

## APOSTILA DE SQL

O objetivo da Apostila é trazer os principais comandos SQL usados em aplicações ORACLE, com exemplos, comentários e explicações.

## FINALIDADE DO CAPITULO:

No final deste capítulo você será capaz de criar scripts permitindo que o usuário entre com valores diversos.

A finalidade dos exemplos abaixo é demonstrar como permitir que o usuário entre com os valores que possibilitem a execução de um comando **SQL**, isso tudo em tempo de execução.

Verifique o uso do "&" que permite que o usuário entre com dados. No exemplo abaixo temos um pedido de entrada de um número, por isso é que o **&numero\_do\_dept** não está entre aspas. Podemos notar também que aparece o **old** valor e o **new** valor, isso é devido ao cmdo **set verify on**, se quisermos que não apareça, devemos usar o **set verify off**.

Vamos ao Exemplo:

```
SQL> set verify on
SQL> select id,last_name,salary
  2  from    s_emp
  3  where   dept_id=&numero_do_dept;
Enter value for numero_do_dept: 31
old   3: where   dept_id=&numero_do_dept
new   3: where   dept_id=31
```

ID	LAST_NAME	SALARY
3	Nagayama	1400
11	Magée	1400

Agora não aparecerá o new e old valor porque estamos usando o **SET VERIFY OFF**.

```
SQL> set verify off
SQL> select id,last_name,salary
  2  from    s_emp
  3  where   dept_id=&numero_do_dept;
Enter value for numero_do_dept: 31
```

ID	LAST_NAME	SALARY
----	-----------	--------

3 Nagayama	1400
11 Magee	1400

Vimos anteriormente como que temos que proceder para que o usuário entre com valores numéricos, para que seja realizada

uma pesquisa, agora vamos possibilitar que o usuário ente com valores do tipo caracter, a principal diferença é o uso das aspas, mas temos que ter atenção para o formato na hora de digitarmos, porque tem que ser o mesmo do conteúdo que está na tabela, com maiúsculas e minúsculas.

Vamos ao Exemplo:

```
SQL> select id,last_name,salary
      2   from s_emp
      3   where title = '&job_title';      {possibilitando que o
                                          usuário entre com o nome
                                          do title}
```

Enter value for job\_title: Stock Clerk

ID	LAST_NAME	SALARY
16	Maduro	1400
17	Smith	940
18	Nozaki	1200
19	Patel	795
20	Newman	750
21	Markarian	850
22	Chang	800
23	Patel	795
24	Dancs	860
25	Schwartz	1100

10 rows selected.

Nos exemplos anteriores o usuário entrou com a condição de pesquisa quer seja numérica ou caracter, cada qual com suas características, agora vamos ver como que possibilitamos ao usuário entrar com o nome da coluna e com a própria condição de pesquisa que será estabelecida na cláusula **WHERE**. Neste caso temos um exemplo onde é requerida a entrada de um dado numérico.

Vamos ao exemplo:

```
SQL> select id,&nome_coluna
  2   from s_emp
  3   where &condição;
Enter value for nome_coluna: LAST_NAME
Enter value for condição: SALARY > 100
```

```
      ID LAST_NAME
-----
      1 Velasquez
      2 Ngao
      3 Nagayama
      4 Quick-To-See
      5 Ropeburn
      6 Urguhart
      7 Menchu
      8 Biri
```

Temos um outro exemplo em que o usuário entra com o valor da coluna e da condição da **WHERE**.

```
SQL> select id,&nome_coluna
  2   from s_ord
  3   where &condição;
Enter value for nome_coluna: date_ordered
Enter value for condição: total>30000
```

```
      ID DATE_ORDE
-----
     100 31-AUG-92
     104 03-SEP-92
     107 07-SEP-92
     108 07-SEP-92
     109 08-SEP-92
      97 28-AUG-92
```

6 rows selected.

## CONSTRUINDO SCRIPTS USANDO OPÇÕES PARA O USUÁRIO ENTRE COM DADOS E TAMBÉM PARA SÓ MOSTRAR NA TELA O PROMPT.

No exemplo abaixo estamos usando os seguintes comandos que na da mais é do que a soma de tudo aquilo que vimos neste capítulo mas também algo mais:

O cmdo **SET ECHO OFF** serve para em tempo de execução não se exiba os comandos do **SQL** e **SET ECHO ON** serve para retornar a forma anterior.

O cmdo **ACCEPT** serve para que preparemos um **PROMPT** para receber um valor. Neste exemplo também temos a criação de uma variável chamada **V\_NAME** que recebe valores.

Estamos preparando o ambiente para receber valores que serão armazenados dentro de uma variável, para após isto, serem feitas comparações dentro da cláusula **WHERE**.

Estamos usando duas tabelas **S\_DEPT** e **S\_REGION**, o **AND** é uma função onde complementa a cláusula **WHERE**, e o **UPPER** no **DPT.NAME** está passando o conteúdo do nome do dept para maiúsculo para que seja efetuada a comparação com um nome que será digitado pelo usuário, que por sua vez recebe um **UPPER** que o transforma em maiúsculo. Essa alternativa é feita porque não sabemos qual é o formato do dado na tabela.

Vamos ao exemplo:

```
SET ECHO OFF
ACCEPT V_NAME PROMPT 'DÊ O NOME DO DEPARTAMENTO:'
SELECT DPT.NAME, REG.ID, REG.NAME " NOME DA REGIÃO"
FROM S_DEPT DPT, S_REGION REG {veja a criação de apelidos}
WHERE DPT.REGION_ID = REG.ID {veja o join}
AND UPPER(DPT.NAME) LIKE UPPER('%&V_NAME%') {valor digitado }
/
SET ECHO ON
```

Como estamos gerando um **SCRIPT**, os comandos devem ficar armazenados dentro de um arquivo que possua a extensão **SQL** e preparado da forma descrita acima quando estivermos no **SQL** e desejarmos executar o nosso **SCRIPT** temos que seguir o procedimento descrito abaixo, usando "@" e o nome do arquivo, ou "**START**" e o nome do arquivo.

Vamos ao exemplo:

```
SET ECHO ON
SQL> @TEST.SQL
SQL> SET ECHO OFF
DÊ O NOME DO DEPARTAMENTO:sales
old    4: AND UPPER(DPT.NAME) LIKE UPPER('%&V_NAME%')
new    4: AND UPPER(DPT.NAME) LIKE UPPER('%sales%')
```

NAME	ID	NOME DA REGIÃO
-----	-----	-----
Sales	1	North America
Sales	2	South America
Sales	3	África / Middle
East		
Sales	4	Ásia
Sales	5	Europe

Podemos notar que por ocasião da execução o exemplo anterior mostrou o **OLD** e o **NEW** valores da variável, para não mostrar temos que usar o **SET VERIFY OFF** veja abaixo:

Vamos ao exemplo:

```
SET VERIFY OFF
SET ECHO OFF
ACCEPT V_NAME PROMPT 'DÊ O NOME DO DEPARTAMENTO:'
SELECT DPT.NAME, REG.ID, REG.NAME " NOME DA REGIÃO"
FROM S_DEPT DPT, S_REGION REG
WHERE DPT.REGION_ID = REG.ID
AND UPPER(DPT.NAME) LIKE UPPER('%&V_NAME%')
/
SET ECHO ON
```

Executando o SCRIPT:

```
SQL> START TEST.SQL
SQL> SET VERIFY OFF
SQL> SET ECHO OFF
DÊ O NOME DO DEPARTAMENTO:SALES
```

NAME	ID	NOME DA REGIÃO
-----	-----	-----
Sales	1	North America

Sales	2 South America
Sales	3 Africa / Middle
East	
Sales	4 Ásia
Sales	5 Europe

Input truncated to 11 characters

#### OUTRO EXEMPLO DE GERAÇÃO DE SCRIPT:

Mais uma vez vamos usar alguns comandos já vistos em exemplos anteriores, mas também alguns comandos novos e precisamos

Mostrar a descrição de tais comandos:

O comando **SET ECHO OFF** tem a finalidade de não deixar aparecer os cmdos feitos para a execução.

O cmdo **VERIFY OFF** tem a finalidade de não mostrar os valores recebidos pelas as variáveis que no exemplo em questão são **MENOR\_DT** e **MAIOR\_DT**.

O **ACCEPT** é usado para criar variáveis e o **PROMPT** para receber valores para as variáveis.

Criadas as variáveis, observe a sintaxe e o "-" entre a definição do formato de data o uso de **DATE**, o **FORMAT** e a especificação '**MM/DD/YY**'.

Estamos usando também o **COLUMN FORMAT A30** para formatar o tamanho da coluna **EMPREGADO** e logo depois estamos concatenando as colunas **FIRST\_NAME** e **LAST\_NAME**.

Estamos também usando o **BETWEEN** para pesquisarmos valores que estão entre uma data e outra.

Observe o uso do **TO\_DATE** antes da colocação da variável com o "&" para receber valores que o usuário digita, e observe também a forma que foi colocado o formato da data '**MM/DD/YY**' e o uso do **AND** que faz parte da **WHERE**.

O **UNDEFINE** serve para que as variáveis percam os valores após a execução.

Vamos ao Exemplo:

```
SET ECHO OFF
```

```
SET VERIFY OFF
```

```
ACCEPT MENOR_DT DATE FORMAT 'MM/DD/YY' -  
PROMPT 'ENTRE MENOR DATA (MM/DD/AA) :'  
ACCEPT MAIOR_DT DATE FORMAT 'MM/DD/YY' -  
PROMPT 'ENTRE MAIOR DATA (MM/DD/AA) :'
```

```
COLUMN EMPREGADO FORMAT A30
SELECT USERID, FIRST_NAME || ' ' || LAST_NAME "EMPREGADO",
START_DATE FROM S_EMP
WHERE START_DATE BETWEEN TO_DATE('&MENOR_DT', 'MM/DD/YY')
AND TO_DATE('&MAIOR_DT', 'MM/DD/YY')
```

```
/
UNDEFINE MENOR_DT
UNDEFINE MAIOR_DT
COLUMN EMPREGADO CLEAR
SET ECHO ON
SET VERIFY ON
```

OUTRO EXEMPLO DE SCRIPT:

Estamos montando um script com a finalidade de receber um valor que pesquise registros, no caso estamos querendo pesquisar o ID e o NOME de um CUSTOMER e para isso recebendo um dos nomes do CUSTOMER, por isso estamos usando o **LIKE** e "**%&NOME%**" estamos transformando a coluna **NAME** da tabela a ser pesquisada em maiúsculo para que qualquer nome que seja digitado em maiúsculo seja pesquisado.

Está sendo criado a variável **NOME** pelo **ACCEPT** e o **PROMPT** possibilita a recepção de um valor, observe a sintaxe "-" após a variável NOME.

Vamos ao exemplo:

```
SET ECHO OFF
SET VERIFY OFF
ACCEPT NOME -
PROMPT ' ENTRE COM O NOME DESEJADO : '
SELECT ID, NAME FROM S_CUSTOMER
WHERE UPPER(NAME) LIKE '%&NOME%'
/
SET ECHO ON
SET VERIFY ON
```

## **FINAL DE CAPITULO**

## FINALIDADE DO CAPITULO:

O COMANDO ALTER TABLE E SUAS VARIAÇÕES:

Adicionado uma coluna em uma tabela:

No exemplo a seguir estamos usando o cmdo **ALTER TABLE** para adicionar uma coluna em uma tabela, a coluna adicionada a tabela sempre será a última, importante saber que não podemos deletar uma coluna de uma tabela mas somente adicionar, se precisarmos dropar uma coluna a solução é dropar a tabela e recriá - la sem a coluna. Vamos ao exemplo:

```
SQL> ALTER TABLE S_REGION
ADD      (comments VARCHAR(255));
```

Table altered.

O COMD MODIFY:

No exemplo abaixo estamos modificando uma coluna usando o comando **modify**, no caso estamos aumentando o tamanho da coluna **title** para **50**, é permitido aumentar o tamanho da coluna mas diminuir não é permitido.

```
SQL> ALTER TABLE s_emp
2  MODIFY      (title VARCHAR(50));
```

Table altered.

