

Oracle PL/SQL

MasterTraining – Treinamento e Tecnologia

www.mastertraining.com.br

Márcio Konrath

marcio@mastertraining.com.br

MasterTraining – Treinamento e Tecnologia	1
www.mastertraining.com.br	1
Márcio Konrath	1
marcio@mastertraining.com.br	1
Capítulo 01	21
Introdução ao PL/SQL	21
Objetivos	21
Objetivo da Lição	22
Sobre PL/SQL	22
Sobre PL/SQL	22
Integração	23
PL/SQL nas Ferramentas Oracle	24
Desempenho Melhorado	25
Estrutura de Bloco PL/SQL	26
Executando Instruções e Blocos PL/SQL a partir do Código SQL*Plus	27
Tipos de Bloco	29
Blocos Anônimos	29
Subprogramas	29
Construções de Programa	30
Construções de Programa (continuação)	32
Uso de Variáveis	32
Uso de Variáveis	32
Tratando Variáveis em PL/SQL	33
Tratando Variáveis em PL/SQL	33
Tipos de Variáveis	34

Tipos de Variáveis	35
Usando Variáveis SQL*Plus nos Blocos PL/SQL.....	35
Tipos de Variáveis.....	37
Declarando Variáveis PL/SQL	37
Sintaxe	37
Declarando Variáveis PL/SQL	37
Declarando Variáveis PL/SQL	38
Diretrizes	38
Regras para Nomeação	39
Regras para Nomeação	39
Atribuindo Valores às Variáveis	40
Sintaxe	40
Atribuindo Valores às Variáveis	41
Palavras-chave e Inicialização de Variáveis	41
Tipos de Dados Escalares	43
Tipos de Dados Escalares	43
Tipos de Dados Escalares Básicos	43
Tipos de Dados Escalares Básicos.....	44
Tipos de Dados Escalares Básicos	45
Declarando Variáveis Escalares.....	46
Declarando Variáveis Escalares	46
O Atributo %TYPE.....	47
O Atributo %TYPE	47
Declarando Variáveis com o	48
Atributo %TYPE	48
Declarando Variáveis com o Atributo %TYPE.....	48

Declarando Variáveis Booleanas	49
Declarando Variáveis Booleanas	49
Estrutura de Registro PL/SQL	50
Tipos de Dados Compostos	50
Variáveis de Tipo de Dados LOB	51
Variáveis de Tipo de Dados LOB	51
Variáveis de Ligação	52
Variáveis de Ligação	52
Criando Variáveis de Ligação	53
Exibindo Variáveis de Ligação	53
Referenciando Variáveis	53
Não-PL/SQL	53
Atribuindo Valores às Variáveis	53
Exemplo	54
DBMS_OUTPUT.PUT_LINE	54
Uma outra Opção	54
Sumário	55
Sumário	56
Sumário	56
Sumário (continuação)	56
Capítulo 02	57
Criando Instruções	57
Executáveis	57
Objetivo da Lição	58
Diretrizes e Sintaxe de Bloco PL/SQL	58
Diretrizes e Sintaxe de Bloco PL/SQL	58
Delimitadores	59

Diretrizes e Sintaxe de Bloco PL/SQL	59
Identificadores	59
Diretrizes e Sintaxe de Bloco PL/SQL.....	60
Literais	60
Comentando Código	61
Comentando Código.....	61
Exemplo.....	61
Funções SQL em PL/SQL	62
Funções SQL em PL/SQL.....	62
Exemplo.....	63
Funções PL/SQL.....	63
Funções PL/SQL	63
Conversão de Tipo de Dados	64
Conversão de Tipo de Dados.....	64
Sintaxe	64
Conversão de Tipo de Dados.....	65
Conversão de Tipo de Dados.....	65
Blocos Aninhados e Escopo de Variável.....	66
Blocos Aninhados	66
Escopo da Variável	66
Blocos Aninhados e Escopo de Variável.....	66
Identificadores	66
Blocos Aninhados e Escopo de Variável.....	67
Blocos Aninhados e Escopo de Variável	67
Escopo	67
Visibilidade	67

Operadores em PL/SQL.....	68
Ordem das Operações.....	68
Operadores em PL/SQL.....	69
Operadores em PL/SQL	69
Usando Variáveis de Ligação.....	70
Imprimindo Variáveis de Ligação	70
Diretrizes de Programação.....	70
Diretrizes de Programação	71
Convenções de Códigos.....	71
Convenções para Nomeação de Código	71
Convenções para Nomeação de Código.....	72
Endentando o Código	73
Endentando o Código.....	73
Determinando o Escopo da Variável	74
Exercício de Classe	74
Sumário	75
Sumário	75
Capítulo 03	77
Interagindo com o Oracle Server	77
Objetivo da Lição.....	78
Instruções SQL em PL/SQL.....	78
Visão Geral	78
Comparando os Tipos de Instrução SQL e PL/SQL	78
Instruções SELECT em PL/SQL.....	79
Recuperando Dados em PL/SQL.....	79
Instruções SELECT em PL/SQL.....	80
Cláusula INTO	80

As Consultas Devem Retornar Apenas Uma Linha.....	80
Recuperando Dados em PL/SQL.....	81
Diretrizes	81
Recuperando Dados em PL/SQL.....	82
Diretrizes (continuação)	82
Manipulando Dados Usando o PL/SQL	83
Manipulando Dados Usando o PL/SQL.....	83
Inserindo Dados	84
Inserindo Dados	84
Atualizando Dados	84
Atualizando e Deletando Dados.....	84
Deletando Dados.....	85
Deletando Dados.....	85
Convenções para Nomeação	86
Convenções para Nomeação.....	86
Convenções para Nomeação	86
Convenções para Nomeação (continuação)	87
Observação:.....	87
Instruções COMMIT e ROLLBACK	87
Controlando Transações	87
Cursor SQL.....	88
Cursor SQL.....	88
Atributos do Cursor SQL.....	89
Atributos do Cursor SQL.....	89
Atributos do Cursor SQL.....	90
Atributos do Cursor SQL (continuação).....	90

Sumário	90
Sumário	91
Sumário	91
Sumário (continuação)	91
Capítulo 04	93
Criando Estruturas para Controle	93
Objetivo da Lição	94
Controlando o Fluxo de Execução PL/SQL	94
Instruções IF	95
Instruções IF	95
Instruções IF Simples	96
Instruções IF Simples	96
Diretrizes	96
Fluxo de Execução da Instrução IF-THEN-ELSE	97
Fluxo de Execução da Instrução IF-THEN-ELSE	97
Instruções IF Aninhadas	97
Instruções IF-THEN-ELSE	98
Fluxo de Execução da Instrução	99
IF-THEN-ELSIF	99
Fluxo de Execução da Instrução IF-THEN-ELSIF	99
Instruções IF-THEN-ELSIF	100
Instruções IF-THEN-ELSIF	100
Elaborando Condições Lógicas	101
Desenvolvendo Condições Lógicas	101
Tabelas Lógicas	102
Condições Booleanas	103
Desenvolvendo Condições Lógicas	103

Controle Iterativo: Instruções LOOP	104
Controle Iterativo: Instruções LOOP	104
Loop Básico.....	105
Loop Basico.....	105
Loop Básico.....	106
Loop Básico (continuação)	106
Loop FOR	107
Loop FOR	107
Loop FOR	108
Loop FOR (continuação)	108
Loop FOR	109
Loop For.....	109
Loop WHILE	110
Loop WHILE	110
Loop WHILE	111
Loop WHILE (continuação).....	111
Loops e Labels Aninhados	111
Loops e Labels Aninhados	111
Loops e Labels Aninhados	112
Loops e Labels Aninhados	112
Sumário	113
Sumário	113
Capítulo 05	115
Trabalhando com Tipos de Dados Record e Collections	
.....	115
Objetivo da Lição.....	116
Tipos de Dados Compostos	116

RECORDS e TABLES.....	116
Registros PL/SQL.....	117
Registros PL/SQL	117
Criando um Registro PL/SQL	118
Definindo e Declarando um Registro PL/SQL.....	118
Criando um Registro PL/SQL	119
Criando um Registro PL/SQL	119
Estrutura de Registro PL/SQL	120
Fazendo Referência e Inicializando Registros	120
O Atributo %ROWTYPE	121
Declarando Registros com o Atributo %ROWTYPE	121
Vantagens de Usar %ROWTYPE.....	122
Declarando Registros com o Atributo %ROWTYPE (continuação)	122
O Atributo %ROWTYPE	123
Tabelas PL/SQL	124
Tabelas PL/SQL	124
Criando uma Tabela PL/SQL	125
Criando uma Tabela PL/SQL	125
Estrutura de Tabela PL/SQL.....	126
Estrutura de Tabela PL/SQL.....	126
Criando uma Tabela PL/SQL	127
Criando uma Tabela PL/SQL	127
Fazendo Referência a uma Tabela PL/SQL.....	127
Usando Métodos de Tabela PL/SQL.....	128
Tabela de Registros PL/SQL.....	129
Tabela de Registros PL/SQL.....	129

Fazendo Referência a uma Tabela de Registros.....	130
Exemplo de Tabela de Registros PL/SQL.....	130
Exemplo de Tabela de Registros PL/SQL	130
Sumário	131
Sumário	131
Capítulo 06	132
Criando Cursores Explícitos.....	132
Objetivo da Lição.....	133
Sobre os Cursores	133
Cursores Implícitos e Explícitos	133
Cursores Implícitos	133
Funções do Cursor Explícito	134
Cursores Explícitos	134
Controlando Cursores Explícitos	135
Cursores Explícitos (continuação)	136
Controlando Cursores Explícitos	136
Cursores Explícitos (continuação)	137
Declarando o Cursor	137
Declaração de Cursor Explícito.....	137
Declarando o Cursor	138
Declaração de Cursor Explícito (continuação).....	138
Abrindo o Cursor	138
Instrução OPEN	139
Instrução OPEN (continuação)	139
Extraindo Dados do Cursor.....	140
Instrução FETCH	140
Extraindo Dados do Cursor.....	141

Instrução FETCH (continuação)	141
Fechando o Cursor	142
Instrução CLOSE	142
Atributos do Cursor Explícito	143
Atributos do Cursor Explícito	143
Controlando Várias Extrações	143
Controlando Várias Extrações de Cursores Explícitos	144
O Atributo %ISOPEN	144
Atributos do Cursor Explícito	144
Os Atributos %NOTFOUND e %ROWCOUNT	145
Exemplo (continuação).....	145
Cursores e Registros	146
Cursores e Registros.....	146
Loops FOR de Cursor	147
Loops FOR de Cursor	147
Loops FOR de Cursor	148
Loops FOR do Cursor Usando Subconsultas	149
Loops FOR do Cursor Usando Subconsultas.....	149
Sumário	150
Sumário	150
Cursores com Parâmetros	151
Cursores com Parâmetros.....	152
Cursores com Parâmetros	153
A Cláusula FOR UPDATE	154
A Cláusula FOR UPDATE	154
A Cláusula FOR UPDATE	155

A Cláusula FOR UPDATE (continuação)	155
A Cláusula WHERE CURRENT OF.....	156
A Cláusula WHERE CURRENT OF	156
A Cláusula WHERE CURRENT OF.....	156
A Cláusula WHERE CURRENT OF (continuação)	157
Cursors com Subconsultas	157
Subconsultas	157
Sumário	158
Sumário	158
Capítulo 07	160
Tratando Exceções	160
Objetivo da Lição.....	161
Tratando Exceções com Código PL/SQL	161
Visão Geral	161
Dois Métodos para Criar uma Exceção	161
Tratando Exceções	162
Capturando uma Exceção.....	162
Propagando uma Exceção	162
Tipos de Exceção.....	163
Tipos de Exceção	163
Capturando Exceções	164
Capturando Exceções	164
Handler de Exceção WHEN OTHERS.....	165
Diretrizes para a Captura de Exceções	165
Diretrizes	165
Capturando Erros Predefinidos do Oracle Server	166
Capturando Erros Predefinidos do Oracle Server	166

Exceções Predefinidas	167
Exceção Predefinida	168
Sintaxe	168
Capturando Exceções Predefinidas do Oracle Server	169
Capturando Erros Não Predefinidos do Oracle Server	169
Capturando Erros Não Predefinidos do Oracle Server	169
Erro Não Predefinido	170
Capturando uma Exceção Não Predefinida do Oracle Server	170
Funções para Captura de Exceções	171
Funções para Captura de Erro	171
Funções para Captura de Exceções	172
Funções para Captura de Erro	172
Capturando Exceções Definidas pelo Usuário	173
Capturando Exceções Definidas pelo Usuário	173
Exceção Definida pelo Usuário	174
Capturando Exceções Definidas pelo Usuário (continuação)	174
Ambientes de Chamada	175
Propagando Exceções	175
Propagando Exceções	176
Propagando uma Exceção para um Sub-bloco	176
Procedimento	177
RAISE_APPLICATION_ERROR	177
Procedimento RAISE_APPLICATION_ERROR	177
Procedimento	178
RAISE_APPLICATION_ERROR	178

Sumário	178
Sumário	179
Capítulo 08	181
Procedimentos de Banco de Dados	181
Objetivos	181
Procedures	182
Criando Procedimentos de Banco de Dados	185
Criando Procedimentos de Banco de Dados (continuação).....	185
Parâmetros	185
Parâmetros IN	186
Parâmetros OUT.....	186
Parâmetros IN OUT	186
Parâmetros OUT e IN OUT por referência.....	187
Utilizando Múltiplos Parâmetros	187
Passando Parâmetros - Método Posicional.....	187
Passando Parâmetros - Método Combinado	188
Executando Procedimentos	188
Executando Procedimentos (continuação)	188
Removendo Procedimentos de Banco de Dados	188
Capítulo 09	190
Funções de Banco de Dados	190
Criando Funções de Banco de Dados	191
Criando uma função que retorna o preço de um curso (NUMBER)	191
Parâmetros em Funções.....	191
Executando Funções a partir de um bloco PL/SQL ..	192
Executando Funções a partir de um bloco PL/SQL (continuação)	192

Removendo Funções de Banco de Dados.....	193
Procedimentos X Funções	193
Capítulo 10	195
Desenvolvendo e utilizando Packages.....	195
Objetivos	195
O Que são Packages?	196
Desenvolvendo Packages - Visão Geral.....	196
PACKAGE SPECIFICATION	196
PACKAGE BODY	196
Desenvolvendo Packages - Visão Geral (continuação)	
.....	197
Construções Públicas em Packages.....	198
Construções Privadas em Packages	198
Passos para a criação de uma Package	198
Criando a Package Specification	198
Criando o Package Body.....	199
Definindo um Procedimento de Única Execução.....	199
Removendo a Package	199
Removendo o Package Body	199
Invocando Construções de Packages	200
Benefícios do Uso de Packages	200
Gerenciando Dependências em Packages.....	200
Re-compilando Packages	200
Capítulo 11	202
Desenvolvendo e Utilizando Databases Triggers.....	202
Objetivos	202
Triggers em Nível de Linha e em Nível de Comando.....	204

Ordem de disparo das Triggers	204
Criando uma Trigger em Nível de Comando	205
Tempos de Disparo da Trigger	205
Criando uma Trigger Combinando Vários Eventos...	206
Triggers em Nível de Linha	206
Criando Triggers em Nível de Linha	207
Valores OLD e NEW	207
Execução Condicional: Cláusula WHEN.....	208
Cláusula Referencing	208
Triggers INSTEAD OF	209
Criando Triggers INSTEAD OF	209
Mutating Tables	209
Mutating Tables – Exemplo 1	209
Mutating Tables – Exemplo 2	210
Resolvendo o erro de Mutating Tables	212
Resolvendo o erro de Mutating Tables (continuação)	212
Resolvendo o erro de Mutating Tables (continuação)	212
Habilitando e Desabilitando Database Triggers.....	213
Removendo uma Database Trigger	213
Gerenciando Database Triggers	213
Consultando o Código Fonte de Database Triggers .	214
Capítulo 12	216
Operadores SET	216
Objetivos	216
Operadores SET	217

União – UNION.....	217
União – UNION (continuação).....	217
União – UNION (continuação).....	218
União – UNION (continuação).....	218
Interseção - INTERSECT.....	218
Interseção – INTERSECT (continuação)	219
Diferença - MINUS	219
Diferença – MINUS (continuação).....	219
Usos Incomuns do Comando SELECT.....	220
Capítulo 13	222
Aperfeiçoando a Cláusula GROUP BY.....	222
Objetivos	222
Aperfeiçoando o GROUP BY.....	223
A Função ROLLUP.....	223
A Função CUBE.....	223
Identificando Sub-Grupos	223
Usando a função GROUPING	224
Função GROUPING com CASE.....	224
Usando a Função GROUPING_ID().....	225
GROUPING_ID() com CASE	225
Usando a Cláusula GROUPING SETS.....	225
Funções Analíticas	227
Função RANK().....	227
RANK() com Particionamento	227
Função DENSE_RANK()	227
Função RATIO_TO_REPORT	228
RATIO_TO_REPORT com particionamento	228

LAG() e LEAD()	228
Capítulo 14	230
Database Link	230
Objetivos	230
Acesso Remoto de Dados	231
Entendendo Database Link (continuação)	231
Utilizando Database Links	231
Criando Database Link	232
Database Links Public x Private	232
Login Default x Explícito	232
Connect String	232
Exemplo de Criação de Database Link	233
Visões do dicionário de dados	233
Capítulo 15	236
Comandos DML Avançado	236
Objetivos	236
Multi Table Insert	237
INSERT Incondicional	237
ALL INSERT Condicional	237
ALL INSERT Condicional (continuação)	238
FIRST INSERT Condicional	238
FIRST INSERT Condicional (continuação)	238
Comando MERGE	238
Comando MERGE (continuação)	239
Capítulo 16	240
SQL Dinâmico em PL/SQL	240
Objetivos	240

Conceito	241
Usando SQL Dinâmico	241
Usando SQL Dinâmico (continuação)	241
Porque utilizar SQL Dinâmico?	241
O Comando EXECUTE IMMEDIATE	242
Utilizando o Comando EXECUTE IMMEDIATE – Exemplo 1	242
Utilizando o Comando EXECUTE IMMEDIATE – Exemplo 2	243
Utilizando o Comando EXECUTE IMMEDIATE – Exemplo 3	243
Algumas considerações	243
Os Comandos OPEN-FOR, FETCH e CLOSE	244
Utilizando Seleções Dinâmicas	244
Passando Argumentos NULL	245

Capítulo 01

Introdução ao PL/SQL

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Listar os benefícios do código PL/SQL
- Reconhecer o bloco PL/SQL básico e suas seções
- Descrever o significado das variáveis no código PL/SQL
- Declarar Variáveis PL/SQL
- Executar o bloco PL/SQL

Objetivo da Lição

Esta lição apresenta as regras e estruturas básicas para criação e execução dos blocos de códigos

PL/SQL. Ela também mostra como declarar variáveis e atribuir tipos de dados a elas.

Sobre PL/SQL

- A linguagem PL/SQL é uma extensão da linguagem SQL com recursos de design de linguagens de programação.
- As instruções de consulta e a manipulação de dados em SQL estão incluídas nas unidades procedurais de código.

Sobre PL/SQL

A linguagem PL/SQL (Procedural Language/SQL) é uma extensão de linguagem procedural da

Oracle Corporation para SQL, a linguagem de acesso a dados padrão para bancos de dados

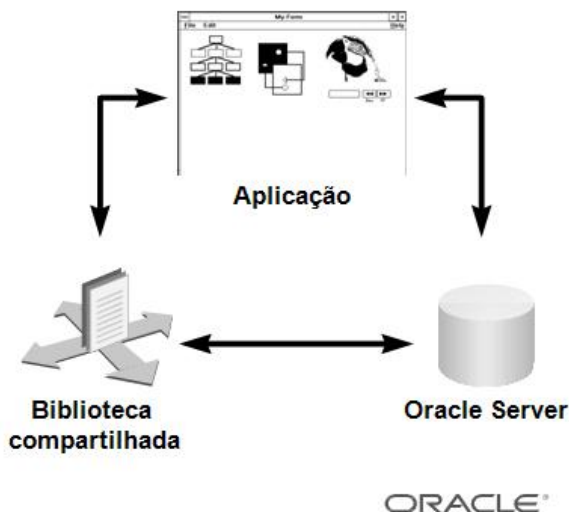
relacionais. A linguagem PL/SQL oferece recursos de engenharia de software modernos, como, por exemplo, a encapsulação de dados, o tratamento de exceções, a ocultação de informações e a orientação a objeto, etc., trazendo os recursos de programação mais modernos para o Oracle Server e o Toolset.

A linguagem PL/SQL incorpora muitos recursos avançados criados em linguagens de programação projetadas durante as décadas de 70 e 80. Além de aceitar a manipulação de dados, ele também permite que as instruções de consulta da linguagem SQL sejam incluídas em unidades procedurais

de código e estruturadas em blocos, tornando a linguagem SQL uma linguagem avançada de processamento de transações. Com a linguagem PL/SQL, você pode usar as instruções SQL para refinar os dados do Oracle e as instruções de controle PL/SQL para processar os dados.

Benefícios da Linguagem PL/SQL

Integração



Integração

A linguagem PL/SQL desempenha um papel central tanto para o Oracle Server (através de procedimentos armazenados, funções armazenadas, gatilhos de banco de dados e pacotes) quanto para as ferramentas de desenvolvimento Oracle (através de gatilhos de componente do Oracle Developer).

As aplicações do Oracle Developer fazem uso das bibliotecas compartilhadas que armazenam código (procedimentos e funções) e que podem ser acessadas local ou remotamente. O Oracle Developer consiste no Oracle Forms, Oracle Reports e Oracle Graphics.

Os tipos de dados SQL também podem ser usados no código PL/SQL. Combinados com o acesso direto que a linguagem SQL fornece, esses tipos de dados compartilhados integram a linguagem PL/SQL com o dicionário de dados do Oracle Server. A linguagem PL/SQL une o acesso conveniente à tecnologia de banco de dados com a necessidade de capacidade de programação procedural.

PL/SQL nas Ferramentas Oracle

Várias ferramentas Oracle, inclusive o Oracle Developer, têm o seu próprio mecanismo PL/SQL, o qual é independente do mecanismo presente no Oracle Server.

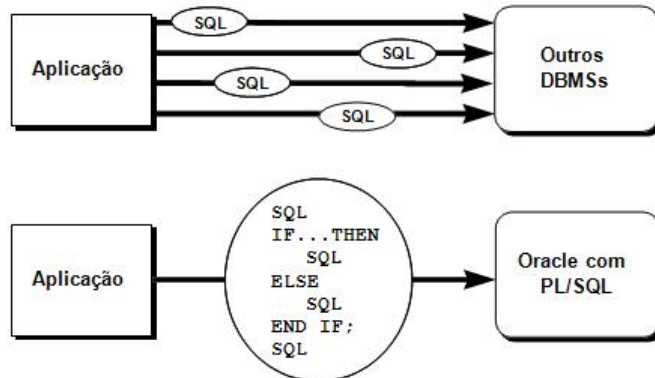
O mecanismo filtra as instruções SQL e as envia individualmente ao executor da instrução SQL no Oracle Server. Ele processa as instruções procedurais restantes no executor da instrução procedural, que está no mecanismo PL/SQL.

O executor da instrução procedural processa os dados que são locais para a aplicação (que já estão no ambiente do cliente, em vez de estarem no banco de dados). Isso reduz o trabalho enviado ao

Oracle Server e o número de cursores de memória necessários.

Benefícios da Linguagem PL/SQL

Melhorar desempenho



ORACLE®

Desempenho Melhorado

A linguagem PL/SQL pode melhorar o desempenho de uma aplicação. Os benefícios diferem dependendo do ambiente de execução.

- A linguagem PL/SQL pode ser usado para agrupar as instruções SQL em um único bloco e enviar esse bloco inteiro para o servidor em uma única chamada, reduzindo assim o tráfego da rede. Sem o código PL/SQL, as instruções SQL seriam enviadas ao Oracle Server uma de

cada vez. Cada instrução SQL resulta em uma outra chamada do Oracle Server e uma maior sobrecarga de desempenho. Em um ambiente de rede, a sobrecarga pode se tornar significativa. Como ilustra o slide, se sua aplicação for SQL intensiva, você pode usar os subprogramas e blocos PL/SQL para agrupar as instruções SQL antes de enviá-las ao Oracle Server para execução.

- A linguagem PL/SQL também pode cooperar com as ferramentas de desenvolvimento de aplicação do Oracle Server como, por exemplo, Oracle Developer Forms e Reports. Ao adicionar recursos de processamento procedural a essas ferramentas, a linguagem PL/SQL aumenta o desempenho.

Observação: Os procedimentos e as funções declaradas como parte de uma aplicação do Developer são distintos daqueles armazenados no banco de dados, embora as suas estruturas gerais sejam as mesmas. Os subprogramas armazenados são objetos de banco de dados e estão armazenados no dicionário de dados. Eles podem ser acessados por qualquer número de aplicações, incluindo as aplicações do Developer.

Estrutura de Bloco PL/SQL

- DECLARE – Opcional
Variáveis, cursores, exceções definidas pelo usuário
- BEGIN – Obrigatório
 - Instruções SQL
 - Instruções PL/SQL
- EXCEPTION – Opcional
Ações a serem desempenhadas quando ocorrem erros
- END; – Obrigatório

```
DECLARE
  ...
BEGIN
  ...
EXCEPTION
  ...
END;
```

ORACLE®

Estrutura de Bloco PL/SQL

A linguagem PL/SQL é uma linguagem estruturada em blocos, o que significa que os programas podem ser divididos em blocos lógicos. Um bloco PL/SQL consiste em até três seções: declarativa (opcional), executável (necessária) e tratamento de exceção (opcional). Apenas as palavras-chave BEGIN e END são necessárias. Você poderá declarar variáveis localmente para o bloco que as usa.

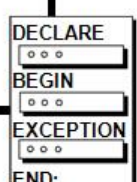
As condições de erro (conhecidas como exceções) podem ser tratadas especificamente no bloco aos quais elas se aplicam. Você poderá armazenar e alterar os valores em um bloco PL/SQL declarando e referenciando variáveis e outros identificadores. A tabela a seguir descreve as três seções de bloco:

Seção	Descrição	Inclusão
Declarativa	Contém todas as variáveis, constantes, cursores e exceções definidas pelo usuário que são referenciadas nas seções executável e declarativa	Opcional
Executável	Contém instruções SQL para manipular dados no banco de dados e instruções PL/SQL para manipular dados no bloco	Obrigatória
Tratamento de exceção	Especifica as ações a desempenhar quando erros e condições anormais surgem na seção executável	Opcional

Estrutura de Bloco PL/SQL

```

DECLARE
    v_variable  VARCHAR2(5);
BEGIN
    SELECT      column name
    INTO        v_variable
    FROM        table_name;
EXCEPTION
    WHEN exception_name THEN
        ...
END;
```



ORACLE®

Executando Instruções e Blocos PL/SQL a partir do Código SQL*Plus

- Coloque um ponto-e-vírgula (;) no final de uma instrução SQL ou instrução de controle PL/SQL.
- Use uma barra (/) para executar o bloco anônimo PL/SQL no buffer de SQL*Plus. Quando o bloco for executado corretamente, sem

erros que não possam ser tratados, a saída de mensagem deverá ser a seguinte:

```
PL/SQL procedure successfully completed  
(Procedimento PL/SQL concluído corretamente)
```

- Coloque um ponto (.) para fechar um buffer de SQL*Plus. Um bloco PL/SQL é tratado como uma instrução contínua no buffer e os ponto-e-vírgulas no bloco não fecham ou executam o buffer.

Observação: Em PL/SQL, um erro é chamado de exceção.

As palavras-chave de seção DECLARE, BEGIN e EXCEPTION não são seguidas por ponto-e- vírgulas. Entretanto, END e todas as outras instruções PL/SQL necessitam de um ponto-e-vírgula para terminar a instrução. Você poderá criar uma string de instruções na mesma linha, mas esse método não é recomendado pela clareza ou edição.

Tipos de Bloco

Anônimo

```
[DECLARE]  
  
BEGIN  
    --statements  
  
[EXCEPTION]  
  
END;
```

Procedimento

```
PROCEDURE name  
IS  
  
BEGIN  
    --statements  
  
[EXCEPTION]  
  
END;
```

Função

```
FUNCTION name  
RETURN datatype  
IS  
BEGIN  
    --statements  
    RETURN value;  
[EXCEPTION]  
  
END;
```

ORACLE®

Tipos de Bloco

Toda unidade PL/SQL compreende um ou mais blocos. Esses blocos podem ser inteiramente separados ou aninhados um dentro do outro. As unidades básicas (procedimentos e funções, também conhecidas como subprogramas e blocos anônimos) que compõem um programa PL/SQL são blocos lógicos, os quais podem conter qualquer número de sub-blocos aninhados. Por isso, um bloco pode representar uma pequena parte de um outro bloco, o qual, por sua vez pode ser parte da unidade de código inteira.

Dos dois tipos de construções PL/SQL disponíveis, blocos anônimos e subprogramas, apenas os blocos anônimos são abordados neste curso.

Blocos Anônimos

Os blocos anônimos são blocos sem nome. Eles são declarados em um ponto do aplicativo onde eles devem ser executados e são passados para o mecanismo PL/SQL para serem executados em tempo de execução. Você poderá incorporar um bloco anônimo em um programa pré-compilador e em SQL*Plus ou Server Manager. Os gatilhos nos componentes do Oracle Developer consistem nesses blocos.

Subprogramas

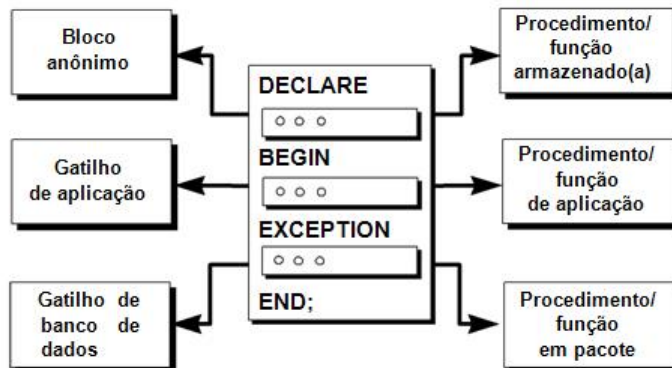
Os subprogramas são blocos PL/SQL nomeados que podem assumir parâmetros e podem ser chamados. Você pode declará-los como procedimentos ou como funções. Geralmente, você deve usar um procedimento para desempenhar uma ação e uma função para calcular um valor.

Você poderá armazenar subprogramas no servidor ou no nível de aplicação. Ao usar os componentes

do Oracle Developer (Forms, Relatórios e Gráficos), você poderá declarar os procedimentos e funções como parte da aplicação (um form ou relatório) e chamá-los a partir de outros procedimentos, funções e gatilhos (veja a próxima página) na mesma aplicação sempre que necessário.

Observação: Uma função é similar a um procedimento, exceto que uma função deve retornar um valor. Os procedimentos e funções são abordados no próximo curso sobre PL/SQL.

Construções de Programa



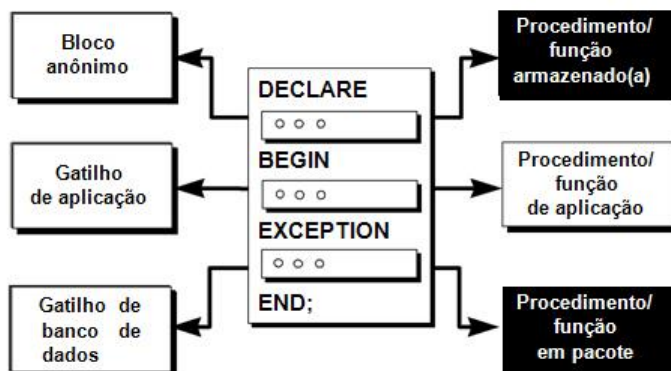
ORACLE®

Construções de Programa

A tabela a seguir descreve em linhas gerais uma variedade de construções de programa PL/SQL que usam o bloco PL/SQL básico. Eles estão disponíveis de acordo com o ambiente no qual eles são executados.

Construção de Programa	Descrição	Disponibilidade
Bloco anônimo	O bloco PL/SQL não nomeado é incorporado em uma aplicação ou é emitido interativamente	Todos os ambientes PL/SQL
Função ou procedimento armazenado	O bloco PL/SQL nomeado armazenado no Oracle Server que pode aceitar parâmetros e pode ser chamado repetidamente pelo nome	Oracle Server
Função ou procedimento de aplicação	O bloco PL/SQL nomeado armazenado na aplicação Oracle Developer ou na biblioteca compartilhada que pode aceitar parâmetros e pode ser chamado repetidamente pelo nome	Componentes do Oracle Developer — por exemplo, Forms
Pacote	Módulo PL/SQL nomeado que agrupa procedimentos, funções e identificadores relacionados	Componentes do Oracle Server e Oracle Developer — por exemplo, Forms

Construções de Programa



ORACLE®

Construções de Programa (continuação)

Construção de Programa	Descrição	Disponibilidade
Gatilho de banco de dados	O bloco PL/SQL que é associado com uma tabela de banco de dados e que é acionado automaticamente quando disparado pela instrução DML	Oracle Server
Gatilho de aplicação	O bloco PL/SQL que é associado com uma evento de aplicação e que é acionado automaticamente	Componentes do Oracle Developer — por exemplo, Forms

Uso de Variáveis

Usar variáveis para:

- Armazenamento temporário de dados
- Manipulação de valores armazenados
- Reutilização
- Facilidade de manutenção

Uso de Variáveis

Com o código PL/SQL, você poderá declarar variáveis e depois usá-las em instruções procedurais e SQL onde uma expressão possa ser usada.

- Armazenamento temporário de dados

Os dados podem ser armazenados temporariamente em uma ou mais variáveis para uso quando na validação da entrada de dados para processamento posterior no processo de fluxo de dados.

- Manipulação de valores armazenados

As variáveis podem ser usadas para cálculo e manipulação de outros dados sem acessar o banco de dados.

- Reutilização

Quando declaradas, as variáveis podem ser usadas repetidamente em uma aplicação simplesmente referenciando-as em outras instruções, incluindo outras instruções declarativas.

- De fácil manutenção

Ao usar %TYPE e %ROWTYPE (mais informações sobre %ROWTYPE são abordadas em uma lição subsequente), você declara variáveis, baseando as declarações nas definições das colunas de banco de dados. As variáveis PL/SQL ou variáveis de cursor anteriormente declaradas no escopo atual poderão usar também os atributos %TYPE e %ROWTYPE como especificadores de tipos de dados. Se uma definição subjacente for alterada, a declaração de variável se altera de acordo durante a execução. Isso permite a independência dos dados, reduz custos de manutenção e permite que os programas se adaptem, conforme o banco de dados for alterado, para atender às novas necessidades comerciais.

Tratando Variáveis em PL/SQL

- Declarar e inicializar as variáveis na seção de declaração.
- Atribuir novos valores às variáveis na seção executável.
- Passar valores aos blocos PL/SQL através de parâmetros.
- Ver os resultados pelas variáveis de saída.

Tratando Variáveis em PL/SQL

- Declarar e inicializar as variáveis na seção de declaração.

Você poderá declarar variáveis na parte declarativa de qualquer subprograma, pacote ou bloco

PL/SQL. As declarações alocam espaço de armazenamento para um valor, especificam seus tipos de dados e nomeiam a localização de armazenamento para que você possa referenciá-

los. As declarações poderão também atribuir um valor inicial e impor a restrição NOT NULL.

- Atribuir novos valores às variáveis na seção executável.

- O valor existente da variável é substituído pelo novo.

- Não são permitidas referências posteriores. Você deve declarar um variável antes de referenciá-la em outras instruções, incluindo outras instruções declarativas.

- Passar valores aos subprogramas PL/SQL através de parâmetros.

Há três modos de parâmetros, IN (o default), OUT e IN OUT. Você deve usar o parâmetro IN para passar valores aos subprogramas que estão sendo chamados. Você deve usar o parâmetro OUT para retornar valores ao chamador de um subprograma. E você deve usar o parâmetro IN OUT para passar valores iniciais ao subprograma que está sendo chamado e para retornar

valores ao chamador. Os parâmetros dos subprogramas IN e OUT são abordados em outro curso.

