Introdução ao Oracle: SQL e PL/SQL

Guia do Estudante • Volume 2

41010BP13
Produção 1.3
Fevereiro de 2000
M08945-BP

ORACLE®

Autores

Neena Kochhar
Ellen Gravina
Priya Nathan

Colaboradores Técnicos e Revisores

Claire Bennet
Christa Miethaner
Tony Hickman
Sherin Nassa
Nancy Greenberg
Hazel Russell
Kenneth Goetz
Piet van Zon
Ulrike Dietrich
Helen Robertson
Thomas Nguyen
Lisa Jansson
Kuljit Jassar

Editor

Jerry Brosnan

Education Products, Oracle Corporation, 500 Oracle Parkway, Box SB-6, Redwood Shores, CA 94065.
Distribuidor no Brasil: Oracle do Brasil Sistemas Ltda.
Rua José Guerra, 127, São Paulo, SP - 04719-030 -
Brasil
CGC: 59.456.277/0001-76.

Sumário

14

Controlando o Acesso do Usuário

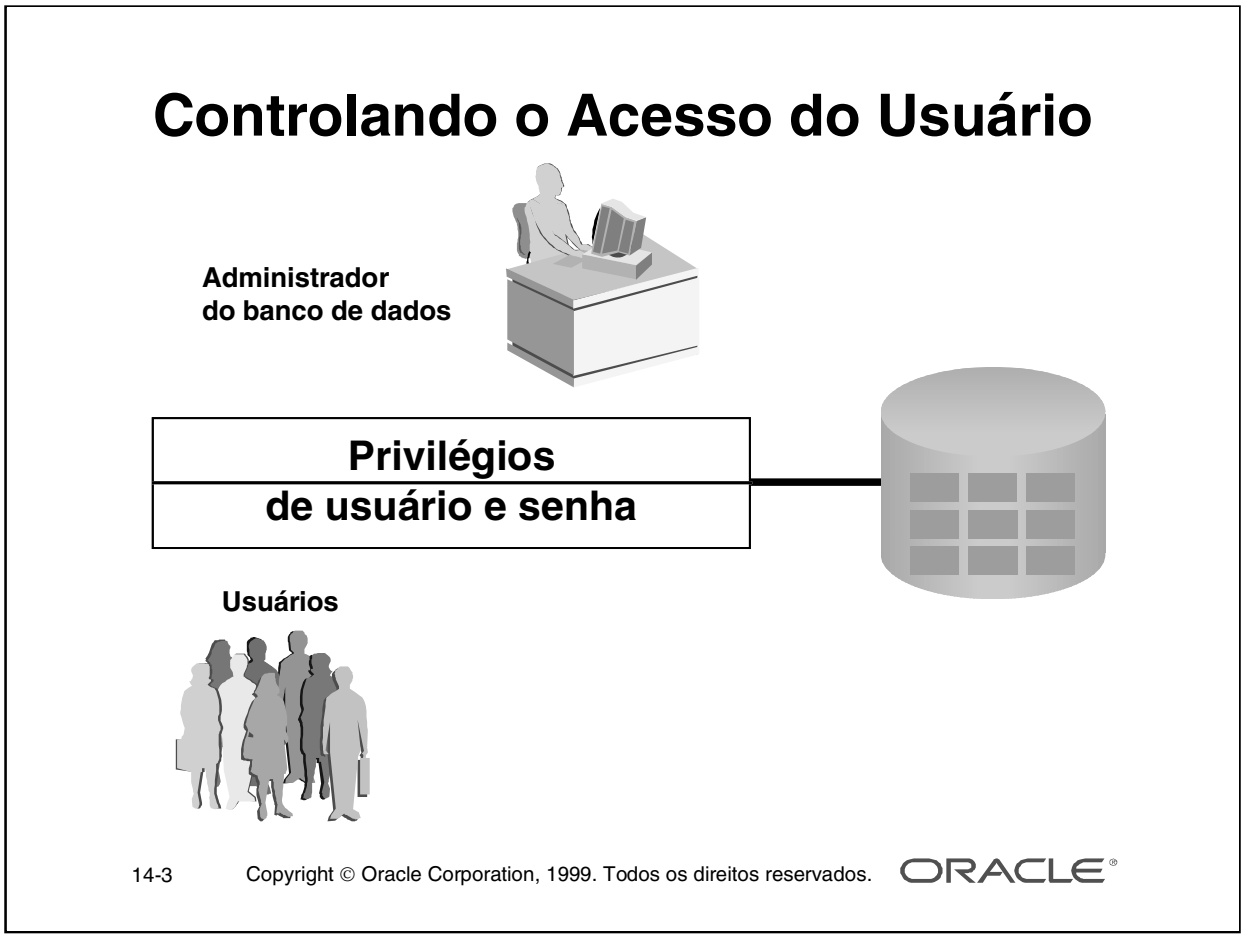
Objetivos

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

- **Criar usuários**
- **Criar funções para facilitar a configuração e manutenção do modelo de segurança**
- **Usar as instruções GRANT e REVOKE para conceder e revogar os privilégios de objeto**

Objetivo da Lição

Nesta lição, você aprenderá como controlar o acesso de banco de dados a objetos específicos e adicionar novos usuários com diferentes níveis de privilégio de acesso.



Controlando o Acesso do Usuário

Em um ambiente de vários usuários, você deseja manter a segurança de acesso e de uso do banco de dados. Com a segurança de banco de dados do Oracle Server, você pode:

- Controlar o acesso ao banco de dados
- Conceder acesso a objetos específicos no banco de dados
- Confirmar *privilégios* concedidos e recebidos com o dicionário de dados Oracle
- Criar sinônimos para os objetos de banco de dados

A segurança de banco de dados pode ser classificada em duas categorias: segurança de sistema e segurança de banco de dados. A segurança de sistema cobre o acesso e o uso do banco de dados no nível de sistema como, por exemplo, nome de usuário e senha, espaço em disco alocado aos usuários e operações do sistema permitidas pelo usuário. A segurança de banco de dados cobre o acesso e o uso dos objetos de banco de dados e as ações que esses usuários possam ter sobre os objetos.

Privilégios

- **Segurança de banco de dados:**
  - **Segurança de sistema**
  - **Segurança de dados**
- **Privilégios de sistema: Obter acesso ao banco de dados**
- **Privilégios de objeto: Manipular o conteúdo dos objetos de banco de dados**
- **Esquema: Coleção de objetos como, por exemplo, tabelas, views e seqüências**

Privilégios

Os privilégios constituem o direito de executar instruções SQL particulares. O administrador de banco de dados é um usuário de alto nível com a habilidade de conceder aos usuários acesso ao banco de dados e aos objetos. Os usuários requerem *privilégios de sistema* para obter acesso aos *privilégios de objeto* e de banco de dados para manipular o conteúdo dos objetos no banco de dados. Também pode ser fornecido aos usuários o privilégio de conceder privilégios adicionais a outros usuários ou a *funções*, que são grupos nomeados de privilégios relacionados.

Esquema

Um esquema é uma coleção de objetos como, por exemplo, tabelas, views e seqüências. O esquema pertence a um usuário de banco de dados e tem o mesmo nome do usuário.

Para obter mais informações, consulte o *Oracle Server Application Developer's Guide,* Release 8, seção "Establishing a Security Policy" e *Oracle Server Concepts Manual*, Release 8, tópico "Database Security".

Privilégios de Sistema

- **Mais de 80 privilégios estão disponíveis.**
- **O DBA possui privilégios de sistema de alto nível:**
  - **Criar novos usuários**
  - **Remover usuários**
  - **Remover tabelas**
  - **Fazer back up de tabelas**

Privilégios de Sistema

Mais de 80 privilégios estão disponíveis para usuários e funções. Os privilégios de sistema são tipicamente fornecidos pelo administrador do banco de dados.

Privilégios de DBA Típicos

Privilégio de Sistema	Operações Autorizadas
CREATE USER	Permite que o cedente crie outros usuários Oracle (um privilégio requerido para uma função DBA)
DROP USER	Elimina um outro usuário
DROP ANY TABLE	Elimina uma tabela em qualquer esquema
BACKUP ANY TABLE	Faz back up de tabela nos esquemas com o utilitário de exportação

Criando Usuários

O DBA cria usuários usando a instrução CREATE USER.

```
CREATE     USER usuário
IDENTIFIED BY   senha;
```

```
SQL> CREATE     USER  scott
  2  IDENTIFIED  BY   tiger;
User created.
```


Criando um Usuário

O DBA cria o usuário executando a instrução CREATE USER. O usuário não possui privilégios nesse ponto. O DBA pode então conceder um número de privilégios àquele usuário. Esses privilégios determinam o que o usuário pode fazer no nível de banco de dados.

O slide fornece a sintaxe resumida para criar um usuário.

Na sintaxe:

usuário	é o nome do usuário a ser criado
senha	especifica que o usuário deve estabelecer login com essa senha

Para obter mais informações, consulte o *Oracle Server SQL Reference,* Release 8, "GRANT" (System Privileges and Roles) e "CREATE USER".

Privilégios de Sistema de Usuário

- **Quando o usuário for criado, o DBA poderá conceder privilégios de sistema específicos para ele.**

```
GRANT privilégio [, privilégio...]
TO usuário [, usuário...];
```

- **Um desenvolvedor de aplicação pode ter os seguintes privilégios de sistema:**
  - **CREATE SESSION**
  - **CREATE TABLE**
  - **CREATE SEQUENCE**
  - **CREATE VIEW**
  - **CREATE PROCEDURE**

Privilégios Típicos de Usuário

Agora que o DBA criou um usuário, o DBA poder atribuir privilégios àquele usuário.

Privilégio de Sistema	Operações Autorizadas
CREATE SESSION	Conectar-se ao banco de dados
CREATE TABLE	Criar tabelas no esquema do usuário
CREATE SEQUENCE	Criar uma seqüência no esquema do usuário
CREATE VIEW	Criar uma view no esquema do usuário
CREATE PROCEDURE	Criar um função, pacote ou procedimento armazenado no esquema do usuário

Na sintaxe:

privilégio é o privilégio de sistema a ser concedido

usuário é o nome do usuário

Concedendo Privilégios de Sistema

O DBA pode conceder privilégios de sistema específicos a um usuário.

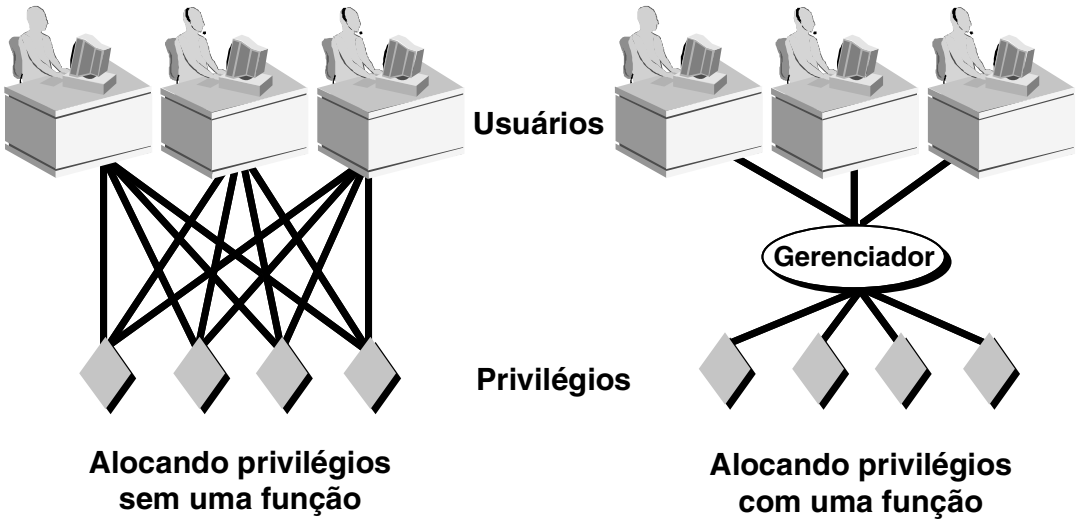
```
SQL> GRANT  create table, create sequence, create view
  2  TO     scott;
Grant succeeded.
```


Concedendo Privilégios de Sistema

O DBA usa a instrução GRANT para alocar os privilégios de sistema para o usuário. Quando forem concedidos privilégios ao usuário, ele pode usá-los imediatamente.

No exemplo do slide, foram atribuídos privilégios ao usuário Scott para criar tabelas, seqüências e views.

O Que É uma Função?



  ORACLE®

O Que É uma Função?

Uma função é um grupo nomeado de privilégios relacionados que podem ser concedidos ao usuário. Isso faz com que a concessão e revogação de privilégios se torne mais fácil de desempenhar e manter.

Um usuário pode ter acesso a várias funções e vários usuários podem ter a mesma função atribuída a eles. As funções são criadas tipicamente para uma aplicação de banco de dados.

Criando e Atribuindo uma Função

Primeiro, o DBA deve criar uma função. O DBA pode então atribuir privilégios e usuários às funções.

Sintaxe

```
CREATE   ROLE  função;
```

onde: *função* é o nome da função a ser criada

Agora que a função foi criada, o DBA pode usar a instrução GRANT para atribuir usuários às funções e privilégios à função.

Criando e Concedendo Privilégios a uma Função

```
SQL> CREATE ROLE manager;
Role created.
```

```
SQL> GRANT create table, create view
  2         to manager;
Grant succeeded.
```

```
SQL> GRANT manager to BLAKE, CLARK;
Grant succeeded.
```

  ORACLE®

Criando uma Função

O exemplo no slide cria um gerente de função e permite que os gerentes criem tabelas e views. Ele atribui então a função de gerentes a Blake e a Clark. Agora Blake e Clark podem criar tabelas e views.

Alterando Sua Senha

- **O DBA cria a sua conta de usuário e inicializa a sua senha.**
- **Você pode alterar sua senha usando a instrução ALTER USER.**

```
SQL> ALTER USER scott
  2         IDENTIFIED BY lion;
User altered.
```

  ORACLE®

Alterando Sua Senha

O DBA cria uma conta de usuário e inicializa uma senha para cada usuário. Você pode alterar sua senha usando a instrução ALTER USER.

Sintaxe

```
ALTER USER usuário IDENTIFIED BY senha;
```

onde:	*usuário*	é o nome do usuário
	senha	especifica a nova senha

Embora essa instrução possa ser usada para alterar a senha, há várias outras opções. Você deve ter o privilégio ALTER USER para alterar uma opção.

Para obter mais informações, consulte o *Oracle Server SQL Reference,* Release 8, "ALTER USER".

Privilégios de Objeto

Privilégio de Objeto	Tabela	View	Seqüência	Procedimento
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√			
SELECT	√	√	√	
UPDATE	√	√		

  ORACLE®

Privilégios de Objeto

Um *privilégio de objeto* é um privilégio ou direito de desempenhar uma determinada ação em uma tabela, view, seqüência ou procedimento específico. Cada objeto tem um conjunto determinado de privilégios concedíveis. A tabela no slide lista os privilégios de vários objetos. Observe que apenas os privilégios que se aplicam a uma seqüência são SELECT e ALTER. UPDATE, REFERENCES e INSERT podem ser restringidos especificando-se um subconjunto das colunas atualizáveis. Um privilégio SELECT pode ser restringido criando uma view com um subconjunto de colunas e concedendo o privilégio SELECT na view. Uma concessão em um sinônimo é convertida para uma concessão na tabela base referenciada pelo sinônimo.

Privilégios de Objeto

- **Os privilégios de objeto variam de objeto para objeto.**
- **Um proprietário tem todos os privilégios sobre o objeto.**
- **Um proprietário pode fornecer privilégios específicos sobre o objeto de proprietário.**

```
GRANT        object_priv [(colunas)]
ON           objeto
TO           {usuário|função|PUBLIC}
[WITH GRANT OPTION];
```


Concedendo Privilégios de Objeto

Privilégios de objeto diferentes estão disponíveis para tipos diferentes de objetos de esquema. Um usuário tem automaticamente todos os privilégios de objeto para objetos de esquema contidos no esquema do usuário. Um usuário pode conceder qualquer privilégio de objeto sobre qualquer objeto de esquema que o usuário possua para qualquer outro usuário ou função. Se a concessão incluir a instrução GRANT OPTION, o cedente pode reconceder o privilégio de objeto para outros usuários; senão, o cedente pode usar o privilégio, mas não pode concedê-lo a outros usuários.

Na sintaxe:

object_priv	é um privilégio de objeto a ser concedido
ALL	especifica todos os privilégios de objeto.
colunas	especifica a coluna de uma tabela ou view sobre as quais os privilégios são concedidos
ON *objeto*	é o objeto sobre o qual os privilégios são concedidos
TO	identifica a quem o privilégio é concedido
PUBLIC	concede privilégios de objeto a todos os usuários
WITH GRANT OPTION	permite que o cedente conceda privilégios de objeto a outros usuários e funções

Concedendo Privilégios de Objeto

- **Concede privilégios de consulta na tabela EMP.**

```
SQL> GRANT    select
  2  ON       emp
  3  TO       sue, rich;
Grant succeeded.
```

- **Concede privilégios para atualizar colunas específicas aos usuários e funções.**

```
SQL> GRANT    update (dname, loc)
  2  ON       dept
  3  TO       scott, manager;
Grant succeeded.
```

  ORACLE®

Diretrizes

- Para conceder privilégios sobre um objeto, ele deve estar no seu próprio esquema ou você deve ter recebido os privilégios de objeto WITH GRANT OPTION.
- Um proprietário de objeto pode conceder qualquer privilégio de objeto sobre o objeto para qualquer outro usuário ou função do banco de dados.
- O proprietário de um objeto adquire automaticamente todos os privilégios de objeto sobre aquele objeto.

O primeiro exemplo no slide concede aos usuários Sue e Rich o privilégio de consultar a tabela EMP. O segundo exemplo concede privilégios UPDATE sobre colunas específicas na tabela DEPT a Scott e à função de gerente.

Se Sue ou Rich tiverem que selecionar dados da tabela emp, a sintaxe que eles terão de usar é:

```
SQL> SELECT *
  2  FROM  scott.emp;
```

Outra alternativa é criar um sinônimo para a tabela e selecionar no sinônimo.

```
SQL> CREATE SYNONYM emp FOR scott.emp
SQL> SELECT * FROM emp;
```

Observação: Os DBAs geralmente alocam privilégios de sistema e qualquer usuário que possua um objeto pode conceder privilégios de objeto.

Usando palavras chave WITH GRANT OPTION e PUBLIC

- **Dar autoridade a um usuário para passar os privilégios.**

```
SQL> GRANT    select, insert
  2  ON       dept
  3  TO       scott
  4  WITH GRANT OPTION;
Grant succeeded.
```

- **Permitir que todos os usuários no sistema consultem dados na tabela DEPT de Alice.**

```
SQL> GRANT    select
  2  ON       alice.dept
  3  TO       PUBLIC;
Grant succeeded.
```

  ORACLE®

Palavra-chave WITH GRANT OPTION

Um privilégio que é concedido WITH GRANT OPTION pode ser passado para outros usuários e funções pelo cedente. Os privilégios de objeto que foram concedidos WITH GRANT OPTION são revogados quando o privilégio do concessor for revogado.

O exemplo no slide dá ao usuário Scott acesso à sua tabela DEPT com o privilégio de consultar a tabela e adicionar linhas a ela. O exemplo também permite que Scott conceda esse privilégio a outros.

Palavra-chave PUBLIC

O proprietário de uma tabela pode conceder acesso a todos os usuários usando a palavra-chave PUBLIC.

O segundo exemplo permite que todos os usuários no sistema consultem dados na tabela DEPT de Alice.

Confirmando Privilégios Concedidos

Tabela de Dicionário de Dados	Descrição
ROLE_SYS_PRIVS	Privilégios de sistema concedidos a funções
ROLE_TAB_PRIVS	Privilégios de tabela concedidos a funções
USER_ROLE_PRIVS	Funções acessíveis ao usuário
USER_TAB_PRIVS_MADE	Os privilégios de objeto concedidos aos objetos do usuário
USER_TAB_PRIVS_RECD	Os privilégios de objeto concedidos ao usuário
USER_COL_PRIVS_MADE	Os privilégios de objeto concedidos às colunas dos objetos do usuário
USER_COL_PRIVS_RECD	Os privilégios de objeto concedidos ao usuário em colunas específicas

Confirmando Privilégios Concedidos

Se você tentar executar uma operação não autorizada — por exemplo, deletar uma linha de uma tabela para a qual você não tem o privilégio DELETE — o Oracle Server não permitirá que a operação ocorra.

Se você receber a mensagem de erro "tabela ou view não existe" ("table or view does not exist" do Oracle Server, você poderá ter:

- Nomeado uma tabela ou view que não existe
- Tentado executar uma operação em uma tabela ou view para a qual você não tem o privilégio apropriado

Você pode acessar os dicionários de dados para ver os privilégios que você tem. A tabela no slide descreve várias tabelas de dicionários de dados.

Como Revogar Privilégios de Objeto

- **Use a instrução REVOKE para revogar os privilégios concedidos a outros usuários.**
- **Os privilégios concedidos a outros usuários por WITH GRANT OPTION também serão revogados.**

```
REVOKE {privilégio [, privilégio...] | ALL}
ON     objeto
FROM   {usuário[, usuário...] | função | PUBLIC}
[CASCADE CONSTRAINTS];
```


Revogando Privilégios de Objeto

A remoção de privilégios concedidos a outros usuários usando a instrução REVOKE. Quando você usa a instrução REVOKE, os privilégios que você especificar são revogados dos usuários que você nomear e dos outros usuários para quem esses privilégios foram concedidos pela cláusula WITH GRANT OPTION.

Na sintaxe:

CASCADE	é obrigatório para remover quaisquer restrições de integridade feitas ao
CONSTRAINTS	objeto por meio do privilégio REFERENCES

Para obter mais informações, consulte o *Oracle Server SQL Reference,* Release 8, "REVOKE".

Revogando Privilégios de Objeto

Assim como a usuária Alice, revogue os privilégios SELECT e INSERT fornecidos ao usuário Scott na tabela DEPT.

```
SQL> REVOKE   select, insert
  2  ON       dept
  3  FROM     scott;
Revoke succeeded.
```


Revogando Privilégios de Objeto (continuação)

O exemplo no slide revoga os privilégios SELECT e INSERT concedidos ao usuário Scott na tabela DEPT.

Observação: Se um usuário obtiver o privilégio WITH GRANT OPTION, ele também poderá conceder o privilégio WITH GRANT OPTION para que uma longa corrente de cedentes seja possível, mas não são permitidas concessões circulares. Se um proprietário revogar um privilégio de um usuário que concedeu o mesmo privilégio a outros usuários, a instrução REVOKE se aplicará também aos outros usuários.

Por exemplo, se um usuário A concede o privilégio SELECT em uma tabela ao usuário B incluindo WITH GRANT OPTION, o usuário B pode conceder ao usuário C o privilégio SELECT chamado WITH GRANT OPTION e o usuário C pode conceder ao usuário D o privilégio SELECT. Se o usuário A revoga o privilégio do usuário B, esse mesmo privilégio, concedido aos usuários C e D, são revogados.

Sumário

Instrução	Ação
CREATE USER	Permite que o DBA crie um usuário
GRANT	Permite que o usuário conceda a outros usuários privilégios para acessar os objetos do usuário
CREATE ROLE	Permite que o DBA crie um conjunto de privilégios
ALTER USER	Permite que os usuários alterem as suas senhas
REVOKE	Remove os privilégios sobre um objeto dos usuários

Sumário

Os DBAs estabelecem segurança de banco de dados inicial para usuários atribuindo privilégios aos outros usuários.

- O DBA cria usuários que devem ter uma senha. O DBA também é responsável por estabelecer os privilégios de sistema iniciais dos usuários.
- Quando o usuário tiver criado um objeto, o usuário pode passar quaisquer privilégios de objeto disponíveis a outros usuários ou para todos os usuários usando a instrução GRANT.
- Um DBA pode criar funções usando a instrução CREATE ROLE para passar um conjunto de privilégios de sistema ou de objeto a vários usuários. As funções tornam a concessão e a revogação de privilégios mais fáceis de manter.
- Os usuários podem alterar sua senha usando a instrução ALTER USER.
- Você pode remover privilégios de usuários usando a instrução REVOKE.
- As views do dicionário de dados permitem que os usuários vejam os privilégios concedidos a eles e aqueles que são concedidos sobre os seus objetos.

Visão Geral do Exercício

- **Concedendo a outros usuários privilégios sobre sua tabela**
- **Modificando a tabela de um outro usuário através de privilégios concedidos a você**
- **Criando um sinônimo**
- **Consultando as views do dicionário de dados relacionados aos privilégios**

Visão Geral do Exercício

Junte-se a outros alunos para este exercício de controle de acesso aos objetos de banco de dados.

Exercício 14

1. Qual o privilégio que um usuário deve receber para estabelecer login no Oracle Server? É um privilégio de objeto ou sistema?

   _____

2. Qual o privilégio que um usuário deve receber para criar tabelas?

   _____

3. Se você criar uma tabela, quem poderá passar privilégios para outros usuários sobre sua tabela?

   _____

4. Você é o DBA. Você está criando muitos usuários que estão exigindo os mesmos privilégios de sistema. O que você faria para tornar seu trabalho mais fácil?

   _____

5. Que comando você usa para alterar sua senha?

   _____

6. Conceda acesso à tabela DEPT a outro usuário. Faça com que o usuário lhe conceda acesso de consulta à tabela DEPT dele(a).
7. Consulte todas as linhas na tabela DEPT.

```
DEPTNO DNAME      LOC
------ ---------- ---------
    10 ACCOUNTING NEW YORK
    20 RESEARCH   DALLAS
    30 SALES      CHICAGO
    40 OPERATIONS BOSTON
```

8. Adicione uma nova linha à tabela DEPT. A equipe 1 deve adicionar Education como o departamento número 50. A Equipe 2 deve adicionar o departamento Administration como o departamento número 50. Torne as alterações permanentes.
9. Crie um sinônimo para a tabela DEPT da outra equipe.

Exercício 14 (continuação)

10. Consulte todas as linhas na tabela DEPT da outra equipe, usando o sinônimo.

Resultados da instrução SELECT para a Equipe 1.

```
DEPTNO DNAME          LOC
------ -------------- ---------
    10 ACCOUNTING     NEW YORK
    20 RESEARCH       DALLAS
    30 SALES          CHICAGO
    40 OPERATIONS     BOSTON
    50 ADMINISTRATION
```

Resultados da instrução SELECT para a Equipe 2.

```
DEPTNO DNAME          LOC
------ -------------- ---------
    10 ACCOUNTING     NEW YORK
    20 RESEARCH       DALLAS
    30 SALES          CHICAGO
    40 OPERATIONS     BOSTON
    50 EDUCATION
```

11. Consulte o dicionário de dados USER_TABLES para ver as informações sobre as tabelas que você possui.

```
TABLE_NAME
----------------
BONUS
CUSTOMER
DEPARTMENT
DEPT
DUMMY
EMP
EMPLOYEE
ITEM
MY_EMPLOYEE
ORD
PRICE
PRODUCT
SALGRADE
13 rows selected.
```

Exercício 14 (continuação)

12. Consulte a view de dicionário de dados ALL_TABLES para ver as informações sobre todas as tabelas que você pode acessar. Exclua as suas próprias tabelas.

```
TABLE_NAME   OWNER
-----------  ------------
DEPT         <user2>
```

13. Revogue o privilégio SELECT da outra equipe.

15

SQL Workshop

Visão Geral do Workshop

- Criando tabelas e seqüências
- Modificando dados nas tabelas
- Modificando uma definição de tabela
- Criando uma view
- Criando scripts que contenham comandos SQL e SQL*Plus
- Gerando um relatório simples

Visão Geral do Workshop

Neste workshop você estruturará um conjunto de tabelas de bancos de dados para uma aplicação de vídeo. Ao criar as tabelas, você inserirá, atualizará e deletará registros em um banco de dados de armazenamento de vídeo e gerará um relatório. O banco de dados contém apenas as tabelas essenciais.

Observação: Se você quiser estruturar as tabelas, poderá executar o script `buildtab.sql` no SQL*Plus. Se você quiser eliminar as tabelas, poderá executar o script `dropvid.sql` no SQL*Plus. Você poderá então executar o script `buildvid.sql` no SQL*Plus para criar e povoar a tabela. Se você usar o script `buildvid.sql` para estruturar e povoar as tabelas, inicie com o Exercício 6b.