ADICIONANDO UMA CONSTRAINT:

Para adicionarmos uma constraint temos que usar O **ALTER TABLE** criar um nome para nossa constraint no caso '**s\_emp\_manager\_id\_fk**', escrevendo de onde ela é colocando a referência com o id de s\_emp.

Vamos ao exemplo:

```
SQL>
ALTER TABLE s_emp
ADD CONSTRAINT s_emp_manager_id_fk
FOREIGN KEY (MANAGER_ID)
REFERENCES s_emp(id)
```

DROPANDO UMA CONSTRAINT:

Este exemplo mostra como dropar uma CONSTRAINT, Neste caso o nome da constraint é **s\_emp\_manager\_id\_fk**, verifique a forma como foi escrito o exemplo abaixo:

```
SQL> ALTER TABLE S_EMP
      2 DROP CONSTRAINT s_emp_manager_id_fk;
```

Table altered.

DROPANDO REGISTROS EMCACATA:

Para dropar uma chave primaria em cascata ou seja, deletá - la de forma que seja deletada em todas as tabelas com quem tenha relação, temos prosseguir da seguinte forma:

Vamos ao exemplo:

```
SQL> ALTER TABLE s_dept
      2 DROP primary key cascade; {observe o cmdo cascade}
```

Table altered.

Desabilitando CONSTRAINTS de uma tabela:

Verifique o cmdo **CASCADE** sendo usado no **ALTER TABLE**, a finalidade de seu uso é permitir que todos os objetos que fazem referência ao campo ID de S\_EMP aceitem a desabilitação da constraint.

Vamos ao exemplo:

```
SQL> ALTER TABLE s_emp
      2 DISABLE CONSTRAINT s_emp_id_pk CASCADE;
```

Table altered.

Habilitando uma constraint:

Para habilitarmos uma constraint não precisamos usar o cmdo cascade, porque é criada de forma automática a UK ou FK.

Vamos ao exemplo

```
SQL> ALTER TABLE s_emp  
      2  ENABLE CONSTRAINT s_emp_id_pk;
```

Table altered.

**FINAL DE CAPITULO**

FINALIDADE DO CAPITULO:

O COMANDO SELECT TO\_CHAR:

O exemplo abaixo temos um select que está selecionando id,date\_ordered da tabela s\_ord onde o sales\_rep\_id é igual a 11, podemos notar que a date\_ordered(data) é igual ao formato de data padrão do oracle, ou seja, mês por extenso abreviado e o ano com os dois últimos números do ano.

Vamos ao exemplo:

```
SQL> SELECT ID,DATE_ORDERED
      2 FROM S_ORD
      3 WHERE
      4 SALES_REP_ID = 11;
```

```
          ID DATE_ORDE
-----
100 31-AUG-92
105 04-SEP-92
109 08-SEP-92
110 09-SEP-92
111 09-SEP-92
```

No exemplo abaixo estamos com o comando " **to\_char** " mudando o formato da data para o modelo que nós estamos querendo mostrar em tela, ou seja, estamos mudando o formato padrão do oracle para um novo formato correspondente ao que queremos, no caso '**mm/yy**' que corresponde ao mês e o ano.

Vamos ao exemplo:

```
SQL> SELECT ID,TO_CHAR(DATE_ORDERED,'MM/YY') ORDERED
      2 FROM S_ORD
      3 WHERE SALES_REP_ID = 11;
```

```
          ID ORDERED
-----
100 08/92
105 09/92
109 09/92
110 09/92
111 09/92
```

Agora vamos converter a data colocando o dia por extenso ou seja usando, **"day" (extenso)**, para o dia e colocando o mês por extenso usando, **"month"** para mês e colocando o ano por extenso usando o **"yyyy"** para ano.

Vamos ao Exemplo:

```
SELECT ID,TO_CHAR (DATE_ORDERED,'DAY/MONTH/YYYY') ORDERED
FROM S_ORD
WHERE SALES_REP_ID = 11
```

```
ID ORDERED
-----
100 MONDAY    /AUGUST     /1992
105 FRIDAY    /SEPTEMBER/1992
109 TUESDAY   /SEPTEMBER/1992
110 WEDNESDAY/SEPTEMBER/1992
111 WEDNESDAY/SEPTEMBER/1992
```

Agora estamos colocando o nome da ano escrito por extenso usando a FUNÇÃO **"YEAR"**.

```
SELECT ID,TO_CHAR (DATE_ORDERED,'DAY/MONTH/YEAR') ORDERED
FROM S_ORD
WHERE SALES_REP_ID = 11;
```

```
ID ORDERED
-----
100 MONDAY    /AUGUST     /NINETEEN NINETY-TWO
105 FRIDAY    /SEPTEMBER/NINETEEN NINETY-TWO
109 TUESDAY   /SEPTEMBER/NINETEEN NINETY-TWO
110 WEDNESDAY/SEPTEMBER/NINETEEN NINETY-TWO
111 WEDNESDAY/SEPTEMBER/NINETEEN NINETY-TWO
```

Agora estamos usando o cmdo **" dy "** para mudar o dia da semana, passando da escrita por extenso para abreviado em três dígitos, observe o exemplo.

Vamos ao exemplo:

```
SQL> SELECT ID,TO_CHAR (DATE_ORDERED,'DY/MONTH/YEAR') ORDERED
2     FROM S_ORD
3     WHERE SALES_REP_ID = 11;
```

ID ORDERED

-----  
100 MON/AUGUST /NINETEEN NINETY-TWO  
105 FRI/SEPTEMBER/NINETEEN NINETY-TWO  
109 TUE/SEPTEMBER/NINETEEN NINETY-TWO  
110 WED/SEPTEMBER/NINETEEN NINETY-TWO  
111 WED/SEPTEMBER/NINETEEN NINETY-TWO

O exemplo abaixo demonstra como colocar o dia em formato numérico "FMDD" e o mês e o ano por extenso.

```
Select LAST_NAME, TO_CHAR(START_DATE, 'FMDD "OF" MONTH  
YYYY') HIREDATE  
FROM S_EMP  
WHERE START_DATE LIKE '%91'
```

LAST\_NAME

-----  
HIREDATE  
-----

Nagayama  
17 OF JUNE 1991

Urguhart  
18 OF JANUARY 1991

Havel  
27 OF FEBRUARY 1991

Sedeghi  
18 OF FEBRUARY 1991

Dumas  
09 OF OCTOBER 1991

Nozaki  
09 OF FEBRUARY 1991

Patel

LAST\_NAME

-----  
HIREDATE  
-----

06 OF AUGUST 1991

Newman  
21 OF JULY 1991

Markarian  
26 OF MAY 1991

Dancs  
17 OF MARCH 1991

Schwartz  
09 OF MAY 1991

No exemplo abaixo estamos usando a função **to\_char** novamente, agora para formatarmos um numero e concatenarmos uma coluna numérica com um comentário caracter.

Vamos ao Exemplo:

SQL>

```
SELECT 'O PEDIDO ' || TO_CHAR(ID) || '  
TEM UM TOTAL DE: ' || TO_CHAR(TOTAL, 'FM$9,999,999')  
FROM S_ORD  
WHERE DATE_SHIPPED = '21-SEP-92'  
/
```

```
'OPEDIDO' || TO_CHAR(ID) || 'TEMUMTOTALDE: ' ||  
TO_CHAR(TOTAL, 'FM$9,999,999')
```

-----  
-----

```
O PEDIDO 107  
TEM UM TOTAL DE: $142,171
```

```
O PEDIDO 110  
TEM UM TOTAL DE: $1,539
```

```
O PEDIDO 111  
TEM UM TOTAL DE: $2,770
```

**FINAL DE CAPÍTULO.**

Finalidade do Capítulo:

Demonstrar o uso do Comando **TO\_DATE**.

Verifique que no exemplo abaixo estamos querendo fazer uma pesquisa usando um campo **data** onde escrevemos a data, o mês e o ano e o cmdo **to\_date** transforma o que digitamos em data para a pesquisa, só que deve ser obedecida uma seqüência lógica nas posições por isso o exemplo abaixo está incorreto.

Vamos ao exemplo:

```
SQL>  SELECT DATE_ORDERED
      2     FROM S_ORD
      3     WHERE DATE_ORDERED =
      4     TO_DATE('31 1992, AUGUST', 'DD MONTH, YYYY');
ERROR:
ORA-01843: not a valid month
```

Vamos tentar corrigir o exemplo acima. A função do cmdo **to\_date** é converter valores digitados, numéricos e caracteres para data, a ordem da data procurada no exemplo abaixo tem que ser a mesma do formato de data colocado da seguinte forma:

```
SQL>  SELECT DATE_ORDERED
      2     FROM S_ORD
      3     WHERE DATE_ORDERED =
      4     TO_DATE('31 1992, AUGUST', 'DD, YYY MONTH');
```

```
DATE_ORDE
-----
31-AUG-92
31-AUG-92
31-AUG-92
31-AUG-92
31-AUG-92
```

## FINAL DE CAPÍTULO

Finalidade do Capítulo:

Demonstrar o uso dos comandos COMMIT, ROLLBACK E SAVE POINT.

Os cmdos **COMMIT** , **ROLLBACK** e **SAVE POINT** são usados para controle de execução, de confirmação e de retorno. Veja abaixo que estamos alterando o valor de SALARY na tabela S\_EMP usando o **UPDATE** e **SET**, após alterá-los nós criamos um **SAVEPOINT**, que serve simplesmente para o controle de ponto, para se quisermos dar um **ROLLBACK** nas alterações feitas até então, termos condições de limitar o espaço atingindo pelo **ROLLBACK**.

Vamos ao Exemplo de Update:

```
SQL> EDIT
Wrote file afiedt.buf
  1  UPDATE S_EMP
  2  SET    SALARY = SALARY * 1.1
  3* WHERE TITLE ='Stock Clerk'
SQL> /
```

10 rows updated.

Agora vamos criar um **SAVEPOINT**.

Vamos ao Exemplo:

```
SQL> savepoint update_ponto;
```

Savepoint created.

Agora usando o cmdo **INSERT INTO** estamos inserindo registros na tabela s\_region.

```
SQL> insert into s_region (id,name)
  2  values (8,'central');
```

1 row created.

Estamos selecionando os registros de s\_region para confirmarmos a inserção ou seja, se realmente foi incluído registros nesta tabela.

```
SQL> select * from
      2  s_region
      3  where id = 8;
```

```
      ID NAME
-----
      8 central
```

Agora após confirmarmos a inclusão dos registros, nós decidimos que não queremos que seja incluído registros na referida tabela `s_region` e para voltarmos a nossa ação temos então que dar um **ROLLBACK**, para desfazer o **INSERT**, só que somente até o ponto da inserção, ou seja até o **SAVE POINT**, que foi criado anteriormente ou seja tudo que está antes do **SAVE POINT** continua com suas alterações preservadas, é bom observar que para o **SAVE POINT** foi criado um nome no caso **UPDATE\_PONTO**, isso serve para que possamos referenciar de forma correta até que ponto queremos cancelar nossas ações ou seja até que ponto não queremos que sejam salvas nossas ações.

Vamos ao exemplo:

```
SQL> ROLLBACK TO UPDATE_PONTO;
```

Rollback complete.

Agora vamos dar novamente um **select** para confirmarmos se o nosso **ROLLBACK** realmente fez efeito, observe que as alterações feitas anteriormente não foram concretizadas.

```
SQL> select * from
      2  s_region
      3  where id = 8;
```

no rows selected

O CMDO **ROLLBACK** serve para que seja desfeitas as alterações que foram efetuadas em uma tabela.