- Ver os resultados em um bloco PL/SQL através de variáveis de saída.

Você poderá usar variáveis de referências para a entrada ou saída em instruções de manipulação de dados SQL.

Tipos de Variáveis

- Variáveis PL/SQL:
 - Escalar
 - Composta
 - Referência
 - LOB (objetos grandes)
- Variáveis Não-PL/SQL: Variáveis de ligação e de host

Todas as variáveis PL/SQL têm um tipo de dados, o qual especifica um formato de armazenamento,

restrições e uma faixa válida de valores. A linguagem PL/SQL suporta quatro categorias de tipos de dados — escalar, composta, referencial e LOB (objeto grande) — que você pode usar para declarar variáveis, constantes e indicadores.

- Os tipos de dados escalares armazenam um único valor. Os principais tipos de dados são aqueles que correspondem aos tipos de coluna nas tabelas do Oracle Server; a linguagem PL/SQL também suporta variáveis booleanas.

- Os tipos de dados compostos como, por exemplo, os registros, permitem que os grupos de campos sejam definidos e manipulados nos blocos PL/SQL. Os tipos de dados compostos são mencionados apenas brevemente neste curso.

- Os tipos de dados referenciais armazenam valores, chamados de indicadores, que designam outros itens de programa. Os tipos de dados referenciais não são abordados neste curso.

- Os tipos de dados LOB armazenam valores, chamados de endereços, que especificam a localização de objetos grandes (imagens gráficas, por exemplo) que são armazenadas fora de linha. Os tipos de dados LOB são abordados apenas brevemente neste curso.

As variáveis não-PL/SQL incluem variáveis da linguagem de host declaradas em programas do pré- compilador, campos de tela nas aplicações Forms e variáveis de host SQL*Plus.

Para obter mais informações sobre os LOBs, consulte o PL/SQL User's Guide and Reference, Release 8, "Fundamentals".

Tipos de Variáveis

- Variáveis PL/SQL:

- Escalar
- Composta
- Referência
- LOB (objetos grandes)

- Variáveis Não-PL/SQL: Variáveis de ligação e de host

Usando Variáveis SQL*Plus nos Blocos PL/SQL

O código PL/SQL não tem capacidade de entrada/saída própria. Você deverá contar com o ambiente no qual o código PL/SQL estiver sendo executado para passar valores para e de um bloco PL/SQL.

No ambiente SQL*Plus, as variáveis de substituição SQL*Plus permitem que partes da sintaxe de comando sejam armazenadas e depois editadas no comando antes de executá-lo. As variáveis de substituição são variáveis que você pode usar para passar valores de tempo de execução, números

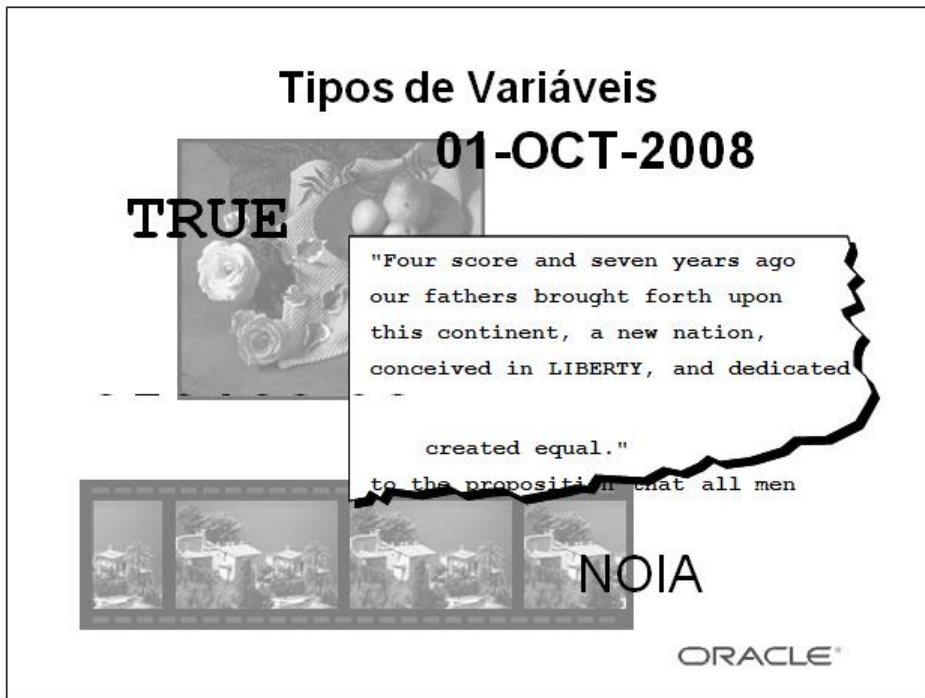
ou caracteres, a um bloco PL/SQL. Você poderá referenciá-las em um bloco PL/SQL com um "e"

comercial precedendo-as do mesmo modo que você referencia as variáveis de substituição

SQL*Plus em uma instrução SQL. Os valores do texto são substituídos no bloco PL/SQL antes que esse bloco seja executado. Por isso, você não poderá substituir os valores diferentes para as

variáveis de substituição usando um loop. Apenas um valor substituirá os valores de substituição.

As variáveis de host (ou "ligação") SQL*Plus podem ser usadas para passar valores de tempo de execução do bloco PL/SQL de volta para o ambiente SQL*Plus. Você poderá referenciá-las em um bloco PL/SQL com dois-pontos precedendo-as. As variáveis de ligação são discutidas em mais detalhes mais adiante nesta lição.



Tipos de Variáveis

O slide ilustra os seguintes tipos de dados de variável:

- TRUE representa um valor booleano.
- -OCT-2008 representa uma variável DATE (data).
- A fotografia representa uma variável BLOB.
- O texto de um discurso representa uma variável LONG RAW.
- 256120.08 representam um tipo de dados NUMBER com precisão e escala.
- O filme representa uma variável BFILE.
- O nome da cidade representa uma variável VARCHAR2.

Declarando Variáveis PL/SQL

Sintaxe

```
identificador [CONSTANT] tipo de dados [NOT NULL]
[:= | DEFAULT expr];
```

Exemplos

```
Declare
v_hiredate      DATE;
v_deptno        NUMBER(2)    NOT    NULL    :=    10;
v_location      VARCHAR2(13) := 'Atlanta';
c_comm          CONSTANT NUMBER := 1400;
```

Declarando Variáveis PL/SQL

Você precisa declarar todos os identificadores PL/SQL na seção de declaração antes de referenciá-los no bloco PL/SQL. Você tem a opção de atribuir um valor inicial. Você não precisa atribuir um valor a uma variável para declará-la. Se você referenciar outras variáveis em uma declaração, você deverá estar certo de declará-las separadamente em uma instrução anterior.

Na sintaxe:

Identificador é o nome da variável

CONSTANT restringe as variáveis para que o seu valor não possa ser alterado; as constantes devem ser inicializadas

tipo de dados são tipos de dados escalares, compostos, referenciais ou LOB (Esse curso aborda apenas os tipos de dados escalares e compostos.)

NOT NULL restringe a variável para que ela contenha um valor (variáveis NOT NULL devem ser inicializadas.)

expr é uma expressão PL/SQL que pode ser uma literal, uma outra variável ou uma expressão que envolve operadores e funções

Declarando Variáveis PL/SQL

Diretrizes

- Seguir as convenções de nomeação.
- Inicializar as variáveis designadas como NOT NULL e CONSTANT.
- Inicializar os identificadores usando o operador de atribuição (:=) ou a palavra reservada DEFAULT.
- Declarar no máximo um identificador por linha.

Diretrizes

A expressão atribuída pode ser uma literal, uma outra variável ou uma expressão que envolve operadores e funções.

- Nomeie o identificador de acordo com as mesmas regras usadas por objetos SQL.
- Você poderá usar convenções de nomeação — por exemplo, v_name para representar uma variável e c_name e representar uma variável constante.
- Inicialize a variável em uma expressão com o operador de atribuição (:=) ou, equivalentemente, com a palavra reservada DEFAULT. Se você não atribuir um valor inicial, a nova variável conterá NULL por default até que você o atribua mais tarde.
- Se você usar a restrição NOT NULL, você deve atribuir um valor.

- A declaração de apenas um identificador por linha torna o código mais fácil de ler e de manter.
- Em declarações constantes, a palavra-chave **CONSTANT** deve preceder o especificador de tipo. A declaração a seguir nomeia a constante **NUMBER**, subtipo **REAL** e atribui o valor de 50000 à constante. Uma constante deve ser inicializada em sua declaração; senão, você obterá uma mensagem de erro de compilação quando a declaração for elaborada (compilada).

```
v_sal CONSTANT REAL := 50000.00;
```

Regras para Nomeação

- Duas variáveis podem ter o mesmo nome, contanto que elas estejam em blocos diferentes.
- O nome da variável (identificador) não deve ser igual ao nome das colunas de tabela usado no bloco.

```
DECLARE
    empno NUMBER(4);

BEGIN
    SELECT empno
    INTO empno
    FROM emp
    WHERE ename = 'SMITH';

END;
```

Regras para Nomeação

Dois objetos podem ter o mesmo nome, contanto que eles sejam definidos em diferentes blocos. Onde eles coexistirem, apenas o objeto declarado no bloco atual pode ser usado.

Você não deve escolher o nome (identificador) para uma variável igual ao nome das colunas de tabela usado no bloco. Se as variáveis PL/SQL ocorrerem nas instruções SQL e tiverem o mesmo nome que uma coluna, o Oracle Server supõe que seja a coluna que esteja sendo referenciada. Embora o código de exemplo no slide funcione, o código criado usando o mesmo nome para uma tabela de banco de dados e para uma variável não é fácil de ler ou manter.

Considere a adoção de uma convenção de nomeação de vários objetos como o seguinte exemplo. O uso de v_ as como um prefixo que representa uma variável e g_ representando uma variável global impede conflitos de nomeação com os objetos do banco de dados.

```
DECLARE
    v_hiredate    date;
    g_deptno      number(2) NOT NULL := 10;
BEGIN
...

```

Observação: Os identificadores não devem ter mais de 30 caracteres. O primeiro caracter deve ser uma letra; os restantes podem ser letras, números ou símbolos especiais.

Atribuindo Valores às Variáveis

Sintaxe

```
identificador := expr;
```

Exemplos

Determine uma data de admissão predefinida para novos empregados.

```
v_hiredate := '31-DEC-98';
```

Defina o nome do funcionário para Maduro.

```
v_ename := 'Maduro';
```


Atribuindo Valores às Variáveis

Para atribuir ou reatribuir um valor a uma variável, crie uma instrução de atribuição PL/SQL. Você deve nomear explicitamente a variável para receber o novo valor à esquerda do operador de atribuição (: =).

Na sintaxe:

identificador
expr

é o nome da variável escalar
pode ser uma chamada variável, literal ou
funcional, mas não uma coluna de banco
de dados

Os exemplos de atribuição de valor da variável são definidos com se segue:

- Defina o identificador v_hiredate para um valor de 31-DEC-98.
- Armazene o nome "Maduro" no identificador v_ename.

Uma outra maneira de atribuir valores às variáveis é selecionar ou trazer valores de banco de dados

para dentro delas. O exemplo a seguir calcula um bônus de 10% ao seleciona o salário do funcionário:

```
SQL> SELECT sal * 0.10  
2 INTO v_bonus  
3 FROM emp  
4 WHERE empno = 7369;
```

Depois você poderá usar a variável v_bonus em um outro computador ou inserir seu valor em uma tabela de banco de dados.

Observação: Para atribuir um valor em uma variável do banco de dados, use uma instrução SELECT ou FETCH.

Palavras-chave e Inicialização de Variáveis

Usando:

- Operador de atribuição (: =)
- Palavra-chave DEFAULT
- Restrição NOT NULL

As variáveis são inicializadas toda vez que um bloco ou subprograma for informado. Por default, as

variáveis são inicializadas em NULL. A não ser que você expressamente inicialize uma variável, os seus valores são indefinidos.

- Use o operador de atribuição (:=) para as variáveis que não têm valor típico.

```
v_hiredate := '15-SEP-1999'
```

Observação: Essa atribuição é possível apenas em Oracle8i. As versões anteriores podem requerer o uso da função TO_DATE.

Como o formato de data default definido no Oracle Server pode diferir de banco de dados para banco de dados, você poderá querer atribuir valores de data de uma maneira genérica, como no exemplo anterior.

- **DEFAULT:** Você poderá usar a palavra-chave DEFAULT em vez do operador de atribuição para inicializar as variáveis. Use DEFAULT para as variáveis que têm um valor típico.

```
g_mgr NUMBER(4) DEFAULT 7839;
```

- **NOT NULL:** Imponha a restrição NOT NULL quando a variável tiver que conter um valor.

Você não poderá atribuir valores nulos a uma variável definida como NOT NULL. A restrição

NOT NULL deve ser seguida por uma cláusula de inicialização.

```
v_location VARCHAR2(13) NOT NULL := 'CHICAGO';
```

(continuação)

Observação: As literais de string devem ser incluídas entre aspas simples — por exemplo, 'Hello, world'. Se houver uma aspa simples na string, crie uma aspa simples duas vezes — por exemplo, para inserir um valor FISHERMAN'S DRIVE, a string seria 'FISHERMAN'S DRIVE'.

Tipos de Dados Escalares

- Armazenar um valor único
- Não ter componentes internos

Tipos de Dados Escalares

Um tipo de dados escalares armazena um valor único e não possui componentes internos. Os tipos de dados escalares podem ser classificados em quatro categorias: número, caractere, data e booleano. Os tipos de dados de caractere e número possuem subtipos que associam um tipo básico a uma restrição. Por exemplo, INTEGER e POSITIVE são subtipos do tipo básico NUMBER.

Para obter mais informações e completar a lista de tipos de dados escalares, consulte o PL/SQL User's Guide and Reference, Release 8, "Fundamentals".

Tipos de Dados Escalares Básicos

- VARCHAR2 (maximum_length)
- NUMBER [(precisão, escala)]
- DATE
- CHAR [(maximum_length)]
- LONG
- LONG RAW
- BOOLEAN
- BINARY_INTEGER
- PLS_INTEGER

Tipos de Dados Escalares Básicos

Tipo de Dados	Descrição
VARCHAR2 (<i>maximum_length</i>)	Tipo básico para dados de caractere de tamanho variável com até 32.767 bytes. Não há tamanho default para as constantes e variáveis VARCHAR2.
NUMBER [(<i>precisão, escala</i>)]	Tipo básico para números fixos e de ponto flutuante.
DATE	Tipo básico para datas e horas. Valores DATE incluem a hora do dia em segundos desde a meia-noite. A faixa para datas é entre 4712 A.C. e 9999 D.C.
CHAR [(<i>maximum_length</i>)]	Tipo básico para dados de caractere de tamanho fixo até 32.767 bytes. Se você não especificar um <i>comprimento_máx</i> , o tamanho default será definido como 1.
LONG	Tipo básico para dados de caracter de tamanho variável até 32.760 bytes. A largura máxima de uma coluna de banco de dados LONG é de 2.147.483.647 bytes.

Tipos de Dados Escalares Básicos

- VARCHAR2 (maximum_length)
- NUMBER [(precisão, escala)]
- DATE
- CHAR [(maximum_length)]
- LONG
- LONG RAW
- BOOLEAN
- BINARY_INTEGER
- PLS_INTEGER

Tipos de Dados Escalares Básicos (continuação)

Tipo de Dados	Descrição
LONG RAW	Tipo básico para dados binários e strings de byte de até 32.760 bytes. Dados LONG RAW são não interpretados pelo código PL/SQL.
BOOLEAN	Tipo básico que armazena um de três possíveis valores usados para cálculos lógicos: TRUE, FALSE ou NULL.
BINARY_INTEGER	Tipo básico para inteiros entre -2.147.483.647 e 2.147.483.647.
PLS_INTEGER	Tipo básico para inteiros designados entre -2.147.483.647 e 2.147.483.647. Os valores PLS_INTEGER requerem menos armazenamento e são mais rápidos que os valores NUMBER e BINARY_INTEGER.

Observação: O tipo de dados LONG é similar ao VARCHAR2, exceto que pelo fato de que o comprimento máximo de um valor

LONG é de 32.760 bytes. Por isso, valores maiores que 32.760 bytes não poderão ser selecionados de uma coluna de banco de dados LONG para uma variável LONG PL/SQL.

Declarando Variáveis Escalares

Exemplos

```
v_job          VARCHAR2(9);  
v_count        BINARY_INTEGER := 0;  
v_total_sal    NUMBER(9,2) := 0;  
v_orderdate    DATE := SYSDATE + 7;  
c_tax_rate     CONSTANT NUMBER(3,2) := 8.25;  
v_valid        BOOLEAN NOT NULL := TRUE;
```

Declarando Variáveis Escalares

Os exemplos de declaração de variável mostrados no slide são definidos como se segue:

- **v_job**: Variável declarada para armazenar o cargo de um funcionário.
- **v_count**: Variável declarada para contar as iterações de um loop e inicializar a variável em 0.
- **v_total_sal**: Variável declarada para acumular o salário total de um departamento e inicializar a variável em 0.
- **v_orderdate**: Variável declarada para armazenar a data de entrega de uma ordem de compra e inicializar a variável em uma semana a partir do dia de hoje.
- **c_tax_rate**: Constante declarada para a taxa de imposto, que nunca se altera no bloco PL/SQL.
- **v_valid**: Indicador declarado para indicar se os dados são válidos ou inválidos e inicializar a variável como TRUE.

O Atributo %TYPE

- Declarar uma variável de acordo com:
 - Uma definição de coluna de banco de dados
 - Uma outra variável anteriormente declarada
- Prefixo %TYPE com:
 - A coluna e tabela do banco de dados
 - O nome da variável anteriormente declarada

O Atributo %TYPE

Ao declarar variáveis PL/SQL para armazenar os valores de coluna, você deverá garantir que a variável seja de precisão e tipo de dados corretos. Caso contrário, uma mensagem de erro PL/SQL ocorrerá durante a execução.

Em vez de embutir no código o tipo de dados e a precisão de uma variável, você poderá usar o atributo %TYPE para declarar uma variável de acordo com uma outra coluna de banco de dados ou variável declarada anteriormente. O atributo %TYPE é usado com mais frequência quando o valor armazenado na variável for derivado de uma tabela no banco de dados ou se ocorre alguma gravação na variável. Para usar o atributo no lugar do tipo de dados necessário na declaração da variável, crie um prefixo para ele com o nome da coluna e da tabela de banco de dados. Se estiver referenciando uma variável declarada anteriormente, crie um prefixo para o nome da variável relativa ao atributo.

O código PL/SQL determina o tipo de dados e o tamanho da variável quando o bloco for compilado, de modo que ele seja sempre compatível com a coluna usada para preenchê-lo. Isso definitivamente é uma vantagem para criar e manter o código, porque não há necessidade de se preocupar com alterações no tipo de dados da coluna feitos no nível de banco de dados. Você poderá também declarar uma variável de acordo com uma outra declarada anteriormente pela prefixação do nome da variável relativa ao atributo.

Declarando Variáveis com o Atributo %TYPE

Exemplos

```
...  
    v_ename          emp.ename%TYPE;  
    v_balance        NUMBER(7,2);  
    v_min_balance    v_balance%TYPE := 10;  
...
```

Declarando Variáveis com o Atributo %TYPE

Declare as variáveis para armazenar o nome de um funcionário.

```
...  
    v_ename          emp.ename%TYPE;  
    ...
```

Declare as variáveis para armazenar o saldo de uma conta corrente bancária, assim como um saldo mínimo, que inicie em 10.

```
...  
    v_balance        NUMBER(7,2);  
    v_min_balance    v_balance%TYPE := 10;  
    ...
```

Uma restrição da coluna NOT NULL não se aplica a variáveis declaradas usando %TYPE. Por isso, se você declarar uma variável que estiver usando o atributo %TYPE usando uma coluna de banco de dados definida como NOT NULL, você poderá atribuir um valor NULL à variável.

Declarando Variáveis Booleanas

- Apenas os valores TRUE, FALSE e NULL podem ser atribuídos a uma variável booleana.
- As variáveis são conectadas pelos operadores lógicos AND, OR e NOT.
- As variáveis sempre produzem TRUE, FALSE ou NULL.
- Expressões aritméticas, de caracteres e de dados podem ser usadas para retornar um valor booleano.

Declarando Variáveis Booleanas

Com o código PL/SQL você poderá comparar variáveis em instruções SQL e procedurais. Essas comparações, chamadas expressões booleanas, consistem em expressões simples ou complexas separadas por operadores relacionais. Em uma instrução SQL, você poderá usar expressões booleanas para especificar as linhas em uma tabela que são afetadas por uma instrução. Em instruções

procedurais, as expressões booleanas são a base para o controle condicional. NULL significa um valor ausente, inaplicável ou desconhecido.

Exemplos

```
v_sal1 := 50000;  
v_sal2 := 60000;
```

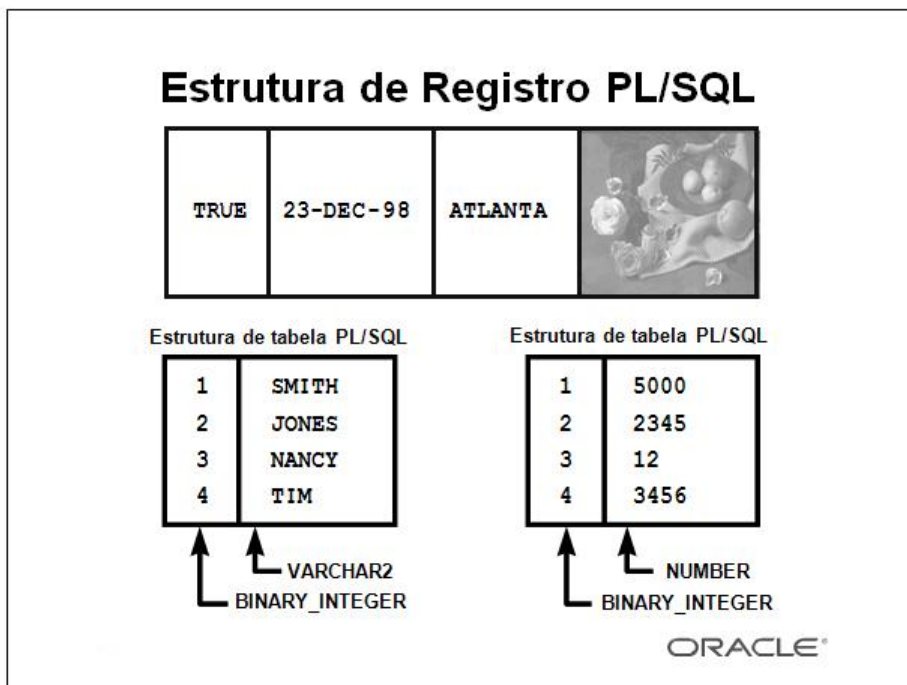
A expressão a seguir produz TRUE:

```
v_sal1 < v_sal2
```

Declare e inicialize uma variável booleana:

```
v_comm_sal BOOLEAN := (v_sal1 < v_sal2);
```

Estrutura de Registro PL/SQL

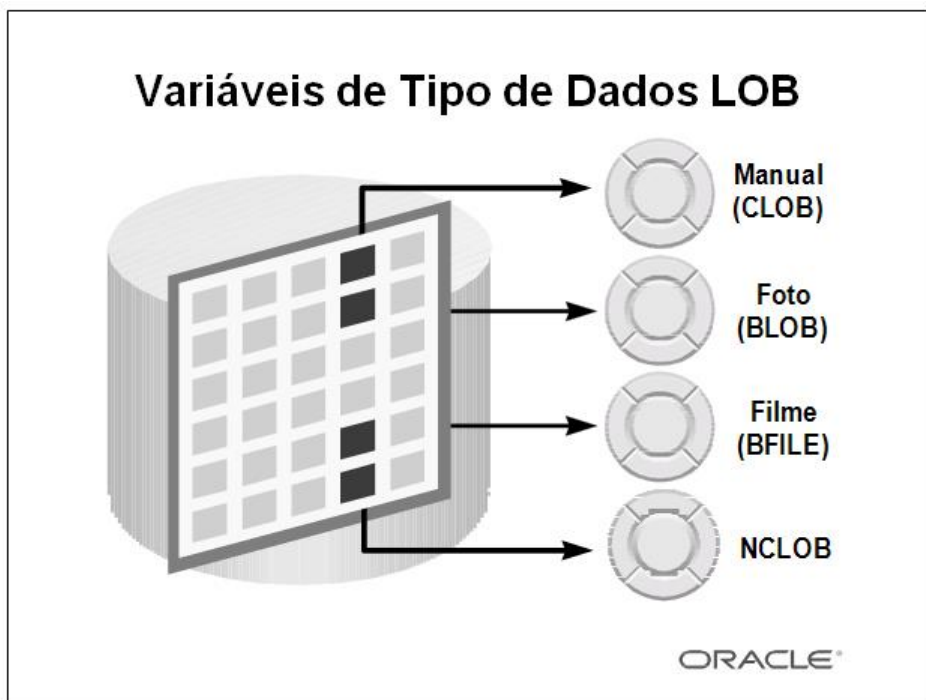


Tipos de Dados Compostos

Os tipos de dados compostos (também conhecidos como conjuntos) são TABLE, RECORD, NESTED TABLE e VARRAY. Utilize o tipo de dados RECORD para manipular os dados relacionados, porém diferentes, como uma unidade lógica. Utilize o tipo de dados TABLE para fazer referência e manipular conjuntos de dados como um objeto inteiro. Os dois tipos de dados RECORD e TABLE são cobertos em detalhe em uma lição subsequente. Os tipos de dados NESTED TABLE e VARRAY não são abordados neste curso.

Para obter mais informações, consulte o PL/SQL User's Guide and Reference, Release 8, "Collections and Records".

Variáveis de Tipo de Dados LOB



Variáveis de Tipo de Dados LOB

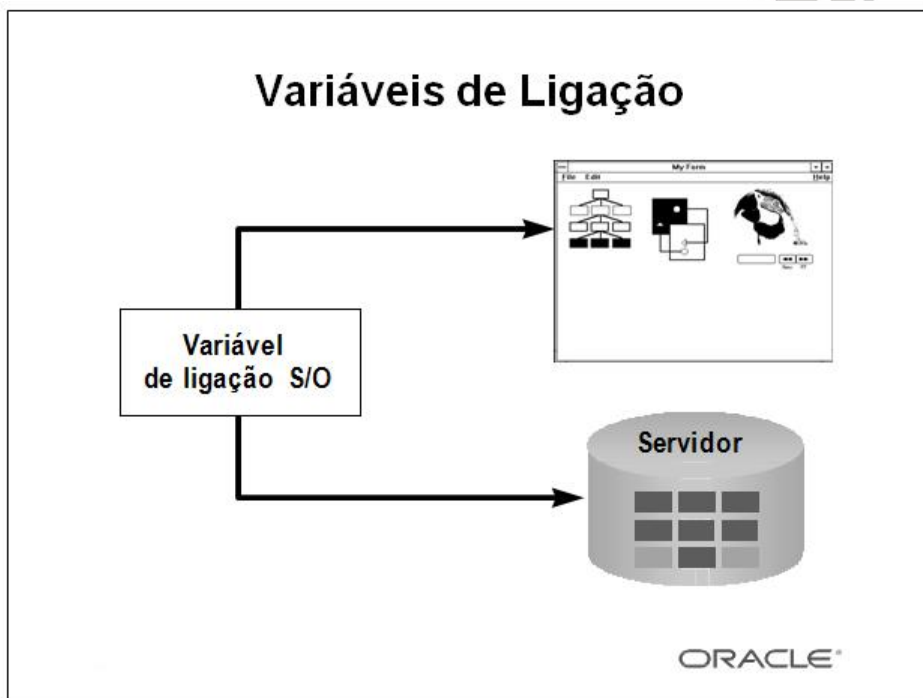
Com o tipo de dados LOB (large object, objeto grande) do Oracle8, você poderá armazenar blocos de dados não estruturados (como, por exemplo, texto, imagens gráficas, vídeos e formatos de arquivo para armazenar sons) de até 4 gigabytes em tamanho. Os tipos de dados LOB fornecem acesso eficiente, aleatório e em intervalos aos dados, podendo ser atributos de um tipo de objeto.

Os LOBs também suportam acesso aleatório a dados.

- O tipo de dados CLOB (character large object, objeto grande de caractere) é usado para armazenar blocos grandes de dados com caracteres de um único byte no banco de dados.
- O tipo de dados BLOB (binary large object, objeto grande binário) é usado para armazenar objetos binários grandes no banco de dados em linha (dentro de uma linha de tabela) ou fora de linha (fora da linha de tabela).

- O tipo de dados BFILE (binary file, arquivo binário) é usado para armazenar objetos grandes binários em arquivos do sistema operacional fora do banco de dados.
- O tipo de dados NCLOB (objeto grande de caractere do idioma nacional) é usado para armazenar blocos grandes de dados NCHAR de byte único ou de bytes múltiplos de largura fixa no banco de dados, dentro e fora de linha.

Variáveis de Ligação



Variáveis de Ligação

Uma variável de ligação é uma variável que você declara em um ambiente de host e usa para passar valores de tempo de execução, número ou caractere, para ou de um ou mais programas PL/SQL, os quais podem usá-la como usariam qualquer outra variável. Você poderá referenciar variáveis declaradas em ambientes de host ou chamada em instruções PL/SQL, a não ser que a instrução esteja em um procedimento, função ou pacote. Isso inclui as variáveis de linguagem declaradas em programas do pré-compilador,

campos de tela em aplicações de Form do Oracle Developer e as variáveis de ligação SQL*Plus.

Criando Variáveis de Ligação

Para declarar uma variável de ligação no ambiente SQL*Plus, você deve usar o comando VARIABLE. Por exemplo, você poderá declarar uma variável de tipo NUMBER e VARCHAR2 como se segue:

```
VARIABLE return_code NUMBER
VARIABLE return_msg      VARCHAR2(30)
```

Tanto o código SQL quanto o SQL*Plus poderão referenciar a variável de ligação, e o código SQL*Plus poderá exibir seus valores.

Exibindo Variáveis de Ligação

Para exibir o valor atual das variáveis de ligação no ambiente SQL*Plus, você deverá usar o comando PRINT. Entretanto, PRINT não poderá ser usado em um bloco PL/SQL como um comando SQL*Plus. O exemplo a seguir ilustra um comando PRINT:

```
SQL> VARIABLE g_n NUMBER
...
SQL> PRINT g_n
```

Referenciando Variáveis

Não-PL/SQL

Armazenar o salário anual em uma variável de host SQL*Plus.

```
:g_monthly_sal := v_sal / 12;
```

- Referenciar variáveis não-PL/SQL como variáveis de host.
- Criar um prefixo para as referências com dois-pontos (:).

Atribuindo Valores às Variáveis

Para referenciar variáveis de host, você deve criar prefixos para as referências com dois-pontos (:) para distingui-las das variáveis PL/SQL.

Exemplo

Esse exemplo calcula o salário mensal, de acordo com o salário anual fornecido pelo usuário. Esse script contém tanto os comandos SQL*Plus quanto um bloco PL/SQL completo.

```
VARIABLE      g_monthly_sal      NUMBER
ACCEPT        p_annual_sal      PROMPT 'Please enter the
annual salary: '

DECLARE
v_sal NUMBER(9,2) := &p_annual_sal; BEGIN
:g_monthly_sal := v_sal/12; END;
/

PRINT g_monthly_sal
```

DBMS_OUTPUT.PUT_LINE

- Um procedimento de pacote fornecido pela Oracle
- Uma alternativa para exibir dados a partir de um bloco PL/SQL
- Deverá ser ativado em SQL*Plus com SET SERVEROUTPUT ON

Uma outra Opção

Você viu que pode declarar uma variável de host, referenciá-la em um bloco PL/SQL e exibir o conteúdo dela em SQL*Plus usando o comando PRINT. Uma outra opção para exibir as informações de um bloco PL/SQL é DBMS_OUTPUT.PUT_LINE. O procedimento DBMS_OUTPUT é um pacote fornecido pela Oracle e o PUT_LINE é um procedimento no pacote.

Em um bloco PL/SQL, referencie o procedimento DBMS_OUTPUT.PUT_LINE e, entre parênteses, as informações que você deseja imprimir na tela. O pacote deve primeiro ser ativado na sessão SQL*Plus. Para fazer isso, execute comando SET SERVEROUTPUT ON do código SQL*Plus.

Exemplo

Esse script calcula o salário mensal e imprime-as na tela, usando o procedimento DBMS_OUTPUT.PUT_LINE.

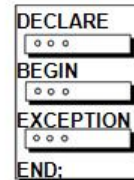
```
SET SERVEROUTPUT ON
ACCEPT p_annual_sal PROMPT 'Please enter the annual
salary: '

DECLARE
v_sal NUMBER(9,2) := &p_annual_sal; BEGIN
v_sal := v_sal/12;
DBMS_OUTPUT.PUT_LINE ('The monthly salary is ' ||
TO_CHAR(v_sal)); END;
/
```

Sumário

Sumário

- Os blocos PL/SQL são compostos pelas seguintes seções:
 - Declarativa (opcional)
 - Executável (necessária)
 - Tratamento de exceção (opcional)
- Um bloco PL/SQL pode ser um procedimento, função ou bloco anônimo.



ORACLE®

Sumário

Um bloco PL/SQL é a unidade básica e sem nome de um programa PL/SQL. Ele consiste em um conjunto de instruções SQL ou PL/SQL e desempenha uma função lógica única. A parte declarativa é a primeira parte de um bloco PL/SQL e é usada para declarar objetos como, por exemplo, variáveis, constantes, cursores e definições de situações de erro chamadas de exceções. A parte executável é a parte obrigatória de um bloco PL/SQL e contém instruções SQL e PL/SQL para consultar e manipular dados. A parte de tratamento de exceção está embutida na parte executável de um bloco e é colocada no final da parte executável.

Um bloco PL/SQL anônimo é a unidade básica, sem nome de um programa PL/SQL. Os procedimentos e funções podem ser compilados separadamente e armazenadas permanentemente em um banco de dados Oracle, prontos para serem executados.

Sumário

- Identificadores PL/SQL:
 - São definidos na seção declarativa
 - Podem ser tipo de dados escalares, compostos, referenciais ou LOB
 - Podem ser baseados em estruturas de uma outra variável ou objeto de banco de dados
 - Podem ser inicializados

Sumário (continuação)

Todos os tipos de dados PL/SQL são escalares, compostos, referenciais ou LOB. Os tipos de dados escalares não têm quaisquer componentes neles, enquanto que os tipos de dados sim. As variáveis PL/SQL são declaradas e inicializadas na seção declarativa.

Capítulo 02

Criando Instruções

Executáveis

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Reconhecer a importância da seção executável
- Criar instruções na seção executável
- Criar regras de blocos aninhados
- Executar e testar um bloco PL/SQL
- Usar convenções de codificação

Objetivo da Lição

Nesta lição, você aprenderá como criar códigos executáveis no bloco PL/SQL. Você também aprenderá as regras de aninhamento dos blocos PL/SQL de código, assim como executar e testar seu código PL/SQL.

Diretrizes e Sintaxe de Bloco PL/SQL

- As instruções podem continuar por várias linhas.
- As unidades lexicais podem ser separadas por:
 - Espaços
 - Delimitadores
 - Identificadores
 - Literais
 - Comentários

Diretrizes e Sintaxe de Bloco PL/SQL

Como o PL/SQL é uma extensão do SQL, as regras gerais de sintaxe que se aplicam ao SQL, também se aplicam à linguagem PL/SQL.

- As unidades lexicais (por exemplo, identificadores e literais) poderão ser separadas por um ou mais espaços ou outros delimitadores que não podem ser confundidos como parte da unidade lexical.

Você não poderá embutir espaços em unidades lexicais, exceto para literais de string e comentários.

