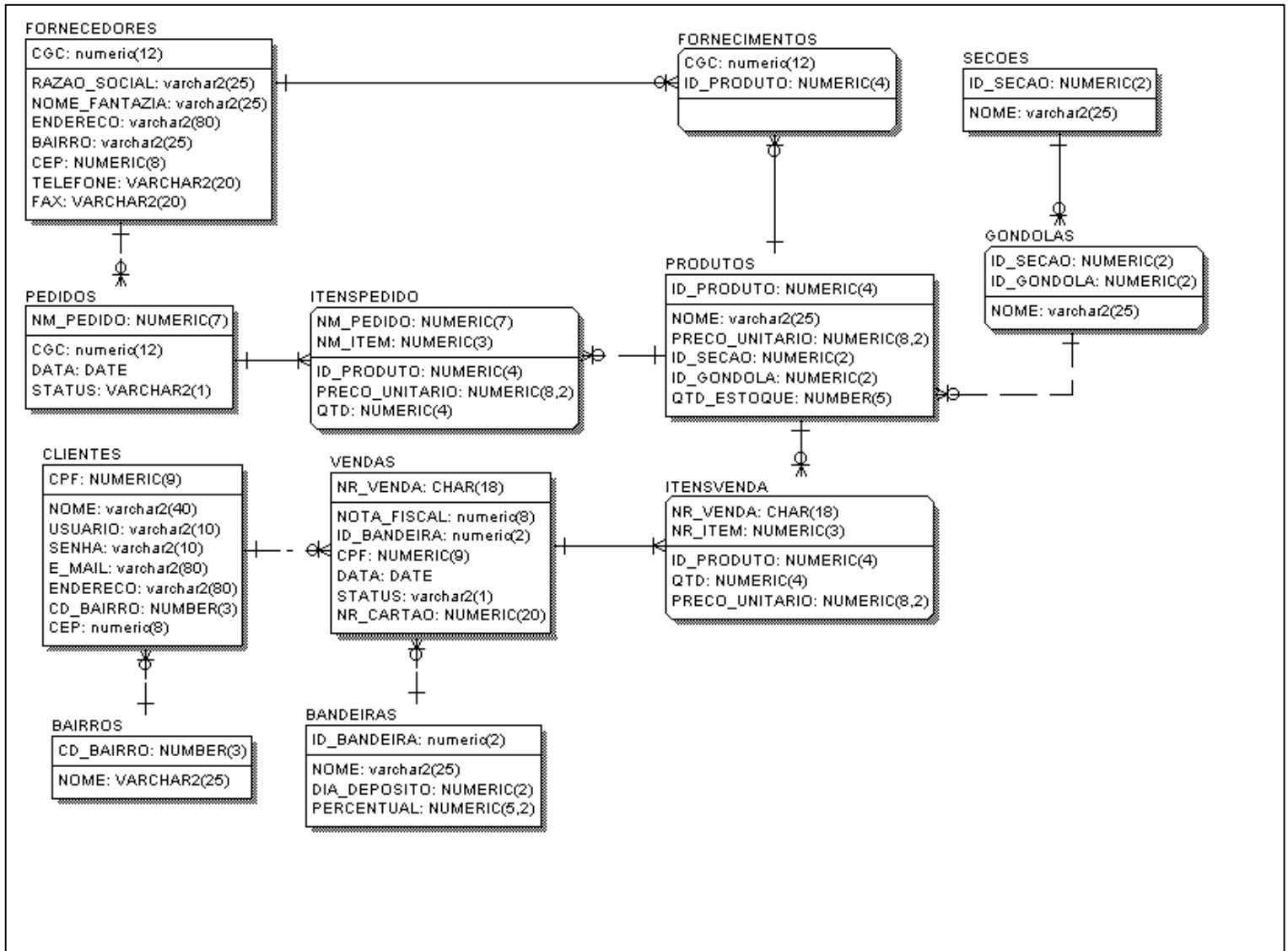


3. CGC e Razão Social de todos os fornecedores sendo que a Razão Social deve aparecer apenas com a primeira letra de cada palavra em maiúsculo. Para cada fornecedor, o nr_pedido, data no formato 'DD/MM/YYYY' e o status ('PENDENTE', 'CANCELADO', 'FECHADO') para os seguintes valores retornados: 'P', 'C', 'F' respectivamente. Ordenado por Razão Social e data do pedido.
4. Resolva a questão 3 sendo que os fornecedores que nunca receberam nenhum pedido, deve aparecer.
5. CGC e Razão Social de todos os fornecedores, para cada fornecedor o nr_pedido, data e status, para cada pedido, nome do produto, quantidade e preço pago por cada produto. Ordenado por Razão Social, data do pedido, e nome do produto.
6. CGC e Razão Social de todos os fornecedores, para cada fornecedor o nr_pedido, data ,quantidade de itens e valor total de cada pedido. Ordenado por Razão Social, data do pedido.
7. CGC e Razão Social de todos os fornecedores, para cada fornecedor o nr_pedido, data ,quantidade de itens e valor total de cada pedido. Sendo que os fornecedores que nunca forneceram nada, devem constar como número de itens e valor total iguais a zero.
8. CGC e Razão Social de todos os fornecedores , para cada fornecedor, a quantidade total de pedidos, e o valor total já pedido. Caso o valor total seja maior que 5000, ele deve ser classificado como 'BOM FORNECEDOR', caso contrário 'FORNECEDOR FRACO'. Ordenado por Razão Social e CGC.
9. Resolva a questão 8 só que aparecendo os fornecedores que nunca forneceram nada com os totais iguais a 0.
10. CGC e Razão Social de todos os fornecedores , para cada fornecedor, a quantidade total de pedidos, e o valor total já pedido, apenas para fornecedores com total já pedido maior que 5000.
11. CGC e Razão Social de todos os fornecedores, para cada fornecedor os pedidos com status 'C' ou 'P'. Ordenado por Razão Social.(Utilizar a função 'IN').
12. CGC e Razão Social de todos os fornecedores, que nunca receberam nenhum pedido. Ordenado por Razão Social.
13. Faça uma query para verifica se existe alguma gôndola em produtos, que não existe na tabela gôndolas.
14. CPF e nome de todos os clientes que nunca compraram nada. Ordenado por nome.
15. Para cada produto, a quantidade total já comprada menos a quantidade total já vendida. Ordenado pelo nome do produto. Lembrando que se um produto nunca foi vendido, ele deve constar como saldo igual ao que foi comprado.

17 Exercícios

Os exercícios deste curso serão baseados no modelo abaixo.



17.1 SQL

Codifique em SQL a solução para os seguintes problemas.

1. CGC e Razão Social de todos os fornecedores ordenados por Razão Social.
2. CGC e Razão Social de todos os fornecedores sendo que a Razão Social deve aparecer apenas com a primeira letra de cada palavra em maiúsculo. Ordenado por Razão Social.

BEFORE UPDATE comando
BEFORE DELETE linha
BEFORE DELETE comando
BEFORE INSERT linha
BEFORE INSERT comando
AFTER UPDATE linha
AFTER UPDATE comando
AFTER DELETE linha
AFTER DELETE comando
AFTER INSERT linha
AFTER INSERT comando

Não se permite no corpo de uma trigger o uso de comandos COMMIT e ROLLBACK. Também não podem ser alteradas chaves primárias, únicas ou estrangeiras.

O comando DROP TRIGGER serve para eliminá-la.

Podem ainda se habilitadas e desabilitadas:

ALTER TRIGGER nome_da_trigger ENABLE
ALTER TRIGGER nome_da_trigger DISABLE

ALTER TABLE nome_da_tabela ENABLE ALL_TRIGGERS (habilita todas de uma tabela)
ALTER TABLE nome_da_tabela DISABLE ALL_TRIGGERS (desabilita todas de uma tabela).

As informações sobre triggers são encontradas na tabela USER_TRIGGERS.

A lista de erros de uma package é obtida através do comando:

```
SHOW ERRORS PACKAGE Nome_da_Package  
SHOW ERRORS PACKAGE BODY Nome_da_Package
```

15.3 Execução de Estruturas Públicas de uma Package:

Subprogramas podem ser executados em vários ambientes:

```
EXECUTE Nome_da_Package.Nome_procedures(Lista_de_Parâmetros)  
SELECT Nome_da_Package.Nome_Função(Lista_de_Parâmetros) FROM DUAL.
```

Recompilando Packages:

```
ALTER PACKAGE nome_da_Package COMPILE PACKAGE --compila a especificação e o corpo  
ALTER PACKAGE nome_da_Package COMPILE PACKAGE ESPECIFICATION --a especificação  
ALTER PACKAGE nome_da_Package COMPILE PACKAGE BODY --compila o body
```

16 Triggers

São disparados implicitamente quando ocorrem eventos (INSERT, UPDATE, DELETE) em uma tabela. O Trigger deve estar obrigatoriamente associado a uma tabela.

Sintaxe:

```
CREATE OR REPLACE TRIGGER nome_da_trigger  
{BEFORE / AFTER}  
DELETE OR INSERT OR UPDATE OF (nome_coluna1, nome_coluna2, ....) ON  
nome_tabela  
REFERENCING [OLD as antigo] [NEW as novo]  
FOR EACH ROW  
WHEN condição  
DECLARE  
    Variáveis, constantes, etc.  
BEGIN  
    .....  
END ;
```

Os tempos de uma trigger podem ser:

- BEFORE – disparada antes do evento
- AFTER – disparada depois do evento

A trigger pode ser disparada pelos eventos INSERT, UPDATE e DELETE. No uso do UPDATE, as colunas devem ser especificadas após a palavra OF.

Uma trigger pode ainda ser do tipo COMANDO, que manipula dados dentro de uma tabela e executa uma única vez; ou do tipo LINHA, que manipula linhas de uma tabela e pode ser executada uma ou mais vezes.

16.1 Criação de triggers

São permitidas até 12 triggers para cada tabela, incluindo todas as combinações possíveis entre tempos, eventos de disparo e tipos de trigger. Também não são permitidas triggers com o mesmo tempo, evento de disparo e tipo para uma mesma tabela.

16.2 Triggers possíveis para uma tabela:

BEFORE UPDATE Linha

Quando um objeto com o qual o subprograma possui uma relação de dependência for modificado, o status do subprograma na tabela USER_OBJECT ficará como INVALID.

Recompilando um subprograma que esteja como o status INVALID:

```
ALTER PROCEDURES / FUNCTION nome_da_procedure / nome_da_function COMPILE
```

Relações de dependência dos objetos podem ser encontradas na tabela USER_DEPENDENCIES.

Exemplo de como verificar os objetos que dependem de um outro objeto:

```
SELECT NAME, TYPE
```

```
FROM USER_DEPENDENCIES
```

```
WHERE REFERENCED_NAME = 'Nome_Objeto_que_se_quer_alterar';
```

15 Packages

São objetos equivalentes a bibliotecas que guardam Procedures, Functions, definições de cursores, variáveis e constantes, definições de exceções.

15.1 Especificação

Na área de **especificação** são feitas as declarações públicas (variáveis, constantes, cursores, exceções e subprogramas que estarão disponíveis para uso externo à package).

Sintaxe:

```
CREATE OR REPLACE PACKAGE Nome_da_package IS
```

Para procedures e functions só os cabeçalhos (interface)

```
PROCEDURE Nome_da_Procedure (Lista_de_Parâmetros);
```

```
FUNCTION Nome_da_Function (lista_de_Parâmetros
```

Declaração de variáveis, constantes, exceções e cursores públicos

```
END Nome_da_Package;
```

15.2 Body

No **corpo** são feitas as declarações privadas, que estarão disponíveis internamente à package

Sintaxe:

```
CREATE OR REPLACE PACKAGE BODY Nome_da_Package IS
```

Declaração de variáveis, constantes, exceções e cursores privados

```
PROCEDURE Nome_da_Procedure (Lista_de_Parâmetros)
```

```
IS
```

```
BEGIN
```

```
END;
```

```
FUNCTION Nome_da_Function (Lista_de_Parâmetros)
```

```
RETURN tipo
```

```
BEGIN
```

```
RETURN
```

```
END;
```

```
END;
```

Para eliminar uma Package, utiliza-se DROP PACKAGE e DROP PACKAGE BODY.

14.1 Parâmetros

Podem ser de entrada, saída ou entrada/Saída:

- IN (padrão) - passa um valor do ambiente chamador para o subprograma e este valor não pode ser alterado dentro do subprograma.
- OUT- passa um valor do subprograma para o ambiente chamador.
- IN OUT- passa um valor do ambiente chamador para o subprograma; este valor pode ser alterado dentro do subprograma e retornado com o valor atualizado para o ambiente chamador.

14.2 Procedures

```
CREATE OR REPLACE PROCEDURE Nome_Proc
    [(Argumento[{IN | OUT | IN OUT}] tipo,
    .....
    Argumento [{IN | OUT | IN OUT}] tipo] {IS | AS}
Corpo_procedimento
```

Onde, nome_procedimento é como se chama o procedimento, argumento é o nome de um parâmetro de procedimento, tipo é o tipo do parâmetro associado e corpo_procedimento é um bloco de PL/SQL que constitui o código do procedimento.

14.3 Functions:

Retornam resultado ou valor. Podem ser utilizadas em atribuições a variáveis ou como argumento em comando Select.

```
CREATE OR REPLACE FUNCTION Nome_função
    [(Argumento[{IN | OUT | IN OUT}] tipo,
    .....
    Argumento [{IN | OUT | IN OUT}] tipo]
RETURN tipo_retorno {IS | AS}
Corpo_função
```

Tipo_retorno é o tipo do valor que a função devolve.

14.4 Executando subprogramas através do SQL * Plus:

```
EXECUTE Nome_Procedure(Lista_de_parâmetros)
SELECT Nome_Função(Lista_de_parâmetros) FROM DUAL
```

14.5 Eliminando um subprograma:

```
DROP PROCEDURE / FUNCTION      nome_do_Procedimento / nome_da_função
```

14.6 Análise das dependências:

Um subprograma pode depender diretamente de tabelas, visões, seqüências e outros subprogramas. Pode ainda possuir dependências indiretas de outros objetos. Por exemplo, se uma procedure A depender diretamente de uma function B, que depende diretamente de uma tabela C, então a procedure A dependerá indiretamente da tabela C.

13 DBMS_output.put_line()

Coloca uma linha no buffer, e mostra na tela.

```
SQL> set serveroutput on;  
SQL> exec dbms_output.put_line(sysdate);  
10-MAR-01
```

14 Subprogramas (Procedures e Functions)

São blocos PL/SQL, armazenados na base de dados e chamados sempre que necessários.

O nome de um subprograma pode ter, no máximo, 30 caracteres. No momento de sua criação pode ser incluído o parâmetro OR REPLACE, o qual substituirá um subprograma já existente por uma nova versão. As vantagens dessa opção é manter os privilégios existentes, criar mesmo que haja erro de sintaxe, marcar objetos dependentes para compilação.

Para que um subprograma seja criado como Público utilizam-se os seguintes comandos:

```
GRANT EXECUTE ON Nome_do_Subprograma TO PUBLIC;  
  
CREATE PUBLIC SYNONYM Nome_do_Subprograma  
FOR Sinônimo_Público_do_Subprograma;
```

Informações dos subprogramas são armazenadas na tabela USER_OBJECTS, a qual pode ser acessada pelo usuário para obter dados como nome, tipo, data de criação, data de compilação, etc.

Já o texto das procedures e functions encontram-se na tabela USER_SOURCE.

Exemplo de como obter o código de um subprograma:

```
SELECT TEXT  
FROM USER_SOURCE  
WHERE NAME = 'Nome_do_subprograma em maiúsculo'  
ORDER BY LINE;
```

A lista de argumentos de um subprograma é obtida através do comando DESCRIBE.