Vamos ao exemplo:

```
SQL> delete from test;
25,000 rows deleted
```

para cancelar o **DELETE** na tabela `test`.

```
SQL> ROLLBACK;  
Rollback complete.
```

O uso do COMMIT:

O comando **COMMIT** é o contrário do **ROLLBACK** ou seja serve para confirmar as alterações que por ventura sejam feitas. Uma vez dado um **COMMIT** não podemos retornar mais atrás.

```
SQL> delete from test;  
25,000 rows deleted
```

Vamos confirmar o DELETE com o **COMMIT**.

```
SQL> COMMIT;  
Commit complete.
```

## **FINAL DE CAPÍTULO**

Finalidade do Capítulo é demonstrar como criar comentários a respeito de tabelas.

O comentários criado a respeito de tabelas e colunas são armazenados no dicionário de dados em:

```
ALL_COL_COMMENTS
USER_COL_COMMENTS
ALL_TAB_COMMENTS
USER_TAB_COMMENTS
```

Criando comentários para uma tabela:

```
SQL> COMMENT ON TABLE s_emp IS ' informação sobre
funcionário';
```

Comment created.

Como criar comentários para uma coluna:

```
SQL> COMMENT ON COLUMN s_emp.last_name IS ' ultimo';
```

Comment created.COMMENT ON COLUMN s\_emp.last\_name IS '
último'

Como verificar os comentários existentes dentro de uma tabela ou coluna:

Primeiramente vamos verificar os comentários relativos as colunas pertencentes a uma tabela **all\_col\_comments**, para depois realizarmos um **select** em seu conteúdo pesquisando pelo nome da tabela ou coluna para sabermos qual é o comentário específico a respeito.

```
SQL> desc all_col_comments
```

Name	Null?	Type
OWNER	NOT NULL	
VARCHAR2(30)		
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
COMMENTS		
VARCHAR2(2000)		

Vamos realizando um **SELECT ALL\_COL\_COMMENTS** para vermos quais os comentários relativos a todas as colunas da tabela s\_emp que são armazenados em **COMMENTS** :

```
SQL> select COMMENTS
      2   from all_col_comments
      3   where TABLE_NAME = 'S_EMP';
```

COMMENTS

-----

ultimo

COMMENTS

-----

Agora queremos saber o comentário a respeito da tabela s\_emp veja abaixo:

```
SQL> SELECT COMMENTS
      2   FROM ALL_TAB_COMMENTS
      3   WHERE TABLE_NAME = 'S_EMP';
```

COMMENTS

-----

informação sobre funcionário

**FIM DE CAPÍTULO**

Finalidade do Capítulo Tratar de Assuntos Relativos a Constraints.

Vamos ver como verificar as **constraints** de uma tabela O seus tipos e nomes ou seja se é not null, se é foreing key, unique key ou primary key.

```
SQL> SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE,  
SEARCH_CONDITION,  
2 R_CONSTRAINT_NAME  
3 FROM USER_CONSTRAINTS  
4 WHERE TABLE_NAME = 'S_DEPT';
```

```
CONSTRAINT_NAME          C  
-----  
SEARCH_CONDITION  
-----  
R_CONSTRAINT_NAME  
-----  
S_DEPT_ID_NN             C  
ID IS NOT NULL  
  
S_DEPT_NAME_NN          C  
NAME IS NOT NULL  
  
S_DEPT_ID_PK            P  
  
S_DEPT_NAME_REGION_ID_UK    U  
  
S_DEPT_REGION_ID_FK      R  
-----  
S_REGION_ID_PK
```

Para vermos o nome das colunas envolvidas com constraints e seus respectivos nomes naturais temos que fazer um select na tabela **user\_cons\_columns** conforme o exemplo a seguir:

```
SQL> SELECT CONSTRAINT_NAME, COLUMN_NAME
       2 FROM USER_CONS_COLUMNS
       3 WHERE TABLE_NAME = 'S_DEPT';
```

CONSTRAINT_NAME	COLUMN_NAME
S_DEPT_ID_NN	ID
S_DEPT_ID_PK	ID
S_DEPT_NAME_NN	NAME
S_DEPT_NAME_REGION_ID_UK	NAME
S_DEPT_NAME_REGION_ID_UK	REGION_ID
S_DEPT_REGION_ID_FK	REGION_ID

6 rows selected.

## **FINAL DE CAPÍTULO**

Finalidade do Capítulo é o Uso do Create, Drop Table.

O comando **DROP** é usado para deletar a tabela ou seja, apagá-la fisicamente.

O comando **CREATE** é usado na criação de tabelas e o **INSERT** na inserção de dados em tabelas.

Vamos ao exemplo de como **DROPAR** uma tabela, VIEW ou **SEQUENCE**:

```
DROP TABLE EMP;
DROP TABLE DEPT;
DROP TABLE BONUS;
DROP TABLE SALGRADE;
DROP TABLE DUMMY;
DROP TABLE ITEM;
DROP TABLE PRICE;
DROP TABLE PRODUCT;
DROP TABLE ORD;
DROP TABLE CUSTOMER;
DROP VIEW SALES;
DROP SEQUENCE ORCID;
DROP SEQUENCE CUSTID;
DROP SEQUENCE PROPID;
```

Vamos ao exemplo de como criar uma tabela, observe os detalhes em relação aos parênteses, nome das colunas ,data type e **constraints**.

```
CREATE TABLE DEPT (
  DEPTNO          NUMBER(2) NOT NULL,
  DNAME          CHAR(14),
  LOC            CHAR(13),
  CONSTRAINT DEPT_PRIMARY_KEY PRIMARY KEY (DEPTNO));
```

Após criarmos a tabela vamos inserir dados a mesma, em um comando **INSERT**

```
INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');
```

```

CREATE TABLE EMP
( EMPNO                NUMBER(4) NOT NULL,
  ENAME                CHAR(10),
  JOB                  CHAR(9),
  MGR                  NUMBER(4) CONSTRAINT EMP_SELF_KEY
REFERENCES EMP (EMPNO),
  HIREDATE             DATE,
  SAL                  NUMBER(7,2),
  COMM                 NUMBER(7,2),
  DEPTNO               NUMBER(2) NOT NULL,
  CONSTRAINT EMP_FOREIGN_KEY FOREIGN KEY (DEPTNO) REFERENCES
DEPT (DEPTNO),
  CONSTRAINT EMP_PRIMARY_KEY PRIMARY KEY (EMPNO));

```

Inserindo Registros na Tabela Criada Usando o cmdo **INSERT**.

```

INSERT INTO EMP VALUES (7839, 'KING', 'PRESIDENT', NULL, '17-NOV-
81', 5000, NULL, 10);
INSERT INTO EMP VALUES (7698, 'BLAKE', 'MANAGER', 7839, '1-MAY-
81', 2850, NULL, 30);
INSERT INTO EMP VALUES (7782, 'CLARK', 'MANAGER', 7839, '9-JUN-
81', 2450, NULL, 10);
INSERT INTO EMP VALUES (7566, 'JONES', 'MANAGER', 7839, '2-APR-
81', 2975, NULL, 20);
INSERT INTO EMP VALUES (7654, 'MARTIN', 'SALESMAN', 7698, '28-
SEP-81', 1250, 1400, 30);
INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', 7698, '20-FEB-
81', 1600, 300, 30);
INSERT INTO EMP VALUES (7844, 'TURNER', 'SALESMAN', 7698, '8-SEP-
81', 1500, 0, 30);
INSERT INTO EMP VALUES (7900, 'JAMES', 'CLERK', 7698, '3-DEC-
81', 950, NULL, 30);
INSERT INTO EMP VALUES (7521, 'WARD', 'SALESMAN', 7698, '22-FEB-
81', 1250, 500, 30);
INSERT INTO EMP VALUES (7902, 'FORD', 'ANALYST', 7566, '3-DEC-
81', 3000, NULL, 20);
INSERT INTO EMP VALUES (7369, 'SMITH', 'CLERK', 7902, '17-DEC-
80', 800, NULL, 20);
INSERT INTO EMP VALUES (7788, 'SCOTT', 'ANALYST', 7566, '09-DEC-
82', 3000, NULL, 20);
INSERT INTO EMP VALUES (7876, 'ADAMS', 'CLERK', 7788, '12-JAN-
83', 1100, NULL, 20);
INSERT INTO EMP VALUES (7934, 'MILLER', 'CLERK', 7782, '23-JAN-
82', 1300, NULL, 10);

```

Outro Exemplo de Como Criar uma Tabela:

```
CREATE TABLE BONUS (  
  ENAME          CHAR(10),  
  JOB            CHAR(9),  
  SAL            NUMBER,  
  COMM           NUMBER);
```

```
CREATE TABLE SALGRADE (  
  GRADE          NUMBER,  
  LOSAL          NUMBER,  
  HISAL          NUMBER);
```

Inserindo Valores na Nova Tabela.

```
INSERT INTO SALGRADE VALUES (1,700,1200);  
INSERT INTO SALGRADE VALUES (2,1201,1400);  
INSERT INTO SALGRADE VALUES (3,1401,2000);  
INSERT INTO SALGRADE VALUES (4,2001,3000);  
INSERT INTO SALGRADE VALUES (5,3001,9999);
```

Para criarmos uma tabela temos antes que mais nada ter permissão para isto ou seja **"GRANT"**, que nos é concedido através do **DBA**, depois não podemos nos esquecer de criar as **CONSTRAINTS** que são as **PK** os campos **NOT NULL** as **FK**. O observe a sintaxe da criação de tabela, a definição do tipo e tamanho das colunas (**Data Type**), a referencia a chave estrangeira e a criação, as colunas de formato **DATE** usa-se o padrão **SYSDATE** que corresponde a data do sistema.

Vamos ao Exemplo:

```
CREATE TABLE S_TEST  
(ID          NUMBER(7)  
CONSTRAINT S_TEST_ID_PK PRIMARY KEY,  
NAME        VARCHAR2(25)  
CONSTRAINT S_TEST_NAME_NN NOT NULL,  
  REGION_ID NUMBER(7)  
CONSTRAINT S_TEST_REGION_ID_FK REFERENCES  
S_REGION(ID),  
START_DATE  DATE DEFAULT SYSDATE)  
/
```



```

TITLE
VARCHAR2(25)
DEPT_ID NUMBER(7)
SALARY
NUMBER(11,2)
COMMISSION_PCT NUMBER(4,2)

```

Veja os dados da nova tabela.

```

SQL> select id,last_name
      2   from EMP_41;

```

```

      ID LAST_NAME
-----
      2 Ngao
      6 Urguhart
     16 Maduro
     17 Smith

```

Observe que os dados abaixo são os mesmos da tabela criada que foram justamente criada a partir do departamento 41.

```

SQL>  select id,last_name
      2   from s_emp
      3  where dept_id = 41;

```

```

      ID LAST_NAME
-----
      2 Ngao
      6 Urguhart
     16 Maduro
     17 Smith

```

Como descobrir quais são as tabelas que estão a nossa disposição:

```

select object_name from user_Objects
where object_type = 'TABLE';

```

**FIM DE CAPÍTULO**

Finalidade do Capítulo:  
Demonstrar o uso do comando DELETE:

O Cmdo **DELETE** serve para delatarmos registros em uma tabela, usamos o **DELETE** from e o nome da tabela. Neste exemplo abaixo estamos querendo deletar todos os registros da tabela s\_emp que tenham como start\_date o dia 01.01.1996, para isso temos que usar na clausula **WHERE** o **TO\_DATE** colocando o valor do que nós estamos querendo e seu correspondente dia, mês e ano conforme o formato, no caso não deletou nenhum registro porque não existia nenhum que satisfazia as condições.

Vamos ao exemplo:

```
SQL> EDIT
Wrote file afiedt.buf
  1  DELETE FROM s_emp
  2  WHERE START_DATE >
  3* TO_DATE ('01.01.1996','DD.MM.YYYY')
SQL> /
```

0 rows deleted

Agora estamos tentando deletar um registro que existe, como o registro existente é uma FK que logicamente existe em outra tabela não se aceita a deleção, veja a mensagem de erro de violação de **constraint**.

```
SQL> DELETE FROM s_emp
2  WHERE START_DATE > TO_DATE ('01.01.1991','dd.mm.yyyy');
DELETE FROM s_emp
      *
```

```
ERROR at line 1:
ORA-02292: integrity constraint
(GUIMA.S_WAREHOUSE_MANAGER_ID_FK)
violated - child record found
```

Para deletar todos os registros de uma tabela procede-se da seguinte forma:

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

25,000 rows deleted.