Exercício 15

1. Crie as tabelas de acordo com as tabelas de exemplo a seguir. Escolha os tipos de dados apropriados e certifique-se de adicionar restrições de integridade.

   a. Nome da tabela: MEMBER

Column_ Name	MEMBER_ ID	LAST_ NAME	FIRST_NAME	ADDRESS	CITY	PHONE	JOIN_ DATE
Key Type	PK						
Null/ Unique	NN,U	NN					NN
Default Value							System Date
Datatype	Number	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	Date
Length	10	25	25	100	30	15	

   b. Nome da tabela: TITLE

Column_ Name	TITLE_ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_ DATE
Key Type	PK					
Null/ Unique	NN,U	NN	NN			
Check				G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMEN TARY	
Datatype	Number	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	Date
Length	10	60	400	4	20	

Exercício 15 (continuação)

c. Nome da tabela: TITLE_COPY

Column Name	COPY_ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/ Unique	NN,U	NN,U	NN
Check			AVAILABLE, DESTROYED, RENTED, RESERVED
FK Ref Table		TITLE	
FK Ref Col		TITLE_ID	
Datatype	Number	Number	VARCHAR2
Length	10	10	15

d. Nome da tabela: RENTAL

Column Name	BOOK_ DATE	MEMBER_ ID	COPY_ ID	ACT_RET_ DATE	EXP_RET_ DATE	TITLE_ ID
Key Type	PK	PK,FK1	PK,FK2			PK,FK2
Default Value	System Date				System Date + 2 days	
FK Ref Table		MEMBER	TITLE_ COPY			TITLE_ COPY
FK Ref Col		MEMBER_ ID	COPY_ ID			TITLE_ID
Datatype	Date	Number	Number	Date	Date	Number
Length		10	10			10

Exercício 15 (continuação)

e. Nome da tabela: RESERVATION

Column Name	RES_ DATE	MEMBER_ ID	TITLE_ ID
Key Type	PK	PK,FK1	PK,FK2
Null/ Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE
FK Ref Column		MEMBER_ID	TITLE_ID
Datatype	Date	Number	Number
Length		10	10

2. Verifique se as tabelas e as restrições foram criadas adequadamente, verificando o dicionário de dados.

```
TABLE_NAME
-----------
MEMBER
RENTAL
RESERVATION
TITLE
TITLE_COPY


CONSTRAINT_NAME                C TABLE_NAME
------------------------------ - --------------
MEMBER_LAST_NAME_NN            C MEMBER
MEMBER_JOIN_DATE_NN            C MEMBER
MEMBER_MEMBER_ID_PK            P MEMBER
RENTAL_BOOK_DATE_COPY_TITLE_PK P RENTAL
RENTAL_MEMBER_ID_FK            R RENTAL
RENTAL_COPY_ID_TITLE_ID_FK     R RENTAL
RESERVATION_RESDATE_MEM_TIT_PK P RESERVATION
RESERVATION_MEMBER_ID          R RESERVATION
RESERVATION_TITLE_ID           R RESERVATION
...
17 rows selected.
```

Exercício 15 (continuação)

3. Crie seqüências para identificar exclusivamente cada linha das tabelas MEMBER e TITLE.
   a. Número de membro para tabela MEMBER: inicie com 101; não permita o armazenamento em cache dos valores. Nomeie a seqüência member_id_seq.
   b. Número de título para a tabela TITLE: inicie com 92; não permita o armazenamento em cache. Nomeie a seqüência title_id_seq.
   c. Verifique a existência das seqüências no dicionário de dados.

```
SEQUENCE_NAME  INCREMENT_BY LAST_NUMBER
-------------- ------------ -----------
TITLE_ID_SEQ              1          92
MEMBER_ID_SEQ             1         101
```

4. Adicione dados às tabelas. Crie um script para cada conjunto de dados a adicionar.
   a. Adicione títulos de filmes à tabela TITLE . Crie um script para fornecer as informações sobre os filmes. Salve o arquivo de script como `p8q4.sql`. Use as seqüências para identificar exclusivamente cada título. Informe as datas de lançamento no formato DD-MON-YYYY. Lembre-se que as aspas simples devem ser tratadas com atenção em um campo de caractere. Verifique as adições.

```
TITLE
------------------------
Willie and Christmas Too
Alien Again
The Glob
My Day Off
Miracles on Ice
Soda Gang
6 rows selected.
```

Exercício 15 (continuação)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York	PG	COMEDY	12-JUL-1995
Miracles on Ice	A six-year-old has doubts about Santa Claus, but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-1995
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

Exercício 15 (continuação)

b. Adicione dados à tabela MEMBER. Crie um script nomeado `p15q4b.sql` para pedir informações aos usuários. Execute o script. Certifique-se de usar a seqüência para adicionar o membro números.

First_ Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

Exercício 15 (continuação)

c. Adicione as seguintes cópias de filmes à tabela TITLE_COPY:
Observação: Obtenha os números do title_id para este exercício.

Title	Copy_Id	Status
Willie and Christmas Too	1	AVAILABLE
Alien Again	1	AVAILABLE
	2	RENTED
The Glob	1	AVAILABLE
My Day Off	1	AVAILABLE
	2	AVAILABLE
	3	RENTED
Miracles on Ice	1	AVAILABLE
Soda Gang	1	AVAILABLE

d. Adicione os seguintes aluguéis à tabela RENTAL:
Observação: O número de título pode ser diferente, dependendo no número de seqüência.

Title_ Id	Copy_ Id	Member_ Id	Book_date	Exp_Ret_Date	Act_Ret_Date
92	1	101	3 days ago	1 day ago	2 days ago
93	2	101	1 day ago	1 day from now	
95	3	102	2 days ago	Today	
97	1	106	4 days ago	2 days ago	2 days ago

Exercício 15 (continuação)

5. Crie uma view nomeada TITLE_AVAIL para mostrar os títulos dos filmes e a disponibilidade de cada cópia e a data esperada de devolução, caso esteja alugado. Consulte todas as linhas a partir da view. Ordene os resultados por título.

```
TITLE              COPY_ID STATUS     EXP_RET_D
------------------ ------- ---------- ------------
Alien Again              1 AVAILABLE
Alien Again              2 RENTED     05-NOV-97
Miracles on Ice          1 AVAILABLE
My Day Off               1 AVAILABLE
My Day Off               2 AVAILABLE
My Day Off               3 RENTED     06-NOV-97
Soda Gang                1 AVAILABLE  04-NOV-97
The Glob                 1 AVAILABLE
Willie and Christmas Too 1 AVAILABLE  05-NOV-97
9 rows selected.
```

6. Faça alterações nos dados das tabelas.
   a. Adicione um novo título. O filme se chama "Interstellar Wars", que tem censura PG e é classificado como ficção científica. O data de lançamento é 07-JUL-77. A descrição é filme de ação "Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?" Certifique-se de adicionar um registro da cópia do título para as duas cópias.
   b. Informe duas reservas. Uma reserva é para Carmen Velasquez, que deseja alugar "Interstellar Wars". Outra é para Mark Quick-to-See, que deseja alugar "Soda Gang".

Exercício 15 (continuação)

c. A cliente Carmen Velasquez aluga a cópia 1 do filme "Interstellar Wars". Remova a sua reserva do filme. Registre as informações sobre o aluguel. Autorize o valor padrão da data de devolução estimada a ser usado. Verifique se o aluguel foi registrado usando a view que você criou.

```
TITLE                     COPY_ID STATUS    EXP_RET_D
------------------------  ------- --------- ----------------
Alien Again                     1 AVAILABLE
Alien Again                     2 RENTED    05-NOV-97
Interstellar Wars               1 RENTED    08-NOV-97
Interstellar Wars               2 AVAILABLE
Miracles on Ice                 1 AVAILABLE
My Day Off                      1 AVAILABLE
My Day Off                      2 AVAILABLE
My Day Off                      3 RENTED    06-NOV-97
Soda Gang                       1 AVAILABLE 04-NOV-97
The Glob                        1 AVAILABLE
Willie and Christmas Too        1 AVAILABLE 05-NOV-97
11 rows selected.
```

7. Faça uma modificação em uma das tabelas.

a. Adicione uma coluna PRICE à tabela TITLE para registrar o preço de compra o vídeo. A coluna deve ter um comprimento total de oito registros e duas casas decimais. Verifique as modificações.

```
Name            Null?     Type
--------------- --------- -----
TITLE_ID        NOT NULL  NUMBER(10)
TITLE           NOT NULL  VARCHAR2(60)
DESCRIPTION     NOT NULL  VARCHAR2(400)
RATING                    VARCHAR2(4)
CATEGORY                  VARCHAR2(20)
RELEASE_DATE              DATE
PRICE                     NUMBER(8,2)
```

Exercício 15 (continuação)

b. Crie um script denominado `p15q7b.sql` para atualizar cada vídeo com o preço de acordo com a lista a seguir.

Observação: Obtenha os title_id numbers para este exercício.

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

c. Certifique-se de que no futuro todos os títulos contenham um preço. Verifique a restrição.

```
CONSTRAINT_NAME      C  SEARCH_CONDITION
-------------------  -- -----------------
TITLE_PRICE_NN       C  PRICE IS NOT NULL
```

8. Crie um relatório intitulado Relatório Histórico de Cliente. Esse relatório conterá o histórico de aluguel de vídeos de cada cliente. Certifique-se de incluir o nome do cliente, o filme alugado, as datas e a duração do aluguel. Número total de aluguéis de todos os clientes no período de criação do relatório. Salve o script em um arquivo nomeado `p15q8.sql`.

```
MEMBER           TITLE                     BOOK_DATE DURATION
---------------- ------------------------- --------- --------
Carmen Velasquez Willie and Christmas Too  03-NOV-97        1
                 Alien Again               09-AUG-98
                 Interstellar Wars         10-AUG-98


LaDoris Ngao     My Day Off                08-AUG-98


Molly Urguhart   Soda Gang                 06-AUG-98        2
```

16

Declarando Variáveis

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

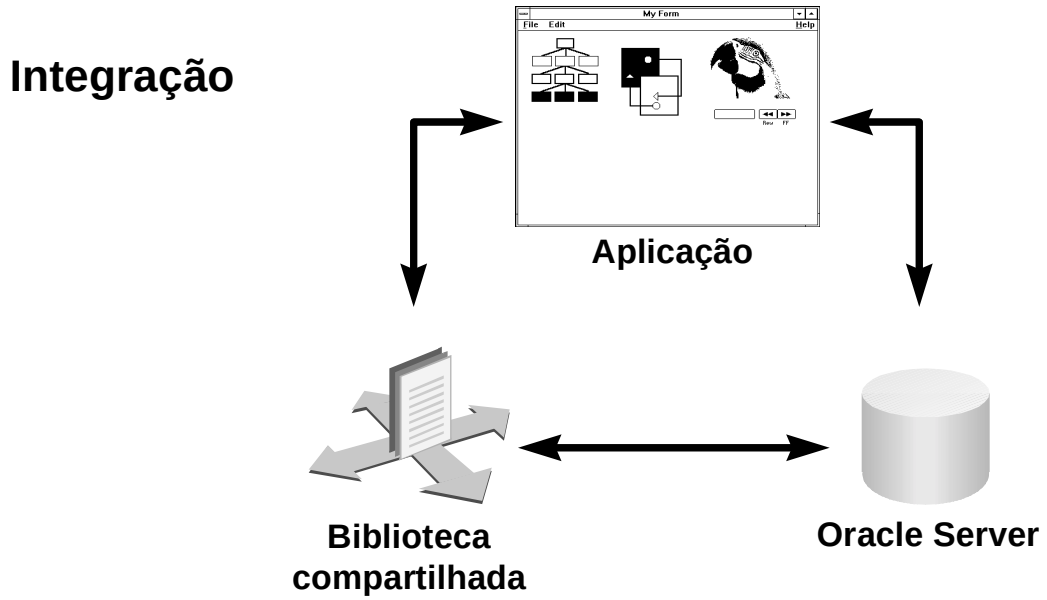
- **A linguagem PL/SQL é uma extensão da linguagem SQL com recursos de design de linguagens de programação.**
- **As instruções de consulta e a manipulação de dados em SQL estão incluídas nas unidades procedurais de código.**

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



16-4 ORACLE®

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

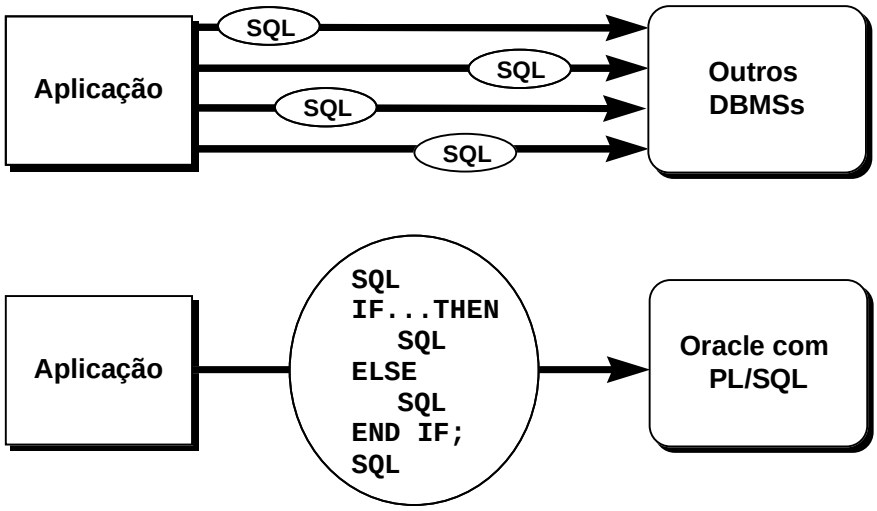
Várias ferramentas Oracle, inclusive o Oracle Developer, têm o seu próprio mecanismo PL/SQL, o qual é independente do mecanismo presente no Oracle Server.

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

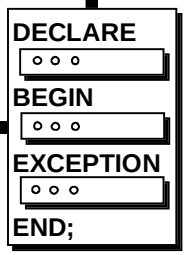
A linguagem PL/SQL pode melhorar o desempenho de uma aplicação. Os benefícios diferem dependendo do ambiente de execução.

- 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**



Estrutura de Bloco PL/SQL

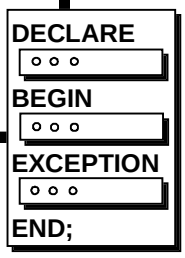
A linguagem PL/SQL é uma linguagem estruturada em blocos, o que significa que os programas podem ser divididos em blocos lógicos. Um bloco PL/SQL consiste em até três seções: declarativa (opcional), executável (necessária) e tratamento de exceção (opcional). Apenas as palavras-chave BEGIN e END são necessárias.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



16-7 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:

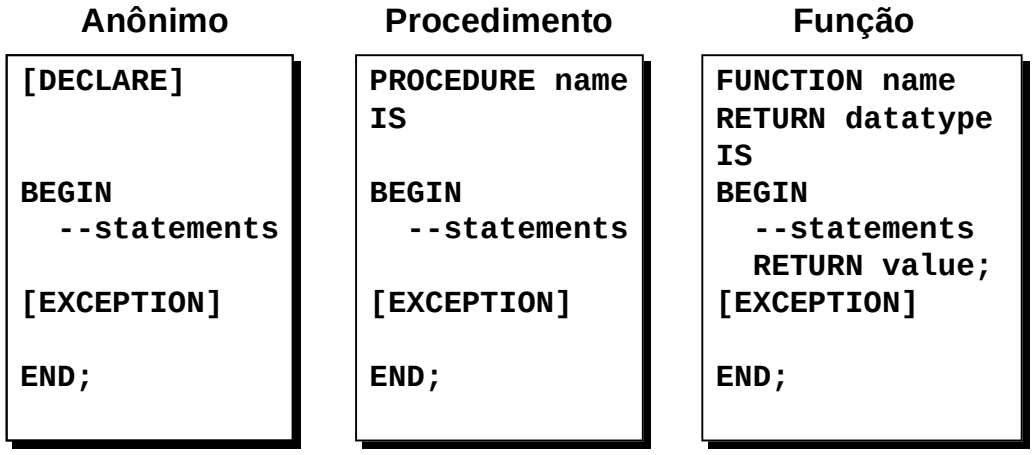
```
PL/SQL procedure successfully completed
(Procedimento PL/SQL concluído corretamente)
```

- 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



  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.

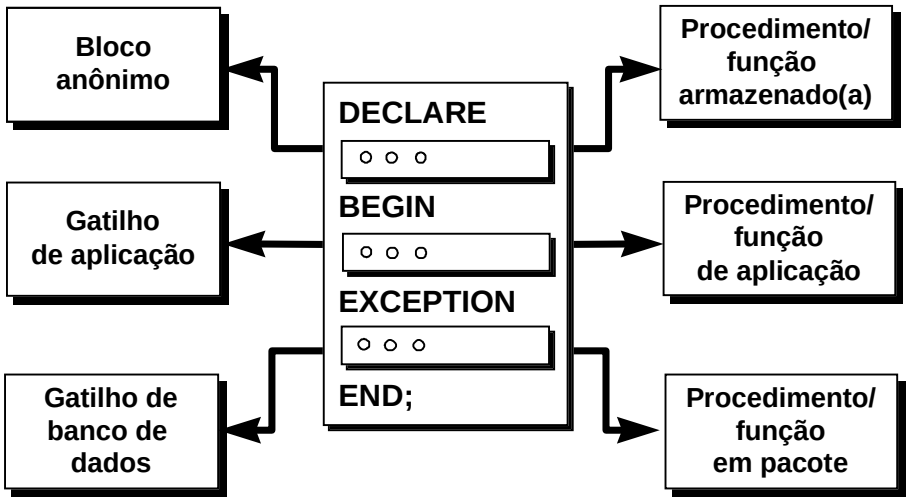
Subprogramas

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



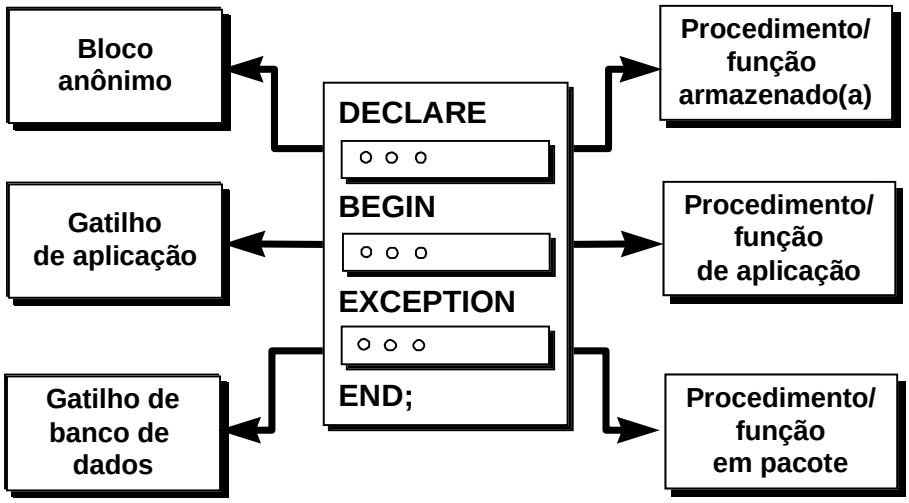
  ORACLE®

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



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 subseqüente), 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.**

  ORACLE®

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**

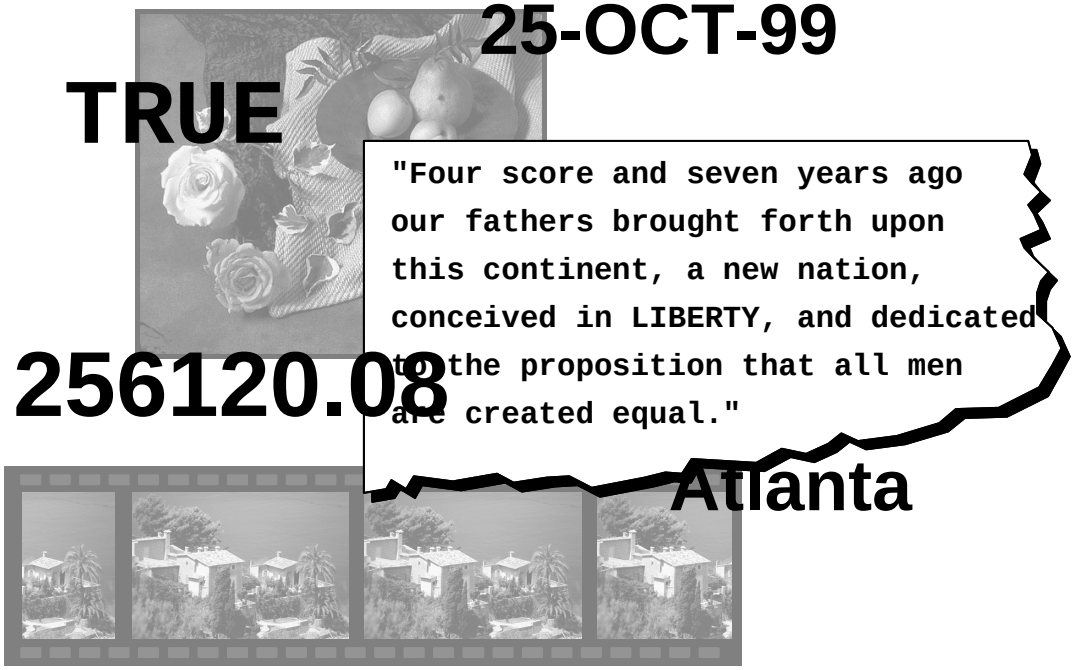
Usando Variáveis SQL*Plus nos Blocos PL/SQL

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



Tipos de Variáveis

O slide ilustra os seguintes tipos de dados de variável:

- TRUE representa um valor booleano.
- 25-OCT-99 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.**

  ORACLE®

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;
```

Adote uma convenção de nomeação para identificadores PL/SQL: por exemplo, v_empno

Regras para Nomeação

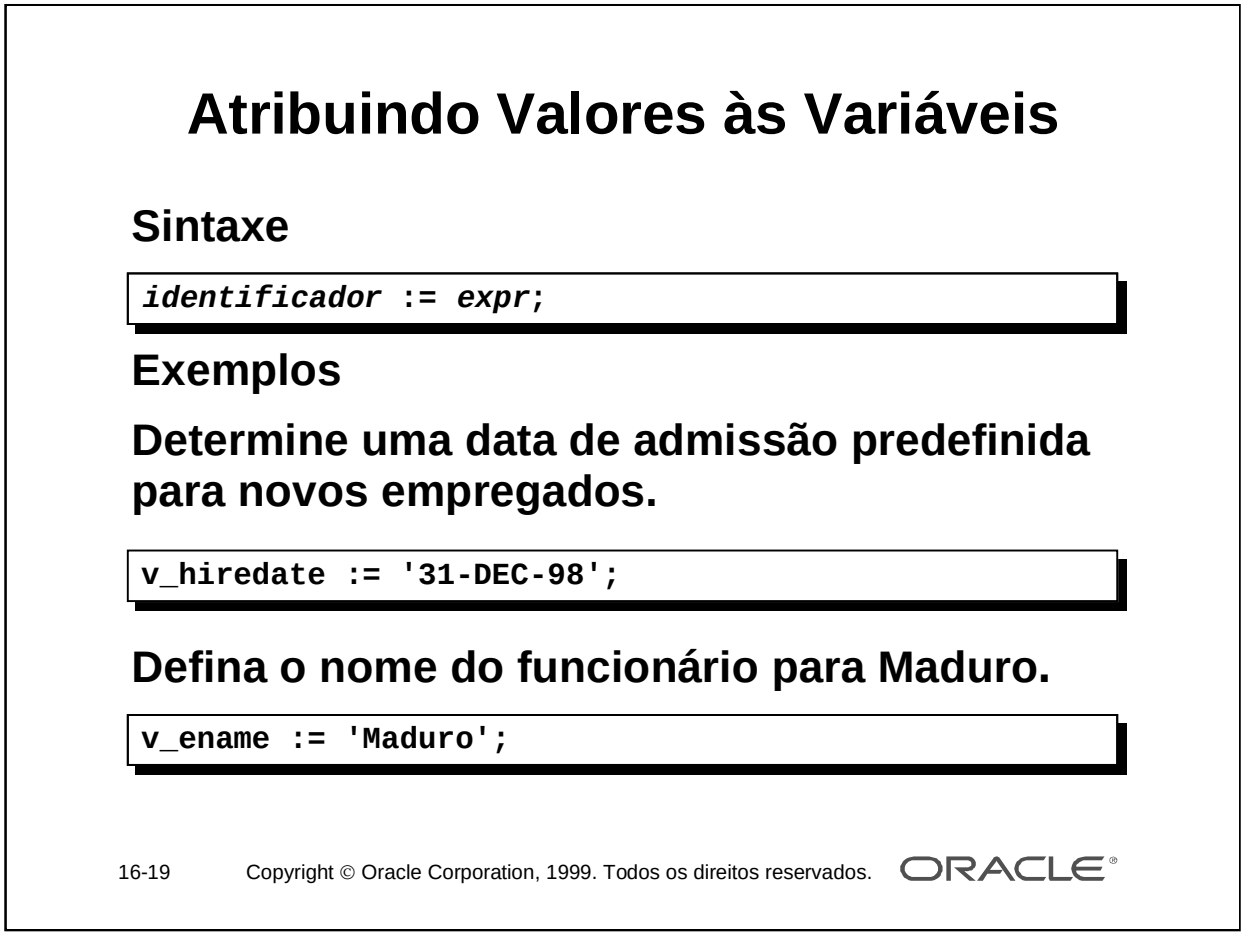
Dois objetos podem ter o mesmo nome, contanto que eles sejam definidos em diferentes blocos. Onde eles coexistirem, apenas o objeto declarado no bloco atual pode ser usado.

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

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	é o nome da variável escalar
expr	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**

  ORACLE®

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 Oracle8*i*. 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.

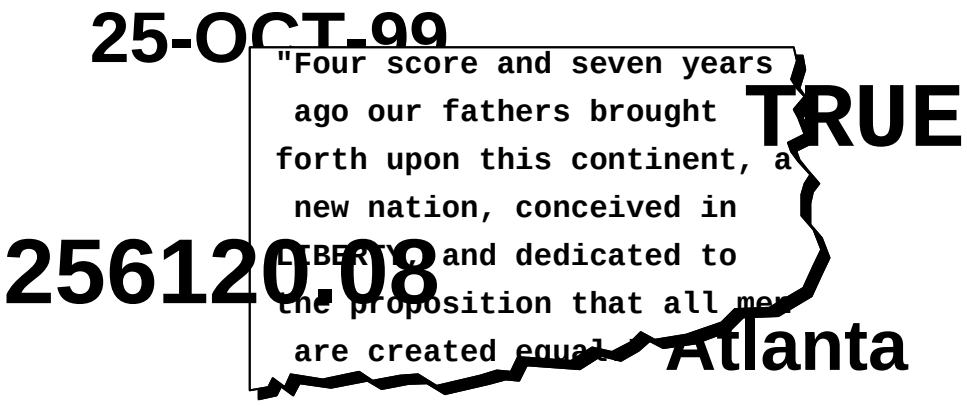
```
v_location  VARCHAR2(13) NOT NULL := 'CHICAGO';
```

(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**

  ORACLE®

Tipos de Dados Escalares Básicos

Tipo de Dados	Descrição
VARCHAR2 (*maximum_length*)	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 [(*precisão, escala*)]	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 [(*maximum_length*)]	Tipo básico para dados de caractere de tamanho fixo até 32.767 bytes. Se você não especificar um *comprimento_máx*, 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**

  ORACLE®

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**

  ORACLE®

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 freqüência 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 preenche-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 uma 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:

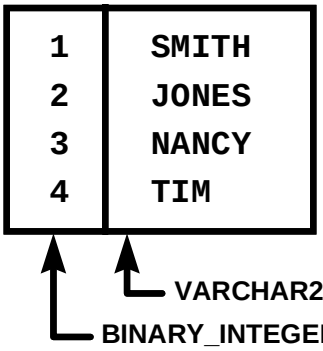
```
v_sal1 < v_sal2
```

Declare e inicialize uma variável booleana:

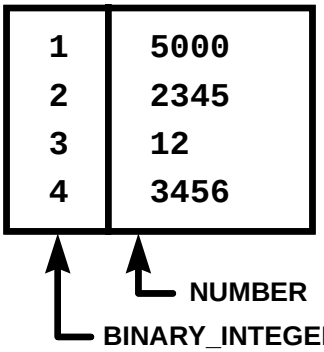
```
v_comm_sal BOOLEAN := (v_sal1 < v_sal2);
```

Estrutura de Registro PL/SQL

Estrutura de tabela PL/SQL



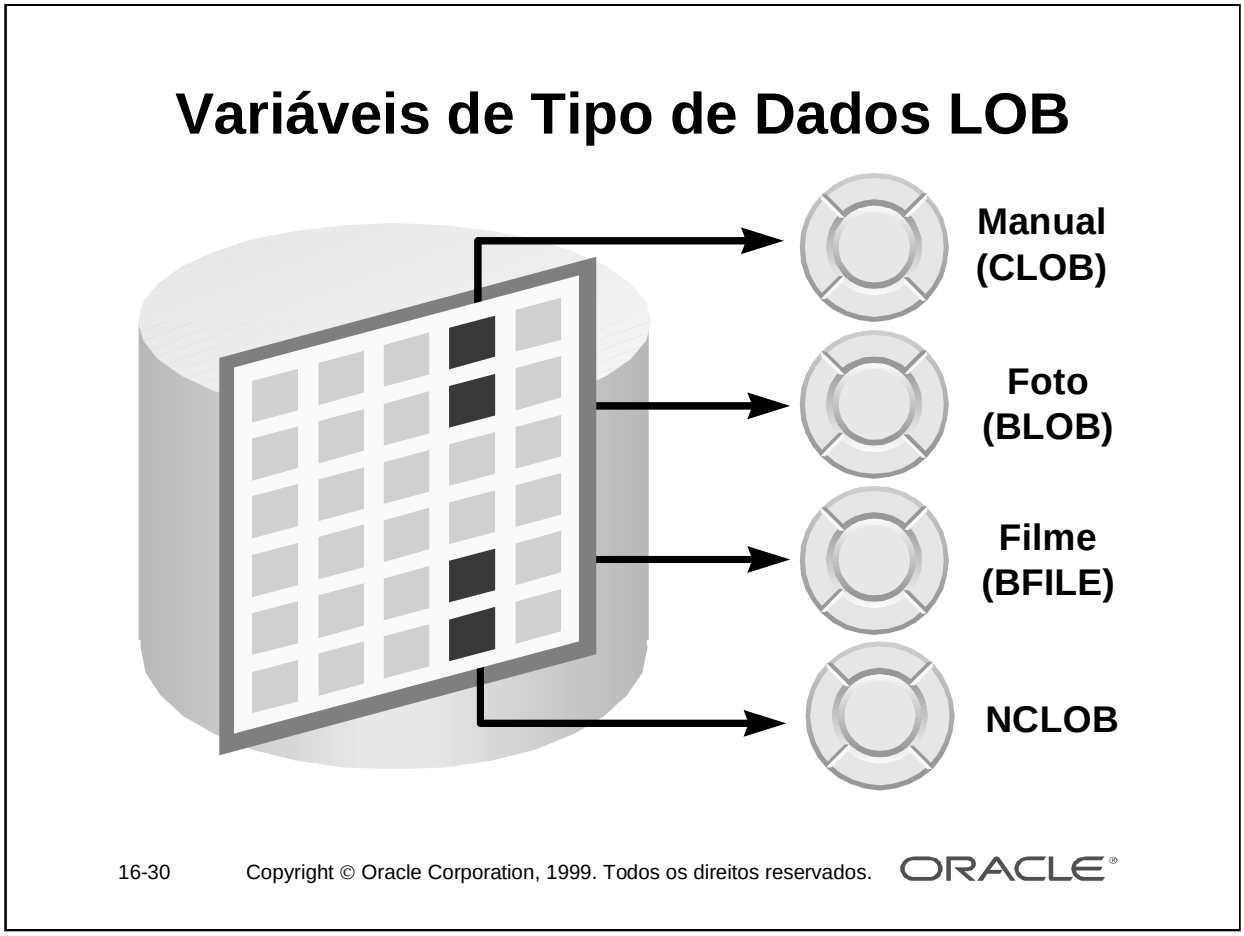
Estrutura de tabela PL/SQL



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 subseqüente. 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".

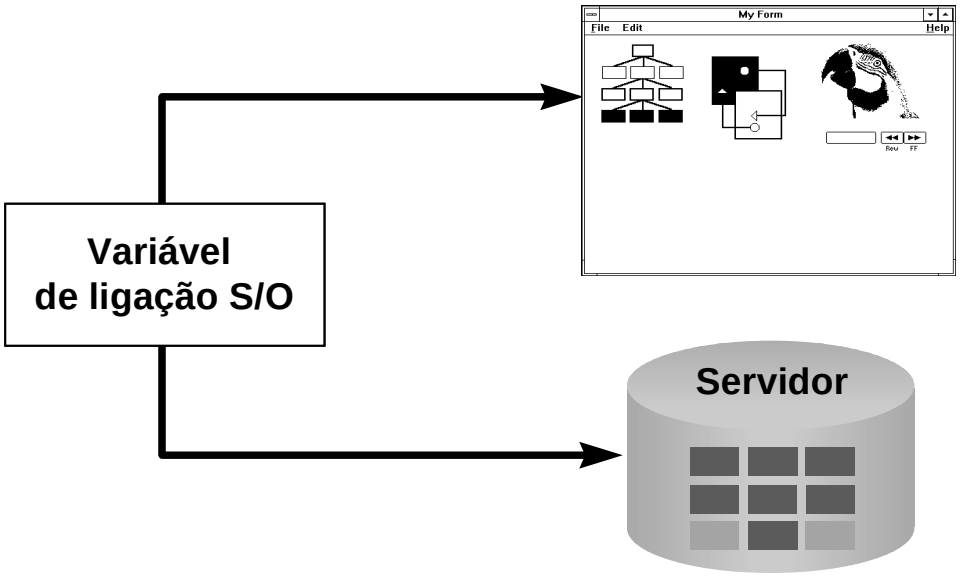


Variáveis de Tipo de Dados LOB

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, videoclipes 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 objeto 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 (:).**

  ORACLE®

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

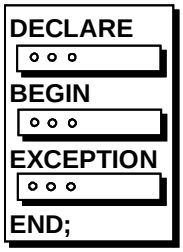
Esse script calcula o salário mensal e imprime-as na tela, usando o procedimento DBMS_OUTPUT.PUT_LINE.