Sintaxe:

```
DESCRIBE nome_da_procedure / function
```

Os erros de compilação podem ser obtidos de duas formas:

Ex.1:

```
SELECT LINE, POSITION, TEXT  
FROM USER_ERRORS  
WHERE NAME = 'Nome_subprograma em maiúsculo'  
ORDER BY LINE
```

Ou

Ex.2:

```
SHOW ERRORS PROCEDURE Nome_subprograma;
```

12.2 Exceções Definidas pelo Usuário

Precisam ser declaradas e chamadas explicitamente pelo comando RAISE. Somente podem ser declaradas na área de declarações de um bloco PL/SQL, subprograma ou package.

Sintaxe:

```
Declare
    Nome_da_exceção EXCEPTION;
Begin
    Relação_de_comandos
    If .....then
        RAISE Nome_da_exceção;
    End if;
    Relação_de_comandos
Exception
    WHEN Nome_da_exceção THEN
        Relação_de_comandos
End;
```

12.2.1 Utilizando OTHERS ou PRAGMA EXCEPTION_INIT

Um pragma renomeia um erro oracle para o compilador, permitindo escrever um tratamento específico, sem que uma exception seja explicitamente chamada. Os pragmas são chamados em tempo de compilação e não afetam o significado do programa. Deve ser declarado na área de declarações de um bloco, subprograma ou package e precisa Ter o nome da exceção previamente declarado.

Sintaxe:

```
Declare
    Nome_da_exceção Exception;
    Pragma Exception_Init (nome_da_exceção, código_erro);
Begin
    Relação de comandos
Exception
    When Nome_da_exceção then
        Relação_de_comandos
End;
```

12.2.2 Raise_Application_Error

É uma procedure que permite a emissão de mensagens de erro, definidas pelo usuário. Os erros podem ser relatados e evita-se o retorno de exceções não tratadas.

Sintaxe:

```
RAISE_APPLICATION_ERROR (código_erro, 'texto')
```


12 Tratamento de Exceções

As exceções podem ser predefinidas ou definidas pelo usuário.

Sintaxe:

<pre>EXCEPTION WHEN nome_da_exceção THEN Relação_de_comandos WHEN nome_da_exceção THEN Relação_de_comandos</pre>
--

12.1 Exceções Predefinidas

Este tipo de exceção é disparada implicitamente quando, no bloco PL/SQL, uma regra Oracle é violada ou um limite de sistema é excedido. Estas exceções podem ser identificadas por um nome e um número.

- **CURSOR_ALREADY_OPEN** (ORA-06511, SQLCODE -06511) ocorre quando se tenta abrir um cursor que já está aberto.
- **DUP_VAL_ON_INDEX** (ORA-00001, SQLCODE -00001) ocorre na tentativa de armazenar um valor duplicado em uma coluna de uma tabela que possui chave única ou primária.
- **INVALID_CURSOR** (ORA-01001, SQLCODE -01001) ocorre quando se tenta executar uma operação ilegal com um cursor, como fechar um cursor que não esteja aberto.
- **INVALID_NUMBER** (ORA-01722, SQLCODE -01722) ocorre na tentativa de converter uma string em número, caso a string não represente um número válido.
- **LOGIN_DENIED** (ORA-01017, SQLCODE -01017) ocorre na tentativa de conexão com o banco com um username/passvord inválido.
- **NO_DATA_FOUND** (ORA-01403, SQLCODE -01403) ocorre quando um comando SELECT ... INTO não retornar nenhuma linha.
- **NOT_LOGGED_ON** (ORA-01012, SQLCODE -01012) ocorre na tentativa de acessar o banco de dados sem que se esteja conectado a ele.
- **PROGRAM ERROR** (ORA-06501, SQLCODE -06501) ocorre em caso de problemas internos do PL/SQL.
- **ROWTYPE_MISMATCH** (ORA-06504, SQLCODE -06504) ocorre se o retorno do cursor e a variável PL/SQL para retorno de um cursor sejam de tipos incompatíveis.
- **STORAGE_ERROR** (ORA-06500, SQLCODE -06500) ocorre se não houver memória suficiente para a execução de um bloco PL/.
- **TIMEOUT_ON_RESOURCE** (ORA-00051, SQLCODE -00051) ocorre quando acontecer um timeout enquanto o Oracle estiver aguardando um recurso.
- **TOO_MANY_ROWS** (ORA-01422, SQLCODE -01422) ocorre quando um comando SELECT... INTO retornar mais de uma linha.
- **VALUE-ERROR** (ORA-06502, SQLCODE -06502) ocorre quando houver um erro aritmético, de conversão, truncagem ou tamanho, como quando um valor numérico for selecionado para dentro de uma variável character, ou o valor for maior do que o declarado para a variável.
- **ZERO-DIVIDE** (ORA-01476, SQLCODE -01476) ocorre na tentativa de dividir qualquer número por zero.
- **OTHERS** permite tratar outros erros, com a ajuda das funções SQLCODE e SQLERRM, que retorna o número do erro Oracle e o texto da mensagem de erro, respectivamente.

Variáveis que possuam restrição de limites (inferior e superior) pela definição de tipo ou subtipo dispararão restrição de VALUE-ERROR caso estes limites sejam ultrapassados.

Sintaxe:

```
CURSOR nome_do_cursor IS  
SELECT ....  
FOR UPDATE OF campos_a_atualizar  
UPDATE ...  
WHERE CURRENT OF nome_do_cursor
```

11.4 Cursores Implícitos:

Atributos de cursores que podem ser verificados com significado e retorno:

- SQL%FOUND – retorna true se algum registro foi afetado ou se retornou algum registro;
- SQL%NOTFOUND – retorna true se nenhum registro foi afetado.
- SQL%ROWCOUNT – retorna numero de registros afetados ou a última quantidade de registros afetados(deverá ser sempre 1)
- SQL%ISOPEN – sempre retornará false

Nome_do_cursor%ISOPEN - retorna TRUE caso o cursor esteja aberto e FALSE, caso contrário.

11.1.3 Close

Libera a área de memória utilizada pelo cursor.

Sintaxe:

```
CLOSE nome_do_cursor
```

Ex.:

```
CLOSE Preferencia;
```

Exemplo Completo:

```
Declare
Cursor Preferencia (Cliente Number) IS
  Select Distinct      I.Cd_Produto,
                      Rpad(P.Nm_Produto, 30) Nome
  From  Item_Nota_Fiscal I, Nota_Fiscal N, Produto P
  Where N.Cd_Cliente = Cliente
  And   N.Nr_Nota = I.Nr_Nota
  And   I.Cd_Produto = P.Cd_Produto;
Reg_Pref Preferencia %Rowtype;
Begin
  Open Preferencia (1);

  Loop
    Fetch Preferencia
    Into Reg_Pref;
    Exit When Preferencia%Notfound;

    Dbms_output.Put_line ('Produto: '||Reg_Pref.Nome);
  End Loop;
  Close Preferencia;
End;
```

11.2 O Comando For para abrir Cursores:

- Cria a variável do tipo registro
- Abre o cursor
- Realiza a cópia das linhas (Fetch)
- Controle o final do cursor
- Fecha o mesmo

Para sair do laço, o cursor deve ser fechado explicitamente com o comando close.

Sintaxe:

```
FOR Nome_da_variavel_tipo_registro
IN   Nome_do_cursor (Lista_de_parametros) LOOP
  Relação_de_comandos
END LOOP;
```

11.3 Atualização na tabela da linha atual do cursor:

Declara-se o cursor como For Update e a atualização será feita com base no indicador de linha corrente (CURRENT OF). O ROWID do registro será carregado com os demais itens do cursor e no UPDATE, a comparação será feita internamente, com o ROWID.

<pre>FETCH nome_do_cursor INTO lista_de_variáveis</pre>

Ex.1:

```
Declare  
Cursor Preferencia IS  
Select Distinct          I.Cd_Produto,  
                        Rpad(P.Nm_Produto, 30) Nome  
  
From Item_Nota_Fiscal I, Nota_Fiscal N, Produto P  
Where N.Cd_Cliente = cliente  
      N.Nr_Nota = I.Nr_Nota  
      I.Cd_Produto = P.Cd_Produto;  
  
Codigo Produto.Cd_Produto%Type;  
Nome   Produto.Nm_Produto%Type;  
Begin  
...  
Fetch Preferencia  
Into Codigo, Nome;  
  
Dbms_Output.Put_Line ('Produto: '||Nome);  
.....  
End;
```

Ex. 2:

```
Declare  
Cursor Preferencia (Cliente Number) IS  
Select Distinct I.Cd_Produto,  
                Rpad(P.Nm_Produto, 30) Nome  
From Item_Nota_Fiscal I, Nota_Fiscal N, Produto P  
Where N.Cd_Cliente = Cliente  
      And N.Nr_Nota = I.Nr_Nota  
      And I.Cd_Produto = P.Cd_Produto;  
Reg_Pref Preferencia %RowType;  
Begin  
.....  
Fetch Preferencia  
Into Reg_Pref;  
  
Dbms_Output.Put_Line ('Produto: '|| Reg_Pref.Nome);  
.....  
End;
```

Para cada cursor, quatro atributos podem ser verificados a cada execução do comando FETCH:.

Nome_do_cursor%FOUND - retorna TRUE caso o FETCH consiga retornar alguma linha e FALSE, caso contrário. Se nenhum FETCH tiver sido executado será retornado NULL.

Nome_do_cursor%NOTFOUND - retorna FALSE caso o FETCH consiga retornar alguma linha e TRUE, caso contrário. Se nenhum FETCH tiver sido executado será retornado NULL.

Nome_do_cursor%ROWCOUNT - retorna o número de linhas já processadas pelo cursor. Se nenhum FETCH tiver sido executado será retornado 0 (zero)

```
LOOP
    Relação de comandos
    IF Condição_de_saida then EXIT
END LOOP;
```

11 Cursores:

- Guarda resultados de uma seleção em memória, permitindo a manipulação deste resultado de uma maneira procedural;
- deve ser declarado na área de declarações;
- o nome não pode ser igual ao da tabela;
- para dar um nome a uma coluna da seleção basta colocar o nome do alias logo após a definição da coluna.

Sintaxe:

```
CURSOR nome_do_cursor IS
SELECT ...
[FOR UPDATE OF colunas]
```

Em que Relação_de_parâmetros pode Ter o seguinte formato:

Nome_do_parâmetro tipo_de_dado {:= / DEFAULT} valor_inicial.

Obs.: O tamanho do parâmetro não pode ser declarado, somente seu tipo.

Ex.:

```
CURSOR Preferencia IS
SELECT DISTINCT I.Cd_Produto, NM_Produto Nome
FROM Item_Nota_Fiscal I, Nota_Fiscal N, Produto P
WHERE N.Cd_Cliente = cliente
AND      N.Nr_Nota = I.Nr_Nota
AND      I.Cd_Produto = P.Cd_Produto;
```

11.1 Comandos de Manipulação do cursor:

11.1.1 Open:

Cria numa área de memória conhecida como Private SQL Area, uma tabela com um ponteiro apontando para o primeiro registro. Parâmetros devem ser passados neste momento.

Sintaxe:

Open Preferencia ;

11.1.2 Fetch

Transfere as linhas armazenadas no cursor, para variáveis, além de posicionar o ponteiro no próximo registro do cursor.

A lista de variáveis que aparece na sintaxe do comando FETCH deve ter o mesmo número de variáveis, na mesma seqüência e com tipos correspondentes às colunas selecionadas no comando SELECT da declaração do cursor.

Sintaxe:

```
IF expressão_booleana1 THEN  
    Sequencia_de_comandos1;  
[ELSIF expressão_booleana2 THEN  
    Sequencia_de_comandos2;]  
[ELSE  
    Sequencia_de_comandos3;]  
END IF;
```

Ex:

```
IF V_SALARIO < 100 THEN  
    V_FUNCIONARIO := 'COITADO';  
    V_PRIVILEGIOS := 0;  
ELSIF V_SALARIO = 100 THEN  
    V_FUNCIONARIO := 'ESTA MELHORANDO';  
    V_PRIVILEGIOS := 0;  
ELSE  
    V_PRIVILEGIO := 1;  
END IF;
```

10.2 WHILE-LOOP

Sintaxe:

```
WHILE condição LOOP  
    Sequencia_de_comandos1;  
END LOOP;
```

Ex.:

```
WHILE V_CONTADOR < 50 LOOP  
    V_CONTADOR := V_CONTADOR + 1;  
END LOOP ;
```

10.3 FOR-LOOP

Sintaxe:

```
FOR variável_contador IN [REVERSE]  
    valor_inicial..valor_final  
LOOP  
    Sequencia_de_comandos1;  
END LOOP;
```

Ex:

```
FOR V_CONTADOR IN 1..50 LOOP  
    V_TESTE := V_CONTADOR;  
END LOOP;
```

10.4 LOOP

Sintaxe

```
COMMIT;  
ROLLBACK;  
SAVEPOINT Nome_do_Ponto;  
ROLLBACK TO Nome-do-Ponto;
```

Ex.:

Declare

```
Nr_Nova_Nota    Nota_Fiscal.Nr_Nota%Type;
```

Begin

-- **insere dados nas tabelas Empresa e Filial**

```
Insert into Empresa (Cd_Empresa, Nm_Empresa)
```

```
Values (3, 'Empresa teste 3');
```

```
Insert into Filial (Cd_Empresa, Cd_Filial, Nm_Filial)
```

```
Values (3, 1, 'Filial I - BC');
```

```
SavepointLocal;
```

-- **insere dados nas tabelas Estado e município**

```
Insert into Estado (Cd_Pais, Sg_Estado, Nm_Estado)
```

```
Values (1, 'SP', 'São Paulo');
```

```
Insert into municipio (Cd_Pais, Sg_Estado, cd_Municipio,
```

```
Nm_municipio)
```

```
Values (1, 'SP', 10, 'Sorocaba');
```

```
Savepoint Pessoa;
```

-- **Insere dados na tabela Cliente**

```
Insert into Cliente
```

```
(Cd_Cliente, Nm_Cliente, Ds-Endereco, Cd-Municipio, Sg_Estado, Nr_Cep,  
NR_DDD, Nr_Fone, le_Sexo, Ie_Fisica_juridica, Cd_Estado_Civil, Cd_Pais,  
Ie_Situacao)
```

```
Values (50, 'Fulano da Silva', 'Estrada Geral de Ida e Vinda',
```

```
10, 'Sp', '890103301', 047, 2312331, 'M', 'S', 1, 1, 'A');
```

```
Savepoint Notas;
```

-- **Insere dados nas tabelas Nota_Fiscal e Item_Nota_Fiscal**

```
Select Nvl (Max (Nr_Nota) , 0) + 1
```

```
Into Nr_Nova_Nota
```

```
From      Nota_Fiscal;
```

```
Insert into Nota_Fiscal
```

```
(Nr_Nota,          Cd_Cliente,          Dt_Emissao,          le_Tipo_Nota)
```

```
Values
```

```
(Nr_Nova_Nota,      50,                  sysdate,          'C');
```

```
Insert into Item_Nota_Fiscal
```

```
(Nr_Nota,      Nr_ltem,          Cd_Produto,          Qt_Produto, VI_Unitario)
```

```
Values
```

```
(Nr_Nova_Nota,      1 ,      1,                  15,                  3.25);
```

```
Rollback to Pessoa;
```

```
Commit;
```

End,

10 Estruturas de Controle

10.1 IF – THEN – ELSE

Sintaxe:

```
Declare
    VI_Custo          Varchar2;
    NVI_Custo         Number;
Begin
    --Estão disponíveis VI_Custo de tipo varchar2, NVI Csto e Nm Produto.
End;

--Estão disponíveis VI-Custo de tipo number e Nm Produto.

End;
```

9 Codificação de Comando SQL Dentro de PL/SQL

Comandos como INSERT, DELETE, UPDATE e SELECT e as funções (manipulações de strings, numéricas, de datas e genéricas), podem ser utilizados dentro de blocos PL/SQL.