Para confirmarmos a deleção usamos o comando **select** na table deletada.

```
SELECT * FROM  
2 test;
```

no rows selected;

**FIM DE CAPÍTULO**

Finalidade do Capítulo é Mostrar com Trabalhar com o Dicionário de Dados.

O dicionário de dados serve dentre outras coisas para possibilitar a visualização de quais objetos que estão disponíveis para manipulação, por exemplo: se temos acesso a **tabelas, views** ou etc.

No exemplo abaixo estamos fazendo um select na tabela **user\_objects** usando o comando '**distinct**' com a finalidade de vermos quais os objetos a serem manipulados.

Vamos ao exemplo:

```
SQL> SELECT DISTINCT OBJECT_TYPE
      2 FROM USER_OBJECTS;
```

```
OBJECT_TYPE
-----
INDEX
SEQUENCE
TABLE
/
```

Para vermos quais são as tabelas ou outros objetos que estão a nossa disposição, ou seja aqueles que temos permissão para manipulá - los temos que fazer um **select** no dicionário de dados, incluindo neste o **object\_name** na tabela **user\_objects** usando o tipo desejado '**table**' ou '**view**' e etc.

```
SELECT OBJECT_NAME
FROM USER_OBJECTS
WHERE OBJECT_TYPE = 'TABLE'
/
```

```
OBJECT_NAME
-----
FUNCIONÁRIO
SEÇÃO
S_CUSTOMER
S_DEPT
S_EMP
S_IMAGE
S_INVENTORY
S_ITEM
S_LONGTEXT
S_ORD
S_PRODUCT
```

```
S_REGION
S_TITLE
S_WAREHOUSE
```

14 rows selected.

Ainda usando o dicionário de dados podemos verificar quais são as **constraints** de uma tabela, é bom lembrar que estamos fazendo uma consulta na tabela **USER\_CONSTRAINTS** que assim como a **USER\_OBJECTS** pertence ao dicionário de dados.

```
SQL> select constraint_name,constraint_type,search_condition,
         r_constraint_name
        2   from user_constraints
        3   where table_name = 'S_EMP';
```

```
CONSTRAINT C SEARCH_CONDITION      R_CONSTRAINT_NAME
-----
S_EMP_ID_N C ID IS NOT NULL
N

S_EMP_LAST C LAST_NAME IS NOT N
_NAME_NN   ULL

S_EMP_USER C USERID IS NOT NULL
ID_NN

S_EMP_ID_P P
K

S_EMP_USER U
ID_UK

S_EMP_COMM C commission_pct IN
SSION_PCT   (10, 12.5, 15, 17.
_CK         5, 20)

S_EMP_MANA R                      S_EMP_ID_PK
GER_ID_FK

CONSTRAINT C SEARCH_CONDITION      R_CONSTRAINT_NAME
-----
S_EMP_DEPT R                      S_DEPT_ID_PK
```

\_ID\_FK

S\_EMP\_TITL R  
E\_FK

S\_TITLE\_TITLE\_PK

9 rows selected.

Tentando refinar ainda mais o exemplo acima vamos verificar quais são as constraints referentes as colunas de uma determinada tabela que no caso é a s\_emp.

Vamos ao Exemplo:

SQL>

```
SELECT CONSTRAINT_NAME,COLUMN_NAME  
FROM USER_CONS_COLUMNS  
WHERE TABLE_NAME ='S_EMP'
```

CONSTRAINT_NAME	COLUMN_NAME
S_EMP_COMMISSION_PCT_CK	COMMISSION_PCT
S_EMP_DEPT_ID_FK	DEPT_ID
S_EMP_ID_NN	ID
S_EMP_ID_PK	ID
S_EMP_LAST_NAME_NN	LAST_NAME
S_EMP_MANAGER_ID_FK	MANAGER_ID
S_EMP_TITLE_FK	TITLE
S_EMP_USERID_NN	USERID
S_EMP_USERID_UK	USERID

9 rows selected.

## FIM DE CAPÍTULO

Finalidade deste capítulo é apresentar o uso de funções de grupo do **cmdo group by**, **funções de grupo MAX,MIN,SUM,AVG** e

ainda o uso do **HAVING**:

Dica importante: geralmente o que está dentro do comando **select** deve estar no **group by**.

Neste exemplo estamos vendo o uso da função **AVG** que tem a finalidade de trazer a média de uma determinada soma. Estamos também usando a função **MAX** e **MIN** que tem como função trazer o máximo e o mínimo valor. Também temos a função **SUM** que faz a soma de valores de colunas, todas essas funções são funções de grupo. Ainda no exemplo abaixo temos o uso da função **UPPER** que transforma o conteúdo a ser pesquisado e a função **LIKE** que faz a pesquisa somente nos registros que comecem por **'sales'**.

```
SQL> select avg(SALARY),MAX(SALARY),MIN(SALARY),
2          SUM(SALARY)
3 FROM     S_EMP
4 WHERE    UPPER(TITLE) LIKE 'SALES%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1476	1525	1400	7380

Neste exemplo abaixo, temos o menor nome e o maior nome dentro da ordem alfabética, demonstrando que não somente os valores numéricos que são manipuláveis com as funções **MIN** e **MAX**, é bom salientar que as datas também são passíveis de manipulação.

Vamos ao exemplo:

```
SQL> select MIN(LAST_NAME),MAX(LAST_NAME)
2 FROM S_EMP;
```

MIN(LAST_NAME)	MAX(LAST_NAME)
Biri	Velasquez

No exemplo abaixo estamos tentando selecionar o ID de departamento igual a 41 trazendo seu numero de repetições, mas não dá certo pois a função **COUNT** é de grupo e não estamos usando o **GROUP BY**.

Vamos ao exemplo:

```
SQL> select dept_id, count(*) "number"
      2   from      s_emp
      3   where    dept_id = 41;
select dept_id, count(*) "number"
      *
```

ERROR at line 1:

ORA-00937: not a single-group group function

Fazendo o **select** abaixo temos quantas vezes o dept 41 se repete.

```
SQL> select dept_id
      2   from      s_emp
      3   where    dept_id = 41;
```

```
DEPT_ID
-----
      41
      41
      41
      41
```

Agora usando a função de grupo **COUNT(\*)** que no caso esta contando a quantidade de registros onde o dept\_id é igual a 41 ou seja quantas vezes ele se repete, teremos o grupamento feito. O que está no select deve estar no group by.

Vamos ao exemplo:

```
SQL> select dept_id, count(*) "number"
      2   from      s_emp
      3   where    dept_id = 41
      4   group by dept_id;
```

```
DEPT_ID      number
-----
      41              4
```

Agora estamos contando quantos registros temos para cada CREDIT\_RATING da tabela s\_customer.

```
SQL> select credit_rating, count(*) "# custo"
      2   from    s_customer
      3   group by credit_rating;
```

CREDIT_RA	# custo
EXCELLENT	9
GOOD	3
POOR	3

No próximo exemplo é bom destacar o uso do **NOT LIKE** onde determina que não seja trazido registros parecidos com uma certa condição. Veja também o uso da função de grupo **SUM** que trás a soma dos salários.

```
SQL> select title, sum(salary) soma_total
      2   from    s_emp
      3   where  title not like 'VP%'
      4   group by title
      5   order by
      6   sum(salary);
```

TITLE	SOMA_TOTAL
President	2500
Warehouse Manager	6157
Sales Representative	7380
Stock Clerk	9490

Um outro exemplo do uso de ORDER BY, onde estamos grupando por Title.

```
SQL> select title, max(salary)
      2   from    s_emp
      3   group by title;
```

TITLE	MAX(SALARY)
President	2500
Sales Representative	1525
Stock Clerk	1400
VP, Administration	1550
VP, Finance	1450

VP, Operations	1450
VP, Sales	1400
Warehouse Manager	1307

8 rows selected.

Nossa intenção no select abaixo é fazermos a média dos salários dos campos da coluna SALARY da tabela s\_emp, e trazer a tela somente a média que der acima de 2000. Não dá certo porque não usamos a clausula **where** dentro do **group by** mas sim a função **HAVING**, isso quando queremos trazer dados a tela obedecendo uma certa condição.

```
SQL> SELECT DEPT_ID,AVG(SALARY)
2 FROM S_EMP
3 WHERE AVG(SALARY) > 2000
4 GROUP BY DEPT_ID;
WHERE AVG(SALARY) > 2000
*
```

ERROR at line 3:  
ORA-00934: group function is not allowed here

Veja a maneira correta de se fazer o exercício anterior onde usamos a função **HAVING** que se dá da seguinte forma , sempre após o group by.

```
SQL> SELECT DEPT_ID,AVG(SALARY)
2 FROM S_EMP
3 GROUP BY DEPT_ID
4 HAVING AVG(SALARY) > 2000;
```

```
DEPT_ID AVG(SALARY)
-----
50          2025S
```

No exemplo abaixo estamos selecionando, contando e separando em grupos os departamentos isso pelo dept\_id e depois o title, perceba a seqüência do grupo.

```
SQL> select dept_id,title, count(*)
2 from s_emp
3 group by dept_id,title;
```

```
DEPT_ID TITLE COUNT(*)
-----
```

10	VP, Finance	1
31	Sales Representative	1
31	VP, Sales	1
32	Sales Representative	1
33	Sales Representative	1
34	Sales Representative	1
34	Stock Clerk	1
35	Sales Representative	1
41	Stock Clerk	2
41	VP, Operations	1
41	Warehouse Manager	1
42	Stock Clerk	2
42	Warehouse Manager	1
43	Stock Clerk	2
43	Warehouse Manager	1
44	Stock Clerk	1
44	Warehouse Manager	1
45	Stock Clerk	2
45	Warehouse Manager	1
50	President	1
50	VP, Administration	1

21 rows selected.

Agora estamos grupando primeiro pelo title e depois pelo dept\_id, veja a diferença.

```
SQL> select title,dept_id, count(*)
  2   from     s_emp
  3   group by title,dept_id;
```

TITLE	DEPT_ID	COUNT(*)
-----	-----	-----
President	50	1
Sales Representative	31	1
Sales Representative	32	1
Sales Representative	33	1
Sales Representative	34	1
Sales Representative	35	1
Stock Clerk	34	1
Stock Clerk	41	2
Stock Clerk	42	2
Stock Clerk	43	2
Stock Clerk	44	1
Stock Clerk	45	2
VP, Administration	50	1

VP, Finance	10	1
VP, Operations	41	1
VP, Sales	31	1
Warehouse Manager	41	1
Warehouse Manager	42	1
Warehouse Manager	43	1
Warehouse Manager	44	1
Warehouse Manager	45	1

Observe o uso do **HAVING** em substituição a clausula **WHERE**, além de outros **SELECTS**.

Quando quisermos trazer um registro que inicie por uma letra qualquer temos podemos usamos o cmdo **LIKE** procedendo da seguinte forma:

```
SQL> select last_name, title
1   from s_emp
2  where last_name like 'V%'
```

LAST_NAME	TITLE
-----	-----
Velasquez	President

No exemplo abaixo estamos fazendo a seleção de todos os cargos de funcionários existentes assim como contando seus componentes **COUNT** e ainda trazendo as suas respectivas medias salariais usando a função **AVG**, multiplicadas por 12.