```
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

- **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.**



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.

Visão Geral do Exercício

- **Determinando a validade das declarações**
- **Desenvolvendo um bloco PL/SQL simples**

Visão Geral do Exercício

Esse exercício enfatiza os fundamentos do código PL/SQL aprendidos nesta lição, incluindo tipos de dados, definições legais de identificadores validação de expressões. Você deve colocar todos esses elementos juntos para criar um bloco PL/SQL simples.

Questões Impressas

As questões 1 e 2 são questões impressas.

Exercício 16

1. Avalie cada uma das declarações a seguir. Determine quais delas *não* são legais e explique por quê.

   a.
   ```
   DECLARE
     v_id              NUMBER(4);
   ```

   b.
   ```
   DECLARE
     v_x, v_y, v_z     VARCHAR2(10);
   ```

   c.
   ```
   DECLARE
     v_birthdate       DATE NOT NULL;
   ```

   d.
   ```
   DECLARE
     v_in_stock        BOOLEAN := 1;
   ```

Exercício 16 (continuação)

2. Em cada uma das seguintes atribuições, determine o tipo de dados da expressão resultante.
   a. `v_days_to_go := v_due_date - SYSDATE;`

   b. `v_sender := USER || ': ' || TO_CHAR(v_dept_no);`

   c. `v_sum := $100,000 + $250,000;`

   d. `v_flag := TRUE;`

   e. `v_n1 := v_n2 > (2 * v_n3);`

   f. `v_value := NULL;`

3. Crie um bloco anônimo para a saída da frase "My PL/SQL Block Works" na tela.

```
G_MESSAGE
------------------------
My PL/SQL Block Works
```

Exercício 16 (continuação)

Se você tiver tempo, complete o exercício abaixo:

4. Crie um bloco que declare duas variáveis. Atribua o valor dessas variáveis PL/SQL às variáveis de host SQL*Plus e imprima os resultados das variáveis PL/SQL na tela. Execute o bloco PL/SQL. Salve o bloco PL/SQL no arquivo `p16q4.sql`.

```
V_CHAR Character (variable length)
V_NUM  Number
```

Atribua valores a essas variáveis do seguinte modo:

```
Variable Value
-------- -------------------------------------
V_CHAR   The literal '42 is the answer'
V_NUM    The first two characters from V_CHAR


G_CHAR
---------------------
42 is the answer

    G_NUM
---------
       42
```

17

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**

  ORACLE®

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 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

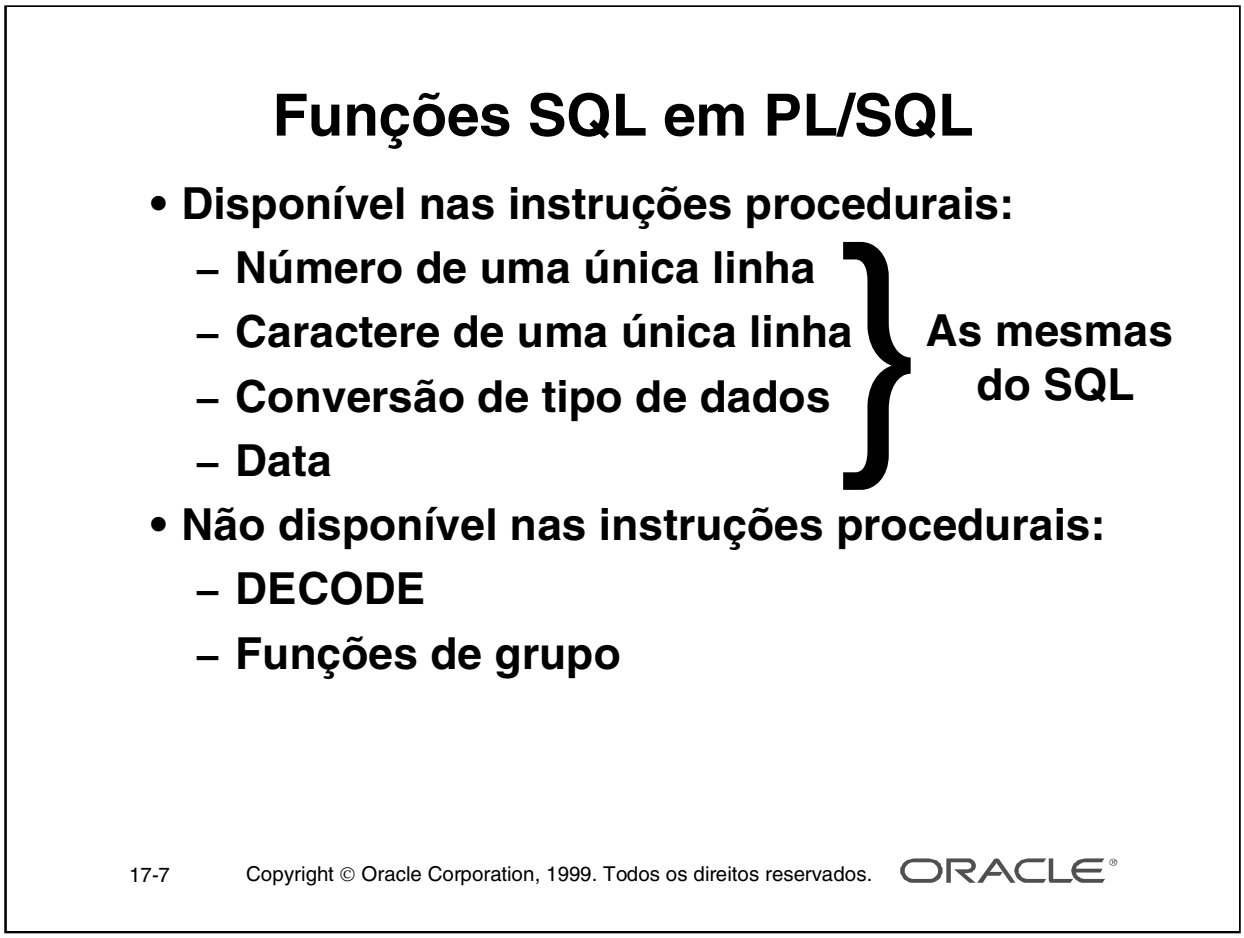
```
...
  v_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_sal := &p_monthly_sal * 12;
END;     -- This is the end of the block
```


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

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

- **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;
```


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.**

  ORACLE®

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.

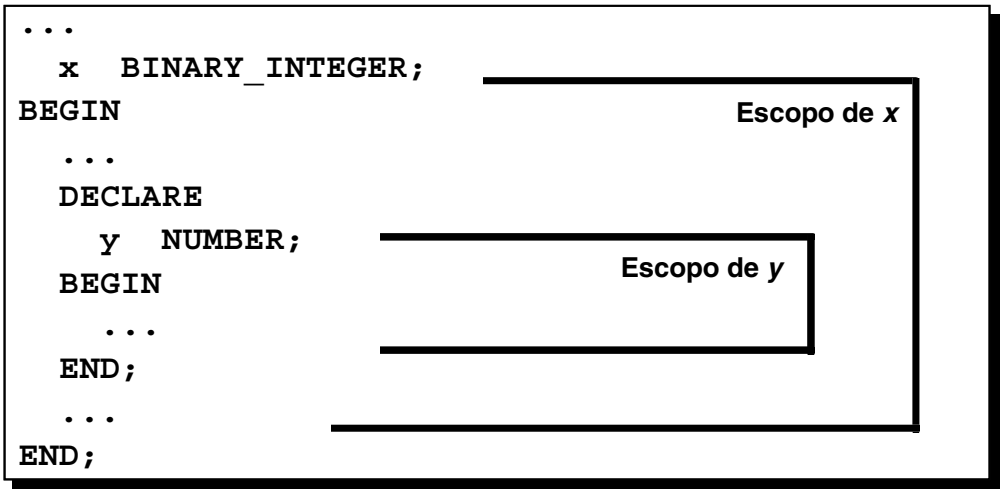
O escopo se aplica a todos os objetos declarados, incluindo variáveis, cursores, exceções definidas pelo usuário e constantes.

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



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

- Lógico
- Aritmético
- Concatenação
- Parênteses para controlar a ordem das operações

} Os mesmos do SQL

- Operador exponencial (**)

  ORACLE®

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);
```

  ORACLE®

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 seqüência 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;
/
```

  ORACLE®

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**

  ORACLE®

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 outro identificadores, de um banco de dados e de outros objetos nomeados.

Identificador	Convenção de Nomeação	Exemplo
Variável	v_*name*	v_sal
Constante	c_*name*	c_company_name
Cursor	*name*_cursor	emp_cursor
Exceção	e_*name*	e_too_many
Tipo de tabela	*name*_table_type	amount_table_type
Tabela	*name*_table	order_total_table
Tipo de registro	*name*_record_type	emp_record_type
Registro	*name*_record	customer_record
Variável de substituição SQL*Plus (também referida como parametro de substituição)	p_*name*	p_sal
Variável global SQL*Plus (também referida como host ou variável de ligação)	g_*name*	g_year_sal

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;
```


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

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;
```

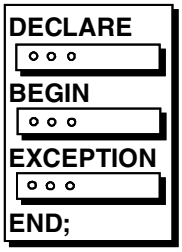

Exercício de Classe

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

- **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**



  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, o formato 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.

Visão Geral do Exercício

- **Revisando as regras e aninhamento e de criação de escopos**
- **Desenvolvendo e testando os blocos PL/SQL**

  ORACLE®

Visão Geral do Exercício

Este exercício consolida os fundamentos sobre o PL/SQL apresentados na lição, incluindo as regras para aninhar os blocos PL/SQL do código e modo de executar e testar o seu código PL/SQL.

Questões Impressas

As questões 1 e 2 são questões impressas.

Exercício 17

Bloco PL/SQL

```
DECLARE
  v_weightNUMBER(3) := 600;
  v_message           VARCHAR2(255) := 'Product 10012';
BEGIN
                      /* SUB-BLOCK */
    DECLARE
      v_weight          NUMBER(3) := 1;
      v_message         VARCHAR2(255) := 'Produto 11001';
      v_new_locn        VARCHAR2(50) := 'Europa';
    BEGIN
      v_weight := v_weight + 1;
      v_new_locn := 'Ocidental ' || v_new_locn;
    END;

  v_weight := v_weight + 1;
  v_message := v_message || ' is in stock';
  v_new_locn           := 'Western ' || v_new_locn;
END;
```

1. Avalie o bloco PL/SQL acima e determine o tipo de dados e o valor de cada uma das variáveis a seguir de acordo com as regras de criação de escopos.
   a. O valor de V_WEIGHT no sub-bloco é:

   b. O valor de V_NEW_LOCN no sub-bloco é:

   c. O valor de V_WEIGHT no bloco principal é:

   d. O valor de V_MESSAGE no bloco principal é:

   e. O valor de V_NEW_LOCN no bloco principal é:

Exercício 17 (continuação)

Exemplo de Escopo

```
DECLARE
  v_customer         VARCHAR2(50) := 'Womansport';
  v_credit_rating    VARCHAR2(50) := 'EXCELLENT';
BEGIN
  DECLARE
    v_customer   NUMBER(7) := 201;
    v_name  VARCHAR2(25) := 'Unisports';
  BEGIN
   v_customer      v_name   v_credit_rating
  END;

   v_customer      v_name   v_credit_rating
END;
```

2. Suponha que você tenha embutido um sub-bloco em um bloco, como mostrado acima. Você declara duas variáveis, V_CUSTOMER e V_CREDIT_RATING, no bloco principal. Você também declara duas variáveis, V_CUSTOMER e V_NAME, no sub-bloco. Determine os valores e tipos de dados para cada um dos casos a seguir.
   a. O valor de V_CUSTOMER no sub-bloco é:

   b. O valor de V_NAME no sub-bloco é:

   c. O valor de V_CREDIT_RATING no sub-bloco é:

   d. O valor de V_CUSTOMER no bloco principal é:

   e. O valor de V_NAME no bloco principal é:

   f. O valor de V_CREDIT_RATING no bloco principal é:

Exercício 17 (continuação)

3. Crie e execute um bloco PL/SQL que aceite dois números através das variáveis de substituição do SQL*Plus. O primeiro número deve ser dividido pelo segundo e ter o segundo adicionado ao resultado. O resultado deve ser armazenado em uma variável PL/SQL e impresso na tela, ou o resultado deve ser gravado na variável SQL*Plus e impresso na tela.

   a. Quando uma variável PL/SQL é usada:

```
Please enter the first number: 2
Please enter the second number: 4
4.5
PL/SQL procedure successfully completed.
```

   b. Quando uma variável SQL*Plus é usada:

```
Please enter the first number: 2
Please enter the second number: 4

PL/SQL procedure successfully completed.

  G_RESULT
  --------
      4.5
```

4. Elabore um bloco PL/SQL que compute a remuneração total por um ano. O salário anual e a porcentagem do bônus anual são repassados ao bloco PL/SQL pelas variáveis de substituição SQL*Plus e os benefícios precisam ser convertidos de um número inteiro para um decimal (por exemplo, 15 para 0.15). Se o salário for nulo, defina-o para zero antes de computar a remuneração total. Execute o bloco PL/SQL. *Aviso:* Use a função NVL para tratar os valores nulos.

   Observação: Para testar a função NVL, digite NULL no prompt; o pressionamento de [Return] resulta em um erro de expressão ausente.

```
Please enter the salary amount: 50000
Please enter the bonus percentage: 10

PL/SQL procedure successfully completed.

G_TOTAL
-------
  55000
```

18

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

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;
```


Diretrizes (continuação)

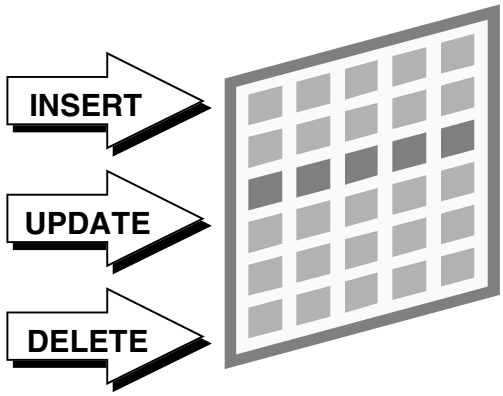
- Para garantir que os tipos de dados dos identificadores correspondam aos tipos de dados das colunas, use o atributo %TYPE. O tipo de dados e o número de variáveis na cláusula INTO correspondem àqueles na lista SELECT.
- Use as funções de grupo como, por exemplo, SUM, em uma instrução SQL, porque as 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

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

- **INSERT**
- **UPDATE**
- **DELETE**



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 subseqüente 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 subseqüente.

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
  DELETE FROM  item
  WHERE        ordid = v_ordid;
  :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 desfaça 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.

Visão Geral do Exercício

- **Criar um bloco PL/SQL para selecionar dados de tabela**
- **Criar um bloco PL/SQL para inserir dados em uma tabela**
- **Criar um bloco PL/SQL para atualizar dados em uma tabela**
- **Criar um bloco PL/SQL para deletar um registro de uma tabela**

Visão Geral do Exercício

Neste exercício, você deve criar procedimentos para selecionar, entrar, atualizar e deletar informações em uma tabela, usando as instruções DML e a consulta SQL básica em um bloco PL/SQL.

Exercício 18

1. Criar um bloco PL/SQL que seleciona o número máximo de departamento na tabela DEPT e o armazena em uma variável SQL*Plus. Imprima os resultados na tela. Salve o bloco PL/SQL em um arquivo nomeado `p18q1.sql`.

```
G_MAX_DEPTNO
------------
          40
```

2. Modificar o bloco PL/SQL que você criou no exercício 1 para inserir um novo departamento na tabela DEPT. Salve o bloco PL/SQL em um arquivo nomeado `p18q2.sql`.
   a. Em vez de imprimir o número do departamento recuperado do exercício 1, adicione 10 a ele e use-o como o número do departamento do novo departamento.
   b. Use uma variável de substituição do SQL*Plus para o número do departamento.
   c. Deixe um valor nulo na localização por enquanto.
   d. Execute o bloco PL/SQL.

```
Please enter the department name: EDUCATION

PL/SQL procedure successfully completed.
```

   e. Exiba o novo departamento criado.

```
DEPTNO DNAME      LOC
------ ---------- -----
    50 EDUCATION
```

3. Crie um bloco PL/SQL que atualize a localização para um departamento existente. Salve o bloco PL/SQL em um arquivo denominado `p18q3.sql`.
   a. Use uma variável de substituição do SQL*Plus para o número de departamento.
   b. Use uma variável de substituição do SQL*Plus para a localização de departamento.
   c. Teste o bloco PL/SQL.

```
Please enter the department number: 50
Please enter the department location: HOUSTON

PL/SQL procedure successfully completed.
```

Exercício 18 (continuação)

d. Exiba o nome e o número do departamento, além da localização para o departamento atualizado.

```
DEPTNO DNAME      LOC
------ ---------- ---------------
    50 EDUCATION HOUSTON
```

e. Exiba o departamento que você atualizou.

4. Crie um bloco PL/SQL que delete o departamento criado no exercício 2. Salve o bloco PL/SQL em um arquivo denominado `p18q4.sql`.

   a. Use uma variável de substituição do SQL*Plus para o número do departamento.

   b. Imprima o número de linhas afetadas na tela.

   c. Teste o bloco PL/SQL.

```
Please enter the department number: 50
PL/SQL procedure successfully completed.

G_RESULT
-------------------------------------------------------------
1 row(s) deleted.
```

   d. O que acontece se você informar um número de departamento que não existe?

```
Please enter the department number: 99
PL/SQL procedure successfully completed.

G_RESULT
-------------------------------------------------------------
0 row(s) deleted.
```

   e. Confirme se o departamento foi deletado.

```
no rows selected
```

19

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

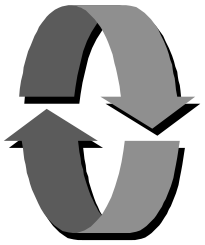
Nesta lição, você aprenderá sobre o controle condicional dentro do bloco PL/SQL usando loops e instruções IF.

Controlando o Fluxo de Execução PL/SQL

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;
```

  ORACLE®

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;
. . .
```

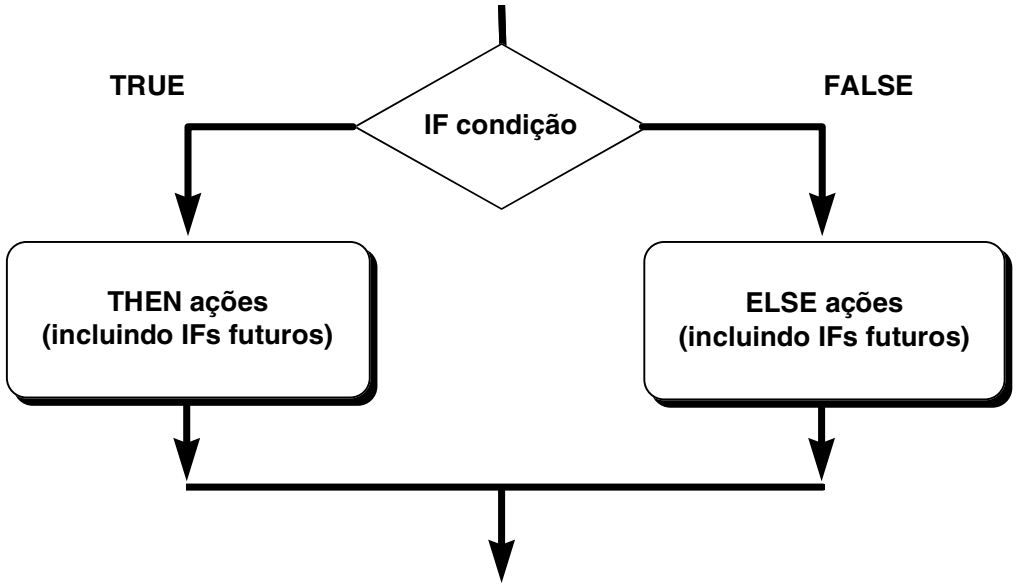

Instruções IF Simples

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

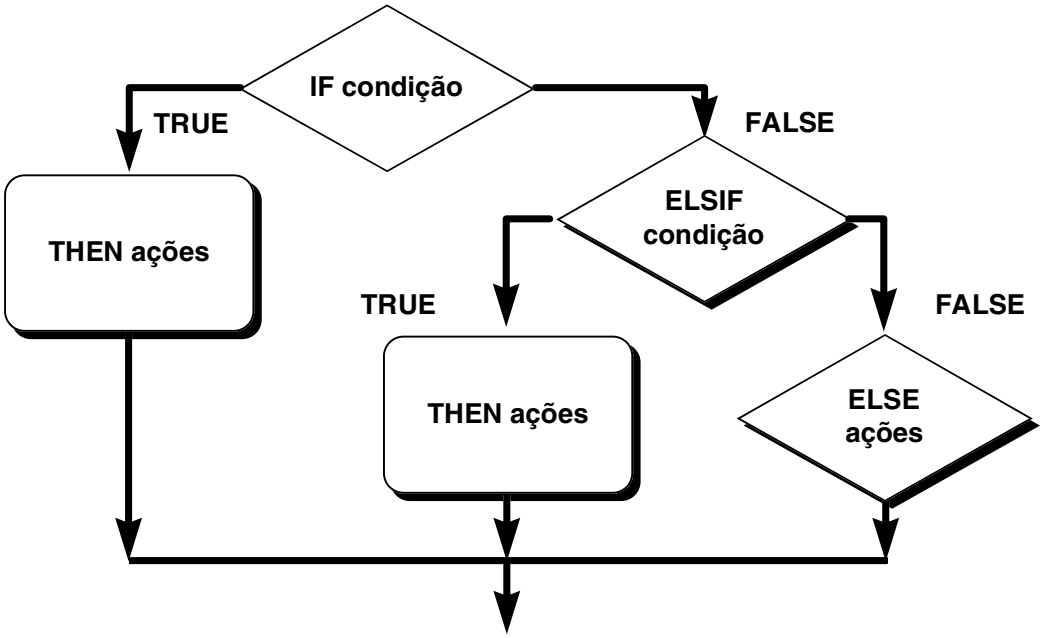
```
...
IF v_shipdate - v_orderdate < 5 THEN
   v_ship_flag := 'Aceitável';
ELSE
   v_ship_flag := 'Inaceitável';
END IF;
...
```


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.**

  ORACLE®

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

Desenvolver uma condição Booleana simples com um operador de comparação.

AND	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

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;
```

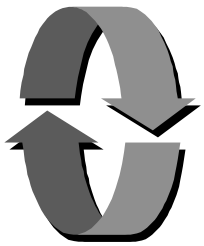
V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
NULL	TRUE	NULL
NULL	FALSE	FALSE

Desenvolvendo Condições Lógicas

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

Controle Iterativo: Instruções LOOP

- **Loops repetem uma instrução ou seqüência 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 seqüência 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 subseqüente.

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 Basico

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.**

  ORACLE®

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 seqüência 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 seqüência 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

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.

Loop WHILE

Você pode usar o loop WHILE para repetir uma seqüência 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;
/
```

  ORACLE®

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;
```

  ORACLE®

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.

Visão Geral do Exercício

- **Executando ações condicionais usando a instrução IF**
- **Executando etapas iterativas usando a estrutura de loop**

Visão Geral do Exercício

Neste exercício, você cria blocos PL/SQL que incorporam estruturas para controle condicional e loops.

Exercício 19

1. Execute o script `lab19_1.sql` para criar a tabela MESSAGES. Crie um bloco PL/SQL para inserir números na tabela MESSAGES.
   a. Insira os números de 1 a 10, excluindo 6 e 8.
   b. Efetue um commit antes do final do bloco.
   c. Realize seleções na tabela MESSAGES para verificar se o bloco PL/SQL funcionou.

```
RESULTS
---------
1
2
3
4
5
7
9
10
```

2. Crie um bloco PL/SQL que compute o valor da comissão de um determinado funcionário com base no salário do funcionário.
   a. Execute o script `lab19_2.sql` para inserir um novo funcionário na tabela EMP. **Observação:** O funcionário terá um salário NULL.
   b. Aceite o número do funcionário como entrada do usuário com uma variável de substituição do SQL*Plus.
   c. Se o salário do funcionário for inferior a US$ 1.000, defina o valor da comissão do funcionário para 10% do salário.
   d. Se o salário do funcionário estiver entre US$ 1.000 e US$ 1.500, defina o valor da comissão do funcionário para 15% do salário.
   e. Se o salário do funcionário exceder US$ 1.500, defina o valor da comissão do funcionário para 20% do salário.
   f. Se o salário do funcionário for NULL, defina o valor da comissão do funcionário para 0.
   g. Efetue um commit.
   h. Teste o bloco PL/SQL para cada caso usando os casos de teste a seguir e verifique cada comissão atualizada.

Número do Funcionário	Salário	Commissão Resultante
7369	800	80
7934	1300	195
7499	1600	320
8000	NULL	0

Exercício 19 (continuação)

```
EMPNO ENAME     SAL       COMM
----- ------ ----- ---------
 8000 DOE                   0
 7499 ALLEN   1600        320
 7934 MILLER  1300        195
 7369 SMITH    800         80
```

Se você tiver tempo, complete os exercícios abaixo:

3. Modifique o arquivo `p19q4.sql` para inserir o texto "Número é ímpar" ou "Número é par", dependendo se o valor for impar ou par, na tabela MESSAGES. Consulte a tabela MESSAGES para determinar se o bloco PL/SQL funcionou.