O comando SELECT receberá, **obrigatoriamente**, a cláusula INTO, pois o resultado do mesmo não poderá ser visualizado no momento de sua execução, e será armazenado em variáveis ou estruturas.

O comando SELECT deve ser criado para que retorne somente uma linha selecionada. Caso nenhuma linha seja retornada ocorrerá um erro do tipo "no_data_found" e se mais de uma linha for retomada ocorrerá um erro do tipo "too_many_rows".

```
Ex.:
Declare
    Nome              Cliente.Nm_Cliente%Type;
    Codigo            Cliente.Cd_Cliente%Type;
Begin
    SELECT            Nm_cliente, Cd_Cliente
    INTO              Nome, Codigo
    FROM              Cliente
    WHERE Cd_Cliente = &Cod;
End;
```

9.1 Tratamento de Transações

Comandos de terminação das transações:

- CONNECT/DISCONNECT
- Comandos DDL (que gere alterações no dicionário de dados)
- Encerramento explícito por meio dos comandos COMMIT/ ROLLBACK.

Savepoints:

- COMMIT - finaliza a transação efetivando as atualizações no banco, de forma que os demais usuários (sessões) consigam acessar essas alterações.
- ROLLBACK - finaliza a transação desfazendo **todas** as alterações feitas no banco durante a transação.
- ROLLBACK TO - desfaz as alterações realizadas no banco a partir da primeira instrução após o ponto especificado. Ou seja, permite desfazer apenas **parte** de uma transação. O ponto a partir do qual será desfeita a transação deve ter sido especificado com o comando SAVEPOINT.
- SAVEPOINT - permite a especificação de um ponto de processamento dentro de uma transação.

Sintaxe


```
--Subtipo predefinido
Subtype      Character      Is Char;
--Baseado em um tipo tabela
Type         NameTab        Is TABLE OF Varchar2(10)
                                INDEX BY BINARY INTEGER;
Subtype      EnameTab        IS NameTab;
--Baseado em um tipo registro
Type         TpTime          Is RECORD
              (minuto        Integer,
              hora            Integer);
Subtype      Momento         Is TpTime;
--Baseado em uma coluna
Subtype      TpCodigo        is Cliente.CdCliente%Type;
--Baseado no registro de um cursor
Cursor       Cur-Pais        is Select * From Pais;
Subtype      Rec-Pais        is Cur_Pais%Rowtype;
--Utilizando subtipos criados
NmClienteAux                               Nome,
ContAux                                     Contador;
Begin
    Null;
End;
```

8.4 Atribuindo Valores às Variáveis

Pode-se utilizar o operador (**:=**), ou a partir de um comando **SELECT** atribuir valores com o uso da cláusula **INTO**. Neste caso o comando deve retornar apenas uma linha.

```
Ex:
Select Qt_Produto, VI_Unitario
Into   Qtdd, Valor
From   Item_Nota_Fiscal
Where  nr_nota = 1;
```

```
Total := Qtdd * Valor;
```

```
Existe := (Conta < 1200);
```

8.5 Escopo de Variáveis

Quando uma variável é definida dentro de um bloco, ela não é reconhecida fora do bloco, ou seja, ela é local para este bloco e global para os sub-blocos. Como um bloco somente pode incluir variáveis locais e globais, blocos isolados não poderão referenciar variáveis declaradas em outros blocos.

Se uma variável global for declarada em um sub-bloco, a declaração local prevalecerá,

```
Ex.:
Declare
    Nm_Produto          Produto.Nm_Produto%Type;
    VI_Custo            Number;
Begin
--Estão disponíveis VI_Custo de tipo number e Nm_Produto.
```

```
...  
END;
```

No exemplo acima são definidos dois tipos T_NAME e T_DATE como sendo do tipo TABLE. As variáveis V_NAME e V_DATE são declaradas como T_NAME e T_DATE respectivamente. Ao longo do bloco V_NAME de índice 1 recebe 'JONNAS' e V_DATE de índice 1 recebe a data atual do sistema.

8.2.2.2.1 Atributos do tipo TABLE.

Atributo	Tipo Retornado	Descrição
COUNT	NUMBER	Retorna o número de linhas na tabela
DELETE	N/A	Deleta uma linha na tabela
EXISTS	BOOLEAN	Retorna true se existe o índice na tabela
FIRST	BINARY_INTEGER	Retorna o índice da primeira linha da tabela
LAST	BINARY_INTEGER	Retorna o índice da última linha da tabela
NEXT	BINARY_INTEGER	Retorna o índice da próxima linha
PRIOR	BINARY_INTEGER	Retorna o índice da linha anterior

8.2.2.3 Usando %ROWTYPE

Assim como o %TYPE, o %ROWTYPE fará com que a variável assuma a mesma estrutura de uma linha da tabela, ou seja, a variável será um tipo RECORD que terá todos os campos da tabela que aparece na declaração.

Ex:

```
DECLARE  
    R_DEPT SCOTT.DEPT%ROWTYPE ;  
BEGIN  
    ...  
    R_DEPT.DEPTNO := 1;  
    ...  
END;
```

No exemplo acima R_DEPT é declarada com a mesma estrutura da tabela DEPT do schema SCOTT. Logo abaixo, o campo DEPTNO do RECORD R_DEPT recebe o valor 1.

8.3 Subtype

Podem ser definidos pelo usuário e servem para colocar restrições opcionais aos tipos já existentes. Um subtipo deve ser antes baseado em um tipo já existente:

Sintaxe:

```
SUBTYPE nome_do_subtipo IS nome_do_tipo_base;
```

Ex.:

```
DECLARE  
--Baseado no Tipo escalar(Date) ou subtipo (Natural) padrão  
Subtype EmpDate Is Date;  
Subtype Contador Is Natural;  
--Baseado no tipo varchar2 e restrição de tamanho máximo (50) AuxNome  
AuxNome Varchar2(50);  
Subtype Nome Is Aux_nome%Type;
```

8.2.2.1 RECORDS

Sintaxe:

```
TYPE record_type IS RECORD (  
    Campo1 type1 [NOT NULL] [:=exp1],  
    Campo1 type2 [NOT NULL] [:=exp2],  
    ...  
    Campo1 type3 [NOT NULL] [:=exp3])
```

Ex:

```
DECLARE  
    TYPE T_EMP IS RECORD (  
        EMPNO NUMBER(4),  
        ENAME VARCHAR2(10),  
        JOB VARCHAR2(9));  
  
    R_EMP T_EMP;  
  
BEGIN  
    ...  
    R_EMP.EMPNO := 1;  
    ...  
END;
```

No exemplo acima T_EMP é declarado como um tipo RECORD com os campos EMPNO, ENAME, JOB. Depois R_EMP é declarado como o tipo T_EMP.

8.2.2.2 TABLE

Tables são similares a vetores, porém é implementado de uma maneira diferente pelo Oracle.

Sintaxe

```
TYPE tabletype IS TABLE OF type INDEX BY BINARY_INTEGER
```

Onde:

TYPE : indica a declaração de um tipo definido pelo usuário.

Tabletype : é o nome do tipo que está sendo criado.

IS TABLE OF : significa que o tipo será do tipo TABLE.

Type : Indica que será um tipo TABLE deste tipo. Podendo ser um tipo RECORD.

INDEX BY BINARY_INTEGER : faz parte da sintaxe. Futuramente pode ser indexado por outros tipos, porém na versão atual apenas por BINARY_INTEGER.

Ex:

```
DECLARE  
    TYPE T_NAME IS TABLE OF SCOTT.EMP.ENAME%TYPE  
        INDEX BY BINARY_INTEGER;  
    TYPE T_DATE IS TABLE OF DATE  
        INDEX BY BINARY_INTEGER;  
    V_NAME T_NAME;  
    V_DATE T_DATE;  
BEGIN  
    ...  
    V_NAME(1) := 'JONNAS';  
    V_DATE(1) := SYSDATE;
```

DATE

- **Date** - data e hora (formato-padrão: DD-MON-YY, 26-APR-74).

ROWID

- **Rowid** - utilizado para armazenar valores de Rowid, selecionados de linhas de tabelas. Este tipo normalmente é empregado quando se quer eliminar ou alterar um registro, o qual tenha sido previamente selecionado, visto que Rowid é uma forma rápida de acessar uma linha em uma tabela. Formato do Rowid: OOOOOOFFFFBBBBBSSS. Onde OOOOOO- identifica o objeto, FFF-identifica o Data File,BBBBBB- identifica o bloco dentro do Data file e o SSS identifica a linha dentro do bloco.

RAW

- **Raw** - armazena valores hexadecimais com tamanho variável (máximo de 2K). Normalmente, este tipo de campo é utilizado para armazenamento de imagens.
- **Long Raw** - armazena valores hexadecimais com tamanho variável (máximo de 2G). Também utilizado para armazenamento de imagens.

BOOLEAN

- **Boolean** - permite armazenar os valores TRUE, FALSE ou NULL. No momento de utilizar este tipo de variáveis em atribuições, pode-se atribuir um dos valores permitidos explicitamente ou uma condição(>, <, =, <>, ...) cujo valor de veracidade será atribuído à variável.

8.2.1.1 Usando %TYPE

Na maioria dos casos os programas em PL/SQL irá manipular dados que estão armazenados em tabelas do Banco de Dados. Neste caso a variável deve ser declarada com o mesmo tipo do campo da tabela.

```
Ex:
DECLARE
    V_DEPTNO SCOTT.DEPT.DEPTNO%TYPE ;
    V_EMPNO  SCOTT.EMP.EMPNO%TYPE   ;
BEGIN
    ...
```

No exemplo acima V_DEPTNO é declarada com o mesmo tipo do campo DEPTNO da tabela DEPT do schema SCOTT. V_EMPNO é declarada com o mesmo tipo do campo EMPNO da tabela EMP do schema SCOTT.

8.2.2 Tipos de Dados Composto(TABLES e RECORDS PL/SQL).

Ambas as composições são tipos definidos pelo usuário, para usá-los, é necessário primeiro definir um tipo RECORD ou TABLE e depois declarar uma variável com o tipo que foi definido.

.....
END;

8.2 Tipos de Dados PL/SQL

Existem basicamente duas categorias de tipos de dados PL/SQL, que são escalar, composto.

8.2.1 Escalar

Tipos escalares podem ser divididos em seis famílias: Number, character, date, raw, rowid, boolean.

NUMBER

- **Number** - numérico com tamanho máximo de 38 caracteres. Na especificação de tipos de dados numéricos com casas decimais, primeiro é informado o número total de dígitos, que inclui as casas decimais, cujo número de dígitos estará separado do primeiro por uma vírgula.
 - Number (4) - no máximo 4 caracteres numéricos.
 - Number (12,2) - no máximo 12 números (10 inteiros e 2 decimais).
 - Number (-3,8) - 0 inteiros e 8 decimais, em que somente as 3 últimas podem ter valor.

Este tipo de variável possui alguns subtipos derivados:

- **Decimal** - subtipo idêntico ao tipo Number.
- **Dec, DoublePrecision, Integer, Int, Numeric, Real e SmallInt** - são subtipos do tipo Number que apresentam, como diferença deste, apenas diferentes faixas de valores permitidos.
- **Float** - permite armazenar valores de até 126 dígitos binários.
- **PLS_Integer** - armazena valores numéricos, positivos e negativos, entre -2147483647 e 2147483647. Este tipo de variável requer menor espaço de armazenamento do que uma variável do tipo Number e permite melhor desempenho em cálculos do que os tipos Binary-integer ou Number.
- **Binary-Integer** - armazena valores numéricos, positivos e negativos, entre -2147483647 e 2147483647. Este tipo de variável requer menor espaço de armazenamento do que uma variável do tipo Number e possui quatro subtipos derivados desta:
 - **Natural** - pode armazenar valores entre 0 e 2147483647.
 - **NaturalN** - armazena valores entre 0 e 2147483647 e não pode receber valores nulos.
 - **Positive** - pode armazenar valores entre 1 e 2147483647.
 - **PositiveN** - armazena valores entre 1 e 2147483647 e não pode receber valores nulos.

CHARACTER

- **Char** - alfanumérico de tamanho fixo, máximo de 255 caracteres.
- **Character** - subtipo idêntico ao tipo Char.
- **Varchar2** - alfanumérico de tamanho máximo de 2.000 caracteres. A principal diferença deste tipo para o tipo Char é que com o tipo Varchar2 o número de caracteres que não for utilizado não ocupa espaço no banco de dados.
- **Varchar e String** - subtipos idênticos ao tipo Varchar2, mas são utilizados apenas para manter compatibilidade com versões diferentes ou anteriores do SQL.
- **Long** - alfanumérico com tamanho máximo de 2G. (O tamanho não pode ser informado.) Só pode existir um por tabela e não pode ser utilizado na cláusula WHERE de consultas.

SHARE ROW EXCLUSIVE – nenhum outro usuário poderá bloquear a tabela. Diferentemente dos outros bloqueios tipo SHARE, somente um usuário por vez poderá fazer o COMMIT na tabela.

EXCLUSIVE – só o usuário gerador do bloqueio poderá fazer alterações.

NOWAIT – libera o processo, caso não for possível efetuar o bloqueio.

Ex.:

```
LOCK TABLE BLOQUEIO  
In Exclusive Mode;
```

O LOCK termina com o fim da transação (COMMIT/ROLLBACK).

7.1.2 Select for Update

Bloqueia antecipadamente as linhas que serão alteradas ou excluídas até o fim da transação.

Sintaxe:

```
SELECT...  
FOR UPDATE OF nome_da_coluna1 [,nome_da_colunaN]  
[NOWAIT]
```

Ex.:

```
SELECT COD, TIPO_BLOQ  
FROM BLOQUEIO  
WHERE COD = 1  
FOR UPDATE OF COD;
```

O uso do Select for Update não é permitido caso o comando select tenha as cláusulas DISTINCT, GROUP BY, UNION, INTERSECT ou MINUS e funções de grupo(Count, AVG, Sum, Max, etc)

8 Declarações

8.1 Declarações de Variáveis

Variáveis são declaradas na seção de declaração de variáveis. Em geral a sintaxe para a declaração de variáveis é:

Sintaxe :

```
Nome_variavel type [CONSTANT] [NOT NULL] [:=value]
```

Onde:

Nome_variável : é o nome da variável.

Type : é o tipo que a variável irá assumir.

CONSTANT : se utilizado, a variável precisa ser inicializada com um valor, e este valor não pode ser alterado ao longo do programa.

NOT NULL : significa que a variável não poderá assumir NULL ao longo do programa. Quando utilizada a variável deve ser inicializada com um valor.

Value : é um valor para inicializar a variável.

Ex:

```
DECLARE  
  Const_Temp  CONSTANT  NUMBER(5)  :=  15  ;  
              V_SEQ     NUMBER(5)      ;  
  
BEGIN
```

6 Índices

São estruturas que permitem a recuperação rápida de dados.

Sintaxe:

```
CREATE [UNIQUE] INDEX nome_do_indice ON nome_da_tabela  
(nome_da_coluna ASC/DESC[,nome_da_coluna ASC/DESC]...)
```

Ex.:

```
CREATE INDEX FONE ON CLIENTE  
(nr_fone);
```

Os índices não podem ser alterados. É necessário excluí-los e criá-los novamente. Para excluir um índice utiliza-se o comando DROP INDEX.

6.1 Recuperando informações sobre Índices:

Algumas informações sobre os índices (INDEX_NAME, TABLE_OWNER, TABLE_NAME) são armazenadas numa tabela de controle chamada USER_INDEXES e podem ser recuperados a partir do seguinte comando:

Sintaxe:

```
SELECT * FROM USER_INDEXES  
WHERE TABLE_NAME = 'nome_da_tabela';
```

O nome da tabela além de estar entre aspas simples deve ser digitado em letras maiúsculas. Também podem ser recuperadas informações sobre colunas (INDEX_NAME, TABLE_NAME, COLUMN_NAME, COLUMN_POSITION, COLUMNS_LENGTH). Estas informações ficam armazenadas na tabela de controle USER_IND_COLUMNS.

