

FIAP GRADUAÇÃO

Tecnologia em Análise e Desenvolvimento de Sistemas

Application Development For Databases

PROF. MILTON

Criando Procedures

Objetivos

Ao concluir esta lição, você será capaz de:

- Identificar os benefícios do design de subprogramas modularizados e em camadas
- Criar e chamar procedures
- Usar parâmetros formais e reais
- Usar notação posicional, nomeada ou combinada para especificar parâmetros
- Identificar os modos disponíveis de especificar parâmetros
- Tratar exceções em procedures
- Remover um procedure
- Exibir as informações dos procedures

Objetivo da Lição

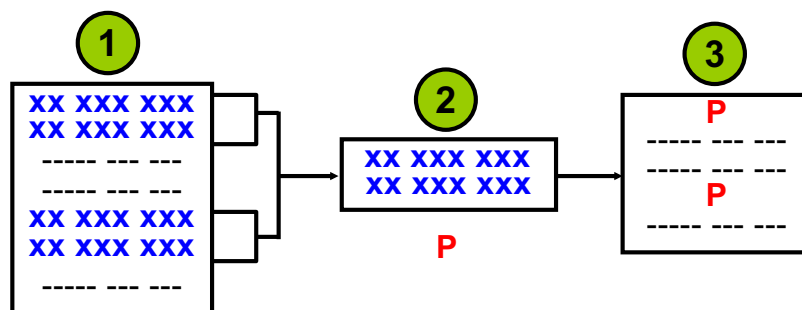
Nesta lição, você aprenderá a criar, executar e remover procedures com ou sem parâmetros. Os procedures são a base da programação modular em PL/SQL. Para tornar os procedures mais flexíveis, é importante que dados variáveis sejam calculados ou especificados em um procedure por meio de parâmetros de entrada. Os resultados calculados podem ser retornados ao chamador do procedure por meio de parâmetros OUT.

Para que seus programas sejam mais resistentes a falhas, você sempre deve gerenciar condições de exceção usando os recursos de tratamento de exceções da linguagem PL/SQL.

Agenda de Lições

- Usando um design de subprograma modularizado e em camadas e identificando os benefícios de subprogramas
- Trabalhando com procedures:
 - Criando e chamando procedures
 - Identificando os modos disponíveis de especificar parâmetros
 - Usando parâmetros formais e reais
 - Usando notação posicional, nomeada ou combinada
- Tratando exceções em procedures, removendo um procedure e exibindo as informações dos procedures

Criando um Design de Subprograma Modularizado



Modularize o código em subprogramas.

1. Localize sequências de código repetidas mais de uma vez.
2. Crie o subprograma P contendo o código repetitivo
3. Modifique o código original para chamar o novo subprograma.

Criando um Projeto de Subprograma em Camadas e Modularizado

O diagrama ilustra o princípio de modularização com subprogramas: a criação de partes gerenciáveis menores de código flexível e reutilizável. A flexibilidade é obtida com o uso de subprogramas com parâmetros, o que, por sua vez, torna o mesmo código reutilizável para valores de entrada diferentes. Para modularizar o código existente, execute as seguintes etapas:

1. Localize e identifique as sequências de código repetitivas.
2. Mova o código repetitivo para um subprograma PL/SQL.
3. Substitua o código repetitivo original por chamadas para o novo subprograma PL/SQL.

A adoção dessa abordagem modular e em camadas pode ajudá-lo a criar códigos de manutenção mais fácil, especialmente quando as regras de negócios forem alteradas. Além disso, a adoção de uma lógica SQL simples e livre de negócios complexos pode ser facilitada pelo otimizador do Oracle Database, que pode reutilizar as instruções SQL submetidas a parse para melhorar o uso dos recursos do servidor.

Criando um Design de Subprograma em Camadas

Crie camadas de subprograma para a aplicação.

- Camada de subprograma para acesso a dados com lógica SQL
- Camada de subprograma para lógica de negócios, que pode usar ou não a camada de acesso a dados

Criando um Design de Subprograma em Camadas

Como a linguagem PL/SQL permite incorporar facilmente instruções SQL à lógica, é muito fácil ter instruções SQL distribuídas por todo o código. Entretanto, recomenda-se manter a lógica SQL separada da lógica de negócios, ou seja, criar um projeto de aplicação em camadas com, no mínimo, duas camadas:

- **Camada de acesso a dados:** Para que as sub-rotinas acessem os dados usando instruções SQL
- **Camada de lógica de negócios:** Para que os subprogramas implementem as regras de processamento de negócios, que podem ou não chamar as rotinas da camada de acesso a dados.

Modularizando o Desenvolvimento com Blocos PL/SQL

- PL/SQL é uma linguagem estruturada em blocos. O bloco PL/SQL ajuda a modularizar o código usando:
 - Blocos anônimos
 - Procedures e funções
 - Pacotes
 - Triggers de banco de dados
- As vantagens do uso de estruturas de programas modulares são:
 - Manutenção fácil
 - Maior segurança e integridade dos dados
 - Melhor desempenho
 - Maior clareza do código

Modularizando o Desenvolvimento com Blocos PL/SQL

Um subprograma baseia-se em estruturas PL/SQL padrão. Ele contém uma seção declarativa, uma seção executável e uma seção opcional de tratamento de exceções (por exemplo, blocos anônimos, procedures, funções, pacotes e triggers). Os subprogramas podem ser compilados e armazenados no banco de dados, fornecendo modularidade, extensibilidade, bem como capacidade de manutenção e reutilização.

A modularização converte blocos grandes de código em grupos de código menores denominados módulos. Após a modularização, os módulos podem ser reutilizados pelo mesmo programa ou compartilhados com outros programas. É mais fácil manter e depurar um código composto de módulos menores do que um código contido em um único programa grande. Os módulos podem ser facilmente expandidos para personalização, incorporando mais funcionalidade, se necessário, sem afetar os demais módulos do programa.

Os subprogramas permitem fácil manutenção porque o código se encontra em apenas um local e, portanto, as modificações necessárias no subprograma podem ser executadas apenas nesse local. Os subprogramas garantem maior segurança e integridade dos dados. Os objetos de dados são acessados por meio do subprograma, e o usuário só poderá chamar o subprograma se tiver recebido o privilégio de acesso adequado.

Blocos Anônimos: Visão Geral

Blocos anônimos:

- Formam a estrutura básica dos blocos PL/SQL
- Iniciam as tarefas de processamento PL/SQL em aplicações
- Podem estar aninhados na seção executável de qualquer bloco PL/SQL

```
[DECLARE      -- Declaration Section (Opcional)
  variable declarations; ... ]
BEGIN         -- Executable Section (Obrigatório)
  SQL or PL/SQL statements;
[EXCEPTION   -- Exception Section (Opcional)
  WHEN exception THEN statements; ]
END;         -- End of Block (Obrigatório)
```

Blocos Anônimos: Visão Geral

Os blocos anônimos geralmente são usados para:

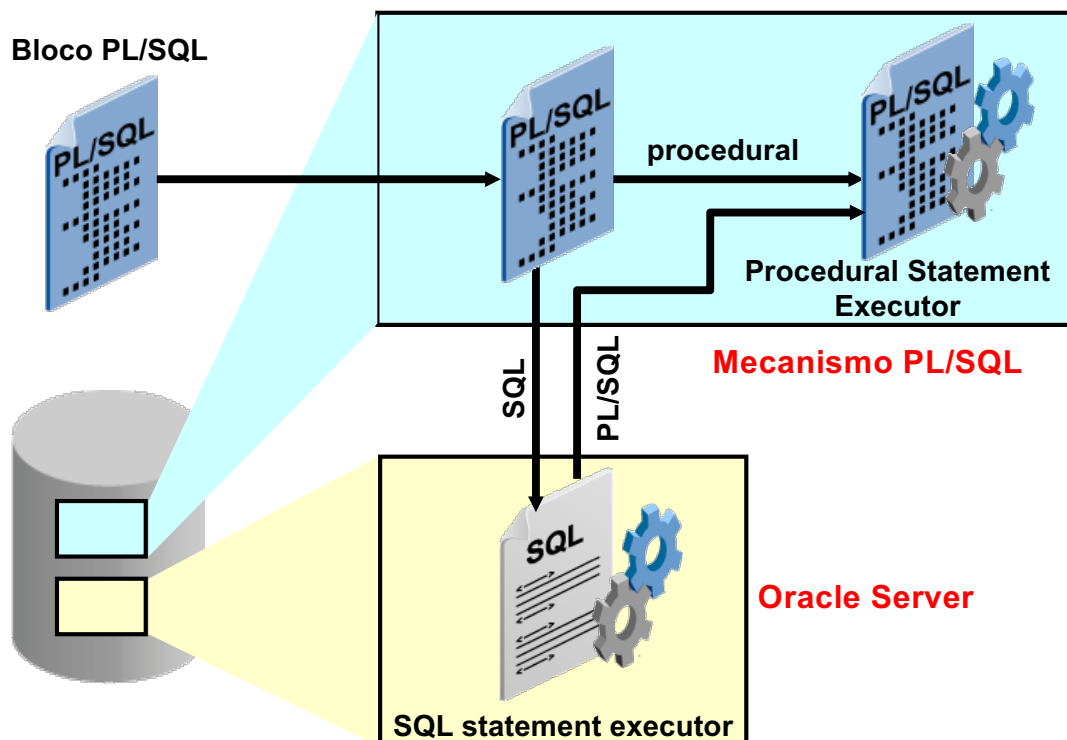
- Criar código de trigger para os componentes do Oracle Forms
- Iniciar chamadas a procedures, funções e estruturas de pacote
- Isolar o tratamento de exceções em um bloco de código
- Gerenciar o controle do fluxo de código por meio de seu aninhamento em outros blocos PL/SQL

A palavra-chave DECLARE é opcional, mas se tornará obrigatória se você declarar variáveis, constantes e exceções para serem usadas no bloco PL/SQL.

BEGIN e END são obrigatórias e exigem que exista entre elas pelo menos uma instrução, seja SQL, PL/SQL ou ambas.

A seção de exceções é opcional e é usada para tratar os erros que ocorrem no escopo do bloco PL/SQL. As exceções podem ser propagadas para o chamador do bloco anônimo por meio da exclusão de um handler de exceções para a exceção específica, criando, assim, uma exceção *não tratada*.