- As instruções podem ser divididas pelas linhas, mas palavras-chave não.

Delimitadores

Os Delimitadores são símbolos simples ou compostos que têm significado especial para o PL/SQL.

Símbolos Simples

Símbolos Compostos

Símbolo	Significado	Símbolo	Significado
+	Operador de adição	<>	Operador relacional
-	Operador de subtração/negação	!=	Operador relacional
*	Operador de multiplicação		Operador de concatenação
/	Operador de divisão	--	Indicador de comentário de uma única linha
=	Operador relacional	/*	Delimitador de comentário inicial
@	Indicador de acesso remoto	*/	Delimitador de comentário final
;	Finalizador de instrução	:=	Operador de atribuição

Para obter mais informações, consulte *PL/SQL User's Guide and Reference*, Release 8, "Fundamentals".

Diretrizes e Sintaxe de Bloco PL/SQL

Identificadores

- Podem conter até 30 caracteres
- Não podem conter palavras reservadas, a não ser que estejam entre aspas duplas
- Devem ser iniciados por um caractere alfabético
- Não devem ter o mesmo nome de uma coluna de tabela de banco de dados

Identificadores

Os Identificadores são usados para nomear itens e unidades do programa PL/SQL, os quais podem incluir constantes, variáveis, exceções, cursores, variáveis de cursores, subprogramas e pacotes.

- Os identificadores podem conter até 30 caracteres, mas eles deverão ser iniciados por um caractere alfabético.

- Não escolha o mesmo nome para os identificadores e para as colunas na tabela usada no bloco.

Se os identificadores PL/SQL estiverem nas mesmas instruções SQL e tiverem o mesmo nome de uma coluna, o Oracle supõe que ele é a coluna que está sendo referenciada.

- As palavras reservadas não poderão ser usadas como identificadores, a não ser que elas estejam entre aspas duplas (por exemplo, "SELECT").

- As palavras reservadas deverão ser criadas em letra maiúscula para facilitar a leitura. Para obter uma lista completa de palavras reservadas, consulte o PL/SQL User's Guide and Reference, Release 8, "Appendix F".

Diretrizes e Sintaxe de Bloco PL/SQL

- Literais

- Caracteres e literais de data devem estar entre aspas simples.

```
v_ename := 'Henderson';
```

- Os números poderão ser valores simples ou notações científicas.

- Um bloco PL/SQL é finalizado por uma barra (/) em uma linha sozinha.

Literais

- Uma literal é um valor booleano, um valor numérico explícito, um caractere ou uma string não representados por um identificador.

- As literais de caracteres incluem todos os caracteres imprimíveis no conjunto de caracteres do PL/SQL: letras, numerais, espaços e símbolos especiais.

- As literais numéricas poderão ser representadas por um valor simples (por exemplo, -32.5) ou por uma notação científica (por exemplo, 2E5, significando $2 * (10 \text{ à potência de } 5) = 200000$).

- Um bloco PL/SQL é finalizado por uma barra (/) em uma linha sozinha.

Comentando Código

- Crie prefixos de dois hifens (--) para comentários de uma única linha.
- Coloque os comentários de várias linhas entre os símbolos /* e */.

Exemplo

```
...  
    v_sal NUMBER (9,2);  
BEGIN  
    /* Compute the annual salary based on the  
       monthly salary input from the user */  
    v_sal := &p_monthly_sal * 12;  
END; -- This is the end of the block
```

Comentando Código

Comente códigos para documentar cada fase e para auxiliar na depuração. Comente o código PL/SQL com dois hifens (--) se o comentário estiver em uma única linha ou coloque o comentário entre os símbolos /* e */ se o comentário englobar várias linhas. Os comentários são estritamente informativos e não impõem nenhuma condição ou comportamento nos dados ou na lógica comportamental. Os comentários bem colocados são extremamente valiosos para a boa leitura do código e para a futura manutenção dele.

Exemplo

No exemplo no slide, a linha entre /* e */ é o comentário que explica o código que a segue.

Funções SQL em PL/SQL

Funções SQL em PL/SQL

- Disponível nas instruções procedurais:
 - Número de uma única linha
 - Caractere de uma única linha
 - Conversão de tipo de dados
 - Data
- Não disponível nas instruções procedurais:
 - DECODE
 - Funções de grupo

ORACLE®

Funções SQL em PL/SQL

A maioria das funções disponíveis no SQL também são válidas nas expressões PL/SQL:

- Funções de números em uma única linha
- Funções de caractere em uma única linha
- Funções de conversão de tipo de dados
- Funções de data
- GREATEST, LEAST
- Funções diversas

As funções a seguir não estão disponíveis nas instruções procedurais:

- DECODE
- Funções de grupo: AVG, MIN, MAX, COUNT, SUM, STDDEV e VARIANCE. As funções de grupo se aplicam a grupos de linhas em uma tabela e por isso estão disponíveis apenas em instruções SQL em um bloco PL/SQL

Exemplo

Compute o total de todos os números armazenados na tabela NUMBER_TABLE do PL/SQL. Esse exemplo produz um erro de compilação.

```
v_total      := SUM(number_table);
```

Funções PL/SQL

Exemplos

- Elaborar a lista de correspondência de uma empresa.

```
v_mailing_address:=      v_name||CHR(10)||  
                          v_address||CHR(10)||v_state||  
                          CHR(10)||v_zip;
```

- Converter o nome do funcionário para letra minúscula.

```
v_ename      := LOWER(v_ename);
```

Funções PL/SQL

O PL/SQL fornece muitas funções úteis que ajudam a manipular dados. Essas funções embutidas caem na categoria a seguir:

- Relatório de erro
- Número
- Caractere
- Conversão
- Data
- Diversos

Os exemplos de função no slide são definidos como se segue:

- Elaborar a lista de correspondência de uma empresa.
- Converter o nome para letra minúscula.

CHR é a função SQL que converte um código ASCII em seu caracter correspondente; 10 é o código para uma alimentação de linha.

Para obter mais informações, consulte PL/SQL User's Guide and Reference, Release 8, "Fundamentals".

Conversão de Tipo de Dados

Conversão de Tipo de Dados

- Converter dados em tipos de dados comparáveis.
- Os tipos de dados mistos poderão resultar em um erro e afetar o desempenho.
- Funções de conversão:
 - TO_CHAR
 - TO_DATE
 - TO_NUMBER

```
DECLARE
    v_date VARCHAR2(15);
BEGIN
    SELECT TO_CHAR(hiredate,
        'MON. DD, YYYY')
    INTO    v_date
    FROM    emp
    WHERE   empno = 7839;
END;
```

ORACLE®

Conversão de Tipo de Dados

O PL/SQL tenta converter os tipos de dados dinamicamente se estiverem misturados em uma instrução. Por exemplo, se você atribuir um valor NUMBER a uma variável CHAR, o PL/SQL traduzirá dinamicamente o número em uma representação de caracteres para que ele possa ser armazenado na variável CHAR. A situação inversa também se aplica, contanto que a expressão do caractere represente um valor numérico.

Desde que eles sejam compatíveis, você também poderá atribuir caracteres às variáveis DATE e vice-versa.

Em uma expressão, você deve certificar-se que os tipos de dados sejam os mesmos. Se ocorrerem tipos de dados mistos em uma expressão, você deve usar a função de conversão apropriada para converter os dados.

Sintaxe

```
TO_CHAR (valor, fmt)
TO_DATE (valor, fmt)
TO_NUMBER (valor, fmt)
```


onde: **valor** é uma string de caractere, número ou data
fmt é o modelo de formato usado para converter o
valor

Conversão de Tipo de Dados

Essa instrução produz um erro de compilação, se a variável `v_date` for declarada como tipo de dados `DATE`.

```
v_date := 'January 13, 1998';
```

Para corrigir o erro, use a função de conversão `TO_DATE`.

```
v_date := TO_DATE ('January 13, 1998',  
'Month DD, YYYY');
```

Conversão de Tipo de Dados

Os exemplos de conversão no slide são definidos como se seguem:

- Armazenar uma string de caractere representando uma data em uma variável declarada de tipo de dados `DATE`. Esse código causa um erro de sintaxe.
- Para corrigi-lo, converta a string em uma data com a função de conversão `TO_DATE`.

O PL/SQL tenta a conversão, se possível, mas o êxito depende das operações que estão sendo executadas. A execução explícita de conversões de tipo de dados é um bom exercício de programação, porque elas podem afetar de maneira favorável o desempenho e continuarem válidas mesmo com uma alteração nas versões de software.

Blocos Aninhados e Escopo de Variável

- As instruções podem ser aninhadas onde quer que uma instrução executável seja permitida.
- Um bloco aninhado torna-se uma instrução.
- Uma seção de exceção pode conter blocos aninhados.
- O escopo de um objeto é a região do programa que pode referir-se ao objeto.

Blocos Aninhados

Uma das vantagens que o PL/SQL tem sobre o código é a habilidade de aninhar instruções. Você poderá aninhar blocos onde quer que uma instrução executável seja permitida, transformando o bloco aninhado em uma instrução. Por isso, você poderá dividir a parte executável de um bloco em blocos menores. A seção de exceção poderá também conter blocos aninhados.

Escopo da Variável

O escopo de um objeto é a região do programa que pode referir-se ao objeto. Você poderá referenciar a variável declarada na seção executável.

Blocos Aninhados e Escopo de Variável

Um identificador é visível nas regiões nas quais você poderá referenciar o identificador não qualificado:

- Um bloco poderá procurar pelo bloco delimitador acima.
- Um bloco não poderá procurar pelo bloco delimitado abaixo.

Identificadores

Um identificador é visível no bloco no qual ele é declarado em todos os procedimentos, funções e sub-blocos aninhados. Se o bloco não localizar o identificador declarado localmente, ele procura acima pela seção declarativa dos blocos delimitadores (ou pais). O bloco nunca procura abaixo pelos blocos delimitados (ou filhos) ou transversalmente por blocos irmãos.

O escopo se aplica a todos os objetos declarados, incluindo variáveis, cursores, exceções definidas pelo usuário e constantes.

Observação: Qualifique um identificador usando o prefixo do label do bloco.

Para obter mais informações sobre os rótulos do bloco, consulte o PL/SQL User's Guide and Reference, Release 8, "Fundamentals".

Blocos Aninhados e Escopo de Variável

Exemplo

```
...  
x      BINARY_INTEGER;  
BEGIN Escopo de x  
... DECLARE  
y      NUMBER;  
BEGIN Escopo de y  
... END;  
... END;
```

Blocos Aninhados e Escopo de Variável

No bloco aninhado mostrado no slide, a variável nomeada de y poderá referenciar a variável nomeada de x. A variável x, entretanto, não poderá referenciar a variável y. Se a variável nomeada de y no bloco aninhado tiver o mesmo nome que a variável nomeada de x no bloco exterior, o seu valor é válido apenas pela duração do bloco aninhado.

Escopo

O escopo de um identificador é aquela região de uma unidade de programa (bloco, subprograma ou pacote) no qual você pode referenciar o identificador.

Visibilidade

Um identificador é visível apenas nas regiões a partir das quais você pode referenciar o identificador usando um nome não qualificado.

Operadores em PL/SQL

Os mesmos do SQL

- Lógico
- Aritmético
- Concatenação
- Parênteses para controlar a ordem das operações
- Operador exponencial (**)

Ordem das Operações

As operações na expressão são executadas em uma determinada ordem, dependendo de suas precedências (prioridade). A tabela a seguir mostra a ordem padrão das operações de cima a baixo:

Operador	Operação
**, NOT	Exponenciação, negação, lógica
+, -	Identidade, negação
*, /	Multiplicação, divisão
+, -,	Adição, subtração, concatenação
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparação
AND	Conjunção
OR	Inclusão

Observação: Não é necessário usar parênteses com as expressões booleanas, mas isso facilita a leitura do texto.

Para obter mais informações sobre os operadores, consulte PL/SQL User's Guide and Reference, Release 8, "Fundamentals".

Operadores em PL/SQL

Exemplos

- Incrementar o contador para um loop.

```
v_count      := v_count + 1;
```

- Definir o valor de um sinalizador booleano.

```
v_equal      := (v_n1 = v_n2);
```

- Validar o número de um funcionário se ele contiver um valor.

```
v_valid      := (v_empno IS NOT NULL);
```

Operadores em PL/SQL

Ao trabalhar com nulos, evite alguns erros bastante comuns mantendo em mente as seguintes regras:

- As comparações que envolvem nulos sempre produzem NULL.
- A aplicação do operador lógico NOT em um nulo produz NULL.
- Em instruções de controle condicionais, se a condição produzir NULL, sua sequência associada de instruções não será executada.

Usando Variáveis de Ligação

Para referenciar uma variável de ligação no PL/SQL, você deverá criar um prefixo antes do nome usando dois-pontos (:).

Exemplo

```
VARIABLE      g_salary NUMBER
DECLARE
    v_sal emp.sal%TYPE;
BEGIN
    SELECT      sal
    INTO        v_sal
    FROM        emp
    WHERE       empno = 7369;
    :g_salary   := v_sal;
END;
/
```

Imprimindo Variáveis de Ligação

No SQL*Plus, você poderá exibir o valor da variável de ligação usando o comando PRINT.

```
SQL> PRINT g_salary
```

```
G_SALARY
-----
800
```

Diretrizes de Programação

Facilite a manutenção do código:

- Documentando o código com comentários
- Desenvolvendo uma convenção de maiúsculas e minúsculas para o código
- Desenvolvendo convenções de nomeação para os identificadores e outros objetos
- Melhorando a leitura por endentação

Diretrizes de Programação

Siga essas diretrizes de programação para produzir códigos claros e reduzir a manutenção ao desenvolver um bloco PL/SQL.

Convenções de Códigos

A tabela a seguir fornece as diretrizes para a criação de códigos em letra maiúscula ou letra minúscula para ajudá-lo a distinguir as palavras-chave de objetos nomeados.

Categoria	Convenção de Maiúscula/Minúscula	Exemplos
Instruções SQL	Letra maiúscula	SELECT, INSERT
Palavras-chave PL/SQL	Letra maiúscula	DECLARE, BEGIN, IF
Tipos de dados	Letra maiúscula	VARCHAR2, BOOLEAN
Identificadores e parâmetros	Letra minúscula	v_sal, emp_cursor, g_sal, p_empno
Tabelas e colunas de banco de dados	Letra minúscula	emp, orderdate, deptno

Convenções para Nomeação de Código

Evitar ambigüidade:

- Os nomes de variáveis locais e parâmetros formais têm precedência sobre os nomes das tabelas de banco de dados.
- Os nomes de colunas têm precedência sobre os nomes das variáveis locais.

Convenções para Nomeação de Código

A tabela a seguir mostra um conjunto de prefixos e sufixos para distinguir um identificador de outros identificadores, de um banco de dados e de outros objetos nomeados.

Identificador	Convenção de Nomeação	Exemplo
Variável	<i>v_name</i>	v_sal
Constante	<i>c_name</i>	c_company_name
Cursor	<i>name_cursor</i>	emp_cursor
Exceção	<i>e_name</i>	e_too_many
Tipo de tabela	<i>name_table_type</i>	amount_table_type
Tabela	<i>name_table</i>	order_total_table
Tipo de registro	<i>name_record_type</i>	emp_record_type
Registro	<i>name_record</i>	customer_record
Variável de substituição SQL*Plus (também referida como parametro de substituição)	<i>p_name</i>	p_sal
Variável global SQL*Plus (também referida como host ou variável de ligação)	<i>g_name</i>	g_year_sal

Endentando o Código

Endentando o Código

Para maior clareza, endente cada nível de código.

Exemplo

```
BEGIN
  IF x=0 THEN
    y:=1;
  END IF;
END;
```

```
DECLARE
  v_deptno      NUMBER(2);
  v_location    VARCHAR2(13);
BEGIN
  SELECT deptno,
         loc
  INTO   v_deptno,
         v_location
  FROM   dept
  WHERE  dname = 'SALES';
  ...
END;
```

ORACLE®

Endentando o Código

Para maior clareza e para melhor leitura, endente cada nível de código. Para mostrar a estrutura, você poderá dividir as linhas usando o retorno de carro e endentar as linhas usando espaços e tabs. Compare as instruções IF a seguir para obter maior facilidade de leitura:

```
IF x>y THEN v_max:=x;ELSE v_max:=y;END IF;
```

```
IF x > y THEN
v_max := x; ELSE
v_max := y; END IF;
```

Determinando o Escopo da Variável

Determinando o Escopo da Variável

Exercício de Classe

```
...  
DECLARE  
  V_SAL          NUMBER(7,2) := 60000;  
  V_COMM         NUMBER(7,2) := V_SAL * .20;  
  V_MESSAGE      VARCHAR2(255) := ' eligible for commission';  
BEGIN ...  
  
  DECLARE  
    V_SAL          NUMBER(7,2) := 50000;  
    V_COMM         NUMBER(7,2) := 0;  
    V_TOTAL_COMP   NUMBER(7,2) := V_SAL + V_COMM;  
  BEGIN ...  
    V_MESSAGE := 'CLERK not' || V_MESSAGE;  
  END;  
  
  V_MESSAGE := 'SALESMAN' || V_MESSAGE;  
END;
```

ORACLE®

Exercício de Classe

Avaliar o bloco PL/SQL no slide. Determinar cada um dos valores a seguir de acordo com as regras de escopos:

1. O valor de V_MESSAGE no sub-bloco.
2. O valor de V_TOTAL_COMP no bloco principal.
3. O valor de V_COMM no sub-bloco.
4. O valor de V_COMM no bloco principal.
5. O valor de V_MESSAGE no bloco principal.

Sumário

Sumário

- **Estrutura de bloco PL/SQL: Regras de aninhamento de blocos e criação de escopos**
- **Programação em PL/SQL:**
 - Funções
 - Conversão de tipo de dados
 - Operadores
 - Variáveis de ligação
 - Convenções e diretrizes

```
DECLARE
  ...
BEGIN
  ...
EXCEPTION
  ...
END;
```

ORACLE®

Sumário

Um bloco poderá ter qualquer número de blocos aninhados definidos na parte executável. Os blocos definidos em um bloco são chamados de sub-blocos. Você poderá aninhar blocos apenas em partes executáveis de um bloco.

O PL/SQL fornece muitas funções úteis que ajudam a manipular dados. As funções de conversão convertem um valor de um tipo de dados para outro. O PL/SQL fornece muitas funções úteis que ajudam a manipular dados. As funções de conversão convertem um valor de um tipo de dados para outro. Geralmente, os formatos dos nomes de função seguem a convenção tipo de dados TO tipo de dados. O primeiro tipo de dados é o de entrada. O segundo tipo de dados é o de saída.

Os operadores de comparação comparam uma expressão com outra. O resultado é sempre verdadeiro falso ou nulo. Tipicamente, você deve usar operadores de comparação em instruções de controle condicional e na cláusula WHERE das instruções de manipulação de dados SQL. Os operadores relacionais permitem que você compare expressões complexas arbitrariamente.

As variáveis declaradas no SQL*Plus são chamadas de variáveis de ligação. Para referenciar essas variáveis em programas PL/SQL, elas devem ser precedidas por dois-pontos.

Master Training

Capítulo 03

Interagindo com o Oracle Server

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Criar uma instrução SELECT em PL/SQL
- Declarar o tipo de dados e tamanho de uma variável PL/SQL dinamicamente
- Criar instruções SELECT em PL/SQL
- Controlar transações em PL/SQL
- Determinar o resultado das instruções DML SQL

Objetivo da Lição

Nesta lição, você aprenderá a incorporar instruções INSERT, UPDATE, DELETE e SELECT SQL padrões nos blocos PL/SQL. Você também aprenderá como controlar transações e determinar o resultado das instruções DML SQL em PL/SQL.

Instruções SQL em PL/SQL

- Extrair uma linha de dados de um banco de dados usando o comando SELECT.
Apenas um único conjunto de valores pode ser retornado.
- Fazer alterações nas linhas no banco de dados usando os comandos DML.
- Controlar uma transação com o comando COMMIT, ROLLBACK ou SAVEPOINT.
- Determinar o resultado do DML com cursores implícitos.

Visão Geral

Quando você precisar extrair informações ou aplicar alterações aos bancos de dados, você deverá usar o SQL. O PL/SQL suporta integralmente a linguagem de manipulação de dados e os comandos de controle de transação no SQL. Você poderá usar as instruções SELECT para preencher as variáveis com os valores consultados em uma linha na tabela. Seus comandos DML (manipulação de dados) podem processar várias linhas.

Comparando os Tipos de Instrução SQL e PL/SQL

- Um bloco PL/SQL não é uma unidade de transação. Os comandos COMMIT, SAVEPOINT e ROLLBACK são independentes dos blocos, mas você pode emitir esses comandos em um bloco.
- O PL/SQL não suporta instruções em DDL (Data Definition Language) como, por exemplo, CREATE TABLE, ALTER TABLE ou DROP TABLE.
- O PL/SQL não suporta instruções em DCL (Data Control Language) como, por exemplo, GRANT ou REVOKE.

Para obter mais informações sobre o pacote DBMS_SQL, consulte o Oracle8 Server Application Developer's Guide, Release 8.

Instruções SELECT em PL/SQL

Recuperar dados do banco de dados com
SELECT. Sintaxe

```
SELECT      select_list
INTO        {variable_name[, variable_name]...
            | record_name}
FROM        tabela
WHERE       condição;
```

Recuperando Dados em PL/SQL

Use a instrução SELECT para recuperar dados no banco de dados.

Na sintaxe:

- select_list*** é uma lista de pelo menos uma coluna e pode incluir funções de linhas, de grupo ou expressões SQL
- variable_name*** é a variável escalar para armazenar o valor recuperado
- record_name*** é o registro PL/SQL para armazenar os valores recuperados
- tabela*** especifica o nome da tabela do banco de dados
- condição*** é composta de nomes de coluna, expressões, constantes e operadores de comparação, incluindo as variáveis PL/SQL e operadores

Aproveite a faixa completa de sintaxe do Oracle Server para a instrução SELECT. Lembre-se que as variáveis de host devem ter dois-pontos como prefixo.

Instruções SELECT em PL/SQL

A cláusula INTO é necessária. Exemplo

```
DECLARE
    v_deptno    NUMBER(2);
    v_loc        VARCHAR2(15);

BEGIN
    SELECT      deptno, loc
    INTO        v_deptno, v_loc
    FROM        dept
    WHERE       dname = 'SALES';

    ...
END;
```

Cláusula INTO

A cláusula INTO é mandatória e ocorre entre as cláusulas SELECT e FROM. Ela é usada para especificar os nomes das variáveis que armazenarão os valores que o SQL retorna a partir da cláusula SELECT. Você deve oferecer uma variável para cada item selecionado e a ordem delas deve corresponder aos itens selecionados.

Você deve usar a cláusula INTO para preencher as variáveis PL/SQL ou as variáveis de host.

As Consultas Devem Retornar Apenas Uma Linha

As instruções SELECT em um bloco PL/SQL caem na classificação ANSI de SQL Embutido (Embedded SQL), para a qual se aplicam as regras a seguir: as consultas devem retornar apenas uma linha. Mais de uma ou nenhuma linha geram mensagens de erro.

O PL/SQL lida com essas mensagens de erro destacando as exceções padrão, as quais você poderá capturar na seção de exceções do bloco com as exceções NO_DATA_FOUND e TOO_MANY_ROWS (o tratamento de exceções é abordado em uma lição posterior). Você deverá codificar as instruções SELECT para retornar uma única linha.

Recuperando Dados em PL/SQL

Recuperar a data da ordem de compra e de entrega para a ordem especificada.

Exemplo

```
DECLARE
    v_orderdate ord.orderdate%TYPE;
    v_shipdate  ord.shipdate%TYPE;
BEGIN
    SELECT      orderdate, shipdate
    INTO  v_orderdate, v_shipdate
    FROM    ord
    WHERE id = 620;
    ...
END;
```

Diretrizes

Siga essas diretrizes para recuperar os dados em PL/SQL:

- Finalize cada instrução SQL com um ponto-e-vírgula (;).
- A cláusula INTO é obrigatória para a instrução SELECT quando ela está embutida no PL/SQL.
- A cláusula WHERE é opcional e poderá ser usada para especificar variáveis de entrada, constantes, literais ou expressões PL/SQL.
- Especifique o mesmo número de variáveis de saída na cláusula INTO como colunas de banco de dados na cláusula SELECT. Certifique-se de que elas correspondam em posição e que os seus tipos de dados sejam compatíveis.

Recuperando Dados em PL/SQL

Recuperando Dados em PL/SQL

Retornar o total de salários de todos os funcionários no departamento especificado.

Exemplo

```
DECLARE
  v_sum_sal    emp.sal%TYPE;
  v_deptno    NUMBER NOT NULL := 10;
BEGIN
  SELECT      SUM(sal) -- group function
  INTO        v_sum_sal
  FROM        emp
  WHERE       deptno = v_deptno;
END;
```

ORACLE®

Diretrizes (continuação)

- Para garantir que os tipos de dados dos identificadores correspondam aos tipos de dados das colunas, use o atributo %TYPE. O tipo de dados e o número de variáveis na cláusula INTO correspondem àqueles na lista SELECT.

- Use as funções de grupo como, por exemplo, SUM, em uma instrução SQL, porque as funções de grupo se aplicam a grupos de linhas em uma tabela.

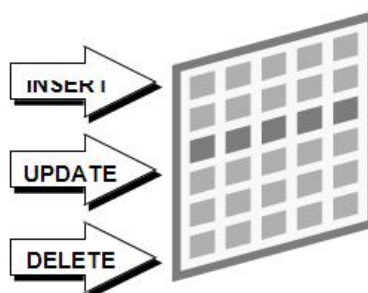
Observação: As funções de grupo não poderão ser usadas na sintaxe do PL/SQL. Elas são usadas em instruções SQL em um bloco PL/SQL.

Manipulando Dados Usando o PL/SQL

Manipulando Dados Usando o PL/SQL

Alterar as tabelas de banco de dados usando os comandos DML:

- INSERT
- UPDATE
- DELETE



ORACLE®

Manipulando Dados Usando o PL/SQL

Você poderá manipular os dados no banco de dados usando os comandos DML (manipulação de dados). Você poderá emitir os comandos DML, por exemplo, INSERT, UPDATE e DELETE sem restrições no PL/SQL. A inclusão das instruções COMMIT ou ROLLBACK no código PL/SQL libera os bloqueios de linha (e bloqueios de tabela).

- A instrução INSERT adiciona novas linhas de dados à tabela.
- A instrução UPDATE modifica linhas existentes na tabela.
- A instrução DELETE remove linhas não desejadas da tabela.

Inserindo Dados

Adicionar informações sobre novos funcionários na tabela EMP.

Exemplo

```
BEGIN
  INSERT INTO emp(empno, ename, job, deptno)
  VALUES      (empno_sequence.NEXTVAL, 'HARDING',
    'CLERK', 10);
END;
```

Inserindo Dados

- Use as funções SQL como, por exemplo, USER e SYSDATE.
- Gere valores de chaves primárias usando as seqüências de banco de dados.
- Crie valores no bloco PL/SQL.
- Adicione valores default de coluna.

Observação: Não há possibilidade de ambigüidade entre os identificadores e nomes de coluna na instrução INSERT. Qualquer identificador na cláusula INSERT deve ser um nome de coluna do banco de dados.

Atualizando Dados

Aumentar o salário de todos os funcionários na tabela EMP e que sejam Analistas (Analysts).

Exemplo

```
DECLARE
  v_sal_increase      emp.sal%TYPE := 2000;
BEGIN
  UPDATE      emp
  SET      sal = sal + v_sal_increase
  WHERE job = 'ANALYST';
END;
```

Atualizando e Deletando Dados

Poderá haver ambigüidade na cláusula SET da instrução UPDATE porque, embora o identificador à esquerda do operador de atribuição seja sempre uma coluna de banco de dados, o identificador à

direita podem ser uma coluna de banco de dados ou uma variável PL/SQL.

Lembre-se de que a cláusula WHERE é usada para determinar que linhas são afetadas. Se nenhuma linha for modificada, não ocorrerão erros, diferente da instrução SELECT no PL/SQL.

Observação: As atribuições de variável PL/SQL sempre usam :=, e as atribuições de coluna SQL sempre usam =. Lembre-se de que os nomes de coluna e de identificadores são idênticos na cláusula WHERE, o Oracle Server procura primeiro pelo nome no banco de dados.

Deletando Dados

Deletar linhas que pertençam ao departamento 10 da tabela EMP. Exemplo

```
DECLARE
    v_deptno    emp.deptno%TYPE := 10;
BEGIN
    DELETE FROM emp
    WHERE deptno =    v_deptno;
END;
```

Deletando Dados

Deletar uma ordem de compra especificada.

```
DECLARE
    v_ordid     ord.ordid%TYPE := 605;
BEGIN
    DELETE FROM item
    WHERE ordid = v_ordid;
END;
```

Convenções para Nomeação

- Usar uma convenção de nomeação para evitar ambigüidades na cláusula WHERE.

- Os identificadores e as colunas de banco de dados devem ter nomes diferentes.

- Podem surgir erros de sintaxe, pois o PL/SQL verifica primeiro a coluna no banco de dados.

Convenções para Nomeação

Evite a ambigüidade na cláusula WHERE aderindo a uma convenção de nomeação que distingue os nomes de coluna do banco de dados dos nomes de variável PL/SQL.

- Os identificadores e as colunas de banco de dados devem ter nomes diferentes.

- Podem surgir erros de sintaxe, pois o PL/SQL verifica primeiro a coluna na tabela.

Convenções para Nomeação

```
DECLARE
    orderdate    ord.orderdate%TYPE;
    shipdate     ord.shipdate%TYPE;
    ordid ord.ordid%TYPE := 601;
```

```
BEGIN
    SELECT    orderdate, shipdate
    INTO      orderdate, shipdate
    FROM      ord
    WHERE     ordid =      ordid;
```

```
END;
```

```
SQL> /
```

```
DECLARE
```

```
*
```

```
ERRO na linha 1(ERROR at line 1:)
```

```
ORA-01422: busca exata retornou mais linhas do que as
solicitadas (exact fetch returns more than requested number
of rows)
```

```
ORA-06512: na linha 6 (at line 6)
```

Convenções para Nomeação (continuação)

O exemplo mostrado no slide está definido como se segue:
Recuperar as datas de ordem de compra e
de entrega na tabela ord para o número de ordem 601. Isso
gera uma exceção de tempo de execução não tratável.

O PL/SQL verifica se um identificador é uma coluna no banco de dados; se não, supõe-se que seja um identificador PL/SQL.

Observação: Não há possibilidade de ambigüidade na cláusula SELECT, pois qualquer identificador nesse cláusula deve ser um nome de coluna do banco de dados. Não há possibilidade de ambigüidade na cláusula INTO, pois os identificadores nessa cláusula devem ser variáveis PL/SQL. Apenas na cláusula WHERE há essa possibilidade.

Mais informações sobre TOO_MANY_ROWS e outras exceções são abordadas em lições subseqüentes.

Instruções COMMIT e ROLLBACK

- Iniciar uma transação com o primeiro comando DML para seguir uma instrução COMMIT ou ROLLBACK.
- Usar instruções SQL como, por exemplo, COMMIT e ROLLBACK, para finalizar uma transação explicitamente.

Controlando Transações

Você deverá controlar a lógica das transações com as instruções COMMIT e ROLLBACK SQL, tornando permanentes as alterações em alguns grupos de bancos de dados ao descartar outros. Assim como ocorre com o Oracle Server, as transações DML são iniciadas no primeiro comando seguindo uma instrução

COMMIT ou ROLLBACK e são finalizadas na próxima instrução COMMIT ou ROLLBACK correta.

Essas ações podem ocorrer em um bloco PL/SQL ou como resultado dos eventos no ambiente do host (por exemplo, o encerramento de uma sessão SQL*Plus automaticamente compromete a transação pendente).

Para marcar um ponto intermediário no processo de transação, use SAVEPOINT.

Sintaxe

```
COMMIT [WORK];
```

```
SAVEPOINT savepoint_name;
```

```
ROLLBACK [WORK];
```

```
ROLLBACK [WORK] TO [SAVEPOINT] savepoint_name;
```

onde: WORK é para a adequação aos padrões ANSI

Observação: Os comandos de controle de transação são todos válidos no PL/SQL, embora o ambiente do host possa impor algumas restrições ao usuário.

Você também poderá incluir comandos explícitos de bloqueios (como, por exemplo, LOCK TABLE e SELECT ... FOR UPDATE) em um bloco (uma lição subsequente abordará mais informações sobre o comando FOR UPDATE). Eles estarão ativos até o final da transação. Além disso, um bloco PL/SQL não implica necessariamente uma transação.

Cursor SQL

- Um cursor é uma área de trabalho SQL particular.
- Há dois tipos de cursores:
 - Cursores implícitos
 - Cursores explícitos
- O Oracle Server usa cursores implícitos para analisar e executar as instruções SQL.
- Os cursores explícitos são declarados especificamente pelo programador.

Cursor SQL

Sempre que você emitir uma instrução SQL, o Oracle Server abrirá uma área de memória na qual o comando é analisado e executado. Essa área é chamada de cursor.

Quando a parte executável de um bloco emite uma instrução SQL, o PL/SQL cria um cursor implícito, o qual tem o identificador SQL. O PL/SQL gerencia esse cursor automaticamente. O programador declara e nomeia um cursor explícito. Há quatro atributos disponíveis no PL/SQL que poderão aplicar-se aos cursores.

Observação: Mais informações sobre os cursores explícitos são abordadas em uma lição subsequente.

Para obter mais informações, consulte PL/SQL User's Guide and Reference, Release 8, "Interaction with Oracle".

Atributos do Cursor SQL

Ao usar os atributos do cursor SQL, você poderá testar os resultados das instruções SQL.

SQL%ROWCOUNT	Número de linhas afetadas pela instrução SQL mais recente (um valor inteiro)
SQL%FOUND	Atributo booleano avaliado para TRUE se a instrução SQL mais recente afetar uma ou mais linhas
SQL%NOTFOUND	Atributo booleano avaliado para TRUE se a instrução SQL mais recente não afetar uma ou mais linhas
SQL%ISOPEN	Sempre é avaliado para FALSE porque o PL/SQL fecha os cursores implícitos imediatamente após a execução

Atributos do Cursor SQL

Os atributos do cursor SQL permitem que você avalie o que aconteceu quando o cursor implícito foi usado pela última vez. Você deve usar esses atributos nas instruções PL/SQL como, por exemplo, as funções. Você não poderá usá-las em instruções SQL.

Você poderá usar os atributos SQL%ROWCOUNT, SQL%FOUND, SQL%NOTFOUND e SQL%ISOPEN na seção de exceção de um bloco para coletar informações sobre a execução de uma instrução de manipulação de dados. Ao contrário da instrução

SELECT, que retorna uma exceção, o PL/SQL não considera uma instrução DML que não afete linhas onde ocorreram falhas.

Atributos do Cursor SQL

Deletar linhas que especificaram um número de ordem de compra a partir da tabela ITEM. Imprimir o número de linhas deletadas.