7 Controle de Concorrência

O uso da função **lock** mantém a consistência do banco de dados no caso de acesso simultâneo.

7.1 Tipos de Lock

- Lock implícito – gerado internamente pelo banco.
- Lock explícito – criado via comando SQL e pode ser gerado a partir dos comandos **Lock table** e **Select for update**.

7.1.1 Lock table – sobrepõe o lock implícito

Sintaxe:

```
LOCK TABLE [nome_da_tabela] IN  
ROW SHARE / ROW EXCLUSIVE/  
SHARE UPDATE / SHARE /  
SHARE ROW EXCLUSIVE / EXCLUSIVE  
MODE [NOWAIT]
```

Nome_da_tabela – indica qual tabela sofrerá o bloqueio.

ROW SHARE – nenhum outro usuário poderá bloquear exclusivamente a tabela.

ROW EXCLUSIVE – permite acesso simultâneo à tabela, bloqueando linhas individuais.

SHARE UPDATE – Somente bloqueia linhas simples, permitindo a continuidade na consulta de dados.

SHARE – os outros usuários podem consultar os dados mas não podem alterá-los.

5 Seqüências

São estruturas criadas no banco de dados que retornam valores diferentes a cada acesso. Por default este valor cresce seqüencialmente.

Sintaxe:

```
CREATE SEQUENCE nome_da_sequencia  
[STAR WITH valor_inicial] (default1)  
[INCREMENT BY incremento] (default1)  
[MAXVALUE valor_maximo_da_sequencia/NOMAXVALUE]  
[MINVALUE valor_minimo_da_sequencia/NOMINVALUE]  
[CYCLE/NOCYCLE] (se for cíclica, quando atingir o máximo volta ao início)
```

Ex.:

```
CREATE SEQUENCE SEQ  
START WITH 6;
```

Para pegar o valor corrente da seqüência, utiliza-se o valor CURRVAL e para pegar o próximo valor, utiliza-se NEXTVAL. Somente se inicia uma seqüência com o primeiro NEXTVAL referente a ela.

Sintaxe:

```
SELECT nome_da_sequencia.CURRVAL from Dual;  
SELECT nome_da_sequencia.NEXTVAL from Dual;
```

Ex.:

```
SELECT SEQ.NEXTVAL from Dual;  
6  
SELECT SEQ.CURRVAL from Dual;  
6  
SELECT SEQ.NEXTVAL from Dual;  
7
```

5.1 Alterando uma seqüência

Sintaxe:

```
ALTER SEQUENCE nome_da_sequencia  
Opção
```

Ex.:

```
ALTER SEQUENCE SEQ  
INCREMENT BY 2;
```

5.2 Eliminando uma seqüência

Sintaxe:

```
DROP SEQUENCE nome_da_sequencia
```

Ex.:

```
DROP SEQUENCE SEQ;
```


Ex.:

```
CREATE TABLE SALGRADE_TTEMP  
AS SELECT GRADE, LOSAL  
FROM SALGRADE_TEMP;
```

4.1.5 Rename

Sintaxe:

RENAME nome_antigo_da_tabela TO nome_novo_da_tabela

Ex.:

```
RENAME SALGRADE_TTEMP TO SALGRADE_T;
```

4.1 Comandos de Manipulação de Dados (DML)

4.1.1 Insert

Sintaxe:

```
INSERT INTO  
nome_tabela(campo1,campo2...)  
          VALUES (valor1,valor2...);
```

Ex:

```
INSERT INTO SALGRADE(GRADE,LOSAL,HISAL)  
VALUES(6, 1500, 3000);
```

Sintaxe:

```
INSERT INTO  
nome_tabela(campo1,campo2...)  
          SELECT campo1, campo2  
          FROM nome_tabela2;
```

Ex:

```
create table SALGRADE_TEMP (GRADE NUMBER, LOSAL NUMBER, HISAL NUMBER);
```

```
INSERT INTO SALGRADE_TEMP(GRADE,LOSAL,HISAL)  
SELECT GRADE,LOSAL,HISAL  
FROM SALGRADE;
```

4.1.2 Update

Sintaxe:

```
UPDATE nome_tabela  
SET campo1 = valor1 [,campo2 = valor2...]  
[WHERE condição]
```

Ex:

```
UPDATE SALGRADE_TEMP SET HISAL = '2000' WHERE LOSAL < 2000;
```

4.1.3 Delete

Sintaxe:

```
DELETE FROM nome_tabela  
[WHERE condição]
```

Ex:

```
DELETE FROM SALGRADE_TEMP WHERE LOSAL < 1000;
```

4.1.4 Create Table.....As Select

Sintaxe:

```
CREATE TABLE nome_da_tabela  
(nome_da_coluna [restrições] [,  
  nome_da_coluna [restrições]] [, restrições])  
AS SELECT.....
```

Exemplos

```
SQL> ACCEPT SALARY NUMBER PROMPT 'Salary figure : '
```

```
Salary figure : 30000
```

```
SQL> ACCEPT PASSWORD CHAR PROMPT 'Password : ' HIDE
```

```
Password :
```

```
SQL> ACCEPT COM NUMBER NOPROMPT
```

```
500
```

```
SQL> DEFINE
```

```
DEFINE SALARY          =      30000 (NUMBER)
DEFINE PASSWORD         =      "FREEBIES" (CHAR)
DEFINE COM              =      500 (NUMBER)
```

Como fazer variáveis permanecerem definidas? Até que você as redefina UNDEF(INE) ou até você sair do SQL*Plus.

Duas outras maneiras para definir uma variável:

```
SQL> ACCEPT variável (tipo) (PROMPT 'texto') (HIDE)
```

```
SQL> COLUMN nome coluna NEW_VALUE variável
```

4 Recuperando valores da base de dados

Você pode atribuir resultados de queries para variáveis, porém tomando cuidado para que a query sempre retorne apenas uma linha.

Ex.:

```
SELECT NOTAFISCAL , DATA
INTO V_NOTAFISCAL , V_DATA
WHERE NUMERO_VENDA = 2334;
```

O comando acima atribui os valores dos campos NOTAFISCAL e DATA para as variáveis previamente definidas V_NOTAFISCAL e V_DATA respectivamente.

MILLER 10 CLERK

É perguntado uma vez e não mais.

3.3 O Comando DEFINE

Um valor pode ser atribuído para uma variável usando o comando DEF[INE] do SQL*Plus. O valor atribuído pode ser referenciado na declaração SELECT ou pelo nome de variável predefinido de um (&).

Sintaxe:

```
SQL>DEFINE var = valor
```

No exemplo seguinte, uma variável tem seu conteúdo definido como uma expressão aritmética que calcula a remuneração. Na subsequente declaração, a variável REM é referenciada para um número de vezes. A variável é então esvaziada usando UNDEF(INE):

```
DEFINE REM = "SAL*12+NVL(COMM,0)";
```

```
SELECT ENAME, JOB, &REM  
FROM   EMP  
ORDER BY &REM;
```

```
UNDEFINE REM
```

NOME	OCUP	SAL*12+NVL(COM,0)
SMITH	CLERK	9600
JAMES	CLERK	11400
ADAMS	CLERK	13200
WARD	SALESMAN	15500
MILLER	CLERK	15600
MARTIN	SALESMAN	16400
TURNER	SALESMAN	18000
ALLEN	SALESMAN	19500

As Aspas duplas em volta da expressão são opcionais a menos que a expressão tenha espaços.

3.4 O comando ACCEPT

O comando ACCEPT permite criar uma variável. O ACCEPT é geralmente usado num arquivo comando. Esta variável então pode ser referenciada na declaração do SQL. Existem benefícios em usar o ACCEPT para definir Variáveis Substituíveis.

- Dados tipo Data podem ser checados.
- A mensagem de entrada de dados pode ser mais explicativa
- Valores da resposta podem ser escondidos

A sintaxe do comando é:

```
ACC(EPT)variable(NUMERICO/ALFANUMERICO)(PROMPT/NOPROMPT 'texto') (HIDE)
```

Sintaxe	Descrição
NUMBER/CHAR	determina o tipo de variável. Se o valor entrado for inválido uma mensagem será mostrada.
PROMPT 'texto'	mostra o texto se for especificado
NOPROMPT	faz o ACCEPT omitir uma linha aguardando a entrada
HIDE	esconde entrada para o usuário, por exemplo, no caso de senha.

1.1 Programação em Oracle8 PL/SQL

Valores alfanuméricos ou datas, precisam ser incluídos entre aspas simples na entrada. Para evitar a entrada das aspas simples na execução, declara-se a variável entre aspas simples.

Na declaração seguinte, as variáveis estão incluídas entre aspas simples, só que as aspas simples não são requeridas na execução:

```
SELECT      ENAME, DEPTNO, SAL*12
FROM        EMP
WHERE JOB = '&JOB';
```

Enter value for OCUP: MANAGER

ENAME	DEPTNO	SAL*12
JONES	20	35700
BLAKE	30	34200
CLARK	10	29400

O tamanho da variável é indefinido e o valor será pedido toda vez que for executada a declaração. É ainda possível entrar com um nome de coluna de um tabela na execução. No seguinte exemplo você entrará com um expressão aritmética:

```
SELECT DEPTNO, &ARITHMETIC_EXPRESSION
FROM EMP;
Enter value for arithmetic_expression: sal/12
```

DEPTNO	&ARITHMETIC_EXPRESSION
20	66,666667
30	133,33333
30	104,16667
20	247,91667
30	104,16667
30	237,5
10	204,16667
20	250
10	416,66667
30	125
20	91,666667
30	79,166667
20	250
10	108,33333

3.2 Duplo & para Variáveis substituíveis

Se uma variável é prefixada com um duplo "&" comercial(&&), o SQL*Plus preenche o valor da variável com o primeiro valor fornecido na execução da declaração SQL.

Exemplo:

```
SELECT      ENAME, DEPTNO, JOB
FROM        EMP
WHERE DEPTNO = &&SETOR_PLEASE;
```

Enter value for SETOR_please: 10

ENAME	DEPTNO	JOB
CLARK	10	MANAGER
KING	10	PRESIDENT

Você pode unicamente usar WHERE para restringir linhas individuais. Para restringir colunas de grupos usa-se a cláusula HAVING:

```
SELECT DEPTNO, AVG(SAL)
FROM   EMP
GROUP BY DEPTNO
HAVING AVG(SAL) > 2000;
```

SETOR	AVG(SAL)
10	2916.66667
20	2175

2.7.6 A Ordem das cláusulas na declaração SELECT.

SELECT	<i>coluna(s)</i>
FROM	<i>tabela(s)</i>
WHERE	<i>condição linha</i>
GROUP BY	<i>coluna(s)</i>
HAVING	<i>condição de grupo de linhas</i>
ORDER BY	<i>coluna(s);</i>

3 Executando Pesquisas Padrões com Variáveis Substituíveis

3.1 Única Variável Substituível

Você pode usar variáveis substituíveis para representar valores, em tempo de execução. Uma variável pode ser uma idéia de como um valor pode ser armazenado temporariamente. Uma variável é representada por um único "e" comercial(&), e o valor é atribuído na mesma.

A seguinte declaração apresenta ao usuário um número de departamento na execução:

```
SELECT      EMPNO, ENAME, SAL
FROM        EMP
WHERE DEPTNO = &DEP_NR;
```

Enter value for DEP_NR : 10

EMPNO	ENAME	SAL
7782	CLARK	2450
7839	KING	5000
7934	MILLER	1300

O exemplo acima usa a condição WHERE SETOR = 10

Com o único "e" comercial o usuário é solicitado toda vez que o comando é executado porque a variável não é definida e consequentemente o valor digitado não é salvo.

2.7.4.2 Funções de Grupo e Resultados Individuais

A seguinte declaração SQL retorna o máximo salário para cada grupo:

```
SELECT MAX(SAL), JOB
FROM   EMP
GROUP BY JOB;
```

MAX(SAL)	OCUP
3000	ANALYST
1300	CLERK
2975	MANAGER
5000	PRESIDENT
1600	SALESMAN

2.7.5 A cláusula HAVING

Use a cláusula HAVING se você quiser especificar qual grupo será mostrado.

Para mostrar a média salarial para todos os departamentos que tiverem mais de três empregados, faça:

```
SELECT      DEPTNO, AVG(SAL)
FROM        EMP
GROUP BY DEPTNO
HAVING      COUNT(1) > 3;
```

DEPTNO	AVG(SAL)
20	2175
30	1566.6667

Para mostrar só os cargos, onde o máximo salário é maior ou igual a \$3000, faça:

```
SELECT JOB, MAX(SAL)
FROM   EMP
HAVING MAX(SAL) >= 3000
GROUP BY JOB;
```

OCUP	MAX(SAL)
ANALYST	3000
PRESIDENT	5000

A cláusula HAVING deve preceder uma cláusula GROUP BY e é recomendado que seja colocada primeiro, pois é mais lógico.

A cláusula WHERE não pode ser usada para restringir itens de grupo.
A seguinte declaração da cláusula WHERE é errada.

```
SELECT DEPTNO, AVG(SAL)
FROM EMP
WHERE AVG(SAL) > 2000
GROUP BY DEPTNO;
```

ERROR at line 3: ORA-0934: set function is not allowed here

Para encontrar o mínimo salário ganho por um escriturário, faça:

```
SELECT MIN(SAL)
FROM EMP
WHERE JOB = 'CLERK';
```

MIN(SAL)

800

2.7.3 COUNT

Para encontrar o número de empregados do departamento 20, faça:

```
SELECT COUNT(*)
FROM EMP
WHERE DEPTNO = 20;
```

COUNT(*)

5

2.7.4 A cláusula GROUP BY

Pode ser usada para dividir as linhas de uma tabela em um grupo menor. Funções de grupo devem ser usadas para resumir informações por cada grupo.

Para calcular a média salarial de cada grupo de cargo, faça:

```
SELECT JOB, AVG(SAL)
FROM EMP
GROUP BY JOB;
```

JOB	AVG(SAL)
ANALYST	3000
CLERK	1037.5
MANAGER	2758.33333
PRESIDENT	5000
SALESMAN	1400

2.7.4.1 Grupos dentro de Grupos

Podemos então usar a cláusula GROUP BY para prover resultados para grupos dentro de grupos. Para mostrar a média salarial mensal faturado por cada cargo dentro de um departamento, faça:

```
SELECT DEPTNO, JOB, AVG(SAL) FROM EMP
GROUP BY DEPTNO, JOB;
```

DEPTNO	JOB	AVG(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	3000
20	CLERK	950
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	1400

2.6.5 VSIZE

VSIZE(col/valor) retorna o número de bytes interno do ORACLE representando um col/valor.

```
SELECT  DEPTNO, VSIZE(DEPTNO), VSIZE(HIREDATE), VSIZE(SAL), VSIZE(ENAME)
FROM    EMP
WHERE   DEPTNO = 10;
```

SETOR	VSIZE(SETOR)	VSIZE(DATA)	VSIZE(SAL)	VSIZE(NOME)
10	2	7	3	5
10	2	7	2	4
10	2	7	2	6

2.7 Funções de Grupo

Funções de grupo operam sobre conjuntos de linhas. Elas retornam resultados baseados sobre um grupo de linhas, antes que um resultado por linha tenha retornado como uma função de linha única. Como padrão todas as linhas de um tabela são trilhadas como um grupo. A clausula GROUP BY da declaração do SELECT é usada para agrupar as linhas em menores grupos.

