para o subprograma evitam analisar novamente o código. Como o código PL / SQL é analisado no tempo de compilação, a sobrecarga de análise das instruções SQL é evitada no tempo de execução. O código pode ser



escrito para reduzir o número de chamadas de rede para o banco de dados e, portanto, diminuir o tráfego de rede.

A clareza melhorada do código pode ser alcançada usando nomes e convenções apropriados para descrever a ação das rotinas, reduzindo a necessidade de comentários e aprimorando a clareza do código.

OBRIGADO



profalexandre.barcelos@fiap.com.br



<https://www.linkedin.com/in/alexandrebarcelos>

FIAP

Copyright © 2023 | Professor Me. Alexandre Barcelos  
Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente  
proibido sem consentimento formal, por escrito, do professor/autor.