Exemplo

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
    v_ordid      NUMBER := 605;
BEGIN
    :rows_deleted := (SQL%ROWCOUNT ||
        ' rows deleted. ');
END;
/
PRINT rows_deleted
```

Atributos do Cursor SQL (continuação)

O exemplo mostrado no slide está definido como se segue: Deletar as linhas de uma tabela de ordem de compra para a ordem número 605. Ao usar o atributo SQL%ROWCOUNT, você poderá imprimir o número de linhas deletadas.

Sumário

- Embutir o SQL no bloco PL/SQL: SELECT, INSERT, UPDATE, DELETE
- Embutir as instruções de controle de transação em um bloco PL/SQL:

COMMIT, ROLLBACK, SAVEPOINT

Sumário

Os comandos DML, por exemplo, INSERT, UPDATE e DELETE, poderão ser usados nos programas PL/SQL sem restrições. A instrução COMMIT finaliza a transação atual e torna as alterações feitas durante essa transação permanente. A instrução ROLLBACK finaliza a transação atual e desfaz quaisquer alterações feitas durante essa transação. SAVEPOINT nomeia e marca o ponto atual no processamento de uma transação. Usados com a instrução ROLLBACK TO, as instruções SAVEPOINT permitem que você desfaga partes de uma transação, em vez da transação inteira.

Sumário

- Há dois tipos de cursor: implícito e explícito.
- Os atributos de cursor implícitos verificam o resultado das instruções DML:
 - SQL%ROWCOUNT
 - SQL%FOUND
 - SQL%NOTFOUND
 - SQL%ISOPEN
- Os cursores explícitos são definidos pelo programador.

Sumário (continuação)

Um cursor implícito é declarado pelo PL/SQL para cada instrução de manipulação de dados SQL.

O PL/SQL fornece quatro atributos para cada cursor. Esses atributos fornecem informações úteis sobre as operações que são executadas com cursores. Os cursores explícitos são definidos pelo programador.

Master Training

Capítulo 04

Criando Estruturas para Controle

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Identificar os usos e tipos de estruturas para controle
- Construir uma instrução IF
- Construir e identificar diferentes instruções loop
- Usar tabelas lógicas
- Controlar fluxo de blocos usando loops e labels aninhados

Objetivo da Lição

Nesta lição, você aprenderá sobre o controle condicional dentro do bloco PL/SQL usando loops e instruções IF.

Controlando o Fluxo de Execução PL/SQL

Controlando o Fluxo de Execução PL/SQL

Pode-se alterar o fluxo lógico de instruções usando estruturas para controle de loop e instruções IF condicionais.

Instruções IF condicionais:

- IF-THEN-END IF
- IF-THEN-ELSE-END IF
- IF-THEN-ELSIF-END IF



ORACLE®

Você pode alterar o fluxo lógico de instruções dentro do bloco PL/SQL com diversas estruturas para controle. Esta lição aborda os dois tipos de estruturas para controle do PL/SQL: construções condicionais com a instrução IF e estruturas para controle LOOP (abordadas posteriormente nesta lição). Existem três formatos de instruções IF:

- IF-THEN-END IF
- IF-THEN-ELSE-END IF
- IF-THEN-ELSIF-END IF

Instruções IF

Sintaxe

```
IF condição THEN
    instruções;
[ELSIF condição THEN
    instruções;]
[ELSE
    instruções;]
END IF;
```

Instrução IF Simples:

Definir o ID do gerente como 22 se o nome do funcionário for Osborne.

```
IF v_ename = 'OSBORNE' THEN
    v_mgr := 22;
END IF;
```

Instruções IF

A estrutura da instrução IF do PL/SQL é semelhante à estrutura das instruções IF em outras linguagens procedurais. Ela permite que o PL/SQL execute ações de modo seletivo com base em condições.

Na sintaxe:

condição é uma expressão ou variável Booleana (TRUE, FALSE ou NULL) (Ela está associada a uma seqüência de instruções, que será executada somente se a expressão produzir TRUE.)

THEN é uma cláusula que associa a expressão Booleana que a precede com a seqüência de instruções posterior instruções pode ser uma ou mais instruções SQL ou PL/SQL. (Elas podem incluir mais instruções IF contendo diversos IFs, ELSEs e ELSIFs aninhados.)

ELSIF é uma palavra-chave que introduz uma expressão Booleana. (Se a primeira condição produzir FALSE ou NULL, a palavra-chave ELSIF introduzirá condições adicionais.)

ELSE é uma palavra-chave que se for atingida pelo controle, executará a seqüência de instruções que segue a palavra-chave

Instruções IF Simples

Definir o título da tarefa como Salesman, o número do departamento como 35 e a comissão como 20% do salário atual se o sobrenome for Miller.

Exemplo

```
. . .  
IF      v_ename          = 'MILLER' THEN  
    v_job := 'SALESMAN';  
    v_deptno := 35;  
    v_new_comm := sal * 0.20;  
END IF;  
. . .
```

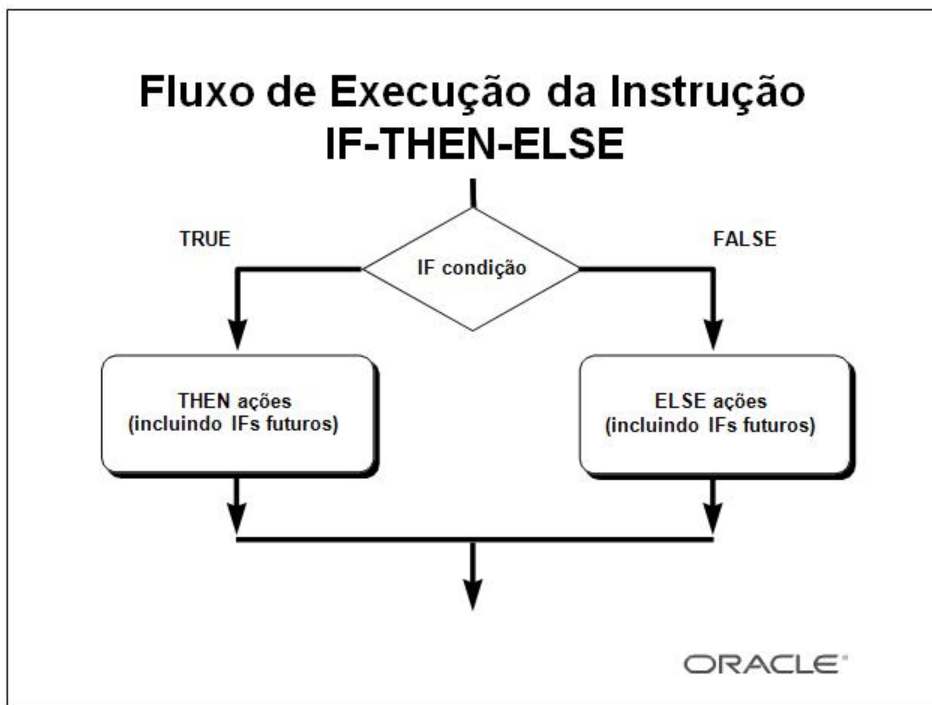
Instruções IF Simples

No exemplo do slide, o PL/SQL executará essas três ações (definindo as variáveis v_job, v_deptno e v_new_comm) somente se a condição for TRUE. Se a condição for FALSE ou NULL, o PL/SQL as ignorará. Em ambos os casos, o controle será reiniciado na próxima instrução do programa após END IF.

Diretrizes

- Você pode executar ações de maneira seletiva com base nas condições atendidas.
- Ao criar um código, lembre-se da grafia das palavras-chave:
 - ELSIF é uma palavra.
 - END IF são duas palavras.
- Se a condição Booleana para controle for TRUE, a seqüência de instruções associada será executada; se ela for FALSE ou NULL, a seqüência de instruções associadas será ignorada.
 - É permitido qualquer quantidade de cláusulas ELSIF.
- Pode haver no máximo uma cláusula ELSE.
- Endente instruções executadas condicionalmente para clareza.

Fluxo de Execução da Instrução IF-THEN-ELSE



Fluxo de Execução da Instrução IF-THEN-ELSE

Se a condição for FALSE ou NULL, você poderá usar a cláusula ELSE para executar outras ações. Assim como com a instrução IF simples, o controle reiniciará no programa a partir de END IF. Por exemplo:

```
IF condição1 THEN
    instrução1;
ELSE
    instrução2;
END IF;
```

Instruções IF Aninhadas

Os dois conjuntos de ações do resultado da primeira instrução IF podem incluir mais instruções IF antes que as ações específicas sejam executadas. As cláusulas THEN e ELSE podem incluir instruções IF.

Cada instrução IF aninhada deve ser terminada com um END IF correspondente.

```
IF condição1 THEN
```

```
        instrução1; ELSE  
    IF condição2 THEN  
        instrução2; END IF;  
END IF;
```

Instruções IF-THEN-ELSE

Definir um indicador para pedidos quando houver menos de cinco dias entre a data do pedido e a data da entrega.

Exemplo

```
...  
IF v_shipdate - v_orderdate < 5 THEN  
    v_ship_flag := 'Aceitável';  
ELSE  
    v_ship_flag := 'Inaceitável';  
END IF;  
...
```

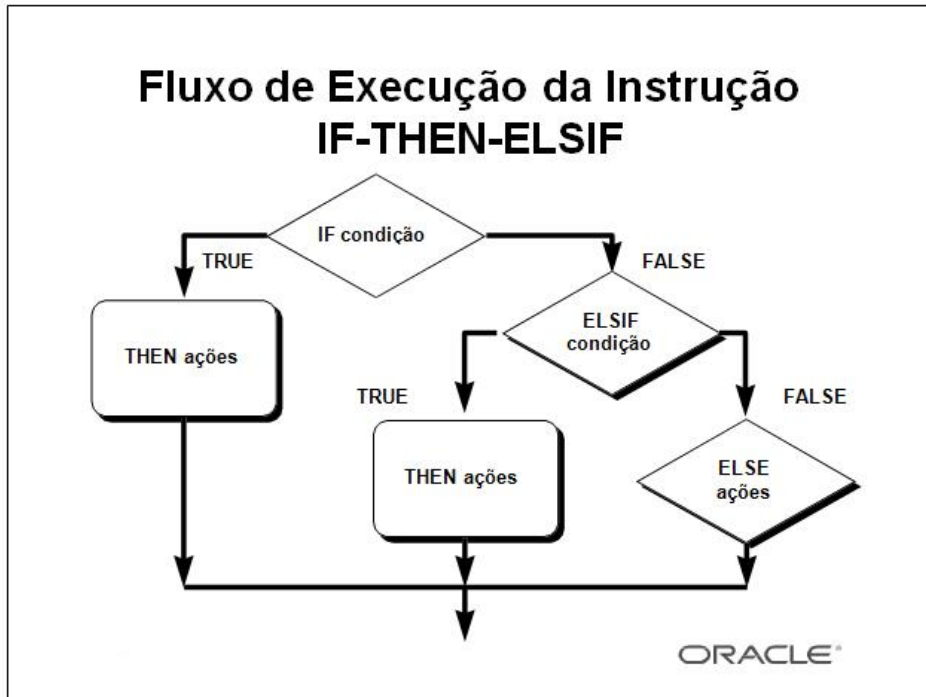
Exemplo

Defina a tarefa para Manager se o nome do funcionário for King. Se o nome do funcionário for diferente de King, defina a tarefa para Clerk.

```
IF v_ename = 'KING' THEN  
    v_job := 'MANAGER';  
ELSE  
    v_job := 'CLERK';  
END IF;
```

Fluxo de Execução da Instrução

IF-THEN-ELSIF



Fluxo de Execução da Instrução IF-THEN-ELSIF

Determine o bônus de um funcionário com base em seu departamento.

```
...  
IF v_deptno = 10 THEN  
    v_comm := 5000;  
ELSIF v_deptno = 20 THEN  
    v_comm := 7500;  
ELSE  
    v_comm := 2000;  
END IF;  
...
```

No exemplo, a variável `v_comm` será usada para atualizar a coluna `COMM` na tabela `EMP` e `v_deptno` representa o número do departamento de um funcionário.

Instruções IF-THEN-ELSIF

Para um determinado valor, calcular um percentual desse valor com base em uma condição.

Exemplo

```
. . .  
IF      v_start > 100 THEN  
    v_start := 2 * v_start;  
ELSIF v_start >= 50 THEN  
    v_start := .5 * v_start;  
ELSE  
    v_start := .1 * v_start;  
END IF;  
. . .
```

Instruções IF-THEN-ELSIF

Quando possível, use a cláusula ELSIF em vez de aninhar instruções IF. O código fica mais fácil de ler e entender e a lógica é identificada claramente. Se a ação na cláusula ELSE consistir puramente de outra instrução IF, será mais conveniente usar a cláusula ELSIF. Isso torna o código mais claro pois elimina a necessidade de END IFs aninhadas ao final de cada conjunto de condições e ações futuras.

Exemplo

```
IF condição1 THEN  
    instrução1;  
ELSIF condição2 THEN  
    instrução2;  
ELSIF condição3 THEN  
    instrução3;  
END IF;
```

A instrução IF-THEN-ELSIF de exemplo acima é definida ainda mais da seguinte forma:

Para um determinado valor, calcule um percentual do valor original. Se o valor for superior a 100, o valor calculado será o dobro do valor inicial. Se o valor estiver entre 50 e 100, o valor calculado será 50% do valor inicial. Se o valor informado for menor que 50, o valor calculado será 10% do valor inicial.

Observação: Qualquer expressão aritmética contendo valores nulos será avaliada como nula.

Elaborando Condições Lógicas

- Você pode manipular os valores nulos com o operador IS NULL.
- Qualquer expressão aritmética contendo um valor nulo será avaliada como NULL.
- Expressões concatenadas com valores nulos tratam valores nulos como uma string vazia.

Desenvolvendo Condições Lógicas

Você pode desenvolver uma condição Booleana simples combinando expressões de data, números ou caracteres com um operador de comparação. Normalmente, manipule valores nulos com o operador IS NULL.

Null em Expressões e Comparações

- A condição IS NULL será avaliada como TRUE somente se a variável que ela estiver verificando for NULL.
- Qualquer expressão contendo um valor nulo é avaliada como NULL, com exceção de uma expressão concatenada, que trata os valores nulos como uma string vazia.

Exemplos

Essas duas expressões serão avaliadas como NULL se v_sal for NULL.

```
v_sal > 1000
```

```
v_sal * 1.1
```

No exemplo a seguir, a string não será avaliada como NULL se v_string for NULL.

```
'PL' || v_string || 'SQL'
```

Tabelas Lógicas

Tabelas Lógicas

Desenvolver uma condição Booleana simples com um operador de comparação.

AND	TRUE	FALSE	NULL	OR	TRUE	FALSE	NULL	NOT	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL

ORACLE®

Condições Booleanas com Operadores Lógicos

Você pode criar uma condição Booleana complexa combinando condições Booleanas simples com os operadores lógicos AND, OR e NOT. Nas tabelas lógicas mostradas no slide, FALSE tem precedência sobre uma condição AND e TRUE tem precedência em uma condição OR. AND retornará TRUE somente se seus dois operandos forem TRUE. OR retornará FALSE somente se seus dois operandos forem FALSE. NULL AND TRUE sempre será avaliado como NULL porque não se sabe se o segundo operando será avaliado como TRUE ou não.

Observação: A negação de NULL (NOT NULL) resulta em um valor nulo porque valores nulos são indeterminados.

Condições Booleanas

Qual é o valor de V_FLAG em cada caso?

```
v_flag := v_reorder_flag AND v_available_flag;
```

V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
NULL	TRUE	NULL
NULL	FALSE	FALSE

Desenvolvendo Condições Lógicas

A tabela lógica de AND pode ajudar você a avaliar as possibilidades da condição Booleana no slide.

Controle Iterativo: Instruções LOOP

Controle Iterativo: Instruções LOOP

- Loops repetem uma instrução ou sequência de instruções várias vezes.
- Existem três tipos de loop:
 - Loop básico
 - Loop FOR
 - Loop WHILE



• Loops repetem uma instrução ou sequência de instruções várias vezes.

- Existem três tipos de loop:
 - Loop básico
 - Loop FOR
 - Loop WHILE

Controle Iterativo: Instruções LOOP

O PL/SQL oferece diversos recursos para estruturar loops para repetirem uma instrução ou sequência de instruções várias vezes.

As construções em loop são o segundo tipo de estrutura para controle:

- Loop básico para fornecer ações repetitivas sem condições gerais

- Loops FOR para fornecer controle iterativo para ações com base em uma contagem
- Loops WHILE para fornecer controle iterativo para ações com base em uma condição

- Instrução EXIT para terminar loops

Para obter mais informações, consulte PL/SQL User's Guide and Reference, Release 8, "Control Structures".

Observação: Outro tipo de loop FOR, loop FOR de cursor, será discutido em uma lição subsequente.

Loop Básico

Sintaxe

```
LOOP                                -- delimitador

    instrução1;                    -- instruções

    EXIT [WHEN condição];          -- instrução EXIT

END LOOP;                          -- delimitador
```

...

onde: condição é uma variável Booleana ou expressão
(TRUE, FALSE, ou NULL);

Loop Básico

O formato mais simples da instrução LOOP é o loop básico (ou infinito), que delimita uma seqüência de instruções entre as palavras-chave LOOP e END LOOP. Sempre que o fluxo de execução atinge a instrução END LOOP, o controle retorna à instrução LOOP correspondente acima. Um loop básico permite a execução de sua instrução pelo menos uma vez, mesmo que a condição já esteja atendida no momento em que o loop foi informado. Sem a instrução EXIT, o loop seria infinito.

A instrução EXIT

Você pode terminar um loop usando a instrução EXIT. O controle passa para a próxima instrução após a instrução END LOOP. Pode-se emitir EXIT como uma ação dentro de uma instrução IF ou

como uma instrução independente dentro do loop. A instrução EXIT deve ser colocada dentro de um loop. No segundo caso, você pode anexar uma cláusula WHEN para permitir a terminação condicional do loop. Quando a instrução EXIT é encontrada, a condição na cláusula WHEN é avaliada. Se a condição produzir TRUE, o loop finalizará e o controle passará para a próxima instrução após o loop. Um loop básico pode conter várias instruções EXIT..

Loop Básico

Exemplo

```
DECLARE
    v_ordid    item.ordid%TYPE :=    601;
    v_counter  NUMBER(2) := 1;
BEGIN
    LOOP
        INSERT INTO item(ordid, itemid)
        VALUES(v_ordid, v_counter);
        v_counter := v_counter + 1;
        EXIT WHEN v_counter > 10;
    END LOOP;
END;
```

Loop Básico (continuação)

O loop básico de exemplo mostrado no slide está definido como se segue: Inserir os 10 primeiros novos itens de linha para o número do pedido 601.

Observação: O loop básico permite a execução de sua instrução pelo menos uma vez, mesmo que a condição já esteja atendida ao entrar com o loop, contanto que a condição esteja colocada no loop de forma a ser verificada somente após essas instruções. Entretanto, se a condição exit for colocada no início do loop, antes de qualquer outra instrução executável, e ela for true, ocorrerá a saída do loop e as instruções jamais serão executadas.

Loop FOR

Sintaxe

```
FOR contador in [REVERSE]
    lower_bound..upper_bound LOOP
    instrução1;
    instrução2;
    . . .
END LOOP;
```

- Usar um loop FOR para desviar o teste para o número de iterações.
- Não declarar o contador; ele é declarado implicitamente.

Loop FOR

Os loops FOR têm a mesma estrutura geral do loop básico. Além disso, eles têm uma instrução para controle no início da palavra-chave LOOP para determinar o número de iterações que o PL/SQL executa.

Na sintaxe:

contador é um inteiro declarado implicitamente cujo valor aumenta ou diminui automaticamente (diminuirá se a palavra-chave REVERSE for usada) em 1 cada iteração do loop até o limite superior ou inferior a ser alcançado

REVERSE faz o contador decrescer a cada iteração a partir do limite superior até o limite inferior. (Note que o limite inferior ainda é referenciado primeiro.)

limite_inferior especifica o limite inferior da faixa de valores do contador

limite_superior especifica o limite superior da faixa de valores do contador

Não declare o contador, ele é declarado implicitamente como um inteiro.

Observação: A sequência de instruções é executada sempre que o contador é incrementado, conforme determinado pelos dois limites. Os limites superior e inferior da faixa do loop podem ser

literais, variáveis ou expressões, mas devem ser avaliados para inteiros. Se o limite inferior da faixa do loop for avaliado para um inteiro maior do que o limite superior, a sequência de instruções não será executada. Por exemplo, a instrução1 é executada somente uma vez:

```
FOR i IN 3..3 LOOP instrução1; END LOOP;
```

Loop FOR

Diretrizes

- Referenciar o contador dentro do loop somente; ele é indefinido fora do loop.
- Usar uma expressão para referenciar o valor existente de um contador.
- Não referenciar o contador como o destino de uma atribuição.

Loop FOR (continuação)

Observação: Os limites inferior e superior de uma instrução LOOP não precisam ser literais numéricas. Eles podem ser expressões que convertem para valores numéricos.

Exemplo

```
DECLARE
    v_lower    NUMBER := 1;
    v_upper    NUMBER := 100;
BEGIN
    FOR i IN v_lower..v_upper LOOP
        ...
    END LOOP;
END;
```

Loop FOR

Inserir os 10 primeiros novos itens de linha para o número do pedido 601.

Exemplo

```
DECLARE
    v_ordid      item.ordid%TYPE := 601;
BEGIN
    FOR i IN 1..10 LOOP
        INSERT INTO item(ordid, itemid)
            VALUES(v_ordid, i);
    END LOOP;
END;
```

Loop For

O exemplo mostrado no slide está definido como se segue: Inserir os 10 primeiros novos itens de linha para o número do pedido 601. Para isso, usa-se um loop FOR.

Loop WHILE

Loop WHILE

Sintaxe

```
WHILE condição LOOP  
    instrução1;  
    instrução2;  
    . . .  
END LOOP;
```

← A Condição é avaliada ao início de cada iteração.

Usar o loop WHILE para repetir instruções enquanto uma condição for TRUE.

ORACLE®

Loop WHILE

Você pode usar o loop WHILE para repetir uma sequência de instruções até a condição para controle não ser mais TRUE. A condição é avaliada ao início de cada iteração. O loop terminará quando a condição for FALSE. Se a condição for FALSE no início do loop, nenhuma iteração futura será executada.

Na sintaxe:

condição é uma expressão ou variável Booleana (TRUE, FALSE, ou NULL)

instrução pode ser uma ou mais instruções SQL ou PL/SQL

Se as variáveis envolvidas nas condições não se alterarem no curso do corpo do loop, a condição permanecerá TRUE e o loop não terminará.

Observação: Se a condição produzir NULL, o loop será ignorado e o controle passará para a próxima instrução.

Loop WHILE

Exemplo

```
ACCEPT p_new_order PROMPT 'Incluir o número do pedido: '
ACCEPT p_items -
      PROMPT 'Incluir o número de itens neste pedido: '
DECLARE
v_count      NUMBER(2) := 1;
BEGIN
      WHILE v_count <= &p_items LOOP
            INSERT INTO item (ordid, itemid)
            VALUES (&p_new_order, v_count);
            v_count := v_count + 1;
      END LOOP;
      COMMIT;
END;
/
```

Loop WHILE (continuação)

No exemplo do slide, os itens de linha estão sendo adicionados à tabela ITEM de um pedido especificado. O usuário é solicitado a fornecer o número do pedido (p_new_order) e o número de itens desse pedido (p_items). Com cada iteração através do loop WHILE, um contador (v_count) é incrementado. Se o número de iterações for menor ou igual ao número de itens do pedido, o código dentro do loop será executado e será inserida uma linha dentro da tabela ITEM. Quando o contador exceder o número de itens do pedido, a condição que controla o loop será avaliada como falsa e o loop terminará.

Loops e Labels Aninhados

- Aninhar loops para vários níveis.
- Usar labels para distinguir entre blocos e loops.
- Sair do loop externo com a instrução EXIT referenciando o label.

Loops e Labels Aninhados

Pode-se aninhar loops para vários níveis. Você pode aninhar loops básicos, FOR e WHILE um dentro

do outro. A terminação de um loop aninhado não terminará o loop delimitado a menos que seja criada uma exceção. Entretanto, você pode colocar labels em loops e sair do loop externo com a instrução EXIT.

Os nomes de label seguem as mesmas regras de outros identificadores. Um label é colocado antes de uma instrução, seja na mesma linha ou em uma linha separada. Coloque o label no loop colocando-o antes da palavra LOOP dentro dos delimitadores de label (<<label>>).

Se for atribuído um label ao loop, o nome do label poderá ser opcionalmente incluído após a instrução END LOOP para clareza.

Loops e Labels Aninhados

```
...
BEGIN
    <<Outer_loop>>
    LOOP
        v_counter := v_counter+1;
    EXIT WHEN v_counter>10;
        <<Inner_loop>> LOOP
            ...
        EXIT Outer_loop WHEN total_done = 'YES';
        -- Leave both loops
    EXIT WHEN inner_done = 'YES';
        -- Leave inner loop only
    ...
    END LOOP Inner_loop;
    ...
    END LOOP Outer_loop; END;
```

Loops e Labels Aninhados

No exemplo do slide, existem dois loops. O loop externo é identificado pelo label, <<Outer_Loop>> e o loop interno é identificado pelo label <<Inner_Loop>>. O loop interno está aninhado dentro do loop externo. Os nomes de label são incluídos após a instrução END LOOP para clareza.

Sumário

Alterar o fluxo lógico de instruções usando estruturas para controle.

- Condicional (instrução IF)
- Loops:
 - Loop básico
 - Loop FOR
 - Loop WHILE
 - Instrução EXIT

Sumário

Uma construção para controle condicional verifica a validade de uma condição e executa uma ação correspondente de acordo. Use a construção IF para uma execução condicional de instruções.

Uma construção para controle iterativo executa uma seqüência de instruções repetidamente, contanto que uma condição especificada se mantenha TRUE. Use as diversas construções de loop para executar operações iterativas.

Master Training

Capítulo 05

Trabalhando com Tipos de Dados Record e Collections

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Criar registros PL/SQL definidos pelo usuário
- Criar um registro com o atributo %ROWTYPE
- Criar uma tabela PL/SQL
- Criar uma tabela PL/SQL de registros
- Descrever a diferença entre registros, tabelas e tabelas de registros

Objetivo da Lição

Nesta lição, você aprenderá mais sobre os tipos de dados compostos e seus usos.

Tipos de Dados Compostos

- Tipos:
 - PL/SQL RECORDS
 - PL/SQL TABLES
- Contêm componentes internos
- São reutilizáveis

RECORDS e TABLES

Como variáveis escalares, as variáveis compostas também possuem um tipo de dados. Os tipos de dados compostos (também chamados conjuntos) são RECORD, TABLE, Nested TABLE, e VARRAY. Utilize o tipo de dados RECORD para manipular os dados relacionados, porém diferentes, como uma unidade lógica. Utilize o tipo de dados TABLE para fazer referência e manipular conjuntos de dados como um objeto inteiro. Os tipos de dados Nested TABLE e VARRAY não são abordados neste curso.

Um registro é um grupo de itens de dados relacionados armazenados em campos, cada um com seu próprio nome e tipo de dados. Uma tabela contém uma coluna e uma chave primária para fornecer a você acesso a linhas semelhante a array. Uma vez definidos, as tabelas e registros podem ser reutilizados.

Para obter mais informações, consulte o PL/SQL User's Guide and Reference, Release 8, "Collections and Records".

Registros PL/SQL

- Devem conter um ou mais componentes de qualquer tipo de dados escalar, RECORD ou PL/SQL TABLE, chamados campos
- São semelhantes em estrutura a registros em um 3GL
- Não são iguais a linhas em uma tabela de banco de dados
- Tratam um conjunto de campos como uma unidade lógica
- São convenientes para extrair uma linha de dados de uma tabela para processamento

Registros PL/SQL

Um registro é um grupo de itens de dados relacionados armazenados em campos, cada um com seu próprio nome e tipo de dados. Por exemplo, suponha que você tenha tipos diferentes de dados sobre um funcionário, como nome, salário, data de admissão etc. Esses dados são diferentes em tipo porém estão relacionados logicamente. Um registro que contém campos, como o nome, o salário e a data de admissão de um funcionário permite manipular os dados como uma unidade lógica. Quando você declara um tipo de registro para esses campos, eles podem ser manipulados como uma unidade.

- Cada registro definido pode ter tantos campos quantos forem necessários.
- Os registros podem receber atribuição de valores iniciais e podem ser definidos como NOT NULL.
- Os campos sem valores iniciais são inicializados para NULL.
- A palavra-chave DEFAULT também pode ser usada ao definir campos.
- Você pode definir tipos RECORD e declarar registros definidos pelo usuário na parte declarativa de qualquer bloco, subprograma ou pacote.
- Pode-se declarar e fazer referência a registros aninhados. Um registro pode ser o componente de outro registro.

Criando um Registro PL/SQL

Sintaxe

```
TYPE type_name IS RECORD
    (field_declaration[, field_declaration]...);
identifier type_name;
```

Onde field_declaration é

```
field_name {field_type | variable%TYPE
            | tabela.coluna%TYPE | tabela%ROWTYPE}
            [[NOT NULL] {:= | DEFAULT} expr]
```

Definindo e Declarando um Registro PL/SQL

Para criar um registro, defina um tipo RECORD e, em seguida, declare registros desse tipo.

Na sintaxe:

type_name é o nome do tipo RECORD. (Esse identificador é usado para declarar registros.)

field_name é o nome de um campo dentro do registro

field_type é o tipo de dados do campo. (Ele representa qualquer tipo de dados PL/SQL exceto REF CURSOR. Você pode usar os atributos %TYPE e ROWTYPE.)

expr é o field_type ou um valor inicial

A restrição NOT NULL impede a atribuição de nulos a esses campos. Certifique-se de inicializar campos NOT NULL.

Criando um Registro PL/SQL

Declarar variáveis para armazenar o nome, a tarefa e o salário de um novo funcionário.

Exemplo

```
...  
    TYPE emp_record_type IS RECORD  
        (ename      VARCHAR2(10),  
         job        VARCHAR2(9),  
         sal         NUMBER(7,2));  
    emp_record emp_record_type;  
...
```

Criando um Registro PL/SQL

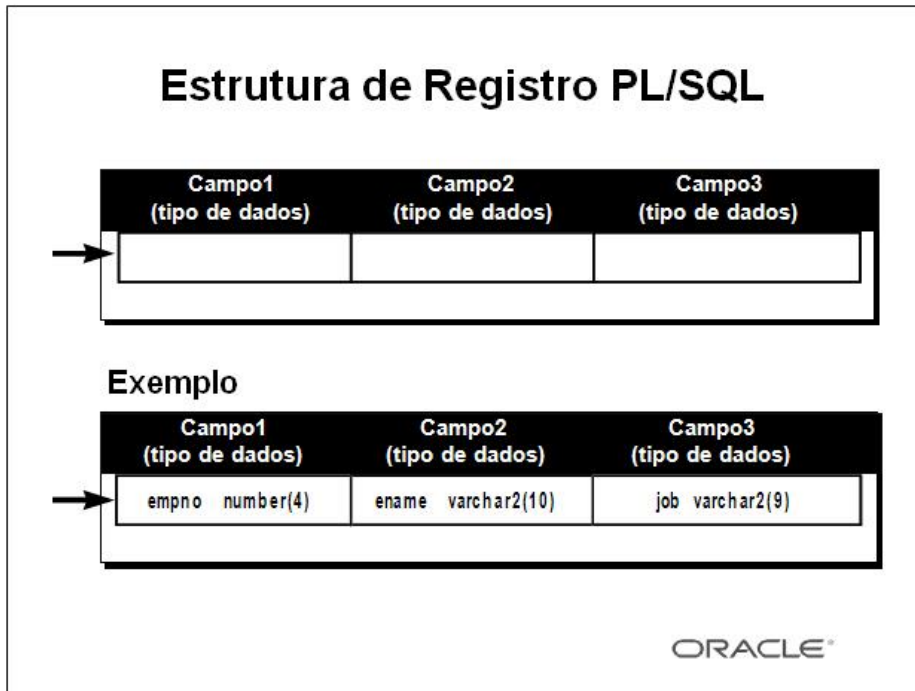
Declarações de campo são semelhantes a declarações de variável. Cada campo tem um nome exclusivo e um tipo de dados específico. Não existem tipos de dados predefinidos para registros PL/SQL, como existem para variáveis escalares. Assim, você deve primeiro criar o tipo de dados e, em seguida, declarar um identificador usando esse tipo de dados.

O exemplo a seguir mostra que você pode usar o atributo %TYPE para especificar um tipo de dados de campo:

```
DECLARE  
    TYPE emp_record_type IS RECORD  
        (empno      NUMBER(4) NOT NULL := 100,  
         ename       emp.ename%TYPE,  
         job         emp.job%TYPE);  
    emp_record emp_record_type;  
...
```

Observação: Você pode adicionar a restrição NOT NULL a qualquer declaração de campo para impedir a atribuição de nulos a esse campo. Lembre-se, os campos declarados como NOT NULL devem ser inicializados.

Estrutura de Registro PL/SQL



Fazendo Referência e Inicializando Registros

Os campos em um registro são acessados por nome. Para fazer referência ou inicializar um campo individual, use notação em pontos e a seguinte sintaxe:

```
record_name.field_name
```

Por exemplo, você faz referência ao campo job no registro emp_record do seguinte modo:

```
emp_record.job ...
```

É possível, em seguida, atribuir um valor ao campo de registro do seguinte modo:

```
emp_record.job := 'CLERK';
```


Em um bloco ou subprograma, os registros definidos pelo usuário são concretizados quando você inclui o bloco ou subprograma e deixam de existir quando você sai do bloco ou subprograma.

Atribuindo Valores aos Registros

Você pode atribuir uma lista de valores comuns a um registro usando a instrução SELECT ou FETCH. Certifique-se de que os nomes de colunas aparecem na mesma ordem dos campos no registro. Você também pode atribuir um registro a outro se eles tiverem o mesmo tipo de dados.

Um registro definido pelo usuário e um registro %ROWTYPE nunca têm o mesmo tipo de dados.

O Atributo %ROWTYPE

- Declarar uma variável segundo um conjunto de colunas em uma view ou tabela de banco de dados.
- Prefixar %ROWTYPE com a tabela de banco de dados.
- Campos no registro obtêm seus nomes e tipos de dados a partir das colunas da tabela ou view.

Declarando Registros com o Atributo %ROWTYPE

Para declarar um registro com base em um conjunto de colunas em uma view ou tabela de banco de dados, use o atributo %ROWTYPE. Os campos no registro obtêm seus nomes e tipos de dados a partir das colunas da tabela ou view. O registro também pode armazenar uma linha inteira de dados extraída de um cursor ou variável de cursor.

No exemplo a seguir, um registro é declarado usando %ROWTYPE como um especificador de tipo de dados.

```
DECLARE  
emp_record emp%ROWTYPE;  
...
```

O registro, *emp_record*, terá uma estrutura consistindo nos campos a seguir, cada um deles representando uma coluna na tabela EMP.

Observação: Isso não é código, mas simplesmente a estrutura da variável composta.

(empno	NUMBER(4),
ename	VARCHAR2(10),
job	VARCHAR2(9),
mgr	NUMBER(4),
hiredate	DATE,
sal	NUMBER(7,2),
comm	NUMBER(7,2),
deptno	NUMBER(2))

Vantagens de Usar %ROWTYPE

- O número e tipos de dados das colunas de banco de dados subjacentes podem não ser conhecidas.
- O número e tipos de dados da coluna de banco de dados subjacente pode alterar no tempo de execução.
- O atributo é útil ao recuperar uma linha com a instrução SELECT.

Declarando Registros com o Atributo %ROWTYPE (continuação)

Sintaxe

```
DECLARE  
    identificador      referência%ROWTYPE;
```

onde:

`identificador` é o nome escolhido para o registro como um todo

`referência` é o nome da tabela, view, cursor ou variável de cursor em que o registro deve ser baseado. (Você deve certificar-se de que essa referência é válida ao declarar o registro, isto é, a tabela ou view precisa existir.)

Para fazer referência a um campo individual, use notação em pontos e a seguinte sintaxe:

```
record_name.field_name
```

Por exemplo, você faz referência ao campo comm no registro emp_record do seguinte modo:

```
emp_record.comm
```

Pode-se, em seguida, atribuir um valor ao campo de registro do seguinte modo:

```
emp_record.comm := 750;
```

O Atributo %ROWTYPE

Exemplos

Declarar uma variável para armazenar a mesma informação sobre um departamento que está armazenada na tabela DEPT.

```
dept_record dept%ROWTYPE;
```

Declarar uma variável para armazenar a mesma informação sobre um funcionário que está armazenada na tabela EMP.

```
emp_record emp%ROWTYPE;
```

Exemplos

A primeira declaração no slide cria um registro com os mesmos nomes de campo e tipos de dados de campo da linha na tabela DEPT. Os campos são DEPTNO, DNAME e LOCATION.

A segunda declaração cria um registro com os mesmos nomes de campo e tipos de dados de campo de uma linha na tabela EMP. Os campos são EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM e DEPTNO.

No exemplo a seguir, um funcionário está se aposentando. As informações sobre esse funcionário são adicionadas a uma tabela que contém informações sobre funcionários aposentados. O usuário fornece o número do funcionário.