As funções de grupos são listadas abaixo:

<u>Função</u>	<u>Valor Retornado</u>
AVG([DISTINCT/ALL]n)	Valor médio de n, ignorando os valores nulos.
COUNT([DISTINCT/ALL]expr*)	Contador * conta todas as linhas selecionadas, incluindo duplicadas e linhas nulas
MAX([DISTINCT/ALL]expr)	valor máximo da expressão
MIN([DISTINCT/ALL]expr)	valor mínimo da expressão
STDDEV([DISTINCT/ALL]n)	Desvio padrão de n, ignorando valores nulos.
SUM([DISTINCT/ALL]n)	Valor soma de n, ignorando valores nulos.
VARIANCE([DISTINCT/ALL],n)	variação de n, ignorando valores nulos.

Todas as funções acima operam sobre um número de linhas (por exemplo, uma tabela inteira) e são portanto funções de GRUPO.

DISTINCT faz uma função de grupo considerar valores não duplicados; ALL considera todos os valores.

Todas as funções de grupo exceto o COUNT(*) ignoram os valores nulos.

2.7.1 AVG

Para calcular a média salarial dos empregados, faça:

```
SELECT  AVG(SAL)
FROM    EMP;
```

AVG(SAL)

2073,2143

Note que as linhas da tabela EMP são trilhadas num único grupo.

2.7.2 MIN

Uma função de grupo pode ser usada para subconjunto de linhas de uma tabela usando a clausula WHERE.

1.1 Programação em Oracle8 PL/SQL

Esse exemplo ilustra que com a função DECODE, o valor retornado é forçado a ter um tipo de dado no terceiro argumento.

2.6.2 NVL

NVL(col/valor,valor) converte um valor nulo para um valor desejado. Tipo de dados devem combinar(col/valor e valor).

```
SELECT SAL*12+NVL(COMM,0), NVL(COMM,1000), SAL*12+NVL(COMM,1000)
FROM EMP
WHERE DEPTNO = 10;
```

SAL*12+NVL(COM,0)	NVL(COM,1000)	SAL*12+NVL(COM,1000)
29400	1000	30400
60000	1000	61000
15600	1000	16600

2.6.3 GREATEST

GREATEST(col/valor1,col/valor2,...) retorna o maior da lista de valores. Todos os col/valores são convertidos para um valor antes da comparação.

```
SELECT SAL, COMM, GREATEST(1000,2000), GREATEST(SAL,COMM)
FROM EMP
WHERE DEPTNO = 30;
```

SAL	COMM	GREATEST(1000,2000)	GREATEST(SAL,COMM)
1600	300	2000	1600
1250	500	2000	1250
1250	1400	2000	1400
2850		2000	
1500	0	2000	1500
950		2000	

Na função GREATEST quando na lista de valores existe um valor nulo ele é considerado como o maior.

2.6.4 LEAST

LEAST(col/valor1,col/valor2,...) retorna o menor valor de um lista de valores. Todos os valores são convertidos antes da comparação.

```
SELECT SAL, COMM, LEAST(1000,2000), LEAST(SAL,COMM)
FROM EMP
WHERE DEPTNO = 30;
```

SAL	COMM	LEAST(1000,2000)	LEAST(SAL,COMM)
1600	300	1000	300
1250	500	1000	500
1250	1400	1000	1250
2850		1000	
1500	0	1000	0
950		1000	

Na função LEAST quando na lista de valores existe um valor nulo ele é considerado como o menor.

2.6 Funções que Aceitam Vários Tipos de Entrada de Dados

2.6.1 DECODE

DECODE é a mais potente função do SQL. Ele facilita pesquisas condicionais fazendo o trabalho de 'ferramentas' ou comandos 'IF-THEN-ELSE'.

Sintaxe:

DECODE(col/expressão, procurado1,resultado1, procurado2,resultado2...,padrão)
--

Col/expressão é comparado com cada um dos valores procurados e retorna o resultado se a col/expressão é igual ao valor procurado. Se não for encontrado nenhum dos valores procurados, a função DECODE retorna o valor padrão. Se o valor padrão for omitido ele retornará um valor nulo.

O seguinte exemplo decodifica os cargos dos tipos MANAGER e CLERK unicamente. Os outros cargos serão padrão, alterados para UNDEFINED:

```
SELECT ENAME, JOB,  
       DECODE(JOB, 'CLERK', 'WORKER',  
               'MANAGER', 'BOSS')  
FROM   EMP;
```

ENAME	JOB	DECODE
SMITH	CLERK	WORKER
ALLEN	SALESMAN	
WARD	SALESMAN	
JONES	MANAGER	BOSS
MARTIN	SALESMAN	
BLAKE	MANAGER	BOSS
CLARK	MANAGER	BOSS
SCOTT	ANALYST	
KING	PRESIDENT	
TURNER	SALESMAN	
ADAMS	CLERK	WORKER
JAMES	CLERK	WORKER
FORD	ANALYST	
MILLER	CLERK	WORKER

Para mostrar a gratificação percentual dependendo do grau do salário:

```
SELECT GRADE,  
       DECODE(GRADE, '1', '15%',  
               '2', '10%',  
               '3', '8%',  
               '4', '5%') BONUS FROM SALGRADE;
```

GRADE	BONUS FROM SALGRADE
1	15%
2	10%
3	8%
4	5%
5	

2.5.3 TO_DATE

Para mostrar todos os empregados admitidos em 4 de junho de 1984 (não formato padrão), nós podemos usar a função TO_DATE:

```
SELECT EMPNO, ENAME, HIREDATE
FROM   EMP
WHERE  HIREDATE = TO_DATE ('September 8, 1981', 'Month dd, yyyy');
```

EMPNO	ENAME	HIREDATE
7844	TURNER	08-SEP-81

O conteúdo é convertido para data e comparado com o valor de DATA.

Para entrar um linha na tabela EMP com a data não no formato padrão:

```
INSERT INTO      EMP (EMPNO, DEPTNO, HIREDATE)
VALUES          (7777, 20, TO_DATE('19/08/90 00:00:00', 'DD/MM/YY HH:MI:SS'));
```

1.1 Programação em Oracle8 PL/SQL

WW ou W	Semana do ano ou mês
DDD ou DD ou D	dia do ano, mês ou semana
DAY	nome do dia, espaçado com brancos com 9 caracteres de tamanho
DY	nome do dia, 3 letras abreviadas
J	data Juliana, o número de dias desde 31 dezembro 4713 antes de Cristo
AM ou PM	Indicador meridiano
A.M. ou P.M.	indicador meridiano com períodos
HH ou HH12	horas do dia (1-12)
HH24	horas do dia (0-23)
MI	minuto
SS	segundos
SSSSS	segundos passado meia-noite(0-86399)
/,etc.	pontuação é reproduzida no resultado
"..."	cotas de linhas são representadas no resultado.

Os sufixos abaixo devem ser adicionados em frente dos códigos:

TH Ex.: SELECT TO_CHAR(SYSDATE,'DAY, DD TH MONTH YYYY') FROM DUAL; SEXTA-FEIRA , 01ST DEZEMBRO 2000
SP Ex.: SELECT TO_CHAR(SYSDATE,'DAY, DD SP MONTH YYYY') FROM DUAL; SEXTA-FEIRA , ONE DEZEMBRO 2000
SPTH ou thsp Ex.: SELECT TO_CHAR(SYSDATE,'DAY, DD SPTH MONTH YYYY') FROM DUAL; SEXTA-FEIRA , FIRST DEZEMBRO 2000

DAY	MONDAY
Day	Monday
Month	July
Ddth	14th
DdTh	14Th

2.5.2 TO_NUMBER

No seguinte exemplo a função TO_NUMBER é usada para transformar um número armazenado como um alfanumérico para um tipo numérico:

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE SAL > TO_NUMBER('1500');
```

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	1600
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7788	SCOTT	ANALYST	3000
7839	KING	PRESIDENT	5000
7902	FORD	ANALYST	3000

1.1 Programação em Oracle8 PL/SQL

Formatos Numéricos

Máscara	Significado	Exemplo	
9	posição numérica (número de 9s determinam a largura mostrada)	999999	1234
0	mostra zeros	0999999	001234
\$	mostra sinal de dólar	\$999999	\$1234
.	ponto decimal na posição especificada	999999.99	1234.00
,	vírgula na posição especificada	999,999	1,234
MI	sinal de menos à direita(valores negativos)	999999MI	1234-
PR	parênteses para números negativos	999999PR	<1234>
EEEE	notação científica(formato de conter quatro Es unicamente)	99.999EEEE	1.234E+03
V	multiplica pela décima potência 10n(n = número 9s depois da V)	9999V99	123400
B	mostra valores zero em branco, não zero	B9999.99	1234.00

```
SELECT SAL, TO_CHAR(SAL, '$9,999')  
FROM EMP;
```

SAL	TO_CHAR(SAL,'\$9,999')
800	\$800
1600	\$1,600
1250	\$1,250
2975	\$2,975
1250	\$1,250
2850	\$2,850
2450	\$2,450
3000	\$3,000
5000	\$5,000
1500	\$1,500
1100	\$1,100
950	\$950
3000	\$3,000
1300	\$1,300

Os formatos das máscaras são opcionais. Se a 'máscara' é omitida, a data é convertida para um alfanumérico que é padrão DD-MON-YY. Se a 'máscara' não é especificada, o número é convertido para alfanumérico.

Formatos de Data

Máscara

SCC ou CC
YYYY ou SYYYY
YYY ou YY ou Y
Y,YYY
SYEAR ou YEAR
BC ou AD
B.C. ou A.D.
Q
MM
MONTH
MON

Significado

Século, prefixo 'S' "BC" data com '-'
Ano, prefixo 'S' "BC" data com '-'
Último 3, 2 ou 1 dígito(s) do ano
Ano com vírgula nessa posição
Ano, soletrado na saída 'S' prefixo "BC" data com '-'
BC/AD período
BC/AD indicador com períodos
Um quarto do Ano
Mês
nome do mês, espaçamento com brancos do tamanho de 9 caracteres
nome do mês, 3 letras abreviadas

TRUNC é usado se você quiser remover o horário do dia. O horário contido no dia é removido por default.

2.5 Funções de Conversão

SQL possui um número de funções para controlar os tipos de conversão de dados. Essas funções convertem um valor de um tipo de dado para outro.

TO_CHAR(número,data,'formato')	converte números e datas para formatos alfanuméricos
TO_NUMBER(caracter)	converte alfanuméricos para numéricos.
TO_DATE('caracter','formato')	converte um alfanumérico representando uma data, para um valor de data de acordo com o formato especificado. Se o formato é omitido o formato padrão é 'DD-MON-YY'.

2.5.1 TO_CHAR

TO_CHAR(data,'máscara') Especifica que a data está sendo convertida para um novo formato na saída.

Para converter a data corrente do formato padrão (DD-MON-YY) para uma nova máscara:

```
SELECT TO_CHAR(SYSDATE, 'DAY, DD MONTH YYYY' )
FROM    DUAL;
```

```
TO_CHAR(SYSDATE,'DAY, DD MONTH YYYY')
```

```
-----
SEXTA-FEIRA , 01 DEZEMBRO 2000
```

Note que:

- A 'máscara' deve estar entre aspas simples e pode ser incluída em vários formatos. A coluna e 'máscara' devem ser separadas por uma vírgula.
- DAY e MONTH na saída são espaçados automaticamente com brancos no tamanho de 9 caracteres

TO_CHAR pode também ser usado para extrair o horário de um único dia, e mostrá-lo no formato especificado.

```
SELECT TO_CHAR(SYSDATE, 'HH:MI:SS' )
FROM    DUAL;
```

```
TO_CHAR(SYSDATE,'HH:MI:SS')
```

```
-----
08:16:24
```

A função TO_CHAR é também usada para converter um valor do tipo numérico para um valor do tipo alfanumérico.

1.1 Programação em Oracle8 PL/SQL

```
SELECT HIREDATE, TO_CHAR(HIREDATE, 'DAY'), NEXT_DAY(HIREDATE, 5)
FROM EMP
WHERE DEPTNO = 20;
```

HIREDATE	TO_CHAR(HIREDATE,'DAY')	NEXT_DAY(HIREDATE,5)
17/12/80	QUARTA-FEIRA	18/12/80
02/04/81	QUINTA-FEIRA	09/04/81
19/04/87	DOMINGO	23/04/87
23/05/87	SÁBADO	28/05/87
03/12/81	QUINTA-FEIRA	10/12/81

2.4.6 LAST_DAY

LAST_DAY(data) encontra a data do ultimo dia do mês da data especificada

```
SELECT SYSDATE, LAST_DAY(SYSDATE), HIREDATE,
       LAST_DAY(HIREDATE), LAST_DAY('15-02-88')
FROM EMP
WHERE DEPTNO = 20;
```

SYSDATE	LAST_DAY(SYSDATE)	HIREDATE	LAST_DAY(HIREDATE)	LAST_DAY('15-02-88')
01/12/00	31/12/00	17/12/80	31/12/80	29/02/88
01/12/00	31/12/00	02/04/81	30/04/81	29/02/88
01/12/00	31/12/00	19/04/87	30/04/87	29/02/88
01/12/00	31/12/00	23/05/87	31/05/87	29/02/88
01/12/00	31/12/00	03/12/81	31/12/81	29/02/88

2.4.7 ROUND

A função **ROUND** pode ser aplicada para datas.

ROUND(data) retorna a data com o horário em 12:00(meio-dia) Usamos isso quando comparamos datas que tenham diferentes horários.

ROUND(data,'MONTH') retorna o primeiro dia do mês da data, se a data estiver na primeira metade do mês; se não retorna o primeiro do mês seguinte.

ROUND(data,'YEAR') retorna o primeiro dia do ano da data se data estiver na primeira metade do ano; se não retorna o primeiro do ano seguinte.

```
SELECT SYSDATE, ROUND(SYSDATE, 'MONTH'), ROUND(SYSDATE, 'YEAR')
FROM DUAL;
```

SYSDATE	ROUND(SYSDATE,'MONTH')	ROUND(SYSDATE,'YEAR')
04-DEC-89	01-DEC-89	01-JAN-90

2.4.8 TRUNC

TRUNC(data,'caracter') encontra a data do primeiro dia do mês quando caracter = 'MONTH'. Se o caracter = 'YEAR' ele encontra o primeiro dia do ano.

```
SELECT SYSDATE, TRUNC(SYSDATE, 'MONTH'), TRUNC(SYSDATE, 'YEAR')
FROM DUAL;
```

SYSDATE	TRUNC(SYSDATE,'MONTH')	TRUNC(SYSDATE,'YEAR')
04-DEC-89	01-DEC-89	01-JAN-89

1.1 Programação em Oracle8 PL/SQL

HIREDATE	HIREDATE +7	HIREDATE -7	SYSDATE- HIREDATE
17/12/80	24/12/80	10/12/80	7289,7189
03/12/81	10/12/81	26/11/81	6938,7189
03/12/81	10/12/81	26/11/81	6938,7189

2.4.3 MONTHS_BEETWEEN

MONTHS_BETWEEN(data1,data2)

encontra o número de meses entre data 1 e data2. O resultado pode ser positivo ou negativo. Se a data 1 for posterior a data2, então o resultado será positivo, se a data 1 for menor que a data 2 o resultado será negativo.

```
SELECT  MONTHS_BETWEEN(SYSDATE, HIREDATE) ,  
        MONTHS_BETWEEN('01-01-84', '05-11-88')  
FROM      EMP  
WHERE     MONTHS_BETWEEN(SYSDATE, HIREDATE) > 59;
```

MONTHS_BETWEEN(SYSDATE,DATA)	MONTHS_BETWEEN('01-JAN-84','05-NOV-88')
239,50724	-58,12903
237,41047	-58,12903
237,34595	-58,12903
235,99111	-58,12903
230,1524	-58,12903
235	-58,12903
233,76531	-58,12903
163,44273	-58,12903
228,50724	-58,12903
230,79756	-58,12903
162,31369	-58,12903
227,95885	-58,12903
227,95885	-58,12903
226,31369	-58,12903

A parte não inteira do resultado representa uma parcela do mês.

2.4.4 ADD_MONTHS

ADD_MONTHS(data,n) adiciona n números de meses na data; n deve ser inteiro e pode ser negativo.