Arquitetura de Runtime de PL/SQL



Arquitetura de Runtime de PL/SQL

O diagrama mostra um bloco PL/SQL que está sendo executado pelo mecanismo PL/SQL. Esse mecanismo reside:

- No banco de dados Oracle para executar subprogramas armazenados
- No Oracle Forms Client, quando estiver executando aplicações cliente/servidor, ou no Oracle Application Server, quando estiver usando o Oracle Forms Services para executar o Forms na Web

Independentemente do ambiente de runtime PL/SQL, a arquitetura básica permanece a mesma. Portanto, todas as instruções PL/SQL são processadas no Procedural Statement Executor, e todas as instruções SQL devem ser enviadas ao SQL Statement Executor para serem processadas pelos processos do servidor Oracle. O ambiente SQL também pode chamar o ambiente PL/SQL quando uma função for usada em uma instrução select.

O mecanismo PL/SQL é uma máquina virtual residente na memória que processa as instruções m-code PL/SQL. Quando o mecanismo PL/SQL encontra uma instrução SQL, ocorre uma alternância de contexto a fim de especificar a instrução SQL para os processos do servidor Oracle. O mecanismo PL/SQL aguarda o término da instrução SQL e o retorno dos resultados para depois continuar processando as instruções subsequentes do bloco PL/SQL. O mecanismo PL/SQL do Oracle Forms é executado no cliente para a implementação de cliente/servidor e no servidor de aplicações para a implementação do Forms Services. Em ambos os casos, as instruções SQL geralmente são enviadas pela rede para processamento em um servidor Oracle.

O que São os Subprogramas PL/SQL?

- Um subprograma PL/SQL é um bloco PL/SQL nomeado que pode ser chamado com um conjunto de parâmetros.
- Você pode declarar e definir um subprograma de duas formas: ou em um bloco PL/SQL ou em outro subprograma.
- Um subprograma consiste em uma especificação e um corpo.
- Um subprograma pode ser um procedure ou uma função.
- Em geral, o procedure é usado para realizar uma ação e a função para calcular e retornar um valor.
- Os subprogramas pode ser agrupados em pacotes PL/SQL.



O que São os Subprogramas PL/SQL?

Um subprograma PL/SQL é um bloco PL/SQL nomeado que pode ser chamado com um conjunto de parâmetros. Você pode declarar e definir um subprograma de duas formas: ou em um bloco PL/SQL ou em outro subprograma.

Partes do Subprograma

Um subprograma consiste em uma especificação (spec) e um corpo. Para declarar um subprograma, é preciso fornecer a especificação, que inclui descrições de quaisquer parâmetros. Para definir um subprograma, é preciso fornecer tanto a especificação como o corpo. É possível declarar um subprograma primeiro e defini-lo depois no mesmo bloco ou subprograma ou declará-lo e defini-lo ao mesmo tempo.

Tipos de Subprograma

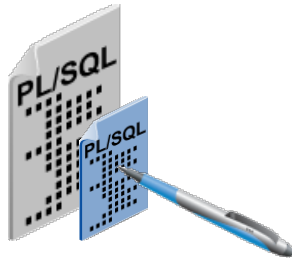
Um código PL/SQL tem dois tipos de subprogramas: procedures e funções. Em geral, o procedure é usado para realizar uma ação e a função para calcular e retornar um valor.

Um procedure e uma função têm a mesma estrutura, exceto que apenas uma função tem alguns itens adicionais como a cláusula RETURN ou a instrução RETURN.

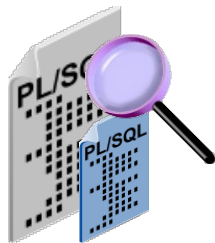
A cláusula RETURN especifica o tipo de dados do valor de retorno (obrigatório). A instrução RETURN especifica o valor de retorno (obrigatório). As funções são abordadas com mais detalhes na próxima lição intitulada “Criando Funções e Depurando Subprogramas.”

Os subprogramas podem ser agrupados em pacotes PL/SQL, que tornam o código ainda mais reutilizável. Os pacotes são abordados nas lições sobre pacotes.

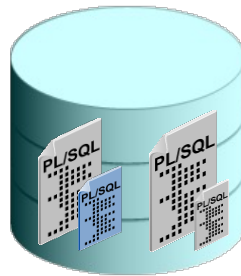
Os Benefícios de Usar os Subprogramas PL/SQL



Manutenção fácil



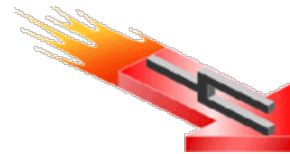
Maior clareza do código



**Subprogramas:
Procedures e funções
armazenados**



**Maior segurança e
integridade dos dados**



Melhor desempenho

Vantagens dos Subprogramas

Os procedures e as funções oferecem várias vantagens por causa da modularização do código:

- **A manutenção fácil** é possível porque os subprogramas estão localizados em um só lugar. As modificações precisam ser feitas apenas em um local, de modo a afetar várias aplicações e minimizar o número de testes.
- **A maior segurança dos dados** pode ser obtida por meio do controle do acesso indireto aos objetos do banco de dados por usuários não privilegiados com privilégios de segurança. Por default, os subprogramas são executados com os direitos do definidor. O privilégio EXECUTE não permite ao usuário que faz a chamada o acesso direto a objetos que podem ser acessados pelo subprograma.
- **A integridade dos dados** é gerenciada com a execução em conjunto de ações relacionadas ou não.
- **O melhor desempenho** pode ser alcançado com a reutilização do código PL/SQL submetido a parse, o qual é disponibilizado na área SQL compartilhada do servidor. As chamadas subsequentes ao subprograma evitam que o código seja submetido a parse novamente. Como o código PL/SQL é submetido a parse no momento da compilação, o overhead de parse das instruções SQL é evitado durante o runtime. O código pode ser escrito para diminuir o número de chamadas de rede feitas ao banco de dados e, portanto, reduzir o tráfego da rede.
- **A maior clareza do código** pode ser obtida com a utilização de convenções e de nomes apropriados para descrever a ação das rotinas, o que reduz a necessidade de comentários e aumenta a clareza do código.

Diferenças entre Blocos Anônimos e Subprogramas

Blocos Anônimos	Subprogramas
Blocos PL/SQL não nomeados	Blocos PL/SQL nomeados
Compilados todas as vezes	Compilados apenas uma vez
Não armazenados no banco de dados	Armazenados no banco de dados
Não podem ser chamados por outra aplicação	São nomeados e, portanto, podem ser chamados por outras aplicações
Não retornam valores	Funções chamadas por subprogramas devem retornar valores
Não podem aceitar parâmetros	Podem aceitar parâmetros

Diferenças entre Blocos Anônimos e Subprogramas

A tabela do slide não mostra apenas as diferenças entre blocos anônimos e subprogramas, mas também destaca as vantagens dos subprogramas.

Os blocos anônimos não são objetos de banco de dados persistentes. Eles são compilados e executados

apenas uma vez. Eles não são armazenados no banco de dados para reutilização. Se quiser reutilizá-los, você deverá executar novamente o script que cria o bloco anônimo, o que causará a recompilação e a execução.

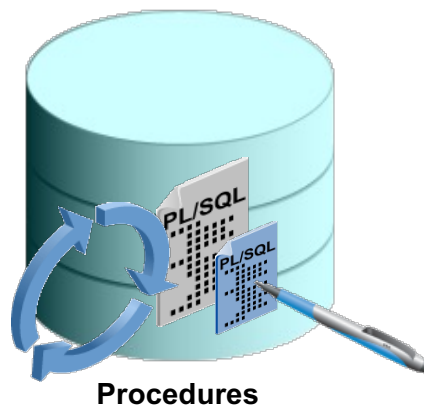
Funções e procedures são compilados e armazenados no banco de dados em formato compilado. Eles serão recompilados apenas se forem modificados. Como as funções e os procedures são armazenados no banco de dados, qualquer aplicação pode usar esses subprogramas com base em permissões apropriadas. A aplicação que está fazendo a chamada poderá especificar parâmetros para os procedures se eles tiverem sido projetados para aceitar parâmetros. Da mesma forma, uma aplicação que está fazendo uma chamada poderá recuperar um valor se chamar uma função ou um procedure.

Agenda de Lições

- Usando um design de subprograma modularizado e em camadas e identificando os benefícios de subprogramas
- Trabalhando com procedures:
 - Criando e chamando procedures
 - Identificando os modos disponíveis de especificar parâmetros
 - Usando parâmetros formais e reais
 - Usando notação posicional, nomeada ou combinada
- Tratando exceções em procedures, removendo um procedure e exibindo as informações dos procedures

O Que São Procedures?

- É um tipo de subprograma que executa uma ação
- Pode ser armazenado no banco de dados como um objeto de esquema
- Promove a capacidade de reutilização e manutenção