```
RESULTS
---------------
Número é par
```

4. Adicione uma nova coluna chamada STARS, de tipo de dado VARCHAR2 e comprimento 50, à tabela EMP para armazenar asteriscos (*).
5. Crie um bloco PL/SQL que premie um funcionário, anexando um asterisco à coluna STARS para cada US$ 100 do salário do funcionário. Salve o bloco PL/SQL em um arquivo nomeado `p19q5.sql`.

   a. Aceite o ID do funcionário como entrada do usuário com uma variável de substituição do SQL*Plus.

   b. Inicialize uma variável que conterá uma string de asteriscos.

   c. Anexe um asterisco à string para cada US$ 100 do salário. Por exemplo, se o funcionário recebe um salário de US$ 800, a string de asteriscos deverá conter oito asteriscos. Se o funcionário recebe um salário de US$ 1.250, a string de asteriscos deve conter 13 asteriscos.

   d. Atualize a coluna STARS do funcionário com a string de asteriscos.

   e. Efetue um commit.

   f. Teste o bloco dos funcionários que não têm salário e de um funcionário que tem um salário.

```
Informe o número do funcionário: 7934
  Procedimento PL/SQL concluído corretamente.
  Informe o número do funcionário: 8000
  Procedimento PL/SQL concluído corretamente.

  EMPNO    SAL STARS
  ----- ------ ----------------
   8000
   7934   1300 *************
```

20

Trabalhando com Tipos de Dados Compostos

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.

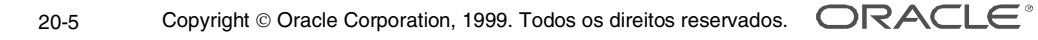
Criando um Registro PL/SQL

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

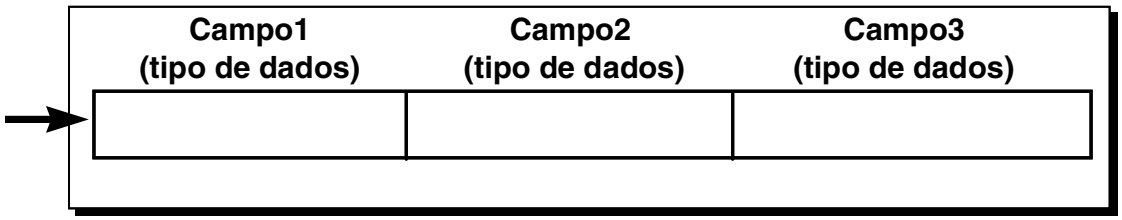
Declarações de campo são semelhantes a declarações de variável. Cada campo tem um nome exclusivo e um tipo de dados específico. Não existem tipos de dados predefinidos para registros PL/SQL, como existem para variáveis escalares. Assim, você deve primeiro criar o tipo de dados e, em seguida, declarar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

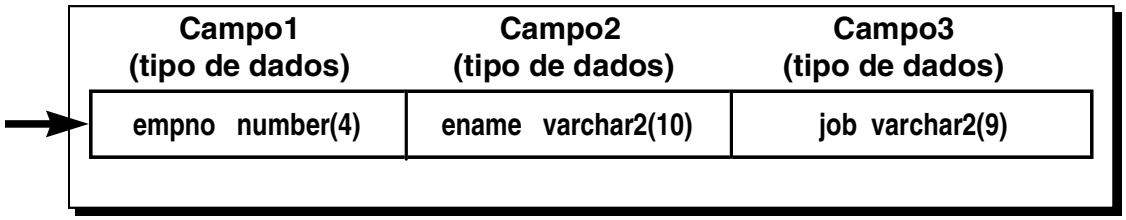
```
DECLARE
  TYPE emp_record_type IS RECORD
    (empno      NUMBER(4) NOT NULL := 100,
     ename      emp.ename%TYPE,
     job        emp.job%TYPE);
  emp_record  emp_record_type;
...
```

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



Exemplo



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_emps(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

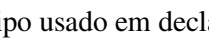
Sintaxe

```
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:

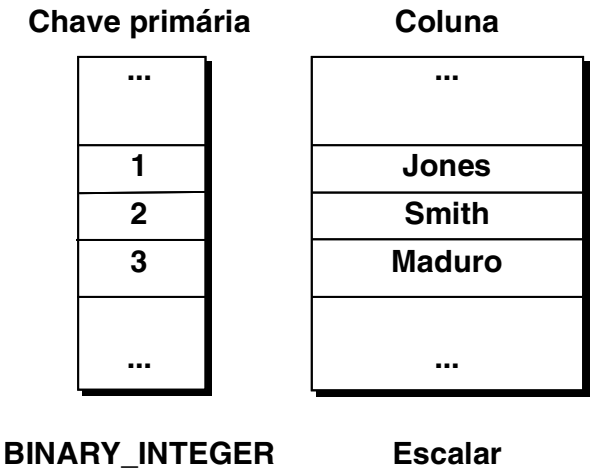
type_name	é o nome do tipo TABLE. (Ele é um especificador de tipo usado em declarações subseqüentes de tabelas PL/SQL.)
column_type	é qualquer tipo de dados escalares (não composto), como VARCHAR2, DATE ou NUMBER. (Você pode usar o atributo %TYPE para fornecer o tipo de dados da coluna.)
identificador	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**

  ORACLE®

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(*n*)	Retornará TRUE se o enésimo elemento em uma tabela PL/SQL existir.
COUNT	Retorna o número de elementos contidos uma tabela PL/SQL atualmente.
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(*n*)	Retorna o número de índice que precede o índice *n* em uma tabela PL/SQL.
NEXT(*n*)	Retorna o número do índice que sucede o índice *n* em uma tabela PL/SQL.
EXTEND(*n*, *i*)*	Aumenta o tamanho de uma tabela PL/SQL. EXTEND anexa um elemento nulo a uma tabela PL/SQL. EXTEND(*n*) anexa *n* elementos nulos a uma tabela PL/SQL. EXTEND(*n*, *i*) anexa *n* cópias do elemento *i* a uma tabela PL/SQL.
TRIM*	TRIM remove um elemento do final de uma tabela PL/SQL. TRIM(*n*) remove *n* elementos do final de uma tabela PL/SQL.
DELETE	DELETE remove todos os elementos de uma tabela PL/SQL. DELETE(*n*) remove o enésimo elemento de uma tabela PL/SQL. DELETE(*m*, *n*) remove todos os elementos na faixa *m* ... *n* 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.

Visão Geral do Exercício

- **Declarando tabelas PL/SQL**
- **Processando dados usando tabelas PL/SQL**
- **Declarando um registro PL/SQL**
- **Processando dados usando um registro PL/SQL**

Visão Geral do Exercício

Nesta prática, você define, cria e usa tabelas PL/SQL e um registro PL/SQL.

Exercício 20

1. Crie um bloco PL/SQL para recuperar o nome de cada departamento da tabela DEPT e imprima o nome de cada departamento na tela, incorporando uma tabela PL/SQL.
   a. Declare uma tabela PL/SQL, MY_DEPT_TABLE, para armazenar temporariamente o nome de departamentos
   b. Usando um loop, recupere o nome de todos os departamentos atualmente na tabela DEPT e armazene-os na tabela PL/SQL. Cada número de departamento é um múltiplo de 10.
   c. Usando outro loop, recupere os nomes de departamentos da tabela PL/SQL e imprima-os na tela, usando DBMS_OUTPUT.PUT_LINE.

```
SQL> START p20_1
ACCOUNTING
RESEARCH
SALES
OPERATIONS


Procedimento PL/SQL concluído corretamente.
```

2. Crie um bloco PL/SQL para imprimir as informações sobre um determinado pedido.
   a. Declare um registro PL/SQL com base na estrutura da tabela ORD.
   b. Use uma variável de substituição do SQL*Plus para recuperar todas as informações sobre um pedido específico e armazene-as no registro PL/SQL.
   c. Use DBMS_OUTPUT. PUT_LINE para imprimir informações selecionadas sobre o pedido.

```
SQL> START p20_2
Informe um número de pedido: 614
Pedido 614 colocado em 01-FEB-87 e enviado em 05-FEB-87 para um
total de  US$23,940.00


Procedimento PL/SQL concluído corretamente.
```

Exercício 20 (continuação)

Se você tiver tempo, complete o exercício abaixo.

3. Modifique o bloco criado na prática 1 para recuperar todas as informações sobre cada departamento da tabela DEPT e imprima as informações na tela, incorporando uma tabela de registros PL/SQL.
   a. Declare uma tabela PL/SQL, MY_DEPT_TABLE, para armazenar temporariamente o número, o nome e a localização de todos os departamentos.
   b. Usando um loop, recupere as informações de todos os departamentos atualmente na tabela DEPT e armazene-as na tabela PL/SQL. Cada número de departamento é um múltiplo de 10.
   c. Usando outro loop, recupere as informações de departamento da tabela PL/SQL e imprima-as na tela, usando DBMS_OUTPUT.PUT_LINE.

```
SQL> START p20_3
Dept. 10, ACCOUNTING está localizado em NEW YORK
Dept. 20, RESEARCH está localizado em DALLAS
Dept. 30, SALES está localizado em CHICAGO
Dept. 40, OPERATIONS está localizado em BOSTON

Procedimento PL/SQL concluído corretamente.
```

21

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.

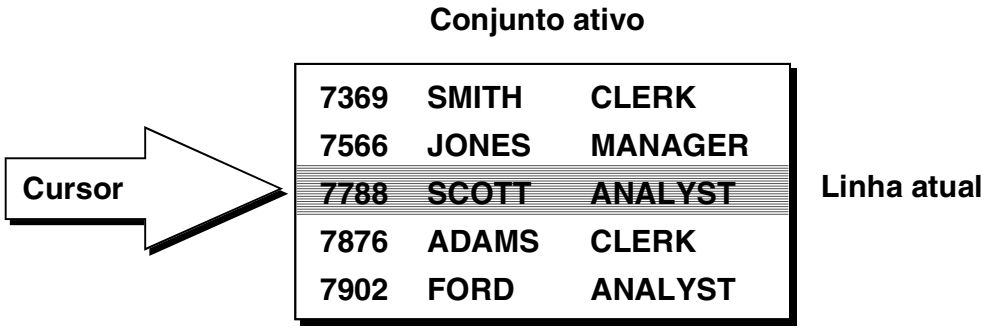
Tipo de Cursor	Descrição
Implícito	Os cursores implícitos são declarados implicitamente pelo PL/SQL para todas as instruções DML e PL/SQL SELECT, incluindo consultas que retornam somente uma linha.
Explícito	Para consultas que retornam mais de uma linha. Os cursores explícitos são declarados e nomeados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



Cursores 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.

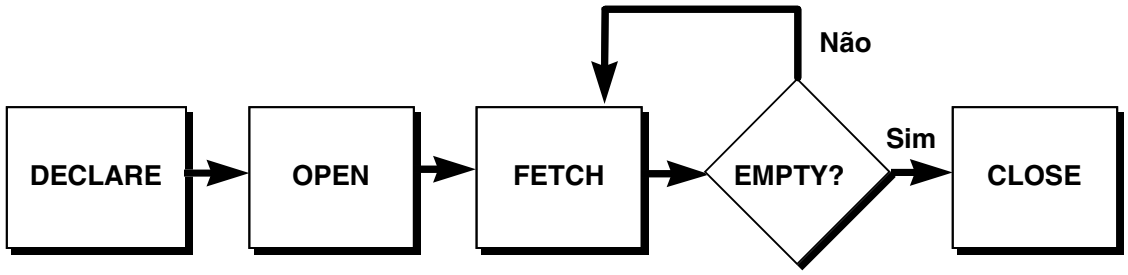
Um programa PL/SQL abre um cursor, processa linhas retornadas por uma consulta e, em seguida, fecha o cursor. O cursor marca a posição atual no conjunto ativo.

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



- Cria uma área SQL nomeada
- Identifica o conjunto ativo
- Carrega a linha atual para variáveis
- Testa para linhas existentes
- Retorna para FETCH se encontrar linhas
- Libera o conjunto ativo

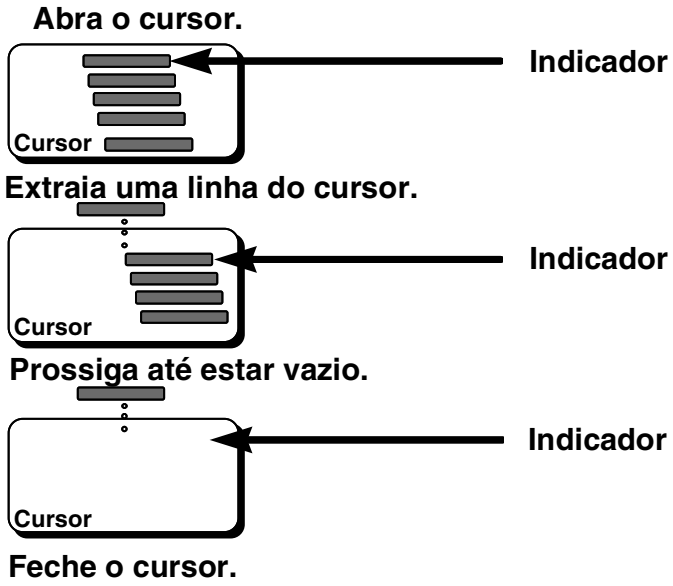
Cursores Explícitos (continuação)

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



Cursores 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 seqüência 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.**

  ORACLE®

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.

Cursores 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;
  ...
```


Cursores 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 cusor 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 subseqüente.
- 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.**

  ORACLE®

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%TYPE;
  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;
/
```

Visão Geral do Exercício

- **Declarando e usando cursores explícitos para consultar linhas de uma tabela**
- **Usando um loop FOR de cursor**
- **Aplicando atributos de cursor para testar o status do cursor**

Visão Geral do Exercício

Este exercício aplica o seu conhecimento sobre cursores para processar um número de linhas de uma tabela e preencher outra tabela com os resultados usando um loop FOR de cursor.

Exercício 21

1. Execute o script `lab21_1.sql` para criar uma nova tabela para armazenar funcionários e seus salários.

```
SQL> CREATE TABLE    top_dogs
   2  (name          VARCHAR2(25),
   3   salary        NUMBER(11,2));
```

2. Crie um bloco PL/SQL que determine os funcionários com os maiores salários.
   a. Aceite um número *n* como entrada de usuário com um parâmetro de substituição do SQL*Plus.
   b. Em um loop, obtenha os sobrenomes e salários das *n* pessoas com maiores salários da tabela EMP.
   c. Armazene os nomes e salários na tabela TOP_DOGS.
   d. Pressuponha que não existam dois funcionários com salários iguais.
   e. Teste diversos casos especiais, como *n* = 0 ou onde *n* seja maior do que o número de funcionários na tabela EMP. Esvazie a tabela TOP_DOGS depois de cada teste.

```
Please enter the number of top money makers: 5

NAME         SALARY

----------- ------
KING           5000
FORD           3000
SCOTT          3000
JONES          2975
BLAKE          2850
```

3. Considere o caso em que vários funcionários recebem o mesmo salário. Se uma pessoa estiver listada, todos que tiverem o mesmo salário também deverão estar listados.
   a. Por exemplo, se o usuário informar um valor 2 para *n*, King, Ford e Scott deverão ser exibidos. (Esses funcionários são reunidos pelo segundo maior salário.)
   b. Se o usuário informar um valor 3, King, Ford, Scott e Jones deverão ser exibidos.
   c. Delete todas as linhas de TOP_DOGS e teste o exercício.

```
Please enter the number of top money makers : 2
NAME          SALARY
------------ ------
KING            5000
FORD            3000
SCOTT           3000
```

Exercício 21 (continuação)

```
Please enter the number of top money makers : 3
NAME         SALARY
------------ ------
KING           5000
FORD           3000
SCOTT          3000
JONES          2975
```

22

Conceitos de Cursor Explícito Avançados

Objetivos

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

- **Criar um cursor que utilize parâmetros**
- **Determinar quando uma cláusula FOR UPDATE em um cursor é necessária**
- **Determinar quando usar a cláusula WHERE CURRENT OF**
- **Criar um cursor que utiliza uma subconsulta**

Objetivo da Lição

Nesta lição, você aprenderá mais sobre como criar cursores explícitos, especificamente sobre como criar cursores que utilizam parâmetros.

Cursores 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.**

Cursores 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 feito 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.**

  ORACLE®

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**

  ORACLE®

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.

Visão Geral do Exercício

- **Declarando e usando cursores explícitos com parâmetros**
- **Usando um cursor FOR UPDATE**

Visão Geral do Exercício

Este exercício aplica o seu conhecimento sobre cursores com parâmetros para processar diversas linhas de várias tabelas.

Exercício 22

1. Use um cursor para recuperar o número e o nome do departamento a partir da tabela DEPT. Passe o número do departamento para outro cursor recuperar os detalhes sobre o nome do funcionário, cargo, data de admissão e salário de todos os funcionários que trabalham naquele departamento a partir da tabela EMP.

```
Department Number : 10  Department Name : ACCOUNTING

KING    PRESIDENT   17-NOV-81   5000
CLARK    MANAGER   09-JUN-81   2450
MILLER    CLERK   23-JAN-82   1300

Department Number : 20  Department Name : RESEARCH

JONES    MANAGER   02-APR-81   2975
FORD    ANALYST   03-DEC-81   3000
SMITH    CLERK   17-DEC-80   800
SCOTT    ANALYST   09-DEC-82   3000
ADAMS    CLERK   12-JAN-83   1100

Department Number : 30  Department Name : SALES

BLAKE    MANAGER   01-MAY-81   2850
MARTIN    SALESMAN   28-SEP-81   1250
ALLEN    SALESMAN   20-FEB-81   1600
TURNER    SALESMAN   08-SEP-81   1500
JAMES    CLERK   03-DEC-81   950
WARD    SALESMAN   22-FEB-81   1250

Department Number : 40  Department Name : OPERATIONS
```

2. Modifique o `p19q5.sql` para incorporar a funcionalidade de FOR UPDATE e WHERE CURRENT OF no processamento do cursor.

```
  EMPNO    SAL STARS
  ----- ------ ----------------------
   8000
   7900    950 **********
   7844   1500 ***************
```

23

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.**

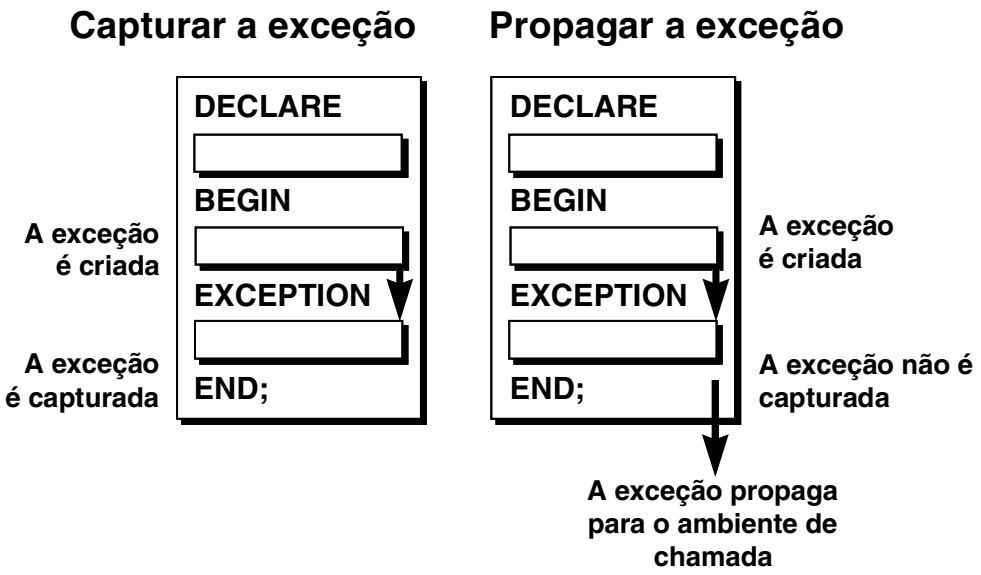
Visão Geral

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

- **Predefinida pelo Oracle Server**
- **Não predefinida pelo Oracle Server**

} **Criada implicitamente**

- **Definida pelo usuário** — **Criada explicitamente**

  ORACLE®

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 freqüência 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;
    . . .]
```

  ORACLE®

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 seqüência 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

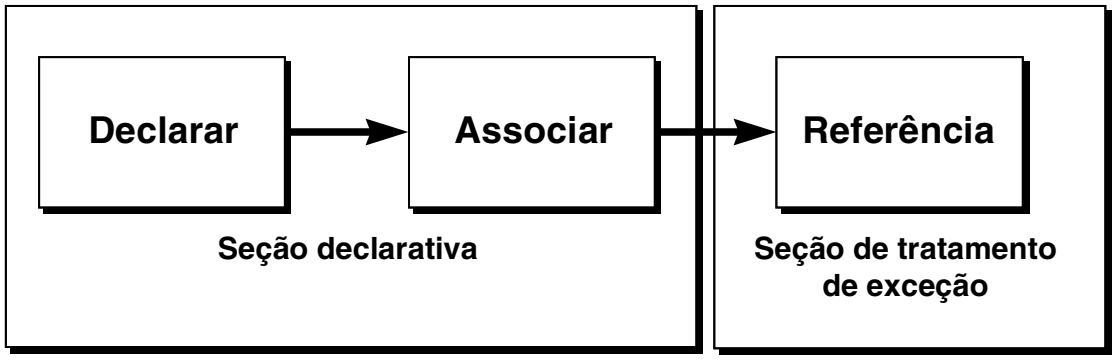
Sintaxe

```
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



- Nomeie a exceção
- Codifique o PRAGMA EXCEPTION_INIT
- Trate a exceção criada

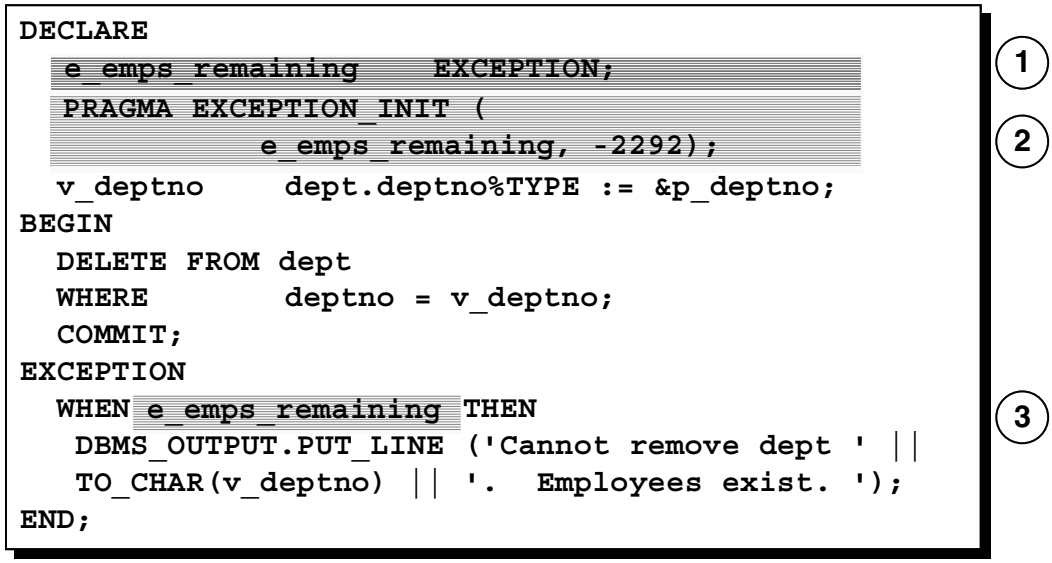
Capturando Erros Não Predefinidos do Oracle Server

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

Capturar por número de erro do Oracle Server –2292, uma violação de restrição de integridade.



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 subseqüente 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

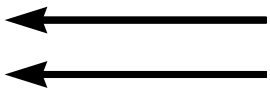
Exemplo de Valores SQLCODE

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

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;
```



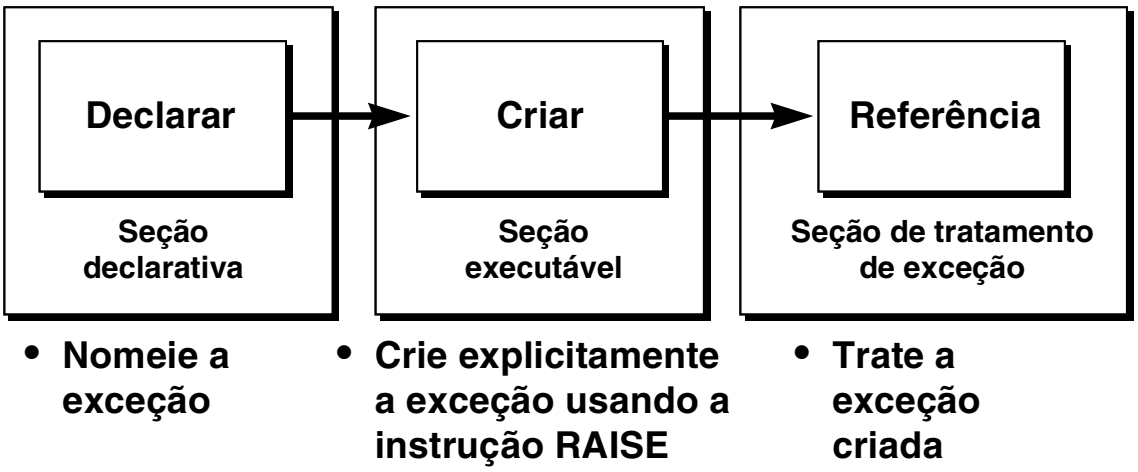
Funções para Captura de Erro

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

Exemplo

```
DECLARE
  e_invalid_product  EXCEPTION;                          1
BEGIN
  UPDATE     product
  SET        descrip = '&product_description'
  WHERE      prodid = &product_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_product;                             2
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_product THEN                            3
    DBMS_OUTPUT.PUT_LINE('Invalid product number.');
END;
```


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

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**

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

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;
```


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.

Visão Geral do Exercício

- **Tratando exceções nomeadas**
- **Criando e chamando exceções definidas pelo usuário**

Visão Geral do Exercício

Neste exercício, você cria handlers de exceção para situações específicas.

Exercício 23

1. Crie um bloco PL/SQL para selecionar o nome do funcionário com um determinado salário.
   a. Se o salário informado retornar mais de uma linha, trate a exceção com um handler de exceção apropriado e insira na tabela MESSAGES a mensagem "More than one employee with a salary <*salary*>".
   b. Se o salário informado não retornar qualquer linha, trate a exceção com um handler de exceção apropriado e insira na tabela MESSAGES a mensagem "No employee with a salary of <*salary*>".
   c. Se o salário informado retornar somente uma linha, insira na tabela MESSAGES o nome do funcionário e o valor do salário.
   d. Trate quaisquer outras exceções com um handler de exceção apropriado e insira na tabela MESSAGES a mensagem "Some other error ocurred".
   e. Teste o bloco para uma variedade de casos de teste.

```
RESULTS
-------------------------------------------
SMITH - 800
More than one employee with a salary of 3000
No employee with a salary of 6000
```

2. Modifique o `p18q3.sql` para adicionar um handler de exceção.
   a. Crie um handler de exceção para o erro passar uma mensagem ao usuário de que o departamento especificado não existe.
   b. Execute o bloco PL/SQL informando um departamento que não existe.

```
Please enter the department number: 50
Please enter the department location: HOUSTON
PL/SQL procedure successfully completed.

G_MESSAGE
-------------------------------------
Department 50 is an invalid department
```

3. Crie um bloco PL/SQL que imprima o número de funcionários que recebem mais ou menos US$ 100 do salário fornecido.
   a. Se não houver funcionário nessa faixa de salário, imprima uma mensagem para o usuário indicando que é esse o caso. Use uma exceção para esse caso.
   b. Se houver um ou mais funcionários dentro dessa faixa, a mensagem deverá indicar quantos funcionários estão naquela faixa salarial.
   c. Trate quaisquer outras exceções com um handler de exceção apropriado. A mensagem deve indicar que ocorreu algum outro erro.