```
SELECT  HIREDATE, ADD_MONTHS(HIREDATE, 3), ADD_MONTHS(HIREDATE, -3)  
FROM      EMP  
WHERE     DEPTNO = 20;
```

HIREDATE	ADD_MONTHS(HIREDATE,3)	ADD_MOSTHS(HIREDATE,-3)
17/12/80	17/03/81	17/09/80
02/04/81	02/07/81	02/01/81
19/04/87	19/07/87	19/01/87
23/05/87	23/08/87	23/02/87
03/12/81	03/03/82	03/09/81

2.4.5 NEXT_DAY

NEXT_DAY(data1,'caracter') data do próximo dia especificado da semana. Caracter deve ser um número representado um dia, ou o dia da semana descrito em inglês.

2.4 Funções de Data

Funções de data operam sobre datas do ORACLE. Todas as funções de datas retornam valores de tipo data exceto MONTHS_BETWEEN o qual retorna um valor numérico.

Armazenamento de Datas no ORACLE

- Século
- Ano
- Mês
- Dia
- Horas
- Minutos
- Segundos

O padrão de data mostrados nas pesquisas é DD-MON-YY.

2.4.1 Sysdate

Sysdate é uma coluna que retorna a data e horário corrente. Você pode usar o SYSDATE como uma outra coluna qualquer. Por exemplo, você pode mostrar data corrente selecionando o sysdate de uma tabela simulada chamada DUAL. A tabela DUAL é uma tabela do sistema e deve ser permitido acessá-la para todos os usuários. Ela contém uma coluna DUMMY e uma linha com o valor X. A tabela DUAL é usada quando você quer retornar apenas uma linha.

Para mostrar a data corrente:

```
SELECT SYSDATE FROM DUAL;
```

SYSDATE

01/12/00

Você poderia facilmente selecionar o sysdate da tabela EMP, mas seriam retornados tantos sysdate quanto o número de linhas na tabela EMP.

2.4.2 Usando Operadores Aritméticos

Devido ao fato das datas serem armazenadas como número, é possível fazer cálculos com elas usando operadores aritméticos tal como adição e subtração. Você pode adicionar e subtrair números bem como data.

As operações que você pode realizar são:

data + número	Adicionando um número de dias em uma data, produzindo uma nova data
data – número	subtraindo um número de dias de uma data, produzindo uma nova data
data – data	subtraindo uma data de outra, produzindo um número de dias
data+número/24	adicionando um número de horas em uma data produzindo um nova data

Exemplo:

```
SELECT HIREDATE, HIREDATE +7, HIREDATE -7, SYSDATE - HIREDATE  
FROM EMP  
WHERE HIREDATE LIKE '%12%';
```

2.3.7 SIGN

SIGN(col/value) retorna -1 se a coluna, expressão ou valor for negativo ou zero e 1 se for positivo.

```
SELECT SAL-COMM, SIGN(SAL-COMM), COMM-SAL, SIGN(COMM-SAL)
FROM EMP
WHERE DEPTNO = 30;
```

SAL-COM	SIGN(SAL-COM)	COM-SAL	SIGN(SAL-COM)
1300	1	-1300	-1
750	1	-750	-1
-150	-1	150	1
1500	1	-1500	-1

Freqüentemente a função SIGN é usada para testar se um valor é menor, maior ou igual a um segundo valor. O seguinte exemplo apresenta todos os empregados os quais o salário é maior que sua comissão:

```
SELECT ENAME, SAL, COMM
FROM EMP
WHERE SIGN(SAL-COMM) = 1;
```

NOME	SAL	COM
ALLEN	1600	300
WARD	1250	500
TURNER	1500	0

2.3.8 ABS

ABS(col/value) encontra o valor absoluto de um coluna, expressão ou valor.

```
SELECT SAL, COMM, COMM-SAL, ABS(COMM-SAL), ABS(-35)
FROM EMP
WHERE DEPTNO = 30;
```

SAL	COM	COM-SAL	ABS(COMM-SAL)	ABS(-35)
1600	300	-1300	1300	35
1250	500	-750	750	35
1250	1400	150	150	35
2850				35
1500	0	-1500	1500	35
950				35

2.3.9 MOD

MOD(val1,val2) encontra o resto da divisão val1 por val2

```
SELECT SAL, COMM, MOD(SAL,COMM), MOD(100,40)
FROM EMP
WHERE DEPTNO = 30
ORDER BY COMM;
```

SAL	COM	MOD(SAL,COM)	MOD(100,40)
1500	0	1500	20
1600	300	100	20
1250	500	250	20
1250	1400	1250	20
2850			20
950			20

2.3.3 CEIL

CEIL(col/value) Arredonda acima.

Exemplo:

```
SELECT CEIL(SAL), CEIL(99.9), CEIL(101.76), CEIL(-11.1)
FROM   EMP
WHERE  SAL BETWEEN 3000 AND 5000;
```

CEIL(SAL)	CEIL(99.9)	CEIL(101.76)	CEIL(-11.1)
3000	100	102	-11
5000	100	102	-11
3000	100	102	-11

2.3.4 FLOOR

FLOOR(col/value) Arredonda para baixo.

```
SELECT FLOOR(SAL), FLOOR(99.9), FLOOR(101.76), FLOOR(-11.1)
FROM   EMP
WHERE  SAL BETWEEN 3000 AND 5000;
```

FLOOR(SAL)	FLOOR(99.9)	FLOOR(101.76)	FLOOR(-11.1)
3000	99	101	-12
5000	99	101	-12
5000	99	101	-12

2.3.5 POWER

POWER(col/value,n) eleva uma coluna, expressão ou valor para uma potência; n pode ser negativo mas deve ser um número, se não um erro será retornado

```
SELECT SAL, POWER(SAL,2), POWER(SAL,3), POWER(50,5)
FROM   EMP
WHERE  DEPTNO = 10;
```

SAL	POWER(SAL,2)	POWER(SAL,3)	POWER(50,5)
2450	6002500	1,471E+10	312500000
5000	25000000	1,250E+11	312500000
1300	1690000	2,197E+09	312500000

2.3.6 SQRT

SQRT(col/value) encontra a raiz quadrada da coluna ou valor. Se a coluna ou valor for menor que zero será retornado nulo.

```
SELECT SAL, SQRT(SAL), SQRT(40), SQRT(COMM)
FROM   EMP
WHERE  COMM > 0;
```

SAL	SQRT(SAL)	SQRT(40)	SQRT(COMM)
1600	40	6,3245553	17,320508
1250	35,355339	6,3245553	22,36068
1250	35,355339	6,3245553	37,416574

CLERK ANALYST CLERK	CLERK ANALYST CLERK
---------------------------	---------------------------

A Função REPLACE é um complemento da função TRANSLATE que substitui caracteres um a um e o REPLACE substitui um linha por outra.

2.2 Aninhamento de Funções

Funções de linhas únicas podem ser aninhadas para várias finalidades. Se funções são aninhadas, elas são avaliadas de dentro para fora.

Exemplo:

X(D(A(B(C(caracter))))) ordem lógica de execução "C,B,A,D e X."

2.3 Funções Numéricas

As funções aceitam entrada de números e retornam valores numéricos.

2.3.1 ROUND

ROUND(col/value,n) arredonda uma coluna, expressão ou valor para n casas decimais. Se n for negativo, os números para esquerda do decimal são arredondados.

```
SELECT      ROUND(45.923,1),  
            ROUND(45.923),  
            ROUND(45.323,1),  
            ROUND(45.323,-1),  
            ROUND(SAL/32,2)  
FROM        EMP  
WHERE       DEPTNO = 10;
```

ROUND(45.923,1)	ROUND(45.923)	ROUND(45.323,1)	ROUND(45.323,-1)	ROUND(SAL/32,2)
45.9	46	45.3	50	76.56
45.9	46	45.3	50	156.25
45.9	46	45.3	50	40.63

2.3.2 TRUNC

TRUNC(col/value,n) trunca a coluna, expressão ou valor para n casas decimais. Se n é negativo os números para esquerda das casas decimais são truncados para zero.

```
SELECT      TRUNC(45.923,1),  
            TRUNC(45.923),  
            TRUNC(45.323,1),  
            TRUNC(45.323,-1),  
            TRUNC(SAL/32,2)  
FROM        EMP  
WHERE       DEPTNO = 10;
```

TRUNC(45.923,1)	TRUNC(45.923)	TRUNC(45.323,1)	TRUNC(45.323,-1)	TRUNC(SAL/32,2)
45.9	45	45.3	40	76.56
45.9	45	45.3	40	156.25
45.9	45	45.3	40	40.62

NOME SOUNDEX(NOME)

FORD F630

2.1.8 LENGTH

LENGTH(col/value) retorna o número de caracteres na coluna ou valor literal.

```
SELECT LENGTH('CURSO SQL'), LENGTH(DEPTNO), LENGTH(DNAME)
FROM DEPT;
```

LENGTH('CURSO SQL')	LENGTH(DEPTNO)	LENGTH(DNAME)
9	2	10
9	2	8
9	2	5
9	2	10

Note que a função INSTR, LENGTH retornam um valor numérico.

2.1.9 TRANSLATE e REPLACE

As funções TRANSLATE e REPLACE são usadas para substituir caracteres.

TRANSLATE(col/value,from,to) Faz a substituição de um ou mais caracteres por outros.

Exemplo:

```
SELECT ENAME, TRANSLATE(ENAME,'C','P'), JOB,
       TRANSLATE(JOB,'AR','IT')
```

FROM EMP

WHERE DEPTNO = 10;

ENAME	TRANSLATE(ENAME,'C','P')	JOB	TRANSLATE(JOB,'AR','IT')
CLARK	PLARK	MANAGER	MINIGET
KING	KING	PRESIDENT	PTESIDENT
MILLER	MILLER	CLERK	CLETK

REPLACE(col/value,linha,linha_alterada)

Retorna o valor da coluna com toda a ocorrência da linha de alteração. Se a linha alterada for omitida toda a linha especificada será removida.

```
SELECT JOB, REPLACE(JOB,'SALESMAN','SALESPERSON')
FROM     EMP;
```

JOB, REPLACE(JOB,'SALESMAN','SALESPERSON')
CLERK
SALESMAN
SALESMAN
MANAGER
SALESMAN
MANAGER
MANAGER
MANAGER
ANALYST
PRESIDENT
SALESMAN
CLERK

2.1.5 INSTR

`INSTR(col/value,'character')` encontra a primeira ocorrência do 'character'.

`INSTR(col/value,'character',pos,n)` encontra a posição do character na coluna ou valor literal iniciando na posição que se encontra dentro de uma repetição desta variável ou literal.

```
SELECT DNAME, INSTR(DNAME,'A'), INSTR(DNAME,'ES'), INSTR(DNAME,'C',1,2)
FROM   DEPT;
```

DNAME	INSTR(DNAME,'A')	INSTR(DNAME,'ES')	INSTR(DNAME,'C',1,2)
ACCOUNTING	1	0	3
RESEARCH	5	2	0
SALES	2	4	0
OPERATIONS	5	0	0

2.1.6 LTRIM e RTRIM

Removem específicos caracteres de um linha.

`LTRIM(col/value,'caractere(s)')` Se o character não é especificado cortará os brancos da esquerda.

Exemplo:

```
SELECT DNAME, LTRIM(DNAME,'A'), LTRIM(DNAME,'AS'), LTRIM(DNAME,'ASOP')
FROM   DEPT;
```

DNAME	LTRIM(DNAME,'A')	LTRIM(DNAME,'AS')	LTRIM(DNAME,'ASOP')
ACCOUNTING	ACCOUNTING	ACCOUNTING	ACCOUNTING
RESEARCH	RESEARCH	RESEARCH	RESEARCH
SALES	SALES	LES	LES
OPERATIONS	OPERATIONS	OPERATIONS	ERATIONS

`RTRIM(col/value,'caractere(s)')` Se os caracteres não forem especificados serão removidos os brancos.

```
SELECT DNAME, RTRIM(DNAME,'G'), RTRIM(DNAME,'GHS'), RTRIM(DNAME,'N')
FROM   DEPT;
```

DNAME	RTRIM(DNAME,'G')	RTRIM(DNAME,'GHS')	RTRIM(DNAME,'N')
ACCOUNTING	ACCOUNTIN	ACCOUNTIN	ACCOUNTING
RESEARCH	RESEARCH	RESEARC	RESEARCH
SALES	SALES	SALE	SALES
OPERATIONS	OPERATIONS	OPERATION	OPERATIONS

2.1.7 SOUNDEX

`SOUNDEX(col/value)` retorna uma linha de caracteres representando o som da palavra para uma coluna ou um valor literal. Esta função retorna a fonética representação de uma palavra.

```
SELECT ENAME, SOUNDEX(ENAME)
FROM   EMP
WHERE  SOUNDEX(ENAME) = SOUNDEX('FRED');
```

2.1.4 LPAD e RPAD

LPAD(col/value,n,'character') Preenche a coluna ou valor literal da esquerda para o total tamanho de n posições. Os principais espaços estão preenchidos com o 'character'. Se o character for omitido o valor padrão é espaços.

```
SELECT LPAD(DNAME,20,'*'), LPAD(DNAME,20), LPAD(DEPTNO,20,'.')
```

```
FROM DEPT;
```

LPAD(DNSMR,20,'*')	LPAD(DNAME,20)	LPAD(DEPTNO,20,'.')
*****RESEACH	RESEACH20
*****SALES	SALES30
*****OPERATIONS	OPERATIONS40
*****ACCOUNTING	ACCOUNTING10

RPAD(col/value,n,'character') preenche a coluna ou valor literal da direita para o total tamanho de n posições. Os espaços a direita são preenchidos com o 'character'. Se o 'character' for omitido o preenchimento fica em branco.

Exemplo:

```
SELECT RPAD(DNAME,20,'*'), RPAD(DNAME,20), RPAD(DEPTNO,20,'.')
```

```
FROM DEPT;
```

RPAD(DNAME,20,'*')	RPAD(DNAME,20)	RPAD(DEPTNO,20,'.')
ACCOUNTING*****	ACCOUNTING	10.....
RESEACH*****	RESEACH	20.....
SALES*****	SALES	30.....
OPERATIONS*****	OPERATIONS	40.....

A segunda coluna é alinhada para a direita com brancos por padrão.

2.1.4 SUBSTR

SUBSTR(col/value,pos,n) Retorna uma linha de n caracteres da coluna ou valor literal, iniciando na posição número pos. Se n é omitido a linha é extraída da posição pos até o fim.

Exemplo:

```
SELECT SUBSTR('ORACLE',2,4), SUBSTR(DNAME,2), SUBSTR(DNAME,3,5)
```

```
FROM DEPT;
```

SUBSTR('ORACLE',2,4)	SUBSTR(DNAME,2)	SUBSTR(DNAME,3,5)
RACL	CCOUNTING	COUNT
RACL	ESEARCH	SEARC
RACL	ALES	LES
RACL	PERATIONS	ERATI

Note que os valores estão alinhados para a esquerda. Isso porque SQL*Plus sempre mostra dados alfanuméricos alinhados para a esquerda por default.

2.1 Funções Alfanuméricas

Funções Alfanuméricas aceitam dados alfanuméricos e podem retornar valores alfanuméricos ou numéricos.

A função seguinte influencia na construção de valores alfanuméricos.

2.1.1 LOWER

LOWER(col/value) fornece valores alfanuméricos os quais estão em letra maiúscula ou minúscula e retornam em letra minúscula

Para mostrar o nome dos departamentos em letra minúscula e a constante CURSO_SQL, faça:

```
SELECT LOWER(DEPTNO), LOWER('CURSO_SQL')  
FROM DEPT;
```

LOWER(DEPTNO)	LOWER('CURSO_SQL')
10	curso_sql
20	curso_sql
30	curso_sql
40	curso_sql

2.1.2 UPPER

UPPER(col/value) fornece caracteres alfanuméricos, os quais estão em letra maiúscula ou minúscula e retorna em letra maiúscula.

No exemplo seguinte, a função UPPER força o usuário entrar em letra maiúscula.

