

UNIP
SISTEMA DE BANCO DE DADOS
PROF: LY FREITAS

PL / SQL
PL / SQL

AGOSTO / 2002

ÍNDICE

1. SELECIONANDO LINHAS	5
1.1. COMANDOS SQL.....	6
1.2. BLOCO BÁSICO DE CONSULTA(QUERY)	6
1.3. SINTAXE.....	6
1.4. SELECIONADO TODAS AS COLUNAS E LINHAS.....	7
1.5. SELECIONANDO ALGUMAS COLUNAS.....	7
1.6. EXPRESSÕES ARITMÉTICAS	9
1.7. APELIDOS(ALIAS) DE COLUNA	9
1.8. OPERADOR DE CONCATENAÇÃO.....	10
1.9. COMO EVITAR SELEÇÃO DE LINHAS DUPLICADAS	11
1.10. EXIBINDO A ESTRUTURA DA TABELA.....	11
2. LIMITANDO LINHAS SELECIONADAS	12
2.1. ORDENANDO LINHAS COM A CLÁUSULA ORDER BY	13
2.2. ORDENANDO POR POSIÇÃO	14
2.3. ORDENANDO POR VÁRIAS COLUNAS.....	14
2.4. LIMITANDO AS LINHAS COM CLÁUSULA WHERE	15
2.4.1. Operadores de Comparação Lógica	15
2.4.2. Operadores Comparação SQL	16
2.4.3. Operadores Lógicos	16
2.4.4. Operadores Lógicos de Negação.....	16
2.4.5. Operadores Comparação SQL	16
3. SCRIPTS E SPOOLS.....	22
3.1. COMANDOS DE ARQUIVO	23
3.2. COMO CRIAR UM SCRIPT	23
3.2.1. 1ª Forma	23
3.2.2. 2ª Forma	23
3.3. PARA CRIAR UM SPOOL:	24
3.4. EXECUTAR SCRIPT COM SAÍDA EM SPOOL:	24
4. JOIN	25
4.1. MÉTODOS DE JOIN:.....	26
4.2. MÉTODOS ADICIONAIS	26
4.3. CONSULTA COM JOIN SIMPLES	26
4.4. PESQUISA DE JOIN SIMPLES	27
4.4.1. Equijoin	27
4.5. CONDIÇÕES ADICIONAIS DE PESQUISA USANDO O OPERADOR AND	28
4.6. ALIAS DE TABELA	29
4.7. RETORNANDO REGISTROS SEM CORRESPONDÊNCIA DIRETA.....	29
4.8. UNINDO UMA TABELA A SI MESMA	30
4.9. SELF JOIN	30
5. SUBQUERIES	32
5.1. REGRAS GERAIS	33
6. VISÃO GERAL DE MODELAGEM DE DADOS – O PROJETO DO BD.....	35
6.1. INTRODUÇÃO.....	36
6.2. CICLO DE DESENVOLVIMENTO DO SISTEMA.....	36
6.3. ESTÁGIOS DE DESENVOLVIMENTO	36
6.3.1. Estratégia e Análise.....	36
6.3.2. Projeto	36
6.3.3. Elaboração e Documentação	36
6.3.4. Transição	37
6.3.5. Produção	37
6.4. PROJETO DE BANCO DE DADOS	37
6.5. DESEMPENHO	37
6.6. APLICAÇÃO INTEGRADA.....	37

6.7. INTEGRAÇÃO COM OUTROS SISTEMAS	37
6.8. DOCUMENTAÇÃO E COMUNICAÇÃO	38
6.9. ESCALABILIDADE	38
6.10. EVITE A REINVENÇÃO DA RODA	38
6.11. MODELO DE DADOS	38
6.11.1. <i>Propósito dos Modelos</i>	38
6.12. MODELO DE ENTIDADE-RELACIONAMENTO.....	38
6.12.1. <i>Entidades</i>	39
6.12.2. <i>Atributos</i>	39
6.12.3. <i>Identificadores Únicos</i>	39
6.13. CONVENÇÕES DE MODELAGEM DE ENTIDADE-RELACIONAMENTO	40
6.14. RELACIONAMENTOS	40
6.15. RELACIONAMENTOS RECURSIVOS.....	41
6.16. CONVENÇÕES DE DIAGRAMAÇÃO DE RELACIONAMENTO	41
6.17. SUBCONJUNTO DO MODELO DE ENTIDADE-RELACIONAMENTO	42
6.18. TIPOS DE RELACIONAMENTO.....	42
6.18.1. <i>Um para um</i>	42
6.18.2. <i>Muitos para um</i>	42
6.18.3. <i>Muitos para muitos</i>	42
6.19. CONSTRAINTS E CHAVES DE INTEGRIDADE	43
6.19.1. <i>Chaves Primárias</i>	43
6.19.2. <i>Chaves Candidatas</i>	43
6.19.3. <i>Chaves Estrangeiras</i>	44
7. CRIAÇÃO DE TABELAS.....	45
7.1. CONSTRAINTS DE INTEGRIDADE DE DADOS.....	46
7.2. CRIANDO TABELAS	47
7.2.1. <i>Sintaxe Abreviada</i>	47
7.3. DEFININDO AS CONSTRAINTS.....	47
8. MANIPULANDO DADOS	49
8.1. ADICIONANDO UMA NOVA LINHA EM UMA TABELA	50
8.2. ATUALIZANDO LINHAS	51
8.3. DELETANDO LINHAS	52
8.4. PROCESSAMENTO DE TRANSAÇÕES	52
8.4.1. <i>Tipos de Transações</i>	53
8.4.2. <i>Quando Inicia e Termina uma Transação?</i>	53
8.4.3. <i>Comandos Explícitos de Controle de Transação</i>	53
8.4.4. <i>Processamento Implícito de Transações</i>	54
8.5. SUBMETENDO ALTERAÇÕES.....	54
8.5.1. <i>Estado dos Dados Antes de um COMMIT ou ROLLBACK</i>	54
8.6. FAZENDO O ROLLBACK DA ALTERAÇÕES.....	55
9. ALTERANDO TABELAS.....	56
9.1. ADICIONANDO UMA COLUNA	57
9.1.1. <i>Regras Gerais</i>	57
9.2. ALTERANDO UMA COLUNA	58
9.2.1. <i>Regras Gerais</i>	58
9.3. ELIMINANDO UMA TABELA	58
9.3.1. <i>Regras Gerais</i>	59
9.4. RENOMEANDO E TRUNCANDO UMA TABELA	59
10. VISÃO GERAL DE PL /SQL.....	60
10.1. BENEFÍCIOS DO PL/SQL.....	61
10.1.1. <i>Estrutura de Bloco PL/SQL</i>	61
10.2. CONSTRUÇÕES DO PROGRAMA PL/SQL.....	61
10.2.1. <i>Bloco Anônimo</i>	61
10.2.2. <i>Subprogramas</i>	61
10.3. TIPOS DE BLOCO	62
10.3.1. <i>Anônimo</i>	62

10.3.2. Procedure	62
10.3.3. Function.....	62
11. DESENVOLVENDO UM.....	63
BLOCO PL / SQL	63
11.1. DECLARANDO VARIÁVEIS E CONSTANTES DE PL/SQL	64
11.2. TIPOS DE DADOS ESCALARES.....	64
11.3. ATRIBUINDO VALORES A VARIÁVEIS	65
11.4. FUNCTIONS.....	66
11.5. CONVERSÃO DE TIPO DE DADO	67
11.6. FAZENDO REFERÊNCIAS A VARIÁVEIS NÃO-PL/SQL.....	67
12. INTERAGINDO COM ORACLE	68
12.1. RECUPERANDO DADOS USANDO PL/SQL	69
12.2. SINTAXE RESUMIDA	69
12.3. REGRAS GERAIS	69
12.4. EXCEPTION TOO_MANY_ROWS	70
12.5. EXCEPTION NO_DATA_FOUND.....	71
12.6. MANIPULANDO DADOS USANDO PL/SQL.....	71
12.6.1. Inserindo Dados	71
12.6.2. Atualizando e Deletando Dados	71
13. CONTROLANDO O FLUXO EM BLOCOS PL / SQL	73
13.1. COMANDO IF	74
13.2. COMANDOS LOOP.....	74
13.2.1. Loop Básico	74
13.3. LOOP FOR.....	75
13.4. LOOP WHILE	76

1. Seleccionando Linhas

O SQL (Structured Query Language ou Linguagem Estruturada de Consultas) é uma linguagem própria para realização de operações relacionais. Esta linguagem permite recuperar, atualizar ou eliminar dados do BD (Banco de Dados) relacional e criar ou modificar a estrutura do BD.

1.1. Comandos SQL

Há vários comandos disponíveis no SQL. A tabela abaixo descreve os comandos abrangidos nesta apostila.

Comando	Descrição
SELECT	Recupera os dados do Banco de Dados. É o comando mais usado.
INSERT UPDATE DELETE	Insere novas linhas, altera as existentes e remove as linhas indesejadas de tabelas do banco de dados.
CREATE ALTER DROP RENAME TRUNCATE	Define, altera e remove estruturas de dados das tabelas.

1.2. Bloco Básico de Consulta (Query)

Um comando SELECT recupera as informações do banco de dados, implementando todos os operadores algébricos.

1.3. Sintaxe

```
SELECT [DISTINCT] {*,coluna [alias],...}  
FROM tabela ;
```

Onde:	SELECT	lista de, no mínimo, uma coluna.
	DISTINCT	remove duplicatas.
	*	seleciona todas as colunas
	coluna	seleciona a coluna identificada.
	alias	atribui um cabeçalho diferente às colunas selecionadas
	FROM tabela	especifica a tabela que contém as colunas

1.4. Selecionado Todas as Colunas e Linhas

O asterisco(*) seleciona todas as colunas da tabela.

Exemplo:

Seja a tabela abaixo S_DEPT

<u>ID</u>	<u>NAME</u>	<u>REGION_ID</u>
10	Finance	1
31	Sales	1
32	Sales	2
33	Sales	3
34	Sales	4
35	Sales	5
41	Operations	1
42	Operations	2
43	Operations	3
44	Operations	4
45	Operations	5
50	Administration	1

12 rows selected.

Listagem de todas as colunas e todas as linhas da tabela S_DEPT.

```
SQL>SELECT * FROM s_dept; ou
      SELECT id, name, region_id FROM s_dept;
```

<u>ID</u>	<u>NAME</u>	<u>REGION_ID</u>
10	Finanças	1
31	Vendas	1
32	Vendas	2
33	Vendas	3
34	Vendas	4
41	Operações	1
42	Operações	2
43	Operações	3
50	Administração	1

09 rows selected.

1.5. Selecionando Algumas Colunas

Você pode limitar a consulta(Query) para exibir **determinadas** colunas, especificando seus nomes, separados por vírgulas, na cláusula SELECT.

Exemplo:

Seja a tabela abaixo S_EMP.