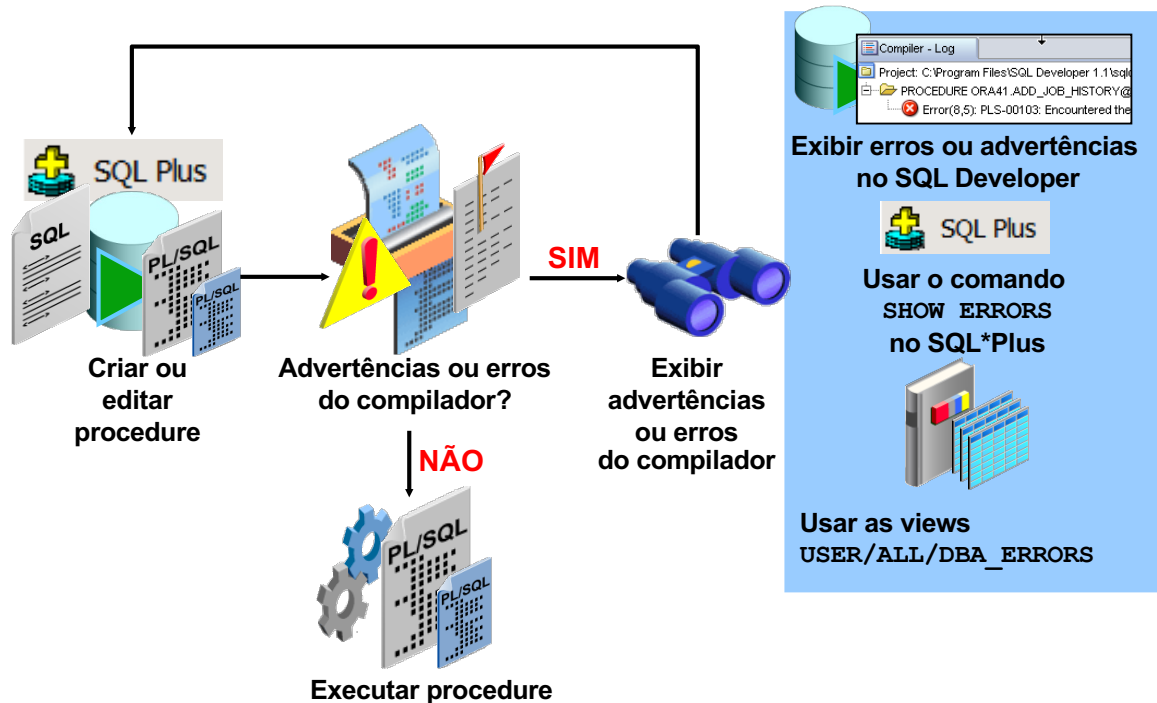
Definição de Procedure

Um procedure é um bloco PL/SQL nomeado que pode aceitar parâmetros (conhecidos como argumentos). Geralmente, os procedures são utilizados para executar uma ação. Eles contêm um cabeçalho, uma seção declarativa, uma seção executável e uma seção opcional de tratamento de exceções. O procedure é chamado quando o seu nome é utilizado na seção executável de outro bloco PL/SQL.

Os procedures são compilados e armazenados no banco de dados como objetos de esquema. Se você estiver usando procedures com o Oracle Forms e o Oracle Reports, esses procedures poderão ser compilados nos arquivos executáveis desses sistemas.

Os procedures promovem a capacidade de manutenção e reutilização. Quando validadas, elas podem ser usadas em várias aplicações. Se as necessidades mudarem, somente o procedure precisará ser atualizado.

Criando Procedures: Visão Geral



Criando Procedures: Visão Geral

Para desenvolver um procedure usando uma ferramenta como o SQL Developer, faça o seguinte:

1. Crie o procedure usando a árvore Object Navigation do SQL Developer ou a área SQL Worksheet.
2. Compile o procedure. O procedure é criado no banco de dados e é compilado. A instrução `CREATE PROCEDURE` compila e armazena no banco de dados o código-fonte e o *m-code* compilado. Para compilar o procedure, clique com o botão direito do mouse no nome do procedure na árvore Object Navigator e depois clique em Compile.
3. Se houver erros de compilação, o *m-code* não será armazenado, e você terá de editar o código-fonte para fazer as correções. Não é possível chamar um procedure que contém erros de compilação. É possível exibir os erros de compilação no SQL Developer, SQL*Plus, ou nas views apropriadas do dicionário de dados, conforme mostrado no slide.
4. Após o término bem-sucedido da compilação, execute o procedure para realizar a ação desejada. Você pode executar o procedure usando o SQL Developer ou o comando `EXECUTE` no SQL*Plus.

Observação: Se ocorrerem erros durante a compilação, use uma instrução `CREATE OR REPLACE PROCEDURE` para sobregravar o código existente, se você tiver usado anteriormente uma instrução `CREATE PROCEDURE`. Caso contrário, primeiro elimine o procedure com o comando (usando `DROP`) e, em seguida, execute a instrução `CREATE PROCEDURE`.