```
SQL> SELECT TITLE, 12 * AVG(SALARY) " Salário Anual",
2   count(*) " numero de funcionários"
3   from s_emp
4   group by title;
```

TITLE	Salário Anual	numero de
funcionários		
President	\$30,000.00	1
Sales Representative	\$17,712.00	5
Stock Clerk	\$11,388.00	10

```

VP, Administration          $18,600.00
1
VP, Finance                 $17,400.00
1
VP, Operations             $17,400.00
1
VP, Sales                  $16,800.00
1
Warehouse Manager         $14,776.80
5

```

8 rows selected.

Um pouco diferente do exemplo anterior o abaixo está primeiramente formatando a coluna **Salário Anual** para receber valores após ter sido feito um select em **TITLE**. Estamos multiplicando por 12 a soma da média dos salários feita pela função **AVG**, e então usamos a função **COUNT** para contar o numero de funcionários de s\_emp grupados por **title** e que tenha no máximo 2 funcionários, olhando para o exemplo anterior podemos entender melhor e também destacar que o **HAVING** está fazendo a função de um **WHERE** dentro do **GROUP BY**.

Vamos ao Exemplo:

```

SQL> column " Salário Anual" FORMAT $99,999.99
SQL> SELECT TITLE, 12 * AVG(SALARY) " Salário Anual",
2   count(*) " numero de funcionários"
3   from s_emp
4   group by title
5   having count(*) > 2;

```

TITLE	Salário Anual	numero de funcionários
Sales Representative	\$17,712.00	5
Stock Clerk	\$11,388.00	10
Warehouse Manager	\$14,776.80	5

Mais um exemplo do uso do HAVING:

```

SQL> select title,sum(salary) soma
2   from s_emp

```

```
3  where title not like 'VP%'
4  group by title
5  having sum(salary) > 5000
6  order by sum(salary);
```

TITLE	SOMA
-----	-----
Warehouse Manager	6157
Sales Representative	7380
Stock Clerk	9490

**FIM DE CAPÍTULO**

Finalidade do capítulo é demonstrar como criar **GRANTS**.

Para criar um novo usuário (isto é executado pelo DBA) devemos proceder da seguinte forma, "**NOVO**" é o nome do usuário e **IDENTIFIED BY "tiger"** é sua senha.

```
SQL> CREATE USER NOVO
      2 IDENTIFIED BY TIGER;
```

User created.

Agora após termos criado um novo usuário vamos dar-lhe os seus **GRANTS** ou seja as permissões devidas como, **delete, update** e etc, veja como se processa isto e veja também a sintaxe:

```
SQL> GRANT create table, create sequence, create view
      2 to NOVO;
```

Grant succeeded.

Criando uma **ROLE** de privilégios para ser usada para vários usuários, uma **role** é o conjunto de privilégios que guardamos armazenados com a finalidade de proporcionar uma facilitação de manuseio na hora de se conceder privilégios, evitando a digitação de todos os privilégios, passando a ter que digitar somente o nome da **role**. No exemplo, o nome da **ROLE** é "**manager**" e por enquanto não existe nenhum privilégio dentro dela.

```
SQL> CREATE ROLE manager;
```

Role created.

Agora estamos definindo os privilégios para a **ROLE "manager"**, que são: criar tabela e view. Observe como se processa isto:

```
SQL> GRANT create table, create view TO manager;
```

Grant succeeded.

Agora que nossa **ROLE** já tem os privilégios definidos, no exemplo abaixo vou passá-la para algum usuário. No caso o usuário "**NOVO**" está recebendo os **GRANTS** da **ROLE MANAGER**, que são somente criar tabela e view:

```
SQL> GRANT MANAGER TO NOVO;
```

Grant succeeded

Como mudar a senha de um usuário: no caso estamos mudando do user **novο** que tinha como "senha" ou seja **IDENTIFIED** igual a **TIGER** passando a ser **LION**.

```
SQL> ALTER USER NOVO IDENTIFIED BY LION;
```

User altered.

Agora estamos dando um novo **GRANT** para o USER "NOVO" referente a tabela s\_emp, neste caso o **GRANT** é só de consulta.

```
SQL> GRANT select
  2  ON s_emp
  3  TO novo;
```

Grant succeeded.

O exemplo abaixo mostra como adicionar um novo **GRANT** a uma **ROLE**, relacionado com colunas específicas de uma tabela no caso s\_dept e atribuindo a permissão de **update** para o usuário "NOVO" e para a **ROLE** "manager".

```
SQL> GRANT update(name,region_id)
  2  ON      s_dept
  3  TO      novo,manager;
```

Grant succeeded.

Este exemplo mostra como dar GRANT para um USER em uma tabela e este usuário ter permissão de passar este mesmo GRANT para outros usuários.

```
SQL> GRANT select
  2  ON      s_emp
  3  TO      scott
  4  WITH    GRANT OPTION;
```

Grant succeeded.

Agora estou tornando publico de todos o SELECT a tabela S\_DEPT referente ao usuário GUIMA, ou seja todos podem dar **select** a tabela s\_dept do usuário GUIMA.

```
SQL> GRANT SELECT
      2 ON      GUIMA.S_DEPT
      3 TO      PUBLIC;
```

Grant succeeded.

Para sabermos quais privilégios nós possuímos podemos acessar no **DICIONÁRIO DE DADOS** nas seguintes tabelas:

```
ROLE_SYS_PRIVS
ROLE_TAB_PRIVS
USER_ROLE_PRIVS
USER_TAB_PRIVS_MADE
USER_TAB_PRIVS_RECD
USER_COL_PRIVS_MADE
USER_COL_PRIVS_RECD
```

Como ver as **ROLES** que eu tenho acesso:

```
SQL> SELECT GRANTED_ROLE, OS_GRANTED
      2 FROM USER_ROLE_PRIVS
      3 WHERE USERNAME = 'GUIMA';
```

GRANTED_ROLE	OS_
CONNECT	NO
DBA	NO
MANAGER	NO

Para tirar os privilégios de um USER procede-se da seguinte forma:

```
SQL> REVOKE select
      2 ON s_emp
      3 FROM NOVO;
```

Revoke succeeded.

No exemplo acima estamos tirando os privilégios de **select** do USER "NOVO".

Como criar sinônimo:

Criando um sinônimo para uma tabela: no caso está sendo criado um sinônimo com o nome "**HXXH**" para a tabela **s\_dept** pertencente a **alice**:

```
SQL> CREATE SYNONYM HXXH
      FOR      alice.s_dept;
Synonym created.
```

Agora estamos criando um sinônimo para uma **VIEW**, nossa **VIEW** se chama **dept\_sum\_vu** e o nome do sinônimo criado é **d\_sum**:

```
SQL> create SYNONYM d_sum
      2 for      dept_sum_vu;

Synonym created.
```

Agora estamos criando um sinônimo publico, o nome dele é "**DDD**" e é referente a tabela **s\_dept** de **alice**:

```
SQL> create public SYNONYM DDD
      2 FOR      alice.s_dept;

Synonym created.
```

Para dropar um sinônimo é simples:

```
SQL> DROP SYNONYM d_sum;

Synonym dropped.
```

**FIM DE CAPÍTULO**

Finalidade do Capítulo é Tratar Sobre **INDEX** e **JOIN** e **OUTER JOIN**:

**INDEX**:

Uma **INDEX** pode ser criada automaticamente (quando é criada uma **PK** ou **UK** constraint em uma tabela) ou manualmente. Para delatarmos uma **INDEX** temos que usar o cmdo **DROP INDEX**:

Vamos ao Exemplo:

```
SQL> DROP INDEX s_emp_last_name_idx;
```

Index dropped.

Para criar uma **INDEX** usa-se o cmdo **CREATE INDEX**:

Vamos ao Exemplo:

```
SQL> CREATE INDEX S_EMP_last_name_idx  
2 on S_EMP(last_name);
```

Index created.

Para selecionar as **INDEX** de uma tabela faz-se um **SELECT** na tabela **USER\_INDEXES** e **USER\_IND\_COLUMNS**:

O uso do **JOIN**:

Join é a ligação que fazemos entre duas tabelas na pesquisa de dados, necessariamente deve existir em um join a chave primaria fazendo relação com uma chave estrangeira, esta é a condição e ligação.

No exemplo abaixo estamos selecionando o **last\_name** e **dept\_id** da tabela **s\_emp** e também selecionando a coluna **name** da tabela **s\_dept** isto onde a coluna **id** de **s\_dept** for igual a coluna **dept\_id** de **s\_emp**, completando assim a condição do **JOIN**.

Vamos ao exemplo:

```
SQL> SELECT S_EMP.LAST_NAME, S_EMP.DEPT_ID, S_DEPT.NAME  
2 FROM S_EMP, S_DEPT  
3 WHERE S_EMP.DEPT_ID = S_DEPT.ID;
```

LAST_NAME	DEPT_ID	NAME
Velasquez	50	Administration
Ngao	41	Operations
Nagayama	31	Sales
Quick-To-See	10	Finance
Ropeburn	50	Administration
Urguhart	41	Operations
Menchu	42	Operations
Biri	43	Operations
Catchpole	44	Operations
Havel	45	Operations
Magee	31	Sales
Giljum	32	Sales
Sedeghi	33	Sales
Nguyen	34	Sales
Dumas	35	Sales
Maduro	41	Operations
Smith	41	Operations
Nozaki	42	Operations
Patel	42	Operations
Newman	43	Operations
Markarian	43	Operations

No exemplo abaixo como no anterior temos a realização de um **JOIN**.

Vamos ao exemplo.

```
SQL> SELECT S_DEPT.ID " COD DO DEPT",
2 S_REGION.ID "COD DA REG",
3 S_REGION.NAME"NOME DA REGIÃO"
4 FROM S_DEPT,S_REGION
5 WHERE S_DEPT.REGION_ID = S_REGION.ID;
```

COD DO DEPT	COD DA REG	NOME DA REGIÃO
10	1	North America
31	1	North America
32	2	South America
33	3	Africa / Middle East
34	4	Ásia
35	5	Europe
41	1	North America
42	2	South America

```

43          3 Africa / Middle East
44          4 Ásia
45          5 Europe
50          1 North America

```

12 rows selected.

No exemplo abaixo temos mais um exemplo de **join** onde temos o uso de apelidos para as tabelas manipuladas, é importante observar como é usado o apelido e como este tem que ser referenciado no **FROM**, no exemplo abaixo vemos o apelido de **"E"** para a tabela **S\_EMP**, de **"D"** para a tabela **S\_DEPT**. Outra característica importante no exemplo é o uso do **JOIN** onde relacionamos as duas tabelas através da chave estrangeira. O uso do **INITCAP** serve para procurarmos o registro que comece com a letra maiúscula

```

SQL> select E.last_name,E.dept_id,
2   D.name
3   from s_emp E,s_dept D
4   where e.dept_id = D.id
5   and initcap(E.last_name)='Menchu';

```

```

LAST_NAME          DEPT_ID NAME
-----
Menchu              42 Operations

```

No exemplo abaixo temos a realização de dois **JOINS**, uso do **AND** no exemplo abaixo proporciona a adição de mais de uma condição ou seja além do **join** entre **s\_emp** e **s\_dept** também ocorre um **Join** entre **s\_region** e **s\_dept** proporcionado pelo uso do **AND**. Também está sendo usado um novo **AND** colocando a **commission\_pct** da tabela **s\_emp** > 0.

Vamos ao exemplo:

```

SQL> select E.last_name,R.name,E.commission_pct
2   from s_emp E, s_dept D, s_region R
3   where E.dept_id = D.id
4   and D.region_id = R.id
5   and E.commission_pct > 0;

```

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

No exemplo abaixo estamos mostrando um join que traz todos os registros que satisfaçam a condição da cláusula **where** ou seja traz somente aquele que satisfaça o **JOIN** não traz os registros que contem espaços em branco, em função da relação entre as duas tabelas no caso entre s\_emp e s\_customer. Mas e se quisermos que pesquisar inclusive os registros que não tem relacionamento, ou seja que estejam com o espaço em branco? Para isso temos que acrescentarmos o sinal de (+).

Vamos ao Exemplo:

```
SQL> select e.last_name,e.id,c.name
      2   from s_emp e, s_customer c
      3   where e.id = c.sales_rep_id
      4   order by e.id;
```

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

O exemplo a seguir é idêntico ao anterior com a diferença que está acrescido do sinal de (+) para, mesmo satisfazendo a clausula **where**, traga os registros que não tenham satisfeito a condição de relacionamento da clausula where, no caso, estamos então fazendo um **OUTER JOIN** pegando o **LAST\_NAME** e o **ID** da tabela **s\_emp** e o **NAME** da tabela **s\_customer** onde o **id** da tabela **s\_emp** corresponda ao **sales\_rep\_id** dentro da tabela **s\_customer** e com o sinal de (+) traga também os **NAME** de **s\_customer** que não tenham relação com **s\_emp**.