```
SELECT ENAME  
FROM EMP  
WHERE ENAME = UPPER('&NOME');
```

Enter value for NOME : smith

ENAME

SMITH

2.1.3 INITCAP

INITCAP(col/value) força a primeira letra da palavras ser em maiúscula e o resto minúscula

Exemplo:

```
SELECT INITCAP(DNAME), INITCAP(LOC)  
FROM DEPT;
```

INITCAP(DNAME)	INITCAP(LOC)
Accounting	New York
Research	Dallas
Sales	Chicago
Operations	Boston

Suponha que você queira encontrar todos os gerentes, em vários departamentos, e todos os escriturários no departamento 10 unicamente:

```
SELECT *  
FROM EMP  
WHERE JOB = 'MANAGER'  
OR (JOB = 'CLERK' AND DEPTNO = 10);
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02/04/81	2975		20
7698	BLAKE	MANAGER	7839	01/05/81	2850		30
7782	CLARK	MANAGER	7839	09/06/81	2450		10
7934	MILLER	CLERK	7782	23/01/82	1300		10

O parênteses acima não é necessário, por que o AND é precedente ao OR, mas ele esclarece o significado da expressão.

2 Funções

As Funções são usadas para manipular dados. Elas aceitam um ou mais argumentos e retornam um valor. Um argumento é uma constante, refere-se à variável, expressão ou coluna.

Sintaxe:

<code>função_nome (argumento1, argumento2, ...)</code>
--

Funções podem ser usadas para:

- Cálculos sobre datas
- modificar valores de itens individuais
- manipular saída para grupos de linhas
- alterar formatos de datas para mostrá-los

Existem diferentes tipos de funções:

- ALFANUMÉRICAS
- NUMÉRICAS
- DATA
- CONVERSÃO
- FUNÇÕES QUE ACEITAM VÁRIOS TIPOS DE DADOS
- GRUPO

Algumas funções operam unicamente sobre linhas simples; outras manipulam grupos de linhas.

Funções de Linha Única:

- agem sobre cada linha retornada na pesquisa
- retornam um resultado por linha
- esperam um ou mais argumento do usuário
- podem ser aninhadas
- podem ser usadas com variáveis do usuário, colunas, expressões; podem ser usadas, por exemplo, nas cláusulas SELECT, WHERE, ORDER BY.

1.1 Programação em Oracle8 PL/SQL

Para encontrar todos os empregados que são escriturários ou todos que ganhem entre 1000 e 2000 faça:

```
SELECT      EMPNO, ENAME, JOB, SAL
FROM        EMP
WHERE SAL BETWEEN 1000 AND 2000
OR          JOB = 'CLERK';
```

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	800
7499	ALLEN	SALESMAN	1600
7521	WARD	SALESMAN	1250
7654	MARTIN	SALESMAN	1250
7844	TURNER	SALESMAN	1500
7876	ADAMS	CLERK	1100
7900	JAMES	CLERK	950
7934	MILLER	CLERK	1300

Você pode combinar AND e OR na mesma expressão lógica. Quando AND e OR aparecerem na mesma cláusula WHERE, o AND é realizado primeiro depois o OR.

Se AND não interfere sobre o OR a seguinte declaração SQL retornará todos os gerentes com salário acima de 1500 e todos os vendedores:

```
SELECT      EMPNO, ENAME, JOB, SAL, DEPTNO
FROM        EMP
WHERE      SAL > 1500
AND        JOB = 'MANAGER'
OR         JOB = 'SALESMAN';
```

EMPNO	ENAME	JOB	SAL	DEPTNO
7499	ALLEN	SALESMAN	1600	30
7521	WARD	SALESMAN	1250	30
7566	JONES	MANAGER	2975	20
7654	MARTIN	SALESMAN	1250	30
7698	BLAKE	MANAGER	2850	30
7782	CLARK	MANAGER	2450	10
7844	TURNER	SALESMAN	1500	30

Todos os operadores são organizados em uma hierarquia. Numa Expressão, as operações são feitas na ordem de sua precedência, do maior para o menor. Onde os operadores de igual precedentes são usados próximo a outro, eles são feitos da esquerda para direita.

- 1- Todos os comparativos e Operadores SQL tem igual precedente:
=, !=, <, >, <=, >=, BETWEEN ... AND ..., IN, LIKE, IS NULL.
- 2- NOT (para inverter o resultado das expressões lógicas: WHERE NOT (SAL>2000))
- 3- AND
- 4- OR

Sempre que você estiver em dúvida sobre qual dos dois operadores será feito primeiro quando a expressão é avaliada, use parênteses aumentar a legibilidade.

FORD	ANALYST
MILLER	CLERK

Para encontrar todos os empregados que tenham um gerente, faça:

```
SELECT      ENAME, MGR
FROM        EMP
WHERE MGR IS NOT NULL;
```

ENAME	MGR
SMITH	7902
ALLEN	7698
WARD	7698
JONES	7839
MARTIN	7698
BLAKE	7839
CLARK	7839
SCOTT	7566
TURNER	7698
ADAMS	7788
JAMES	7698
FORD	7566
MILLER	7782

Nota:

Se um valor nulo é usado em uma comparação, então o operador de comparação deve ser IS ou IS NOT NULL. Se esses operadores não forem usados e valores nulos forem comparados, o resultado será sempre FALSO.

Por exemplo, COMM <> NULL será sempre FALSO. O resultado será falso porque um valor nulo não pode ser igual ou não igual a qualquer outro valor.

1.16 Pesquisando Dados com Múltiplas Condições

Os operadores AND e OR devem ser usados para fazer composições de expressões lógicas.

O predicado AND espera que ambas as condições sejam verdadeiras, enquanto que o predicado OR espera que uma das condições seja verdadeira.

Nos seguintes exemplos as condições são as mesmas, o predicado é diferente. Veja como o resultado é dramaticamente alterado.

Para encontrar todos os escriturários que ganhem entre 1000 e 2000, faça:

```
SELECT      EMPNO, ENAME, JOB, SAL
FROM        EMP
WHERE SAL BETWEEN 1000 AND 2000
AND JOB = 'CLERK';
```

COD	ENAME	JOB	SAL
-----	-----	-----	-----
7876	ADAMS	CLERK	1,100.00
7934	MILLER	CLERK	1,300.00

1.14.4 Operador IS NULL

Para encontrar unicamente todos os empregados que não tenham gerente, você testará um valor nulo:

```
SELECT      ENAME, MGR
FROM        EMP
WHERE MGR IS NULL;
```

```
ENAME      MGR
-----
KING
```

1.15 Expressões Negativas

Os operadores seguintes são testes de negação:

<>	diferente (todos sistemas operacionais)
NOT BETWEEN	tudo que estiver fora da faixa
NOT IN	tudo que não esteja na lista
NOT LIKE	tudo que não contenha a linha de caracteres
IS NOT NULL	tudo que não for nulo

Para encontrar empregados que tenham o salário fora da faixa, faça:

```
SELECT      ENAME, SAL
FROM        EMP
WHERE SAL NOT BETWEEN 1000 AND 2000;
```

ENAME	SAL
SMITH	800
JONES	2975
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
JAMES	950
FORD	3000

Para encontrar os empregados os quais o cargo não comece com a letra M, faça:

```
SELECT      ENAME, JOB
FROM        EMP
WHERE JOB NOT LIKE 'M%';
```

ENAME	JOB
SMITH	CLERK
ALLEN	SALESMAN
WARD	SALESMAN
MARTIN	SALESMAN
SCOTT	ANALYST
KING	PRESIDEN
TURNER	SALESMAN
ADAMS	CLERK
JAMES	CLERK

MILLER	1300
--------	------

1.14.2 O Operador IN

Testa os valores especificados em uma lista.

Para encontrar empregados que tenham um dos três números de MAT, faça:

```
SELECT      EMPNO, ENAME, SAL, MGR
FROM        EMP
WHERE MGR IN (7902,7566,7788)
```

EMPNO	ENAME	SAL	MGR
7369	SMITH	800	7902
7788	SCOTT	3000	7566
7876	ADAMS	1100	7788
7902	FORD	3000	7566

Podemos usar o comando select dentro da cláusula in.

1.14.3 O Operador LIKE

Algumas vezes você precisa procurar valores que não conhece exatamente. Usando o operador LIKE é possível selecionar linhas combinando parâmetros alfanuméricos. Dois símbolos podem ser usados para construir uma linha de procura.

Símbolo:	
%	Várias seqüência de zero ou mais caracteres
_	um número desejado de caracteres

Para listar todos os empregados os quais o nome começa com a letra S, faça:

```
SELECT      ENAME
FROM        EMP
WHERE ENAME LIKE 'S%';
```

ENAME

SMITH
SCOTT

Eles podem ser usados para encontrar um determinado número de caracteres.

Por exemplo para listar todos empregados que tenham exatamente quatro caracteres de tamanho do nome:

```
SELECT      ENAME
FROM        EMP
WHERE ENAME LIKE '____';
```

ENAME

WARD
KING
FORD

O % e o _ podem ser usados em várias combinações com literais alfanuméricos.

Para listar os nomes, números, emprego e departamentos de todos os escriturários(CLERK):

```
SELECT      ENAME, EMPNO, JOB, DEPTNO
FROM        EMP
WHERE JOB = 'CLERK';
```

ENAME	EMPNO	JOB	DEPTNO
SMITH	7369	CLERK	20
ADAMS	7876	CLERK	20
JAMES	7900	CLERK	30
MILLER	7934	CLERK	10

Para encontrar todos os nomes de departamentos com número maior que 20, faça:

```
SELECT      DEPTNO, DNAME
FROM        DEPT
WHERE DEPTNO > 20;
```

DEPTNO	DNAME
30	SALES
40	OPERATIONS

Comparando uma coluna com outra coluna na mesma linha:

Exemplo:

```
SELECT      ENAME, SAL, COMM
FROM        EMP
WHERE COMM > SAL;
```

ENAME	SAL	COMM
-----	-----	-----
MARTIN	1250	1400

1.14 Operadores SQL

Existem quatro operadores SQL que operam com todos os tipos de dados:

BETWEEN ... AND ...	Entre dois valores (inclusive)
IN(Lista)	Compara uma lista de valores
LIKE	Compara um parâmetro alfanumérico
IS NULL	É um valor nulo

1.14.1 O Operador BETWEEN

Testa um faixa de valores inclusive.

Exemplo:

```
SELECT      ENAME, SAL
FROM        EMP
WHERE SAL BETWEEN 1000 AND 2000;
```

ENAME	SAL
ALLEN	1600
WARD	1250
MARTIN	1250
TURNER	1500
ADAMS	1100

Ordenação decedente

Para inverter essa ordem, use o comando DESC(Decrescente) depois do nome das colunas da clausula ORDER BY.

Exemplo:

```
SELECT      ENAME, JOB, HIREDATE DATA DESC
FROM        EMP
ORDER BY    HIREDATE DESC;
```

ENAME	JOB	DATA
ADAMS	CLERK	23/05/87
SCOTT	ANALYST	19/04/87
MILLER	CLERK	23/01/82
JAMES	CLERK	03/12/81
FORD	ANALYST	03/12/81
KING	PRESIDENT	17/11/81
MARTIN	SALESMAN	28/09/81
TURNER	SALESMAN	08/09/81
CLARK	MANAGER	09/06/81
BLAKE	MANAGER	01/05/81
JONES	MANAGER	02/04/81
WARD	SALESMAN	22/02/81
ALLEN	SALESMAN	20/02/81
SMITH	CLERK	17/12/80

Ordenação por várias colunas.

É possível na clausula ORDER BY usar mais de uma coluna. O limite de colunas é o número de colunas da tabela. Se algumas ou todas são invertidas especifique DESC depois de alguma ou cada uma das colunas.

A clausula ORDER BY é usada na pesquisa. O comando não altera a ordem dos dados que estão armazenados no Banco de Dados ORACLE.

1.13 A Clausula WHERE

A clausula WHERE corresponde aos Operadores de Restrições da Álgebra Relacional.

Sintaxe:

SELECT	coluna(s)
FROM	tabela(s)
WHERE	certa condição a ser encontrada

A clausula WHERE pode comparar valores em uma coluna, valores literais, expressões aritméticas ou funções.

Operadores de comparação são usados na clausula WHERE e podem ser divididos em duas categorias: Lógicos e SQL.

Operadores Lógicos

Esses operadores testam as seguintes condições:

=	igual a
>	maior que
>=	maior e igual a
<	menor que
<=	menor e igual a

Alfanuméricos e Datas na clausula WHERE devem estar entre aspas simples.

Várias colunas podem ser especificadas depois do DISTINCT

Exemplo:

```
SELECT DISTINCT DEPTNO, JOB  
FROM EMP;
```

DEPTNO JOB

```
-----  
10  CLERK  
10  MANAGER  
10  PRESIDENT  
20  ANALYST  
20  CLERK  
20  MANAGER  
30  CLERK  
30  MANAGER  
30  SALESMAN
```

1.12 A cláusula ORDER BY

A cláusula ORDER BY pode ser usada para ordenar as linhas. Se usado, o ORDER BY deve ser a última cláusula da declaração SELECT.

Para ordenar pelo NOME, faça:

Exemplo:

```
SELECT ENAME, JOB, SAL*12, DEPTNO  
FROM EMP  
ORDER BY ENAME;
```

ENAME	JOB	SAL*12	DEPTNO
ADAMS	CLERK	13200	20
ALLEN	SALESMAN	19200	30
BLAKE	MANAGER	34200	30
CLARK	MANAGER	29400	10
FORD	ANALYST	36000	20
JAMES	CLERK	11400	30
JONES	MANAGER	35700	20
KING	PRESIDENT	60000	10
MARTIN	SALESMAN	15000	30
MILLER	CLERK	15600	10
SCOTT	ANALYST	36000	20
SMITH	CLERK	9600	20
TURNER	SALESMAN	18000	30
WARD	SALESMAN	15000	30

ENAME	GANHO_ANUAL
SMITH	
ALLEN	19500
WARD	15500
JONES	
MARTIN	16400
BLAKE	
CLARK	
SCOTT	
KING	
TURNER	18000
ADAMS	
JAMES	
FORD	
MILLER	

Na ordem para realizar o resultado para todos os empregados, é necessário converter os valores nulos para numéricos. Nós usamos a função NVL para converter valores nulos para não nulos. NVL conta com dois argumentos:

- 1- uma expressão
- 2- um valor que será retornado caso o valor do primeiro argumento for nulo.

Note que você pode usar a função NVL para converter qualquer tipo de valor nulo.

NVL(COLUNA_DATA_NULA,'30-OCT-74')

NVL(COLUNA_NUMÉRICA_NULA,21)

NVL(COLUNA_ALFANUMÉRICA_NULA,'QUALQUER VALOR')

1.11 Previnindo a Seleção de Linhas Duplicadas – Cláusula Distinct

Para eliminar valores duplicados no resultado, incluímos o DISTINCT

Exemplo sem a clausula distinct:

```
SELECT DEPTNO  
FROM EMP ;
```

DEPTNO

20
30
30
20
30
30
10
20
10
30
20
30
20
10

Para eliminar os valores Duplicados:

```
SELECT DISTINCT DEPTNO  
FROM EMP ;
```

DEPTNO

10
20
30

EMPREGADO
7369SMITH
7499ALLEN
7521WARD
7566JONES
7654MARTIN
7698BLAKE
7782CLARK
7788SCOTT
7839KING
7844TURNER
7876ADAMS
7900JAMES
7902FORD
7934MILLER

1.9 Literais

Literais são um ou mais caracteres, expressões ou números, incluídos na lista do SELECT.