Criando Procedures com a Instrução SQL CREATE OR REPLACE

- Use a cláusula `CREATE` para criar um procedure standalone que é armazenado no banco de dados Oracle.
- Adicione a opção `OR REPLACE` para sobregravar um procedure existente.

```
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

Criando Procedures com a Instrução SQL CREATE OU REPLACE

Você pode usar a instrução SQL `CREATE PROCEDURE` para criar procedures standalone que são armazenados em um banco de dados Oracle. Um procedure é similar a uma miniatura de programa: ele executa uma ação específica. Você especifica o nome do procedure, seus parâmetros, suas variáveis locais e o bloco `BEGIN-END` que contém seu código e trata qualquer exceção.

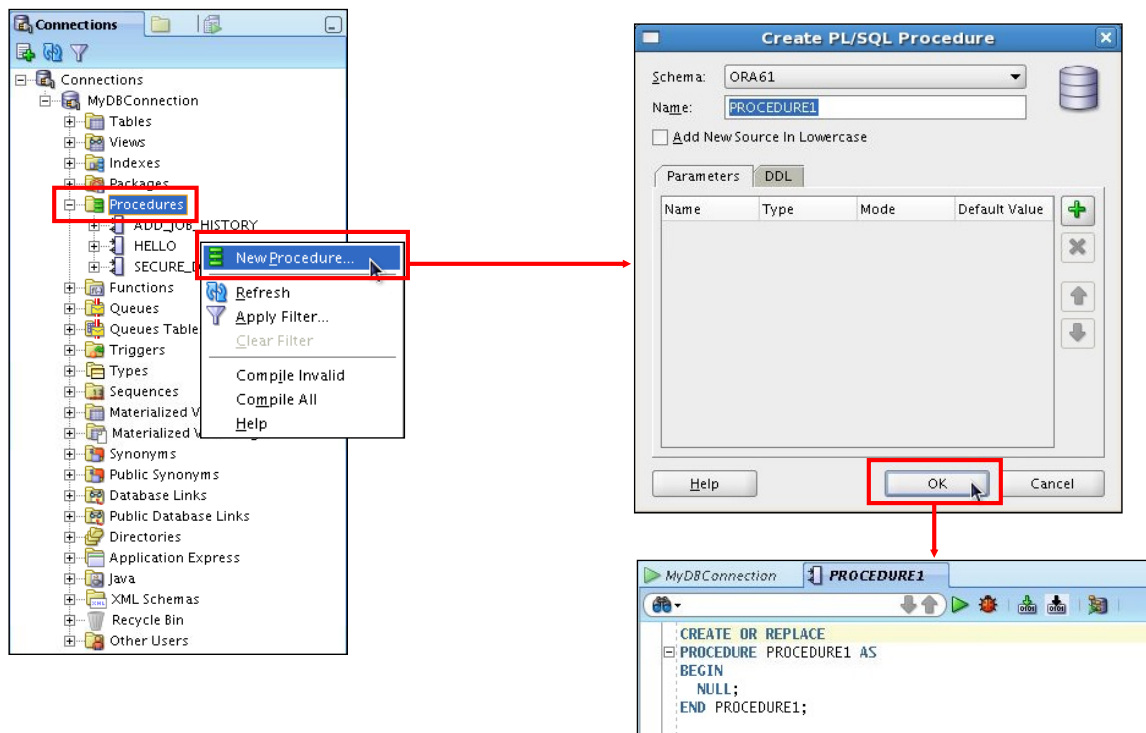
- Os blocos PL/SQL começam com uma instrução `BEGIN`, podendo ser precedidos da declaração de variáveis locais, e terminam com uma instrução `END` ou `END nome_do_procedure`.
- A opção `REPLACE` indica que, se o procedure existir, ele será eliminado e substituído pela nova versão criada pela instrução. A opção `REPLACE` não elimina nenhum dos privilégios associados ao procedure.

Outros Elementos Sintáticos

- *parâmetro1* representa o nome de um parâmetro.
- A opção *modo* define o modo de utilização do parâmetro: `IN` (default), `OUT` ou `IN OUT`.
- *tipodedados1* especifica o tipo de dados do parâmetro, sem nenhuma precisão.

Observação: Os parâmetros podem ser considerados como variáveis locais. As variáveis de substituição e de host (bind) não podem ser referenciadas em nenhuma parte da definição de um procedure PL/SQL armazenado. A opção `OR REPLACE` não requer alteração na segurança do objeto, desde que você seja o proprietário do objeto e tenha o privilégio `CREATE [QUALQUER] PROCEDURE`.

Criando Procedures Usando o SQL Developer



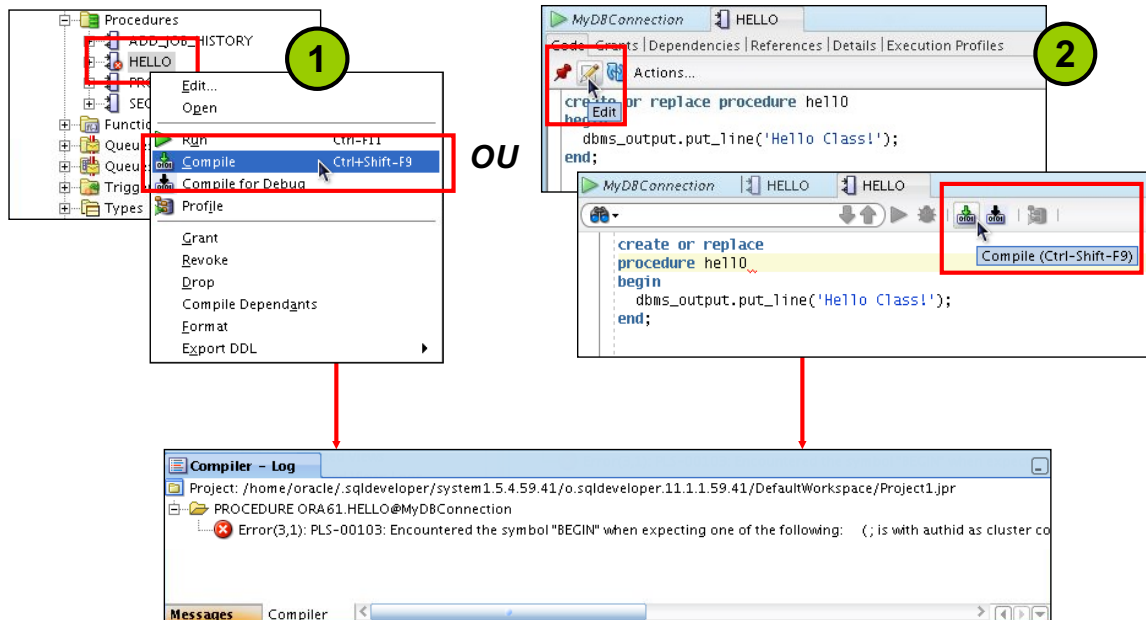
Criando Procedures Usando o SQL Developer

1. Clique com o botão direito do mouse no nó **Procedures** na página com a tab **Connections**.
2. Selecione **New Procedure** no menu de atalho. A caixa de diálogo **Create PL/SQL Procedure** é exibida. Especifique a informação para o novo procedure e clique em OK para criar o subprograma e exibi-lo na janela do editor, onde você pode inserir os detalhes.

Os componentes da caixa de diálogo **Create PL/SQL Procedure** são os seguintes:

- **Schema:** O esquema de banco de dados no qual criar o subprograma PL/SQL
- **Name:** O nome do subprograma que deve ser exclusivo em um esquema
- **Add New Source in Lowercase:** Se essa opção for selecionada, será exibido novo texto em minúsculas, independentemente do fato de você ter inserido letras maiúsculas ou minúsculas. Esta opção afeta somente a aparência do código, pois o código PL/SQL não faz distinção entre maiúsculas e minúsculas na sua execução.
- **Tab Parameters:** Para adicionar um parâmetro, clique no ícone Add (+). Em cada parâmetro no procedure a ser criado, especifique o nome do parâmetro, tipo de dado, modo e, opcionalmente, o Value default. Use o ícone Remove (X) e os ícones de seta para deletar e mover um parâmetro para cima ou para baixo na lista, respectivamente.
- **Tab DDL:** Esta tab contém uma exibição somente para leitura de uma instrução SQL que reflete a definição atual do subprograma.

Compilando Procedures e Exibindo Erros de Compilação no SQL Developer

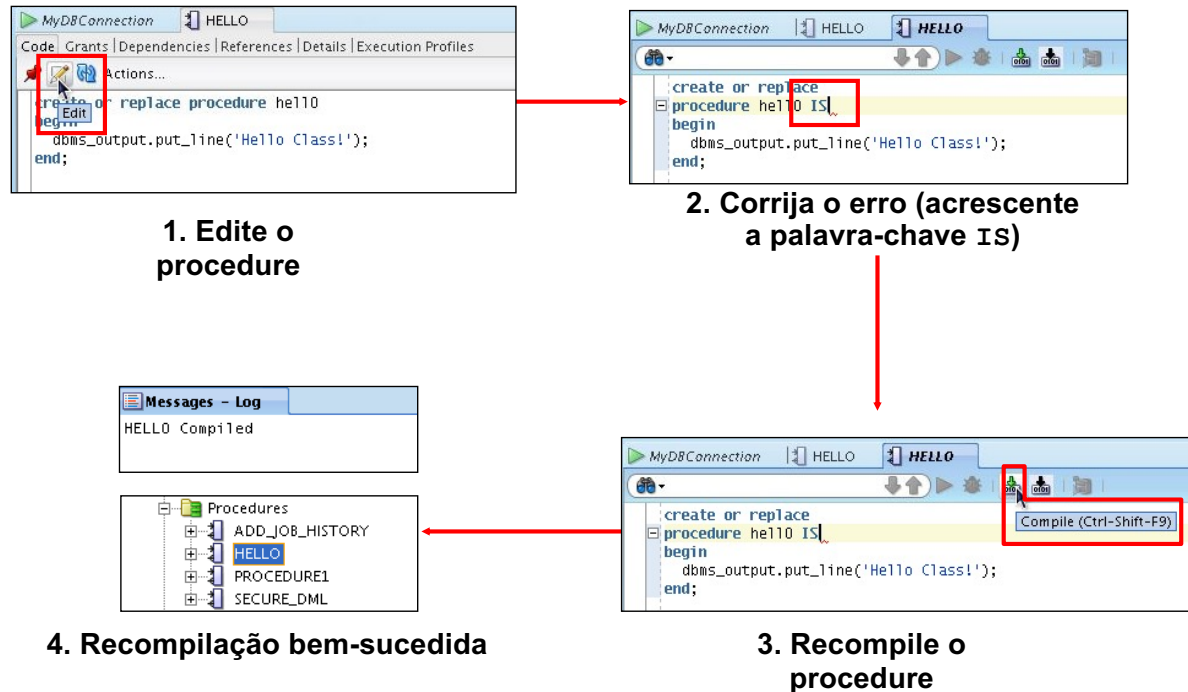


Compilando Procedures e Exibindo Erros de Compilação no SQL Developer

É possível compilar procedures usando um dos seguintes métodos:

- Navegue até o nó Procedures na árvore Object Navigator. Clique com o botão direito do mouse no nome do procedure e, em seguida, selecione Compile no nome do atalho. Para exibir qualquer mensagem de compilação, exiba a subtab Messages na tab **Compiler – Log**.
- Edite o procedure usando o ícone Edit na barra de ferramentas do código do procedure. Faça as edições necessárias e depois clique no ícone Compile na barra de ferramentas do código. Para exibir qualquer mensagem de compilação, exiba a subtab Messages na tab **Compiler – Log**.

Corrigindo Erros de Compilação no SQL Developer



Corrigindo Erros de Compilação no SQL Developer

1. Edite o procedure usando o ícone Edit na barra de ferramentas do código do procedure. Uma nova tab de código do procedure é aberta no modo Read/Write.
2. Faça as correções necessárias.
3. Clique no ícone Compile da barra de ferramentas do código.
4. Para exibir qualquer mensagem de compilação, exiba a subtab Messages na tab **Compiler – Log**. Além disso, se o procedure foi compilado com sucesso, o X vermelho no nome do procedure na árvore Object Navigator é removido.

Convenções de Nomeação das Estruturas PL/SQL Usadas Neste Curso

Estrutura PL/SQL	Convenção	Exemplo
Variável	v_ <i>variable_name</i>	v_rate
Constante	c_ <i>constant_name</i>	c_rate
Parâmetro de subprograma	p_ <i>parameter_name</i>	p_id
Variável de bind (host)	b_ <i>bind_name</i>	b_salary
Cursor	cur_ <i>cursor_name</i>	cur_emp
Registro	rec_ <i>record_name</i>	rec_emp
Tipo	type_ <i>name_type</i>	ename_table_type
Exceção	e_ <i>exception_name</i>	e_products_invalid
Handle de arquivo	f_ <i>file_handle_name</i>	f_file

Convenções de Nomeação das Estruturas PL/SQL Usadas Neste Curso

A tabela do slide exibe alguns exemplos das convenções de nomeação das estruturas PL/SQL usadas neste curso.

O Que São Parâmetros e Modos de Parâmetros?

- São declarados após o nome do subprograma no cabeçalho PL/SQL
- Especificam ou transmitem dados entre o chamador e o subprograma
- São usados como variáveis locais, mas dependem do modo como são especificados:
 - Um parâmetro `IN` (default) fornece valores a serem processados por um subprograma.
 - Um parâmetro `OUT` retorna um valor para o chamador.
 - Um parâmetro `IN OUT` fornece um valor de entrada, que pode ser retornado (saída) como um valor modificado.

O que São Parâmetros?

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

Eles estão sujeitos a um dos três modos de especificação de parâmetros: `IN`, `OUT` ou `IN OUT`.

- Um parâmetro `IN` especifica um valor constante do ambiente de chamada para o procedure.
- Um parâmetro `OUT` especifica um valor do procedure para o ambiente de chamada.
- Um parâmetro `IN OUT` especifica um valor do ambiente de chamada para o procedure, e um valor possivelmente diferente é retornado do procedure para esse ambiente por meio do mesmo parâmetro.

Os parâmetros podem ser considerados 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 para esse ambiente quando o subprograma retorna o controle ao chamador.

Parâmetros Formais e Reais

- Parâmetros formais: Variáveis locais declaradas na lista de parâmetros da especificação de um subprograma
- Parâmetros reais (ou argumentos): Expressões, variáveis ou valores literais usados na lista de parâmetros do subprograma que fez a chamada

```
-- Procedure definition, Formal parameters
CREATE PROCEDURE raise_sal(p_id NUMBER, p_sal NUMBER) IS
BEGIN
. . .
END raise_sal;

-- Procedure calling, Actual parameters (arguments)
v_emp_id := 100;
raise_sal(v_emp_id, 2000)
```

Parâmetros Formais e Reais

Os parâmetros formais são variáveis locais declaradas na lista de parâmetros da especificação de um subprograma. No primeiro exemplo, no procedure `raise_sal`, os identificadores de variável `p_id` e `p_sal` representam os parâmetros formais.

Os parâmetros reais podem ser expressões, variáveis ou valores literais fornecidos na lista de parâmetros do subprograma que fez a chamada. No segundo exemplo, é feita uma chamada a `raise_sal`, em que a variável `v_emp_id` fornece o valor do parâmetro real para o parâmetro formal `p_id` e `2000` é fornecido como o valor do parâmetro real para `p_sal`. Os parâmetros reais:

- São associados aos parâmetros formais durante a chamada do subprograma
- Também podem ser expressões, como no seguinte exemplo:
`raise_sal(v_emp_id, raise+100);`

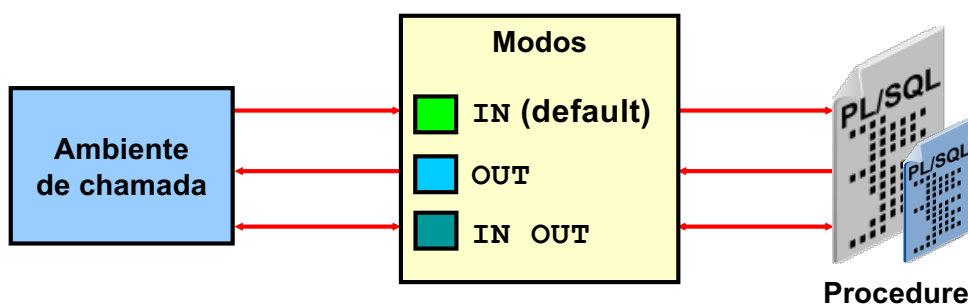
Os parâmetros formais e reais devem ter tipos de dados compatíveis. Se necessário, antes de designar o valor, o código PL/SQL converterá o tipo de dados do valor do parâmetro real no tipo de dados do parâmetro formal.

Observação: Parâmetros reais também são conhecidos como *argumentos reais*.

Modos de Parâmetro Procedurais

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