```
DECLARE
    emp_rec      emp%ROWTYPE;
BEGIN
    SELECT * INTO emp_rec
    FROM emp
    WHERE empno =      &employee_number;
    INSERT INTO retired_ems(empno, ename, job, mgr,
        hiredate, leavedate, sal, comm, deptno)
    VALUES (emp_rec.empno, emp_rec.ename, emp_rec.job,
        emp_rec.mgr, emp_rec.hiredate, SYSDATE,
        emp_rec.sal, emp_rec.comm, emp_rec.deptno);
    COMMIT;
END;
```

Tabelas PL/SQL

- São compostas de dois componentes:
 - Chave primária de tipo de dados `BINARY_INTEGER`
 - Coluna de tipo de dados escalares ou de registro
- Crescem dinamicamente por não conter restrições

Tabelas PL/SQL

Os objetos do tipo `TABLE` são chamados tabelas PL/SQL. Eles são modelados como (mas não iguais a) tabelas de banco de dados. As tabelas PL/SQL usam uma chave primária para fornecer a você acesso a linhas semelhante a array.

Uma tabela PL/SQL:

- É semelhante a um array
- Deve conter dois componentes:
 - Uma chave primária do tipo de dados `BINARY_INTEGER` que indexa a PL/SQL `TABLE`
 - Uma coluna de um tipo de dados escalares ou de registro, que armazena os elementos de PL/SQL `TABLE`
- Pode crescer dinamicamente por não conter restrições

Criando uma Tabela PL/SQL

Sintaxe

```
TYPE type_name IS TABLE OF
    {column_type | variável%TYPE
    | tabela.coluna%TYPE} [NOT NULL]
    [INDEX BY BINARY_INTEGER];
identificador type_name;
```

Declarar uma tabela PL/SQL para armazenar nomes.

Exemplo

```
...
TYPE ename_table_type IS TABLE OF emp.ename%TYPE
    INDEX BY BINARY_INTEGER;
ename_table ename_table_type;
...
```

Criando uma Tabela PL/SQL

Existem duas etapas envolvidas na criação de uma tabela PL/SQL.

1. Declarar um tipo de dados TABLE.
2. Declarar uma variável desse tipo de dados.

Na sintaxe:

type_name

é o nome do tipo TABLE. (Ele é um especificador de tipo usado em declarações subseqüentes de tabelas PL/SQL.)

column_type

é qualquer tipo de dados escalares (não composto), como VARCHAR2, DATE ou NUMBER. (Você pode usar o atributo %TYPE para fornecer o tipo de dados da coluna.)

identificador

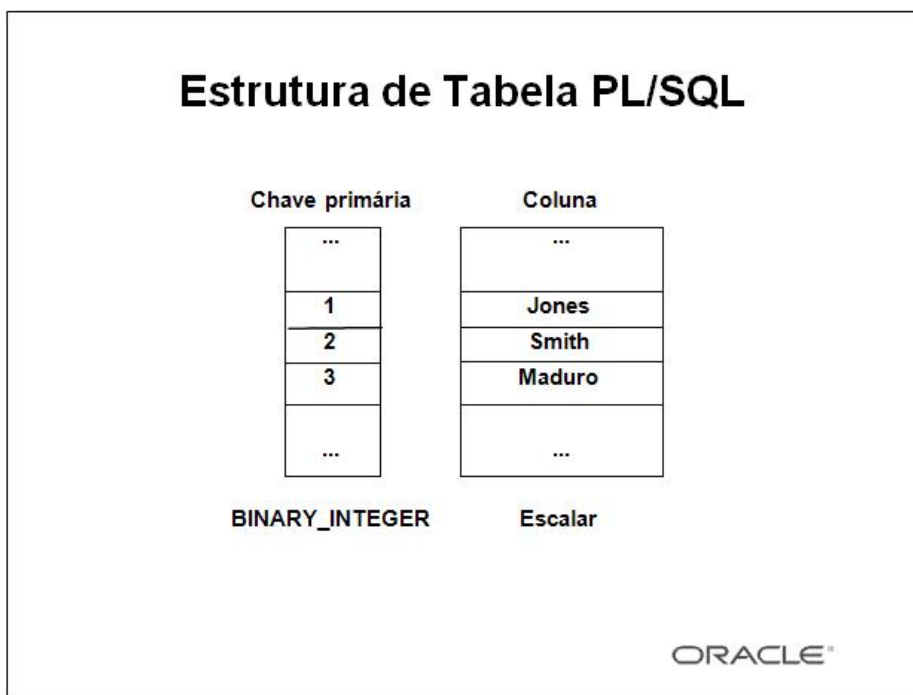
é o nome do identificador que representa uma tabela PL/SQL inteira

A restrição NOT NULL impede que nulos sejam atribuídos à PL/SQL TABLE desse tipo. Não inicialize a PL/SQL TABLE.

Declare uma tabela PL/SQL para armazenar datas.

```
DECLARE
  TYPE date_table_type IS TABLE OF DATE
    INDEX BY BINARY_INTEGER;
  date_table date_table_type;
```

Estrutura de Tabela PL/SQL



Estrutura de Tabela PL/SQL

Da mesma forma que o tamanho de uma tabela de banco de dados, o tamanho de uma tabela PL/SQL não tem restrição. Isto é, o número de linhas em uma tabela PL/SQL pode aumentar dinamicamente, assim a tabela PL/SQL pode crescer à medida que novas linhas são adicionadas.

As tabelas PL/SQL podem ter uma coluna e uma chave primária, nenhum desses itens pode ser nomeado. A coluna pode pertencer a qualquer tipo de dados escalares ou de registro, porém a

chave primária deve pertencer ao tipo `BINARY_INTEGER`. Não é possível inicializar uma tabela PL/SQL em sua declaração.

Criando uma Tabela PL/SQL

```
DECLARE
    TYPE ename_table_type IS TABLE OF emp.ename%TYPE
        INDEX BY BINARY_INTEGER;
    TYPE hiredate_table_type IS TABLE OF DATE
        INDEX BY BINARY_INTEGER;
    ename_table ename_table_type;
    hiredate_table hiredate_table_type;
BEGIN
    ename_table(1) := 'CAMERON';
    hiredate_table(8) := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
        INSERT INTO ...
    ...
END;
```

Criando uma Tabela PL/SQL

Não existem tipos de dados predefinidos para tabelas PL/SQL, como existem para variáveis escalares. Portanto, você precisa primeiro criar o tipo de dados e, em seguida, declarar um identificador usando esse tipo de dados.

Fazendo Referência a uma Tabela PL/SQL

Sintaxe

`pl/sql_table_name(primary_key_value)`

onde: `primary_key_value` pertence ao tipo `BINARY_INTEGER`.

Faça referência à terceira linha em uma tabela PL/SQL `ename_table`.

`ename_table(3) ...`

A faixa de magnitude de um `BINARY_INTEGER` é - 2147483647 ... 2147483647, então o valor da chave primária pode ser negativo. A indexação não precisa iniciar em 1.

Observação: A instrução `tabela.EXISTS(i)` retornará TRUE se pelo menos uma linha com índice `i` for retornada. Use a instrução `EXISTS` para impedir que ocorra um erro em relação a um elemento não existente na tabela.

Usando Métodos de Tabela PL/SQL

Os métodos a seguir facilitam o uso de tabelas PL/SQL:

- EXISTS
- COUNT
- FIRST and LAST
- PRIOR
- NEXT
- EXTEND
- TRIM
- DELETE

Um método de Tabela PL/SQL é uma função ou procedimento interno que opera sobre tabelas e é chamado usando notação em pontos. Os métodos abaixo assinalados com um asterisco estão disponíveis para tabelas PL/SQL versão 8 somente.

Sintaxe

`table_name.method_name[(parâmetros)]`

Método	Descrição
EXISTS(<i>n</i>)	Retornará TRUE se o <i>n</i> ésimo elemento em uma tabela PL/SQL
COUNT	Retorna o número de elementos contidos uma tabela PL/SQL
FIRST LAST	Retorna o primeiro e último (menor e maior) números de índice de uma tabela PL/SQL. Retornará NULL se a tabela PL/SQL estiver vazia.

PRIOR(<i>n</i>)	Retorna o número de índice que precede o índice <i>n</i> em uma tabela PL/SQL.
NEXT(<i>n</i>)	Retorna o número do índice que sucede o índice <i>n</i> em uma tabela PL/SQL.
EXTEND(<i>n</i> , <i>i</i>)*	Aumenta o tamanho de uma tabela PL/SQL. EXTEND anexa um elemento nulo a uma tabela PL/SQL. EXTEND(<i>n</i>) anexa <i>n</i> elementos nulos a uma tabela PL/SQL. EXTEND(<i>n</i> , <i>i</i>) anexa <i>n</i> cópias do elemento <i>i</i> a uma tabela PL/SQL.
TRIM*	TRIM remove um elemento do final de uma tabela PL/SQL. TRIM(<i>n</i>) remove <i>n</i> elementos do final de uma tabela PL/SQL.
DELETE	DELETE remove todos os elementos de uma tabela PL/SQL. DELETE(<i>n</i>) remove o <i>n</i> ésimo elemento de uma tabela PL/SQL. DELETE(<i>m</i> , <i>n</i>) remove todos os elementos na faixa <i>m</i> ... <i>n</i> de uma tabela PL/SQL.

Tabela de Registros PL/SQL

- Definir uma variável TABLE com um tipo de dados PL/SQL permitido.
- Declarar uma variável PL/SQL para armazenar informações de departamento.

Exemplo

```
DECLARE
  TYPE dept_table_type IS TABLE OF dept%ROWTYPE
    INDEX BY BINARY_INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```

Tabela de Registros PL/SQL

Como somente uma definição de tabela é necessária para armazenar informações sobre todos os campos de uma tabela de banco de dados, a tabela de registros aumenta bastante a funcionalidade de tabelas

PL/SQL.

Fazendo Referência a uma Tabela de Registros

No exemplo fornecido no slide, você pode fazer referência a campos no registro `dept_table` porque cada elemento dessa tabela é um registro.

Sintaxe

```
table(index).field
```

Exemplo

```
dept_table(15).loc := 'Atlanta';
```

LOC representa um campo em DEPT_TABLE.

Observação: Você pode usar o atributo `%ROWTYPE` para declarar um registro que representa uma linha em uma tabela de banco de dados. A diferença entre o atributo `%ROWTYPE` e o tipo de dados composto `RECORD` é que `RECORD` permite que você especifique os tipos de dados de campos no registro ou que declare campos próprios.

Exemplo de Tabela de Registros PL/SQL

```
DECLARE
    TYPE e_table_type IS TABLE OF emp.Ename%Type
        INDEX BY BINARY_INTEGER;
    e_tab e_table_type;
BEGIN
    e_tab(1) := 'SMITH';
    UPDATE emp
    SET sal = 1.1 * sal
    WHERE Ename = e_tab(1);
    COMMIT;
END;
/
```

Exemplo de Tabela de Registros PL/SQL

O exemplo no slide declara uma tabela PL/SQL `e_table_type`. Usando essa tabela PL/SQL, outra tabela, `e_tab`, é declarada. Na seção executável do bloco PL/SQL, a tabela `e_tab` é usada para atualizar o salário do funcionário, Smith.

Sumário

- Definir e fazer referência a variáveis PL/SQL de tipos de dados compostos:
 - Registros PL/SQL
 - Tabelas PL/SQL
 - Tabela de registros PL/SQL
- Definir um registro PL/SQL usando o atributo %ROWTYPE.

Sumário

Um registro PL/SQL é um conjunto de campos individuais que representam uma linha na tabela. Eles são exclusivos e cada linha tem seu próprio nome e tipo de dados. O registro como um todo não tem qualquer valor. Usando registros você pode agrupar os dados em uma estrutura e, em seguida, manipular essa estrutura para uma entidade ou unidade lógica. Isso ajuda a reduzir a codificação e torna o código mais fácil de manter e compreender.

Assim como registros PL/SQL, a tabela é outro tipo de dados composto. As tabelas PL/SQL são objetos do tipo TABLE e têm aparência semelhante a tabelas de banco de dados, mas com uma ligeira diferença. As tabelas PL/SQL usam uma chave primária para proporcionar a você acesso a linha semelhante a array. O tamanho de uma tabela PL/SQL não tem restrição. A tabela PL/SQL pode ter uma coluna e uma chave primária, nenhum desses itens pode ser nomeado. A coluna pode ter qualquer tipo de dados, mas a chave primária deve ser do tipo BINARY_INTEGER.

Uma tabela de registros PL/SQL aprimora a funcionalidade de tabelas PL/SQL, já que somente uma definição de tabela é necessária para armazenar informações sobre todos os campos.

O %ROWTYPE é usado para declarar uma variável composta cujo tipo é igual ao de uma linha de uma tabela de banco de dados.

Capítulo 06

Criando Cursores Explícitos

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Distinguir entre um cursor implícito e um explícito
- Usar uma variável de registro PL/SQL
- Criar um loop FOR de cursor

Objetivo da Lição

Nesta lição, você aprenderá as diferenças entre cursores implícitos e explícitos. Aprenderá também quando e porque usar um cursor explícito.

Você pode precisar usar uma instrução SELECT de várias linhas no PL/SQL para processar diversas linhas. Para isso, você declara e controla cursores explícitos, que são usados em loops, incluindo o loop FOR de cursor.

Sobre os Cursores

Cada instrução SQL executada pelo Oracle

Server tem um cursor individual associado:

- Cursores implícitos: Declarados para todas as instruções DML e PL/SQL SELECT
- Cursores explícitos: Declarados e nomeados pelo programador

Cursores Implícitos e Explícitos

O Oracle Server usa áreas de trabalho chamadas áreas SQL particulares para executar instruções

SQL e para armazenar informações de processamento. Você pode usar cursores do PL/SQL para nomear uma área SQL particular e acessar suas informações armazenadas. O cursor orienta todas as fases do processamento.

Tipo de Cursor	Descrição
Implícito	Os cursores implícitos são declarados implicitamente pelo PL/SQL para todas as instruções DML e PL/SQL SELECT, incluindo consultas que retornam somente uma linha.
Explícito	Para consultas que retornam mais de uma linha. Os cursores explícitos são declarados e nomeados pelo programador e manipulados através de instruções específicas nas ações executáveis do bloco.

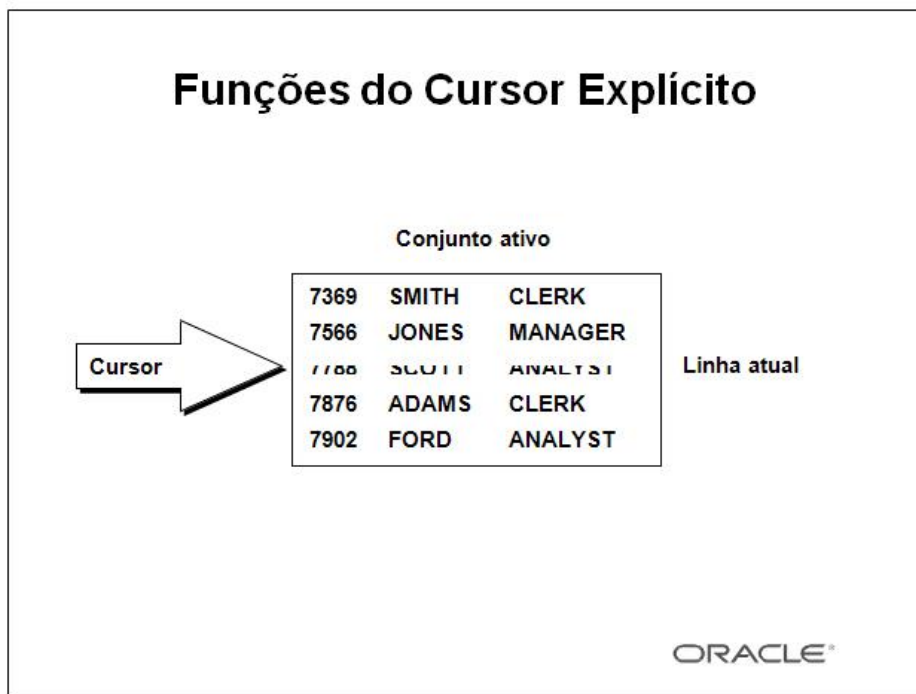
Cursores Implícitos

O Oracle Server abre implicitamente um cursor a fim de processar cada instrução SQL não associada a um cursor declarado

explicitamente. O PL/SQL permite que você consulte o cursor implícito mais recente como o cursor SQL.

Não é possível usar as instruções OPEN, FETCH e CLOSE para controlar o cursor SQL, mas você pode usar atributos de cursor para obter informações sobre a instrução SQL executada mais recentemente.

Funções do Cursor Explícito



Cursorres Explícitos

Use cursores explícitos para processar individualmente cada linha retornada por uma instrução

SELECT de várias linhas.

O conjunto de linhas retornado por uma consulta de várias linhas é chamado conjunto ativo. Seu tamanho é o número de linhas que atende aos critérios da pesquisa. O diagrama no slide mostra como um cursor explícito "aponta" para a linha atual do conjunto ativo. Isso permite que o programa processe as linhas uma de cada vez.

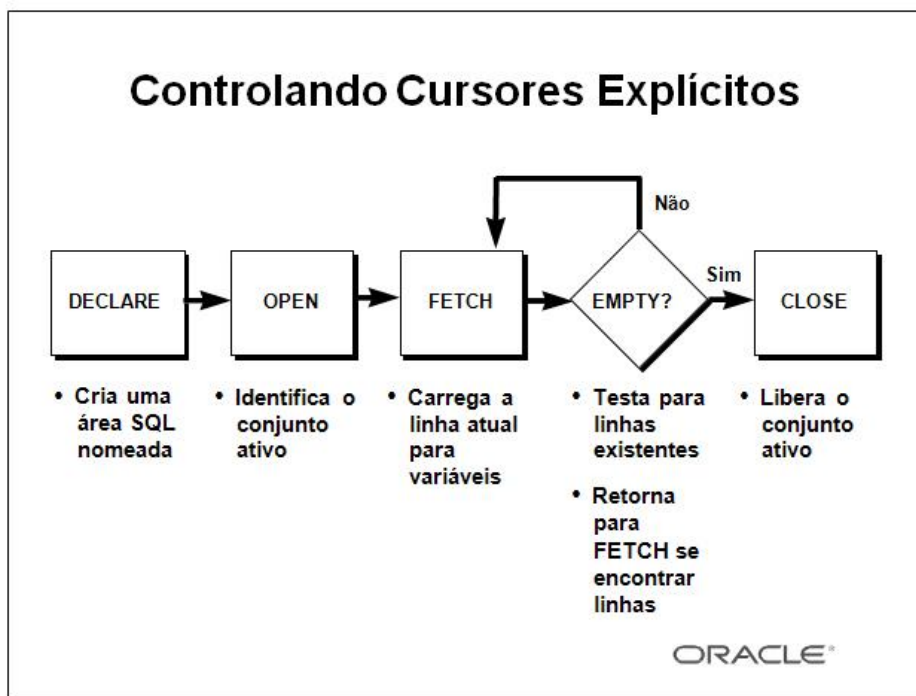
Um programa PL/SQL abre um cursor, processa linhas retornadas por uma consulta e, em seguida, fecha o cursor. O cursor marca a posição atual no conjunto ativo.

Funções do cursor explícito:

- Pode processar além da primeira linha retornada pela consulta, linha por linha
- Controla que linha está sendo processada no momento
- Permite que o programador controle as linhas manualmente no bloco PL/SQL

Observação: A extração de um cursor implícito é uma extração de array e a existência de uma segunda linha ainda criará a exceção `TOO_MANY_ROWS`. Além disso, você pode usar cursores explícitos para realizar diversas extrações e para executar novamente consultas analisadas na área de trabalho.

Controlando Cursores Explícitos



Cursores Explícitos (continuação)

Agora que você obteve uma compreensão conceitual dos cursores, verifique as etapas para usá-los.

A sintaxe de cada etapa pode ser encontrada nas páginas a seguir.

Controlando Cursores Explícitos Usando Quatro Comandos

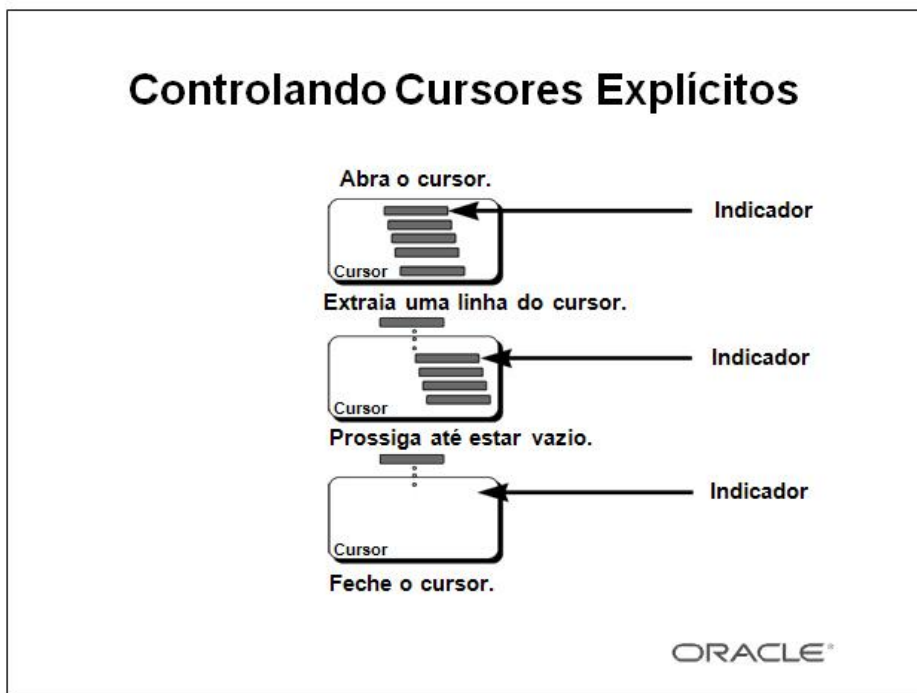
1. Declare o cursor nomeando-o e definindo a estrutura da consulta a ser executada dentro dele.

2. Abra o cursor. A instrução OPEN executa a consulta e vincula as variáveis que estiverem referenciadas. As linhas identificadas pela consulta são chamadas conjunto ativo e estão agora disponíveis para extração.

3. Extraia dados do cursor. No diagrama de fluxo mostrado no slide, após cada extração você testa o cursor para qualquer linha existente. Se não existirem mais linhas para serem processadas, você precisará fechar o cursor.

4. Feche o cursor. A instrução CLOSE libera o conjunto ativo de linhas. Agora é possível reabrir o cursor e estabelecer um novo conjunto ativo.

Controlando Cursores Explícitos



Cursosores Explícitos (continuação)

Você usa as instruções OPEN, FETCH e CLOSE para controlar um cursor. A instrução OPEN executa a consulta associada ao cursor, identifica o conjunto ativo e posiciona o cursor (indicador) antes da primeira linha. A instrução FETCH recupera a linha atual e avança o cursor para a próxima linha. Após o processamento da última linha, a instrução CLOSE desativa o cursor.

Declarando o Cursor

Sintaxe

```
CURSOR cursor_name IS  
select_statement;
```

- Não inclua a cláusula INTO na declaração do cursor.
- Caso seja necessário o processamento de linhas em uma sequência específica, use a cláusula ORDER BY na consulta.

Declaração de Cursor Explícito

Use a instrução CURSOR para declarar um cursor explícito. Pode-se fazer referência a variáveis dentro da consulta, mas você deve declará-las antes da instrução CURSOR.

Na sintaxe:

cursor_name

é um identificador do PL/SQL

select_statement

é uma instrução SELECT sem uma cláusula INTO

Observação: Não inclua a cláusula INTO na declaração de cursor porque ela aparecerá posteriormente na instrução FETCH.

Declarando o Cursor

Exemplo

```
DECLARE
    CURSOR emp_cursor IS
        SELECT empno, ename
        FROM emp;

    CURSOR dept_cursor IS
        SELECT *
        FROM dept
        WHERE deptno = 10;
BEGIN
    ...
```

Declaração de Cursor Explícito (continuação)

Recupere os funcionários um a um.

```
DECLARE
    v_empno    emp.empno%TYPE;
    v_ename    emp.ename%TYPE;
    CURSOR emp_cursor IS
        SELECT empno, ename
        FROM emp;
BEGIN
    ...
```

Observação: Pode-se fazer referência à variáveis na consulta, mas você deve declará-las antes da instrução CURSOR.

Abrindo o Cursor

Sintaxe

```
OPEN cursor_name;
```

- Abra o cursor para executar a consulta e identificar o conjunto ativo.
- Se a consulta não retornar qualquer linha, não será criada exceção.

- Use atributos de cursor para testar o resultado após uma extração.

Instrução OPEN

Abre o cursor para executar a consulta e identificar o conjunto ativo, que consiste em todas as linhas que atendem aos critérios de pesquisa da consulta. O cursor aponta agora para a primeira linha do conjunto ativo.

Na sintaxe:

cursor_name é o nome do cursor declarado anteriormente

OPEN é uma instrução executável que realiza as seguintes operações:

1. Aloca memória dinamicamente para uma área de contexto que finalmente conterá informações cruciais de processamento.
2. Analisa a instrução SELECT.
3. Vincula as variáveis de entrada — isto é, define o valor das variáveis de entrada obtendo seus endereços de memória.
4. Identifica o conjunto ativo — isto é, o conjunto de linhas que satisfaz os critérios de pesquisa. As linhas no conjunto ativo não são recuperadas para variáveis quando a instrução OPEN é executada. Em vez disso, a instrução FETCH recupera as linhas.
5. Posiciona o indicador imediatamente antes da primeira linha no conjunto ativo.

Instrução OPEN (continuação)

Observação: Se a consulta não retornar qualquer linha quando o cursor for aberto, o PL/SQL não criará uma exceção. Entretanto, você pode testar o status do cursor após a extração.

Para cursores declarados usando a cláusula FOR UPDATE, a instrução OPEN também bloqueia essas linhas. A cláusula FOR UPDATE será discutida em uma lição posterior.

Extraindo Dados do Cursor

Sintaxe

```
FETCH cursor_name INTO [variável1, variável2, ...]  
                        | record_name];
```

- Recuperar os valores da linha atual para variáveis.
- Incluir o mesmo número de variáveis.
- Fazer a correspondência de cada variável para coincidir com a posição das colunas.
- Testar para verificar se o cursor possui linhas.

Instrução FETCH

A instrução FETCH recupera as linhas no conjunto ativo uma de cada vez. Após cada extração, o cursor avança para a próxima linha no conjunto ativo.

Na sintaxe:

cursor_name é o nome do cursor declarado anteriormente

variável é uma variável de saída para armazenar os resultados

record_name é o nome do registro em que os dados recuperados são armazenados
(A variável de registro pode ser declarada usando o atributo %ROWTYPE.)

Diretrizes

- Inclua o mesmo número de variáveis na cláusula INTO da instrução FETCH do que as colunas na instrução SELECT e certifique-se de que os tipos de dados são compatíveis.
- Faça a correspondência de cada variável para coincidir com a posição das colunas.
- Como alternativa, defina um registro para o cursor e faça referência do registro na cláusula

FETCH INTO.

- Teste para verificar se o cursor possui linhas. Se uma extração não obtiver valores, não existem linhas remanescentes para serem processadas no conjunto ativo e nenhum erro será registrado.

Observação: A instrução FETCH realiza as seguintes operações:

1. Avança o indicador para a próxima linha no conjunto ativo.
2. Lê os dados da linha atual para as variáveis PL/SQL de saída.

Extraindo Dados do Cursor

Exemplos

```
FETCH emp_cursor INTO v_empno, v_ename;

...
OPEN defined_cursor;
LOOP
    FETCH defined_cursor INTO defined_variables
    EXIT WHEN ...;
    ...
    -- Process the retrieved data
... END;
```

Instrução FETCH (continuação)

Use a instrução FETCH para recuperar os valores da linha ativa para variáveis de saída. Após a extração, você pode manipular as variáveis através de instruções futuras. Para cada valor de coluna retornado pela consulta associada ao cursor, deve existir uma variável correspondente na lista INTO. Além disso, seus tipos de dados devem ser compatíveis.

Recupere os primeiros 10 funcionários um por um.

```
DECLARE
    v_empno    emp.empno%TYPE;
    v_ename    emp.ename%TYPE;
    CURSOR emp_cursor IS
        SELECT empno, ename
        FROM emp;

BEGIN
    OPEN emp_cursor;
    FOR i IN 1..10 LOOP
        FETCH emp_cursor INTO v_empno, v_ename;
```

```
...  
    END LOOP;  
END ;
```

Fechando o Cursor

Sintaxe

```
CLOSE cursor_name;
```

- Feche o cursor após completar o processamento das linhas.
- Reabra o cursor, se necessário.
- Não tente extrair dados de um cursor após ele ter sido fechado.

Instrução CLOSE

A instrução CLOSE desativa o cursor e o conjunto ativo se torna indefinido. Feche o cursor após completar o processamento da instrução SELECT. Essa etapa permite que o cursor seja reaberto, se necessário. Assim, você pode estabelecer um conjunto ativo diversas vezes.

Na sintaxe:

cursor_name é o nome do cursor declarado anteriormente.

Não tente extrair dados de um cursor após ele ter sido fechado ou será criada a exceção
INVALID_CURSOR.

Observação: A instrução CLOSE libera a área de contexto.

Embora seja possível terminar o bloco PL/SQL sem fechar cursores, você deve criar o hábito de fechar qualquer cursor declarado explicitamente para liberar recursos.

Existe um limite máximo para o número de cursores abertos por usuário, que é determinado pelo parâmetro OPEN_CURSORS no campo de parâmetros do banco de dados. OPEN_CURSORS = 50 por default.

```
...  
    FOR i IN 1..10 LOOP  
        FETCH emp_cursor INTO v_empno, v_ename;  
        ...  
    END LOOP;  
    CLOSE emp_cursor;
```

END;

Atributos do Cursor Explícito

Obter informações de status sobre um cursor.

Atributo	Tipo	Descrição
%ISOPEN	Booleano	Será avaliado para TRUE se o cursor estiver aberto
%NOTFOUND	Booleano	Será avaliado para TRUE se a extração mais recente não retornar uma linha
%FOUND	Booleano	Será avaliado para TRUE se a extração mais recente não retornar uma linha; complemento de %NOTFOUND
%ROWCOUNT	Número	Será avaliado para o número total de linhas retornadas até o momento

Atributos do Cursor Explícito

Da mesma forma que para cursores implícitos, existem quatro atributos para obter informações de status sobre um cursor. Quando anexado ao nome da variável do cursor, esses atributos retornam informações úteis sobre a execução de uma instrução de manipulação de dados.

Observação: Não é possível fazer referência a atributos de cursor diretamente em uma instrução SQL.

Controlando Várias Extrações

- Processar diversas linhas de um cursor explícito usando um loop.
- Extrair uma linha com cada iteração.
- Usar o atributo %NOTFOUND para criar um teste para uma extração malsucedida.

- Usar atributos de cursor explícito para testar o êxito de cada extração.

Controlando Várias Extrações de Cursores Explícitos

Para processar várias linhas de um cursor explícito, você normalmente define um loop para realizar uma extração em cada iteração. Finalmente todas as linhas no conjunto ativo serão processadas e uma extração malsucedida define o atributo %NOTFOUND para TRUE. Use os atributos de cursor explícito para testar o êxito de cada extração antes que sejam feitas referências futuras para o cursor.

Se você omitir um critério de saída, o resultado será um loop infinito.

Para obter mais informações, consulte o PL/SQL User's Guide and Reference, Release 8, "Interaction With Oracle".

O Atributo %ISOPEN

- Extrair linhas somente quando o cursor estiver aberto.
- Usar o atributo de cursor %ISOPEN antes de executar uma extração para testar se o cursor está aberto.

Exemplo

```
IF NOT emp_cursor%ISOPEN THEN
OPEN emp_cursor;
END IF; LOOP
FETCH emp_cursor...
```

Atributos do Cursor Explícito

- Você pode extrair linhas somente quando o cursor está aberto. Use o atributo de cursor %ISOPEN para determinar se o cursor está aberto, se necessário.

- Extraia linhas em um loop. Use atributos de cursor para determinar o momento para sair do loop.

- Use o atributo de cursor %ROWCOUNT para recuperar um número exato de linhas, extrair as linhas em um loop FOR numérico ou extrair as linhas em um loop simples e determinar o momento para sair do loop.

Observação: %ISOPEN retorna o status do cursor: TRUE se ele estiver aberto e FALSE se não estiver. Não é normalmente necessário examinar %ISOPEN.

Os Atributos %NOTFOUND e %ROWCOUNT

- Use o atributo de cursor %ROWCOUNT para recuperar um número exato de linhas.
- Use o atributo de cursor %NOTFOUND para determinar quando sair do loop.

Exemplo

Recupere os primeiros 10 funcionários um por um.

```
DECLARE
    v_empno      emp.empno%TYPE;
    v_ename      emp.ename%TYPE;
    CURSOR emp_cursor IS
        SELECT empno, ename
        FROM emp;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_empno, v_ename;
        EXIT WHEN emp_cursor%ROWCOUNT > 10 OR
            emp_cursor%NOTFOUND;
        ...
    END LOOP;
    CLOSE emp_cursor;
END ;
```

Exemplo (continuação)

Observação: Antes da primeira extração, %NOTFOUND é avaliado para NULL. Assim, se FETCH nunca executar com êxito, jamais ocorrerá saída do loop. Isso porque a instrução EXIT WHEN executará somente se sua condição WHEN for verdadeira. Como segurança, convém usar a seguinte instrução EXIT:

```
EXIT WHEN emp_cursor%NOTFOUND OR emp_cursor%NOTFOUND
IS NULL;
```

Se usar %ROWCOUNT, adicione um teste para nenhuma linha no cursor usando o atributo

%NOTFOUND, já que a contagem de linhas não será incrementada se a extração não recuperar qualquer linha.

Cursors e Registros

Processar convenientemente as linhas do conjunto ativo extraindo valores para um PL/SQL RECORD.

Exemplo

```
DECLARE
    CURSOR emp_cursor IS
        SELECT      empno, ename
        FROM        emp;
    emp_record      emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_record;
    ...
```

Cursors e Registros

Você já constatou que pode definir registros para usar a estrutura de colunas em uma tabela. Você também pode definir um registro com base na lista selecionada de colunas em um cursor explícito.

Isso é conveniente para processar as linhas do conjunto ativo, porque você pode simplesmente extrair para o registro. Assim, os valores das linhas são carregados diretamente para os campos correspondentes do registro.

Exemplo

Use um cursor para recuperar números e nomes de funcionários e preencher uma tabela de banco de dados temporária com essas informações.

```
DECLARE
    CURSOR emp_cursor IS
        SELECT empno, ename
        FROM emp;
    emp_record emp_cursor%ROWTYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO emp_record;
```

```
EXIT WHEN emp_cursor%NOTFOUND;
INSERT INTO temp_list (empid, empname)
VALUES (emp_record.empno, emp_record.ename);
END LOOP;
COMMIT;
CLOSE emp_cursor;
END ;
```

Loops FOR de Cursor

Sintaxe

```
FOR record_name IN cursor_name LOOP
    instrução1;
    instrução2;
    . . .
END LOOP;
```

- O loop FOR de cursor é um atalho para processar cursores explícitos.
- Ocorrem abertura, extração e fechamento implícitos.
- O registro é declarado implicitamente.

Loops FOR de Cursor

Um loop FOR de cursor processa linhas em um cursor explícito. Ele é um atalho porque o cursor é aberto, linhas são extraídas uma vez para cada iteração no loop e o cursor é fechado automaticamente após o processamento de todas as linhas. O loop em si é terminado automaticamente ao final da iteração quando a última linha for extraída.