Um literal na lista do SELECT terá uma saída para cada linha retornada. Literais de livre formatos de textos podem ser incluídos no resultado da pesquisa e são tratados como uma coluna na lista do SELECT.

Datas e caracteres alfanuméricos devem ser colocados entre aspas simples('); números não precisam de aspas simples.

As declarações seguintes contêm literais selecionados com concatenação e colunas sinônimas.

Exemplo:

```
SELECT EMPNO || '-' || ENAME EMPREGADO, 'WORKS IN DEPARTMENT', DEPTNO
FROM      EMP;
```

EMPREGADO	'WORKS IN DEPARTMENT'	DEPTNO
7369-SMITH	WORKS IN DEPARTMENT	20
7499-ALLEN	WORKS IN DEPARTMENT	30
7521-WARD	WORKS IN DEPARTMENT	30
7566-JONES	WORKS IN DEPARTMENT	20
7654-MARTIN	WORKS IN DEPARTMENT	30
7698-BLAKE	WORKS IN DEPARTMENT	30
7782-CLARK	WORKS IN DEPARTMENT	10
7788-SCOTT	WORKS IN DEPARTMENT	20
7839-KING	WORKS IN DEPARTMENT	10
7844-TURNER	WORKS IN DEPARTMENT	30
7876-ADAMS	WORKS IN DEPARTMENT	20
7900-JAMES	WORKS IN DEPARTMENT	30
7902-FORD	WORKS IN DEPARTMENT	20
7934-MILLER	WORKS IN DEPARTMENT	10

1.10 Manuseando Valores Nulos

Um valor nulo é um valor indisponível e desconhecido. O valor nulo não é zero. Zero é um número. Se algum valor de uma coluna em uma expressão for nulo, o resultado será nulo.

Exemplo:

```
SELECT ENAME, SAL*12+COMM GANHO_ANUAL
FROM      EMP;
```

ENAME	(sal + 250) * 12
SMITH	12600
ALLEN	22200
WARD	18000
JONES	38700
MARTIN	18000
BLAKE	37200
CLARK	32400
SCOTT	39000
KING	63000
TURNER	21000
ADAMS	16200
JAMES	14400
FORD	39000
MILLER	18600

1.7 Colunas Sinônimas

Quando mostramos o resultado de uma pesquisa, o SQL*Plus normalmente usa-se o nome das colunas selecionadas como cabeçalho. Você pode modificar o cabeçalho de uma coluna usando sinônimos(alias).

Exemplo:

```
SELECT SAL*12 GANHO_ANUAL, COMM  
FROM EMP;
```

GANHO_ANUAL	COMM
9600	
19200	300
15000	500
35700	
15000	1400
34200	
29400	
36000	
60000	
18000	0
13200	
11400	
36000	
15600	

1.8 Operador de Concatenação

O Operador de Concatenação (||) permite que as colunas sejam juntadas com outras colunas, expressões aritméticas ou valores constantes para criar uma expressão alfanumérica.

Para combinar COD e NOME e obter o sinônimo EMPREGADO, observe o exemplo abaixo:

Exemplo:

```
SELECT EMPNO || ENAME EMPREGADO  
FROM EMP;
```

1.1 Programação em Oracle8 PL/SQL

WARD	15000	500
JONES	35700	
MARTIN	15000	1400
BLAKE	34200	
CLARK	29400	
SCOTT	36000	
KING	60000	
TURNER	18000	0
ADAMS	13200	
JAMES	11400	
FORD	36000	
MILLER	15600	

A prioridade é *,/, então +,-. No exemplo seguinte a multiplicação (250*12) é avaliada primeiro; então o valor do salário é adicionado no resultado da multiplicação (3000).

Exemplo:

```
Select NOME, sal + 250 * 12
from emp;
```

ENAME	SAL+250*12
SMITH	3800
ALLEN	4600
WARD	4250
JONES	5975
MARTIN	4250
BLAKE	5850
CLARK	5450
SCOTT	6000
KING	8000
TURNER	4500
ADAMS	4100
JAMES	3950
FORD	6000
MILLER	4300

Parênteses podem ser usados para especificar a ordem na qual serão executados os operadores, se, por exemplo, a adição é requerida antes da multiplicação.

Exemplo:

```
Select NOME, (sal + 250) * 12
from emp;
```

1.1 Programação em Oracle8 PL/SQL

Resultado da pesquisa:

EMPNO	ENAME	MGR
7369	SMITH	7902
7499	ALLEN	7698
7521	WARD	7698
7566	JONES	7839
7654	MARTIN	7698
7698	BLAKE	7839
7782	CLARK	7839
7788	SCOTT	7566
7839	KING	
7844	TURNER	7698
7876	ADAMS	7788
7900	JAMES	7698
7902	FORD	7566
7934	MILLER	7782

Exemplo 2:

```
SELECT * FROM EMP;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	MGR SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17/12/80	7902		20
7499	ALLEN	SALESMAN	7698	20/02/81	7698	300	30
7521	WARD	SALESMAN	7698	22/02/81	7698	500	30
7566	JONES	MANAGER	7839	02/04/81	7839		20
7654	MARTIN	SALESMAN	7698	28/09/81	7698	1400	30
7698	BLAKE	MANAGER	7839	01/05/81	7839		30
7782	CLARK	MANAGER	7839	09/06/81	7839		10
7788	SCOTT	ANALYST	7566	19/04/87	7566		20
7839	KING	PRESIDENT		17/11/81			10
7844	TURNER	SALESMAN	7698	08/09/81	7698	0	30
7876	ADAMS	CLERK	7788	23/05/87	7788		20
7900	JAMES	CLERK	7698	03/12/81	7698		30
7902	FORD	ANALYST	7566	03/12/81	7566		20
7934	MILLER	CLERK	7782	23/01/82	7782		10

1.6 Expressões Aritméticas

Expressões Aritméticas podem conter nome de colunas, valores numéricos, constantes e operadores aritméticos:

Operadores	Descrições
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

Exemplo:

```
SELECT      ENAME, SAL*12, COM  
FROM        EMP;
```

ENAME	SAL*12	COMM
SMITH	9600	
ALLEN	19200	300

1.2 PL/SQL

É uma linguagem de programação sofisticada que permite processar códigos diretamente no servidor, tornando o acesso rápido e eficiente.

Procedimentos compatíveis:

- declaração de variáveis.
- tarefas ($X := Y + Z$)
- controles condicionais (IF, THEN, ELSE, ELSIF, GOTO)
- Loop (FOR, WHILE, EXIT, WHEN)
- manuseio de exceções (exception)

1.3 Sintaxe de um bloco PL/SQL:

DECLARE	
Procedure	Área de declaração de variáveis
Procedure	
BEGIN	
Procedure	
SQL	
Procedure	
SQL	Comandos
END	

1.4 Alguns Comandos SQL:

SELECT	Esse é o comando mais comum, usado para retirar dados de uma Tabela
INSERT UPDATE DELETE	São os comandos usados para o preenchimento de novas linhas, modificando linhas existentes e removendo linhas não desejadas das tabelas.
CREATE ALTER DROP	São usados dinamicamente para configurar, modificar e remover várias estruturas de dados como: tabelas, visões, índices.
GRANT REVOKE	Usados para dar ou remover privilégios e direitos de acesso de um usuário ou grupo de usuários ao Banco de Dados ORACLE e às estruturas dentro dele.

1.5 SELECT

```
SELECT * (asterisco seleciona todos os campos)
FROM EMP(nome da tabela)
```

Exemplo 1:

```
SELECT EMPNO, ENAME, MGR
FROM EMP;
```

1.1 Programação em Oracle8 PL/SQL

Comando	Descrição
/	Executa o texto que está no buffer.
EXEC	Executa um simples comando SQL como por exemplo uma stored procedure.

1 Introdução ao SQL

Comandos SQL realizam tarefas como:

- pesquisar dados
- inserir, alterar e apagar linhas em uma tabela
- criar, modificar e apagar objetos do Banco de Dados
- controlar acesso para Banco de Dados e objetos do Banco de Dados
- garantir a consistência do banco de dados

1.1 SQL*Plus

SQL*Plus é uma interface na qual os comandos SQL podem ser entendidos e executados.

Comandos SQL*Plus são feitos no prompt do SQL>, eles não entram no buffer.

Comando	Descrição
SAVE arquivo	permite o corrente contexto do SQL buffer ser salvo em um arquivo
GET arquivo	chama texto previamente salvo
START arquivo	executa um comando previamente salvo em um arquivo. Comandos de arquivo são discutidos na Unidade 10
ED arquivo	usa editor padrão em ordem para edição do arquivo salvo.
SPOOL arquivo	escreve todos os comandos subsequentes de saída no arquivo nomeado. O arquivo SPOOL é estendido por .LIS (LST em algum sistemas)
SPO(OL) OFF/OUT	OFF fecha o arquivo SPOOL e OUT fecha o arquivo SPOOL e emite o arquivo para impressora.
DESC(RIBE) tabela	mostra a estrutura de uma tabela.
HELP	invoca o interno sistema de ajuda ORACLE
\$O/S comando	invoca um comando do sistema operacional.
HOST comando	mesma função acima
CONN(ECT) usuário/senha	Invoca outro usuário ORACLE
EXIT	sai SQL*Plus
PROMPT texto	mostra o texto quando executa o comando arquivo.
;	mostra o texto que está no buffer.

12.2	Exceções Definidas pelo Usuário	62
12.2.1	Utilizando OTHERS ou PRAGMA EXCEPTION_INIT	62
12.2.2	Raise_Application_Error	62
13	<i>DBMS_output.put_line()</i>	63
	Coloca uma linha no buffer, e mostra na tela.	63
14	<i>Subprogramas (Procedures e Functions)</i>	63
14.1	Parâmetros.....	64
14.2	Procedures	64
14.3	Functions:	64
14.4	Executando subprogramas através do SQL* Plus:.....	64
14.5	Eliminando um subprograma:.....	64
14.6	Análise das dependências:	64
15	<i>Packages</i>	65
15.1	Especificação	65
15.2	Body.....	65
15.3	Execução de Estruturas Públicas de uma Package:.....	66
16	<i>Triggers</i>	66
16.1	Criação de triggers.....	66
16.2	Triggers possíveis para uma tabela:	66
17	<i>Exercícios</i>	68
17.1	SQL.....	68

5.2	Eliminando uma sequência	46
6	Índices.....	47
6.1	Recuperando informações sobre Índices:	47
7	Controle de Concorrência.....	47
7.1	Tipos de Lock	47
7.1.1	Lock table – sobrepõe o lock implícito	47
7.1.2	Select for Update.....	48
8	Declarações.....	48
8.1	Declarações de Variáveis.....	48
8.2	Tipos de Dados PL/SQL	49
8.2.1	Escalar.....	49
8.2.1.1	Usando %TYPE	50
8.2.2	Tipos de Dados Composto(TABLES e RECORDS PL/SQL).	50
8.2.2.1	RECORDS	51
8.2.2.2	TABLE.....	51
8.2.2.3	Usando %ROWTYPE.....	52
8.3	Subtype.....	52
8.4	Atribuindo Valores às Variáveis.....	53
8.5	Escopo de Variáveis	53
9	Codificação de Comando SQL Dentro de PL/SQL.....	54
9.1	Tratamento de Transações	54
10	Estruturas de Controle.....	55
10.1	IF – THEN – ELSE	55
10.2	WHILE-LOOP	56
10.3	FOR-LOOP	56
10.4	LOOP	56
LOOP	57
11	Cursors:	57
11.1	Comandos de Manipulação do cursor:	57
11.1.1	Open:.....	57
11.1.2	Fetch.....	57
11.1.3	Close	59
11.2	O Comando For para abrir Cursors:	59
11.3	Atualização na tabela da linha atual do cursor:.....	59
11.4	Cursors Implícitos:.....	60
12	Tratamento de Exceções	61
12.1	Exceções Predefinidas.....	61

2.3.7	SIGN	27
2.3.8	ABS.....	27
2.3.9	MOD	27
2.4	Funções de Data	28
2.4.1	Sysdate	28
2.4.2	Usando Operadores Aritméticos	28
2.4.3	MONTHS_BEETWEEN	29
2.4.4	ADD_MONTHS	29
2.4.5	NEXT_DAY	29
2.4.6	LAST_DAY.....	30
2.4.7	ROUND	30
2.4.8	TRUNC	30
2.5	Funções de Conversão	31
2.5.1	TO_CHAR	31
2.5.2	TO_NUMBER	33
2.5.3	TO_DATE	34
2.6	Funções que Aceitam Vários Tipos de Entrada de Dados.....	35
2.6.1	DECODE	35
2.6.2	NVL	36
2.6.3	GREATEST	36
2.6.4	LEAST	36
2.6.5	VSIZE	37
2.7	Funções de Grupo	37
2.7.1	AVG.....	37
2.7.2	MIN.....	37
2.7.3	COUNT.....	38
2.7.4	A cláusula GROUP BY.....	38
2.7.4.1	Grupos dentro de Grupos	38
2.7.4.2	Funções de Grupo e Resultados Individuais	39
2.7.5	A clausula HAVING.....	39
2.7.6	A Ordem das clausulas na declaração SELECT.	40
3	<i>Executando Pesquisas Padrões com Variáveis Substituíveis</i>	40
3.1	Única Variável Substituível.....	40
3.2	Duplo & para Variáveis substituíveis.....	41
3.3	O Comando DEFINE.....	42
3.4	O comando ACCEPT.....	42
4	<i>Recuperando valores da base de dados</i>	43
4.1	Comandos de Manipulação de Dados (DML)	44
4.1.1	Insert	44
4.1.2	Update	44
4.1.3	Delete	44
4.1.4	Create Table.....As Select.....	44
4.1.5	Rename	45
5	<i>Seqüências</i>	46
5.1	Alterando uma seqüência	46

1	Introdução ao SQL.....	5
1.1	SQL*Plus	5
1.2	PL/SQL	7
1.3	Sintaxe de um bloco PL/SQL:.....	7
1.4	Alguns Comandos SQL:	7
1.5	SELECT	7
1.6	Expressões Aritméticas.....	8
1.7	Colunas Sinônimas	10
1.8	Operador de Concatenação.....	10
1.9	Literais	11
1.10	Manuseando Valores Nulos.....	11
1.11	Prevenindo a Seleção de Linhas Duplicadas – Cláusula Distinct	12
1.12	A cláusula ORDER BY	13
1.13	A Cláusula WHERE	14
1.14	Operadores SQL	15
1.14.1	O Operador BETWEEN.....	15
1.14.2	O Operador IN	16
1.14.3	O Operador LIKE	16
1.14.4	Operador IS NULL	17
1.15	Expressões Negativas	17
1.16	Pesquisando Dados com Múltiplas Condições.....	18
2	Funções.....	20
2.1	Funções Alfanuméricas.....	21
2.1.1	LOWER	21
2.1.2	UPPER.....	21
2.1.3	INITCAP.....	21
2.1.4	LPAD e RPAD.....	22
2.1.4	SUBSTR	22
2.1.5	INSTR.....	23
2.1.6	LTRIM e RTRIM.....	23
2.1.7	SOUNDEX	23
2.1.8	LENGTH	24
2.1.9	TRANSLATE e REPLACE.....	24
2.2	Aninhamento de Funções	25
2.3	Funções Numéricas	25
2.3.1	ROUND	25
2.3.2	TRUNC.....	25
2.3.3	CEIL.....	26
2.3.4	FLOOR	26
2.3.5	POWER	26
2.3.6	SQRT	26