```
CREATE PROCEDURE proc_name(param_name [mode] datatype)  
...
```



Modos de Parâmetro Procedurais

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

O modo de parâmetro `IN` é o modo de especificação default, ou seja, se nenhum modo for especificado na declaração, o parâmetro será considerado como um parâmetro `IN`. Os modos de parâmetro `OUT` e `IN OUT` devem ser especificados explicitamente nas declarações de parâmetro.

O parâmetro *datatype* é definido sem uma especificação de tamanho. Ele pode ser especificado:

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

Observação: É possível declarar um ou mais parâmetros formais, separados por uma vírgula.

Comparando os Modos de Parâmetro

IN	OUT	IN OUT
Modo default	Deve ser especificado	Deve ser especificado
O valor é especificado no subprograma	O valor é retornado para o ambiente de chamada	Valor especificado para o subprograma; valor retornado para o ambiente de chamada
O parâmetro formal funciona como uma constante	Variável não inicializada	Variável inicializada
O parâmetro real pode ser um literal, uma expressão, uma constante ou uma variável inicializada	Deve ser uma variável	Deve ser uma variável
Pode receber um valor default	Não pode receber um valor default	Não pode receber um valor default

Comparando os Modos de Parâmetro

O modo de parâmetro **IN** será o default se nenhum modo for especificado na declaração. Os modos de parâmetro **OUT** e **IN OUT** devem ser especificados explicitamente na declaração do parâmetro.

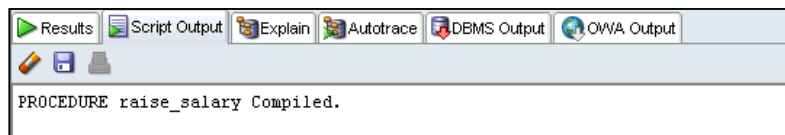
Um parâmetro formal do modo **IN** não pode ter um valor designado nem ser modificado no corpo do procedure. Por default, o parâmetro **IN** é especificado por referência. É possível designar um valor default para um parâmetro **IN** na declaração do parâmetro formal; nesse caso, o chamador não precisará fornecer um valor para o parâmetro se o default se aplicar.

É necessário designar um valor para o parâmetro **OUT** ou **IN OUT** antes de retorná-lo para o ambiente de chamada. Não é possível designar valores default para os parâmetros **OUT** e **IN OUT**. Para melhorar o desempenho com os parâmetros **OUT** e **IN OUT**, a dica de compilador **NOCOPY** poderá ser usada para solicitar a especificação por referência.

Observação: O uso de **NOCOPY** será abordado mais adiante neste curso.

Usando o Modo de Parâmetro IN: Exemplo

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



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

Usando Parâmetros IN: Exemplo

O exemplo no slide mostra um procedure com dois parâmetros IN. Executar o exemplo do primeiro slide cria o procedure `raise_salary` no banco de dados. O segundo exemplo chama o procedure `raise_salary` e fornece o primeiro valor de parâmetro 176 para o ID de funcionário e um percentual de aumento salarial de 10% para o segundo valor de parâmetro.

Para chamar um procedure usando a planilha SQL do SQL Developer ou usando o SQL*Plus, use o seguinte comando EXECUTE exibido no segundo exemplo de código no slide.

Para chamar um procedure a partir de outro, use uma chamada direta dentro de uma seção executável do bloco que faz a chamada. No local da chamada do novo procedure, informe o nome do procedure e os parâmetros reais. Por exemplo:

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

Observação: Os parâmetros IN são especificados como valores somente para leitura do ambiente de chamada para o procedure. As tentativas de alteração do valor do resultado de um parâmetro IN resultarão em erros de compilação.

Usando o Modo de Parâmetro OUT: Exemplo

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

```
SET SERVEROUTPUT ON
DECLARE
    v_emp_name employees.last_name%TYPE;
    v_emp_sal  employees.salary%TYPE;
BEGIN
    query_emp(171, v_emp_name, v_emp_sal);
    DBMS_OUTPUT.PUT_LINE(v_emp_name||' earns '||
        to_char(v_emp_sal, '$999,999.00'));
END;
/
```

Usando os Parâmetros OUT: Exemplo

No exemplo do slide, você cria um procedure com parâmetros OUT para recuperar informações sobre um funcionário. O procedure aceita o valor 171 para o ID de funcionário e recupera o nome e o salário do funcionário de ID 171 nos dois parâmetros OUT. O procedure `query_emp` tem três parâmetros formais. Dois deles são parâmetros OUT que retornam valores para o ambiente de chamada, conforme mostrado na segunda caixa de código no slide. O procedure aceita um valor de ID de funcionário por meio do parâmetro `p_id`. As variáveis `v_emp_name` e `v_emp_salary` são preenchidas com as informações recuperadas da consulta feita a seus dois parâmetros correspondentes OUT. Então este é o resultado de executar o código no segundo exemplo de código no slide. `v_emp_name` contém o valor Smith e `v_emp_salary` contém o valor 7400:



Observação: Verifique se o tamanho do tipo de dados das variáveis dos parâmetros reais usadas para recuperar os valores dos parâmetros OUT é suficiente para armazenar os valores de dados retornados.

Usando o Modo de Parâmetro IN OUT: Exemplo

Ambiente de chamada



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

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

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

Usando os Parâmetros IN OUT: Exemplo

Usando um parâmetro IN OUT, é possível especificar um valor para um procedure que pode ser atualizado. O valor real do parâmetro fornecido pelo ambiente de chamada pode retornar ou o valor não alterado original ou um novo valor que é definido no procedure.

Observação: Um parâmetro IN OUT funciona como uma variável inicializada.

O exemplo do slide cria um procedure com um parâmetro IN OUT para aceitar uma string de 10 caracteres contendo os dígitos de um número de telefone. O procedure retorna o número de telefone formatado com os três primeiros caracteres entre parênteses e um hífen após o sexto dígito; por exemplo, a string de telefone 8006330575 é retornada como (800) 633-0575.

O código a seguir usa a variável de host b_phone_no do SQL*Plus para fornecer o valor de entrada especificado para o procedure FORMAT_PHONE. O procedure é executado e retorna uma string atualizada na variável de host b_phone_no. A saída do seguinte código é exibida no slide acima:

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

Exibindo os Parâmetros OUT : Usando a Sub-rotina DBMS_OUTPUT.PUT_LINE

Use as variáveis PL/SQL que são exibidas na tela com chamadas ao procedure DBMS_OUTPUT.PUT_LINE.

```
SET SERVEROUTPUT ON

DECLARE
  v_emp_name employees.last_name%TYPE;
  v_emp_sal   employees.salary%TYPE;
BEGIN
  query_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE('Name: ' || v_emp_name);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || v_emp_sal);
END;
```

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

Exibindo os Parâmetros OUT: Usando a Sub-rotina DBMS_OUTPUT

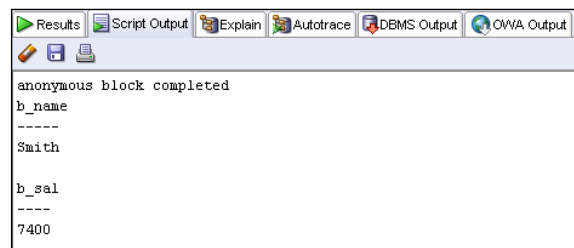
O exemplo do slide mostra como exibir os valores retornados dos parâmetros OUT no SQL*Plus ou na planilha do SQL Developer.

Você pode utilizar variáveis PL/SQL em um bloco anônimo para recuperar os valores do parâmetro OUT. O procedure DBMS_OUTPUT.PUT_LINE é chamado para imprimir os valores armazenados nas variáveis do código PL/SQL. O comando SET SERVEROUTPUT deve ser ON.

Exibindo Parâmetros OUT: Usando as Variáveis de Host do SQL*Plus

1. Use as variáveis de host do SQL*Plus.
2. Execute o procedure `QUERY_EMP` com essas variáveis.
3. Exiba-as na tela com o comando `PRINT`.

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



The screenshot shows the SQL*Plus interface with the following output:

```
anonymous block completed
b_name
-----
Smith

b_sal
-----
7400
```

Exibindo Parâmetros OUT: Usando as Variáveis de Host do SQL*Plus

O exemplo no slide demonstra como usar as variáveis de host do SQL*Plus que são criadas usando o comando `VARIABLE`. Essas variáveis são externas ao bloco PL/SQL e são conhecidas como variáveis de host ou de bind. Para fazer referência a variáveis de host em um bloco PL/SQL, é necessário incluir dois-pontos (:). antes dos nomes dessas variáveis. Para exibir os valores armazenados nas variáveis de host, você deve usar o comando `PRINT` do SQL*Plus, seguido do nome da variável do SQL*Plus (sem os dois-pontos porque não se trata de um comando ou contexto PL/SQL).

Observação: Para obter detalhes sobre o comando `VARIABLE`, consulte o manual SQL*Plus Command Reference.

Notações Disponíveis para Especificar Parâmetros Reais

- Ao chamar um subprograma, você pode gravar os parâmetros reais usando as seguintes notações:
 - Posicional: Lista os parâmetros reais na mesma ordem que os parâmetros formais
 - Nomeada: Lista os parâmetros reais em ordem arbitrária e usa o operador de associação ($=>$) para associar um parâmetro formal nomeado a seu parâmetro real
 - Combinada: Lista alguns dos parâmetros reais como posicionais e outros como nomeados
- Antes do Oracle Database 11g, somente a notação posicional era suportada em chamadas do SQL
- A partir do Oracle Database 11g, a notação nomeada e combinada pode ser usada para especificar argumentos em chamadas para sub-rotinas PL/SQL a partir de instruções SQL

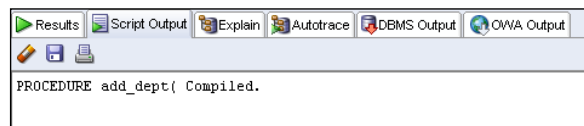
Sintaxe para a Especificação de Parâmetros

Ao chamar um subprograma, você pode gravar os parâmetros reais usando as seguintes notações:

- **Posicional:** Você lista os valores dos parâmetros reais na ordem em que os parâmetros formais foram declarados. Esta notação é compacta, mas se você especificar os parâmetros (especialmente literais) na ordem errada, o erro pode ser difícil de detectar. Você deve alterar seu código se a lista de parâmetros do procedure for alterada.
- **Nomeada:** Você lista os valores reais em ordem arbitrária e usa o operador de associação para associar cada parâmetro real a seu parâmetro formal por nome. O operador de associação do código **PL/SQL** é um sinal de "igual" seguido de um sinal de "maior que", sem espaços: $=>$. A ordem dos parâmetros não é significativa. Esta notação é mais verbosa, porém torna seu código mais fácil de ler e manter. Às vezes, você pode evitar alterar seu código se a lista de parâmetros do procedure for alterada, por exemplo, se os parâmetros forem reordenados ou um novo parâmetro opcional for incluído.
- **Combinada:** Você lista os valores do primeiro parâmetro pela sua posição e os demais usando a sintaxe especial do método nomeado. Esta notação pode ser usada para chamar procedures que tenham alguns parâmetros obrigatórios, seguidos por alguns parâmetros opcionais.

Especificando Parâmetros Reais: Criando o Procedure add_dept

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

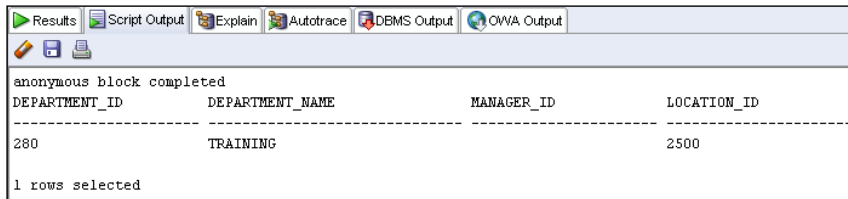


Especificando Parâmetros: Exemplos

No exemplo do slide, o procedure add_dept declara dois parâmetros formais IN: p_name e p_loc. Os valores desses parâmetros são usados na instrução INSERT para definir as colunas department_name e location_id, respectivamente.

Especificando Parâmetros Reais: Exemplos