Na sintaxe:

record_name

é o nome do registro declarado implicitamente

cursor_name

é um identificador PL/SQL para o cursor declarado anteriormente

Diretrizes

- Não declare o registro que controla o loop. Seu escopo é somente no loop.
- Teste os atributos do cursor durante o loop, se necessário.

- Forneça os parâmetros de um cursor, se necessário, entre parênteses após o nome do cursor na instrução FOR. Mais informações sobre parâmetros de cursor são abrangidas em uma lição subsequente.

- Não use um loop FOR de cursor quando as operações do cursor precisarem ser manipuladas manualmente.

Observação: Você pode definir uma consulta no início do próprio loop. A expressão da consulta é chamada subinstrução SELECT e o cursor é interno para o loop FOR. Como o cursor não é declarado com um nome, não é possível testar seus atributos.

Loops FOR de Cursor

Recuperar funcionários um a um até não restar nenhum.

Exemplo

```
DECLARE
    CURSOR emp_cursor IS
        SELECT ename, deptno
        FROM emp;
BEGIN
    FOR emp_record IN emp_cursor LOOP
        -- implicit open and implicit fetch occur
        IF emp_record.deptno = 30 THEN
            ...
        END LOOP; -- implicit close occurs
END;
```

Exemplo

Recupere funcionários um a um e imprima uma lista dos funcionários que estão trabalhando atualmente no departamento Sales. O exemplo do slide é concluído abaixo.

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR emp_cursor IS
        SELECT ename, deptno
        FROM emp;
BEGIN
    FOR emp_record IN emp_cursor LOOP
        --implicit open and implicit fetch occur
        IF emp_record.deptno = 30 THEN
            DBMS_OUTPUT.PUT_LINE ('Funcionário ' ||
emp_record.ename
```

```
                || ' trabalha no Dpt de Vendas.');
```

```
END IF;
```

```
END LOOP;    --implicit close occurs
```

```
END ;
```

```
/
```

Loops FOR do Cursor Usando Subconsultas

Não é necessário declarar o cursor.

Exemplo

```
BEGIN
```

```
  FOR emp_record IN (SELECT ename, deptno
```

```
                      FROM emp) LOOP
```

```
    -- implicit open and implicit fetch occur
```

```
    IF emp_record.deptno = 30 THEN
```

```
      ...
```

```
    END LOOP; -- implicit close occurs
```

```
END;
```

Loops FOR do Cursor Usando Subconsultas

Você não precisa declarar um cursor porque o PL/SQL permite substituição por uma subconsulta. Este exemplo realiza o mesmo do exemplo na página anterior. Ele é o código completo do slide acima.

```
SET SERVEROUTPUT ON
```

```
BEGIN
```

```
  FOR emp_record IN (SELECT ename, deptno
```

```
                      FROM emp) LOOP
```

```
    --implicit open and implicit fetch occur
```

```
    IF emp_record.deptno = 30 THEN
```

```
      DBMS_OUTPUT.PUT_LINE ('Funcionario ' ||
```

```
emp_record.ename
```

```
                        || ' trabalha no Dept. de Vendas. ');
```

```
    END IF;
```

```
  END LOOP;    --implicit close occurs
```

```
END ;
```

```
/
```

Sumário

- Tipos de cursor:
 - Cursores implícitos: Usados em todas as instruções DML e consultas de linha única.
 - Cursores explícitos: Usados para consultas de zero, uma ou mais linhas.
- É possível manipular cursores explícitos.
- É possível avaliar o status do cursor usando atributos de cursor.
- É possível usar loops FOR de cursor.

Sumário

Um cursor implícito é declarado pelo PL/SQL para cada instrução de manipulação de dados SQL.

O PL/SQL permite que você consulte o cursor implícito mais recente como o cursor SQL. O PL/SQL fornece quatro atributos para cada cursor. Esses atributos proporcionam a você informações úteis sobre as operações executadas com cursores. Pode-se usar o atributo de cursor anexando-o ao nome de um cursor explícito. Você pode usar esses atributos somente em instruções PL/SQL.

O PL/SQL permite que você processe linhas retornadas por uma consulta de várias linhas. Para processar individualmente uma linha em um conjunto de uma ou mais linhas retornadas por uma consulta, você pode declarar um cursor explícito.

Exemplo

Recupere os 5 primeiros itens de linha de um pedido um por um. À medida que cada produto é processado para o pedido, calcule o novo total do pedido e imprima-o na tela.

```
SET SERVEROUTPUT ON
ACCEPT p_ordid PROMPT 'Insira o número do pedido: '

DECLARE
    v_prodid                item.prodid%TY
    PE; v_item_total         NUMBER
    (11,2); v_order_total NUMBER
    (11,2) := 0; CURSOR item_cursor IS
        SELECT      prodid, actualprice * qty
        FROM        item
        WHERE        ordid = &p_ordid;
BEGIN
    OPEN
    item_cursor;
    LOOP
        FETCH item_cursor INTO v_prodid,
        v_item_total; EXIT WHEN
        item_cursor%ROWCOUNT > 5 OR
        item_cursor%NOTFOUND;
        v_order_total := v_order_total + v_item_total;
        DBMS_OUTPUT.PUT_LINE ('Número do produto ' || TO_CHAR
        (v_prodid) ||
        ' totaliza este pedido em ' ||
        TO_CHAR (v_order_total,
        'US$ 999,999.99')); END LOOP;
    CLOSE item_cursor;
END;
/
```

Cursors com Parâmetros

Sintaxe

```
CURSOR cursor_name
    [(parameter_name tipo de dados , ...)]
IS
select_statement;
```

- Passar valores de parâmetro para um cursor quando ele for aberto e a consulta for executada.
- Abrir um cursor explícito diversas vezes com um conjunto ativo diferente a cada vez.

Cursor com Parâmetros

Parâmetros permitem que valores sejam passados para um cursor quando ele é aberto e sejam usados na consulta quando ela é executada. Isso significa que você pode abrir e fechar um cursor explícito várias vezes em um bloco, retornando um conjunto ativo diferente em cada ocasião.

Cada parâmetro formal na declaração do cursor deve ter um parâmetro real correspondente na instrução OPEN. Os tipos de dados de parâmetro são iguais aos das variáveis escalares, porém você não define tamanhos para eles. Os nomes de parâmetro são para referência na expressão de consulta do cursor.

Na sintaxe:

cursor_name

é um identificador PL/SQL para o cursor declarado anteriormente

parameter_name

é o nome de um parâmetro (Parâmetro representa a sintaxe a seguir.)

```
cursor_parameter_name [IN] tipo de dados [{:=  
| DEFAULT} expr]
```

tipo de dados

é um tipo de dados escalares do parâmetro

select_statement

é uma instrução SELECT sem a cláusula INTO

Quando o cursor é aberto, você passa valores para cada parâmetro por posição. Pode-se passar valores de variáveis PL/SQL ou de host e também de literais.

Observação: A notação de parâmetro não oferece maior funcionalidade, ela simplesmente permite que você especifique valores de entrada de maneira fácil e clara. Isso é especialmente útil quando é feita referência ao mesmo cursor repetidamente.

Cursores com Parâmetros

Passar o nome do departamento e o título do cargo para a cláusula WHERE.

Exemplo

```
DECLARE
  CURSOR emp_cursor
    (p_deptno NUMBER, p_job VARCHAR2) IS
    SELECT      empno, ename
    FROM        emp
    WHERE deptno = v_deptno
    AND        job = v_job;
BEGIN
  OPEN emp_cursor(10, 'CLERK');
  ...
```

Os tipos de dados de parâmetro são iguais aos das variáveis escalares, porém você não define tamanhos para eles. Os nomes de parâmetro são para referências na consulta do cursor.

No exemplo a seguir, duas variáveis e um cursor são declarados. O cursor é definido com dois parâmetros.

```
DECLARE
  v_emp_job emp.job%TYPE := 'CLERK';
  v_ename   emp.ename%TYPE;
  CURSOR emp_cursor(p_deptno NUMBER, p_job
    VARCHAR2) IS SELECT ...
```

Qualquer das instruções a seguir abre o cursor:

```
OPEN
  emp_cursor(10,
  v_emp_job); OPEN
  emp_cursor(20,
  'ANALYST');
```

Você pode passar parâmetros para o cursor usado em um loop FOR de cursor:

```
DECLARE
    CURSOR emp_cursor(p_deptno NUMBER, p_job
        VARCHAR2) IS SELECT ...
BEGIN
    FOR emp_record IN emp_cursor(10, 'ANALYST') LOOP
        ...
```

A Cláusula FOR UPDATE

Sintaxe

```
SELECT ...
FROM ...
FOR UPDATE [OF column_reference] [NOWAIT];
```

- O bloqueio explícito permite que você negue acesso pela duração de uma transação.
- Bloqueie as linhas antes de atualizar ou deletar.

A Cláusula FOR UPDATE

Convém bloquear linhas antes de efetuar atualização ou exclusão de linhas. Adicione a cláusula FOR UPDATE à consulta de cursor para bloquear as linhas afetadas quando o cursor for aberto. Como o Oracle Server libera bloqueios ao final da transação, você não deve efetuar commit entre extrações a partir de um cursor explícito se a cláusula FOR UPDATE for usada.

Na sintaxe:

column_reference

é uma coluna na tabela na qual a consulta é realizada
(Uma lista de colunas também pode ser usada.)

NOWAIT

retornará um erro do Oracle se as linhas estiverem bloqueadas por outra sessão

A cláusula FOR UPDATE é a última cláusula em uma instrução SELECT, inclusive posterior a ORDER BY, se ela existir.

Ao consultar várias tabelas, você pode usar a cláusula FOR UPDATE para confinar o bloqueio de linhas a determinadas tabelas.

As linhas em uma tabela serão bloqueadas somente se a cláusula FOR UPDATE se referir a uma coluna nessa tabela.

Os bloqueios exclusivos de linha são obtidos no conjunto ativo antes de OPEN retornar quando a cláusula FOR UPDATE é usada.

A Cláusula FOR UPDATE

Recuperar os funcionários que trabalham no departamento 30.

Exemplo

```
DECLARE
  CURSOR emp_cursor IS
    SELECT empno, ename, sal
    FROM emp
    WHERE deptno = 30
    FOR UPDATE OF sal NOWAIT;
```

A Cláusula FOR UPDATE (continuação)

Observação: Se o Oracle Server não puder obter os bloqueios sobre as linhas de que ele necessita em uma instrução SELECT FOR UPDATE, ele aguardará indefinidamente. Você pode usar a cláusula NOWAIT na instrução SELECT FOR UPDATE para testar para código de erro retornado devido a falha na obtenção de bloqueios em um loop. Portanto, você pode tentar abrir novamente o cursor n vezes antes de terminar o bloco PL/SQL. Se tiver uma tabela grande, você terá melhor desempenho usando a instrução LOCK TABLE para bloquear todas as linhas na tabela. Entretanto, ao usar LOCK TABLE, você não pode usar a cláusula WHERE CURRENT OF e deve usar a notação WHERE

coluna = identificador.

Não é obrigatório que a cláusula FOR UPDATE OF se refira a uma coluna, mas é recomendado para melhor legibilidade e manutenção.

A Cláusula WHERE CURRENT OF

Sintaxe

```
WHERE CURRENT OF cursor ;
```

- Usar cursores para atualizar ou deletar a linha atual.
- Incluir a cláusula FOR UPDATE na consulta de cursor para primeiro bloquear as linhas.
- Usar a cláusula WHERE CURRENT OF para fazer referência à linha atual a partir de um cursor explícito.

A Cláusula WHERE CURRENT OF

Ao fazer referência à linha atual de um cursor explícito, use a cláusula WHERE CURRENT OF. Isso permite que você aplique atualizações e deleções à linha que está sendo tratada no momento, sem necessidade de fazer referência explícita a ROWID. Você deve incluir a cláusula FOR UPDATE na consulta do cursor para que as linhas sejam bloqueadas em OPEN.

Na sintaxe:

cursor é o nome de um cursor declarado (O cursor deve ter sido declarado com a cláusula FOR UPDATE.)

A Cláusula WHERE CURRENT OF

Exemplo

```
DECLARE
    CURSOR sal_cursor IS
        SELECT      sal
        FROM        emp
        WHERE deptno = 30
        FOR UPDATE OF sal NOWAIT;

BEGIN
    FOR emp_record IN sal_cursor LOOP
        UPDATE      emp
        SET          sal = emp_record.sal * 1.10
        WHERE CURRENT OF sal_cursor;
    END LOOP;
    COMMIT;
END;
```

A Cláusula WHERE CURRENT OF (continuação)

Você pode atualizar linhas com base em critérios de um cursor.

Além disso, pode criar a instrução DELETE ou UPDATE para conter a cláusula WHERE CURRENT OF cursor_name para fazer referência à linha mais recente processada pela instrução FETCH. Ao utilizar essa cláusula, o cursor ao qual você faz referência precisa existir e deve conter a cláusula FOR UPDATE na consulta do cursor; caso contrário receberá um erro. Essa cláusula permite aplicar atualizações e deleções à linha atual sem que seja necessário fazer referência explícita à pseudocoluna ROWID.

Exemplo

O slide de exemplo efetua loop por cada funcionário no departamento 30, elevando cada salário em 10%.

A cláusula WHERE CURRENT OF na instrução UPDATE refere-se ao registro extraído no momento.

Cursores com Subconsultas

Exemplo

```
DECLARE
  CURSOR my_cursor IS
    SELECT t1.deptno, t1.dname, t2.STAFF
    FROM dept t1, (SELECT deptno,
                        count(*) STAFF
                     FROM emp
                     GROUP BY deptno) t2
    WHERE t1.deptno = t2.deptno
    AND t2.STAFF >= 5;
```

Subconsultas

Uma subconsulta é uma consulta (geralmente entre parênteses) que aparece dentro de outra instrução

de manipulação de dados SQL. Quando avaliada, a subconsulta fornece um valor ou conjunto de valores para a instrução.

As subconsultas são freqüentemente usadas na cláusula WHERE de uma instrução SELECT. Elas também podem ser usadas na cláusula FROM, criando uma origem de dados temporária para essa consulta. Neste exemplo, a subconsulta cria uma origem de dados consistindo em números de departamentos e contagem de

funcionários em cada departamento (conhecido pelo apelido STAFF). Um apelido de tabela, t2, refere-se a essa origem de dados temporária na cláusula FROM. Quando esse cursor for aberto, o conjunto ativo conterá o número do departamento, nome do departamento e a contagem de funcionários dos departamentos que sejam iguais ou superiores a 5.

É possível usar uma subconsulta ou subconsulta correlacionada.

Sumário

- É possível retornar diferentes conjuntos ativos usando cursores com parâmetros.
- É possível definir cursores com subconsultas e subconsultas correlacionadas.
- É possível manipular cursores explícitos com comandos:
 - Cláusula FOR UPDATE
 - Cláusula WHERE CURRENT OF

Sumário

Um cursor explícito pode conter parâmetros. Em uma consulta, você pode especificar um parâmetro de cursor sempre que uma constante possa aparecer. Uma vantagem de usar parâmetros é que você pode decidir o conjunto ativo no tempo de execução. O PL/SQL oferece um método para modificar as linhas que foram recuperadas pelo cursor. O método consiste em duas partes. A cláusula FOR UPDATE na declaração do cursor e a cláusula WHERE CURRENT OF em uma instrução UPDATE ou DELETE.

Master Training

Capítulo 07

Tratando Exceções

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Definir exceções PL/SQL
- Reconhecer exceções não tratáveis
- Listar e usar diferentes tipos de handlers de exceção PL/SQL
- Capturar erros inesperados
- Descrever o efeito da propagação de exceção em blocos aninhados
- Personalizar mensagens de exceção PL/SQL

Objetivo da Lição

Nesta lição, você aprenderá o que são exceções PL/SQL e como lidar com elas usando handlers de exceção predefinidos, não predefinidos e definidos pelo usuário.

Tratando Exceções com Código PL/SQL

- O que é uma exceção?
Identificador em PL/SQL que é criado durante a execução
- Como a exceção é criada?
 - Quando ocorre um erro do Oracle.
 - Quando você a cria explicitamente.
- Como você trata a exceção?
 - Capture-a com um handler.
 - Propaga-a para o ambiente de chamada.

Visão Geral

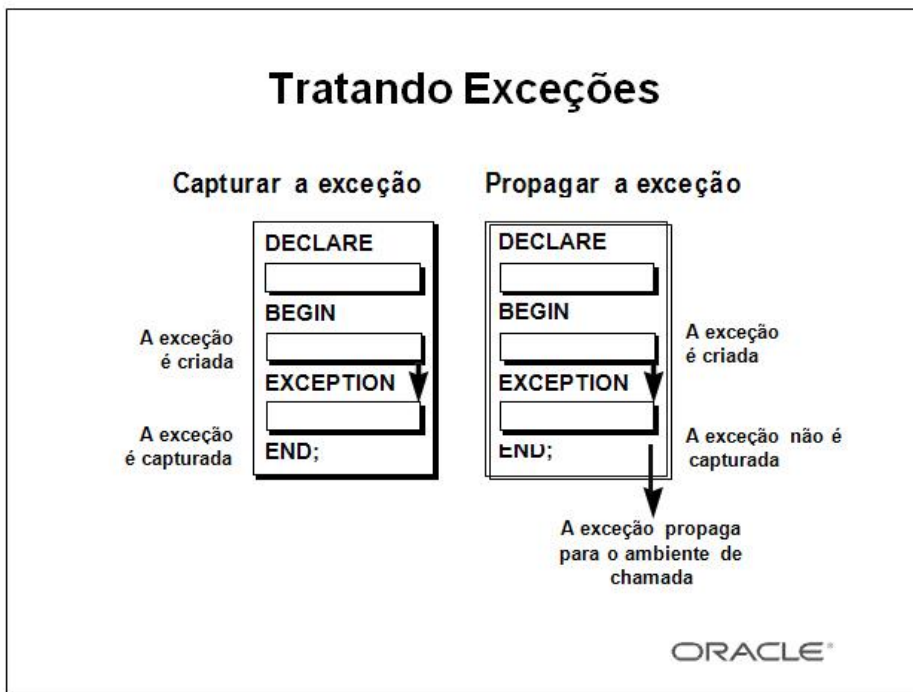
Uma exceção é um identificador em PL/SQL, criado durante a execução de um bloco que termina seu corpo principal de ações. Um bloco sempre termina quando o código PL/SQL cria uma exceção, mas você especifica um handler de exceções para executar ações finais.

Dois Métodos para Criar uma Exceção

- Ocorre um erro do Oracle e a exceção associada é criada automaticamente. Por exemplo, se ocorrer o erro ORA-01403 quando nenhuma linha for recuperada do banco de dados em uma instrução SELECT, em seguida, o código PL/SQL criará a exceção NO_DATA_FOUND.

- Crie uma exceção explicitamente emitindo a instrução RAISE dentro do bloco. A exceção criada pode ser definida ou predefinida pelo usuário.

Tratando Exceções



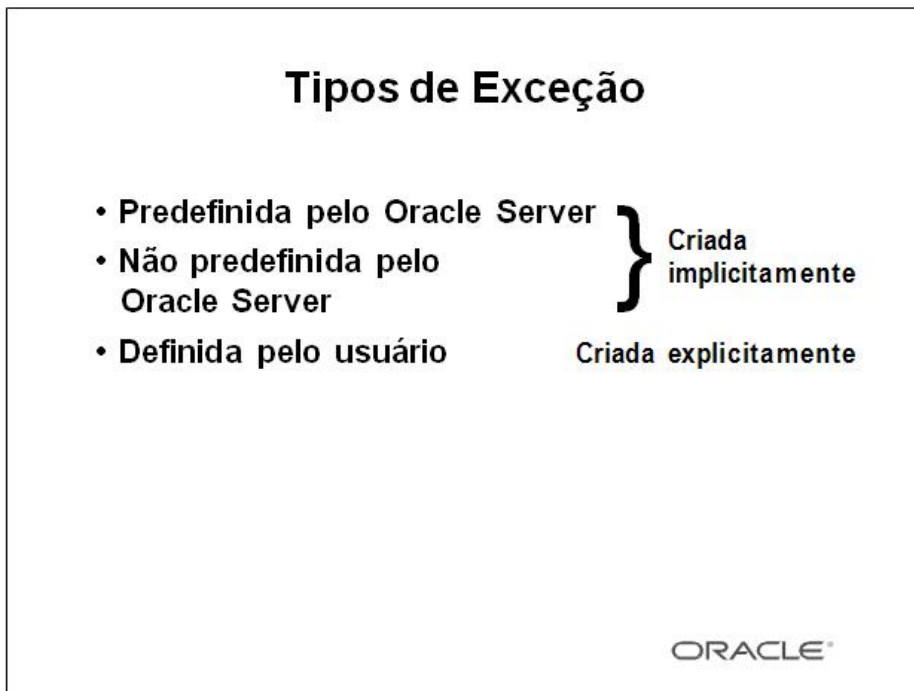
Capturando uma Exceção

Se a exceção for criada na seção executável do bloco, o processamento é desviado para o handler de exceção correspondente na seção de exceção do bloco. Se o código PL/SQL tratar a exceção com êxito, a exceção não propagará para o ambiente ou bloco delimitado. O bloco PL/SQL é concluído com êxito.

Propagando uma Exceção

Se a exceção for criada na seção executável do bloco e não houver handler de exceção correspondente, o bloco PL/SQL terminará com falha e a exceção será propagada para o ambiente de chamada.

Tipos de Exceção



Tipos de Exceção

Você pode programar exceções a fim de evitar transtornos no tempo de execução. Há três tipos de exceções.

Exceção	Descrição	Orientações para Tratamento
Erro predefinido pelo Oracle Server	Um dos cerca de 20 erros que ocorrem com mais frequência no código do PL/SQL	Não declare e permita que o Oracle Server crie as exceções implicitamente
Erro não predefinido pelo Oracle Server	Qualquer outro erro padrão do Oracle Server	Declare dentro da seção declarativa e permita que o Oracle Server crie as exceções de forma implícita

Erro definido pelo usuário	Uma condição que o desenvolvedor determina que seja anormal.	Declare dentro da seção declarativa e crie exceções de forma explícita
----------------------------	--	--

Observação: Algumas ferramentas de aplicação com código PL/SQL cliente, como o Oracle Developer Forms, têm suas próprias exceções.

Capturando Exceções

Sintaxe

```
EXCEPTION
  WHEN exceção1 [OR exceção2 . . .] THEN
    instrução1;
    instrução2;
    . . .
  [WHEN exceção3 [OR exceção4 . . .] THEN
    instrução1;
    instrução2;
    . . .]
  [WHEN OTHERS THEN instrução1; instrução2;
    . . .]
```

Capturando Exceções

Você pode capturar qualquer erro incluindo uma rotina correspondente dentro da seção de tratamento

de exceções do bloco PL/SQL. Cada handler consiste em uma cláusula **WHEN**, que especifica uma exceção, seguida por uma sequência de instruções a serem executadas quando essa exceção for criada.

Na sintaxe:

exceção é o nome padrão de uma exceção predefinida ou o nome de uma exceção definida pelo usuário declarada na seção declarativa

instrução uma ou mais instruções PL/SQL ou SQL

OTHERS é uma cláusula de manipulação de exceção opcional que captura exceções não especificadas.

Handler de Exceção WHEN OTHERS

A seção de tratamento de exceção captura somente as exceções especificadas; quaisquer outras exceções não serão capturadas exceto se você usar o handler de exceção OTHERS. Esse handler captura qualquer exceção que ainda não esteja tratada. Por essa razão, OTHERS é o último handler de exceção definido.

O handler OTHERS captura todas as exceções ainda não capturadas. Algumas ferramentas Oracle têm suas próprias exceções predefinidas que você pode criar para provocar eventos na aplicação. OTHERS também captura essas exceções.

Diretrizes para a Captura de Exceções

- WHEN OTHERS é a última cláusula.
- A palavra-chave EXCEPTION inicia a seção de tratamento de exceções.
- São permitidos vários handlers de exceção.
- Somente um handler é processado antes de se sair do bloco.

Diretrizes

- Inicie a seção de tratamento de exceções do bloco com a palavra-chave EXCEPTION.
- Defina vários handlers de exceção para o bloco, cada um deles com o seu próprio conjunto de ações.
- Quando ocorre uma exceção, o código PL/SQL processa somente um handler antes de sair do bloco.
- Coloque a cláusula OTHERS após todas as outras cláusulas de tratamento de exceção.
- Você pode ter no máximo uma cláusula OTHERS.
- As exceções não podem aparecer em instruções de atribuição ou instruções SQL.

Capturando Erros Predefinidos do Oracle Server

- Fazer referência ao nome padrão na rotina de tratamento de exceção.

- Exceções predefinidas de exemplo:

- NO_DATA_FOUND
- TOO_MANY_ROWS
- INVALID_CURSOR
- ZERO_DIVIDE
- DUP_VAL_ON_INDEX

Capturando Erros Predefinidos do Oracle Server

Capture um erro predefinido do Oracle Server fazendo referência ao seu nome padrão dentro da rotina de tratamento de exceção correspondente.

Para obter uma lista completa de exceções predefinidas, consulte o PL/SQL User's Guide and Reference, Release 8, "Error Handling".

Observação: O código PL/SQL declara exceções predefinidas no pacote STANDARD.

É bom sempre levar em consideração as exceções NO_DATA_FOUND e TOO_MANY_ROWS, que são as mais comuns.

Exceções Predefinidas

Nome da Exceção	Número do Erro do Oracle Server	Descrição
ACCESS_INTO_NULL	ORA-06530	Tentativa de atribuir valores aos atributos de um objeto não inicializado
COLLECTION_IS_NULL	ORA-06531	Tentativa de aplicação de métodos de conjunto diferentes de EXISTS para um varray ou tabela aninhada não inicializada
CURSOR_ALREADY_OPEN	ORA-06511	Tentativa de abertura de um cursor já aberto
DUP_VAL_ON_INDEX	ORA-00001	Tentativa de inserção de um valor duplicado
INVALID_CURSOR	ORA-01001	Ocorreu operação ilegal de cursor
INVALID_NUMBER	ORA-01722	Falha da conversão de string de caracteres para número
LOGIN_DENIED	ORA-01017	Estabelecendo login com o Oracle com um nome de usuário ou senha inválida
NO_DATA_FOUND	ORA-01403	SELECT de linha única não retornou dados
NOT_LOGGED_ON	ORA-01012	O programa PL/SQL emite uma chamada de banco de dados sem estar conectado ao Oracle
PROGRAM_ERROR	ORA-06501	O código PL/SQL tem um problema interno
ROWTYPE_MISMATCH	ORA-06504	Variável de cursor de host e variável de cursor PL/SQL envolvidas em uma atribuição têm tipos de retorno incompatíveis

STORAGE_ERROR	ORA-06500	O PL/SQL esgotou a memória ou a memória está corrompida
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Feita referência ao elemento de varray ou tabela aninhada usando um número de índice maior do que o número de elementos no conjunto
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Feita referência a um elemento de varray ou tabela aninhada usando um número de índice fora da faixa legal (-1, por exemplo)
TIMEOUT_ON_RESOURCE	ORA-00051	Ocorreu timeout enquanto o Oracle está aguardando por um recurso
TOO_MANY_ROWS	ORA-01422	SELECT de uma única linha retornou mais de uma linha
VALUE_ERROR	ORA-06502	Ocorreu erro aritmético, de conversão, truncamento ou restrição de tamanho
ZERO_DIVIDE	ORA-01476	Tentativa de divisão por zero

Exceção Predefinida

Sintaxe

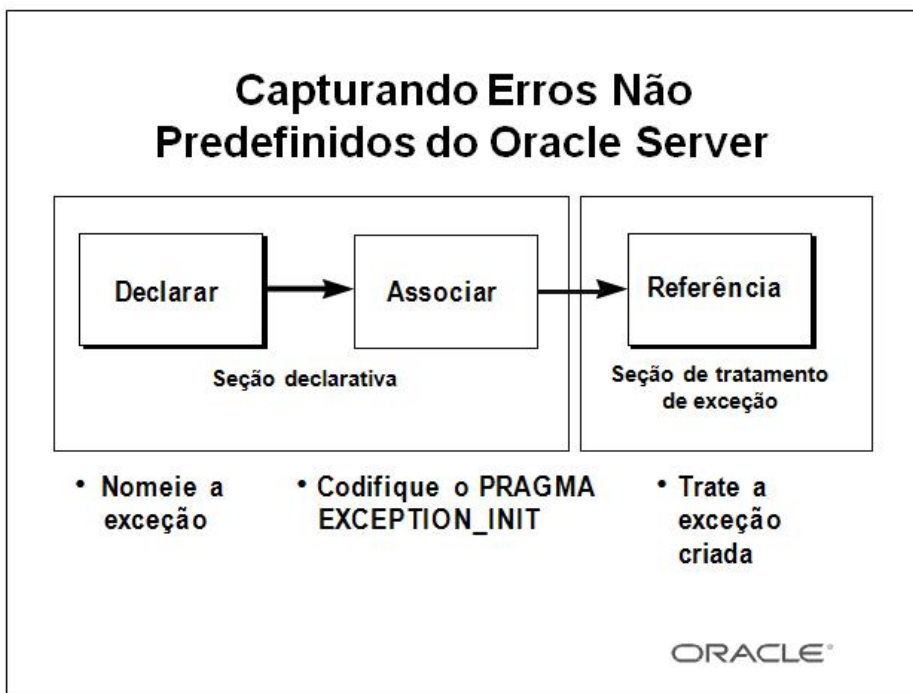
```

BEGIN
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        statement1;
        statement2;
    WHEN TOO_MANY_ROWS THEN
        statement1;
    WHEN OTHERS THEN
        statement1;
        statement2;
        statement3;
END;
```


Capturando Exceções Predefinidas do Oracle Server

Somente uma exceção é criada e tratada a qualquer momento.

Capturando Erros Não Predefinidos do Oracle Server



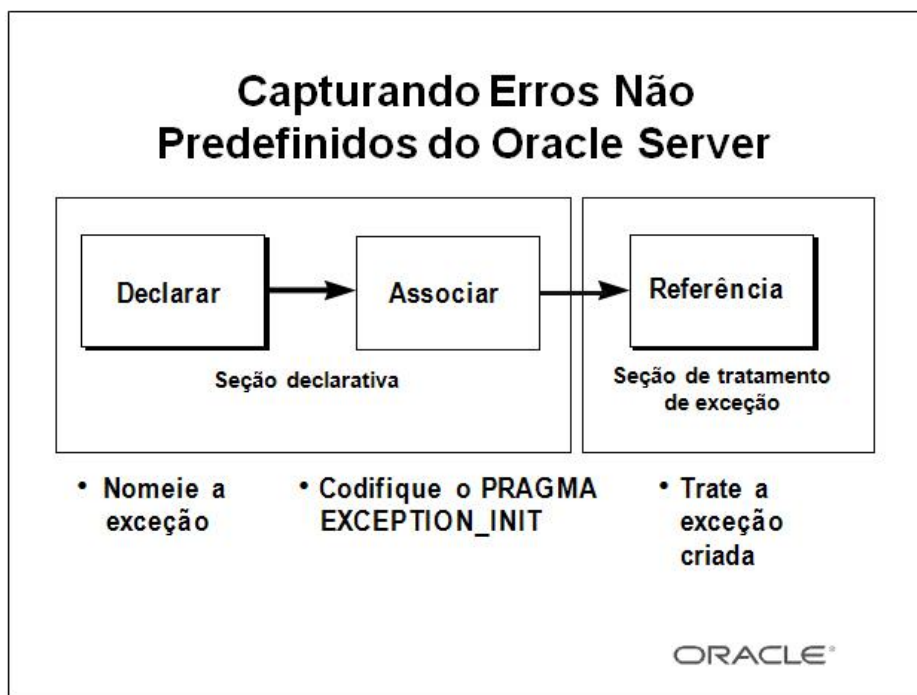
Capturando Erros Não Predefinidos do Oracle Server

Você captura um erro não predefinido do Oracle Server declarando-o primeiro ou usando o handler OTHERS. A exceção declarada é criada implicitamente. Em PL/SQL, o PRAGMA EXCEPTION_INIT informa o compilador para associar um nome de exceção a um número de erro do Oracle. Isso permite que você consulte qualquer exceção interna por nome e crie um handler específico para ela.

Observação: PRAGMA (também chamado pseudo-instruções) é uma palavra-chave que significa que a instrução é uma diretiva de compilador, que não é processada ao executar o bloco PL/SQL.

Em vez disso, ela orienta o compilador do PL/SQL para interpretar todas as ocorrências do nome da exceção dentro do bloco como o número de erro do Oracle Server associado.

Erro Não Predefinido



Capturando uma Exceção Não Predefinida do Oracle Server

1. Declare o nome para a exceção na seção declarativa.

Sintaxe

exceção EXCEPTION;

onde: exceção é o nome da exceção.

2. Associe a exceção declarada ao número de erro padrão do Oracle Server usando a instrução PRAGMA EXCEPTION_INIT.

Sintaxe

```
PRAGMA EXCEPTION_INIT(exceção, error_number);
```

onde: **exceção** é a exceção declarada anteriormente.
error_number é um número de erro padrão do Oracle Server.

3. Faça referência à exceção declarada dentro da rotina de tratamento de exceção correspondente.

Exemplo

Se existirem funcionários em um departamento, imprima uma mensagem para o usuário informando que esse departamento não pode ser removido.

Para obter mais informações, consulte o Oracle Server Messages, Release 8.

Funções para Captura de Exceções

- **SQLCODE**
Retorna o valor numérico do código de erro
- **SQLERRM**
Retorna a mensagem associada ao número de erro

Funções para Captura de Erro

Quando ocorre uma exceção, você pode identificar o código ou a mensagem de erro associado usando duas funções. Com base nos valores do código ou mensagem, você pode decidir qual ação subsequente tomar com base no erro.

SQLCODE retorna o número do erro do Oracle para exceções internas. Você pode passar um número de erro para **SQLERRM**, que então retorna a mensagem associada ao número do erro.

Função	Descrição
SQLCODE	Retorna o valor numérico do código de erro (Você pode atribuí-lo a uma variável NUMBER.)
SQLERRM	Retorna os dados de caracteres que contêm a mensagem associada ao número do erro

Exemplo de Valores SQLCODE

Valor SQLCODE	Descrição
0	Nenhuma exceção encontrada
1	Exceção definida pelo usuário
+100	Exceção NO_DATA_FOUND
número negativo	Outro número de erro do Oracle Server

Funções para Captura de Exceções

Funções para Captura de Exceções

Exemplo

```
DECLARE
    v_error_code      NUMBER;
    v_error_message    VARCHAR2 (255) ;
BEGIN
    ...
EXCEPTION
    ...
    WHEN OTHERS THEN
        ROLLBACK;
        v_error_code := SQLCODE ;
        v_error_message := SQLERRM ;
        INSERT INTO errors
        VALUES (v_error_code, v_error_message);
END;
```

ORACLE®

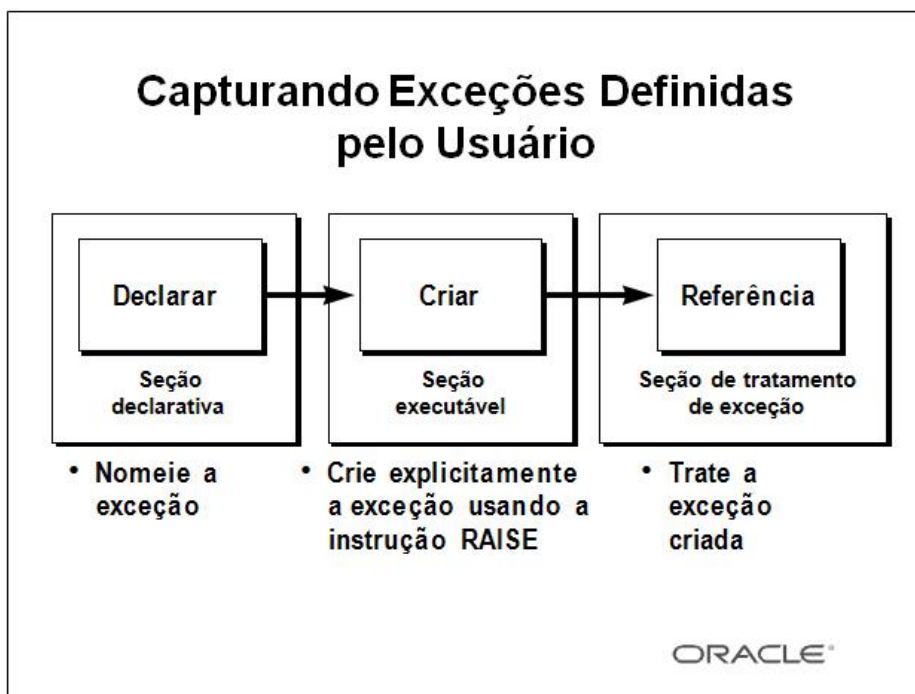
Funções para Captura de Erro

Quando uma exceção é capturada no handler de exceção WHEN OTHERS, você pode usar um conjunto de funções genéricas para identificar esses erros.

O exemplo no slide ilustra os valores de SQLCODE e SQLERRM sendo atribuídos a variáveis e, em seguida, essas variáveis sendo usadas em uma instrução SQL.

Trunque o valor de SQLERRM para um tamanho conhecido antes de tentar gravá-lo para uma variável.

Capturando Exceções Definidas pelo Usuário



Capturando Exceções Definidas pelo Usuário

O PL/SQL permite que você defina suas próprias exceções. As exceções PL/SQL definidas pelo usuário devem ser:

- Declaradas na seção de declaração de um bloco PL/SQL
- Criadas explicitamente com instruções RAISE

Exceção Definida pelo Usuário

Exceção Definida pelo Usuário

Exemplo

```
DECLARE
  e_invalid_product EXCEPTION;
BEGIN
  UPDATE    product
  SET       descrip = '&product description'
  WHERE     prodid = &product_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_product;
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_product THEN
    DBMS_OUTPUT.PUT_LINE('Invalid product number.');
```

1

2

3

ORACLE®

Capturando Exceções Definidas pelo Usuário (continuação)

Você captura uma exceção definida pelo usuário declarando e criando a exceção de forma explícita.

1. Declare o nome da exceção definida pelo usuário dentro da seção declarativa.

Sintaxe

```
exceção      EXCEPTION;
```

onde: *exceção* é o nome da exceção

2. Use a instrução RAISE para criar a exceção explicitamente dentro da seção executável.

Sintaxe

```
RAISE exceção;
```

onde: *exceção* é a exceção declarada anteriormente

3. Faça referência à exceção declarada dentro da rotina de tratamento de exceção correspondente.

Exemplo

Esse bloco atualiza a descrição de um produto. O usuário fornece o número do produto e a nova descrição. Se o usuário incluir um número de produto inexistente, nenhuma linha será atualizada na tabela PRODUCT. Crie uma exceção e imprima uma mensagem para o usuário alertando-os de que foi incluído um número de produto inválido.

Observação: Use a instrução RAISE sozinha em um handler de exceção para criar a mesma exceção de volta no ambiente de chamada.

Ambientes de Chamada



Ambientes de Chamada

SQL*Plus	Exibe a mensagem e o número do erro na tela
Procedure Builder	Exibe a mensagem e o número do erro na tela
Oracle Developer Forms	Acessa a mensagem e o número do erro em um gatilho por meio das funções incluídas ERROR_CODE e ERROR_TEXT
Aplicação pré-compiladora	Acessa número de exceção através da estrutura de dados SQLCA
Um bloco PL/SQL delimitado	Captura a exceção em rotina de tratamento de exceção de bloco delimitado

ORACLE®

Propagando Exceções

No lugar de capturar uma exceção dentro do bloco PL/SQL, propague a exceção para permitir que ela seja tratada pelo ambiente de chamada. Cada ambiente de chamada tem seu próprio modo de exibir e acessar erros.

Propagando Exceções

Propagando Exceções

Os sub-blocos podem tratar uma exceção ou passar a exceção para o bloco delimitado.