```
SQL> select e.last_name,e.id,c.name
      2  from s_emp e, s_customer c
      3  where e.id (+) = c.sales_rep_id
      4  order by e.id;
```

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

15 rows selected.

## FINAL DE CAPÍTULO

Finalidade do capítulo é mostrar o uso do **BETWEEN**.

No exemplo a seguir estamos selecionando as colunas **ename, job, sal** da tabela **EMP** e a coluna **GRADE** da tabela **SALGRADE**, onde através do comando **BETWEEN AND** ocorre uma pesquisa na tabela **SALGRADE** nas colunas **LOSAL** e **HISAL** onde comparamos seus valores com os valores dos campos da coluna **SAL** de **EMP** trazendo somente aqueles valores que estejam entre os de **LOSAL** e **HISAL**. No exemplo não ocorre um **JOIN** mas uma simples pesquisa em uma outra tabela para que seja efetuada uma comparação.

```
SQL> select e.ename,e.job,e.sal,s.grade
      2   from emp e, salgrade s
      3   where e.sal between s.losal and s.hisal
```

ENAME	JOB	SAL	GRADE
SMITH	CLERK	800	1
ADAMS	CLERK	1100	1
JAMES	CLERK	950	1
WARD	SALESMAN	1250	2
MARTIN	SALESMAN	1250	2
MILLER	CLERK	1300	2
ALLEN	SALESMAN	1600	3
TURNER	SALESMAN	1500	3
JONES	MANAGER	2975	4
BLAKE	MANAGER	2850	4
CLARK	MANAGER	2450	4
SCOTT	ANALYST	3000	4
FORD	ANALYST	3000	4
KING	PRESIDENT	5000	5

14 rows selected.

## FINAL DE CAPÍTULO

Finalidade do capítulo é mostrar o uso do **ORDER BY**.

O exemplo abaixo mostra como usar o comando **order by**, onde no caso estamos ordenando a tabela `s_emp` pelo o **last\_name**, é bom lembrar que sempre o comando **order by** vem por último na função `sql`.

Vamos ao exemplo:

```
SELECT LAST_NAME, DEPT_ID, START_DATE
FROM S_EMP
ORDER BY LAST_NAME
```

LAST_NAME	DEPT_ID	START_DAT
Biri	43	07-APR-90
Catchpole	44	09-FEB-92
Chang	44	30-NOV-90
Dancs	45	17-MAR-91
Dumas	35	09-OCT-91
Giljum	32	18-JAN-92
Havel	45	27-FEB-91
Maduro	41	07-FEB-92
Magee	31	14-MAY-90
Markarian	43	26-MAY-91
Menchu	42	14-MAY-90
Nagayama	31	17-JUN-91
Newman	43	21-JUL-91
Ngao	41	08-MAR-90
Nguyen	34	22-JAN-92
Nozaki	42	09-FEB-91
Patel	42	06-AUG-91
Patel	34	17-OCT-90
Quick-To-See	10	07-APR-90
Ropeburn	50	04-MAR-90
Schwartz	45	09-MAY-91

LAST_NAME	DEPT_ID	START_DAT
Sedeghi	33	18-FEB-91
Smith	41	08-MAR-90
Urguhart	41	18-JAN-91
Velasquez	50	03-MAR-90

25 rows selected.

No comando sql **order by** temos a função **desc** que vem com a finalidade de colocar os dados dentro da coluna em ordem decrescente. No exemplo a seguir estamos colocando por ordem decrescente a coluna **start\_date** da tabela **s\_emp**.

```
SQL> SELECT LAST_NAME, DEPT_ID, START_DATE
       2 FROM S_EMP
       3 ORDER BY START_DATE DESC;
```

LAST_NAME	DEPT_ID	START_DAT
Catchpole	44	09-FEB-92
Maduro	41	07-FEB-92
Nguyen	34	22-JAN-92
Giljum	32	18-JAN-92
Dumas	35	09-OCT-91
Patel	42	06-AUG-91
Newman	43	21-JUL-91
Nagayama	31	17-JUN-91
Markarian	43	26-MAY-91
Schwartz	45	09-MAY-91
Dancs	45	17-MAR-91
Havel	45	27-FEB-91
Sedeghi	33	18-FEB-91
Nozaki	42	09-FEB-91
Urguhart	41	18-JAN-91
Chang	44	30-NOV-90
Patel	34	17-OCT-90
Menchu	42	14-MAY-90
Magee	31	14-MAY-90
Quick-To-See	10	07-APR-90
Biri	43	07-APR-90

LAST_NAME	DEPT_ID	START_DAT
Ngao	41	08-MAR-90
Smith	41	08-MAR-90
Ropeburn	50	04-MAR-90
Velasquez	50	03-MAR-90

25 rows selected.

Agora vamos mostrar os dados do resultado do mesmo select anterior sem o uso da função **desc** para o **order by**, observe a diferença da ordenação em relação as datas.

```
SQL> SELECT LAST_NAME, DEPT_ID, START_DATE
       2 FROM S_EMP
       3 ORDER BY START_DATE;
```

LAST_NAME	DEPT_ID	START_DAT
Velasquez	50	03-MAR-90
Ropeburn	50	04-MAR-90
Ngao	41	08-MAR-90
Smith	41	08-MAR-90
Quick-To-See	10	07-APR-90
Biri	43	07-APR-90
Menchu	42	14-MAY-90
Magee	31	14-MAY-90
Patel	34	17-OCT-90
Chang	44	30-NOV-90
Urguhart	41	18-JAN-91
Nozaki	42	09-FEB-91
Sedeghi	33	18-FEB-91
Havel	45	27-FEB-91
Dancs	45	17-MAR-91
Schwartz	45	09-MAY-91
Markarian	43	26-MAY-91
Nagayama	31	17-JUN-91
Newman	43	21-JUL-91
Patel	42	06-AUG-91
Dumas	35	09-OCT-91

LAST_NAME	DEPT_ID	START_DAT
Giljum	32	18-JAN-92
Nguyen	34	22-JAN-92
Maduro	41	07-FEB-92
Catchpole	44	09-FEB-92

25 rows selected.

O exemplo seguinte mostra como usar a posição da coluna dentro do comando sql para definir a ordenação dos registros por aquela coluna, no caso abaixo estamos ordenando nossos dados pela coluna numero 4 que corresponde a **START\_DATE** de **S\_EMP**.

Vamos ao exemplo:

```
SQL> SELECT ID, LAST_NAME, FIRST_NAME, START_DATE
      2  MANAGER_ID, SALARY
      3  FROM S_EMP
      4  ORDER BY 4;
```

ID	LAST_NAME	FIRST_NAME	MANAGER_I	SALARY
1	Velasquez	Carmen	03-MAR-90	2500
5	Ropeburn	Audry	04-MAR-90	1550
2	Ngao	LaDoris	08-MAR-90	1450
17	Smith	George	08-MAR-90	940
4	Quick-To-See	Mark	07-APR-90	1450
8	Biri	Ben	07-APR-90	1100
7	Menchu	Roberta	14-MAY-90	1250
11	Magee	Colin	14-MAY-90	1400
23	Patel	Radha	17-OCT-90	795
22	Chang	Eddie	30-NOV-90	800
6	Urguhart	Molly	18-JAN-91	1200
18	Nozaki	Akira	09-FEB-91	1200
9	Catchpole	Antoinette	09-FEB-92	1300

25 rows selected.

Agora além de ordenar pela coluna numero 4 estamos também colocando em ordem decrescente usando para isto o cmndo **desc**.

```
SQL> SELECT ID, LAST_NAME, FIRST_NAME, START_DATE
      2  MANAGER_ID, SALARY
      3  FROM S_EMP
      4  ORDER BY 4 DESC;
```

ID	LAST_NAME	FIRST_NAME	MANAGER_I	SALARY
9	Catchpole	Antoinette	09-FEB-92	1300
16	Maduro	Elena	07-FEB-92	1400
14	Nguyen	Mai	22-JAN-92	1525
12	Giljum	Henry	18-JAN-92	1490

No próximo exemplo o comando **order by** esta agrupando por departamento com o **dept\_id** e depois por **salário** (dentro do grupo departamento) podemos constatar a melhor verificação do exemplo no departamento 41 onde o salário vem por ordem decrescente.

```
SQL> SELECT LAST_NAME,DEPT_ID,SALARY
       2 FROM S_EMP
       3 ORDER BY DEPT_ID,SALARY DESC;
```

LAST_NAME	DEPT_ID	SALARY
Quick-To-See	10	1450
Nagayama	31	1400
Magee	31	1400
Giljum	32	1490
Sedeghi	33	1515
Nguyen	34	1525
Patel	34	795
Dumas	35	1450
Ngao	41	1450
Maduro	41	1400
Urguhart	41	1200
Smith	41	940
Menchu	42	1250
Nozaki	42	1200
Patel	42	795
Biri	43	1100
Markarian	43	850
Newman	43	750
Catchpole	44	1300
Chang	44	800
Havel	45	1307

LAST_NAME	DEPT_ID	SALARY
Schwartz	45	1100
Dancs	45	860
Velasquez	50	2500
Ropeburn	50	1550

25 rows selected.

Observação: se colocarmos duas ordenações incompatíveis de execução, o sql irá reconhecer a primeira da seqüência, observe que o exemplo abaixo mostra claramente isto. Observe também o departamento 41, onde o sql ordena primeiro por

departamento e depois por nome (last\_name) e despreza a coluna salary porque dentro da prioridade ela é a última.

```
SQL> SELECT LAST_NAME, DEPT_ID, SALARY
       2 FROM S_EMP
       3 ORDER BY DEPT_ID, LAST_NAME, SALARY DESC;
```

LAST_NAME	DEPT_ID	SALARY
Quick-To-See	10	1450
Magee	31	1400
Nagayama	31	1400
Giljum	32	1490
Sedeghi	33	1515
Nguyen	34	1525
Patel	34	795
Dumas	35	1450
Maduro	41	1400
Ngao	41	1450
Smith	41	940
Urguhart	41	1200
Menchu	42	1250
Nozaki	42	1200
Patel	42	795
Biri	43	1100
Markarian	43	850
Newman	43	750
Catchpole	44	1300
Chang	44	800
Dancs	45	860

LAST_NAME	DEPT_ID	SALARY
Havel	45	1307
Schwartz	45	1100
Ropeburn	50	1550
Velasquez	50	2500

25 rows selected.

## FINAL DE CAPÍTULO

Finalidade do capítulo é mostrar o uso do **LENGTH** e também como **concatenar**.

O uso do **length**: serve par contar a quantidade de espaços, incluindo caracteres que um determinado registro ocupa.

O uso do "**substr**" : serve para localizar de forma física determinadas posições de um determinado registro.

No exemplo abaixo vemos como que o cmdo "**substr**" ajuda a encontrarmos determinadas posições dentro de uma clausula where.