```
-- Passing parameters using the positional notation.  
EXECUTE add_dept ('TRAINING', 2500)
```

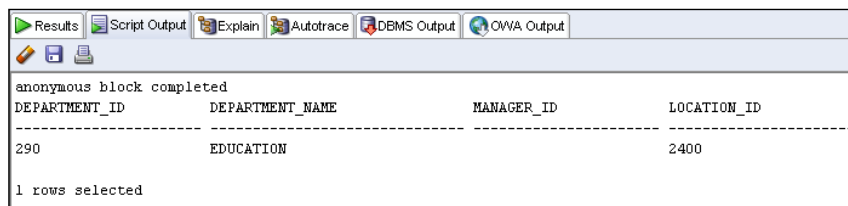


The screenshot shows the SQL Developer interface with the 'Results' tab selected. It displays the output of an 'anonymous block completed'. The output is a table with four columns: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, and LOCATION_ID. The first row shows the result of the add_dept procedure: DEPARTMENT_ID 280, DEPARTMENT_NAME TRAINING, MANAGER_ID (blank), and LOCATION_ID 2500. Below the table, it states '1 rows selected'.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING		2500

1 rows selected

```
-- Passing parameters using the named notation.  
EXECUTE add_dept (p_loc=>2400, p_name=>'EDUCATION')
```



The screenshot shows the SQL Developer interface with the 'Results' tab selected. It displays the output of an 'anonymous block completed'. The output is a table with four columns: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, and LOCATION_ID. The first row shows the result of the add_dept procedure: DEPARTMENT_ID 290, DEPARTMENT_NAME EDUCATION, MANAGER_ID (blank), and LOCATION_ID 2400. Below the table, it states '1 rows selected'.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
290	EDUCATION		2400

1 rows selected

Especificando Parâmetros Reais: Exemplos

Especificar parâmetros reais pela posição é mostrado na primeira chamada para executar `add_dept` no primeiro exemplo de código no slide. O primeiro parâmetro real fornece o valor `TRAINING` para o parâmetro formal `name`. O valor do segundo parâmetro real de `2500` é designado por posição para o parâmetro formal `loc`.

A especificação de parâmetros usando a notação nomeada é exibida no segundo exemplo de código no slide. O parâmetro real `loc` que é declarado como o segundo parâmetro formal, é referenciado por nome na chamada, em que é associado ao valor real de `2400`. O parâmetro `name` é associado ao valor `EDUCATION`. A ordem dos parâmetros reais será irrelevante se todos os valores de parâmetros forem especificados.

Observação: Você deverá fornecer um valor para cada parâmetro, a menos que seja designado um valor default para o parâmetro formal. A especificação de valores default para parâmetros formais é abordada a seguir.

Usando a Opção DEFAULT com Parâmetros

- Define valores default para parâmetros
- Oferece flexibilidade uma vez que combina as sintaxes de especificação de parâmetros posicional e nomeada

```
CREATE OR REPLACE PROCEDURE add_dept(  
  p_name departments.department_name%TYPE := 'Unknown',  
  p_loc  departments.location_id%TYPE DEFAULT 1700)  
IS  
BEGIN  
  INSERT INTO departments(department_id,  
    department_name, location_id)  
  VALUES (departments_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)
```

Usando a Opção DEFAULT com Parâmetros

Você pode designar um valor default a um parâmetro IN como se segue:

- O operador de designação (: =), conforme mostrado para o parâmetro name no slide
- A opção DEFAULT, conforme mostrada para o parâmetro p_loc no slide

Quando valores default são designados a parâmetros formais, é possível chamar o procedure sem fornecer um valor de parâmetro real para o parâmetro. Portanto, você pode especificar números diferentes de parâmetros reais para um subprograma, aceitando ou substituindo os valores default, conforme necessário. Recomenda-se declarar os parâmetros primeiramente sem os valores default. Depois, você poderá adicionar parâmetros formais com valores default sem precisar alterar todas as chamadas do procedure.

Observação: Não é possível designar valores default para os parâmetros OUT e IN OUT.

A segunda caixa de código no slide mostra três formas de chamar o procedure add_dept:

- O primeiro exemplo designa os valores default para cada parâmetro.
- O segundo exemplo ilustra a combinação das notações posicional e nomeada para designar valores. Nesse caso, é apresentado como exemplo o uso da notação nomeada.
- O último exemplo usa o valor default para o parâmetro name, Unknown, e o valor fornecido para o parâmetro p_loc.

Procedures que Fizeram a Chamada

- Você pode chamar os procedures usando blocos anônimos, outro procedure ou pacotes.
- Você deve ser o proprietário do procedure ou ter o privilégio EXECUTE

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

PROCEDURE process_employees Compiled.

Procedures que Fizeram a Chamada

É possível chamar procedures usando:

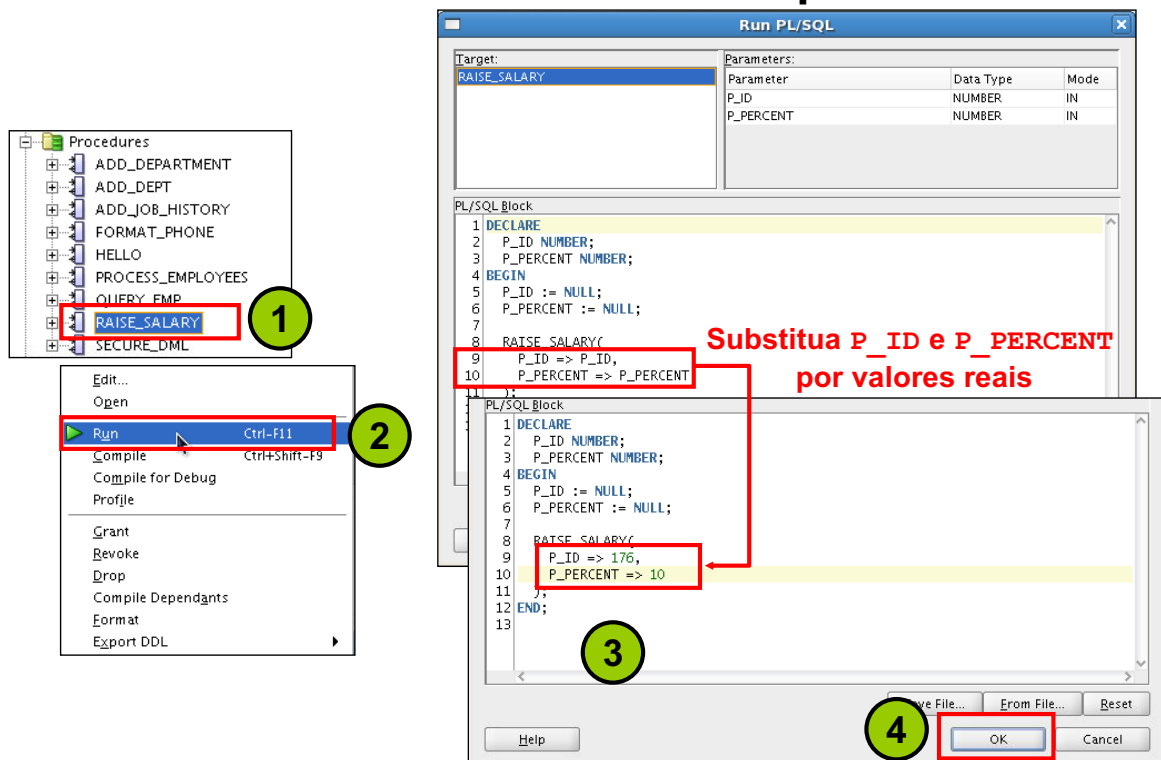
- Blocos anônimos
- Outro procedure ou subprograma PL/SQL

Os exemplos das páginas anteriores ilustraram como usar blocos anônimos (ou o comando EXECUTE no SQL Developer ou SQL*Plus).

Este exemplo mostra como chamar um procedure a partir de outro procedure armazenado. O procedure armazenado PROCESS_EMPLOYEES usa um cursor para processar todos os registros da tabela EMPLOYEES e especifica a ID de cada funcionário para o procedure RAISE_SALARY, resultando em um aumento salarial de 10% para toda a empresa.

Observação: Você deve ser o proprietário do procedure ou ter o privilégio EXECUTE.

Procedures que Fizeram a Chamada Usando o SQL Developer



Procedures que Fizeram a Chamada Usando o SQL Developer

No exemplo do slide, o procedure `raise_salary` é chamado para aumentar o salário atual do funcionário 176 (\$ 8.600) em 10 por cento como se segue:

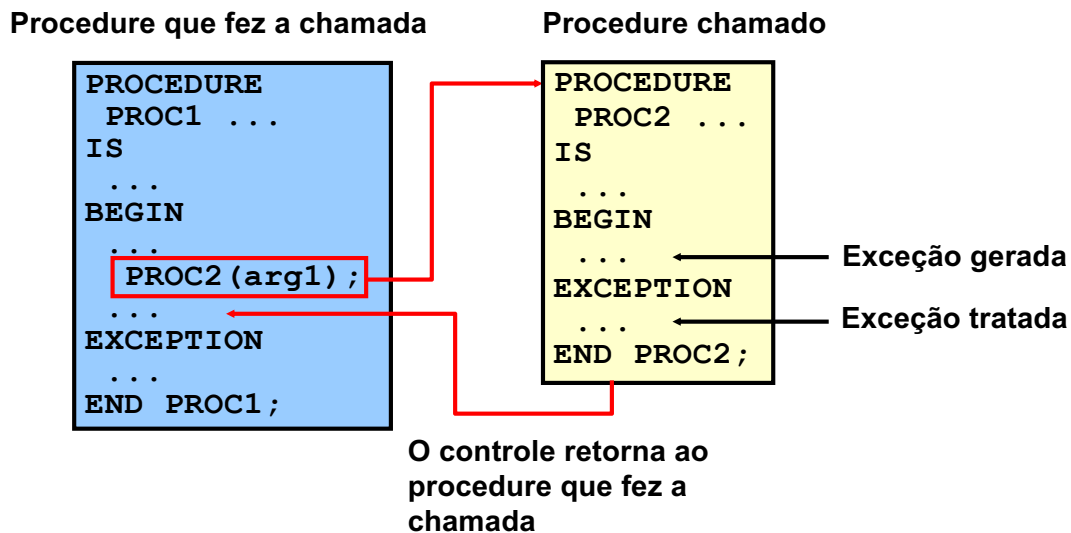
1. Clique com o botão direito do mouse no nome do procedure no nó **Procedures**, e clique em **Run**. A caixa de diálogo **Run PL/SQL** é exibida.
2. Na seção **PL/SQL Block**, altere as especificações de parâmetro formal **IN** e **IN/OUT** exibidas após o operador de associação, “**=>**” para os valores *reais* que você deseja usar para executar ou depurar a função ou procedure. Por exemplo, para aumentar o salário atual do funcionário 176 de 8.600 em 10 por cento, você pode chamar o procedure `raise_salary` conforme mostrado no slide. Forneça os valores para os parâmetros de entrada **ID** e **PERCENT** que estão especificados como 176 e 10 respectivamente. Isto é feito alterando o **ID => ID** com **ID => 176** e **PERCENT => PERCENT** com **PERCENT => 10**.
3. Clique em **OK**. O SQL Developer executa o procedure. O salário atualizado de 9.460 é exibido abaixo:

Results		
Script Output		
Explain		
Autotrace		
DBMS Output		
OWA Output		
Results:		
EMPLOYEE_ID	SALARY	
1	176	9460

Agenda de Lições

- Usando um design de subprograma modularizado e em camadas e identificando os benefícios de subprogramas
- Trabalhando com procedures:
 - Criando e chamando procedures
 - Identificando os modos disponíveis de especificar parâmetros
 - Usando parâmetros formais e reais
 - Usando notação posicional, nomeada ou combinada
- **Tratando exceções em procedures, removendo um procedure e exibindo as informações dos procedures**

Exceções Tratadas



Exceções Tratadas

Ao desenvolver procedures que são chamados a partir de outros procedures, você deve estar ciente dos efeitos que as exceções tratadas e não tratadas têm sobre a transação e o procedure que fez a chamada.

Quando uma exceção é gerada em um procedure chamado, o controle é transferido imediatamente para a seção de exceções desse bloco. Uma exceção será considerada como tratada se a seção de exceções fornecer um handler para a exceção que foi gerada.

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

1. A exceção é gerada.
2. O controle é transferido para o handler de exceções.
3. O bloco é encerrado.
4. O programa/bloco de chamada continua em execução como se nada tivesse ocorrido.

Se alguma transação tiver sido iniciada (ou seja, se uma instrução DML [Data Manipulation Language] tiver sido executada antes da execução do procedure no qual a exceção foi gerada), ela não será afetada. Uma operação DML será submetida a rollback se tiver sido executada no procedure antes da exceção.

Observação: Você pode encerrar uma transação de forma explícita, executando uma operação `COMMIT` ou `ROLLBACK` na seção de exceções.

Exceções Tratadas: Exemplo