ID	LAST_NAME	FIRST_NAME	SALARY	COMMISSION_PCT
1	Velasquez	Carmen	2500	
2	Ngao	LaDoris	1450	
3	Nagayama	Midori	1400	
4	Quick-To-See	Mark	1450	
5	Ropeburn	Audry	1550	
6	Urguhart	Molly	1200	
7	Menchu	Roberta	1250	
8	Biri	Ben	1100	
9	Catchpole	Antoinette	1300	
10	Havel	Marta	1307	
11	Magee	Colin	1400	10
12	Giljum	Henry	1490	12.5
13	Sedeghi	Yasmin	1515	10
14	Nguyen	Mai	1525	15
15	Dumas	Andre	1450	17.5
16	Maduro	Elena	1400	

Exibição de todos os **sobrenomes** e **salários** da tabela S_EMP

SQL>SELECT last_name, salary FROM s_emp;

LAST_NAME	SALARY
Velasquez	2500
Ngao	1450
Nagayama	1400
Quick-To-See	1450
Ropeburn	1550
Urguhart	1200
Menchu	1250
Biri	1100
Catchpole	1300
Havel	1307
Magee	1400
Giljum	1490
Sedeghi	1515
Nguyen	1525
Dumas	1450
Maduro	1400

1.6. Expressões Aritméticas

Operadores Aritméticos	
+	Adição
-	Subtração
*	Multiplificação
/	Divisão

Estes operadores podem ser utilizados nos atributos de saída das queries.

Exemplo 1:

Visualizar os salários dos empregados acrescidos de 10%.

```
SQL>SELECT last_name, salary * 1.10 FROM s_emp;
```

LAST_NAME	SALARY*1.10
Velasquez	2750
Nagayama	1540
Ropeburn	1705
Menchu	1375
Ngao	1595
Quick-To-See	1595
Urguhart	1320

Exemplo 2:

Visualizar os salários dos empregados acrescidos de R\$ 100,00.

```
SQL>SELECT last_name, salary + 100 FROM s_emp;
```

LAST_NAME	SALARY+100
Velasquez	2600
Nagayama	1500
Ropeburn	1650
Menchu	1350
Ngao	1550
Quick-To-See	1550
Urguhart	1300

1.7. Apelidos(alias) de Coluna

Ao exibir o resultado de uma coluna, o SQL*Plus normalmente usa o nome da coluna selecionada como cabeçalho. Em vários casos, o cabeçalho pode ser de difícil

entendimento, ou mesmo não Ter sentido algum. Você pode então alterar o cabeçalho de uma coluna usando um alias para a coluna.

Especifique o alias depois da coluna, na lista SELECT, usando um espaço como separador. Por default, os cabeçalhos dos apelidos (alias) serão obrigatoriamente em letras maiúsculas e não poderão conter espaços em branco, a menos que o alias fique entre aspas (“”).

Exemplo:

Exibição do sobrenome, salário e remuneração anual dos funcionários. Cálculo da remuneração anual com o salário mensal mais um bônus mensal de R\$ 100,00, multiplicado por 12. Daremos o nome de SALARIO_ANUAL para a coluna.

```
SQL>SELECT last_name, salary,  
2          12 * (salary + 100) AS SALARIO_ANUAL  
3          FROM s_emp;
```

OU

```
SQL>SELECT last_name, salary,  
2          12 * (salary + 100) “Salário Anual”  
3          FROM s_emp;
```

1.8. Operador de Concatenação

Podemos ligar colunas a outras colunas, expressões aritméticas ou valores constantes para criar uma expressão de caracteres, usando o operador de concatenação(||). As colunas em cada lado do operador são combinadas para formar uma coluna única de saída.

Exemplo:

Liste os nomes completos dos funcionários usando o cabeçalho “Empregados”.

```
SQL>SELECT first_name || last_name AS “Empregados” FROM s_emp;
```

Empregados

```
-----  
CarmenVelasquez  
LaDorisNgao  
MidoriNagayama  
MarkQuick-To-See  
AudryRopeburn  
MollyUrguhart  
RobertaMenchu  
BenBiri  
AntoinetteCatchpole  
MartaHavel  
ColinMagee
```

1.9. Como Evitar Seleção de Linhas Duplicadas

```
SQL>SELECT name FROM s_dept;
```

```

      NAME
-----
 Finance
 Sales
 Sales
 Sales
 Sales
 Sales
 Sales
 Operations
 Operations
 Operations

```

Para eliminar linhas duplicadas no resultado, inclui-se a palavra **DISTINCT** na cláusula SELECT imediatamente após a palavra de comando SELECT.

Logo a query acima ficaria assim:

```
SQL>SELECT DISTINCT name FROM s_dept;
```

```

      NAME
-----
Administration
 Finance
 Operations
 Sales

```

1.10. Exibindo a Estrutura da Tabela

Para se exibir a estrutura de uma tabela usa-se o comando **DESCRIBE** ou **DESC**. Como resultado, você poderá ver os nomes das colunas, tipos de dados e se uma coluna deve conter dados.

DESC[RIBE] *tablename*

Exemplo:

Mostre a estrutura da tabela S_DEPT.

```
SQL>DESCRIBE s_dept OU SQL>DESC s_dept
```

NAME	NULL?	TYPE
id	not null	number(7)
name	not null	varchar2(25)
region_id		number(7)

2. Limitando Linhas Seleccionadas

Ao recuperar dados de um banco de dados, pode ser necessário restringir quais linhas de dados serão exibidas ou especificar a ordem de exibição. Neste capítulo veremos como devem ser utilizados os comandos para realizar essas ações.

2.1. Ordenando Linhas com a Cláusula ORDER BY

A cláusula ORDER BY nos permite classificar as linhas do resultado alfabética e numericamente, em ordem crescente ou decrescente. O default é a ordem crescente. Esta cláusula é sempre a última da query.

Sintaxe

```
SELECT  expr
FROM    tabela
ORDER BY {coluna,expr} [ASC|DESC];
```

Onde: **ORDER BY** especifica a ordem na qual as linhas retornadas são exibidas.
 ASC ordena as linhas em ordem ascendente.
 DESC ordena as linhas em ordem descendente

Exemplo 1

```
SQL>SELECT  last_name, dept_id, start_date
2  FROM      s_dept
3  ORDER BY last_name;
```

LAST_NAME	DEPT_ID	START_DAT
Biri	43	07-APR-90
Catchpole	44	09-FEB-92
Chang	44	30-NOV-90
Dancs	45	09-OCT-91
Dumas	35	18-JAN-92
Giljum	32	27-FEB-91

A ordem default de classificação é ascendente:

- Os valores numéricos são exibidos a partir dos valores mais baixos, como 1 – 999;
- Os valores de data são exibidos começando com a data mais antiga, por exemplo 01-JAN-92 antes 01-JAN-95.
- Os valores de caracteres são exibidos em ordem alfabética, como de A à Z.
- No Oracle7, os valores nulos são exibidos por último para sequences ascendentes e em primeiro para sequences descendentes.

Exemplo 2

```
SQL>SELECT    last_name, dept_id, start_date
2  FROM      s_dept
3  ORDER BY  last_name DESC;
```

LAST_NAME	DEPT_ID	START_DAT
-----	-----	-----
Velasquez	50	03-MAR-90
Urguhart	41	18-JAN-91
Smith	41	08-MAR-90
Sedeghi	33	18-FEB-91
Schwartz	45	09-MAY-91
Ropeburn	50	04-MAR-90
Quick-To-See	10	07-APR-90
Patel	42	06-AUG-91

2.2. Ordenando por Posição

Outro método para ordenar os resultados da consulta é a classificação de acordo com a posição. Esse método é especialmente útil para ordenar uma expressão longa. Ao invés de digitar a expressão novamente, você pode especificar sua posição na lista SELECT.

Exemplo 3:

```
SQL>SELECT    last_name, salary*12
2  FROM      s_emp
3  ORDER BY  2;
```

LAST_NAME	SALARY*12
-----	-----
Newman	9000
Patel	9540
Patel	9540
Chang	9600
Markarian	10200
Dancs	10320
Smith	11280
Biri	13200
Schwartz	13200

2.3. Ordenando por Várias Colunas

Você pode ordenar os resultados da consulta de acordo com mais de uma coluna. O limite de classificação é o número de colunas na tabela.

Exemplo 4:

Ordenação do resultado pelo número do departamento e então na ordem descendente, por salário.

```
SQL>SELECT last_name, dept_id, salary
2 FROM s_emp
3 ORDER BY dept_id, salary DESC;
```

LAST_NAME	DEPT_ID	SALARY
-----	-----	-----
Quick-To-See	10	1450
Nagayama	31	1400
Magee	31	1400
Giljum	32	1490
Sedeghi	33	1515
Nguyen	34	1525
Patel	34	795

2.4. Limitando as Linhas com Cláusula WHERE

Usamos a cláusula WHERE para especificar as linhas (tuplas) desejadas em uma query.

Sintaxe:

```
SELECT expr
FROM tabela
WHERE condição
ORDER BY {coluna,expr} [ASC|DESC];
```

Onde:

WHERE limita a consulta às linhas que satisfazem uma condição

condição é composta dos nomes de coluna, expressões, constantes e operadores de comparação

Podemos utilizar qualquer dos operadores relacionais na cláusula WHERE.

2.4.1. Operadores de Comparação Lógica

Operador	Significado
=	Igual a
>	Maior que
>=	Maior ou igual a
<	Menor que
<=	Menor ou igual a

2.4.2. Operadores Comparação SQL

Operador	Significado
BETWEEN...AND...	Entre dois valores (inclusive)
IN(<i>list</i>)	Satisfaz todas de uma lista de valores
LIKE	Satisfaz um padrão de caracter
IS NULL	É um valor nulo

2.4.3. Operadores Lógicos

Operador	Significado
AND	Se ambas condições componentes retornarem TRUE, o resultado é TRUE.
OR	Se uma condições componentes retornarem TRUE, o resultado é TRUE.
NOT	Retorna a condição oposta.

2.4.4. Operadores Lógicos de Negação

Operador	Significado
!=	Diferente de (VAX,UNIX,PC)
^=	Diferente de(IBM)
:=	
<>	Diferente de(todos os S.O.)
<	Menor que
<=	Menor ou igual a

2.4.5. Operadores Comparação SQL

Operador	Significado
NOT BETWEEN...AND...	Não entre dois valores especificados
NOT IN(<i>list</i>)	Não na lista de valores especificados
NOT LIKE	Não como cadeia de comparação
IS NOT NULL	Não é um valor nulo

Exemplo 1:

Exibir o nome, sobrenome e cargo do funcionário de sobrenome "Magee".

```
SQL>SELECT    first_name, last_name, title
2  FROM      s_emp
3  WHERE     last_name = 'Magee';
```

```
FIRST_NAME      LAST_NAME      TITLE
-----
Colin           Magee         Sales Representative
```

Exemplo 2:

Exibir os funcionários que tenham salário maior que R\$ 1000,00.

```
SQL>SELECT first_name, salary
2 FROM s_emp
3 WHERE salary >1000;
```

FIRST_NAME	SALARY
Carmen	2500
LaDoris	1450
Midori	1400
Mark	1450
Audry	1550
Molly	1200

Exemplo 3:

Exibir os funcionários que trabalham no departamento de número 41.

```
SQL>SELECT first_name, dept_id
2 FROM s_emp
3 WHERE dept_id = '41';
```

FIRST_NAME	DEPT_ID
LaDoris	41
Molly	41
Elena	41
George	41

Exemplo 4:

Exibir todos os funcionários que recebem salário menor ou igual a R\$ 800,00.

```
SQL>SELECT first_name, salary
2 FROM s_emp
3 WHERE salary <= 800;
```

FIRST_NAME	SALARY
Vikram	795
Chad	750
Eddie	800
Radha	795

Exemplo 5:

Exibir todos os funcionários que recebam salário maior que R\$ 1200,00 e pertençam ao departamento de código 41.

```
SQL>SELECT    first_name, salary, dept_id
 2  FROM      s_emp
 3  WHERE     salary > 1200 AND
 4           dept_id = '41';
```