```
SQL> SELECT name, LENGTH(name)
      2 FROM S_PRODUCT
      3 WHERE SUBSTR(NAME,1,5) = 'Black';
```

NAME	LENGTH(NAME)
Black Hawk Knee Pads	20
Black Hawk Elbow Pads	21

Concatenando colunas distantes usando o **CONCAT**, observe que estamos concatenado duas colunas e jogando-as dentro de uma mesma coluna:

```
SQL> SELECT CONCAT(NAME,COUNTRY) CLIENTE
      2 FROM S_CUSTOMER
      3 WHERE UPPER (CREDIT_RATING) = 'GOOD'
      4 ;
```

CLIENTE
Delhi SportsIndia
Sweet Rock SportsNigeria
Muench SportsGermany

## FINAL DE CAPÍTULO

Finalidade do capítulo é mostrar algumas manipulações com datas:

No próximo exemplo vemos o uso do **sysdate** que corresponde a data do sistema, que está sendo subtraído pela coluna **start\_date** e dividido por 7 para verificar quantas semanas existem entre a data do **start\_date** e **sysdate** isto referente ao departamento 43.

```
SQL> SELECT LAST_NAME, (SYSDATE-START_DATE)/7 SEMANAS
       2 FROM S_EMP
       3 WHERE DEPT_ID = 43;
```

LAST_NAME	SEMANAS
Biri	397.5229
Newman	330.38004
Markarian	338.38004

O exemplo seguinte deveria mostrar em **tenure** a quantidade de meses existentes entre **sysdate** e **start\_date** e depois usando o comdo "**add\_months**" mostrar em **review** a data correspondente a 6 meses a mais de **start\_date**, mas não mostra porque não satisfaz a clausula **where** pois pede que traga os dados entre o **sysdate** e o **start\_date** em 48 meses, coisa que não existe:

Vamos ao Exemplo:

```
SQL> SELECT ID, START_DATE,
       2 MONTHS_BETWEEN( SYSDATE-START_DATE) TENURE,
       3 ADD_MONTHS(START_DATE,6) REVIEW
       4 FROM S_EMP
       5 WHERE MONTHS_BETWEEN (SYSDATE,START_DATE)<48;
```

no rows selected

Agora no exemplo abaixo, satisfazendo a clausula **where** da quantidade de meses na função **between** podemos ver na coluna **tenure** a quantidade de meses entre o **sysdate** e **start\_date** e em **review** mostra o **start\_date** acrescido de 6 meses.

Vamos ao Exemplo:

```
SQL> SELECT ID, START_DATE,  
2 MONTHS_BETWEEN(SYSDATE, START_DATE) TENURE,  
3 ADD_MONTHS(START_DATE, 6) REVIEW  
4 FROM S_EMP  
5 WHERE MONTHS_BETWEEN (SYSDATE, START_DATE) < 72;
```

ID	START_DAT	TENURE	REVIEW
9	09-FEB-92	69.311978	09-AUG-92
12	18-JAN-92		70 18-JUL-92
14	22-JAN-92	69.892624	22-JUL-92
16	07-FEB-92	69.376495	07-AUG-92

Uma nova versão do exemplo anterior:

```
SQL> SELECT ID, START_DATE,  
2 MONTHS_BETWEEN(SYSDATE, START_DATE) TENURE,  
3 ADD_MONTHS(START_DATE, 6) REVIEW  
4 FROM S_EMP  
5 WHERE MONTHS_BETWEEN (SYSDATE, START_DATE) < 84;
```

ID	START_DAT	TENURE	REVIEW
3	17-JUN-91	77.054026	17-DEC-91
6	18-JAN-91		82 18-JUL-91
9	09-FEB-92	69.31209	09-AUG-92
10	27-FEB-91	80.731445	27-AUG-91
12	18-JAN-92		70 18-JUL-92
13	18-FEB-91		81 18-AUG-91
14	22-JAN-92	69.892736	22-JUL-92
15	09-OCT-91	73.31209	09-APR-92
16	07-FEB-92	69.376607	07-AUG-92
18	09-FEB-91	81.31209	09-AUG-91
19	06-AUG-91	75.408865	06-FEB-92
20	21-JUL-91	75.924994	21-JAN-92
21	26-MAY-91	77.763703	26-NOV-91
22	30-NOV-90	83.634671	31-MAY-91
24	17-MAR-91	80.054026	17-SEP-91
25	09-MAY-91	78.31209	09-NOV-91

16 rows selected.

A função **trunc** e **round**:

O exemplo abaixo mostra como usar a função **round** com a seguinte característica:

O **round** respeita o dia dos meses, se for maior que 15 arredonda para o início do próximo mês, se não for vai para o início do mês que se encontra.

O comando **trunc** trunca sempre para o início do mês que se encontra.

Vamos ao Exemplo:

```
SQL> SELECT ID, START_DATE,
 2  ROUND (START_DATE, 'MONTH'),
 3  TRUNC (START_DATE, 'MONTH')
 4  FROM S_EMP
 5  WHERE START_DATE LIKE '%91';
```

ID	START_DAT	ROUND (STA	TRUNC (STA
3	17-JUN-91	01-JUL-91	01-JUN-91
6	18-JAN-91	01-FEB-91	01-JAN-91
10	27-FEB-91	01-MAR-91	01-FEB-91
13	18-FEB-91	01-MAR-91	01-FEB-91
15	09-OCT-91	01-OCT-91	01-OCT-91
18	09-FEB-91	01-FEB-91	01-FEB-91
19	06-AUG-91	01-AUG-91	01-AUG-91
20	21-JUL-91	01-AUG-91	01-JUL-91
21	26-MAY-91	01-JUN-91	01-MAY-91
24	17-MAR-91	01-APR-91	01-MAR-91
25	09-MAY-91	01-MAY-91	01-MAY-91

11 rows selected.

**FINAL DE CAPÍTULO.** Finalidade do capítulo é o uso da função **MOD**.

A função **mod** traz o resto de uma divisão. Neste exemplo temos uma seleção de registros dentro da tabela **s\_emp** onde o salário é maior que 1400.

```
SQL> select LAST_NAME, MOD (SALARY, COMMISSION_PCT)
 2  FROM S_EMP
 3  WHERE SALARY > 1400;
```

LAST_NAME	MOD (SALARY, COMMISSION_PCT)
-----------	------------------------------

```
Velasquez
Ngao
Quick-To-See
Ropeburn
Giljum
2.5
Sedeghi
5
Nguyen
10
Dumas
15
```

8 rows selected.

Agora neste exemplo temos a seleção de todos os registros onde o salário é maior que 1400 e também possui comissão nula.

```
SQL> select LAST_NAME,MOD(SALARY,COMMISSION_PCT)
2 FROM S_EMP
3 WHERE SALARY>1400
4 and commission_pct is null;
```

```
LAST_NAME                                MOD(SALARY,COMMISSION_PCT)
-----
Velasquez
Ngao
Quick-To-See
Ropeburn
```

Neste exemplo temos somente aqueles que ganham mais de 1400 e possui comissão.

```
SQL> select LAST_NAME,MOD(SALARY,COMMISSION_PCT)
2 FROM S_EMP
3 WHERE SALARY>1400
4 and commission_pct is not null;
```

```
LAST_NAME                                MOD(SALARY,COMMISSION_PCT)
-----
Giljum
2.5
Sedeghi
5
```

Nguyen

10

Dumas

15

**FINAL DE CAPÍTULO**

Finalidade do capítulo é fazer arredondamentos de valores usando **"ROUND"** e o **"TRUNC"**.

Vamos ao Exemplo:

```
SQL> select round(45.923,2), round(45.923,0),  
2   round(45.923,-1)  
3   from sys.dual;
```

```
ROUND(45.923,2) ROUND(45.923,0) ROUND(45.923,-1)  
-----  
                45.92                46                50
```

```
SQL> select trunc(45.923,2), trunc(45.923,0),  
trunc(45.923,-1)  
from sys.dual
```

```
TRUNC(45.923,2) TRUNC(45.923,0) TRUNC(45.923,-1)  
-----  
                45.92                45                40
```

**FINAL DE CAPÍTULO**

Finalidade do capítulo é tratar o uso do **SELECT,NVL,DISTINCT**.

O comando **SELECT** usado no exemplo abaixo é básico, pois serve para selecionar todas as linhas da tabela s\_dept, não fazendo neste caso nenhuma distinção.

```
SQL> select * from s_dept;
```

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

12 rows selected.

Agora o comando select é usado para trazer valores de colunas específicas onde o nome das colunas vem entre vírgulas. A tabela **s\_dept** tem como colunas, **id,name** e **region\_id**, mas neste caso só está sendo trazido **id** e **regiao\_id**.

Vamos ao Exemplo:

```
SQL> select ID,REGION_ID FROM S_DEPT;
```

ID	REGION_ID
10	1
31	1
32	2
33	3
34	4
35	5
41	1
42	2
43	3
44	4

45	5
50	1

12 rows selected.

No exemplo abaixo estamos mostrando como trazer campos distintos de uma determinada tabela, ou seja campos que não se repetem. Na coluna name tem vários campos repetidos, veja no exemplo:

```
SQL> SELECT NAME FROM S_DEPT;
```

```
NAME
-----
Finance
Sales
Sales
Sales
Sales
Sales
Sales
Operations
Operations
Operations
Operations
Operations
Administration
```

12 rows selected.

Agora usando o comando **distinct** na coluna **name** da tabela **s\_dept** poderemos obter nomes distintos, sem valor repetido para a coluna **name** de **s\_dept**.

Vamos ao Exemplo:

```
SQL> SELECT DISTINCT NAME FROM S_DEPT;
```

```
NAME
-----
Administration
Finance
Operations
Sales
```

Agora podemos ver como separar em grupos sem repetição os nomes por região, os nomes aparecem conforme existam nas regiões e as regiões que aparecem conforme tenham nomes.

```
SQL> SELECT DISTINCT NAME,REGION_ID
2 FROM S_DEPT;
```

NAME	REGION_ID
Administration	1
Finance	1
Operations	1
Operations	2
Operations	3
Operations	4
Operations	5
Sales	1
Sales	2
Sales	3
Sales	4
Sales	5

12 rows selected.

A função **select** permite que seja feito expressões aritméticas manipulando os valores de seus campos. No exemplo abaixo os valores da coluna **salary**, estão sendo multiplicados por 12 e assumem seus novos valores na própria coluna.

```
SQL> SELECT LAST_NAME,SALARY * 12 FROM
S_EMP;
```

Exemplo de expressão:

```
SQL> select id,last_name,round(salary+(salary*15/10
2 ),0) " NOVO SALÁRIO" FROM S_EMP;
```

ID	LAST_NAME	NOVO SALÁRIO
1	Velasquez	2875
2	Ngao	1668
3	Nagayama	1610
4	Quick-To-See	1668
5	Ropeburn	1783
6	Urguhart	1380

7	Menchu	1438
8	Biri	1265
9	Catchpole	1495
10	Havel	1503
11	Magee	1610
12	Giljum	1714
13	Sedeghi	1742
14	Nguyen	1754
15	Dumas	1668
16	Maduro	1610
17	Smith	1081
18	Nozaki	1380
19	Patel	914
20	Newman	863
21	Markarian	978

ID	LAST_NAME	NOVO SALÁRIO
22	Chang	920
23	Patel	914
24	Dancs	989
25	Schwartz	1265

25 rows selected.

No exemplo abaixo temos o uso da função **nvl**, que tem a finalidade de trazer campos que tem valores nulos atribuindo - lhes o valor zero.

Vamos ao Exemplo:

```
SQL> SELECT LAST_NAME, TITLE, 2 SALARY*COMMISSION_PCT/100
COMISSAO
3 FROM S_EMP;
```

LAST_NAME	TITLE
Velasquez	President
Ngao	VP, Operations
Nagayama	VP, Sales
Quick-To-See	VP, Finance
Ropeburn	VP, Administration
Urguhart	Warehouse Manager
Menchu	Warehouse Manager
Biri	Warehouse Manager
Catchpole	Warehouse Manager
Havel	Warehouse Manager

Magee	Sales Representative
140	
Giljum	Sales Representative
186.25	
Sedeghi	Sales Representative
151.5	
Nguyen	Sales Representative
228.75	
Dumas	Sales Representative
253.75	
Maduro	Stock Clerk
Smith	Stock Clerk
Nozaki	Stock Clerk
Patel	Stock Clerk
Newman	Stock Clerk
Markarian	Stock Clerk

LAST_NAME	TITLE
COMISSAO	

-----	-----
Chang	Stock Clerk
Patel	Stock Clerk
Dancs	Stock Clerk
Schwartz	Stock Clerk

25 rows selected.

Observe que no exemplo anterior os campos com valores nulos vieram em branco veja agora que usando a função **nvl** apareceu o valor zero nos campos nulos.