Exercício 23 (continuação)

```
Please enter the salary: 800
PL/SQL procedure successfully completed.

G_MESSAGE
------------------------------------------------------------
There is/are 1 employee(s) with a salary between 700 and 900

Please enter the salary: 3000
PL/SQL procedure successfully completed.

G_MESSAGE
------------------------------------------------------------
There is/are 3 employee(s) with a salary between 2900 and 3100

Please enter the salary: 6000
PL/SQL procedure successfully completed.

G_MESSAGE
------------------------------------------------
There is no employee salary between 5900 and 6100
```

A

Soluções de Exercícios

Soluções de Exercícios 1

1. Inicie uma sessão SQL*Plus usando um ID e uma senha de usuário fornecidos pelo instrutor.
2. Os comandos SQL*Plus acessam o banco de dados.
   Falso
3. A instrução SELECT será executada corretamente?
   Verdadeiro

```
SQL> SELECT   ename, job, sal Salary
  2  FROM     emp;
```

4. A instrução SELECT será executada corretamente?
   Verdadeiro

```
SQL> SELECT *
  2  FROM   salgrade;
```

5. Há quatro erros de codificação nesta instrução. Você pode identificá-los?

```
SQL> SELECT      empno, ename
  2              salary x 12 ANNUAL SALARY
  3  FROM        emp;
```

   - **A tabela EMP não contém uma coluna chamada salary. A coluna é chamada sal.**
   - **O operador de multiplicação é *, não x, como mostrado na linha 2.**
   - **O apelido ANNUAL SALARY não pode incluir espaços. O apelido deve ser ANNUAL_SALARY ou estar entre aspas duplas.**
   - **Falta uma vírgula após a coluna ENAME.**

6. Mostre a estrutura da tabela DEPT. Selecione todos os dados da tabela DEPT.

```
SQL> DESCRIBE dept
SQL> SELECT *
  2  FROM   dept;
```

7. Mostre a estrutura da tabela EMP. Crie uma consulta para exibir o nome, o cargo, a data de admissão e o número de cada funcionário, com o número do funcionário aparecendo primeiro. Salve a instrução SQL em um arquivo nomeado `p1q7.sql`.

```
SQL> DESCRIBE emp
SQL> SELECT empno, ename, job, hiredate
  2  FROM   emp;
SQL> SAVE p1q7.sql
Created file p1q7.sql
```

Soluções de Exercícios 1 (continuação)

8. Execute a consulta no arquivo `p1q7.sql`.

```
SQL> START p1q7.sql
```

9. Crie uma consulta para exibir os cargos exclusivos a partir da tabela EMP.

```
SQL> SELECT DISTINCT job
  2  FROM   emp;
```

Se você tiver tempo, complete os exercícios abaixo:

10. Carregue o `p1q7.sql` no buffer de SQL. Nomeie os cabeçalhos das colunas como Emp #, Employee, Job, e Hire Date, respectivamente. Execute novamente a consulta.

```
SQL> GET p1q7.sql
  1  SELECT empno, ename, job, hiredate
  2* FROM   emp
SQL> 1 SELECT    empno "Emp #", ename "Employee",
SQL> i
 2i  job "Job", hiredate "Hire Date"
 3i
SQL> SAVE p1q7.sql REPLACE
Created file p1q7.sql
SQL> START p1q7.sql
```

11. Exiba o nome concatenado com o cargo, separado por uma vírgula e espaço, e nomeie a coluna Employee and Title.

```
SQL> SELECT      ename||', '||job "Employee and Title"
  2  FROM        emp;
```

Soluções de Exercícios 1 (continuação)

Se você quiser mais desafios, complete o exercício abaixo:

12. Crie uma consulta para exibir todos os dados a partir da tabela EMP. Separe cada coluna por uma vírgula. Nomeie a coluna como THE_OUTPUT.

```
SQL> SELECT empno || ',' || ename || ','|| job || ',' ||
  2         mgr || ',' || hiredate || ',' || sal || ',' ||
  3         comm || ',' || deptno THE_OUTPUT
  4  FROM   emp;
```

Soluções de Exercícios 2

1. Crie uma consulta para exibir o nome e o salário dos funcionários que recebem mais de US$2.850. Salve a instrução SQL em um arquivo nomeado `p2q1.sql`. Execute a consulta.

```
SQL> SELECT   ename, sal
  2  FROM     emp
  3  WHERE    sal > 2850;

SQL> SAVE p2q1.sql
Created file p2q1.sql
```

2. Crie uma consulta para exibir o nome do funcionário e o número do departamento relativos ao número de funcionário 7566.

```
SQL> SELECT   ename, deptno
  2  FROM     emp
  3  WHERE    empno = 7566;
```

3. Modifique o arquivo `p2q1.sql` para exibir o nome e o salário de todos os funcionários cujos salários não estejam na faixa entre US$1.500 e US$2.850. Salve novamente a instrução SQL em um arquivo nomeado `p2q3.sql`. Execute novamente a consulta.

```
SQL> EDIT p2q1.sql

     SELECT   ename, sal
     FROM     emp
     WHERE    sal NOT BETWEEN 1500 AND 2850
     /

SQL> START p2q3.sql
```

4. Exiba o nome do funcionário, o cargo e a data de admissão dos funcionários admitidos entre 20 de fevereiro de 1981 e 1 de maio de 1981. Ordene a consulta de modo crescente pela data inicial.

```
SQL> SELECT   ename, job, hiredate
  2  FROM     emp
  3  WHERE    hiredate BETWEEN
  4  TO_DATE('20-Feb-1981','DD-MON-YYYY') AND
  5  TO_DATE('01-May-1981','DD-MON-YYYY')
  6  ORDER BY hiredate;
```

Soluções de Exercícios 2 (continuação)

5. Exiba o nome do funcionário e o número do departamento de todos os funcionários dos departamentos 10 e 30 em ordem alfabética de nome.

```
SQL> SELECT     ename, deptno
  2  FROM       emp
  3  WHERE      deptno IN (10, 30)
  4  ORDER BY   ename;
```

6. Modifique o arquivo `p2q3.sql` para listar o nome e o salário dos funcionários que recebem mais que US$1.500 e trabalham no departamento 10 ou 30. Coloque um label nas colunas Employee e Monthly Salary, respectivamente. Salve novamente a instrução SQL em um arquivo nomeado `p2q6.sql`. Execute novamente a consulta.

```
SQL> EDIT p2q3.sql
     SELECT    ename "Employee", sal "Monthly Salary"
     FROM      emp
     WHERE     sal  > 1500
     AND       deptno IN (10, 30)
     /
SQL> START p2q6.sql
```

7. Exiba o nome e a data de admissão de todos os funcionários admitidos em 1982.

```
SQL> SELECT    ename, hiredate
  2  FROM      emp
  3  WHERE     hiredate LIKE '%82';
```

8. Exiba o nome e o cargo de todos os funcionários sem um gerente.

```
SQL> SELECT    ename, job
  2  FROM      emp
  3  WHERE     mgr IS NULL;
```

9. Exiba o nome, o salário e a comissão de todos os funcionários que recebem comissão. Classifique os dados em ordem decrescente de salário e comissão.

```
SQL> SELECT    ename, sal, comm
  2  FROM      emp
  3  WHERE     comm IS NOT NULL
  4  ORDER BY  sal DESC, comm DESC;
```

Soluções de Exercícios 2 (continuação)

Se você tiver tempo, complete os exercícios abaixo.

10. Exiba os nomes de todos os funcionários cuja terceira letra do nome seja *A*.
    Observação: Há dois sublinhados (_) antes de *A* na cláusula WHERE.

```
SQL> SELECT   ename
  2  FROM     emp
  3  WHERE    ename LIKE '__A%';
```

11. Exiba os nomes de todos os funcionários cujo nome tem duas letras *L* no departamento 30 ou cujo gerente seja 7782.

```
SQL> SELECT   ename
  2  FROM     emp
  3  WHERE    ename LIKE '%L%L%'
  4  AND      deptno = 30
  5  OR       mgr = 7782;
```

Se você quiser mais desafios, complete os exercícios a seguir.

12. Exiba o nome, o cargo e o salário de todos os funcionários cujo cargo seja Clerk ou Analyst e cujo salário seja diferente de US$1.000, US$3.000 ou US$5.000.

```
SQL> SELECT   ename, job, sal
  2  FROM     emp
  3  WHERE    job IN ('CLERK', 'ANALYST')
  4  AND      sal NOT IN (1000, 3000, 5000);
```

13. Modifique o arquivo `p2q6.sql` para exibir o nome, o salário e a comissão de todos os funcionários cujo valor da comissão é maior que o salário acrescido por 10%. Execute novamente a consulta. Salve novamente a consulta como `p2q13.sql`.

```
SQL> EDIT p2q6.sql
     SELECT   ename "Employee", sal "Monthly Salary", comm
     FROM     emp
     WHERE    comm  > sal * 1.1
     /
SQL> START p2q13.sql
```

Soluções de Exercícios 3

1. Crie uma consulta para exibir a data atual. Coloque um label na coluna Date.

```
SQL> SELECT sysdate "Date"
  2  FROM   dual;
```

2. Exiba o número do funcionário, o nome, o salário e o aumento salarial de 15% expresso como inteiro. Coloque um label na coluna New Salary. Salve a instrução SQL em um arquivo nomeado `p3q2.sql`.

```
SQL> SELECT empno, ename, sal,
  2         ROUND(sal * 1.15, 0) "New Salary"
  3  FROM   emp;


SQL> SAVE p3q2.sql
Created file p3q2.sql
```

3. Execute a consulta no arquivo `p3q2.sql`.

```
SQL> START p3q2.sql
```

4. Modifique a consulta no arquivo `p3q2.sql` para adicionar uma coluna que subtrairá o salário anterior do novo salário. Coloque um label na coluna Increase. Execute novamente a consulta.

```
SQL> EDIT p3q2.sql
     SELECT  empno, ename, sal,
             ROUND(sal * 1.15, 0) "New Salary",
             ROUND(sal * 1.15, 0) - sal "Increase"
     FROM    emp
     /
 SQL> START p3q2.sql
```

5. Exiba o nome do funcionário, a data de admissão e a data de revisão do salário, que é a primeira segunda-feira após seis meses de serviço. Coloque um label na coluna REVIEW. Formate as datas para que sejam exibidas em formato semelhante a "Sunday, the Seventh of September, 1981".

```
SQL> SELECT ename, hiredate,
  2         TO_CHAR(NEXT_DAY(ADD_MONTHS(hiredate, 6),
  3         'MONDAY'),
  4         'fmDay, "the" Ddspth "of" Month, YYYY') REVIEW
  5  FROM   emp;
```

Soluções de Exercícios 3 (continuação)

6. Exiba o nome de cada funcionário e calcule o número de meses entre a data atual e a data em que o funcionário foi admitido. Coloque um label na coluna MONTHS_WORKED. Ordene os resultados pelo número de meses desde a data de admissão. Arredonde o número de meses para o inteiro mais próximo.

```
SQL> SELECT       ename, ROUND(MONTHS_BETWEEN
  2               (SYSDATE, hiredate)) MONTHS_WORKED
  3  FROM         emp
  4  ORDER BY     MONTHS_BETWEEN(SYSDATE, hiredate);
```

7. Crie uma consulta que produza as seguintes informações para cada funcionário: <nome do funcionário> earns <salário> monthly but wants <salário multiplicado por 3>. Coloque um label na coluna Dream Salaries.

```
SQL> SELECT  ename || ' earns '
  2          || TO_CHAR(sal, 'fm$99,999.00')
  3          || ' monthly but wants '
  4          || TO_CHAR(sal * 3, 'fm$99,999.00')
  5          || '.' "Dream Salaries"
  6  FROM    emp;
```

Se você tiver tempo, complete os exercícios abaixo:

8. Crie uma consulta que exiba o nome e o salário de todos os funcionários. Formate o salário para ter 15 caracteres e apresentar o sinal US$ à esquerda. Coloque um label na coluna SALARY.

```
SQL> SELECT  ename,
  2          LPAD(sal, 15, '$') SALARY
  3  FROM    emp;
```

9. Crie uma consulta que exibirá o nome do funcionário com a primeira letra em maiúscula e todas as outras letras em minúsculas, além do tamanho do nome, para todos os funcionários cujo nome começa por J, A ou M. Forneça um label apropriado para cada coluna.

```
SQL> SELECT  INITCAP(ename) "Name",
  2          LENGTH(ename) "Length"
  3  FROM    emp
  4  WHERE   ename LIKE 'J%'
  5  OR      ename LIKE 'M%'
  6  OR      ename LIKE 'A%';
```

Soluções de Exercícios 3 (continuação)

10. Exiba o nome, a data de admissão e o dia da semana em que o funcionário começou a trabalhar. Coloque um label na coluna DAY. Ordene os resultados por dia da semana, iniciando por Monday.

```
SQL> SELECT   ename, hiredate,
  2           TO_CHAR(hiredate, 'DAY') DAY
  3  FROM     emp
  4  ORDER BY TO_CHAR(hiredate - 1, 'd');
```

Se você quiser mais desafios, complete os exercícios abaixo:

11. Crie uma consulta que exibirá o nome do funcionário e o valor da comissão. Se o funcionário não receber comissão, coloque "No Commission". Coloque um label na coluna COMM.

```
SQL> SELECT   ename,
  2           NVL(TO_CHAR(comm), 'No Commission') COMM
  3  FROM     emp;
```

12. Crie uma consulta que exiba os nomes dos funcionários e indique o valor dos salários através de asteriscos. Cada asterisco representa cem dólares. Classifique os dados em ordem decrescente de salário. Coloque um label na coluna EMPLOYEE_AND_THEIR_SALARIES.

```
SQL> SELECT   rpad(ename, 8) ||' '|| rpad(' ', sal/100+1, '*')
  2            EMPLOYEE_AND_THEIR_SALARIES
  3  FROM     emp
  4  ORDER BY sal DESC;
```

13. Crie uma consulta que exiba a classe de todos os funcionários com base no valor da coluna JOB, de acordo com a tabela mostrada abaixo

JOB	*GRADE*
PRESIDENT	A
MANAGER	B
ANALYST	C
SALESMAN	D
CLERK	E
None of the above	O

```
SQL> SELECT job, decode (job,'CLERK',     'E',
  2                           'SALESMAN',  'D',
  3                           'ANALYST',   'C',
  4                           'MANAGER',   'B',
  5                           'PRESIDENT', 'A',
  6                                        '0')GRADE
  7  FROM emp;
```

Soluções de Exercícios 4

1. Crie uma consulta para exibir o nome, o número do departamento e o nome do departamento de todos os funcionários.

```
SQL> SELECT   e.ename, e.deptno, d.dname
  2  FROM     emp e, dept d
  3  WHERE    e.deptno = d.deptno;
```

2. Crie uma lista única de todos os cargos existentes no departamento 30. Inclua a localização do departamento 30 na saída.

```
SQL> SELECT   DISTINCT e.job, d.loc
  2  FROM     emp e, dept d
  3  WHERE    e.deptno = d.deptno
  4  AND      e.deptno = 30;
```

3. Crie uma consulta para exibir o nome do funcionário, o nome do departamento e a localização de todos os funcionários que recebem comissão.

```
SQL> SELECT   e.ename, d.dname, d.loc
  2  FROM     emp e, dept d
  3  WHERE    e.deptno = d.deptno
  4  AND      e.comm IS NOT NULL;
```

4. Exiba o nome do funcionário e o nome do departamento de todos os funcionários que têm *A* no nome. Salve a instrução SQL no arquivo nomeado `p4q4.sql`.

```
SQL> SELECT   e.ename, d.dname
  2  FROM     emp e, dept d
  3  WHERE    e.deptno = d.deptno
  4  AND      e.ename LIKE '%A%';
```

5. Crie uma consulta para exibir o nome, o cargo, o número do departamento e o nome do departamento de todos os funcionários que trabalham em DALLAS.

```
SQL> SELECT   e.ename, e.job, e.deptno, d.dname
  2  FROM     emp e, dept d
  3  WHERE    e.deptno = d.deptno
  4  AND      d.loc = 'DALLAS';
```

Soluções de Exercícios 4 (continuação)

6. Exiba o nome e o número do funcionário junto com o nome e o número do gerente. Coloque um label nas colunas Employee, Emp#, Manager e Mgr#, respectivamente. Salve a instrução SQL em um arquivo chamado `p4q6.sql`.

```
SQL> SELECT   e.ename "Employee", e.empno "Emp#",
  2           m.ename "Manager", m.empno "Mgr#"
  3           FROM       emp e, emp m
  4           WHERE      e.mgr = m.empno;
SQL> SAVE p4q6.sql
Created file p4q6.sql
```

7. Modifique o `p4q6.sql` para exibir todos os funcionários incluindo King, que não possui um gerente. Salve-o novamente como `p4q7.sql`. Execute o `p4q7.sql`.

```
SQL> EDIT p4q6.sql
                SELECT  e.ename "Employee", e.empno "Emp#",
                        m.ename "Manager", m.empno "Mgr#"
                FROM    emp  e, emp m
                WHERE   e.mgr = m.empno(+)
                /
SQL> START p4q7.sql
```

Se você tiver tempo, complete os exercícios abaixo.

8. Crie uma consulta que exibirá o nome do funcionário, o número do departamento e todos os funcionários que trabalham no mesmo departamento que um determinado funcionário. Forneça a cada coluna um label apropriado.

```
SQL> SELECT   e.deptno department, e.ename employee,
  2           c.ename colleague
  3  FROM     emp e, emp c
  4  WHERE    e.deptno = c.deptno
  5  AND      e.empno <> c.empno
  6  ORDER BY e.deptno, e.ename, c.ename;
```

Soluções de Exercícios 4 (continuação)

9. Mostre a estrutura da tabela SALGRADE. Crie uma consulta que exiba o nome, o cargo, o nome do departamento, o salário e a classificação de todos os funcionários.

```
SQL> DESCRIBE salgrade
SQL> SELECT e.ename, e.job, d.dname, e.sal, s.grade
  2  FROM   emp e, dept d, salgrade s
  3  WHERE  e.deptno = d.deptno
  4  AND    e.sal BETWEEN s.losal AND s.hisal;
```

Se você quiser mais desafios, complete os exercícios abaixo:

10. Crie uma consulta para exibir o nome e a data de admissão de qualquer funcionário admitido após o funcionário Blake.

```
SQL> SELECT emp.ename, emp.hiredate
  2  FROM   emp, emp blake
  3  WHERE  blake.ename = 'BLAKE'
  4  AND    blake.hiredate < emp.hiredate;
```

11. Exiba os nomes e as datas de admissão de todos os funcionários junto com o nome do gerente e a data de admissão de todos os funcionários admitidos antes dos respectivos gerentes. Coloque um label nas colunas Employee, Emp Hiredate, Manager e Mgr Hiredate, respectivamente.

```
SQL> SELECT e.ename "Employee", e.hiredate "Emp Hiredate",
  2         m.ename "Manager", m.hiredate "Mgr Hiredate"
  3  FROM   emp e, emp m
  4  WHERE  e.mgr = m.empno
  5  AND    e.hiredate < m.hiredate;
```

Soluções de Exercícios 5

Determine a validade das seguintes instruções. Marque Verdadeiro ou Falso.

1. As funções de grupo trabalham com várias linhas para produzir um resultado.

   Verdadeiro

2. As funções de grupo incluem nulos nos cálculos.

   Falso. As funções de grupo ignoram valores nulos. Se você quiser incluir valores nulos, use a função NVL.

3. A cláusula WHERE restringe as linhas antes da inclusão em um cálculo de grupo.

   Verdadeiro

4. Exiba os salários maior, médio, menor e a soma de todos os salários de todos os funcionários. Coloque um label nas colunas Maximum, Minimum, Sum e Average, respectivamente. Arredonde os resultados para o número inteiro mais próximo. Salve a instrução SQL em um arquivo chamado `p5q4.sql`.

```
SQL> SELECT   ROUND(MAX(sal),0) "Maximum",
  2           ROUND(MIN(sal),0) "Minimum",
  3           ROUND(SUM(sal),0) "Sum",
  4           ROUND(AVG(sal),0) "Average"
  5  FROM     emp;

SQL> SAVE p5q4.sql
Created file p5q4.sql
```

5. Modifique o `p5q4.sql` para exibir o salário maior, médio, menor e a soma de todos os salários para cada tipo de cargo. Salve novamente o arquivo com o nome `p5q5.sql`. Execute novamente a consulta.

```
SQL> EDIT p5q6.sql
     SELECT   job, ROUND(MAX(sal),0) "Maximum",
              ROUND(MIN(sal),0) "Minimum",
              ROUND(SUM(sal),0) "Sum",
              ROUND(AVG(sal),0) "Average"
     FROM     emp
     GROUP BY job
     /
SQL> START p5q5.sql
```

Soluções de Exercícios 5 (continuação)

6. Crie uma consulta para exibir o número de pessoas com o mesmo cargo.

```
SQL> SELECT   job, COUNT(*)
  2  FROM     emp
  3  GROUP BY job;
```

7. Determine o número de gerentes sem listá-los. Coloque um label na coluna Number of Managers.

```
SQL> SELECT   COUNT(DISTINCT mgr) "Number of Managers"
  2  FROM     emp;
```

8. Crie uma consulta para exibir a diferença entre os maiores e menores salários. Coloque um label na coluna DIFFERENCE.

```
SQL> SELECT   MAX(sal) - MIN(sal) DIFFERENCE
  2  FROM     emp;
```

Se você tiver tempo, complete os exercícios abaixo.

9. Exiba o número do gerente e o salário do funcionário com menor pagamento sob a supervisão desse gerente. Exclua todos cujo gerente não seja conhecido. Exclua qualquer grupo cujo salário mínimo seja menor que US$1.000. Classifique a saída em ordem decrescente de salário.

```
SQL> SELECT   mgr, MIN(sal)
  2  FROM     emp
  3  WHERE    mgr IS NOT NULL
  4  GROUP BY mgr
  5  HAVING   MIN(sal) > 1000
  6  ORDER BY  MIN(sal) DESC;
```

10. Crie uma consulta para exibir o nome do departamento, o nome da localização, o número de funcionários e o salário médio de todos os funcionários do departamento. Coloque um label nas colunas dname, loc, Number of People e Salary, respectivamente. Arredonde o salário médio para duas casas decimais apenas.

```
SQL> SELECT   d.dname, d.loc, COUNT(*) "Number of People",
  2           ROUND(AVG(sal),2) "Salary"
  3  FROM     emp e, dept d
  4  WHERE    e.deptno = d.deptno
  5  GROUP BY d.dname, d.loc;
```

Soluções de Exercícios 5 (continuação)

Se você quiser mais desafios, complete os exercícios abaixo:

11. Crie uma consulta que exiba o número total de funcionários e, desse total, o número total de funcionários contratados em 1980, 1981, 1982 e 1983. Coloque os cabeçalhos apropriados nas colunas.

```
SQL> SELECT COUNT(*) total,
  2         SUM(DECODE(TO_CHAR(hiredate, 'YYYY'),
  3                           1980,1,0))"1980",
  4         SUM(DECODE(TO_CHAR(hiredate, 'YYYY'),
  5                           1981,1,0))"1981",
  6         SUM(DECODE(TO_CHAR(hiredate, 'YYYY'),
  7                           1982,1,0))"1982",
  8          SUM(DECODE(TO_CHAR(hiredate, 'YYYY'),
  9                           1983,1,0))"1983"
 10  FROM   emp;
```

12. Crie uma consulta de matriz para exibir o cargo, o salário relativo a esse cargo com base no número do departamento e o total dos salários do cargo em todos os departamentos, fornecendo um cabeçalho apropriado a cada coluna.

```
SQL> SELECT      job "Job",
  2              SUM(DECODE(deptno, 10, sal)) "Dept 10",
  3              SUM(DECODE(deptno, 20, sal)) "Dept 20",
  4              SUM(DECODE(deptno, 30, sal)) "Dept 30",
  5              SUM(sal) "Total"
  6  FROM        emp
  7  GROUP BY    job;
```

Soluções de Exercícios 6

1. Crie uma consulta para exibir o nome do funcionário e a data de admissão de todos os funcionários do mesmo departamento que Blake. Exclua Blake.

```
SQL> SELECT   ename, hiredate
  2  FROM     emp
  3  WHERE    deptno = (SELECT      deptno
  4                     FROM        emp
  5                     WHERE       ename = 'BLAKE')
  6  AND      ename != 'BLAKE';
```

2. Crie uma consulta para exibir o número e o nome de todos os funcionários que recebem mais que o salário médio. Classifique os resultados, por salário, em ordem decrescente.

```
SQL> SELECT   empno, ename
  2  FROM     emp
  3  WHERE    sal > (SELECT AVG(sal)
  4                  FROM emp)
  5  ORDER BY sal DESC;
```

3. Crie uma consulta que exibirá o nome e o número de todos os funcionários que trabalham em um departamento com qualquer funcionário cujo nome contenha um *T*. Salve a instrução SQL em um arquivo chamado `p6q3.sql`.

```
SQL> SELECT   empno, ename
  2  FROM     emp
  3  WHERE    deptno IN (SELECT     deptno
  4                      FROM       emp
  5                      WHERE      ename LIKE '%T%');

SQL> SAVE p6q3.sql
Created file p6q3.sql
```

4. Exiba o nome do funcionário, o número do departamento e o cargo do funcionário cuja localização do departamento seja Dallas.

```
SQL> SELECT   ename, deptno, job
  2  FROM     emp
  3  WHERE    deptno IN (SELECT     deptno
  4                      FROM       dept
  5                      WHERE      loc = 'DALLAS');
```

Soluções de Exercícios 6 (continuação)

5. Exiba o nome e o salário dos funcionários que se reportam a King.

```
SQL> SELECT ename, sal
  2  FROM   emp
  3  WHERE  mgr = (SELECT  empno
  4                FROM    emp
  5                WHERE   ename = 'KING');
```

6. Exiba o número do departamento, o nome e o cargo de todos os funcionários do departamento de Vendas.

```
SQL> SELECT deptno, ename, job
  2  FROM   emp
  3  WHERE  deptno IN (SELECT deptno
  4                    FROM    dept
  5                    WHERE   dname = 'SALES');
```

Se você tiver tempo, complete o exercício abaixo:

7. Modifique `p6q3.sql` para exibir o nome, o número e o salário de todos os funcionários que recebem mais que o salário médio e trabalham com qualquer funcionário que tenha a letra *T* no nome. Salve novamente como `p6q7.sql`. Execute novamente a consulta.

```
SQL> EDIT p6q3.sql
    SELECT  empno, ename, sal
      FROM    emp
      WHERE sal > (SELECT AVG(sal)
                   FROM emp)
                   AND deptno IN (SELECT deptno
                                  FROM    emp
                                  WHERE   ename LIKE '%T%')
     /
SQL> START p6q7.sql
```

Soluções de Exercícios 7

1. Crie uma consulta para exibir o nome, o número do departamento e o salário de qualquer funcionário cujo número do departamento e o salário correspondem ao número do departamento e ao salário de qualquer funcionário que receba comissão.

```
SQL> SELECT   ename, deptno, sal
  2  FROM     emp
  3  WHERE    (sal, deptno) IN
  4                       (SELECT    sal, deptno
  5                        FROM      emp
  6                        WHERE     comm IS NOT NULL);
```

2. Exiba o nome, o nome do departamento e o salário de qualquer funcionário cujo salário e a comissão correspondam ao salário e à comissão de qualquer funcionário localizado em Dallas.

```
SQL> SELECT   ename, dname, sal
  2  FROM     emp e, dept d
  3  WHERE    e.deptno = d.deptno
  4  AND      (sal, NVL(comm,0)) IN
  5                       (SELECT    sal, NVL(comm,0)
  6                        FROM      emp e, dept d
  7                        WHERE     e.deptno = d.deptno
  8                        AND       d.loc = 'DALLAS');
```

3. Crie uma consulta para exibir o nome, a data de admissão e o salário de todos os funcionários que recebem o mesmo salário e a mesma comissão que Scott.

```
SQL> SELECT   ename, hiredate, sal
  2  FROM     emp
  3  WHERE    (sal, NVL(comm,0)) IN
  4                       (SELECT    sal, NVL(comm,0)
  5                        FROM      emp
  6                        WHERE     ename = 'SCOTT')
  7  AND      ename != 'SCOTT';
```

4. Crie uma consulta para exibir os funcionários que recebem um salário maior que o salário de todos os escriturários. Classifique os resultados sobre salários do maior para o menor.

```
SQL> SELECT   ename, job, sal
  2  FROM     emp
  3  WHERE    sal > ALL (SELECT sal
  4                       FROM   emp
  5                       WHERE  job = 'CLERK')
  6  ORDER BY sal DESC;
```

Soluções de Exercícios 8

Determine se as instruções a seguir são verdadeiras ou falsas:

1. Uma variável de substituição de "e" comercial único é pedida no máximo uma vez.

   Falso
   Entretanto, se for definida, a variável de substituição de "e" comercial não será pedida. Na verdade, ela assumirá o valor da variável predefinida.

2. O comando ACCEPT é um comando SQL.

   Falso
   O ACCEPT é um comando SQL*Plus. Ele é emitido no prompt SQL.