```
DECLARE
  ...
  e_no_rows      exception;
  e_integrity    exception;
  PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
      SELECT ...
      UPDATE ...
      IF SQL%NOTFOUND THEN
        RAISE e_no_rows;
      END IF;
    EXCEPTION
      WHEN e_integrity THEN ...
      WHEN e_no_rows THEN ...
    END;
  END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN ...
  WHEN TOO_MANY_ROWS THEN ...
END;
```

ORACLE®

Propagando uma Exceção para um Sub-bloco

Quando um sub-bloco trata uma exceção, ele termina normalmente e o controle retorna ao bloco delimitado imediatamente após a instrução END do sub-bloco.

Entretanto, se o código PL/SQL criar uma exceção e o bloco atual não tiver um handler para essa exceção, a exceção propagará em blocos delimitados sucessivos até localizar um handler. Se nenhum desses blocos tratar a exceção, o resultado será uma exceção não tratável no ambiente de host.

Quando a exceção propaga para um bloco delimitado, as ações executáveis restantes desse bloco são ignoradas.

Uma vantagem desse comportamento é que você pode delimitar instruções que exigem seus próprios tratamentos de erro exclusivos em seu próprio bloco, enquanto deixa o tratamento de exceção mais geral para o bloco delimitado.

Procedimento

RAISE_APPLICATION_ERROR

Sintaxe

```
raise_application_error (error_number,  
                        mensagem[, {TRUE | FALSE}]);
```

- Um procedimento que permite que você emita mensagens de erro definidas pelo usuário a partir de subprogramas armazenados
- Chamado somente a partir de um subprograma armazenado em execução

Procedimento RAISE_APPLICATION_ERROR

Use o procedimento RAISE_APPLICATION_ERROR para comunicar uma exceção predefinida interativamente retornando um código ou uma mensagem de erro não padronizada. Com RAISE_APPLICATION_ERROR, você pode relatar erros para a aplicação e evitar o retorno de exceções não tratáveis.

Na sintaxe:

error_number é um número especificado pelo usuário para a exceção entre -20000 e -20999.

mensagem é a mensagem especificada pelo usuário para a exceção. Trata-se de uma string de caracteres com até 2.048 bytes.

TRUE | FALSE é um parâmetro Booleano opcional (Se TRUE, o erro será colocado na pilha de erros anteriores. Se FALSE, o default, o erro substituirá todos os erros anteriores.)

Exemplo

```
... EXCEPTION  
  WHEN NO_DATA_FOUND THEN  
    RAISE_APPLICATION_ERROR (-20201,  
                             'Manager is not a valid employee.');
```

END;

Procedimento

RAISE_APPLICATION_ERROR

- Usado em dois locais diferentes:
 - Seção executável
 - Seção de exceção
- Retorna condições de erro para o usuário de maneira consistente com outros erros do Oracle Server

Exemplo

```
...  
DELETE FROM emp  
WHERE mgr = v_mgr;  
IF SQL%NOTFOUND THEN  
    RAISE_APPLICATION_ERROR(-20202,'This is not a valid  
manager');  
END IF;  
...
```

Sumário

- Tipos de Exceção:
 - Erro predefinido do Oracle Server
 - Erro não predefinido do Oracle Server
 - Erro definido pelo usuário
- Captura de exceção
- Tratamento de exceção:
 - Capturar a exceção dentro do bloco PL/SQL.
 - Propagar a exceção.

Sumário

O código PL/SQL implementa o tratamento de erros através de exceções e handlers de exceção.

As exceções predefinidas são condições de erro definidas pelo Oracle Server. As exceções não predefinidas são quaisquer outros erros padrão do Oracle Server. As exceções específicas da aplicação ou que você pode prever durante a criação da aplicação são exceções definidas pelo usuário.

Uma vez ocorrido o erro, (ao ser criada uma exceção) o controle é transferido para a parte de tratamento de exceções do bloco PL/SQL. Se uma exceção associada estiver na parte de tratamento de exceção, o código especificado com o handler de exceção será executado. Se não for localizado um handler de exceção associado ao bloco atual e esse bloco estiver aninhado, o controle propagará para o bloco externo, se houver. Se também não for localizado um handler de exceção nos blocos externos, o código PL/SQL relatará um erro.

Master Training

Capítulo 08

Procedimentos de Banco de Dados

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Criar procedimentos de banco de dados
- Criar procedimentos utilizando parâmetros
- Executar procedimentos a partir do SQL*Plus e de blocos PL/SQL
- Remover procedimentos do banco de dados

Procedures

Uma procedure nada mais é do que um bloco PL/SQL nomeado. A grande vantagem sobre um bloco PL/SQL anônimo é que pode ser compilado e armazenado no banco de dados como um objeto de schema. Graças a essa característica as procedures são de fácil manutenção, o código é reutilizável e permitem que trabalhem com módulos de programa.

Uma procedure é, então, um bloco PL/SQL nomeado que pode aceitar argumentos (também chamado de parâmetros) e pode ser chamada por um programa, uma sessão SQL ou uma trigger.

Durante a instalação do banco de dados Oracle um script é executado automaticamente e cria toda a estrutura necessária para que as procedures sejam executadas. Eventualmente esse procedimento automático pode falhar devido a alguma falha física no disco rígido, nesse caso o usuário SYS pode recriar a estrutura através do script SQL DBMSSTDY.SQL.

Para criar uma procedure o usuário precisa ter o privilégio de sistema CREATE PROCEDURE, para criar a procedure em outros schemas o usuário deve ter o privilégio de CREATE ANY PROCEDURE. Este é um ponto muito interessante sobre as procedures, os privilégios para criação de procedures têm que concedidos explicitamente, ou seja, não pode ser adquirido através de roles.

Para executar uma procedure externa é necessário ter o privilégio de EXECUTE. Caso queira alterar a procedure de outro schema deve ter o privilégio de sistema ALTER ANY PROCEDURE.

A sintaxe básica de uma procedure é:

```
CREATE [OR REPLACE] PROCEDURE [schema.]nome_da_procedure
[(parâmetro1 [modo1] tipodedado1,
  parâmetro2 [modo2] tipodedado2,
  ...)]
IS|AS
Bloco PL/SQL
```

Onde:

REPLACE - indica que caso a procedure exista ela será eliminada e substituída pela nova versão criada pelo comando;

BLOCO PL/SQL - inicia com uma cláusula BEGIN e termina com END ou END nome_da_procedure;

NOME_DA_PROCEDURE - indica o nome da procedure;

PARÂMETRO - indica o nome da variável PL/SQL que é passada na chamada da procedure ou o nome da variável que retornará os valores da procedure ou ambos. O que irá conter em parâmetro depende de MODO;

MODO - Indica que o parâmetro é de entrada (IN), saída (OUT) ou ambos (IN OUT). É importante notar que IN é o modo default, ou seja, se não dissermos nada o modo do nosso parâmetro será, automaticamente, IN;

TIPODE DADO - indica o tipo de dado do parâmetro. Pode ser qualquer tipo de dado do SQL ou do PL/SQL. Pode usar referencias como %TYPE, %ROWTYPE ou qualquer tipo de dado escalar ou composto. Atenção: não é possível fazer qualquer restrição ao tamanho do tipo de dado neste ponto.

IS|AS - a sintaxe do comando aceita tanto IS como AS. Por convenção usamos IS na criação de procedures e AS quando estivermos criando pacotes.

BLOCO PL/SQL - indica as ações que serão executadas por aquela procedure.

Vamos ver um exemplo de procedure para ajudar nosso entendimento:

```
CREATE OR REPLACE PROCEDURE aumenta_sal
(p_empno IN emp.empno%TYPE)
IS
BEGIN
UPDATE scott.emp
SET sal = sal * 1.10
WHERE empno = p_empno;
END aumenta_sal;
/
```

Neste exemplo estamos criando uma procedure para aumentar o salário de um funcionário em 10%. A primeira linha define o NOME DA PROCEDURE, que vai ser AUMENTA_SAL.

A linha dois define o parâmetro P_EMPNO no modo IN. Ou seja, vai ser um dado informado na chamada da procedure. Em seguida determinamos que ele será do mesmo tipo e tamanho que a coluna EMPNO da tabela EMP. Isso é feito através da referencia EMP.EMPNO%TYPE.

Podemos verificar o estado de nossa procedure através de uma simples consulta:

```
SELECT object_name, status
FROM user_objects
WHERE object_name LIKE '%AUMENTA%';
```

Agora podemos verificar o funcionamento de nossa procedure:

```
SELECT empno, sal
FROM scott.emp;
```

EMPNO	SAL
7839	5000
7698	2850
7782	2450

```
CALL AUMENTA_SAL(7839);
```

Ou

```
EXECUTE AUMENTA_SAL(7839);
```

```
SELECT empno, sal
FROM scott.emp;
```

EMPNO	SAL
7839	5500
7698	2850
7782	2450

Podemos notar que o salário do funcionário 7839 aumentou em 10%. É interessante notar que neste momento é possível executar a instrução ROLLBACK;

É possível desfazer as alterações porque os dados passados através dos modos OUT e IN OUT são registrados no arquivo de redo log e no segmento de rollback. Isso é perfeito quando trabalhamos com parâmetros pouco extensos, mas pode causar impacto no sistema quando trabalhamos com parâmetros extensos como, por exemplo, um registro ou um VARRAY. Para resolver esse problema podemos usar a opção de NOCOPY. Nossa procedure ficaria assim com a opção NOCOPY.

Criando Procedimentos de Banco de Dados

```
CREATE [OR REPLACE] PROCEDURE nome_procedure
    [(nome_parâmetro [tipo] datatype,
      nome_parâmetro [tipo] datatype,...)]
IS|AS
    [Declaração de variáveis]
BEGIN
    Comandos SQL
    Comandos PL/SQL
[EXCEPTION
    Tratamento de exceções]
END [nome_procedure];
```

Criando Procedimentos de Banco de Dados (continuação)

Criando um procedimento que atualiza os preços de todos os cursos

```
CREATE OR REPLACE PROCEDURE atualiza_preco_cursos
IS
    BEGIN
        UPDATE tcursos
        SET     preco = preco * 1.1;
END;
/
```

Procedimento criado.

Parâmetros

Em subprogramas PL/SQL, declare parâmetros (argumentos) para estabelecer comunicação entre o subprograma e o ambiente que o disparou

Existem três tipos de parâmetros:

- IN [DEFAULT expr]
- OUT [NOCOPY]
- IN OUT [NOCOPY]

Parâmetros devem ser declarados como um tipo de dado escalar (VARCHAR2, NUMBER, etc.) sem precisão, ou como um tipo %TYPE ou %ROWTYPE

Parâmetros IN

```
CREATE OR REPLACE PROCEDURE aumenta_preco
(pId IN tcursos.id%TYPE)
IS
BEGIN
    UPDATE tcursos
    SET    preco = preco * 1.1
    WHERE id = pId;
END;
/
```

Procedimento criado.

Parâmetros OUT

```
CREATE OR REPLACE PROCEDURE consulta_cliente
(pId          IN  tclientes.id%TYPE,
pNome        OUT tclientes.nome%TYPE,
pDt_nascimento OUT tclientes.dt_nascimento%TYPE,
pCidade      OUT tclientes.cidade%TYPE)
IS
BEGIN
    SELECT nome, dt_nascimento, cidade
    INTO   pNome, pDt_nascimento, pCidade
    FROM   tclientes
    WHERE  id = pId;
END consulta_cliente;
/
```

Procedimento criado.

Parâmetros IN OUT

```
CREATE OR REPLACE PROCEDURE formata_numero
(pTelefone IN OUT VARCHAR2)
IS
BEGIN
    pTelefone := SUBSTR(pTelefone, 1, 4) || '-'
                || SUBSTR(pTelefone, 5, 4);
END;
/
```

Procedimento criado.

Parâmetros OUT e IN OUT por referência

- Os parâmetros do tipo "IN" funcionam internamente por referência, enquanto parâmetros "OUT" e "IN OUT" funcionam por cópia
- Declarar esses parâmetros com a opção NOCOPY

```
CREATE OR REPLACE PROCEDURE consulta_cliente
(pid IN  tclientes.id%TYPE,
 pNome OUT NOCOPY tclientes.nome%TYPE,
 pDt_nascimento OUT NOCOPY
 tclientes.dt_nascimento%TYPE,
 pCidade OUT NOCOPY tclientes.cidade%TYPE)
IS
...
```

Utilizando Múltiplos Parâmetros

Podemos utilizar três métodos de passagem de parâmetros:

- Posicional
- Nomeado
- Combinado

Passando Parâmetros - Método Posicional

Necessitamos saber apenas que tipos de valores devem ser passados

```
BEGIN
  cadastra_curso( 100, 'Lógica de Programação.', 'TLP',
                  600, 18, SYSDATE, NULL);
END;
```

Passando Parâmetros - Método Nomeado

nome_parâmetro => valor

```
BEGIN
  cadastra_curso( pNome => 'Lógica de Programação.',
                  pId => 100,
                  pPreco => 600,
                  pCod_trg => 'TLP',
                  pCarga_horaria => 18,
```

```
pDt_criacao => SYSDATE,  
pPre_requisito => NULL );
```

```
END;
```

Passando Parâmetros - Método Combinado

Os primeiros são passados posicionalmente, e os outros são passados de forma nominal

```
BEGIN  
  cadastra_curso( 100,  
                  'Lógica de Programação.',  
                  pPreco => 600,  
                  pCod_trg => 'TLP',  
                  pCarga_horaria => 18,  
                  pDt_criacao => SYSDATE,  
                  pPre_requisito => NULL );  
END;
```

Executando Procedimentos

- A sintaxe para execução de um procedimento depende de qual ambiente ele está sendo chamado
- No SQL*PLUS:

```
SQL> EXEC[UTE] nome_procedimento(valor1, valor2, ...)
```

Executando Procedimentos (continuação)

- Em um bloco PL/SQL:

```
DECLARE  
  vNome          tclientes.nome%TYPE;  
  vDt_nascimento tclientes.dt_nascimento%TYPE;  
  vCidade        tclientes.cidade%TYPE;  
BEGIN  
  consulta_cliente(130, vNome, vDt_nascimento, vCidade);  
END;
```

Removendo Procedimentos de Banco de Dados

```
SQL> DROP PROCEDURE aumenta_preco;
```

Procedimento eliminado.

Master Training

Capítulo 09

Funções de Banco de Dados

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Criar funções de banco de dados
- Criar funções utilizando parâmetros
- Executar funções a partir do SQL*Plus e de blocos PL/SQL
- Remover funções do banco de dados

Criando Funções de Banco de Dados

```
CREATE [OR REPLACE] FUNCTION nome_função
    [(nome_parâmetro [tipo] datatype,
      nome_parâmetro [tipo] datatype,...)]
RETURN datatype
IS|AS
    [Declaração de variáveis]
BEGIN
    Comandos SQL
    Comandos PL/SQL
[EXCEPTION
    Tratamento de exceções]
END [nome_função];
```

Criando uma função que retorna o preço de um curso (NUMBER)

```
CREATE OR REPLACE FUNCTION consulta_preco
(pId IN tcursos.id%TYPE)
RETURN NUMBER IS
    vPreco    tcursos.preco%TYPE := 0;
BEGIN
    SELECT preco
    INTO    vPreco
    FROM    tcursos
    WHERE   id = pId;
    RETURN(vPreco);
END;
/
Função criada.
```

Parâmetros em Funções

O uso de parâmetros em funções é definido com os mesmos parâmetros de procedimentos

O retorno de uma FUNCTION é definido pela cláusula RETURN

Executando Funções a partir do SQL*Plus

```
SQL> VARIABLE gPreco NUMBER
SQL> EXECUTE :gPreco := consulta_preco( 47 );
```

Procedimento PL/SQL concluído com sucesso.

```
SQL> PRINT gPreco
      GPRECO
-----
          726
```

Executando Funções a partir de um bloco PL/SQL

```
FUNCTION existe_cliente( pId IN tclientes.id%TYPE) RETURN
BOOLEAN
IS
    vVar    NUMBER(1);
BEGIN
    SELECT 1
    INTO    vVar
    FROM    tclientes
    WHERE   id = pId;
    RETURN( TRUE );
EXCEPTION
    WHEN others THEN
        RETURN( FALSE );
END;
```

Executando Funções a partir de um bloco PL/SQL (continuação)

```
PROCEDURE cadastra_cliente
(pId      IN tclientes.id%TYPE,
pNome     IN tclientes.nome%TYPE,
pCidade   IN tclientes.cidade%TYPE,
pEstado   IN tclientes.estado%TYPE)
IS
BEGIN
    IF( NOT existe_cliente(pId) )THEN
        INSERT INTO tclientes (id, nome, cidade, estado)
        VALUES ( pId, pNome, pCidade, pEstado);
    END IF;
END;
```


Removendo Funções de Banco de Dados

```
SQL> DROP FUNCTION consulta_preco;
```

Função alterada.

Procedimentos X Funções

- Podem retornar mais de um valor através de parâmetros de saída (OUT)
- Podem conter as sessões declarativa, executável e de tratamento de exceções
- Aceitam valores default
- Podem ser chamadas utilizando a notação posicional ou nomeada
- Aceitam parâmetros com a opção NOCOPY
- PROCEDIMENTO: retornar mais de um valor
- FUNÇÃO: retornar somente um valor

Master Training

Capítulo 10

Desenvolvendo e utilizando Packages

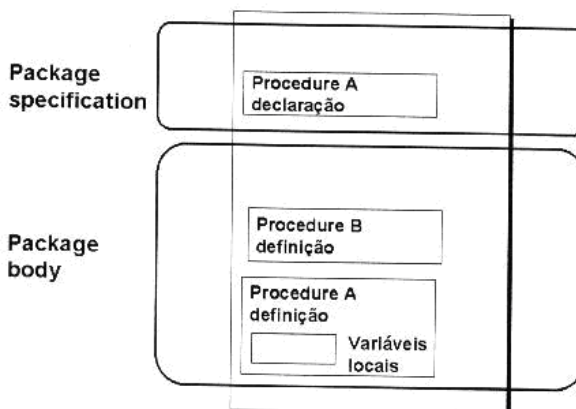
Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Criar Packages PL/SQL de banco de dados para agrupar variáveis, cursores, constantes, exceções, procedimentos e funções
- Criar construções públicas e privadas em uma package
- Invocar uma construção componente de uma package
- Entender as vantagens de uma package
- Identificar dependências que envolvem packages

O Que são Packages?

Uma package possui uma especificação e um corpo (body) armazenados separadamente no banco de dados



Desenvolvendo Packages - Visão Geral

Desenvolva Packages de banco de dados para agrupar objetos PL/SQL (identificadores e rotinas)

Crie uma Package em duas partes:

Package Specification

Package Body

Uma PACKAGE é constituída de duas partes. A primeira é a PACKAGE SPECIFICATION e a segunda é a PACKAGE BODY, onde através delas podemos definir a função do nosso "pacote" dentro do banco de dados. A seguir irei comentar um pouco sobre cada parte.

PACKAGE SPECIFICATION

Tem como função de criar a interface de suas aplicações e definir os tipos de variáveis, cursores, exceções, nomear rotinas, e funções, tudo que se define na parte de SPECIFICATION do seu PACKAGE poderá ser compartilhado com outros scripts ou programas em SQL e PL/SQL.

PACKAGE BODY

Esta parte da sua PACKAGE tem como função executar por completo todas as suas rotinas, cursores e etc, que você definiu na SPECIFICATION, deste modo, todas as funções executadas dentro do BODY não poderão ser compartilhadas com outras aplicações, além

disso no BODY você poderá criar detalhes e declarações que ficarão invisíveis para outras aplicações.

Abaixo está um exemplo em SQL de PACKAGE SPECIFICATION e BODY.

```
SQL > CREATE OR REPLACE PACKAGE nome_package IS
2     TYPE REGTESTE IS RECORD (NOME VARCHAR2(50), SEXO CHAR(1));
3     CURSOR CTESTE (MATRICULA IN NUMBER) RETURN REGTESTE
4     FUNCTION EXCLUIR (MATRICULA IN NUMBER) RETURN BOOLEAN;
5     END nome_package;
6     /
```

```
SQL > CREATE OR REPLACE PACKAGE BODY nome_package IS
2     CURSOR CTESTE (MATRICULA IN NUMBER) RETURN REGTESTE IN
3     SELECT NOME, SEXO FROM Tabela WHERE SEXO=M;
4     BEGIN
5     DELETE FROM Tabela WHERE SEXO=M;
6     END;
7     END nome_package;
8     /
```

Como dito, percebemos que a PACKAGE SPECIFICATION foi compartilhada com a PACKAGE BODY e a própria BODY não poderá ser compartilhada com outras aplicação. Assim permitimos que alguma mudança na programação em SPEFIFICATION poderá afetar outras aplicações que estão utilizando o PACKAGE.

Desenvolvendo Packages - Visão Geral (continuação)

Construções	Descrição
Variável	Identificador que pode ter valores alterados.
Cursor	Identificador associado a um comando SELECT.
Constante	Identificador com um valor fixo.
Exceção	Identificador para uma condição anormal.
Procedimento	Subrotina com argumentos (parâmetros).

Função	Subrotina com argumentos (parâmetros) que retorna obrigatoriamente um valor.
--------	--

Construções Públicas em Packages

- Construções públicas em uma package são aquelas que podem ser invocadas por procedimentos ou funções externas a package
- Devem ser declaradas na Package Specification e definidas no Package Body

Construções Privadas em Packages

- Construções privadas em uma package são aquelas que só podem ser invocadas por procedimentos ou funções da própria package
- Devem ser declaradas e definidas no Package Body

Passos para a criação de uma Package

- 1º. Escreva o comando CREATE PACKAGE, definindo a Package Specification
- 2º. Execute o comando CREATE PACKAGE para criar a Package Specification
- 3º. Escreva o comando CREATE PACKAGE BODY, definindo o Package Body
- 4º. Execute o comando CREATE PACKAGE BODY para criar o Package Body
- 5º. invoque qualquer construção pública (declarada na Package Specification) a partir de um ambiente Oracle

Criando a Package Specification

```
CREATE [OR REPLACE] PACKAGE nome_package
IS|AS
    Declaração de variável
    Declaração de cursor
    Declaração de procedimento
    Declaração de função
END[nome_package];
```

Criando o Package Body

```
CREATE [OR REPLACE] PACKAGE BODY nome_package
IS|AS
    Declaração de variável
    Declaração de cursor
    Declaração de exceção
    Declaração e definição de procedimento
    Declaração e definição de função
END[ nome_package];
```

Definindo um Procedimento de Única Execução

Inicialize a média de desconto (com o resultado da média de percentuais de descontos entre os contratos) na primeira vez em que a package for chamada na sessão

```
CREATE OR REPLACE PACKAGE BODY pck_cursos
IS
...
BEGIN
    SELECT AVG(preco) *.25
    INTO    gDesconto
    FROM    tcursos;
END pck_cursos;
```

Removendo a Package

Remova a Package specification e o respectivo Package Body com o comando DROP PACKAGE

```
DROP PACKAGE nome_package
```

Removendo o Package Body

Remova somente o Package Body com o comando DROP PACKAGE BODY

```
DROP PACKAGE BODY nome_package;
```

Invocando Construções de Packages

- Invoque uma construção de uma package a partir de uma construção da própria package simplesmente invocando seu nome (procedimento ou função) ou identificador (variáveis, cursores, constantes ou exceções), sem prefixos
- Construções externas a package, prefixe o nome da construção com o nome da package

```
SQL> EXECUTE pck_cursos.aumenta_preco( 47 )
```

Benefícios do Uso de Packages

- Modularização do desenvolvimento da Aplicação
- Organização de procedimentos e funções de banco de dados
- Gerenciamento de Segurança
- Possibilita a criação de identificadores globais que podem ser referenciados durante a sessão
- Performance

Gerenciando Dependências em Packages

- Se o Package Body da package referenciada é alterado, e a Package Specification não é alterada, a procedure ou a função que referencia a package não é invalidada
- Se o Package Specification da package referenciada é alterada, então a procedure ou a função que referencia a package é invalidada, assim como o Package Body da package alterada

Re-compilando Packages

```
ALTER PACKAGE nome_package COMPILE;  
ALTER PACKAGE nome_package COMPILE SPECIFICATION;  
ALTER PACKAGE nome_package COMPILE BODY;
```

Não é necessário re-compilar a Package Specification quando re-compilando o Package Body e vice versa

Master Training

Capítulo 11

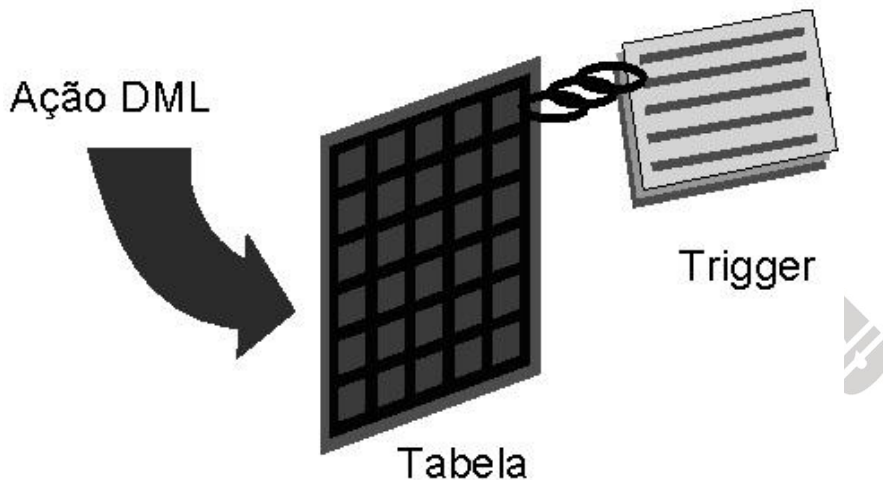
Desenvolvendo e Utilizando Databases Triggers

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Distinguir database triggers de procedures de banco de dados
- Criar database triggers em nível de comando
- Criar database triggers em nível de linha
- Gerenciar database triggers, código fonte e erros de compilação

Database Triggers - Visão Geral



- Triggers (literalmente, gatilhos) não são disparadas explicitamente, como procedimentos ou funções, através de chamadas, mas sim quando ocorrer um evento para o qual elas foram programadas para responder
- Uma trigger de banco (database trigger) fica sempre associada à uma tabela ou visão
- A trigger faz parte da mesma transação do comando que a disparou, não podemos executar os comandos COMMIT, ROLLBACK ou SAVEPOINT
- Algumas utilidades de uma trigger:
 - Manter um nível mais elevado de integridade dos dados, sendo muito complexo para a utilização de constraints
 - Informações de auditoria de uma tabela, armazenando que modificações foram realizadas e também quem as realizou
 - Automaticamente sinalizar para outros programas que ações devem ser realizadas quando são realizadas modificações em uma tabela
- Uma DML trigger é disparada quando um comando INSERT, UPDATE ou DELETE é executado sobre uma tabela do banco de dados
- Pode ser disparada antes ou depois da execução do comando e disparada em nível de comando ou linha
- A combinação destes fatores determina o tipo da trigger
- Uma tabela pode ter qualquer número de triggers definida para ela, incluindo várias de um determinado tipo de comando DML

Elemento	Descrição
Tempo	Quando a Trigger é disparada em relação ao evento que disparou a Trigger. Valores Possíveis: <i>BEFORE</i> (antes do evento) <i>AFTER</i> (depois o evento) <i>INSTEAD OF</i> (substitui o evento): apenas sobre visões
Evento	Que comando(s) de manipulação de dados na tabela causa(m) o disparo da Trigger. Valores Possíveis: <i>INSERT</i> <i>UPDATE [OF coluna]</i> <i>DELETE</i>
Tipo	Quantas vezes o corpo da trigger será executado. Valores Possíveis: <i>Statement</i> (nível de comando – default) <i>FOR EACH ROW</i> (nível de linha)
Corpo	Que ação ou ações a trigger executa. <i>Bloco de comandos PL/SQL</i>

Triggers em Nível de Linha e em Nível de Comando

Trigger em nível de Comando	Trigger em nível de Linha
Executa o corpo da Trigger uma única vez, sempre que um determinado comando de manipulação de dados for emitido para uma determinada tabela.	Executa o corpo da Trigger uma vez para cada linha afetada pelo comando de manipulação que causou o disparo da trigger.

Ordem de disparo das Triggers

- Algoritmo para execução dos comandos:
 - Executar as before statement-level triggers, se existirem.
 - Para cada linha afetada pelo comando:
 - Executar as before row-level triggers, se existirem.
 - Executar o comando
 - Executar as after row-level triggers, se existirem.

- Executar as after statement-level triggers, se existirem

Criando uma Trigger em Nível de Comando

```
CREATE [OR REPLACE] TRIGGER nome_trigger
{BEFORE|AFTER|INSTEAD OF}
evento_trigger1
[OR evento_trigger2
[OR evento_trigger3]]
ON nome_tabela
[DECLARE
    Declaração de variáveis]
BEGIN
    Bloco PL/SQL
[EXCEPTION
    Tratamento de exceções]
END;
```

Tempos de Disparo da Trigger

```
CREATE OR REPLACE TRIGGER bi_valida_horario_trg
BEFORE INSERT
ON tcontratos
BEGIN
    IF (TO_CHAR(SYSDATE, 'DAY') IN ('SABADO', 'DOMINGO') OR
        TO_NUMBER(SYSDATE, 'HH24') NOT BETWEEN 8 AND 18) THEN
        RAISE_APPLICATION_ERROR( -20001, 'O cadastramento de
        contratos é permitido apenas dentro do horário comercial');
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER au_gera_log_alteracoes_trg
AFTER UPDATE OF total
ON tcontratos
DECLARE
BEGIN
    INSERT INTO log( usuario, horario )
    VALUES ( USER, SYSDATE );
END;
```

Criando uma Trigger Combinando Vários Eventos

```
CREATE OR REPLACE TRIGGER biud_valida_horario_trg
BEFORE INSERT OR UPDATE OR DELETE
ON tcontratos
BEGIN
  IF (TO_CHAR(SYSDATE, 'DAY') IN ('SABADO', 'DOMINGO') OR
      TO_NUMBER(SYSDATE, 'HH24') NOT BETWEEN 8 AND 18) THEN
    IF ( INSERTING ) THEN
      RAISE_APPLICATION_ERROR(-20001, 'O cadastramento de
contratos é permitido
                                apenas dentro do horário
comercial.');
```

comercial.');

```
      ELSEIF( DELETING ) THEN
        RAISE_APPLICATION_ERROR(-20002, 'A remoção de contratos
é permitida
                                apenas dentro do horário
comercial.');
```

comercial.');

```
      ELSEIF( UPDATING('TOTAL') ) THEN
        RAISE_APPLICATION_ERROR(-20003, 'Alterações de totais
são permitidas
                                apenas dentro do horário
comercial.');
```

comercial.');

```
      ELSE
        RAISE_APPLICATION_ERROR(-20004, 'Alterações de
cadastros de contratos são
                                permitidas apenas dentro do
horário comercial.');
```

horário comercial.');

```
      END IF;
    END IF;
  END;
```

Triggers em Nível de Linha

```
CREATE [OR REPLACE] TRIGGER nome_trigger
{BEFORE|AFTER|INSTEAD OF}
evento_trigger1
[OR evento_trigger2
[OR evento_trigger3]]
ON nome_tabela
[REFERENCING [OLD AS novo_qualificador1]
              [NEW AS novo_qualificador2]]
FOR EACH ROW
[WHEN condição_de_execução]
```

```
[DECLARE
  Declaração de variáveis]
BEGIN
  Bloco PL/SQL
[EXCEPTION
  Tratamento de exceções]
END;
```

Criando Triggers em Nível de Linha

```
CREATE OR REPLACE TRIGGER aidu_audita_tcontratos
AFTER INSERT OR DELETE OR UPDATE
ON tcontratos
FOR EACH ROW
BEGIN
  IF( DELETING ) THEN
    INSERT INTO log( usuario, horario, informacao )
    VALUES ( USER, SYSDATE, 'Linhas deletadas.' );
  ELSIF( INSERTING ) THEN
    INSERT INTO log( usuario, horario, informacao )
    VALUES ( USER, SYSDATE, 'Linhas inseridas.' );
  ELSIF( UPDATING('TOTAL') ) THEN
    INSERT INTO log( usuario, horario, informacao )
    VALUES ( USER, SYSDATE, 'Atualização do TOTAL do
contrato.' );
  ELSE
    INSERT INTO log( usuario, horario, informacao )
    VALUES ( USER, SYSDATE, 'Atualização das informações do
contrato.' );
  END IF;
END;
```

Valores OLD e NEW

Operação	OLD value	NEW value
INSERT	NULL	Valor inserido
UPDATE	Valor antes do UPDATE	Valor atualizado
DELETE	Valor antes do DELETE	NULL

- Use os qualificadores OLD e NEW em triggers em nível de linha apenas
- Prefixe os valores com dois pontos (:)

Execução Condicional: Cláusula WHEN

```
CREATE OR REPLACE TRIGGER biu_calcula_comissao_trg
BEFORE INSERT OR UPDATE OF total
ON tcontratos
FOR EACH ROW
WHEN(new.total > 5000)
BEGIN
    IF(:new.total <= 10000) THEN
        :new.desconto := :new.total * .3;
    ELSE
        :new.desconto := :new.total * .35;
    END IF;
END;
```

- Dentro da cláusula WHEN não prefixe os valores OLD e NEW com dois pontos

Cláusula Referencing

```
CREATE OR REPLACE TRIGGER biu_calcula_comissao_trg
BEFORE INSERT OR UPDATE OF total
ON tcontratos
REFERENCING OLD AS antigo
              NEW AS novo
FOR EACH ROW
WHEN(novo.total > 5000)
BEGIN
    IF(:novo.total <= 10000) THEN
        :novo.desconto := :novo.total * .3;
    ELSE
        :novo.desconto := :novo.total * .35;
    END IF;
END;
```


Triggers INSTEAD OF

- Além dos tempos de execução BEFORE e AFTER, uma trigger pode ser criada para o tempo INSTEAD OF
- Crie um trigger INSTEAD OF para substituir o processamento padrão de comandos DML sobre uma visão
- O comando DML que dispara a trigger não é executado, apenas as ações definidas no bloco PL/SQL da trigger

Criando Triggers INSTEAD OF

```
CREATE OR REPLACE VIEW vcontratos_pares
AS SELECT id, dt_compra COMPRA, tclientes_id CLIENTE,
desconto, total
FROM tcontratos
WHERE MOD( id,2 ) = 0;

CREATE OR REPLACE TRIGGER iidu_vcontratos_pares_trg
INSTEAD OF INSERT OR DELETE OR UPDATE
ON vcontratos_pares
DECLARE
BEGIN
    INSERT INTO log( usuario, horario, informacao )
    VALUES ( USER, SYSDATE, 'Utilizando comandos DML a partir
da view VCONTRATOS_PARES.' );
END;
```

Mutating Tables

- Existem restrições nas tabelas e colunas que o corpo da trigger pode acessar
- Uma mutating table é uma tabela que está sendo modificada por um comando DML

Mutating Tables – Exemplo 1

- Regra de Mutating Tables: não altere dados em colunas de chaves primárias, chaves estrangeiras ou chaves únicas de tabelas relacionadas àquela na qual a trigger disparada está associada.

- São chamadas Mutating Tables aquelas tabelas que o evento da trigger pode ter necessidade de ler, através de comandos SQL ou através de relações de chave estrangeira. Essas tabelas não devem sofrer alterações a partir da execução do corpo da trigger. Uma tentativa de fazê-lo irá disparar um erro Oracle.

- Essa restrição é válida para todas triggers em nível de linhas, ou em nível de comando disparada como resultado de uma operação DELETE CASCADE.