FIRST_NAME	DEPT_ID	SALARY
LaDoris	41	1450
Elena	41	1400

Exemplo 6:

Exibir os funcionários que recebam salário entre R\$ 1200 e R\$ 1400 inclusive.

```
SQL>SELECT    first_name, salary
 2  FROM      s_emp
 3  WHERE     salary BETWEEN 1200 AND 1400;
```

FIRST_NAME	SALARY
Midori	1400
Molly	1200
Roberta	1250
Antoinette	1300
Marta	1307
Colin	1400

Exemplo 7:

Exibir o primeiro nome e data de admissão dos funcionários cuja data de admissão esteja entre 9 de maio de 1991 e 17 de junho de 1991 inclusive.

```
SQL>SELECT    first_name, start_date
 2  FROM      s_emp
 3  WHERE     start_date BETWEEN '09-MAY-91' AND '17-JUN-91';
```

FIRST_NAME	START_DAT
Midori	17-JUN-91
Alexander	26-MAY-91
Sylvie	09-MAY-91

Exemplo 8:

Exibir os funcionários que pertençam ao departamento 10 ou 45.

```
SQL>SELECT    first_name, dept_id
  2  FROM      s_emp
  3  WHERE     dept_id IN ('10','45');
```

FIRST_NAME	DEPT_ID
Mark	10
Marta	45
Bela	45
Sylvie	45

O operador **LIKE** nos permite usar caracteres máscara para comparar dados em uma condição de pesquisa, ou seja, ao invés dos dados terem que ser iguais, podemos comparar parte dos dados. Existem dois operadores de máscara:

Símbolo	Descrição
<u> </u> (sublinhado)	Qualquer caracter na posição indicada
% (percentual)	Qualquer seqüência de caracteres na posição

Exemplo 9:

Exibir todos os sobrenomes dos funcionários que começam com a letra “M”

```
SQL>SELECT    last_name
  2  FROM      s_emp
  3  WHERE     last_name LIKE 'M%';
```

LAST_NAME
Menchu
Magee
Maduro
Markarian

Exemplo 10:

Exibir todos os sobrenomes dos funcionários que não contém a letra “a”

```
SQL>SELECT    last_name
  2  FROM      s_emp
  3  WHERE     last_name NOT LIKE '%a%';
```

LAST_NAME

Quick-To-See
 Ropeburn
 Menchu
 Biri
 Giljum
 Sedeghi
 Nguyen
 Smith

Exemplo 11:

Exibir os nomes e data de admissão dos funcionários que entraram na empresa em 1991.

```
SQL>SELECT    first_name, start_date
  2  FROM      s_emp
  3  WHERE     start_date LIKE '%91';
```

FIRST_NAME	START_DAT
-----	-----
Midori	17-JUN-91
Molly	18-JAN-91
Marta	27-FEB-91
Yasmin	18-FEB-91
Andre	09-OCT-91
Akira	09-FEB-91
Vikram	06-AUG-91

Exemplo 12:

Exibição do número e nome de todos clientes que não têm um representante de vendas.

```
SQL>SELECT    id, name
  2  FROM      s_customer
  3  WHERE     sales_rep_id IS NULL;
```

ID	NAME	SALES_REP_ID
-----	-----	-----
207	Sweet Rock Sports	

Exemplo 13

Exibição dos sobrenomes e porcentagem de comissão de todos os funcionários que recebem comissão.

```
SQL>SELECT last_name, commission_pct
2 FROM s_emp
3 WHERE commission_pct IS NOT NULL;
```

LAST_NAME	COMMISSION_PCT
Magee	10
Giljum	12.5
Sedeghi	10
Nguyen	15
Dumas	17.5

Exemplo 14:

Exibição do sobrenome e salário dos funcionários que ganham acima de 1350 no departamento 31, 42 ou 50. Identifique a coluna de sobrenomes como Nome do Funcionário e a coluna de salário como Salário Mensal.

```
SQL>SELECT last_name "Nome Funcionário", salary "Salário Mensal"
2 FROM s_emp
3 WHERE salary > 1350
4 AND dept_id IN ('31','42','50');
```

Nome	Salário Mensal
Velasquez	2500
Nagayama	1400
Ropeburn	1550
Magee	1400

3. Scripts e Spools

Scripts são arquivos que podemos gerar ou criar para conter um ou mais comandos do SQL. Na verdade podemos salvar o conteúdo atual do buffer SQL e incluir em um arquivo, ou simplesmente editar um arquivo com os comandos SQL desejados.

3.1. Comandos de Arquivo

Comando	Descrição
SAVE <i>filename</i> [.ext]	Salva o conteúdo atual do buffer SQL em um arquivo. A extensão do arquivo default é .sql
@ <i>filename</i>	Executa um arquivo de comandos salvo anteriormente.
EDIT <i>filename</i> [.ext]	Chama o editor para editar o conteúdo de um arquivo salvo ou criar um novo.
SPOOL <i>filename</i> [.lst] OFF	Armazena os resultados da consulta em um arquivo, e OFF fecha o arquivo de spool.

3.2. Como criar um Script

3.2.1. 1ª Forma

- Digite no prompt do SQL/Plus o comando SQL desejado para que seja armazenado no buffer.
Ex: SQL> **select** first_name **from** s_emp;
- Salve o conteúdo do buffer em um arquivo.
Ex: SQL>**save** teste01
- Para verificar que o script foi criado, edite o arquivo teste01.
Ex: SQL>**edit** teste01
- Caso deseje montar o comando novamente, basta executar o script criado:
Ex: SQL> **@** teste01

3.2.2. 2ª Forma

Você pode também, chamar o editor, inserir os comandos desejados no arquivo e posteriormente executá-los.

- Chame o editor com o nome do arquivo(script) a ser criado.
Ex: SQL> **edit** teste02
- Insira os comandos SQL finalizados por ponto-e-vírgula.
Ex: **select** salary **from** s_emp;
select first_name **from** s_emp;
- Salve e execute o script criado.
Ex: SQL> **@**teste02

Spool são arquivos criados para armazenarem os resultados da consulta em um arquivo.

3.3. Para criar um Spool:

1. SQL>>**spool** *nome*
2. SQL>>**select** last_name **from** s_emp;
3. SQL>>**spool off**
4. SQL>>**edit** nome.lst

3.4. Executar Script com saída em Spool:

1. SQL>>**edit** *arquivo*
2. SQL>>**spool** *arquivo*
3. SQL>>**@***arquivo*
4. SQL>>**spool off**

4. JOIN

Quando forem necessários dados de mais de uma tabela do banco de dados, utiliza-se uma condição de **join**. As linhas em uma tabela podem ser unidas a linhas em outra tabela, de acordo com valores em comum existentes nas colunas correspondentes, ou seja, colunas de chave primária e de chave estrangeira.

4.1. Métodos de Join:

- Equi-joins
- Não-equi-joins

4.2. Métodos Adicionais

- Outer joins
- Self joins
- Operadores Set

4.3. Consulta com JOIN Simples

Podemos exibir dados a partir de uma ou mais tabelas **relacionadas**, escrevendo uma condição de join simples na cláusula WHERE.

Sintaxe

```
SELECT tabela.coluna, tabela.coluna...  
FROM tabela1,tabela2  
WHERE tabela1.coluna1 = tabela2.coluna2
```

Onde: *tabela.coluna* indica a tabela e coluna a partir das quais os dados serão recuperados
tabela1.coluna1 = *tabela2.coluna2* é a condição que une(relaciona) as tabelas

Ao escrever um comando **select** que une as tabelas, colocamos o nome da tabela antes do nome da coluna, para maior clareza e melhor acesso ao banco de dados.

Se o nome da coluna aparecer em mais de uma tabela, deverá ter como prefixo o nome da tabela.

Para unir tabelas, é necessário um número mínimo de condições de **join**, resumidas como o número de tabelas menos um. Portanto, para unir quatro tabelas, seria necessário um mínimo de três joins. Essa tabela pode não ser aplicada, caso a tabela tenha uma chave primária concatenada, pois, nesse caso, é necessária mais de uma coluna para identificar exclusivamente cada linha.

4.4. Pesquisa de Join Simples

4.4.1. Equijoin

Para determinar o nome do departamento de um funcionário, comparamos o valor na coluna DEPT_ID na tabela S_EMP com os valores de ID na tabela S_DEPT. A relação entre as tabelas S_EMP e S_DEPT é um *equijoin*, ou seja, **os valores na coluna DEPT_ID em ambas as tabelas deverão ser iguais**. Geralmente essas colunas são primárias e complementos de chave estrangeira.

Exemplo 1:

Exibir o nome do empregado, número e nome do departamento.

```
SQL>SELECT s_emp.last_name, s_emp.dept_id, s_dept.name
FROM s_emp, s_dept
WHERE s_emp.dept_id = s_dept.id;
```

LAST_NAME	DEPT_ID	NAME
Velasquez	50	Administration
Ngao	41	Operations
Nagayama	31	Sales
Quick-To-See	10	Finance
Ropeburn	50	Administration
Urguhart	41	Operations
Menchu	42	Operations
Biri	43	Operations
Catchpole	44	Operations
Havel	45	Operations
Magee	31	Sales
Giljum	32	Sales
Sedeghi	33	Sales

Cada empregado tem o nome do seu respectivo departamento exibido. As linhas da tabela S_EMP são combinadas com as linhas da tabela S_DEPT, e as linhas somente são retornadas, se os valores de S_EMP.DEPT_ID e S_DEPT.ID forem iguais.

Exemplo 2

Exibir do número do departamento, número e nome da região para todos os departamentos.

```
SQL> SELECT s_dept.id "Departamento ID", s_region.id "Região ID",
s_region.name "Nome da Região"
FROM s_dept, s_region
WHERE s_dept.region_id = s_region.id;
```

Departamento ID	Região ID	Nome Região
10	1	North America
31	1	North America
32	2	South America
33	3	Africa / Middle East
34	4	Asia
35	5	Europe
41	1	North America
42	2	South America
43	3	Africa / Middle East
44	4	Asia
45	5	Europe
50	1	North America

4.5. Condições Adicionais de Pesquisa Usando o Operador AND

Exemplo1:

```
SQL>SELECT s_emp.last_name, s_dept.name, s_region.name
FROM s_emp, s_dept, s_region
WHERE s_emp.dept_id=s_dept.id
AND s_dept.region_id=s_region.id;
```

LAST_NAME	NAME	NAME
Velasquez	Administration	North America
Ngao	Operations	North America
Nagayama	Sales	North America
Quick-To-See	Finance	North America
Ropeburn	Administration	North America
Urguhart	Operations	North America
Menchu	Operations	South America
Biri	Operations	Africa / Middle East
Catchpole	Operations	Asia
Havel	Operations	Europe
Magee	Sales	North America
Giljum	Sales	South America

Exemplo2:

```
SQL> SELECT s_emp.last_name, s_region.name, s_emp.commission_pct
FROM s_emp, s_dept, s_region
WHERE s_emp.dept_id=s_dept.id
AND s_dept.region_id=s_region.id;
AND s_emp.commission_pct >0;
```

LAST_NAME	NAME	COMMISSION_PCT
Magee	North America	10
Giljum	South America	12.5
Sedeghi	Africa / Middle East	10
Nguyen	Asia	15
Dumas	Europe	17.5

4.6. Alias de Tabela

Como ocorre nas colunas, os apelidos de tabela funcionam como um meio de dar outro nome a tabela, visando o comando **SELECT**. Uma vez que você esteja usando o alias de tabela, deverá continuar a qualificar todas as referências de coluna de acordo com o alias da tabela.