```
CREATE PROCEDURE add_department(  
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS  
BEGIN  
    INSERT INTO DEPARTMENTS (department_id,  
        department_name, manager_id, location_id)  
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);  
    DBMS_OUTPUT.PUT_LINE('Added Dept: ' || p_name);  
EXCEPTION  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('Err: adding dept: ' || p_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 Tratadas: Exemplo

Os dois procedures do exemplo são os seguintes:

- O procedure `add_department` cria um novo departamento alocando um novo número de departamento a partir de uma sequência Oracle, e define os valores das colunas `department_name`, `manager_id`, e `location_id` usando os parâmetros `p_name`, `p_mgr`, e `p_loc`, respectivamente.
- O procedure `create_departments` cria mais de um departamento usando chamadas para o procedure `add_department`.

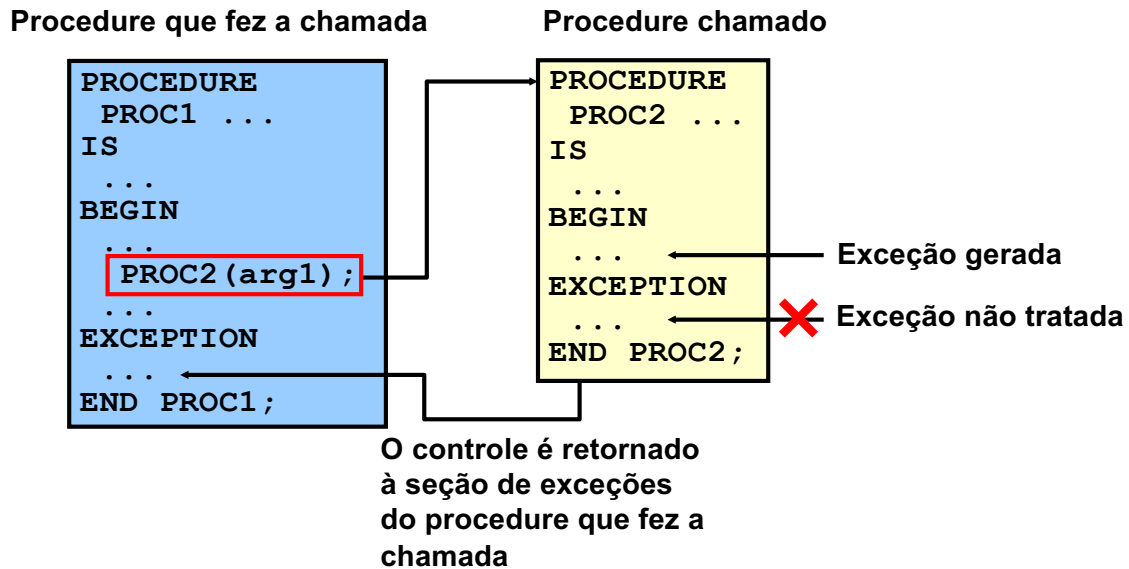
O procedure `add_department` captura todas as exceções geradas em seu próprio handler.

Quando o procedure `create_departments` é executado, é gerada a seguinte saída:

```
Added Dept: Media  
Err: adding dept: Editing  
Added Dept: Advertising
```

O departamento de Edição com um `manager_id` de 99 que não é inserido devido a uma violação de restrição de integridade de chave estrangeira em `manager_id` garante que nenhum gerente tem um ID de 99. Como a exceção foi tratada no procedure `add_department`, o procedure `create_department` continua a ser executado. Uma consulta na tabela `DEPARTMENTS`, em que `location_id` é 1800, mostra que `Media` e `Advertising` são adicionados, mas o registro `Editing` não.

Exceções Não Tratadas



Exceções Não Tratadas

Como vimos anteriormente, quando uma exceção é gerada em um procedure chamado, o controle é transferido imediatamente para a seção de exceções desse bloco. Se a seção de exceções não fornecer um handler para a exceção gerada, esta não será tratada. O seguinte fluxo de código ocorrerá:

1. A exceção é gerada.
2. O bloco é encerrado porque não existe nenhum handler de exceções; as operações DML que tiverem sido executadas no procedure serão submetidas a rollback.
3. A exceção é propagada para a seção de exceções do procedure que fez a chamada, ou seja, o controle é retornado à seção de exceções do bloco de chamada, caso exista alguma.

Se uma exceção não for tratada, todas as instruções DML do procedure que fez a chamada e do procedure chamado serão submetidas a rollback, junto com quaisquer alterações nas variáveis de host. As instruções DML não afetadas são as que foram executadas antes da chamada do código PL/SQL cujas exceções não foram tratadas.