3. Crie um arquivo de script para exibir o nome, o cargo e a data de admissão de todos os funcionários admitidos em um intervalo específico. Concatene o nome e o cargo, separados por um espaço e uma vírgula e coloque um label na coluna Employees. Solicite os dois intervalos ao usuário usando o comando ACCEPT. Use o formato MM/DD/YYYY. Salve o arquivo de script como `p8q3.sql`.

```
SET ECHO OFF
SET VERIFY OFF
ACCEPT low_date DATE FORMAT 'MM/DD/YYYY' -
PROMPT 'Please enter the low date range (''MM/DD/YYYY''): '
ACCEPT high_date DATE FORMAT 'MM/DD/YYYY' -
PROMPT 'Please enter the high date range (''MM/DD/YYYY''): '
COLUMN EMPLOYEES FORMAT A25
SELECT        ename ||', '|| job EMPLOYEES, hiredate
FROM          emp
WHERE         hiredate BETWEEN
                  TO_DATE('&low_date', 'MM/DD/YYYY')
              AND TO_DATE('&high_date', 'MM/DD/YYYY')
/
UNDEFINE low_date
UNDEFINE high_date
COLUMN EMPLOYEES CLEAR
SET VERIFY ON
SET ECHO ON
SQL> START p8q3.sql;
```

Soluções de Exercícios 8 (continuação)

4. Crie um script para exibir o nome do funcionário, o cargo e o nome do departamento para uma determinada localização. A condição de pesquisa deve aceitar pesquisas que façam distinção entre maiúsculas e minúsculas para a localização do departamento. Salve o arquivo de script como `p8q4.sql`.

```
SET ECHO OFF
SET VERIFY OFF
ACCEPT p_location PROMPT 'Please enter the location name: '
COLUMN ename HEADING "EMPLOYEE NAME" FORMAT A15
COLUMN dname HEADING "DEPARTMENT NAME" FORMAT A15
SELECT e.ename, e.job, d.dname
FROM   emp e, dept d
WHERE  e.deptno = d.deptno
AND    LOWER(d.loc) LIKE LOWER('%&p_location%')
/
UNDEFINE p_location
COLUMN ename CLEAR
COLUMN dname CLEAR
SET VERIFY ON
SET ECHO ON
SQL> START p8q4.sql
```

Soluções de Exercícios 8 (continuação)

5. Modifique `p8q4.sql` para criar um relatório que contenha o nome do departamento, o nome do funcionário, a data de admissão, o salário e o salário anual de todos os funcionários de uma determinada localização. Peça a localização ao usuário. Coloque um label nas colunas DEPARTMENT NAME, EMPLOYEE NAME, START DATE, SALARY e ANNUAL SALARY, posicionando os labels em várias linhas. Salve novamente o script como `p8q5.sql`.

```
SET ECHO OFF
SET FEEDBACK OFF
SET VERIFY OFF
BREAK ON  dname
ACCEPT p_location PROMPT 'Please enter the location name: '
COLUMN dname HEADING "DEPARTMENT|NAME" FORMAT A15
COLUMN ename HEADING "EMPLOYEE|NAME" FORMAT A15
COLUMN hiredate HEADING "START|DATE" FORMAT A15
COLUMN sal HEADING "SALARY" FORMAT $99,990.00
COLUMN asal HEADING "ANNUAL|SALARY" FORMAT $99,990.00
SELECT    d.dname, e.ename, e.hiredate,
          e.sal,   e.sal * 12 asal
FROM      emp e,   dept d
WHERE     e.deptno = d.deptno
AND       LOWER(d.loc) LIKE LOWER('%&p_location%')
ORDER BY  dname
/
UNDEFINE p_location
COLUMN dname CLEAR
COLUMN ename CLEAR
COLUMN hiredate CLEAR
COLUMN sal CLEAR
COLUMN asal CLEAR
CLEAR BREAK
SET VERIFY ON
SET FEEDBACK ON
SET ECHO  ON
SQL> START p8q5.sql
```

Soluções de Exercícios 9

Insira os dados na tabela MY_EMPLOYEE.

1. Execute o script `lab9_1.sql` para elaborar a tabela MY_EMPLOYEE que será usada para o label.

```
SQL> START lab9_1.sql
```

2. Descreva a estrutura da tabela MY_EMPLOYEE para identificar os nomes de coluna.

```
SQL> DESCRIBE my_employee
```

3. Adicione a primeira linha de dados à tabela MY_EMPLOYEE a partir dos dados de exemplo a seguir. Não liste as colunas na cláusula INSERT.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	795
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audry	aropebur	1550

```
SQL> INSERT INTO my_employee
  2  VALUES (1, 'Patel', 'Ralph', 'rpatel', 795);
```

4. Preencha a tabela MY_EMPLOYEE com uma segunda linha de dados de exemplo da lista anterior. Desta vez, liste as colunas explicitamente na cláusula INSERT.

```
SQL> INSERT INTO my_employee (id, last_name, first_name,
  2                           userid, salary)
  3  VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

5. Confirme a adição à tabela.

```
SQL> SELECT   *
  2  FROM     my_employee;
```

Soluções de Exercícios 9 (continuação)

6. Crie um script nomeado `loademp.sql` para acrescentar linhas à tabela MY_EMPLOYEE interativamente. Peça ao usuário o ID, o nome, o sobrenome e o salário do funcionário. Concatene a primeira letra do primeiro nome e os sete primeiros caracteres do último nome para produzir o ID do usuário.

```
SET ECHO OFF
SET VERIFY OFF
ACCEPT p_id -
PROMPT 'Please enter the employee number: '
ACCEPT p_first_name -
PROMPT 'Please enter the employee''s first name: '
ACCEPT p_last_name -
PROMPT 'Please enter the employee''s last name: '
ACCEPT p_salary PROMPT 'Please enter the employee''s salary:'
INSERT INTO   my_employee
VALUES        (&p_id, '&p_last_name', '&p_first_name',
              lower(substr('&p_first_name', 1, 1) ||
              substr('&p_last_name', 1, 7)), &p_salary)
/
SET VERIFY ON
SET ECHO ON
```

7. Preencha a tabela com as duas linhas de dados de exemplo a seguir, executando o script que você criou.

```
SQL> START loademp.sql
SQL> START loademp.sql
```

8. Confirme as adições à tabela.

```
SQL> SELECT   *
  2  FROM     my_employee;
```

9. Torne essas adições permanentes.

```
SQL> COMMIT;
```

Soluções de Exercícios 9 (continuação)

Atualize e delete os dados da tabela MY_EMPLOYEE.

10. Altere o sobrenome do funcionário 3 para Drexler.

```
SQL> UPDATE my_employee
  2  SET    last_name = 'Drexler'
  3  WHERE  id = 3;
```

11. Altere o salário de todos os funcionários que ganham menos de 900 para 1000.

```
SQL> UPDATE my_employee
  2  SET    salary = 1000
  3  WHERE  salary < 900;
```

12. Verifique as alterações na tabela.

```
SQL> SELECT last_name, salary
  2  FROM   my_employee;
```

13. Delete Betty Dancs da tabela MY_EMPLOYEE.

```
SQL> DELETE
  2  FROM   my_employee
  3  WHERE  last_name = 'Dancs'
  4  AND    first_name = 'Betty';
```

14. Confirme as alterações na tabela.

```
SQL> SELECT *
  2  FROM   my_employee;
```

15. Faça um commit de todas as alterações pendentes.

```
SQL> COMMIT;
```

Controle as transações de dados na tabela MY_EMPLOYEE.

16. Preencha a tabela com a última linha de dados de exemplo, executando o script que você criou na etapa 6.

```
SQL> START loademp.sql
```

17. Confirme a adição à tabela.

```
SQL> SELECT *
  2  FROM   my_employee;
```

18. Marque um ponto intermediário no processamento da transação.

```
SQL> SAVEPOINT a;
```

19. Esvazie a tabela inteira.

```
SQL> DELETE
  2  FROM   my_employee;
```

20. Confirme se a tabela está vazia.

```
SQL> SELECT *
  2  FROM   my_employee;
```

21. Descarte a operação DELETE mais recente sem descartar a operação INSERT anterior.

```
SQL> ROLLBACK TO SAVEPOINT a;
```

22. Confirme se a nova linha ainda está inalterada.

```
SQL> SELECT *
  2  FROM   my_employee;
```

23. Torne a adição de dados permanente.

```
SQL> COMMIT;
```

Soluções de Exercícios 10

1. Crie a tabela DEPARTMENT de acordo com tabela de exemplo a seguir. Informe a sintaxe em um script chamado `p10q1.sql`, e execute o script para criar a tabela. Confirme se a tabela foi criada.

Column Name	Id	Name
Key Type		
Nulls/Unique		
FK Table		
FK Column		
Datatype	Number	VARCHAR2
Length	7	25

```
SQL> EDIT p10q1.sql
           CREATE TABLE department
           (id NUMBER(7),
           name VARCHAR2(25))
           /
SQL> START p10q1.sql
SQL> DESCRIBE department
```

2. Preencha a tabela DEPARTMENT com os dados a partir da tabela DEPT. Inclua somente as colunas necessárias.

```
SQL> INSERT INTO department
  2  SELECT      deptno, dname
  3  FROM        dept;
```

3. Crie a tabela EMPLOYEE de acordo com a tabela de exemplo a seguir. Informe a sintaxe em um script chamado `p10q3.sql`, e execute o script para criar a tabela. Confirme se a tabela foi criada.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				
FK Column				
Datatype	Number	VARCHAR2	VARCHAR2	Number
Length	7	25	25	7

```
CREATE TABLE  employee
(id           NUMBER(7),
last_name     VARCHAR2(25),
first_name    VARCHAR2(25),
dept_id       NUMBER(7))
/
SQL> START p10q3.sql
SQL> DESCRIBE employee
```

4. Modifique a tabela EMPLOYEE para aceitar os sobrenomes longos dos funcionários. Confirme as modificações.

```
SQL> ALTER TABLE employee
  2  MODIFY (last_name   VARCHAR2(50));
SQL> DESCRIBE employee
```

5. Confirme se as tabelas DEPARTMENT e EMPLOYEE estão armazenadas no dicionário de dados. (*Dica:* USER_TABLES)

```
SQL> SELECT        table_name
  2  FROM          user_tables
  3  WHERE         table_name IN ('DEPARTMENT', 'EMPLOYEE');
```

6. Crie a tabela EMPLOYEE2 de acordo com a estrutura da tabela EMP. Inclua somente as colunas EMPNO, ENAME e DEPTNO. Nomeie as colunas da nova tabela como ID, LAST_NAME e DEPT_ID, respectivamente.

```
SQL> CREATE TABLE  employee2 AS
  2  SELECT        empno id, ename last_name, deptno dept_id
  3  FROM          emp;
```

7. Elimine a tabela EMPLOYEE.

```
SQL> DROP TABLE    employee;
```

8. Renomeie a tabela EMPLOYEE2 para EMPLOYEE.

```
SQL> RENAME employee2 TO employee;
```

Soluções de Exercícios 10 (continuação)

9. Adicione um comentário às definições das tabelas DEPARTMENT e EMPLOYEE, descrevendo as tabelas. Confirme as adições ao dicionário de dados.

```
SQL> COMMENT ON TABLE employee IS 'Employee Information';
SQL> COMMENT ON TABLE department IS 'Department Information';
SQL> COLUMN table_name FORMAT A15
SQL> COLUMN table_type FORMAT A10
SQL> COLUMN comments FORMAT A40
SQL> SELECT *
  2  FROM   user_tab_comments
  3  WHERE  table_name = 'DEPARTMENT'
  4  OR     table_name = 'EMPLOYEE';
```

10. Elimine a coluna LAST_NAME da tabela EMPLOYEE. Confirme as modificações, verificando a descrição da tabela.

```
SQL>   ALTER TABLE employee
  2    DROP COLUMN LAST_NAME;
SQL>   DESCRIBE employee
```

11. Crie a tabela EMPLOYEE2 de acordo com a estrutura da tabela EMP. Inclua apenas as colunas EMPNO, ENAME e DEPTNO. Nomeie as colunas como ID, LAST_NAME e DEPT_ID na nova tabela, respectivamente. Marque a coluna DEPT_ID da tabela EMPLOYEE2 como UNSUED. Confirme a modificação, verificando a descrição da tabela.

```
SQL>   CREATE TABLE    employee2 AS
   2   SELECT empno id, ename last_name, deptno dept_id
   3   FROM   emp
   4   /
SQL>   ALTER TABLE employee2
   2   SET   UNUSED (dept_id);
SQL>   DESCRIBE employee2;
```

12. Elimine todas as colunas UNUSED da tabela EMPLOYEE2. Confirme as modificações, verificando a descrição da tabela.

```
SQL>   ALTER TABLE employee2
  2    DROP UNUSED COLUMNS;
SQL>   DESCRIBE employee2
```

Soluções de Exercícios 11

1. Adicione uma restrição no nível de tabela PRIMARY KEY à tabela EMPLOYEE, usando a coluna ID. A restrição deve ser ativada na criação.

```
SQL> ALTER TABLE          employee
  2  ADD CONSTRAINT       employee_id_pk PRIMARY KEY (id);
```

2. Crie uma restrição PRIMARY KEY na tabela DEPARTMENT usando a coluna ID. A restrição deve ser ativada na criação.

```
SQL> ALTER TABLE          department
  2  ADD CONSTRAINT       department_id_pk PRIMARY KEY(id);
```

3. Adicione uma referência à chave estrangeira à tabela EMPLOYEE para garantir que o funcionário não seja atribuído a um departamento inexistente.

```
SQL> ALTER TABLE    employee
  2  ADD CONSTRAINT employee_dept_id_fk FOREIGN KEY (dept_id)
  3       REFERENCES department(id);
```

4. Confirme se as restrições foram adicionadas, consultando USER_CONSTRAINTS. Note os tipos e nomes das restrições. Salve o texto da instrução em um arquivo chamado `p11q4.sql`.

```
SQL> SELECT   constraint_name, constraint_type
  2  FROM     user_constraints
  3  WHERE    table_name IN ('EMPLOYEE', 'DEPARTMENT');
SQL> SAVE p11q4.sql
```

5. Exiba os tipos e nomes de objeto na view do dicionário de dados USER_OBJECTS para as tabelas EMPLOYEE e DEPARTMENT. Você pode desejar formatar as colunas para torná-las mais legíveis. Note que as novas tabelas e o novo índice foram criados.

```
SQL> COLUMN   object_name  FORMAT A30
SQL> COLUMN   object_type  FORMAT A30
SQL> SELECT   object_name, object_type
  2  FROM     user_objects
  3  WHERE    object_name LIKE 'EMPLOYEE%'
  4  OR       object_name LIKE 'DEPARTMENT%';
```

Se você tiver tempo, complete o exercício abaixo:

6. Modifique a tabela EMPLOYEE. Adicione uma coluna SALARY do tipo de dados NUMBER, precisão 7.

```
SQL> ALTER TABLE employee
  2  ADD (salary NUMBER(7));
```

Soluções de Exercícios 12

1. Crie uma view chamada EMP_VU com base no número de funcionários, nome dos funcionários e número dos departamentos da tabela EMP. Altere o cabeçalho do nome do funcionário para EMPLOYEE.

```
SQL> CREATE VIEW emp_vu AS
  2  SELECT      empno, ename employee, deptno
  3  FROM        emp;
```

2. Exiba o conteúdo da view EMP_VU.

```
SQL> SELECT      *
  2  FROM        emp_vu;
```

3. Selecione o texto e o nome da view no dicionário de dados USER_VIEWS.

```
SQL> COLUMN view_name FORMAT A30
SQL> COLUMN text FORMAT A50
SQL> SELECT      view_name, text
  2  FROM        user_views;
```

4. Usando a view EMP_VU, informe uma consulta para exibir os nomes de todos os funcionários e números dos departamentos.

```
SQL> SELECT      employee, deptno
  2  FROM        emp_vu;
```

5. Crie uma view chamada DEPT20 que contenha o número e o nome do funcionário e o número do departamento de todos os funcionários do departamento 20. Coloque um label nas colunas da view EMPLOYEE_ID, EMPLOYEE e DEPARTMENT_ID. Não permita que um funcionário seja reatribuído a outro departamento através da view.

```
SQL> CREATE VIEW dept20 AS
  2  SELECT      empno employee_id, ename employee,
  3              deptno department_id
  4  FROM        emp
  5  WHERE       deptno = 20
  6  WITH CHECK OPTION CONSTRAINT emp_dept_20;
```

Soluções de Exercícios 12 (continuação)

6. Exiba a estrutura e o conteúdo da view DEPT20.

```
SQL> DESCRIBE dept20
SQL> SELECT *
  2  FROM   dept20;
```

7. Tente reatribuir Smith ao departamento 30.

```
SQL> UPDATE dept20
  2  SET    department_id = 30
  3  WHERE  employee = 'SMITH';
```

Se você tiver tempo, complete o exercício abaixo:

8. Crie uma view chamada SALARY_VU com base no nome do funcionário, nome do departamento, salário e classe de salário de todos os funcionários. Coloque um label nas colunas Employee, Department, Salary e Grade, respectivamente.

```
SQL> CREATE VIEW salary_vu AS
  2  SELECT ename employee, dname department,
  3         sal   salary,   grade
  4  FROM   emp e, dept d, salgrade s
  5  WHERE  e.deptno = d.deptno
  6  AND    e.sal between s.losal and s.hisal;
```

Soluções de Exercícios 13

1. Crie uma seqüência para ser usada como a coluna de chave primária da tabela DEPARTMENT. A seqüência deve iniciar em 60 e ter um valor máximo de 200. O incremento da seqüência deve ser de dez números. Nomeie a seqüência DEPT_ID_SEQ.

```
SQL> CREATE SEQUENCE dept_id_seq
  2  START WITH 60
  3  INCREMENT BY 10
  4  MAXVALUE 200;
```

2. Crie um script para exibir as seguintes informações sobre as seqüências: nome da seqüência, valor máximo, tamanho do incremento e último número. Nomeie o script como `p13q2.sql`. Execute o script.

```
SQL> EDIT p13q2.sql
     SELECT    sequence_name, max_value,
               increment_by, last_number
     FROM      user_sequences
     /
SQL> START p13q2.sql
```

3. Crie um script interativo para inserir uma linha na tabela DEPARTMENT. Nomeie o script `p13q3.sql`. Certifique-se de usar a seqüência criada para a coluna ID. Crie um prompt personalizado para informar o nome do departamento. Execute o script. Adicione dois departamentos nomeados Education e Administration. Confirme as adições.

```
SQL> EDIT p13q3.sql
     SET ECHO OFF
     SET VERIFY OFF
     ACCEPT name PROMPT 'Please enter the department name: '
     INSERT INTO          department (id, name)
     VALUES    (dept_id_seq.NEXTVAL, '&name')
     /
     SET VERIFY ON
     SET ECHO ON
SQL> START p13q3.sql
SQL> SELECT   *
  2  FROM     department;
```

4. Crie um índice não-exclusivo na coluna de chave estrangeira (dept_id) na tabela EMPLOYEE.

```
SQL> CREATE INDEX employee_dept_id_idx ON employee (dept_id);
```

Soluções de Exercícios 13 (continuação)

5. Exiba os índices e as exclusividades existentes no dicionário de dados para a tabela EMPLOYEE. Salve a instrução em um script nomeado `p13q5.sql`.

```
SQL> SELECT index_name, table_name, uniqueness
  2  FROM   user_indexes
  3  WHERE  table_name = 'EMPLOYEE';
SQL> SAVE p13q5.sql
```

Soluções de Exercícios 14

1. Qual o privilégio que um usuário deve receber para efetuar login no Oracle Server? É um privilégio de objeto ou sistema?

   O privilégio de sistema CREATE SESSION

2. Qual o privilégio que um usuário deve receber para criar tabelas?

   O privilégio CREATE TABLE

3. Se você criar uma tabela, quem poderá passar privilégios para outros usuários sobre sua tabela?

   Você ou qualquer pessoa à qual tenha concedido esses privilégios usando WITH GRANT OPTION.

4. Você é o DBA. Você está criando muitos usuários que estão exigindo os mesmos privilégios de sistema. O que faria para tornar seu trabalho mais fácil?

   Crie uma função que contenha os privilégios do sistema e conceda-a aos usuários

5. Que comando você usa para alterar sua senha?

   A instrução ALTER USER

6. Conceda acesso à tabela DEPT a outro usuário. Faça com que o usuário conceda a você o acesso de consulta à tabela dele.

   A equipe 2 executa a instrução GRANT.

```
SQL> GRANT    select
  2  ON       dept
  3  TO       <usuário1>;
```

   A equipe 1 executa a instrução GRANT.

```
SQL> GRANT    select
  2  ON       dept
  3  TO       <usuário2>;
```

   WHERE *usuário1* is the name of team 1 and *usuário2* is the name of team 2.

7. Consulte todas as linhas na tabela DEPT.

```
SQL> SELECT   *
  2  FROM     dept;
```

Soluções de Exercícios 14 (continuação)

8. Adicione uma nova linha à tabela DEPT. A equipe 1 deve adicionar Education como o departamento número 50. A equipe 2 deve adicionar Administration como o departamento número 50. Torne as alterações permanentes.

A equipe 1 executa esta instrução INSERT.

```
SQL> INSERT INTO dept(deptno, dname)
  2  VALUES (50, 'Education');
SQL> COMMIT;
```

A equipe 2 executa esta instrução INSERT.

```
SQL> INSERT INTO dept(deptno, dname)
  2  VALUES (50, 'Administration');
SQL> COMMIT;
```

9. Crie um sinônimo para a tabela DEPT da outra equipe.

A equipe 1 cria um sinônimo chamado team2.

```
SQL> CREATE SYNONYM  team2
  2  FOR <usuário2>.DEPT;
```

A equipe 2 cria um sinônimo chamado team1.

```
SQL> CREATE SYNONYM  team1
  2  FOR <usuário1>.DEPT;
```

10. Consulte todas as linhas na tabela DEPT da outra equipe, usando o sinônimo.

A equipe 1 executa esta instrução SELECT.

```
SQL> SELECT *
  2  FROM   team2;
```

A equipe 2 executa esta instrução SELECT.

```
SQL> SELECT *
  2  FROM   team1;
```

Soluções de Exercícios 14 (continuação)

11. Consulte o dicionário de dados USER_TABLES para ver as informações sobre as tabelas que você possui.

```
SQL> SELECT table_name
  2  FROM   user_tables;
```

12. Consulte a view do dicionário de dados ALL_TABLES para ver as informações sobre todas as tabelas que você pode acessar. Exclua as tabelas que pertencem a você.

```
SQL> SELECT table_name, owner
  2  FROM   all_tables
  3  WHERE  owner  != <sua conta>;
```

13. Revogue o privilégio SELECT da outra equipe.

```
A equipe 1 revoga o privilégio.
SQL> REVOKE select
  2  ON     dept
  3  FROM   usuário2;
A equipe 2 revoga o privilégio.
SQL> REVOKE select
  2  ON     dept
  3  FROM   usuário1;
```

Soluções de Exercícios 15

1. Crie as tabelas de acordo com as tabelas de exemplo a seguir. Escolha os tipos de dados apropriados e certifique-se de adicionar restrições de integridade.

   a. Nome da tabela: MEMBER

Column_ Name	MEMBER_ ID	LAST_ NAME	FIRST_ NAME	ADDRESS	CITY	PHONE	JOIN_ DATE
Key Type	PK						
Null/ Unique	NN,U	NN					NN
Default Value							System Date
Datatype	Number	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	Date
Length	10	25	25	100	30	15	

```
CREATE TABLE member
(member_id     NUMBER(10)
               CONSTRAINT member_member_id_pk PRIMARY KEY,
last_name      VARCHAR2(25)
               CONSTRAINT member_last_name_nn NOT NULL,
first_name     VARCHAR2(25),
address        VARCHAR2(100),
city           VARCHAR2(30),
phone          VARCHAR2(15),
join_date      DATE DEFAULT SYSDATE
               CONSTRAINT member_join_date_nn NOT NULL);
```

Soluções de Exercícios 15 (continuação)

b. Nome da tabela: TITLE

Column_ Name	TITLE_ ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_ DATE
Key Type	PK					
Null/ Unique	NN,U	NN	NN			
Check				G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMEN TARY	
Datatype	Number	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	Date
Length	10	60	400	4	20	

```
CREATE TABLE title
(title_id   NUMBER(10)
     CONSTRAINT title_title_id_pk PRIMARY KEY,
title       VARCHAR2(60)
     CONSTRAINT title_title_nn NOT NULL,
description VARCHAR2(400)
     CONSTRAINT title_description_nn NOT NULL,
rating      VARCHAR2(4)
     CONSTRAINT title_rating_ck CHECK
          (rating IN ('G', 'PG', 'R', 'NC17', 'NR')),
category    VARCHAR2(20),
     CONSTRAINT title_category_ck CHECK
          (category IN ('DRAMA', 'COMEDY', 'ACTION',
          'CHILD', 'SCIFI', 'DOCUMENTARY')),
release_date     DATE);
```

Soluções de Exercícios 15 (continuação)

c. Nome da tabela: TITLE_COPY

Column Name	COPY_ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/Unique	NN,U	NN,U	NN
Check			AVAILABLE, DESTROYED, RENTED, RESERVED
FK Ref Table		TITLE	
FK Ref Col		TITLE_ID	
Datatype	Number	Number	VARCHAR2
Length	10	10	15

```
CREATE TABLE title_copy
(copy_id       NUMBER(10),
 title_id      NUMBER(10)
 CONSTRAINT title_copy_title_if_fk REFERENCES title(title_id),
 status        VARCHAR2(15)
 CONSTRAINT title_copy_status_nn NOT NULL
 CONSTRAINT title_copy_status_ck CHECK (status IN
 ('AVAILABLE', 'DESTROYED','RENTED', 'RESERVED')),
CONSTRAINT title_copy_copy_id_title_id_pk
              PRIMARY KEY (copy_id, title_id));
```

Soluções de Exercícios 15 (continuação)

d. Nome da tabela: RENTAL

Column Name	BOOK_ DATE	MEMBER_ ID	COPY_ ID	ACT_RET_ DATE	EXP_RET_ DATE	TITLE_ID
Key Type	PK	PK,FK1	PK,FK2			PK,FK2
Default Value	System Date				2 days after book date	
FK Ref Table		MEMBER	TITLE_ COPY			TITLE_ COPY
FK Ref Col		MEMBER_ ID	COPY_ID			TITLE_ID
Datatype	Date	Number	Number	Date	Date	Number
Length		10	10			10

```
CREATE TABLE rental
(book_date    DATE DEFAULT SYSDATE,
 member_id    NUMBER(10)
              CONSTRAINT rental_member_id_fk
              REFERENCES member(member_id),
 copy_id      NUMBER(10),
 act_ret_date DATE,
 exp_ret_date DATE DEFAULT SYSDATE + 2,
 title_id     NUMBER(10),
              CONSTRAINT rental_book_date_copy_title_pk
              PRIMARY KEY (book_date, member_id,
              copy_id,title_id),
              CONSTRAINT rental_copy_id_title_id_fk
              FOREIGN KEY (copy_id, title_id)
             REFERENCES title_copy(copy_id, title_id));
```

Soluções de Exercícios 15 (continuação)

e. Nome da tabela: RESERVATION

Column_Name	RES_DATE	MEMBER_ID	TITLE_ID
Key Type	PK	PK,FK1	PK,FK2
Null/ Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE
FK Ref Column		MEMBER_ID	TITLE_ID
Datatype	Date	Number	Number
Length		10	10

```
CREATE TABLE reservation
(res_date   DATE,
 member_id  NUMBER(10)
       CONSTRAINT reservation_member_id
       REFERENCES member(member_id),
 title_id   NUMBER(10)
       CONSTRAINT reservation_title_id
       REFERENCES title(title_id),
 CONSTRAINT reservation_resdate_mem_tit_pk PRIMARY KEY
       (res_date, member_id, title_id));
```

Soluções de Exercícios 15 (continuação)

2. Verifique se as tabelas e as restrições foram criadas adequadamente, verificando o dicionário de dados.

```
SQL> SELECT   table_name
  2  FROM     user_tables
  3  WHERE    table_name IN ('MEMBER', 'TITLE', 'TITLE_COPY',
  4                         'RENTAL', 'RESERVATION');

SQL> COLUMN   constraint_name FORMAT A30
SQL> COLUMN   table_name FORMAT  A15
SQL> SELECT   constraint_name, constraint_type,
  2           table_name
  3   FROM    user_constraints
  4   WHERE   table_name IN ('MEMBER', 'TITLE', 'TITLE_COPY',
  5                         'RENTAL', 'RESERVATION');
```

3. Crie seqüências para identificar exclusivamente cada linha das tabelas MEMBER e TITLE.

   a. Número de membro para tabela MEMBER: inicie com 101; não permita o armazenamento em cache dos valores. Nomeie a seqüência member_id_seq.

```
SQL> CREATE SEQUENCE member_id_seq
  2  START WITH 101
  3  NOCACHE;
```

   b. Número de título para a tabela TITLE: inicie com 92; não permita o armazenamento em cache. Nomeie a seqüência title_id_seq.

```
SQL> CREATE SEQUENCE title_id_seq
  2  START WITH 92
  3  NOCACHE;
```

   c. Verifique a existência das seqüências no dicionário de dados.

```
SQL> SELECT   sequence_name, increment_by, last_number
  2  FROM     user_sequences
  3  WHERE    sequence_name IN ('MEMBER_ID_SEQ',
  4                            'TITLE_ID_SEQ');
```

Soluções de Exercícios 15 (continuação)

4. Adicione dados às tabelas. Crie um script para cada conjunto de dados a adicionar.
   a. Adicione títulos de filmes à tabela TITLE. Crie um script para fornecer as informações sobre os filmes. Salve o script como `p15q4a.sql`. Use as seqüências para identificar exclusivamente cada título. Informe as datas de lançamento no formato DD-MON-YYYY. Lembre-se que as aspas simples em um campo de caracteres devem ser tratadas de forma especial. Verifique as adições.