Exemplo 1:

```
SQL> SELECT c.name "Nome do Cliente", c.region_id "Região ID", r.name "Região"
      FROM   s_customer c, s_region r
      WHERE  c.region_id = r.id;
```

4.7. Retornando Registros sem Correspondência Direta

Se uma linha não satisfizer uma condição de join, a linha não aparecerá no resultado da pesquisa.

As linhas que faltam podem ser retornadas se for usado um operador **outer join** na condição de join. O operador é um sinal de mais colocado entre parênteses(+) e é colocado ao "lado" do join que apresente informações insuficientes. O operador tem o efeito de criar uma ou mais linhas NULL, para as quais podem ser unidas uma ou mais linhas de uma tabela que não apresente deficiências.

Sintaxe:

```
SELECT tabela.coluna, tabela.coluna
FROM   tabela1, tabela2
WHERE  tabela1.coluna = tabela2.coluna(+);
```

```
SELECT tabela.coluna, tabela.coluna
FROM   tabela1, tabela2
WHERE  tabela1.coluna(+) = tabela2.coluna;
```

Onde:

- `tabela1.coluna =` é a condição que une (ou relaciona) as tabelas juntas.

- tabela2.coluna(+) = é o símbolo do outer join; pode ser colocado em qualquer lado da condição da cláusula **WHERE**, mas não em ambos os lados. O símbolo do outer join deve ser seguido pelo nome da tabela sem linhas correspondentes.

Exemplo 1:

Exibição do nome do representante de vendas, do número de funcionário e do nome do cliente para todos os clientes. Inclusão também do nome do cliente, mesmo se não houver um representante de vendas designado a ele.

```
SQL> SELECT e.last_name, e.id, c.name
      2 FROM s_emp e, s_customer c
      3 WHERE e.id (+) = c.sales_rep_id
      4 ORDER by e.id;
```

LAST_NAME	ID	NAME
-----	-----	-----
Magee	11	Womansport
Magee	11	Beisbol Si!
Magee	11	Ojibway Retail
Magee	11	Big John's Sports Emporium
Giljum	12	Unisports
Giljum	12	Futbol Sonora
Sedeghi	13	Hamada Sport
Nguyen	14	Simms Athletics
Nguyen	14	Delhi Sports
Dumas	15	Kam's Sporting Goods
Dumas	15	Sportique
Dumas	15	Muench Sports
Dumas	15	Sporta Russia
Dumas	15	Kuhn's Sports

4.8. Unindo Uma Tabela A Si Mesma

Você pode unir uma tabela a ela mesma usando os apelidos (alias) de tabela para simular a existência de duas tabelas separadas. Isso permite que as linhas em uma tabela sejam unidas às linhas na mesma tabela.

4.9. Self Join

Para simular duas tabelas na cláusula FROM, o exemplo contém um alias para a mesma tabela, S_EMP. Esse é um exemplo de boa combinação de nomes.

Nesse exemplo, a cláusula WHERE contém o join que significa “onde o número do gerente de um empregado corresponde ao número de funcionário para o gerente.”

Exemplo 1:

Exibição dos nomes dos funcionários e de seus respectivos gerentes:

```
SQL> SELECT a.last_name || 'trabalha para ' || b.last_name  
2 FROM s_emp a, s_emp b  
3 WHERE a.manager_id = b.id;
```

A.LAST_NAME	TRABALHA PARA	BLAST_NAME
Ngao	trabalha para	Velasquez
Nagayama	trabalha para	Velasquez
Quick-To-See	trabalha para	Velasquez
Ropeburn	trabalha para	Velasquez
Urguhart	trabalha para	Ngao
Menchu	trabalha para	Ngao
Biri	trabalha para	Ngao
Catchpole	trabalha para	Ngao
Havel	trabalha para	Ngao
Magee	trabalha para	Nagayama
Giljum	trabalha para	Nagayama

5. SUBQUERIES

Podemos ter **queries aninhadas**. Quando definimos o formato comando SELECT, vimos que a cláusula WHERE é seguida de um bloco de condição. Este bloco de condição pode ser uma condição envolvendo inclusive uma outra query(uma subquery). Uma subquery é uma declaração SELECT embutida na cláusula WHERE de uma outra query. As subqueries são executadas primeiro, após a obtenção do seu resultado este é utilizado na execução da query mais externa.

5.1. Regras Gerais

- A subquery deve ser colocada entre parênteses.
- A subquery deve ser colocada depois de um operador de comparação.
- Uma cláusula ORDER BY não deve ser incluída em uma subquery.

Exemplo 1:

Exibir os funcionários cujo código de departamento seja igual ao código de departamento da funcionária 'Roberta'.

```
SQL> SELECT first_name, dept_id
      FROM s_emp
      WHERE dept_id =
          (SELECT dept_id
           FROM s_emp
           WHERE first_name = 'Roberta');
```

FIRST_NAME	DEPT_ID
Roberta	42
Akira	42
Vikram	42

Exemplo 2:

Exibir o sobrenome dos funcionários que tenham o mesmo cargo do Smith.

```
SQL> SELECT first_name, title
      FROM s_emp
      WHERE title =
          (SELECT title
           FROM s_emp
           WHERE last_name = 'Smith');
```

LAST_NAME	TITLE
-----	-----
Maduro	Stock Clerk
Smith	Stock Clerk
Nozaki	Stock Clerk
Patel	Stock Clerk
Newman	Stock Clerk
Markarian	Stock Clerk
Chang	Stock Clerk
Patel	stock Clerk
Dancs	Stock Clerk
Schwartz	Stock Clerk

10 rows selected.

As subqueries que retornam mais de uma linha são chamadas subqueries de várias linhas. Deve-se usar um operador de várias linhas, como por exemplo IN, ao invés de um operador de única linha.

Exemplo 3:

Identificar todos os funcionários que fazem parte do departamento financeiro ou da região 2.

```
SQL>SELECT last_name, first_name, title
        FROM s_emp
        WHERE dept_id IN
              (SELECT id
               FROM s_dept
               WHERE name='Finance' OR region_id = 2);
```

LAST_NAME	FIRST_NAME	TITLE
-----	-----	-----
Quick-To-S	Mark	VP, Finance
Menchu	Roberta	Warehouse Manager
Giljum	Henry	Sales Representative
Nozaki	Akira	Stock Clerk
Patel	Vikram	Stock Clerk

6. VISÃO GERAL DE MODELAGEM DE DADOS – O PROJETO DO BD

6.1. Introdução

Este capítulo destina-se a apresentar o processo de **modelagem de dados**, conceitos de **banco de dados relacional** e **normalização**. É apresentado também o **modelo de entidade relacionamento** para um projeto de B.D.(Banco de Dados). Esses passos são necessários para a construção das tabelas do seu projeto (sistema), seja ele corporativo ou local.

Este capítulo não tem a pretensão de aprofundar-se nos tópicos acima citados, mas tão somente introduzir o aluno nos conceitos básicos .

6.2. Ciclo de Desenvolvimento do Sistema

Para criar os objetos do banco de dados com lógica e sucesso no Oracle 7 Server, você deve completar o ciclo de desenvolvimento do sistema. Cada estágio do ciclo contém atividades específicas que você realiza para obter o melhor projeto possível de banco de dados.

Desde a concepção até a produção, desenvolva um banco de dados usando o ciclo de desenvolvimento do sistema para se obter melhor performance, eficiência e eficácia no projeto. A abordagem sistemática será do tipo Top-Down, que transforma requisitos de informações de negócios em um banco de dados operacional.

6.3. Estágios de Desenvolvimento

6.3.1. Estratégia e Análise

Estude e analise os requisitos comerciais. Entreviste os usuários e os gerentes para identificar as necessidades de informação. Incorpore as instruções da empresa e dos aplicativos bem como quaisquer especificações futuras do sistema.

Elabore modelos do sistema. Transfira a narração comercial desenvolvida na fase de estratégia e análise para uma representação gráfica das necessidades de informações e regras comerciais. Confirme e aprimore o modelo com analistas e peritos.

6.3.2. Projeto

Projete o banco de dados. O diagrama entidade-relacionamento mapeia entidades para tabelas, atributos para colunas, relacionamentos para chaves estrangeiras e regras comerciais para constraints.

6.3.3. Elaboração e Documentação

Elabore o sistema protótipo. Escreva e execute os comandos para criar tabelas e objetos de suporte para o banco de dados.

Desenvolva a documentação de usuário, textos de telas de ajuda e manuais de operação para dar suporte ao uso e à operação do sistema.

6.3.4. Transição

Refinamento do protótipo. Coloque o aplicativo em produção com os testes aceitos pelo usuário, conversão de dados existentes e operações paralelas. Faça as alterações necessárias.

6.3.5. Produção

Apresente o sistema ao usuário. Opere o sistema de produção. Monitore o desempenho e faça melhorias e refinamentos ao sistema.

6.4. Projeto de Banco de Dados

Projetar um sistema de banco de dados relacional envolve a conversão de um modelo a uma representação de software funcional. As entidades (ou objetos) observadas pelo usuário são transformadas em tabelas no banco de dados. Todas as formas de projeto envolvem uma mistura de regras, avaliações a bom senso e o projeto relacional não é diferente disso.

O objetivo é projetar sistemas confiáveis e de alto desempenho usando os dados colhidos durante a análise. Os seguintes fatores-chaves descrevem em detalhes porquê você deve se preocupar em elaborar o projeto.

6.5. Desempenho

O projeto inicial de um sistema tem um impacto enorme em seu desempenho final. De modo geral, o impacto é muito maior do que qualquer tentativa de remediação.

6.6. Aplicação Integrada

Sistemas de aplicação são basicamente elaborados por equipes de desenvolvedores. Se não houver especificações nas quais eles possam se basear, cada um fará a elaboração de acordo com seu próprio estilo. Um bom projeto não só promove uma aparência e sensação coesa, como também ajuda a garantir que todos os componentes do sistema de aplicação resultante estejam integrados uns aos outros.

6.7. Integração com Outros Sistemas

Muitas vezes, é necessário que um sistema novo se integre a outros já existentes, ou mesmo a sistemas ainda por serem elaborados. Um bom projeto estende os benefícios de integração mencionados acima a sistemas corporativos e mundiais.

6.8. Documentação e Comunicação

Uma grande parte do trabalho de um projetista é o de comunicar decisões do projeto a outros participantes. No mínimo, essas decisões precisam ser documentadas.

6.9. Escalabilidade

Trabalhe com questões de desempenho durante o projeto e não durante a produção. Por exemplo, desenvolver uma aplicação em um ambiente pequeno e controlado não irá testar situações do mundo real ou um grande conjunto de dados, fatores que podem revelar falhas de projeto.

6.10. Evite a Reinvenção da Roda

Muitos dos problemas que você enfrentará já foram vividos por outros antes de você. Use soluções de projetos bem-sucedidos sempre que puder.

6.11. Modelo de Dados

Modelos são a base do projeto. Engenheiros elaboram um modelo de um carro antes de colocá-lo em produção para resolver quaisquer problemas. Deste mesmo modo, projetistas de sistemas desenvolvem para explorar idéias e melhorar a compreensão do projeto do banco de dados.