Exceções Não Tratadas: Exemplo

```
SET SERVEROUTPUT ON
CREATE PROCEDURE add_department_noex(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: ' || p_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;
```

X
X
X

Exceções Não Tratadas: Exemplo

O exemplo de código do slide mostra o procedure `add_department_noex`, que não contém uma seção de exceções. Nesse caso, a exceção ocorre quando o departamento `Editing` é adicionado. Como não há tratamento de exceções em nenhum dos subprogramas, não são adicionados novos registros de departamento à tabela `DEPARTMENTS`. A execução do procedure `create_departments_noex` produzirá um resultado semelhante a este:

```
Error starting at line 8 in command:
EXECUTE create_departments_noex
Error report:
ORA-02291: integrity constraint (ORA62.DEPT_MGR_FK) violated - parent key not found
ORA-06512: at "ORA62.ADD_DEPARTMENT_NOEX", line 4
ORA-06512: at "ORA62.CREATE_DEPARTMENTS_NOEX", line 4
ORA-06512: at line 1
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:      A foreign key value has no matching primary key value.
*Action:     Delete the foreign key or add a matching primary key.
```

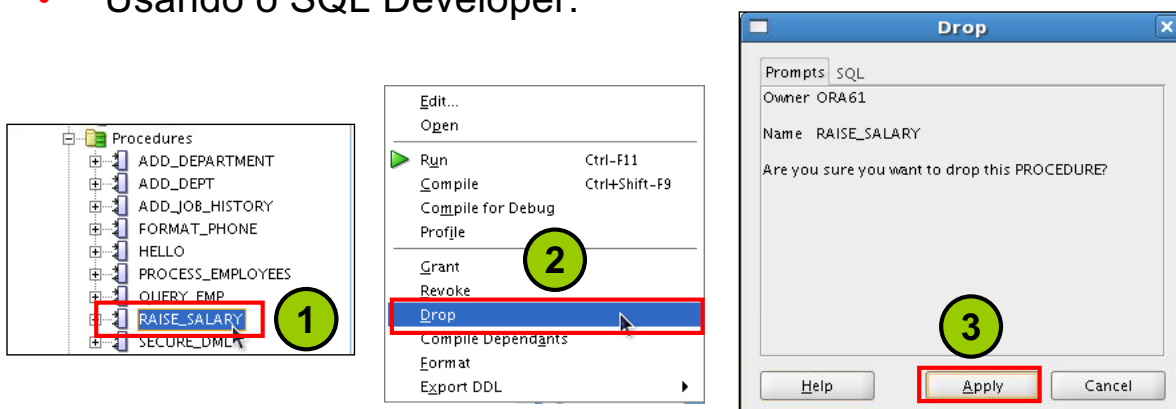
Embora os resultados mostrem que o departamento `Media` foi adicionado, essa operação será submetida a rollback porque a exceção não foi tratada em nenhum dos subprogramas chamados.

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

- Usando a instrução DROP:

```
DROP PROCEDURE raise_salary;
```

- Usando o SQL Developer:



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

Quando um procedure armazenado não for mais necessário, você poderá usar a instrução DROP PROCEDURE do SQL seguida pelo nome do procedure para removê-la, como se segue:

```
DROP PROCEDURE procedure_name
```

Você também pode usar o SQL Developer para eliminar um procedure armazenado como se segue:

1. Clique com o botão direito do mouse no nome do procedure no nó **Procedures**, e clique em **Drop**. A caixa de diálogo **Drop** será exibida.
2. Clique em **Apply** para eliminar o procedure.

Observação

- Sendo bem-sucedida ou não, a execução de um comando DDL (Data Definition Language), como DROP PROCEDURE, submete a commit as transações pendentes que não podem ser submetidas a rollback.
- Talvez seja necessário atualizar o nó **Procedures** para que você possa ver os resultados da operação de eliminação. Para atualizar o nó **Procedures**, clique com o botão direito do mouse no nome do procedure no nó **Procedures** e clique em **Refresh**.

Exibindo Informações do Procedure Usando as Views do Dicionário de Dados

```
DESCRIBE user_source
```

```
DESCRIBE user_source
Name                               Null    Type
-----
NAME                               VARCHA2(30)
TYPE                               VARCHA2(12)
LINE                               NUMBER
TEXT                              VARCHA2(4000)
4 rows selected
```

```
SELECT text
FROM   user_source
WHERE  name = 'ADD_DEPT' AND type = 'PROCEDURE'
ORDER BY line;
```

```
TEXT
1 PROCEDURE add_dept(
2   p_name IN departments.department_name%TYPE,
3   p_loc  IN departments.location_id%TYPE) IS
4
5 BEGIN
6   INSERT INTO departments(department_id, department_name, location_id)
7   VALUES (departments_seq.NEXTVAL, p_name, p_loc);
8 END add_dept;
```

Exibindo o Procedure no Dicionário de Dados

O código-fonte de subprogramas PL/SQL é armazenado nas tabelas do dicionário de dados. O código-fonte pode ser acessado pelos procedures PL/SQL que são compilados com sucesso ou não. Para exibir o código-fonte PL/SQL armazenado no dicionário de dados, execute uma instrução `SELECT` nas seguintes tabelas:

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

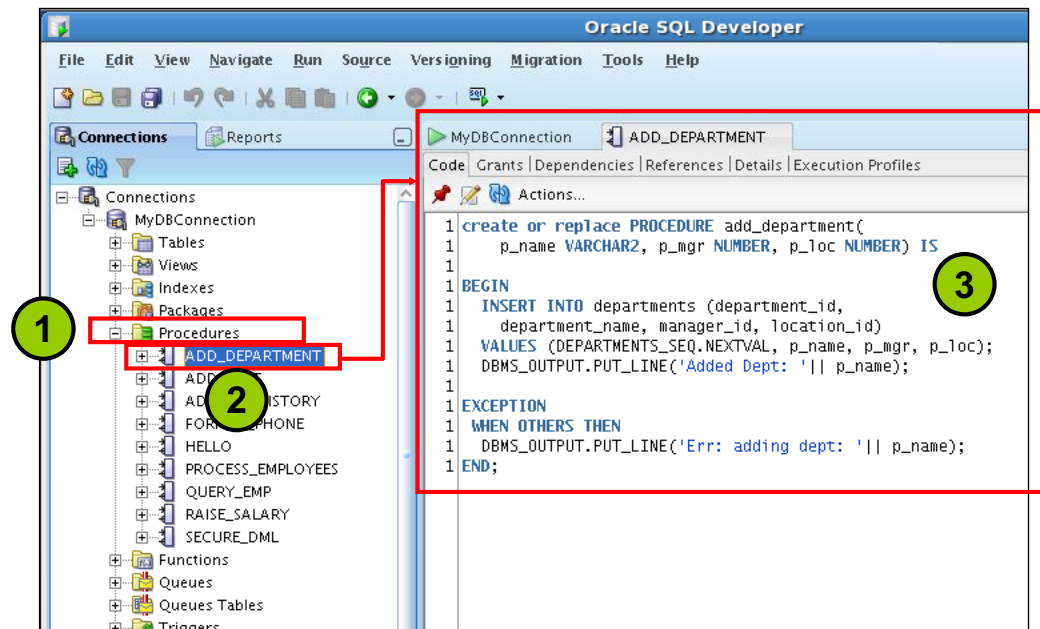
O exemplo de consulta mostra todas as colunas fornecidas pela tabela `USER_SOURCE`:

- A coluna `TEXT` armazena uma linha do código-fonte PL/SQL.
- A coluna `NAME` armazena o nome do subprograma em letras maiúsculas.
- A coluna `TYPE` armazena o tipo de subprograma, como `PROCEDURE` ou `FUNCTION`.
- A coluna `LINE` armazena o número de cada linha do código-fonte.

A tabela `ALL_SOURCE` fornece uma coluna `OWNER`, além das colunas anteriores.

Observação: Não é possível exibir o código-fonte de pacotes PL/SQL Oracle incorporados, nem o código PL/SQL cujo código-fonte tenha sido encapsulado com o utilitário `WRAP`. O utilitário `WRAP` converte o código-fonte PL/SQL em um formato que não pode ser decifrado por outras pessoas.

Exibindo as Informações dos Procedures Usando o SQL Developer



Exibindo as Informações dos Procedures Usando o SQL Developer

Para exibir o código de um procedure no SQL Developer, faça o seguinte:

1. Clique no nó **Procedures** na tab **Connections** .
2. Clique no nome do procedure.
3. O código do procedure é exibido na tab **Code** conforme mostrado no slide.

Questionário

Parâmetros formais são valores literais, variáveis e expressões usados na lista de parâmetros do subprograma que fez a chamada

1. Verdadeiro
2. Falso

Resposta: 2

Parâmetros Formais e Reais (ou argumentos)

- **Parâmetros formais:** Variáveis locais declaradas na lista de parâmetros da especificação de um subprograma.
- **Os parâmetros reais:** Expressões, variáveis ou valores literais usados na lista de parâmetros do subprograma que fez a chamada.

Sumário

Nesta lição, você aprendeu a:

- Identificar os benefícios do design de subprogramas modularizados e em camadas
- Criar e chamar procedures
- Usar parâmetros formais e reais
- Usar notação posicional, nomeada ou combinada para especificar parâmetros
- Identificar os modos disponíveis de especificar parâmetros
- Tratar exceções em procedures
- Remover um procedure
- Exibir as informações dos procedures

Visão Geral do Exercício 9: Criando, Compilando e Chamando Procedures

Este exercício aborda os seguintes tópicos:

- Criando procedures armazenados para:
 - Inserir novas linhas em uma tabela, usando os valores de parâmetro fornecidos
 - Atualizar os dados de uma tabela nas linhas que correspondem aos valores de parâmetro fornecidos
 - Deletar as linhas de uma tabela que correspondem aos valores de parâmetro fornecidos
 - Consultar uma tabela e recuperar dados com base nos valores de parâmetro fornecidos
- Tratando exceções em procedures
- Compilando e chamando procedures

Exercício 9: Visão Geral

Neste exercício, você cria, compila e chama procedures que executam comandos de consulta e DML. Você também aprende a tratar exceções em procedures.

Se você encontrar erros de compilação ao executar procedures, pode usar a tab Compiler-Log no SQL Developer.

Observação: Recomenda-se usar o SQL Developer neste exercício.

Importante

Todos os exercícios neste curso e as soluções deles pressupõem que você cria objetos como procedures, funções e assim por diante, usando a área SQL Worksheet no SQL Developer. Quando se cria um objeto na área SQL Worksheet, é necessário atualizar o nó de objeto na ordem para que o novo objeto seja exibido na árvore Navigator. Para compilar o objeto recém-criado, você pode clicar com o botão direito do mouse no nome do objeto na árvore Navigator e, depois, selecionar Compile no menu de atalho. Por exemplo, depois de inserir o código para criar um procedure na área SQL Worksheet, você clica no ícone Run Script (ou pressiona F5) para executar o código. Isto cria e compila o procedure.

Outra alternativa é criar objetos como procedures usando o nó PROCEDURES na árvore Navigator e, depois, compilar o procedure. Criar objetos usando a árvore Navigator exibe automaticamente o objeto recém-criado.