```
SQL> EDIT p15q4a.sql
     SET ECHO OFF
     INSERT INTO title(title_id, title, description, rating,
           category, release_date)
     VALUES (title_id_seq.NEXTVAL, 'Willie and Christmas Too',
          'All of Willie''s friends make a Christmas list for
           Santa, but Willie has yet to add his own wish list.',
              'G', 'CHILD', TO_DATE('05-OCT-1995','DD-MON-YYYY')
     /
     INSERT INTO title(title_id , title, description, rating,
           category, release_date)
     VALUES (title_id_seq.NEXTVAL, 'Alien Again', 'Yet another
          installment of science fiction history.  Can the
          heroine save the planet from the alien life form?',
          'R', 'SCIFI', TO_DATE( '19-MAY-1995','DD-MON-YYYY'))
     /
     INSERT INTO title(title_id, title, description, rating,
           category, release_date)
     VALUES (title_id_seq.NEXTVAL, 'The Glob', 'A meteor crashes
          near a small American town and unleashes carnivorous
          goo in this classic.', 'NR', 'SCIFI',
             TO_DATE( '12-AUG-1995','DD-MON-YYYY'))
     /
     INSERT INTO title(title_id, title, description, rating,
           category, release_date)
     VALUES (title_id_seq.NEXTVAL, 'My Day Off', 'With a little
          luck and a lot ingenuity, a teenager skips school for
          a day in New York.', 'PG', 'COMEDY',
             TO_DATE( '12-JUL-1995','DD-MON-YYYY'))
     /
     ...
     COMMIT
     /
     SET ECHO ON

     SQL> SELECT   title
       2  FROM     title;
```

Soluções de Exercícios 15 (continuação)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York	PG	COMEDY	12-JUL-1995
Miracles on Ice	A six-year-old has doubts about Santa Claus, but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-1995
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

Soluções de Exercícios 15 (continuação)

b. Adicione dados à tabela MEMBER. Crie um script nomeado `p15q4b.sql` para pedir informações aos usuários. Execute o script. Certifique-se de usar a seqüência para adicionar os números dos membros.

First_ Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

```
SQL> EDIT p15q4b.sql
SET ECHO OFF
SET VERIFY OFF
INSERT INTO member(member_id, first_name, last_name, address,
         city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, '&first_name', '&last_name',
   '&address', '&city', '&phone', TO_DATE('&join_date',
   'DD-MM-YYYY')
/
COMMIT
/
SET VERIFY ON
SET ECHO ON
SQL> START p15q4b.sql
```

Soluções de Exercícios 15 (continuação)

c. Adicione as seguintes cópias de filmes à tabela TITLE_COPY:
Observação: Obtenha os números de title_id para este exercício.

Title	Copy_Id	Status
Willie and Christmas Too	1	AVAILABLE
Alien Again	1	AVAILABLE
	2	RENTED
The Glob	1	AVAILABLE
My Day Off	1	AVAILABLE
	2	AVAILABLE
	3	RENTED
Miracles on Ice	1	AVAILABLE
Soda Gang	1	AVAILABLE

```
SQL> INSERT INTO title_copy(copy_id, title_id, status)
  2  VALUES (1, 92, 'AVAILABLE');
SQL> INSERT INTO title_copy(copy_id, title_id, status)
  2  VALUES (1, 93, 'AVAILABLE');
SQL> INSERT INTO title_copy(copy_id, title_id, status)
  2  VALUES (2, 93, 'RENTED');
SQL> INSERT INTO title_copy(copy_id, title_id, status)
  2  VALUES (1, 94, 'AVAILABLE');
SQL> INSERT INTO title_copy(copy_id, title_id, status)
  2  VALUES (1, 95, 'AVAILABLE');
SQL> INSERT INTO title_copy(copy_id, title_id,status)
  2  VALUES (2, 95, 'AVAILABLE');
SQL> INSERT INTO title_copy(copy_id, title_id,status)
  2  VALUES (3, 95, 'RENTED');
SQL> INSERT INTO title_copy(copy_id, title_id,status)
  2  VALUES (1, 96, 'AVAILABLE');
SQL> INSERT INTO title_copy(copy_id, title_id,status)
  2  VALUES (1, 97, 'AVAILABLE');
```

Soluções de Exercícios 15 (continuação)

d. Adicione os seguintes aluguéis à tabela RENTAL:

Observação: O número do título pode ser diferente, dependendo do número da seqüência.

Title_ Id	Copy_ Id	Member_ Id	Book_date	Exp_Ret_Date	Act_Ret_Date
92	1	101	3 days ago	1 day ago	2 days ago
93	2	101	1 day ago	1 day from now	
95	3	102	2 days ago	Today	
97	1	106	4 days ago	2 days ago	2 days ago

```
SQL> INSERT INTO rental(title_id, copy_id, member_id,
  2             book_date, exp_ret_date, act_ret_date)
  3  VALUES (92, 1, 101, sysdate-3, sysdate-1, sysdate-2);
SQL> INSERT INTO rental(title_id, copy_id, member_id,
  2             book_date, exp_ret_date, act_ret_date)
  3  VALUES (93, 2, 101, sysdate-1, sysdate-1, NULL);
SQL> INSERT INTO rental(title_id, copy_id, member_id,
  2             book_date, exp_ret_date, act_ret_date)
  3  VALUES (95, 3, 102, sysdate-2, sysdate, NULL);
SQL> INSERT INTO rental(title_id, copy_id, member_id,
  2             book_date, exp_ret_date,act_ret_date)
  3  VALUES (97, 1, 106, sysdate-4, sysdate-2, sysdate-2);
SQL> COMMIT;
```

Soluções de Exercícios 15 (continuação)

5. Crie uma view nomeada TITLE_AVAIL para mostrar os títulos dos filmes e a disponibilidade de cada cópia e a data esperada de devolução, caso esteja alugado. Consulte todas as linhas da view. Ordene os resultados por título.

```
SQL> CREATE VIEW title_avail AS
  2    SELECT   t.title, c.copy_id, c.status, r.exp_ret_date
  3    FROM     title t, title_copy c, rental r
  4    WHERE    t.title_id = c.title_id
  5    AND      c.copy_id = r.copy_id(+)
  6    AND      c.title_id = r.title_id(+);

SQL> COLUMN title FORMAT A30

SQL>  SELECT     *
  2   FROM       title_avail
  3   ORDER BY   title, copy_id;
```

6. Faça alterações nos dados das tabelas.

   a. Adicione um novo título. O filme é "Interstellar Wars", classificado como PG e filme de ficção científica. O lançamento é 07-JUL-77. A descrição é "Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?" Certifique-se de adicionar um registro de cópia do título para duas cópias.

```
INSERT INTO title(title_id, title, description, rating,
         category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Interstellar Wars',
        'Futuristic interstellar action movie.  Can the
         rebels save the humans from the evil Empire?',
         'PG', 'SCIFI', '07-JUL-77')
/
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (1, 98, 'AVAILABLE')
/
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (2, 98, 'AVAILABLE')
/
```

   b. Informe duas reservas. Uma reserva é para Carmen Velasquez, que deseja alugar "Interstellar Wars". Outra é para Mark Quick-to-See, que deseja alugar "Soda Gang".

```
SQL> INSERT INTO reservation (res_date, member_id, title_id)
  2  VALUES (SYSDATE, 101, 98);
SQL> INSERT INTO reservation (res_date, member_id, title_id)
  2  VALUES (SYSDATE, 104, 97);
```

c. A cliente Carmen Velasquez aluga o filme "Interstellar Wars", cópia 1. Remova sua reserva do filme. Registre as informações sobre o aluguel. Use o valor padrão para a data de devolução esperada. Verifique se o aluguel foi registrado usando a view criada.

```
SQL> INSERT INTO rental(title_id, copy_id, member_id)
  2  VALUES (98,1,101);
SQL> UPDATE    title_copy
  2  SET       status= 'RENTED'
  3  WHERE     title_id = 98
  4  AND       copy_id = 1;
SQL> DELETE
  2  FROM      reservation
  3  WHERE     member_id = 101;
SQL> SELECT    *
  2  FROM      title_avail
  3  ORDER BY  title, copy_id;
```

7. Faça uma modificação em uma das tabelas.

a. Adicione uma coluna PRICE à tabela TITLE para registrar o preço de compra do vídeo. A coluna deve ter um comprimento total de oito dígitos e duas casas decimais. Verifique as modificações.

```
SQL> ALTER TABLE title
  2  ADD         (price NUMBER(8,2));
SQL> DESCRIBE    title
```

Soluções de Exercícios 15 (continuação)

b. Crie um script chamado `p15q7b.sql` para atualizar o preço de cada vídeo de acordo com a lista a seguir.

Observação: Obtenha os números de title_id para este exercício.

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

```
SET ECHO OFF
SET VERIFY OFF
UPDATE        title
SET           price = &price
WHERE         title_id = &title_id
/
SET VERIFY OFF
SET ECHO OFF
SQL> START p15q7b.sql
```

c. Certifique-se de que, no futuro, todos os títulos conterão um valor de preço. Verifique a restrição.

```
SQL> ALTER TABLE title
  2  MODIFY (price CONSTRAINT title_price_nn NOT NULL);
SQL> SELECT   constraint_name, constraint_type,
  2           search_condition
  3  FROM     user_constraints
  4  WHERE    table_name = 'TITLE';
```

Soluções de Exercícios 15 (continuação)

8. Crie um relatório intitulado Customer History Report. Esse relatório conterá o histórico de aluguel de vídeos de cada cliente. Certifique-se de incluir o nome do cliente, o filme alugado, as datas de retirada e duração dos aluguéis. Some o número de aluguéis de todos os clientes durante o período do relatório. Salve o script em um arquivo nomeado `p15q8.sql`.

```
SQL> EDIT p15q8.sql
SET ECHO OFF
SET VERIFY OFF
SET PAGESIZE 30
COLUMN   member  FORMAT A17
COLUMN   title FORMAT A25
COLUMN   book_date FORMAT A9
COLUMN   duration FORMAT 9999999

TTITLE 'Customer History Report'
BREAK ON member  SKIP 1 ON REPORT
SELECT      m.first_name||' '||m.last_name MEMBER, t.title,
            r.book_date, r.act_ret_date - r.book_date DURATION
FROM        member m, title t, rental r
WHERE       r.member_id = m.member_id
AND         r.title_id = t.title_id
ORDER BY    member
/
CLEAR BREAK
COLUMN member CLEAR
COLUMN title CLEAR
COLUMN book_date CLEAR
COLUMN duration CLEAR

TTITLE OFF
SET VERIFY ON
SET PAGESIZE 24
SET ECHO ON
```

Soluções de Exercícios 16

1. Avalie cada uma das declarações a seguir. Determine quais delas *não* são legais e explique por quê.

   a.
   ```
   DECLARE
     v_id           NUMBER(4);
   ```
   Legal

   b.
   ```
   DECLARE
     v_x, v_y, v_z  VARCHAR2(10);
   ```
   Ilegal pois é permitido apenas um identificador por declaração

   c.
   ```
   DECLARE
     v_birthdate    DATE NOT NULL;
   ```
   Ilegal pois a variável NOT NULL deve ser inicializada

   d.
   ```
   DECLARE
     v_in_stock     BOOLEAN := 1;
   ```
   Ilegal pois 1 não é uma expressão Booleana

Soluções de Exercícios 16 (continuação)

2. Em cada uma das seguintes atribuições, determine o tipo de dados da expressão resultante.

   a. `v_days_to_go := v_due_date - SYSDATE;`

      Número

   b. `v_sender := USER || ': ' || TO_CHAR(v_dept_no);`

      String de caracteres

   c. `v_sum := $100,000 + $250,000;`

      Ilegal; o PL/SQL não consegue converter símbolos especiais como VARCHAR2 em NÚMEROS

   d. `v_flag := TRUE;`

      Booleano

   e. `v_n1 := v_n2 > (2 * v_n3);`

      Booleano

   f. `v_value := NULL;`

      Qualquer tipo de dados escalares

3. Crie um bloco anônimo para a saída da frase "My PL/SQL Block Works" na tela.

```
VARIABLE g_message VARCHAR2(30)
BEGIN
  :g_message := 'My PL/SQL Block Works';
END;
/
PRINT g_message
SQL> START p16q3.sql

G_MESSAGE
------------------------------------------------------
My PL/SQL Block Works
```

Soluções de Exercícios 16 (continuação)

Se você tiver tempo, complete o exercício abaixo:

4. Crie um bloco que declare duas variáveis. Atribua o valor dessas variáveis PL/SQL às variáveis de host SQL*Plus e imprima os resultados das variáveis PL/SQL na tela. Execute o bloco PL/SQL. Salve o bloco PL/SQL em um arquivo nomeado `p16q4.sql`.

   V_CHAR Charactere (variable lenght)

   V_NUM Number

   Atribua valores a essas variáveis do seguinte modo:

```
Variable        Value
---------- ----------------------------------------
V_CHAR     The literal '42 is the answer'
V_NUM      The first two characters from V_CHAR
```

```
VARIABLE g_char VARCHAR2(30)
VARIABLE g_num  NUMBER
DECLARE
  v_char VARCHAR2(30);
  v_num  NUMBER(11,2);
BEGIN
  v_char := '42 is the answer';
  v_num  := TO_NUMBER(SUBSTR(v_char,1,2));
  :g_char := v_char;
  :g_num  := v_num;
END;
/
PRINT g_char
PRINT g_num
SQL> START p16q4.sql


G_CHAR
----------------------
42 is the answer


    G_NUM
---------
       42
```

Soluções de Exercícios 17

```
PL/SQL Block
 DECLARE
   v_weight  NUMBER(3) := 600;
   v_message VARCHAR2(255) := 'Product 10012';
 BEGIN
                  /*SUBBLOCK*/
     DECLARE
       v_weight   NUMBER(3) := 1;
       v_message  VARCHAR2(255) := 'Product 11001';
       v_new_locn VARCHAR2(50) := 'Europe';
     BEGIN
       v_weight := v_weight + 1;
       v_new_locn := 'Western ' || v_new_locn;
     END;


   v_weight := v_weight + 1;
   v_message := v_message || ' is in stock';
   v_new_locn := 'Western ' || v_new_locn;


 END;
```

Soluções de Exercícios 17 (continuação)

1. Avalie o bloco PL/SQL da página anterior e determine o tipo de dados e o valor de cada uma das seguintes variáveis de acordo com as regras de escopo.
   a. O valor de V_WEIGHT no sub-bloco é:

      "2" e o tipo de dados é NUMBER.

   b. O valor de V_NEW_LOCN no sub-bloco é:

      "Western Europe" e o tipo de dados é VARCHAR2.

   c. O valor de V_WEIGHT no bloco principal é:

      "601" e o tipo de dados é NUMBER.

   d. O valor de V_MESSAGE no bloco principal é:

      "Product 10012 is in stock" e o tipo de dados é VARCHAR2.

   e. O valor de V_NEW_LOCN no bloco principal é:

      Ilegal pois v_new_locn não é visível fora do sub-bloco.

Soluções de Exercícios 17 (continuação)

Exemplo de Escopo

```
DECLARE
  v_customer        VARCHAR2(50) := 'Womansport';
  v_credit_rating   VARCHAR2(50) := 'EXCELLENT';
BEGIN
  DECLARE
    v_customer  NUMBER(7) := 201;
    v_name VARCHAR2(25) := 'Unisports';
  BEGIN
   v_customer     v_name  v_credit_rating
  END;

 v_customer      v_name  v_credit_rating

END;
```

Soluções de Exercícios 17 (continuação)

2. Vamos supor que você incorpore um sub-bloco a um bloco, como mostrado na página anterior. Você declara duas variáveis, V_CUSTOMER e V_CREDIT_RATING, no bloco principal. Declara também duas variáveis, V_CUSTOMER e V_NAME, no sub-bloco. Determine os valores e tipos de dados para cada um dos casos a seguir.
   a. O valor de V_CUSTOMER no sub-bloco é:

      "201" e o tipo de dados é NUMBER.

   b. O valor de V_NAME no sub-bloco é:

      "Unisports" e o tipo de dados é VARCHAR2.

   c. O valor de V_CREDIT_RATING no sub-bloco é:

      "EXCELLENT" e o tipo de dados é VARCHAR2.

   d. O valor de V_CUSTOMER no bloco principal é:

      "Womansport" e o tipo de dados é VARCHAR2.

   e. O valor de V_NAME no bloco principal é:

      V_NAME não está visível no bloco principal e você verá um erro.

   f. O valor de V_CREDIT_RATING no bloco principal é:

      "EXCELLENT" e o tipo de dados é VARCHAR2.

Soluções de Exercícios 17 (continuação)

3. Crie e execute um bloco PL/SQL que aceite dois números através das variáveis de substituição do SQL*Plus. O primeiro número deve ser dividido pelo segundo e este deve ser adicionado ao resultado. O resultado deve ser armazenado em uma variável PL/SQL e impresso na tela, ou o resultado deve ser gravado na variável do SQL*Plus e impresso na tela.

   a. Quando uma variável PL/SQL é usada:

```
SET ECHO OFF
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_num1 PROMPT 'Please enter the first number: '
ACCEPT p_num2 PROMPT 'Please enter the second number: '
DECLARE
     v_num1   NUMBER(9,2) := &p_num1;
     v_num2   NUMBER(9,2) := &p_num2;
     v_result  NUMBER(9,2) ;
BEGIN
     v_result :=  (v_num1/v_num2) + v_num2;
     /* Printing the PL/SQL variable */
     DBMS_OUTPUT.PUT_LINE (v_result);
END;
/
SET SERVEROUTPUT OFF
SET VERIFY ON
SET ECHO ON
```

Observação: A solução continua na próxima página.

Soluções de Exercícios 17 (continuação)

b. Quando uma variável do SQL*Plus é usada:

```
SET ECHO OFF
SET VERIFY OFF
VARIABLE g_result NUMBER
ACCEPT p_num1 PROMPT 'Please enter the first number: '
ACCEPT p_num2 PROMPT 'Please enter the second number: '
DECLARE
     v_num1   NUMBER(9,2) := &p_num1;
     v_num2   NUMBER(9,2) := &p_num2;
BEGIN
     :g_result := (v_num1/v_num2) + v_num2;
END;
/
PRINT g_result /* Printing the SQL*Plus variable */
SET VERIFY ON
SET ECHO ON
```

Soluções de Exercícios 17 (continuação)

4. Elabore um bloco PL/SQL que calcule a remuneração total por um ano. O salário anual e o percentual de bônus anual são passados ao bloco PL/SQL através das variáveis de substituição do SQL*Plus e o bônus precisa ser convertido de um número inteiro em um número decimal (por exemplo, 15 em ,15). Se o salário for nulo, defina-o como zero antes de calcular a remuneração total. Execute o bloco PL/SQL. *Aviso*: Use a função NVL para tratar valores nulos.

   Observação: Para testar a função NVL, digite NULL no prompt; o pressionamento de [Return] resulta em um erro de expressão faltando.

   a. Quando uma variável PL/SQL é usada:

```
SET VERIFY OFF
VARIABLE g_total NUMBER
ACCEPT p_salary PROMPT 'Please enter the salary amount: '
ACCEPT p_bonus PROMPT 'Please enter the bonus percentage: '
DECLARE
  v_salary NUMBER := &p_salary;
  v_bonus  NUMBER := &p_bonus;
BEGIN
  :g_total := NVL(v_salary, 0) * (1 + NVL(v_bonus, 0) / 100);
END;
/
PRINT g_total
SET VERIFY ON
SQL> START p17q4.sql
```

   b. Quando uma variável do SQL*Plus é usada:

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_salary PROMPT 'Please enter the salary amount: '
ACCEPT p_bonus PROMPT 'Please enter the bonus percentage: '
DECLARE
  v_salary NUMBER := &p_salary;
  v_bonus  NUMBER := &p_bonus;
BEGIN
  dbms_output.put_line(TO_CHAR(NVL(v_salary, 0) *
                    (1 + NVL(v_bonus, 0) / 100)));
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

Soluções de Exercícios 18

1. Crie um bloco PL/SQL que selecione o número máximo de departamento na tabela DEPT e armazene-o em uma variável do SQL*Plus. Imprima os resultados na tela. Salve o bloco PL/SQL em um arquivo nomeado `p18q1.sql`.

```
VARIABLE g_max_deptno NUMBER
DECLARE
  v_max_deptno  NUMBER;
BEGIN
  SELECT    MAX(deptno)
  INTO      v_max_deptno
  FROM      dept;
  :g_max_deptno := v_max_deptno;
END;
/
PRINT g_max_deptno
SQL> START p18q1.sql

DECLARE
  v_max_deptno  NUMBER;
BEGIN
  SELECT    MAX(deptno)
  INTO      v_max_deptno
  FROM      dept;
  dbms_output.put_line(TO_CHAR(v_max_deptno));
END;
/
```

2. Modifique o bloco PL/SQL criado no exercício 1 para inserir uma nova linha na tabela DEPT. Salve o bloco PL/SQL em um arquivo nomeado `p18q2.sql`.
   a. Em vez de imprimir o número do departamento recuperado do exercício 1, adicione 10 a esse número e use-o como o número do novo departamento.
   b. Use um parâmetro de substituição do SQL*Plus para o nome do departamento.
   c. Deixe um valor nulo na localização por enquanto.

```
SET ECHO OFF
SET VERIFY OFF
ACCEPT p_dept_name PROMPT 'Please enter the department name: '
DECLARE
  v_max_deptno  dept.deptno%TYPE;
BEGIN
  SELECT   MAX(deptno)+10
  INTO     v_max_deptno
  FROM     dept;
  INSERT INTO dept (deptno, dname, loc)
  VALUES   (v_max_deptno, '&p_dept_name', NULL);
  COMMIT;
END;
/
SET ECHO ON
SET VERIFY ON
```

d. Execute o bloco PL/SQL.

```
SQL> START p18q2.sql
```

e. Exiba o novo departamento criado.

```
SELECT *
FROM   dept
WHERE  deptno = :g_max_deptno + 10;
```

3. Crie um bloco PL/SQL que atualize a localização de um departamento existente. Salve o bloco PL/SQL em um arquivo nomeado `p18q3.sql`.
   a. Use um parâmetro de substituição do SQL*Plus para o número do departamento.
   b. Use um parâmetro de substituição do SQL*Plus para a localização do departamento.
   c. Teste o bloco PL/SQL.
   d. Exiba o nome e o número do departamento, além da localização do departamento atualizado.

```
SET VERIFY OFF
ACCEPT p_deptno PROMPT 'Please enter the department number: '
ACCEPT p_loc PROMPT 'Please enter the department location: '
BEGIN
  UPDATE dept
  SET loc = '&p_loc'
  WHERE deptno = &p_deptno;
COMMIT;
END;
/
SET VERIFY ON
SQL> START p18q3.sql
```

e. Exiba o departamento atualizado.

```
SQL> SELECT *
  2  FROM   dept
  3  WHERE  deptno = &p_deptno;
```

4. Crie um bloco PL/SQL que delete o departamento criado no exercício 2. Salve o bloco PL/SQL em um arquivo nomeado `p18q4.sql`.
   a. Use um parâmetro de substituição do SQL*Plus para o número do departamento.
   b. Imprima na tela o número de linhas afetadas.
   c. Teste o bloco PL/SQL.

```
SET VERIFY OFF
VARIABLE g_result VARCHAR2(40)
ACCEPT p_deptno PROMPT 'Please enter the department number: '
DECLARE
  v_result NUMBER(2);
BEGIN
  DELETE
  FROM     dept
  WHERE    deptno = &p_deptno;
  v_result := SQL%ROWCOUNT;
  :g_result := (TO_CHAR(v_result) || ' row(s) deleted.');
  COMMIT;
END;
/
PRINT g_result
SET VERIFY ON
SQL> START p18q4.sql

  ACCEPT p_deptno PROMPT 'Please enter the department number: '
  DECLARE
    v_result NUMBER(2);
  BEGIN
    DELETE
    FROM     dept
    WHERE    deptno = &p_deptno;
    v_result := SQL%ROWCOUNT;
    dbms_output.put_line(TO_CHAR(v_result)||
    ' row(s) deleted.');
    COMMIT;
  END;
  /
```

Soluções de Exercícios 18 (continuação)

d. O que acontece se você informar um número de departamento inexistente?

Se o operador informar um número de departamento inexistente, o bloco PL/SQL será concluído com êxito, pois não constitui uma exceção.

e. Confirme que o departamento foi deletado.

```
SQL> SELECT     *
  2   FROM      dept
  3   WHERE     deptno = 50;
```

Soluções de Exercícios 19

1. Execute o script `lab19_1.sql` para criar a tabela MESSAGES. Crie um bloco PL/SQL para inserir números na tabela MESSAGES.

```
CREATE TABLE messages (results VARCHAR2 (60))
/
```

   a. Insira os números de 1 a 10, excluindo 6 e 8.

   b. Faça um commit antes do final do bloco.

```
BEGIN
FOR i IN 1..10 LOOP
  IF i = 6 or i = 8 THEN
    null;
  ELSE
 INSERT INTO messages(results)
 VALUES (i);
  END IF;
  COMMIT;
END LOOP;
END;
/
```

   c. Faça seleções na tabela MESSAGES para verificar se o bloco PL/SQL funcionou.

```
SQL> SELECT     *
  2  FROM  messages;
```

2. Crie um bloco PL/SQL que calcule a comissão de um determinado funcionário de acordo com o salário dele.

   a. Execute o script `lab19_2.sql` para inserir um novo funcionário na tabela EMP. **Observação:** O funcionário terá um salário NULL.

```
SQL> START lab19_2.sql
```

   b. Aceite o número do funcionário como entrada do usuário com uma variável de substituição do SQL*Plus.

   c. Se o salário do funcionário for menor que US$1.000, defina o valor da comissão do funcionário como 10% do salário.

   d. Se o salário do funcionário estiver entre US$1.000 e US$1.500, defina o valor da comissão do funcionário como 15% do salário.

   e. Se o salário do funcionário exceder US$1.500, defina o valor da comissão do funcionário como 20% do salário.

   f. Se o salário do funcionário for NULL, defina o valor da comissão do funcionário como 0.

   g. Faça um commit.

```
ACCEPT p_empno PROMPT 'Please enter employee number: '
DECLARE
  v_empno         emp.empno%TYPE := &p_empno;
  v_sal           emp.sal%TYPE;
  v_comm               emp.comm%TYPE;
BEGIN
  SELECT sal
  INTO v_sal
  FROM emp
  WHERE empno = v_empno;
  IF v_sal < 1000 THEN
    v_comm := .10;
  ELSIF v_sal BETWEEN 1000 and 1500 THEN
    v_comm := .15;
  ELSIF v_sal > 1500 THEN
   v_comm := .20;
  ELSE
   v_comm := 0;
END IF;
 UPDATE emp
 SET comm = NVL(sal,0) * v_comm
 WHERE empno = v_empno;
COMMIT;
END;
/
```

h. Teste o bloco PL/SQL para cada caso usando os casos de teste a seguir e verifique cada comissão atualizada.

Employee Number	Salary	Resulting Commission
7369	800	80
7934	1300	195
7499	1600	320
8000	NULL	0

Soluções de Exercícios 19 (continuação)

```
SQL> SELECT      empno, sal, comm
  2  FROM        emp
  3  WHERE       empno IN (7369, 7934,7499, 8000)
  4  ORDER BY    comm;
```

Se você tiver tempo, complete os exercícios abaixo:

3. Modifique o arquivo `p16q4.sql` para inserir o texto "O número é ímpar" ou "O número é par", dependendo de o valor ser ímpar ou par, na tabela MESSAGES. Consulte a tabela MESSAGES para determinar se o bloco PL/SQL funcionou.

```
DECLARE
  v_char VARCHAR2(30);
  v_num  NUMBER(11,2);
BEGIN
  v_char := '42 is the answer';
  v_num  := TO_NUMBER(SUBSTR(v_char,1,2));
  IF mod(v_num, 2) = 0 THEN
    INSERT INTO messages (results)
    VALUES ('Number is even');
  ELSE
    INSERT INTO messages (results)
    VALUES ('Number is odd');
  END IF;
END;
/
SQL> SELECT *
  2  FROM messages;
```

4. Adicione uma nova coluna, STARS de tipo de dados VARCHAR2 e o comprimento 50 à tabela EMP para armazenamento de asterisco (*).

```
SQL> ALTER TABLE emp
  2  ADD stars   VARCHAR2(50);
```

5. Crie um bloco PL/SQL que premie um funcionário, anexando um asterisco à coluna STARS para cada US$100 do salário do funcionário. Salve o bloco PL/SQL em um arquivo nomeado `p19q5.sql`.
   a. Aceite o ID do funcionário como entrada do usuário com uma variável de substituição do SQL*Plus.
   b. Inicialize uma variável que conterá uma string de asteriscos.
   c. Anexe um asterisco à string para cada US$100 do salário. Por exemplo, se o funcionário recebe um salário de US$800, a string de asteriscos deve conter oito asteriscos. Se o funcionário recebe um salário de US$1.250, a string de asteriscos deve conter 13 asteriscos.
   d. Atualize a coluna STARS do funcionário com a string de asteriscos.

Soluções de Exercícios 19 (continuação)

e. Faça um commit.

f. Teste o bloco para os funcionários sem salário e para um funcionário com salário.