6.11.1. Propósito dos Modelos

Modelos ajudam a comunicar os conceitos às pessoas. Eles podem ser usados para os seguintes propósitos:

- Comunicar
- Categorizar
- Descrever
- Especificar
- Investigar
- Envolver
- Analisar
- Iniciar

O objetivo é o de gerar um modelo que se ajuste a uma grande gama de usos, que possa ser compreendido pelo usuário final, mas que contenha dados suficientes para que um desenvolvedor possa elaborar um sistema de banco de dados.

6.12. Modelo de Entidade-Relacionamento

Um modelo de entidade-relacionamento é composto de entidades, atributos e relacionamentos.

6.12.1. Entidades

Uma entidade representa o que existe de significativo sobre o sistema da empresa, ou uma categoria discreta ou coleta de dados relacionados. Exemplos são clientes, pedidos e funcionários.

Para representar uma entidade em um modelo, use as seguintes convenções:

- Retângulo arredondado com quaisquer dimensão
- Nome de entidade único.
- Nome da entidade em letras maiúsculas
- Nomes sinônimos opcionais em letras maiúsculas entre parênteses“()”

6.12.2. Atributos

Um atributo descreve entidades e retém informações específicas que devem ser conhecidas sobre uma entidade. Por exemplo, para entidade cliente, os atributos seriam número, nome, número de telefone e endereço do cliente.

Cada um dos atributos é obrigatório ou opcional. Este estado é chamado opcionalidade.

Para representar um atributo em um modelo, use as seguintes convenções:

- Use nomes únicos em letras minúsculas.
- Marque atributos obrigatórios ou valores que devem ser conhecidos, com asterisco ”*”.
- Marque atributos opcionais ou valores que devem ser conhecidos, com um “o”.

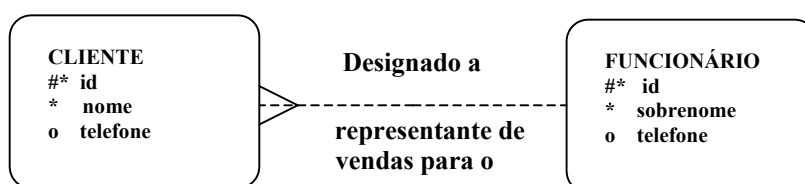
6.12.3. Identificadores Únicos

Um identificador único(UID) é qualquer combinação de atributos ou relacionamentos, ou ambos. Que servem para distinguir ocorrências de uma entidade. Cada ocorrência de entidade deve ser identificável com exclusividade.

- Marque cada atributo que faça parte do UID com um símbolo de jogo da velha (#).
- Marque UIDs secundários com um sinal de jogo da velha entre parênteses (#).

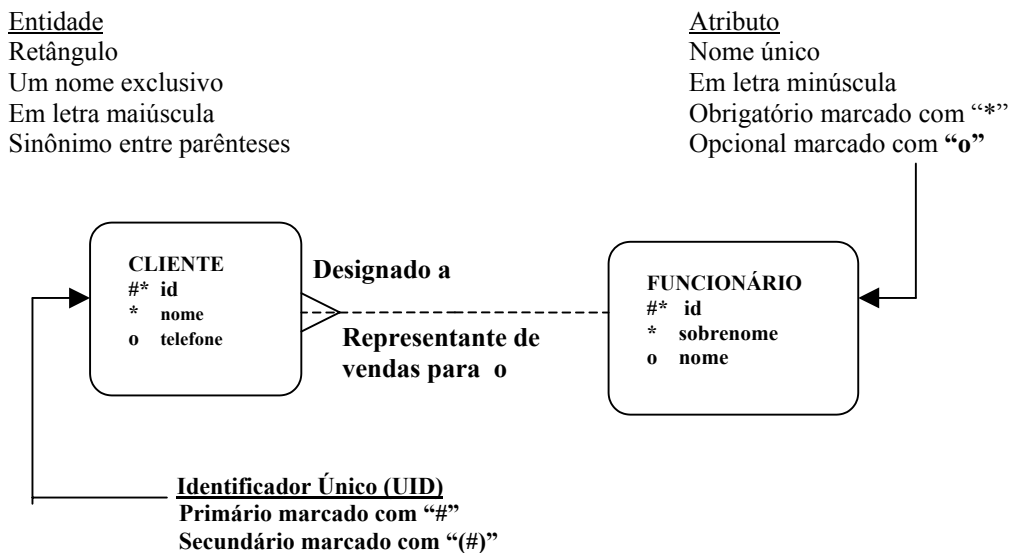
Exemplo:

- Crie um diagrama de entidade-relacionamento a partir de especificações comerciais ou de narrativas.



- Cenário
 - "... Designar um ou mais clientes a um representante de vendas..."
 - "... Alguns representantes de vendas ainda não têm clientes designados e eles..."

6.13. Convenções de Modelagem de Entidade-Relacionamento



6.14. Relacionamentos

Cada entidade deve Ter um relacionamento que represente as necessidades e regras de informação da empresa. O relacionamento é uma associação bidirecional entre duas entidades, ou entre uma entidade e ela mesma. Quando uma entidade tem um relacionamento com ela mesma é identificada como recursiva.

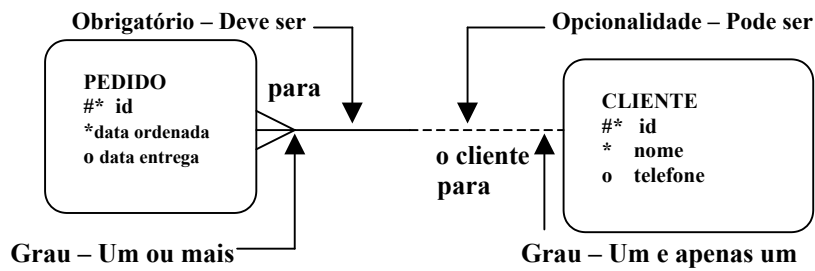
Cada direção do relacionamento contém:

- Um nome, por exemplo, *ensinado por*, *designado a*.
- Uma opcionalidade, seja *deve ser* ou *pode ser*.
- Um grau, *um e apenas um* ou *um ou mais*.

Exemplo:

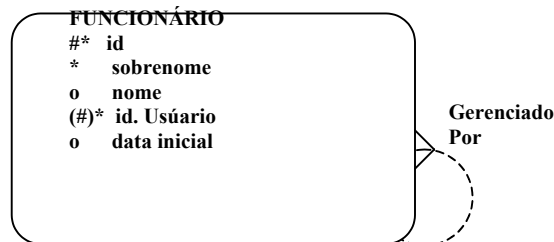
- Sintaxe
- Cada entidade de origem (pode ser/ deve ser) nome do relacionamento (um e único/ um ou mais) entidade de destino.
- Cada PEDIDO deve ser de um e apenas um CLIENTE.

- Cada CLIENTE pode estar em um ou mais PEDIDOS.



6.15. Relacionamentos Recursivos

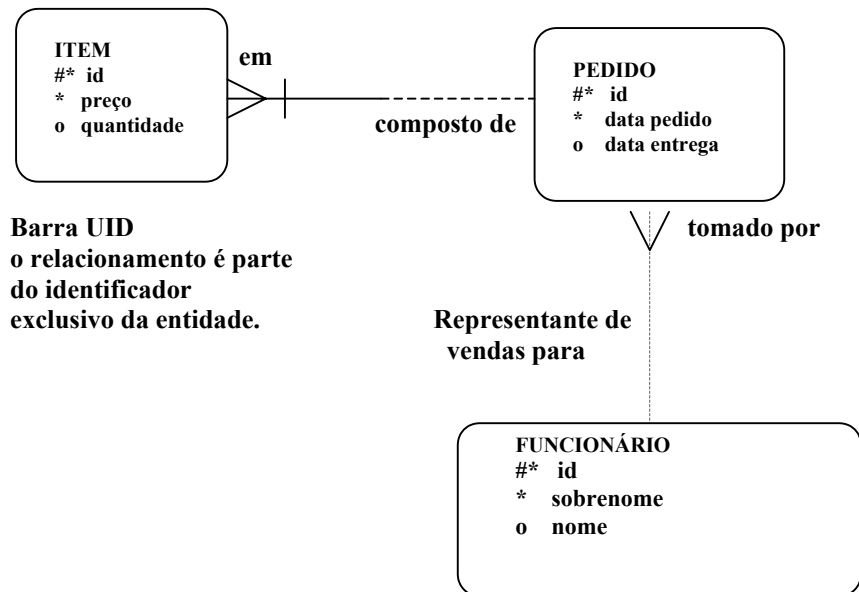
- Defina um relacionamento entre uma entidade e ela mesma como uma relacionamento recursivo.
- Represente tal relacionamento por um semi-círculo.



6.16. Convenções de Diagramação de Relacionamento

Símbolo	Descrição
Linha pontilhada	Pode ser
Linha sólida	Deve ser
Pé-de-galinha	Uma ou mais
Linha única	Um e apenas um

6.17. Subconjunto do Modelo de Entidade-Relacionamento



6.18. Tipos de Relacionamento

6.18.1. Um para um

- Grau de um e apenas um em ambas as direções.
- São raros.

Exemplo: Marido e mulher.

6.18.2. Muitos para um

- Grau de um ou mais em uma direção e um grau de um e apenas um na outra direção
- São muito comuns.

Exemplo: passageiros e avião.

6.18.3. Muitos para muitos

- Grau de um ou mais em ambas as direções.
- São solucionados com uma entidade de intersecção.

Exemplo: Funcionários e habilidades

6.19. Constraints e Chaves de Integridade

Assegure que os usuários realizam apenas operações que deixam o banco de dados em um estado consistente ao forçar constraints de integridade de dados. Todas as constraints de integridade de dados devem ser forçadas pelo servidor do banco de dados ou pelo software da aplicação. Chaves correspondem a constraints de integridade. Os três tipos de chaves são a chave primária, a única e a estrangeira.

Tipo de Constraint de Integridade	Descrição
Entidade	Nenhuma parte de uma chave primária pode ser NULL e o valor deve ser único.
Referencial	Valores de chaves estrangeiras devem corresponder a uma chave primária ou ser NULL.
Coluna	Valores na coluna devem corresponder com o tipo de dado definido.
Definida pelo usuário	Valores devem cumprir com as regras de negócios.

Exemplos de Constraints de Integridade Definidas pelo Usuário

- Um funcionário no departamento de finanças não pode Ter um título de programador.
- Uma comissão de um vendedor não pode exceder a 50% do salário-base.
- Clientes podem ter valores de taxas de crédito Excelentes, Boas ou Ruins.

6.19.1. Chaves Primárias

Cada linha em uma tabela é identificada com exclusividade por uma coluna ou por um conjunto de colunas chamada de chave primária (PK) . A chave primária é definida de modo a não permitir valores duplicados e não pode ser NULL.

Uma chave primária com múltiplas colunas é chamada de chaves primária composta ou chave primária complexa. As colunas de uma chave primária composta devem Ter uma combinação única, enquanto que as colunas individuais podem ter duplicidades. Nenhuma parte de uma chave primária pode conter um valor nulo.

6.19.2. Chaves Candidatas

Uma tabela pode ter diversas chaves candidatas. Uma chave candidata é uma coluna ou combinação de colunas que pode servir como a chave primária para a tabela.

Selecione uma chave candidata para ser a chave primária da tabela. As outras candidatas tornam-se alternativas ou chaves únicas. Elas devem ser UNIQUE e NOT NULL.