```
CREATE OR REPLACE TRIGGER au_atualiza_pre_requisitos_trg
AFTER UPDATE OF id
ON tcursos
FOR EACH ROW
BEGIN
    UPDATE tcursos
    SET     pre_requisito = :new.id
    WHERE  pre_requisito = :old.id;
END;
/
```

```
SQL> UPDATE tcursos
      2  SET id = 1000
      3  WHERE id = 1;
```

```
SQL> UPDATE tcursos
      2  SET id = 1000
      3  WHERE id = 1;
SET id = 1000
*
```

```
ERROR at line 2:
ORA-04091: table INTRO2.TCURSOS is mutating,
trigger/function may not see it
ORA-06512: at "INTRO2.AU_ATUALIZA_PRE_REQUISITOS_TRG", line
2
ORA-04088: error during execution of trigger
'INTRO2.AU_ATUALIZA_PRE_REQUISITOS_TRG'
```

Mutating Tables – Exemplo 2

- Regra de Mutating Tables: Não leia informações de tabelas que estejam sendo modificadas.

- São chamadas Mutating Tables aquelas tabelas que estão sofrendo alterações durante a execução da trigger. A tabela à qual a trigger está associada é sempre uma mutating table, assim como qualquer tabela ligada à essa através de chave estrangeira. Essas

características impedem que uma trigger em nível de linha enxergue um conjunto de dados inconsistentes (alterados, mas não confirmados).

- Essa restrição é válida para todas triggers em nível de linhas, ou em nível de comando disparada como resultado de uma operação DELETE CASCADE.

```
CREATE OR REPLACE TRIGGER biu_verifica_total_trg
BEFORE INSERT OR UPDATE OF total
ON tcontratos
FOR EACH ROW
DECLARE
    vMin_total    tcontratos.total%TYPE;
    vMax_total    tcontratos.total%TYPE;
BEGIN
    SELECT MIN(total), MAX(total)
    INTO    vMin_total, vMax_total
    FROM    tcontratos;
    IF( :new.total < vMin_total OR      -- Novo valor deve estar
entre o valor
        :new.total > vMax_total )THEN  -- Mínimo e Máximo
        RAISE_APPLICATION_ERROR(-20505, 'Valor Total do
contrato inválido');
    END IF;
END;
```

```
SQL> UPDATE tcontratos
2 SET      total = 3500
3 WHERE    id = 1000;
```

```
SQL> UPDATE tcontratos
2 SET      total = 3500
3 WHERE    id = 1000;
UPDATE tcontratos
      *
```

```
ERRO na linha 1:
ORA-04091: table INTRO2.TCONTRATOS is mutating,
trigger/function may not see it
ORA-06512: at "INTRO2.BIU_VERIFICA_TOTAL_TRG", line 5
ORA-04088: error during execution of trigger
'INTRO2.BIU_VERIFICA_TOTAL_TRG'
```

Resolvendo o erro de Mutating Tables

```
CREATE OR REPLACE PACKAGE TcontratosDados AS
  TYPE valor_type IS TABLE OF NUMBER(10,2)
    INDEX BY BINARY_INTEGER;
  vValor  valor_type;
END TcontratosDados;

CREATE OR REPLACE TRIGGER biu_verifica_total_trg_c
BEFORE INSERT OR UPDATE OF total
ON tcontratos
BEGIN
  SELECT MIN(total), MAX(total)
  INTO   TcontratosDados.vValor(1),
        TcontratosDados.vValor(2)
  FROM   tcontratos;
END;
```

Resolvendo o erro de Mutating Tables (continuação)

```
CREATE OR REPLACE TRIGGER biu_verifica_total_trg_l
BEFORE INSERT OR UPDATE OF total
ON tcontratos
FOR EACH ROW
DECLARE
BEGIN
  IF( :new.total < TcontratosDados.vValor(1) OR
      :new.total > TcontratosDados.vValor(2) ) THEN
    RAISE_APPLICATION_ERROR(-20001,'Valor Total do contrato
    inválido');
  END IF;
END;
```

Resolvendo o erro de Mutating Tables (continuação)

```
SQL> UPDATE tcontratos
      2 SET      total = 3500
      3 WHERE   id = 1000;

1 linha atualizada.

SQL> UPDATE tcontratos
      2 SET      total = 30000
```

```

3 WHERE id = 1000;
UPDATE tcontratos
    *
ERRO na linha 1:
ORA-20001: Valor Total do contrato inválido
ORA-06512: at "INTRO2.AIU_VERIFICA_TOTAL_TRG_L", line 4
ORA-04088: error during execution of trigger
'INTRO2.AIU_VERIFICA_TOTAL_TRG_

```

Habilitando e Desabilitando Database Triggers

```

ALTER TRIGGER nome_trigger DISABLE;L'

ALTER TRIGGER nome_trigger ENABLE;

ALTER TABLE nome_tabela DISABLE ALL TRIGGERS;

ALTER TABLE nome_tabela ENABLE ALL TRIGGERS;

```

Removendo uma Database Trigger

```

DROP TRIGGER nome_trigger;

```

Gerenciando Database Triggers

Tipo de Informação	Descrição	Método de acesso
Código Fonte	Código fonte da trigger	Consulte a visão USER_TRIGGERS ou a visão USER_SOURCE.
Erros de Compilação	Erros de sintaxe	Consulte a visão USER_ERRORS ou utilize o comando SHOW ERRORS.
Informações de Debug	Informações sobre o conteúdo de variáveis.	Utilizando as procedures da package DBMS_OUTPUT.

Consultando o Código Fonte de Database Triggers

```
SQL> SELECT trigger_body
       2 FROM   user_triggers
       3 WHERE  trigger_name = 'IIDU_TCONTRATOS_PARES_TRG';
TRIGGER_BODY
```

```
-----
DECLARE
BEGIN
  INSERT INTO log( usuario, horario, informacao )
  VALUES ( USER, SYSDATE, 'Utilizando comandos DML a partir
da
               view TCONTRATOS_PARES.' );
END;
```

Master Training

Capítulo 12

Operadores SET

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Verificar as formas de representar os operadores SET
- Realizar a união, intersecção e diferença entre SELECTs relacionados
- Verificar os usos incomuns do comando SELECT

Operadores SET

- Todos os SELECTs envolvidos devem possuir o mesmo número de colunas
- As colunas correspondentes em cada um dos comandos devem ser do mesmo tipo
- A cláusula ORDER BY só se aplica ao resultado geral da união e deve utilizar indicação posicional em vez de expressões
- As demais cláusulas que compõem um comando SELECT são tratadas individualmente nos comandos a que se aplicam
- Todos os operadores (UNION, INTERSECT, MINUS) possuem igual precedência

União – UNION

- A operação de união efetua uma soma de conjuntos eliminando as duplicidades
- UNION elimina do resultado todas as linhas duplicadas. Esta operação pode realizar SORT para garantir a retirada das duplicadas
- UNION ALL apresenta no resultado todas as linhas produzidas no processo de união, independente de serem duplicadas ou não

UNION ...
UNION ALL ...

União – UNION (continuação)

```
SQL> SELECT nome, cidade, estado
2 FROM   tclientes
3 WHERE  estado = 'RS'
4 UNION
5 SELECT nome, cidade, estado
6 FROM   tclientes
7 WHERE  estado LIKE '%R%'
8 ORDER BY 2,3;
```

NOME	CIDADE	ES
Alfredo Fonseca	Porto Alegre	RS
Ana Maria Prado	Rio de Janeiro	RJ
José Batista	Porto Alegre	RS
João Medeiros	Rio de Janeiro	RJ
Nadia Velasques	Rio de Janeiro	RJ
Ramiro Antunes	Rio de Janeiro	RJ

6 linhas selecionadas.

União – UNION (continuação)

```
SQL> SELECT id, dt_compra, desconto, total, 'UNION 1' QUERY
2  FROM    tcontratos
3  WHERE   dt_compra = '10/01/05'
4  UNION   ALL
5  SELECT id, dt_compra, desconto, total, 'UNION 2' QUERY
6  FROM    tcontratos
7  WHERE   desconto IS NOT NULL
8  UNION
9  SELECT id, dt_compra, desconto, total, 'UNION 3' QUERY
10 FROM    tcontratos
11 WHERE   total > 4000
12 ORDER  BY 5;
```

União – UNION (continuação)

Retirando as constantes de identificação dos comando

```
SQL> SELECT id, dt_compra, desconto, total
2  FROM    tcontratos
3  WHERE   dt_compra = '10/01/05'
4  UNION   ALL
5  SELECT id, dt_compra, desconto, total
6  FROM    tcontratos
7  WHERE   desconto IS NOT NULL
8  UNION
9  SELECT id, dt_compra, desconto, total
10 FROM    tcontratos
11 WHERE   total > 4000
12 ORDER  BY 1;
```

Interseção - INTERSECT

- A operação de interseção restringe o conjunto resultante às tuplas presentes em todos os conjuntos participantes da operação

INTERSECT ...

- A cláusula INTERSECT elimina do resultado todas as linhas duplicadas. Esta operação pode realizar sort para garantir a retirada das duplicadas

Interseção – INTERSECT (continuação)

```
SQL> SELECT id, dt_compra, desconto, total
2 FROM tcontratos
3 WHERE desconto IS NOT NULL
4 INTERSECT
5 SELECT id, dt_compra, desconto, total
6 FROM tcontratos
7 WHERE total > 4000
8 ORDER BY 1;
```

ID	DT_COMPR	DESCONTO	TOTAL
1001	04/01/05	225	4500
1008	06/01/05	700	6218
1016	09/01/05	960	15960

Diferença - MINUS

- A operação de diferença efetua uma subtração de conjuntos eliminando as duplicidades
- As regras já apresentadas na UNION e INTERSECT devem ser obedecidas na diferença

MINUS ...

Diferença – MINUS (continuação)

```
SQL> SELECT id, dt_compra, desconto, total
2 FROM tcontratos
3 WHERE desconto IS NOT NULL
4 MINUS
5 SELECT id, dt_compra, desconto, total
6 FROM tcontratos
7 WHERE total > 4000
8 ORDER BY 1;
```

ID	DT_COMPR	DESCONTO	TOTAL
----	----------	----------	-------

1000	03/01/05	360	3600
1003	04/01/05	985	1995
1005	05/01/05	191	3191
1007	06/01/05	50	1000
1011	07/01/05	1500	3000
1012	08/01/05	60	600
1013	08/01/05	260	2600
1019	11/01/05	60	600

8 linhas selecionadas.

Usos Incomuns do Comando SELECT

•Em INSERTs:

```
SQL> INSERT INTO tclientes (id, nome, dt_nascimento)
2  VALUES ((SELECT NVL(MAX(id),0) + 1
3           FROM   tclientes), 'Vitor Augusto', '30/05/71');
1 linha criada.
```

•Na cláusula SELECT:

```
SQL> SELECT (SELECT MAX(total)
2           FROM   tcontratos) MAIOR,
3           (SELECT MIN(total)
4           FROM   tcontratos) MENOR
5  FROM   dual;
```

MAIOR	MENOR
15960	600

Master Training

Capítulo 13

Aperfeiçoando a Cláusula GROUP BY

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Definir a utilização das funções ROLLUP e CUBE a partir da cláusula GROUP BY
- Entender a montagem de subtotais a partir de SELECTs agrupados
- Utilizar funções analíticas

Aperfeiçoando o GROUP BY

- As operações de GROUP BY e HAVING podem se utilizar de operações chamadas de ROLLUP e CUBE
- São subtotais e tabulações sobre as dimensões
- Úteis para sistemas de DataWare House (Informações gerenciais)

GROUP BY ROLLUP(column, [column])

GROUP BY CUBE(column, [column])

A Função ROLLUP

- Retorna uma única linha sumarizada para os subgrupos.
- Tem grande utilidade na construção de subtotais

```
SQL> SELECT tclientes_id, dt_compra, SUM(desconto),  
2          SUM(total)  
3 FROM      tcontratos  
4 GROUP BY  ROLLUP(tclientes_id, dt_compra);
```

A Função CUBE

- Executa as funções de agregação para subgrupos compostos dos valores de todas as possíveis combinações das expressões (informadas para CUBE)
- Retorna uma única linha sumarizada para cada subgrupo
- Ao final executa totais individuais.

```
SQL> SELECT tclientes_id, dt_compra, SUM(desconto),  
2          SUM(total)  
3 FROM      tcontratos  
4 GROUP BY  CUBE(tclientes_id, dt_compra);
```

Identificando Sub-Grupos

- Algumas funções podem nos ajudar a distinguir um valor NULL que representa um subgrupo (de uma das agregações produzidas pelo ROLLUP e CUBE), de um valor NULL real.

- GROUPING: Retorna 1 se o valor da expressão representar um subgrupo.
- GROUPING_ID: Retorna valores seqüências dos diversos resultados de GROUPING na ordem em que ocorrem.

Usando a função GROUPING

```
SQL> SELECT GROUPING(tclientes_id),tclientes_id,sum(total)
2 FROM tcontratos
3 GROUP BY ROLLUP(tclientes_id);
```

GROUPING(TCLIENTES_ID)	TCLIENTES_ID	SUM(TOTAL)
0	100	13318
0	110	3112
0	120	1800
0	130	5815
0	140	3000
0	150	20160
0	160	1995
0	170	2556
0	180	13000
0	190	3191
0	200	6200
1		74147

12 linhas selecionadas.

Função GROUPING com CASE

```
SQL> SELECT CASE WHEN grouping(tclientes_id)=0 THEN
2 TO_CHAR(tclientes_id)
3 ELSE 'Total Geral :' END Cliente,
4 sum(total)
5 FROM tcontratos
6 GROUP BY ROLLUP(tclientes_id);
```

CLIENTE	SUM(TOTAL)
100	13318
110	3112
120	1800
130	5815
140	3000
150	20160

160	1995
170	2556
180	13000
190	3191
200	6200
Total Geral :	74147
12 linhas selecionadas.	

Usando a Função GROUPING_ID()

- A função GROUPING_ID() é utilizada para a identificação de grupos e sub-grupos

```
SQL> SELECT dt_compra, GROUPING_ID(dt_compra) GDT,  
2          tclientes_id, GROUPING_ID(tclientes_id) GCL,  
3          SUM(TOTAL)  
4 FROM tcontratos  
5 GROUP BY ROLLUP(dt_compra, tclientes_id);
```

GROUPING_ID() com CASE

- Uso da função CASE para identificarmos os sub-totais de data e o total geral

```
SQL> SELECT dt_compra, tclientes_id,  
2 CASE WHEN GROUPING_ID(tclientes_id)=1 AND  
3       GROUPING_ID(dt_compra)=0 THEN 'Total do Dia : '  
4       WHEN GROUPING_ID(tclientes_id)=1 AND  
5       GROUPING_ID(dt_compra)=1 THEN 'Total Geral  : '  
END,  
6 SUM(total)  
7 FROM tcontratos  
8 GROUP BY ROLLUP(dt_compra, tclientes_id);
```

Usando a Cláusula GROUPING SETS

Retorna somente os subtotais

```
SQL> SELECT tclientes_id, dt_compra, SUM(total)  
2 FROM tcontratos  
3 GROUP BY GROUPING SETS (tclientes_id, dt_compra);
```

Master Training

Funções Analíticas

- As funções analíticas são valiosas para todo tipo de processamento, desde suporte à decisão até a geração de relatórios comuns
- Melhoram o desempenho de consultas ao banco de dados e a produtividade dos desenvolvedores

Função RANK()

As funções da família Ranking calculam o rank (posição, ordem) de uma linha em relação às demais linhas, dentro de um conjunto de dados. A função RANK() produz uma classificação ordenada de linhas começando com a posição 1.

```
SQL> SELECT tclientes_id, SUM(total),  
2          RANK() OVER (ORDER BY SUM(total) DESC) "RANK() "  
3 FROM tcontratos  
4 GROUP BY (dt_compra, tclientes_id);
```

RANK() com Particionamento

- A cláusula opcional PARTITION BY é usada para definir onde o cálculo da posição é reinicializado

```
SQL> SELECT dt_compra, tclientes_id, SUM(total),  
2          RANK() OVER (PARTITION BY DT_COMPRA ORDER BY  
3          SUM(total) DESC) "RANK() "  
4 FROM tcontratos  
5 GROUP BY (dt_compra, tclientes_id);
```

Função DENSE_RANK()

A função DENSE_RANK não deixa "abertura" na sequência numérica do ranking.

```
SQL> SELECT id, total,  
2          RANK() OVER (ORDER BY total DESC) "RANK() ",  
3          DENSE_RANK() OVER (ORDER BY total DESC)  
4          "DENSE_RANK() "  
5 FROM tcontratos
```

```
6 GROUP BY id, total;
```

Função **RATIO_TO_REPORT**

A função **RATIO_TO_REPORT** calcula a proporção de um valor em relação à agregação de um conjunto de valores. Por exemplo, comparar o total de contratos de um cliente com o total geral de contratos

```
SQL> SELECT tclientes_id, SUM(total) "Total do Cliente",  
2          ROUND(RATIO_TO_REPORT(SUM(total)) OVER() *100,2)  
3          "% do Total"  
4 FROM tcontratos  
5 GROUP BY tclientes_id;
```

RATIO_TO_REPORT com particionamento

```
SQL> SELECT tclientes_id, dt_compra, SUM(total) "Total do Dia  
do Cliente",  
2          ROUND(RATIO_TO_REPORT(SUM(total))  
OVER(PARTITION BY  
3          tclientes_id)*100,2) "% do Dia"  
4 FROM tcontratos  
5 GROUP BY tclientes_id, dt_compra  
6 ORDER BY tclientes_id,4 DESC;
```

LAG() e LEAD()

- As funções **LAG** e **LEAD** permitem comparações entre duas linhas de um mesmo conjunto de dados
- Por exemplo, analisar mudanças nas vendas mensais do ano corrente comparadas com as de anos anteriores ou analisar a variação entre orçamento e custos reais

```
SQL> SELECT dt_compra, SUM(total) total_dia,  
2          LAG (SUM(total),1) OVER (ORDER BY dt_compra)  
anterior,  
3          LEAD (SUM(total),1) OVER (ORDER BY dt_compra)  
posterior  
4 FROM tcontratos  
5 GROUP BY dt_compra  
6 ORDER BY dt_compra;
```

Master Training

Capítulo 14

Database Link

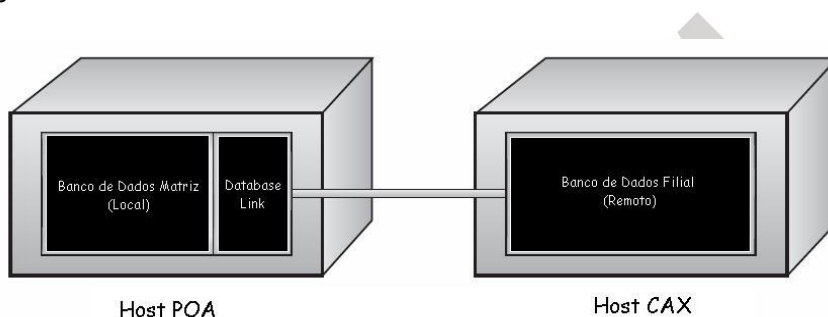
Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Definir um database link
- Utilizar um database link

Acesso Remoto de Dados

- No Oracle o acesso remoto de dados, como por exemplo, consultas e atualizações em bases remotas estão disponíveis através dos chamados database links
- Os database links possibilitam que os usuários tratem um grupo de bases distribuídas como se fosse um único banco de dados integrado



Entendendo Database Link (continuação)

O database link especifica as seguintes informações da conexão:

- O protocolo de comunicação a ser utilizado durante a conexão (TCP/IP)
- A máquina na qual o banco de dados remoto se encontra
- O nome do banco de dados remoto
- O nome de um usuário válido no banco de dados remoto
- A senha do usuário

Utilizando Database Links

- O Oracle realiza a conexão ao banco de dados remoto utilizando o nome do usuário e senha definidos na criação do link
- Com o database link e os privilégios necessários é possível utilizar comandos SELECT, INSERT, UPDATE, DELETE sobre os objetos desejados do banco de dados remoto

```
SELECT *  
FROM tclientes@nome_database_link;
```

Criando Database Link

```
CREATE [PUBLIC] DATABASE LINK REMOTE_CONNECT  
CONNECT TO {CURRENT_USER | USERNAME  
IDENTIFIED BY PASSWORD [AUTHENTICATION CLAUSE]}  
USING 'CONNECT STRING';
```

- Para criar um database link, o usuário deve possuir o privilégio de sistema CREATE DATABASE LINK
- A conta de usuário ao qual será feita a conexão ao banco de dados remoto deve possuir o privilégio de sistema CREATE SESSION

Database Links Public x Private

- Um database link PUBLIC está disponível para todos os usuários do banco de dados
- Um database link PRIVATE está disponível apenas para o usuário que o criou, não sendo possível garantir a outro usuário o acesso através do database link
- Para criar um database link public, o usuário deve possuir o privilégio de sistema CREATE PUBLIC DATABASE LINK.

Login Default x Explícito

- CONNECT TO... IDENTIFIED BY... – login Explícito
- CONNECT TO CURRENT_USER – login default

Quando essa opção é utilizada o Oracle tenta abrir uma conexão ao banco de dados remoto com o mesmo nome e senha do usuário definidos no banco de dados local

Connect String

- O Oracle Net utiliza o service names para identificar as conexões remotas
- Os detalhes das conexões a esses service names estão dentro do arquivo de configuração tnsnames.ora

- O arquivo possui informações para determinar qual o protocolo, nome da máquina(host), e nome do banco de dados, que devem ser utilizados durante a conexão

Exemplo de Criação de Database Link

```
CONNECT gerente/gerente@matriz
```

```
SQL> CREATE DATABASE LINK filial
  2  CONNECT TO diretor IDENTIFIED BY diretor
  3  USING 'FILIAL';
Vínculo de banco de dados criado.
```

```
SELECT *
FROM vendas@filial;
```

```
SELECT *
FROM funcionarios@filial;
```

```
SQL> DROP DATABASE LINK filial;
Vínculo de banco de dados eliminado.
```

Visões do dicionário de dados

```
SQL> SELECT db_link, username, host, created
  2  FROM user_db_links
  3  ORDER BY DB_LINK;
```

DB_LINK	USERNAME	HOST	CREATED
FILIAL	DIRETOR	filial	18/01/06

Master Training

Master Training

Capítulo 15

Comandos DML Avançado

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Realizar inserções em múltiplas tabelas
- Utilizar o comando MERGE

Multi Table Insert

- Permite que a expressão INSERT INTO ... SELECT possa carregar dados em múltiplas tabelas
- Os tipos de multi table inserts são os seguintes:
- INSERT Incondicional
- ALL INSERT Condicional
- FIRST INSERT Condicional

INSERT Incondicional

- Insere todas as linhas em todas as tabelas sem nenhuma condição especial
- Todas as inserções executarão simultaneamente

```
INSERT ALL
  INTO produtos_total (id,hoje,total)
  INTO produtos_historico (id,hoje,total)
  SELECT id, trunc(sysdate) hoje,
sum(preco_unitario*quantidade) total
  FROM pedidos, item_pedidos
 WHERE pedidos.id = item_pedidos.pedidos_id
 AND trunc(data_pedido) = trunc(sysdate)
 GROUP BY id;
```

ALL INSERT Condicional

- Verifica cada cláusula WHEN para verificar em qual tabela a linha vai ser inserida
- Todas as cláusulas WHEN são verificadas
- Algumas linhas podem ser inseridas em mais de um destino enquanto outras podem até não ser inseridas
- A cláusula ELSE pode ser utilizada
- Podem ser utilizadas várias cláusulas INTO com uma condição WHEN

ALL INSERT Condicional (continuação)

```
INSERT ALL
  WHEN tipo_pedido='online' THEN
    INTO pedidos_web
    VALUES (id,data_pedido,total_pedido)
  WHEN loja is not null THEN
    INTO pedidos_loja
    VALUES (id,data_pedido,total_pedido)
  SELECT id,data_pedido,total_pedido,loja,tipo_pedido
  FROM pedidos;
```

FIRST INSERT Condicional

- Verifica cada cláusula WHEN na ordem em que aparece na expressão
- Executa somente a primeira cláusula INTO que é verdadeira
- A cláusula ELSE pode ser utilizada

FIRST INSERT Condicional (continuação)

```
INSERT FIRST
  WHEN total_pedido > 25000 THEN
    INTO vendas_5estrelas VALUES(id)
  WHEN total_pedido > 10000 THEN
    INTO vendas_4estrelas VALUES(id)
  WHEN total_pedido > 5000 THEN
    INTO vendas_3estrelas VALUES(id)
  WHEN total_pedido > 3000 THEN
    INTO vendas_2estrelas VALUES(id)
  ELSE
    INTO vendas_1estrela VALUES(id)
  SELECT id, total_pedido
  FROM pedidos;
```

Comando MERGE

- O comando tem a função de obter linhas de uma determinada tabela para atualizar ou incluir linhas em outra tabela

- INTO: Determina a tabela destino onde faremos a inclusão ou atualização
- USING: Determina o dado (origem) que será incluído ou atualizado
- WHEN MATCHED / NOT MATCHED: Determina a condição a ser avaliada para inclusão ou atualização. Quando a condição for verdadeira ocorrerá a atualização, caso contrário será realizada a inclusão

Comando MERGE (continuação)

Recuperar dados da tabela TCLIENTES e atualizar múltiplas vezes a tabela TCONTRATOS para os contratos do dia '11/01/05'

```
SQL> MERGE INTO tcontratos tcn
2 USING (SELECT id CLIENTE
3        FROM tclientes
4        WHERE estado = 'RS')
5 ON      (tcn.tclientes_id = cliente AND dt_compra =
'11/01/05')
6        WHEN MATCHED THEN
7            UPDATE SET desconto = total * .6
8        WHEN NOT MATCHED THEN
9            INSERT(tcn.id, tcn.dt_compra,
tcn.tclientes_id,
10                  tcn.desconto, tcn.total)
11            VALUES ((SELECT MAX(id)+1 FROM
tcontratos),SYSDATE,
12                  cliente, 0, 0);
2 linhas intercaladas.
```

Capítulo 16

SQL Dinâmico em PL/SQL

Objetivos

Depois de completar esta lição, você poderá fazer o seguinte:

- Entender a utilização de SQLs Dinâmicos
- Criar construções dinâmicas com SQL
- Utilizar o comando EXECUTE IMMEDIATE
- Utilizar os comandos OPEN-FOR, FETCH e CLOSE

Conceito

- Um SQL Dinâmico é um comando SQL ou um bloco PL/SQL válido, codificado dentro de uma string (populada em tempo de execução)
 - Disponível a partir da versão 8i do banco de dados Oracle
 - Temos aplicações que necessitam processar comandos SQL que só se completam a tempo de execução
 - Possibilidade de execução de comandos DDL ou DCL dentro de blocos PL/SQL

Usando SQL Dinâmico

```
DELETE FROM titens  
WHERE total = vTotal AND tclientes_id = vTcl_id;
```

```
DELETE FROM titens  
WHERE total = vValor AND tclientes_id = vCliente;
```

- Comandos de SQL dinâmico são armazenados em strings construídas pelo programa em tempo de execução
 - Estas strings devem conter um comando SQL ou um bloco de PL/SQL válidos

Usando SQL Dinâmico (continuação)

- Para processarmos comandos de SQL, tais como INSERT, UPDATE, DELETE, ou bloco PL/SQL usaremos o comando:
 - Execute Immediate
- Para processarmos SQL SELECT, usaremos os comandos:
 - Open-For
 - Fetch
 - Close

Porque utilizar SQL Dinâmico?

- Executar comandos SQL de definição de dados (como o CREATE), comandos de controle de dados (como o GRANT), ou comandos de controle de sessão (como o ALTER SESSION)

- Por exemplo, você deseja passar o nome de um usuário (schema) como um parâmetro para um procedimento, você poderá utilizar diferentes condições na cláusula WHERE do seu comando SELECT.

O Comando EXECUTE IMMEDIATE

```
EXECUTE IMMEDIATE 'SQL string'  
[INTO {variável[,variável]...| record}]  
[USING [IN | OUT | IN OUT] bind_argument ];
```

- SQL String é uma string que contém aquilo que se deseja executar

- Cláusula INTO , a especificação de variáveis é opcional e indica uma ou mais variáveis para as quais valores selecionados serão atribuídos

- Cláusula USING , a seção bind_argument (parâmetros) é opcional e designa um valor repassado para bind variables na SQL string.

Utilizando o Comando EXECUTE IMMEDIATE – Exemplo 1

Inserir os valores na tabela conforme um parâmetro informado

```
CREATE OR REPLACE PROCEDURE insere_grandes_contratos  
(pTotal IN tcontratos.total%TYPE)  
IS  
BEGIN  
    EXECUTE IMMEDIATE ' INSERT INTO grandes_contratos ' ||  
                      ' SELECT id, dt_compra, total ' ||  
                      ' FROM tcontratos ' ||  
                      ' WHERE total > :1 '  
  
    USING pTotal;  
    COMMIT;  
END;  
  
SQL> EXEC insere_grandes_contratos(4000)
```

Utilizando o Comando EXECUTE IMMEDIATE – Exemplo 2

Retorne a quantidade de linhas de uma tabela de um esquema em determinada condição

```
CREATE OR REPLACE FUNCTION verifica_linhas
(pNomeTabela IN VARCHAR2,
 pWhere      IN VARCHAR2 := NULL,
 pSchema     IN VARCHAR2 := NULL)
RETURN INTEGER
IS
    vQtdLinhas INTEGER;
BEGIN
    EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM ' ||
                      NVL(pSchema,USER) || '.' ||
pNomeTabela ||
                      ' WHERE ' || NVL (pWhere,'1=1')
    INTO vQtdLinhas;
    RETURN vQtdLinhas;
END;
```

Utilizando o Comando EXECUTE IMMEDIATE – Exemplo 3

Criar uma tabela dentro de um procedimento do sistema

```
CREATE OR REPLACE PROCEDURE criar_tabela
(pNome IN VARCHAR2, pUsuario IN VARCHAR2)
IS
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE ' || pNome ||
                      ' (coluna1 VARCHAR2(40) PRIMARY KEY, '
    ||
                      '   coluna2 DATE NOT NULL, ' ||
                      '   coluna3 NUMBER(10) NOT NULL)';
    EXECUTE IMMEDIATE 'GRANT SELECT ON ' || pNome ||
                      ' TO ' || pUsuario;
END;
```

Algumas considerações

- EXECUTE IMMEDIATE não realizará automaticamente o COMMIT de uma transação DML anterior
- Consultas que retornem mais de uma linha não são suportadas como valor de retorno
- Para executar comandos DDL através de SQL dinâmico, o usuário deve ter recebido os privilégios de sistema de forma explícita, não pode ser através de roles

Os Comandos OPEN-FOR, FETCH e CLOSE

- Para efetuarmos o processamento de uma consulta, que retorne diversas linhas, de forma dinâmica, deveremos utilizar três comandos:

```
OPEN <variavel cursor> FOR <string dinamica>
    [USING <bind [,bind ...]>]

FETCH <variável cursor> INTO <registro|variavel>

CLOSE <variável cursor>
```

Utilizando Seleções Dinâmicas

```
CREATE OR REPLACE PROCEDURE consulta_generica
(pColunas IN VARCHAR2, pTabelas IN VARCHAR2, pCondicoes IN
VARCHAR2)
IS
    TYPE refCursor IS REF CURSOR;
    cCursor1      refCursor;

    vRetorno VARCHAR2(1000);

BEGIN
    OPEN cCursor1 FOR ' SELECT '||pColunas||
                      ' FROM '||pTabelas||
                      ' WHERE '||pCondicoes;
    DBMS_OUTPUT.PUT_LINE('<< Retorno Obtido >>');
    LOOP
        FETCH cCursor1 INTO vRetorno;
        EXIT WHEN cCursor1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(vRetorno);
    END LOOP;
    CLOSE cCursor1;
END;
```

Passando Argumentos NULL

```
BEGIN
    EXECUTE IMMEDIATE 'UPDATE tcontratos SET desconto =
100 '||
                                'WHERE desconto IS :DCTO'
        USING NULL;
    END;
    /
    USING NULL;
    *
ERRO na linha 5:
ORA-06550: line 5, column 11:
PLS-00457: expressions have to be of SQL types
ORA-06550: line 2, column 5:
PL/SQL: Statement ignored
```

```
DECLARE
    vValor VARCHAR2(50) := 'NULL';
BEGIN
    EXECUTE IMMEDIATE 'UPDATE tcontratos SET desconto =
100 '||
                                'WHERE desconto IS '||vValor;
END;
```