```
SET VERIFY OFF
ACCEPT p_empno PROMPT 'Please enter the employee number: '
DECLARE
  v_empno   emp.empno%TYPE := &p_empno;
  v_asterisk    emp.stars%TYPE := NULL;
  v_sal     emp.sal%TYPE;
BEGIN
  SELECT NVL(ROUND(sal/100), 0)
  INTO v_sal
  FROM emp
  WHERE empno = v_empno;
  FOR i IN 1..v_sal LOOP
    v_asterisk := v_asterisk ||'*';
  END LOOP;
  UPDATE emp
  SET stars = v_asterisk
  WHERE empno = v_empno;
  COMMIT;
END;
/
SET VERIFY ON
SQL> START p19q5.sql
SQL> SELECT empno, sal, stars
  2  FROM   emp
  3  WHERE  empno IN (7934, 8000);
```

Soluções de Exercícios 20

1. Crie um bloco PL/SQL para recuperar o nome de cada departamento da tabela DEPT e imprima o nome de cada departamento na tela, incorporando uma tabela PL/SQL.
   a. Declare uma tabela PL/SQL, MY_DEPT_TABLE, para armazenar temporariamente o nome do departamento.
   b. Usando um loop, recupere o nome de todos os departamentos que constam atualmente da tabela DEPT e armazene-os na tabela PL/SQL. Cada número de departamento é um múltiplo de 10.
   c. Usando outro loop, recupere os nomes dos departamentos da tabela PL/SQL e imprima-os na tela, usando DBMS_OUTPUT.PUT_LINE.

```
SET SERVEROUTPUT ON
DECLARE
  TYPE dept_table_type is table of dept.dname%TYPE
  INDEX BY BINARY_INTEGER;
  my_dept_table dept_table_type;
  v_count       NUMBER (2);
BEGIN
  SELECT COUNT(*)
  INTO  v_count
  FROM  dept;
  FOR i IN 1..v_count LOOP
    SELECT dname
    INTO my_dept_table(i)
    FROM dept
    WHERE deptno = i*10;
  END LOOP;
  FOR i IN 1..v_count  LOOP
         DBMS_OUTPUT.PUT_LINE (my_dept_table(i));
  END LOOP;
END;
/
```

2. Crie um bloco PL/SQL para imprimir as informações sobre um determinado pedido.
   a. Declare um registro PL/SQL de acordo com a estrutura da tabela ORD.
   b. Use uma variável de substituição do SQL*Plus para recuperar todas as informações sobre um pedido específico e armazene-as no registro PL/SQL.
   c. Use DBMS_OUTPUT. PUT_LINE e imprima informações selecionadas sobre o pedido.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
ACCEPT p_ordid PROMPT 'Please enter an order number: '
DECLARE
 ord_record    ord%ROWTYPE;
BEGIN
SELECT*
INTO  ord_record
FROM  ord
WHERE ordid = &p_ordid;
DBMS_OUTPUT.PUT_LINE ('Order ' || TO_CHAR(ord_record.ordid)
  || ' was placed on ' || TO_CHAR(ord_record.orderdate)
 || ' and shipped on ' || TO_CHAR(ord_record.shipdate) ||
 ' for a total of ' ||
 TO_CHAR(ord_record.total,'$99,999.99'));
END;
/
```

Se você tiver tempo, complete os exercícios abaixo.

3. Modifique o bloco criado no exercício 1 para recuperar todas as informações sobre cada departamento da tabela DEPT e imprima as informações na tela, incorporando uma tabela PL/SQL de registros.
   a. Declare uma tabela PL/SQL, MY_DEPT_TABLE, para armazenar temporariamente o número, o nome e a localização de todos os departamentos.
   b. Usando um loop, recupere todas as informações sobre departamentos que constam atualmente na tabela DEPT e armazene-as na tabela PL/SQL. Cada número de departamento é um múltiplo de 10.
   c. Usando outro loop, recupere as informações sobre departamentos da tabela PL/SQL e imprima-as na tela, usando DBMS_OUTPUT.PUT_LINE.

Soluções de Exercícios 20 (continuação)

```
SET SERVEROUTPUT ON
DECLARE
   TYPE dept_table_type is table of dept%ROWTYPE
   INDEX BY BINARY_INTEGER;
   my_dept_table  dept_table_type;
   v_count        NUMBER (2);
BEGIN
  SELECT COUNT(*)
  INTO   v_count
  FROM   dept;
  FOR i IN 1..v_count
  LOOP
    SELECT *
    INTO my_dept_table(i)
    FROM dept
    WHERE deptno = i*10;
  END LOOP;
  FOR i IN 1..v_count
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Dept. ' || my_dept_table(i).deptno || ', '
    || my_dept_table(i).dname || ' is located in ' ||
       my_dept_table(i).loc);
  END LOOP;
END;
/
```

Soluções de Exercícios 21

1. Execute o script `lab21_1.sql` para criar uma nova tabela de armazenamento de funcionários e salários.

```
SQL> CREATE TABLE top_dogs
  2  (name        VARCHAR2(25),
  3   salary      NUMBER(11,2));
```

2. Crie um bloco PL/SQL que determine os funcionários com os maiores salários.
   a. Aceite um número *n* como entrada de usuário com um parâmetro de substituição do SQL*Plus.
   b. Em um loop, obtenha os sobrenomes e salários dos n funcionários com os maiores salários na tabela EMP.
   c. Armazene os nomes e os salários na tabela TOP_DOGS.
   d. Suponha que nenhum dos funcionários tenha salário igual ao do outro.
   e. Teste vários casos especiais, como *n* = 0 ou em que *n* seja maior que o número de funcionários na tabela EMP. Esvazie a tabela TOP_DOGS depois de cada teste.

```
DELETE FROM top_dogs;
SET ECHO OFF
ACCEPT p_num -
PROMPT 'Please enter the number of top money makers: '
DECLARE
  v_num         NUMBER(3) := &p_num;
  v_ename       emp.ename%TYPE;
  v_sal         emp.sal%TYPE;
  CURSOR        emp_cursor IS
    SELECT      ename, sal
    FROM        emp
    WHERE       sal IS NOT NULL
    ORDER BY    sal DESC;
BEGIN
OPEN emp_cursor;
FETCH emp_cursor INTO v_ename, v_sal;
WHILE emp_cursor%ROWCOUNT <= v_num AND
      emp_cursor%FOUND LOOP
  INSERT INTO top_dogs (name, salary)
  VALUES (v_ename, v_sal);
   FETCH emp_cursor INTO v_ename, v_sal;
END LOOP;
CLOSE emp_cursor;
COMMIT;
END;
/
SELECT * FROM top_dogs;
SET ECHO ON
```

Soluções de Exercícios 21 (continuação)

3. Considere o caso em que vários funcionários recebem o mesmo salário. Se uma pessoa estiver listada, todas as pessoas com o mesmo salário também deverão estar listadas.
   a. Por exemplo, se o usuário informar um valor 2 para *n*, King, Ford e Scott deverão ser exibidos. (Esses funcionários são reunidos pelo segundo maior salário.)
   b. Se o usuário informar um valor 3, King, Ford, Scott e Jones deverão ser exibidos.
   c. Delete todas as linhas da tabela TOP_DOGS e teste o exercício.

```
DELETE FROM top_dogs;
ACCEPT p_num PROMPT 'Please enter the number of top money makers: '
DECLARE
  v_num           NUMBER(3) := &p_num;
  v_ename         emp.ename%TYPE;
  v_current_sal       emp.sal%TYPE;
  v_last_sal      emp.sal%TYPE;
  CURSOR  emp_cursor IS
  SELECT     ename, sal
  FROM            emp
  WHERE      sal IS NOT NULL
  ORDER BY        sal DESC;
BEGIN
 OPEN emp_cursor;
  FETCH emp_cursor INTO v_ename, v_current_sal;
  WHILE emp_cursor%ROWCOUNT <= v_num AND emp_cursor%FOUND LOOP
  INSERT INTO top_dogs (name, salary)
  VALUES (v_ename, v_current_sal);
  v_last_sal := v_current_sal;
  FETCH emp_cursor INTO v_ename, v_current_sal;
  IF v_last_sal = v_current_sal THEN
    v_num := v_num + 1;
  END IF;
  END LOOP;
  CLOSE emp_cursor;
  COMMIT;
END;
/
SELECT * FROM top_dogs;
```

Soluções de Exercícios 22

1. Use um cursor para recuperar o número e o nome de departamento a partir da tabela DEPT. Passe o número do departamento para outro cursor recuperar os detalhes sobre o nome do funcionário, o cargo, a data de admissão e salário de todos os funcionários que trabalham naquele departamento a partir da tabela EMP.

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR dept_cursor IS
    SELECT deptno,dname
    FROM dept
    ORDER BY    deptno;
    CURSOR emp_cursor(v_deptno NUMBER) IS
    SELECT ename,job,hiredate,sal
    FROM emp
    WHERE    deptno = v_deptno;
    v_current_deptno  dept.deptno%TYPE;
    v_current_dname dept.dname%TYPE;
    v_ename emp.ename%TYPE;
    v_job emp.job%TYPE;
    v_mgr emp.mgr%TYPE;
    v_hiredate emp.hiredate%TYPE;
    v_sal emp.sal%TYPE;
    v_line  varchar2(100);
BEGIN
    v_line := '                                                ';
    OPEN dept_cursor;
    LOOP
      FETCH dept_cursor INTO
      v_current_deptno,v_current_dname;
      EXIT WHEN dept_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE ('Department Number : ' ||
      v_current_deptno ||  '  Department Name : ' ||
      v_current_dname);
     DBMS_OUTPUT.PUT_LINE(v_line);
      IF emp_cursor%ISOPEN THEN
          CLOSE emp_cursor;
     END IF;
```

Observação: A solução continua na próxima página.

Soluções de Exercícios 22 (continuação)

```
OPEN emp_cursor (v_current_deptno);
     LOOP
       FETCH emp_cursor INTO  v_ename,v_job,v_hiredate,v_sal;
       EXIT WHEN emp_cursor%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE (v_ename || '    ' ||  v_job || '   '
      ||  v_hiredate || '   ' ||  v_sal);
     END LOOP;
    IF emp_cursor%ISOPEN THEN
          CLOSE emp_cursor;
   END IF;
   DBMS_OUTPUT.PUT_LINE(v_line);
   END LOOP;
   IF emp_cursor%ISOPEN THEN
     CLOSE emp_cursor;
  END IF;
  CLOSE dept_cursor;
  END;
 /
SET SERVEROUTPUT OFF
```

Soluções de Exercícios 22 (continuação)

2. Modifique o `p19q5.sql` para incorporar a funcionalidade FOR UPDATE e WHERE CURRENT OF no processamento do cursor.

```
SET VERIFY OFF
ACCEPT p_empno PROMPT 'Please enter the employee number: '
DECLARE
  v_empno    emp.empno%TYPE := &p_empno;
  v_asterisk emp.stars%TYPE := NULL;
  CURSOR emp_cursor IS
    SELECT   empno, NVL(ROUND(sal/100), 0) sal
    FROM          emp
    WHERE    empno = v_empno
    FOR UPDATE;
BEGIN
  FOR emp_record IN emp_cursor LOOP
    FOR i IN 1..emp_record.sal LOOP
      v_asterisk := v_asterisk ||'*';
    END LOOP;
  UPDATE emp
  SET stars = v_asterisk
  WHERE CURRENT OF emp_cursor;
 v_asterisk := NULL;
  END LOOP;
COMMIT;
END;
/
SET VERIFY ON
SQL> START p22q2.sql
SQL> SELECT  empno, sal, stars
  2  FROM    emp
  3  WHERE   empno IN (7844, 7900, 8000);
```

Soluções de Exercícios 23

1. Crie um bloco PL/SQL para selecionar o nome do funcionário com um determinado salário.
   a. Se o salário informado retornar mais de uma linha, trate a exceção com um handler de exceção apropriado e insira, na tabela MESSAGES, a mensagem "More than one employee with a salary of <*salário*>".
   b. Se o salário informado não retornar qualquer linha, trate a exceção com um handler de exceção apropriado e insira, na tabela MESSAGES, a mensagem "No employee with a salary of <*salário*>".
   c. Se o salário informado retornar apenas uma linha, insira, na tabela MESSAGES, o nome do funcionário e o valor do salário.
   d. Trate qualquer outra exceção com um handler de exceção apropriado e insira, na tabela MESSAGES, a mensagem "Some other error occurred".
   e. Teste o bloco para vários casos.

```
SET VERIFY OFF
ACCEPT p_sal PROMPT 'Please enter the salary value: '
DECLARE
  v_ename  emp.ename%TYPE;
  v_sal    emp.sal%TYPE := &p_sal;
BEGIN
  SELECT   ename
  INTO     v_ename
  FROM     emp
  WHERE         sal = v_sal;
  INSERT INTO messages (results)
  VALUES (v_ename || ' - ' || v_sal);
EXCEPTION
  WHEN no_data_found THEN
    INSERT INTO messages (results)
    VALUES ('No employee with a salary of '|| TO_CHAR(v_sal));
  WHEN too_many_rows THEN
    INSERT INTO messages (results)
    VALUES ('More than one employee with a salary of '||
             TO_CHAR(v_sal));
  WHEN others THEN
    INSERT INTO messages (results)
    VALUES ('Some other error occurred.');
END;
/
SET VERIFY ON
SQL> START p23q1.sql
SQL> START p23q1.sql
SQL> START p23q1.sql
```

Soluções de Exercícios 23 (continuação)

2. Modifique o `p18q3.sql` para adicionar um handler de exceção.
   a. Crie um handler de exceção para o erro a fim de passar uma mensagem ao usuário informando que o departamento especificado não existe.
   b. Execute o bloco PL/SQL informando um departamento que não existe.

```
SET VERIFY OFF
VARIABLE g_message VARCHAR2(40)
ACCEPT p_deptno PROMPT 'Please enter the department number: '
ACCEPT p_loc PROMPT 'Please enter the department location: '
DECLARE
  e_invalid_dept    EXCEPTION;
  v_deptno      dept.deptno%TYPE := &p_deptno;
BEGIN
  UPDATE dept
  SET loc = '&p_loc'
  WHERE deptno = v_deptno;
  IF SQL%NOTFOUND THEN
    raise e_invalid_dept;
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_dept THEN
    :g_message := 'Department '|| TO_CHAR(v_deptno) ||
                  ' is an invalid department';
END;
/
SET VERIFY ON
PRINT g_message
SQL> START p23q2.sql
```

Soluções de Exercícios 23 (continuação)

```
SET VERIFY OFF
ACCEPT p_deptno PROMPT 'Please enter the department number: '
ACCEPT p_loc PROMPT 'Please enter the department location: '
DECLARE
  e_invalid_dept EXCEPTION;
  v_deptno       dept.deptno%TYPE := &p_deptno;
BEGIN
  UPDATE dept
  SET loc = '&p_loc'
  WHERE deptno = v_deptno;
  IF SQL%NOTFOUND THEN
    raise e_invalid_dept;
  END IF;
  COMMIT;

EXCEPTION
  WHEN e_invalid_dept THEN
    dbms_output.put_line('Department '|| TO_CHAR(v_deptno) ||
                  ' is an invalid department');
END;
/
SET VERIFY ON
```

Soluções de Exercícios 23 (continuação)

3. Crie um bloco PL/SQL que imprima o número de funcionários que recebem mais ou menos que US$100 do salário informado.
   a. Se não houver funcionários nessa faixa salarial, imprima uma mensagem para o usuário indicando que é o caso. Use uma exceção para esse caso.
   b. Se houver um ou mais funcionários nessa faixa, a mensagem deverá indicar quantos funcionários estão naquela faixa salarial.
   c. Trate qualquer outra exceção com um handler de exceção apropriado. A mensagem deve indicar que ocorreu um erro.

```
VARIABLE g_message VARCHAR2(100)
SET VERIFY OFF
ACCEPT p_sal PROMPT 'Please enter the salary: '
DECLARE
  v_sal              emp.sal%TYPE := &p_sal;
  v_low_sal               emp.sal%TYPE := v_sal - 100;
  v_high_sal         emp.sal%TYPE := v_sal + 100;
  v_no_emp      NUMBER(7);
  e_no_emp_returned EXCEPTION;
  e_more_than_one_emp    EXCEPTION;
BEGIN
  SELECT   count(ename)
  INTO     v_no_emp
  FROM     emp
  WHERE    sal between v_low_sal and v_high_sal;
  IF v_no_emp  = 0 THEN
    RAISE e_no_emp_returned;
  ELSIF v_no_emp > 0 THEN
    RAISE e_more_than_one_emp;
  END IF;
EXCEPTION
  WHEN e_no_emp_returned THEN
    :g_message := 'There is no employee salary between '||
                  TO_CHAR(v_low_sal) || ' and '||
                      TO_CHAR(v_high_sal);
  WHEN e_more_than_one_emp THEN
    :g_message := 'There is/are '|| TO_CHAR(v_no_emp) ||
                  ' employee(s) with a salary between '||
                TO_CHAR(v_low_sal) || ' and '||
                TO_CHAR(v_high_sal);
END;
/
SET VERIFY ON
PRINT g_message
SQL> START p23q3.sql
```

Soluções de Exercícios 23 (continuação)

```
SET VERIFY OFF
ACCEPT p_sal PROMPT 'Please enter the salary: '
DECLARE
  v_sal          emp.sal%TYPE := &p_sal;
  v_low_sal      emp.sal%TYPE := v_sal - 100;
  v_high_sal     emp.sal%TYPE := v_sal + 100;
  v_no_emp       NUMBER(7);
  e_no_emp_returned   EXCEPTION;
  e_more_than_one_emp EXCEPTION;
BEGIN
  SELECT     count(ename)
  INTO       v_no_emp
  FROM       emp
  WHERE      sal between v_low_sal and v_high_sal;
  IF v_no_emp  = 0 THEN
    RAISE e_no_emp_returned;
  ELSIF v_no_emp > 0 THEN
    RAISE e_more_than_one_emp;
  END IF;
EXCEPTION
  WHEN e_no_emp_returned THEN
    dbms_output.put_line('There is no employee salary
                  between '|| TO_CHAR(v_low_sal) || ' and '||
                  TO_CHAR(v_high_sal));
  WHEN e_more_than_one_emp THEN
    dbms_output.put_line('There is/are '|| TO_CHAR(v_no_emp) ||
                   ' employee(s) with a salary between '||
                  TO_CHAR(v_low_sal) || ' and '||
                  TO_CHAR(v_high_sal));
  WHEN others THEN
    dbms_output.put_line('Some other error occurred.');
END;
/
SET VERIFY ON
```

B

Descrições da Tabela e Dados

Tabela EMP

```
SQL> DESCRIBE emp
```

```
Name                            Null?    Type
------------------------------- -------- ----
EMPNO                           NOT NULL NUMBER(4)
ENAME                                    VARCHAR2(10)
JOB                                      VARCHAR2(9)
MGR                                      NUMBER(4)
HIREDATE                                 DATE
SAL                                      NUMBER(7,2)
COMM                                     NUMBER(7,2)
DEPTNO                          NOT NULL NUMBER(2)
```

```
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17-NOV-81	5000		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Tabela DEPT

```
SQL> DESCRIBE dept
```

```
Name                           Null?    Type
------------------------------ -------- ----
DEPTNO                         NOT NULL NUMBER(2)
DNAME                                   VARCHAR2(14)
LOC                                     VARCHAR2(13)
```

```
SQL> SELECT * FROM dept;
```

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Tabela SALGRADE

```
SQL> DESCRIBE salgrade


Name                            Null?    Type
------------------------------- -------- ----
GRADE                                    NUMBER
LOSAL                                    NUMBER
HISAL                                    NUMBER


SQL> SELECT * FROM salgrade;


     GRADE      LOSAL      HISAL
---------- ---------- ----------
         1        700       1200
         2       1201       1400
         3       1401       2000
         4       2001       3000
         5       3001       9999
```

Tabela ORD

```
SQL> DESCRIBE ord
```

```
Name                            Null?    Type
------------------------------- -------- ----
ORDID                           NOT NULL NUMBER(4)
ORDERDATE                                DATE
COMMPLAN                                 VARCHAR2(1)
CUSTID                          NOT NULL NUMBER(6)
SHIPDATE                                 DATE
TOTAL                                    NUMBER(8,2)
```

```
SQL> SELECT * FROM ord;
```

ORDID	ORDERDATE	C	CUSTID	SHIPDATE	TOTAL
610	07-JAN-87	A	101	08-JAN-87	101.4
611	11-JAN-87	B	102	11-JAN-87	45
612	15-JAN-87	C	104	20-JAN-87	5860
601	01-MAY-86	A	106	30-MAY-86	2.4
602	05-JUN-86	B	102	20-JUN-86	56
604	15-JUN-86	A	106	30-JUN-86	698
605	14-JUL-86	A	106	30-JUL-86	8324
606	14-JUL-86	A	100	30-JUL-86	3.4
609	01-AUG-86	B	100	15-AUG-86	97.5
607	18-JUL-86	C	104	18-JUL-86	5.6
608	25-JUL-86	C	104	25-JUL-86	35.2
603	05-JUN-86		102	05-JUN-86	224
620	12-MAR-87		100	12-MAR-87	4450
613	01-FEB-87		108	01-FEB-87	6400
614	01-FEB-87		102	05-FEB-87	23940
616	03-FEB-87		103	10-FEB-87	764
619	22-FEB-87		104	04-FEB-87	1260
617	05-FEB-87		105	03-MAR-87	46370
615	01-FEB-87		107	06-FEB-87	710
618	15-FEB-87	A	102	06-MAR-87	3510.5
621	15-MAR-87	A	100	01-JAN-87	730

Tabela PRODUCT

```
SQL> DESCRIBE product


Name                            Null?    Type
------------------------------- -------- ----
PRODID                           NOT NULL NUMBER(6)
DESCRIP                                   VARCHAR2(30)


SQL> SELECT * FROM product;


   PRODID DESCRIP
--------- ------------------------------
   100860 ACE TENNIS RACKET I
   100861 ACE TENNIS RACKET II
   100870 ACE TENNIS BALLS-3 PACK
   100871 ACE TENNIS BALLS-6 PACK
   100890 ACE TENNIS NET
   101860 SP TENNIS RACKET
   101863 SP JUNIOR RACKET
   102130 RH: "GUIDE TO TENNIS"
   200376 SB ENERGY BAR-6 PACK
   200380 SB VITA SNACK-6 PACK
```

Tabela ITEM

```
SQL> DESCRIBE item
```

```
Name                           Null?    Type
------------------------------ -------- -----------
ORDID                          NOT NULL NUMBER(4)
ITEMID                         NOT NULL NUMBER(4)
PRODID                                  NUMBER(6)
ACTUALPRICE                             NUMBER(8,2)
QTY                                     NUMBER(8)
ITEMTOT                                 NUMBER(8,2)
```

```
SQL> SELECT * FROM item;
```

ORDID	ITEMID	PRODID	ACTUALPRICE	QTY	ITEMTOT
610	3	100890	58	1	58
611	1	100861	45	1	45
612	1	100860	30	100	3000
601	1	200376	2.4	1	2.4
602	1	100870	2.8	20	56
604	1	100890	58	3	174
604	2	100861	42	2	84
604	3	100860	44	10	440
603	2	100860	56	4	224
610	1	100860	35	1	35
610	2	100870	2.8	3	8.4
613	4	200376	2.2	200	440
614	1	100860	35	444	15540
614	2	100870	2.8	1000	2800
612	2	100861	40.5	20	810
612	3	101863	10	150	1500
620	1	100860	35	10	350
620	2	200376	2.4	1000	2400
620	3	102130	3.4	500	1700
613	1	100871	5.6	100	560
613	2	101860	24	200	4800
613	3	200380	4	150	600
619	3	102130	3.4	100	340
617	1	100860	35	50	1750
617	2	100861	45	100	4500
614	3	100871	5.6	1000	5600

Continuação na próxima página

Tabela ITEM (continuação)

ORDID	ITEMID	PRODID	ACTUALPRICE	QTY	ITEMTOT
616	1	100861	45	10	450
616	2	100870	2.8	50	140
616	3	100890	58	2	116
616	4	102130	3.4	10	34
616	5	200376	2.4	10	24
619	1	200380	4	100	400
619	2	200376	2.4	100	240
615	1	100861	45	4	180
607	1	100871	5.6	1	5.6
615	2	100870	2.8	100	280
617	3	100870	2.8	500	1400
617	4	100871	5.6	500	2800
617	5	100890	58	500	29000
617	6	101860	24	100	2400
617	7	101863	12.5	200	2500
617	8	102130	3.4	100	340
617	9	200376	2.4	200	480
617	10	200380	4	300	1200
609	2	100870	2.5	5	12.5
609	3	100890	50	1	50
618	1	100860	35	23	805
618	2	100861	45.11	50	2255.5
618	3	100870	45	10	450
621	1	100861	45	10	450
621	2	100870	2.8	100	280
615	3	100871	5	50	250
608	1	101860	24	1	24
608	2	100871	5.6	2	11.2
609	1	100861	35	1	35
606	1	102130	3.4	1	3.4
605	1	100861	45	100	4500
605	2	100870	2.8	500	1400
605	3	100890	58	5	290
605	4	101860	24	50	1200
605	5	101863	9	100	900
605	6	102130	3.4	10	34
612	4	100871	5.5	100	550
619	4	100871	5.6	50	280

Tabela CUSTOMER

```
SQL> DESCRIBE customer

Name                                      Null?    Type
----------------------------------------- -------- ----
CUSTID                                    NOT NULL NUMBER(6)
NAME                                               VARCHAR2(45)
ADDRESS                                            VARCHAR2(40)
CITY                                               VARCHAR2(30)
STATE                                              VARCHAR2(2)
ZIP                                                VARCHAR2(9)
AREA                                               NUMBER(3)
PHONE                                              VARCHAR2(9)
REPID                                     NOT NULL NUMBER(4)
CREDITLIMIT                                        NUMBER(9,2)
COMMENTS                                           LONG
```

Tabela CUSTOMER (continuação)

```
SQL> SELECT * FROM customer;
```

CUSTID	NAME	ADDRESS
100	JOCKSPORTS	345 VIEWRIDGE
101	TKB SPORT SHOP	490 BOLI RD.
102	VOLLYRITE	9722 HAMILTON
103	JUST TENNIS	HILLVIEW MALL
104	EVERY MOUNTAIN	574 SURRY RD.
105	K + T SPORTS	3476 EL PASEO
106	SHAPE UP	908 SEQUOIA
107	WOMENS SPORTS	VALCO VILLAGE
108	NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER	98 LONE PINE WAY

CITY	ST	ZIP	AREA	PHONE	REPID	CREDITLIMIT
BELMONT	CA	96711	415	598-6609	7844	5000
REDWOOD CITY	CA	94061	415	368-1223	7521	10000
BURLINGAME	CA	95133	415	644-3341	7654	7000
BURLINGAME	CA	97544	415	677-9312	7521	3000
CUPERTINO	CA	93301	408	996-2323	7499	10000
SANTA CLARA	CA	91003	408	376-9966	7844	5000
PALO ALTO	CA	94301	415	364-9777	7521	6000
SUNNYVALE	CA	93301	408	967-4398	7499	10000
HIBBING	MN	55649	612	566-9123	7844	8000

COMMENTS
Very friendly people to work with -- sales rep likes to be called Mike.
Rep called 5/8 about change in order - contact shipping.
Company doing heavy promotion beginning 10/89. Prepare for large orders during orders during winter
Contact rep about new line of tennis rackets.
Customer with high market share (23%) due to aggressive advertising.
Tends to order large amounts of merchandise at once. Accounting is considering raising their credit limit
Support intensive. Orders small amounts (< 800) of merchandise at a time.
First sporting goods store geared exclusively towards women. Unusual promotional style

Tabela PRICE

```
SQL> DESCRIBE price

Name                            Null?    Type
------------------------------- -------- ----
PRODID                          NOT NULL NUMBER(6)
STDPRICE                                 NUMBER(8,2)
MINPRICE                                 NUMBER(8,2)
STARTDATE                                DATE
ENDDATE                                  DATE

SQL> SELECT * FROM price;

   PRODID  STDPRICE  MINPRICE STARTDATE ENDDATE
--------- --------- --------- --------- ---------
   100871       4.8       3.2 01-JAN-85 01-DEC-85
   100890        58      46.4 01-JAN-85
   100890        54      40.5 01-JUN-84 31-MAY-84
   100860        35        28 01-JUN-86
   100860        32      25.6 01-JAN-86 31-MAY-86
   100860        30        24 01-JAN-85 31-DEC-85
   100861        45        36 01-JUN-86
   100861        42      33.6 01-JAN-86 31-MAY-86
   100861        39      31.2 01-JAN-85 31-DEC-85
   100870       2.8       2.4 01-JAN-86
   100870       2.4       1.9 01-JAN-85 01-DEC-85
   100871       5.6       4.8 01-JAN-86
   101860        24        18 15-FEB-85
   101863      12.5       9.4 15-FEB-85
   102130       3.4       2.8 18-AUG-85
   200376       2.4      1.75 15-NOV-86
   200380         4       3.2 15-NOV-86
```

Índice

B

C

D

E

F

L

M

N

O

P

W