6.19.3. Chaves Estrangeiras

Uma chave estrangeira (FK) é uma coluna ou combinação de colunas em uma tabela que se refere a uma chave primária ou chave única na mesma tabela ou em outra tabela. Chaves estrangeiras são baseadas em valores de dados e são puramente lógicos, não indicadores físicos. Um valor de chave estrangeira deve corresponder com um valor de chave primária ou chave única existente, ou então ser NULL. Se uma chave estrangeira for parte de uma chave primária, ela não pode conter um valor nulo pois nenhuma parte de uma PK pode ser NULL.

7. CRIAÇÃO DE TABELAS

Antes de apresentarmos a sintaxe de criação de tabelas, veremos os tipos de dados do Oracle7.

Há muitos tipos diferentes de colunas. O Oracle7 pode tratar valores de um tipo de dados diferentemente dos valores de outros tipos de dados.

Exemplo de Tipos de Dados do Oracle7

Tipos de Dados	Descrição
<i>VARCHAR2(tamanho)</i>	Valores de caracteres de tamanhos variáveis até o tamanho máximo.
<i>CHAR(tamanho)</i>	Valores de tamanhos fixos.
<i>NUMBER</i>	Número de ponto flutuante com precisão de 38 dígitos significativos.
<i>NUMBER(p,s)</i>	Valor numérico com precisão máxima de p, numa faixa de 1 a 38 e escala máxima de s; a precisão é o número total de dígitos decimais e a escala é o número de dígitos à direita do ponto decimal.
<i>DATE</i>	Valor de data e hora
<i>LONG</i>	Valores de caracteres de tamanhos variáveis de até 2 gigabytes.
<i>RAW e LONG RAW</i>	Equivalente a VARCHAR e LONG, respectivamente, mas usado para armazenar dados orientados por byte ou binários que não serão interpretados pelo Oracle7

7.1. Constraints de Integridade de Dados

Você pode usar constraints para:

- Forçar regras no nível da tabela sempre que uma linha for inserida, atualizada ou excluída. As constraints devem ser cumpridas para que a operação tenha sucesso.
- Evitar a exclusão de uma tabela se houver dependências de outras tabelas.
- Fornecer regras para ferramentas Oracle, como o Developer/2000.

CONSTRAINT	DESCRIÇÃO
not null	Especifica que a coluna não pode ter um valor nulo.
unique	Especifica que uma coluna ou combinação de colunas cujos valores devem ser únicos para todas as linhas da tabela.
primary key	Identifica com exclusividade cada linha na tabela
foreign key	Estabelece e força um relacionamento de chave estrangeira entre a coluna e uma coluna da tabela à qual se faz referência.
check	Especifica uma condição que deve ser verdadeira.

7.2. Criando Tabelas

7.2.1. Sintaxe Abreviada

```
CREATE TABLE [schema.] table
(column datatype [DEFAULT expr] [column_constraint],
...
[table_constraint]);
```

Onde:	<i>schema</i>	é o mesmo que o nome do proprietário.
	<i>table</i>	é o nome da tabela.
	<i>DEFAULT expr</i>	especifica um valor default se um valor for omitido no comando INSERT.
	<i>column</i>	é o nome da coluna
	<i>datatype</i>	é o tipo de dado e o tamanho da coluna
	<i>column_constraint</i>	é uma constraint de integridade como parte da definição da coluna.
	<i>table_constraint</i>	é uma constraint de integridade como parte da definição da tabela.

Exemplo:

```
SQL>CREATE TABLE s_dept
(id          NUMBER(7)
 CONSTRAINT s_dept_id_pk  PRIMARY KEY,
 name       VARCHAR2(25)
 CONSTRAINT s_dept_name_nn NOT NULL,
 region_id  NUMBER(7)
 CONSTRAINT s_dept_region_id_fk REFERENCES
s_region(id),
 CONSTRAINT s_dept_name_region_id_uk UNIQUE
(name, region_id));
```

7.3. Definindo as Constraints

- A constraint de coluna S_DEPT_ID_PK identifica a coluna ID como a chave primária da tabela S_DEPT. Essa constraint assegura que dois departamentos na mesma tabela não tenham o mesmo número de departamento e que nenhum número de departamento seja NULL.
- A constraint de coluna S_DEPT_NAME_NN assegura que cada número de departamento na tabela tenha um nome.
- A constraint de coluna S_DEPT_REGION_ID_FK assegura que qualquer número de região digitado na tabela S_DEPT tenha um valor correspondente na tabela S_REGION. Antes de definir essa constraint, a tabela S_REGION,

incluindo uma constraint PRIMARY KEY ou UNIQUE na coluna ID, deve ser criada.

- A constraint de tabela S_DEPT_NAME_REGION_ID_UK identifica a coluna NAME e a coluna REGION_ID como uma chave composta única para assegurar que a mesma combinação de nome de departamento e número de região não apareçam na tabela mais de uma vez.

8. MANIPULANDO DADOS

A linguagem de manipulação de dados (data manipulation language, DML) é parte central do SQL. Quando você deseja incluir, atualizar ou excluir dados de um banco de dados, você executa um comando DML. Uma coleção de comandos DML que ainda não foram tornados permanentes é chamada de uma **transação** ou uma unidade de trabalho lógica.

8.1. Adicionando uma Nova Linha em uma Tabela

Você pode adicionar novas linhas em uma tabela usando o comando INSERT.

Sintaxe

```
INSERT INTO table [{column [, column...]}]
VALUES      (value [, value...]);
```

Onde: *table* é o nome da tabela.
 column é o nome da coluna na tabela a ser preenchida.
 value é o valor correspondente à coluna.

Nota: Este comando com a cláusula VALUES inclui apenas uma linha por vez na tabela.

Como você pode adicionar uma nova linha com valores para cada coluna, a lista de colunas não é obrigatória na cláusula INSERT. Contudo, os valores devem ser listados de acordo com a ordem default das colunas na tabela.

Exemplo 1:

```
SQL> INSERT INTO            s_dept
      2 VALUES             (11, 'Finance', 2);
```

Exemplo 2:

Inserção de um novo departamento omitindo o número da região. Como o número da região não é listado na cláusula INSERT, um valor nulo é informado, implicitamente, para este número nesta linha.

```
SQL> INSERT INTO            s_dept (id, name)
      2 VALUES             (12, 'MIS');
```

Exemplo 3:

Adição de um valor nulo em uma linha, explicitamente, usando a palavra-chave NULL para valor.

```
SQL> INSERT INTO      s_dept
      2 VALUES      (13, 'Administration', NULL);
```

8.2. Atualizando Linhas

Você pode modificar as linhas existentes usando o comando UPDATE.

Sintaxe:

UPDATE	table
SET	column = value [, column = value...]
[WHERE	condition] ;

onde: table é o nome da tabela.
column é o nome da coluna na tabela a ser preenchida.
Value é o valor ou subquery correspondente para a coluna.
Condition identifica as linhas a serem atualizadas e é composto de nomes de colunas, expressões, constantes, subqueries e operadores de comparação.

Confirme a operação de atualização consultando a tabela para exibir as linhas atualizadas.

Exemplo 4:

Transferência do funcionário número 2 para o departamento 10. Transfira o funcionário número 1 para o departamento 32 e altere seu salário para 2550.

```
SQL> UPDATE      s_emp
      2 SET      dept_id = 10
      3 WHERE    id = 2;

SQL> UPDATE      s_emp
      2 SET      dept_id = 32, salary = 2550
      3 WHERE    id = 1;
```

Exemplo 5:

Comissão de 10% para cada funcionário da empresa. Confirma as alterações.

```
SQL> UPDATE      s_emp
      2 SET      commission_pct = 10;
```

8.3. Deletando Linhas

Você pode remover linhas existentes usando o comando DELETE.

Sintaxe

```
DELETE [FROM] table  
[WHERE condition];
```

onde: table é o nome da tabela.
 Condition identifica as linhas a serem deletadas e é composto de nomes de coluna, expressões, restrições, subqueries e operadores de comparação.

Confirme a operação de exclusão exibindo as linhas deletadas através do comando SELECT.

Exemplo 1:

Remoção de todas as informações sobre funcionários que foram admitidos depois de 01 de janeiro de 1996.

```
SQL > DELETE FROM        s_emp  
      2 WHERE             start_date >  
                          TO_DATE('01.01.1996' , 'DD.MM.YYYY');
```

Se você não incluir uma cláusula WHERE no comando DELETE, todas as linhas da tabela serão deletadas.

Exemplo 2:

Eliminação de todos os dados da tabela TEST.

```
SQL > DELETE FROM test;
```

8.4. Processamento de Transações

O Oracle 7 Server assegura a consistência de dados com base em transações. As transações dão maior flexibilidade e controle na alteração dos dados e asseguram sua consistência caso ocorra uma falha de processamento do usuário ou falha do sistema.

As transações consistem em comandos DML que fazem uma alteração consistente aos dados. Por exemplo, uma transferência de fundos entre duas contas deve incluir o débito em uma conta e o crédito na outra pelo mesmo valor. Ambas as ações devem ocorrer simultaneamente. O crédito não deve ser realizado sem o débito.

8.4.1. Tipos de Transações

Tipo	Descrição
Manipulação de dados (DML)	Consiste em um certo número de comandos DML que o Oracle 7 Server trata como um única entidade ou como um unidade lógica de trabalho.
Definição de dados (DDL)	Consiste em apenas um comando DDL.
Controle de dados (DCL)	Consiste em apenas um comando DCL.

8.4.2. Quando Inicia e Termina uma Transação?

Uma transação inicia quando o primeiro comando executável SQL é encontrado e termina quando ocorre uma das seguintes situações:

- Um comando COMMIT ou ROLLBACK é gerado.
- Um comando DDL, como CREATE ou DCL é gerado.
- Certos erros são detectados, como conflitos.
- O usuário sai do SQL *Plus.
- Há falha de hardware ou o sistema cai.

Após o término de uma transação, o próximo comando SQL executável iniciará a transação seguinte automaticamente.

8.4.3. Comandos Explícitos de Controle de Transação

Controle as transações lógicas usando os comandos COMMIT, SAVEPOINT e ROLLBACK.

Comando	Descrição
COMMIT	Termina a transação atual ao transformar em permanentes todos os dados pendentes.
SAVEPOINT <i>name</i>	Marca um savepoint dentro da transação atual.
ROLLBACK [TO SVEPOINT <i>name</i>]	Termina a transação atual ao descartar todas as alterações de dados pendentes.

8.4.4. *Processamento Implícito de Transações*

Status	Circunstância
Commit automático	O comando DDL ou o comando DCL é gerado.
Commit automático	Saída normal do SQL *Plus, sem gerar explicitamente o COMMIT ou ROLLBACK.
Rollback automático	Encerramento anormal do SQL *Plus ou falha do sistema.

8.5. Submetendo Alterações

Todas as alterações de dados feitas durante a transação são temporárias até a transação ser submetida.

8.5.1. *Estado dos Dados Antes de um COMMIT ou ROLLBACK*

- Operações de manipulação de dados afetam primariamente o buffer do banco de dados; portanto, o estado anterior dos dados pode ser recuperado.
- O usuário atual pode revisar os resultados das operações de manipulação de dados ao consultar as tabelas.
- Outros usuários *não podem* ver os resultados das operações de manipulação de dados do usuário atual. O Oracle 7 institui a consistência de leitura para assegurar que cada usuário veja os dados do modo que existiam no último commit.
- As linhas afetadas são *bloqueadas*; outros usuários não podem alterar dados dentro das linha afetadas.

Torne permanentes alterações pendentes usando o comando COMMIT. Depois de um COMMIT:

- As alterações aos dados são gravadas no banco de dados.
- O estado anterior dos dados é perdido permanentemente.
- Todos os usuários podem verificar os resultados da transação.
- Os bloqueios nas linhas afetadas são liberados; as linhas agora estão disponíveis para outros usuários realizarem novas alterações de dados.
- Todos os savepoints são apagados.

Exemplo:

Criação de um novo departamento “Education” com pelo menos um funcionário. Tornar permanentes as alterações aos dados.

```
SQL > INSERT INTO      s_dept (id, name, region_id)
      2  VALUES        (54, 'Education', 1);
```

```
SQL > UPDATE          s_emp
      2  SET            dept_id = 54
      3  WHERE          id = 2;
```

```
SQL > COMMIT;
```

8.6. Fazendo o Rollback da Alterações

Descarte todas as alterações pendentes usando o comando ROLLBACK. Depois de um ROLLBACK:

- As alterações feitas aos dados são desfeitas.
- O estado anterior dos dados é restaurado.
- Os bloqueios nas linhas afetadas são liberados; as linhas agora estão disponíveis para outros usuários realizarem novas alterações aos dados.

Exemplo:

Ao tentar remover um registro da tabela TEST, esvazie acidentalmente a tabela. Corrija o erro e então gere o comando apropriado novamente e torne a alteração de dados permanente.

```
SQL > DELETE FROM test;

SQL > ROLLBACK;

SQL > DELETE FROM test
2   WHERE      id = 100;

SQL > SELECT *
2   FROM      test
3   WHERE      id = 100;

SQL > COMMIT;
```

9. ALTERANDO TABELAS

Uma vez que você criou as tabelas, pode alterar suas estruturas usando o comando ALTER TABLE. Adicione colunas, altere o tamanho, adicione ou retire constraints e habilite ou desabilite constraints usando este comando.

Se você deseja remover uma tabela, as linhas e a estrutura de dados, use o comando DROP TABLE. Outros comandos que afetam tabelas e que são apresentados nesta lição são:

- RENAME, para alterar o nome de objeto do banco de dados.
- TRUNCATE, para remover todas as linhas de uma tabela.
- COMMENT, para adicionar um comentário sobre um objeto do banco de dados nos dicionário de dados.

Todos esses comandos são de definição de dados (DDL). Quando você gera esses comandos, ocorre um commit automático. Você não pode reexecutar comandos DDL. Portanto, seja muito cuidadoso quando o fizer.

9.1. Adicionando uma Coluna

Você pode adicionar colunas a uma tabela usando comando ALTER TABLE com a cláusula ADD.

Sintaxe:

```
ALTER TABLE    table
ADD            (column datatype [DEFAULT expr] [NOT NULL]
               [ , column datatype] . . . );
```

onde:	table	é o nome da tabela.
	column	é o nome da nova coluna.
	Datatype	é o tipo de dado e o tamanho da nova coluna.
	DEFAULT expr	especifica o valor default para uma nova coluna.
	NOT NULL	adiciona uma constraint NOT NULL à nova coluna.

9.1.1. Regras Gerais

- Você pode adicionar ou alterar colunas, mas não eliminá-las da tabela.
- Você não pode especificar onde a coluna deve aparecer. A nova coluna torna-se a última.

9.2. Alterando uma Coluna

Você pode alterar uma definição de coluna usando o comando ALTER TABLE com a cláusula MODIFY. A alteração de coluna pode incluir alterações ao tipo de dado, tamanho, valor default e à constraint de coluna NOT NULL.

Sintaxe

```
ALTER TABLE    table
MODIFY (column datatype [DEFAULT expr] [NOT NULL]
        [, column datatype] ... );
```

onde: table é o nome da tabela.
column é o nome da coluna.
Datatype é o tipo de dado e o tamanho da coluna.
DEFAULT expr especifica o valor default para a nova coluna.
NOT NULL adiciona uma constraint NOT NULL à nova coluna.

9.2.1. Regras Gerais

- Aumente a largura ou a precisão de uma coluna numérica.
- Diminua a largura de uma coluna se ela contiver apenas valores nulos ou se a tabela não contiver linhas.
- Altere o tipo de dado se a coluna contiver valores nulos.
- Converta uma coluna CHAR ao tipo de VARCHAR2 ou converta um coluna VARCHAR2 ao tipo de dado CHAR se a coluna contiver valores nulos ou se você não alterar o seu tamanho.
- Uma alteração no valor default da coluna afeta apenas inserções subsequentes na tabela.
- Adicione uma constraint NOT NULL apenas se não houver valores nulos na coluna.

9.3. Eliminando uma Tabela

O comando DROP TABLE remove a definição de uma tabela Oracle7. Quando você elimina uma tabela, o banco de dados perde todos os dados na tabela e todos os índices associados a ele. A opção CASCADE CONSTRAINTS também removerá constraints de integridade referencial dependentes.

9.3.1. Regras Gerais

- Todos os dados são excluídos da tabela.
- Todas as visões, sinônimos, procedures armazenadas, funções ou packages permanecerão, mas serão inválidos.
- Todas as transações pendentes são confirmadas.
- Apenas o criador da tabela ou um usuário com o privilégio DROP ANY TABLE pode remover uma tabela.

9.4. Renomeando e Truncando uma Tabela

Comandos DDL adicionais incluem o comando RENAME, usado para renomear uma tabela, uma visão, uma sequence ou sinônimo e o comando TRUNCATE TABLE é usado para remover todas as linhas de uma tabela e para liberar o espaço de armazenamento por ela utilizado.

Sintaxe – Comando RENAME

```
RENAME old_name TO new_name;
```

Sintaxe – Comando TRUNCATE

```
TRUNCATE TABLE table;
```

Resumo

Comando	Descrição
create table	Cria uma tabela e as constraints indicadas.
alter table	Altera estruturas e constraints da tabela.
drop table	Remove as linhas e a estrutura da tabela.
rename	Altera o nome de uma tabela, visão, sequence ou sinônimo.
truncate	Remove todas as linhas de uma tabela e libera o espaço de armazenamento.
comment	Inclui comentários em uma tabela ou visão.

10. VISÃO GERAL DE PL /SQL

PL/SQL(Procedural Language/SQL) é uma extensão do SQL, incorporando muitos dos recursos de projeto de linguagens de programação dos anos recentes. Ela permite a manipulação de dados e comandos de consulta de SQL a serem incluídos nas unidades estruturadas de bloco e unidades de código procedural, tornando o PL/SQL uma poderosa linguagem de processamento de transações.

10.1. Benefícios do PL/SQL

- Modularização do Desenvolvimento de Programas
- Declaração de Identificadores
- Programação com Estruturas de Controle da Linguagem Procedural
- Manipulação de Erros
- Portabilidade
- Integração
- Desempenho

10.1.1. Estrutura de Bloco PL/SQL

<p>DECLARE – Opcional</p> <ul style="list-style-type: none">- Variáveis, constantes, cursores, exceptions definidas pelo usuário <p>BEGIN – Obrigatório</p> <ul style="list-style-type: none">- Comando SQL- Comandos de controle PL/SQL <p>EXCEPTION – Opcional</p> <ul style="list-style-type: none">- Ações que devem ser realizadas quando ocorre erros <p>END; -Obrigatório</p>

10.2. Construções do Programa PL/SQL

10.2.1. Bloco Anônimo

Blocos não nomeados. Eles são declarados no ponto na aplicação onde devem ser executados e passados ao processador PL/SQL para execução no tempo de execução. Você pode embutir um bloco anônimo com um programa de pré-compilação e dentro do SQL*Plus ou Server Manager. Triggers nos componentes Developer/2000 consistem de tais blocos.

10.2.2. Subprogramas

Chamados de bloco PL/SQL. Você pode declará-los como procedures ou como functions. **Procedures fazem ações e functions retornam valores.**

Componentes Developer/2000 permitem que você declare procedures e functions como parte da aplicação(um form ou um relatório) e os carregue a partir de outras procedures, functions e triggers.

10.3. Tipos de Bloco

10.3.1. Anônimo

```
[DECLARE]

BEGIN
--statements

[EXCEPTION]

END;
```

10.3.2. Procedure

```
PROCEDURE name
IS

BEGIN
--statements

[EXCEPTIONS]

END;
```

10.3.3. Function

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

END;
```

11. DESENVOLVENDO UM BLOCO PL / SQL

11.1. Declarando Variáveis e Constantes de PL/SQL

Você precisa declarar todos os identificadores dentro da seção de declaração antes de fazer referência a eles dentro do bloco PL/SQL.

Sintaxe

```
Identifier [CONSTANT] datatype [NOT NULL] [:= | DEFAULT expr];
```

Onde:

identififier é o nome do identificador
 CONSTANT restringe o identificador de modo que o valor não possa alterar; constantes devem ser inicializadas.

Datatype é um tipo de dado de escala ou composto.

NOT NULL restringe a variável de modo que deva conter um valor; variáveis NOT NULL devem ser utilizadas.

Expr é qualquer expressão PL/SQL que possa ser um literal, outra variável ou uma expressão envolvendo operadores e functions.

11.2. Tipos de Dados Escalares

Um tipo de dado escalar retém um só valor e não tem componentes internos. Tipos de dados escalares podem ser classificados em quatro categorias: número, caracter, data e hora ou Boolean.

Tipo de Dado	Descrição
BINARY_INTEGER	Tipos de base para números inteiros incluídos entre -2147483647 e +2147483647.
NUMBER[(<i>precision</i> , <i>scale</i>)]	Tipos de base para números de pontos fixos e flutuantes.
CHAR(<i>maximum_length</i>)	Tipos de base para dados de caracter de tamanho fixo de até 32767 bytes. Se você não especificar um <i>maximum length</i> , o tamanho default é definido em 1.
LONG	Tipo de base para dados de caracter de tamanho variável de até 32760 bytes.
LONG RAW	Tipo de base para dados binários de até 32760 bytes.
VARCHAR2(<i>maximum_length</i>)	Tipo de base para dados de caracter de tamanho variável de até 32767 bytes.
DATE	Tipo de base para datas e horas.
BOOLEAN	Tipo de base armazena um de três valores possíveis usados para cálculos lógicos: TRUE, FALSE ou NULL.

Exemplo

Declaração de uma variável para armazenar o código de sexo (M ou F).

```
v_gender CHAR(1);
```

Declaração de variável para contar as iterações de um loop e inicializar a variável em 0.

```
v_count BINARY_INTEGER := 0;
```

Declaração de uma variável para acumular o salário total para um departamento e inicializar a variável a 0.

```
v_total_sal NUMBER(9,2) := 0;
```

Declaração de uma variável para armazenar a data de entrega de um pedido e inicializar a variável semana a partir de hoje.

```
v_order_date DATE := SYSDATE +7;
```

Declaração de uma constante para taxa de imposto, imutável em todo bloco PL/SQL.

```
v_tax_rate CONSTANT NUMBER(3,2) := 8.25;
```

Declaração de um identificador para indicar se um dado é válido ou inválido e inicializar a variável em TRUE.

```
v_valid BOOLEAN NOT NULL := TRUE;
```

11.3. Atribuindo Valores a Variáveis

Para atribuir um valor a uma variável, você escreve um comando de designação do PL/SQL. Você deve nomear a variável explicitamente para receber o novo valor à esquerda do operador de designação(:=).

Exemplos:

Configuração do identificador de salário máximo V_MAX_SAL ao valor do identificador de salário atual V_SAL.

```
v_max_sal := v_sal;
```

Armazenamento do nome “Maduro” no identificador de índice de 3 em PL/SQL TABLE de sobrenomes.

```
last_name_table(3) := ‘Maduro’;
```

Armazenamento de informações básicas para um funcionário novo em um PL/SQL RECORD.

```
emp_record.last_name := 'Maduro';  
emp_record.first_name := 'Elena';  
emp_record.gender := 'F';
```

Incrementação do índice para um loop.

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

Configuração do valor de um identificador Booleano dependendo se dois números forem iguais.

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

Validação de um número de funcionário se ele contiver um valor.

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

11.4. Functions

A maioria das functions disponíveis em SQL também são válidas nas expressões PL/SQL:

- Functions numéricas de linha
- Functions da caracter de linha
- Functions de conversão de tipo de dado
- Functions de data
- Functions diversas

Exemplos:

Conversão do nome para maiúsculas.

```
v_last_name := UPPER(v_last_name);
```

Cálculo da soma de todos os números armazenados na tabela NUMBER_TABLE PL/SQL.

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

11.5. Conversão de Tipo de Dado

Dentro de uma expressão, você deve certificar-se de que os tipos de dados são os mesmos. Se ocorrerem tipos de dados mistos na mesma expressão, você deve usar a function de conversão apropriada a partir da lista abaixo para converter os dados.

Sintaxe:

```
TO_CHAR (value, fm)
TO_DATE (value, fm)
TO_NUMBER (value, fm)
```

Onde : *value* é uma string de caracteres, números ou data.
fm é o modelo de formato usado para converter *value*.

Exemplo:

Armazenamento de um valor que seja composto do nome do usuário e a data de hoje. *Este código causa um erro de sintaxe.*

```
v_comment := USER || ':' || TO_CHAR(SYSDATE);
```

Para corrigir o erro, conversão de SYSDATE a uma string de caracteres com a function de conversão TO_CHAR.

```
v_comment := USER || ':' || TO_CHAR(SYSDATE);
```

11.6. Fazendo Referências a Variáveis Não-PL/SQL

Você pode fazer referências a variáveis declaradas no ambiente host ou nas chamadas em comandos PL/SQL, a não ser que o comando esteja dentro de uma procedure, function ou package. Isto inclui as variáveis de linguagem de host declaradas nos programas de pré-compilação, campos de telas em uma aplicação Developer/2000 Forms e variáveis de vinculação SQL*Plus.

Para fazer referência a *variáveis host*, você deve inserir prefixos nas referências usando dois pontos (:) para distingui-los das variáveis PL/SQL declaradas.

Exemplo:

Armazenamento do salário anual em uma variável global SQL*Plus.

```
:g_annual_salary := v_salary * 12;
```

12. INTERAGINDO COM ORACLE

Quando precisa extrair informações ou aplicar alterações ao banco de dados, você deve usar SQL. PL/SQL tem suporte para linguagem de manipulação de dados completa e para comandos de controle de transação dentro de SQL. Você pode usar os comandos SELECT para incluir variáveis com valores consultados a partir de uma linha em uma tabela. Seus comandos DML podem processar várias linhas.

12.1. Recuperando Dados Usando PL/SQL

Use o comando SELECT para recuperar dados do banco de dados. O comando SELECT contém uma cláusula obrigatória adicional: a cláusula INTO. Na cláusula INTO, liste as variáveis de saída para o recebimento de dados. O comando SELECT deve retornar exatamente uma linha ou ocorrerá erro.

12.2. Sintaxe Resumida

SELECT	<i>select_list</i>
INTO	<i>variable_name</i> <i>record_name</i>
FROM	<i>table</i>
WHERE	<i>condition</i> :

Onde:

<i>select_list</i>	é uma lista de pelo menos uma coluna e pode incluir expressões SQL, funções de linha ou funções de grupo.
<i>variable_name</i>	é a variável escalar para reter o valor recuperado
<i>record_name</i>	é o RECORD PL/SQL para reter valores recuperados.
<i>table</i>	especifica o nome da tabela do banco de dados
<i>condition</i>	é composto de nomes de colunas, expressões, constraints e operadores de comparação, a incluir variáveis e constantes de PL/SQL.

12.3. Regras Gerais

- Encerre cada comando PL/SQL com um ponto-e-vírgula.
- Atribua valores às tabelas PL/SQL em um loop ao declarar um cursor explícito.
- A cláusula INTO é obrigatória para o comando SELECT quando ela é embutida dentro de PL/SQL.
- A cláusula WHERE é opcional e pode ser usada para especificar variáveis de entrada, constraints, 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 posicionalmente e que seus tipos de dados sejam compatíveis.
- Encerre o bloco PL/SQL com o comando END. Você pode incluir o nome do subprograma após a palavra-chave END para clareza.

Exemplo 1:

```
PROCEDURE ship_date
  (v_ord_id IN NUMBER)

IS
  v_date_ordered s_ord.date_ordered%TYPE
  v_date_shipped s_ord.date_shipped%TYPE

BEGIN
  SELECT date_ordered, date_shipped
  INTO    v_date_ordered, v_date_shipped
  FROM    s_ord
  WHERE  id = v_ord_id;
  ...
END ship_date;
```

Exemplo 2

```
PROCEDURE all_dept
  (v_dept_id IN NUMBER)

IS
  dept_record s_dept%ROWTYPE;

BEGIN
  SELECT *
  INTO  dept_record
  FROM  s_dept
  WHERE id = v_dept_id;
  ...
END all_dept;
```

12.4. Exception TOO_MANY_ROWS

Quando mais de um registro for identificado com um comando SELECT, o Oracle7 gera o número de erro -1422, ao qual também, se refere como TOO_MANY_ROWS, que é o nome de exception predefinido.

12.5. Exception NO_DATA_FOUND

Quando nenhuma linha for identificada com um comando SELECT, a exception NO_DATA_FOUND é gerada, com número de erro +1403 do Oracle7.

12.6. Manipulando Dados Usando PL/SQL

- Manipule dados em um banco de dados usando os comandos DML.
- O comando INSERT adiciona novas linhas de dados na tabela.
- O comando UPDATE altera linhas existentes na tabela.
- O comando DELETE remove linhas indesejadas da tabela.

12.6.1. Inserindo Dados

Quando estiver adicionando linhas à tabela, você pode eliminar argumentos IN desnecessários.

- Use functions SQL, como USER e SYSDATE.
- Derive valores no bloco PL/SQL
- Adicione valores default de coluna.

12.6.2. Atualizando e Deletando Dados

Pode haver ambiguidade na cláusula SET do comando UPDATE pois embora o identificador à esquerda do operador de atribuição seja sempre uma coluna de banco de dados, o identificador à direita pode ser uma coluna de banco de dados ou uma variável PL/SQL.

A cláusula WHERE é usada para determinar quais linhas são afetadas. Se não for alterada nenhuma linha, não ocorrerá erro, ao contrário do comando SELECT em PL/SQL.

Exemplo 1(Atualizando Dados)

```
PROCEDURE
  (v_ord_id  s_ord.id%TYPE,
   v_ship_date s_ord.date_shipped%TYPE)

IS
BEGIN
  UPDATE  s_ord
  SET    date_shipped = v_ship_date
  WHERE  id = v_ord_id;
END new_ship_date;
```

Exemplo 2 (Excluindo Dados)

```
PROCEDURE del_order
  (v_ord_id s_ord.id%TYPE)
IS
BEGIN
  DELETE FROM s_ord
  WHERE id = v_ord_id;
END del_order;
```

13. CONTROLANDO O FLUXO EM BLOCOS PL / SQL

Você pode alterar o fluxo lógico de comandos dentro do bloco PL/SQL com diversas **estruturas de controle**.

Construções condicionais com o comando IF

Construções de Looping(Básico, FOR, WHILE, EXIT)

13.1. Comando IF

Sintaxe

```
IF condition THEN
  Statments;
[ELSIF condition THEN
  statments;]
[ELSE
  statements;]
ENDIF;
```

Exemplo:

Para um dado valor inserido, retorno de um valor calculado. Se o valor inserido for maior do que 100, então o valor calculado será duas vezes o valor inserido. Se o valor inserido for entre 50 e 100, o valor calculado será 50% do valor inicial. Se o valor inserido for inferior a 50, o valor calculado será 10% do valor inicial.

```
FUNCTION calc_val
  (v_start IN NUMBER)
RETURN NUMBER
IS
BEGIN
  IF v_start > 100 THEN
    RETURN (2 * v_start);
  ELSIF v_start >= 50 THEN
    RETURN (.5 * v_start);
  ELSE
    RETURN (.1 * v_start);

  END IF;
END calc_val;
```

13.2. Comandos Loop

13.2.1. Loop Básico

O loop mais simples consiste do corpo de comandos a ser repetido estar inserido entre os delimitadores LOOP e END LOOP. Cada vez que o fluxo da execução atingir o comando END LOOP, o controle retorna ao comando LOOP correspondente acima dele. Para evitar um loop infinito, adicione um comando EXIT.

Sintaxe

```
LOOP
  Statement 1 ;
  Statement 2;

  EXIT [WHEN condition];
END LOOP;
```

Onde:

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

Exemplo:

Inserção dos dez primeiros novos itens de linha para o pedido número 101.

```
...
v_ord_id s_item.ord_id%TYPE := 101;
v_counter NUMBER (2) := 1;
BEGIN
...
  LOOP
    INSERT INTO s_item (ord_id, item_id)
      VALUES (v_ord_id, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
```

13.3. Loop FOR**Sintaxe**

```
FOR index IN [REVERSE] lower_bound..upper_bound LOOP
  Statement 1;
  Statement 2;

END LOOP;
```

Onde: *index* é um número inteiro declarado implicitamente cujo valor aumenta automaticamente ou diminui em 1 em cada interação do loop até ser alcançado o limite superior.

REVERSE faz com que o índice diminua com cada interação a partir do limite superior até o inferior

lower_bound especifica o limite inferior para a faixa de valores de índice.

upper_bound especifica o limite superior para cada faixa de valores de índice.

Exemplo:

Impressão do número de vezes que o loop é executado e o último valor para o índice, baseado nos limites superior e inferior fornecidos.

```
PROCEDURE iterate
  (v_lower  NUMBER,
   v_upper  NUMBER)

IS
  v_counter NUMBER(10) := 0;
  v_output  NUMBER(10);
BEGIN
  FOR i IN v_lower..v_upper LOOP
    v_counter := v_counter + 1;
    v_output  := i;
  END LOOP;
  TEXT_IO.PUT_LINE('Útimo valor é ' || TO_CHAR(v_output)
    || 'Totla de loops = ' || TO_CHAR(v_counter));
END iterate;
```

13.4. Loop WHILE

Você pode usar o loop WHILE para repetir uma sequência de comandos até a condição controladora não ser mais TRUE. A condição é avaliada no início de cada iteração. O loop é encerrado quando a condição for FALSE. Se a condição for FALSE no início do loop, então não são realizadas mais iterações.

Sintaxe

```
WHILE condition LOOP
  Statement 1;
  Statement 2;

END LOOP;
```

Exemplo

Insira os dez primeiros itens de linha para o pedido número 101.

```
v_ord_id s_item.ord_id%TYPE := 101;
v_counter NUMBER(2) := 1;
BEGIN
...
WHILE v_counter <= 10 LOOP
  INSERT INTO s_item (ord_id, item_id)
  VALUES (v_ord, v_counter);
  v_counter := v_counter + 1;
END LOOP;
```