```
SQL> SELECT LAST_NAME, TITLE,
2     SALARY*NVL(COMMISSION_PCT,0)/100 COMISSAO
3     FROM S_EMP;
```

LAST_NAME	TITLE
COMISSAO	
-----	-----
Velasquez	President
0	
Ngao	VP, Operations
0	
Nagayama	VP, Sales
0	

Quick-To-See	VP, Finance
0	
Ropeburn	VP, Administration
0	
Urguhart	Warehouse Manager
0	
Menchu	Warehouse Manager
0	
Biri	Warehouse Manager
0	
Catchpole	Warehouse Manager
0	
Havel	Warehouse Manager
0	
Magee	Sales Representative
140	
Giljum	Sales Representative
186.25	
Sedeghi	Sales Representative
151.5	
Nguyen	Sales Representative
228.75	
Dumas	Sales Representative
253.75	
Maduro	Stock Clerk
0	
Smith	Stock Clerk
0	
Nozaki	Stock Clerk
0	
Patel	Stock Clerk
0	
Newman	Stock Clerk
0	
Markarian	Stock Clerk
0	

LAST\_NAME  
COMISSAO

TITLE

---

Chang	Stock Clerk
0	
Patel	Stock Clerk
0	
Dancs	Stock Clerk
0	

Schwartz  
0

Stock Clerk

25 rows selected.  
/

Um exemplo de **SELECT** com várias funções:

O comando **lower** serve para transformarmos os dados de pesquisa em letra minúscula, o cmdo **initcap** serve para converter a primeira letra em maiúscula e o cmdo **upper** serve para converter em maiúsculo, no exemplo abaixo estamos fazendo primeiro uma concatenação e depois usando a clausula where para trazermos os registros que tem como inicial as letras vp, com o cmdo **like**.

Vamos ao Exemplo:

```
SELECT LOWER(FIRST_NAME||' '||LAST_NAME) VP,  
INITCAP(userid)USERID,  
UPPER (TITLE) TITLE  
FROM S_EMP  
WHERE TITLE LIKE 'VP%'
```

```
VP  
USERID      TITLE  
-----  
ladoris ngao  
Lngao      VP, OPERATIONS  
midori nagayama  
Mnagayam VP, SALES  
mark quick-to-see  
Mquickto VP, FINANCE  
audry ropeburn  
Aropebur VP, ADMINISTRATION
```

Observe no exemplo abaixo que não foi realizada a pesquisa porque o dado da tabela não corresponde ao formato pedido.

```
SQL> SELECT FIRST_NAME, LAST_NAME  
2 FROM S_EMP
```

```
3 WHERE LAST_NAME = 'PATEL';
```

no rows selected

Agora usando o cmdo " **lower** " fazemos a conversão, para a pesquisa , para letra minúscula possibilitando assim o sucesso da execução do exemplo anterior:

```
SQL> SELECT FIRST_NAME, LAST_NAME
2     FROM S_EMP
3     WHERE LOWER(LAST_NAME) = 'patel';
```

FIRST_NAME	LAST_NAME
Vikram	Patel
Radha	Patel

Agora usamos o cmdo "**upper**" para fazermos a conversão para maiúscula.

```
SQL> SELECT FIRST_NAME, LAST_NAME
2     FROM S_EMP
3     WHERE UPPER (LAST_NAME) = 'PATEL';
```

FIRST_NAME	LAST_NAME
Vikram	Patel
Radha	Patel

## **FINAL DE CAPÍTULO**

Finalidade do capítulo é tratar sobre **SELF JOINS**.

O **SELF JOINS** é definido por um alto relacionamento, podemos descrever o exemplo abaixo da seguinte forma: todos os **EMP** da tabela **S\_EMP** possuem uma coluna **ID** e uma coluna **MANAGER\_ID**, portanto queremos saber quem é o gerente de cada funcionário, para isso verificamos o **MANAGER\_ID** que contem um valor correspondente ao **ID** de **EMP** e então a partir do valor de **MANAGER\_ID** descobrimos quem é o gerente do **EMP**. No exemplo abaixo é bom verificar a concatenação.

Vamos ao exemplo:

```
SQL> select worker.last_name|| ' trabalha para ' ||  
2 manager.last_name  
3 from s_emp worker, s_emp manager  
4 where worker.manager_id = manager.id;
```

```
WORKER.LAST_NAME||'TRABALHAPARA' ||MANAGER.LAST_NAME
```

```
-----  
Ngao trabalha para Velasquez  
Nagayama trabalha para Velasquez  
Quick-To-See trabalha para Velasquez  
Ropeburn trabalha para Velasquez  
Urguhart trabalha para Ngao  
Menchu trabalha para Ngao  
Biri trabalha para Ngao  
Catchpole trabalha para Ngao  
Havel trabalha para Ngao  
Magee trabalha para Nagayama  
Giljum trabalha para Nagayama  
Sedeghi trabalha para Nagayama  
Nguyen trabalha para Nagayama  
Dumas trabalha para Nagayama  
Maduro trabalha para Urguhart  
Smith trabalha para Urguhart  
Nozaki trabalha para Menchu  
Patel trabalha para Menchu  
Newman trabalha para Biri  
Markarian trabalha para Biri  
Chang trabalha para Catchpole  
Schwartz trabalha para Havel
```

24 rows selected.

**FINAL DE CAPÍTULO**

Finalidade do capítulo é tratar sobre **SEQUENCE**.

Para ver se existe uma sequence selecionamos o objeto no select **object\_name** colocando a clausula **where** com o **objeto\_type** igual a sequence.

```
SQL> SELECT OBJECT_name from user_objects
      2   where object_type = 'SEQUENCE';
```

```
OBJECT_NAME
-----
S_CUSTOMER_ID
S_DEPART_ID
S_DEPT_ID
S_DEPT_ID_SEQ
S_EMP_ID
S_IMAGE_ID
S_LONGTEXT_ID
S_ORD_ID
S_PRODUCT_ID
S_REGION_ID
S_WAREHOUSE_ID
S_WORKER
WORKER_ID_SEQ
```

13 rows selected.

O que é uma SEQUENCE:

Sequence são números criados pelo ORACLE que fazem a contagem de registros assumindo valores únicos, servindo de ID, uma SEQUENCE pode ser usada por mais de uma tabela, cada qual com seus números, sem que ocorra repetição é claro.

O exemplo seguinte mostra como criar uma **SEQUENCE**, o nome da SEQUENCE é **S\_TESTE\_id**, que está relacionada com o id da tabela **TESTE** ( tabela que foi criada anteriormente), o **INCREMENTE BY** serve para que a **SEQUENCE** evolua de um valor, o **START WITH** serve para que a **SEQUENCE** comece com o numero 51, o **MAXVALUE** é o valor máximo que uma **SEQUENCE** pode assumir, **NOCACHE** especifica se será alocada a memória cash ou não, **NOCYCLE** serve para especificar ou não um ciclo de **SEQUENCE** ou seja os números vão contando em um ciclo de tempo determinado.

Vamos ao Exemplo:

```
SQL> CREATE SEQUENCE S_TESTE_id
 2 INCREMENT BY 1
 3 START WITH 51
 4 MAXVALUE 9999999
 5 NOCACHE
 6 NOCYCLE;
```

Sequence created.

Como mostrar todas as SEQUENCES que estão disponíveis para seu user:

```
SQL> select sequence_name, min_value, max_value,
 2 increment_by, last_number
 3 from user_sequences;
```

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
S_CUSTOMER_ID 216	1	9999999		1
S_DEPT_ID 51	1	9999999		1
S_EMP_ID 26	1	9999999		1
S_IMAGE_ID 1981	1	9999999		1
S_LONGTEXT_ID 1369	1	9999999		1
S_ORD_ID 113	1	9999999		1
S_PRODUCT_ID 50537	1	9999999		1
S_REGION_ID 6	1	9999999		1
S_TESTE 51	1	9999999		1
S_TESTE_ID 51	1	9999999		1
S_WAREHOUSE_ID 10502	1	9999999		1

11 rows selected.

No exemplo abaixo estamos criando uma **SEQUENCE** para o ID de s\_dept, foi escolhido como nome para a SEQUENCE **s\_dept\_id**, sempre para **SEQUENCE** usa-se o "S" no começo do nome da sequence.

```
SQL> CREATE SEQUENCE S_dept_id
2      INCREMENT BY 1
3      START WITH 51
4      MAXVALUE 9999999
5      NOCACHE
6      NOCYCLE;
```

Sequence created.

O uso do **NEXTVAL**:

Agora vamos inserir valores dentro de **s\_dept** sendo que um desses valores é o **NEXTVAL** que tem como função trazer para nós o próximo número da SEQUENCE e no caso específico estamos inserindo em ID o valor da **SEQUENCE**.

Veja abaixo:

```
SQL> insert into s_dept(id,name,region_id)
2  values      (s_dept_id.NEXTVAL, 'FINANCE',2);
```

1 row created.

Verificando a inserção do **NEXTVAL**, como nossa **SEQUENCE** inicia em 51 o primeiro valor a ser inserido é 51.

```
SQL> SELECT ID
2  FROM S_DEPT
3  WHERE NAME = 'FINANCE';
```

```
          ID
-----
          51
```

Alterando uma **SEQUENCE**: Para se alterar uma SEQUENCE temos que ter privilégios para tal, os valores passados não são alterados pela **SEQUENCE**, o **START WITH** de uma **SEQUENCE** não pode ser alterado, para alterá-lo temos que dropar a sequence, dropando-a não quer dizer que os valores já foram inseridos nas primary keys serão apagados eles já foram criados.

```
SQL> EDIT
```

```
Wrote file afiedt.buf
  1 ALTER SEQUENCE S_DEPT_ID
  2 INCREMENT BY 4
  3 MAXVALUE 99999
  4 CYCLE
  5* NOCACHE
SQL> /
```

Sequence altered.

Para dropar uma **SEQUENCE** temos que seguir os passos a seguir:

```
SQL> DROP SEQUENCE s_dept_id;
```

Sequence dropped.

```
SQL> DROP SEQUENCE S_TESTE;
```

Sequence dropped.

```
SQL> DROP SEQUENCE S_TESTE_ID;
```

Sequence dropped.

## **FINAL DE CAPÍTULO**

Finalidade do capítulo é o uso de **SUBQUERYS**:

Uma **subquery** é um cmdo **select** dentro de um outro cmdo **select** onde retorna uma ou mais linhas a fim de satisfazer uma clausula **WHERE**.

No exemplo abaixo temos um **select** em **s\_emp** onde procuramos trazer o **last\_name** e o **title**, onde o **title** pesquisado seja o mesmo do " **Smith**", para isso é realizado uma **subquery** que nada mais é que um **select**, que neste caso retorna um valor somente para a comparação na where.

```
SQL> select last_name, title
      2   from s_emp
      3   where title =
      4         (select title
      5           from s_emp
      6           where last_name = 'Smith');
```

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

10 rows selected.

Outro exemplo de **subquery**, que neste caso está comparando os valores da coluna **SALARY** com a média dos salários da tabela **s\_emp**. A função **AVG** está trazendo a média dos salários.

```
SQL> select last_name, title, salary
       2   from s_emp
       3   where salary<
       4         (select avg(salary)
       5           from s_emp);
```

LAST_NAME	TITLE
Urguhart	Warehouse Manager
1200	
Menchu	Warehouse Manager
1250	
Biri	Warehouse Manager
1100	
Smith	Stock Clerk
940	
Nozaki	Stock Clerk
1200	
Patel	Stock Clerk
795	
Newman	Stock Clerk
750	
Markarian	Stock Clerk
850	
Chang	Stock Clerk
800	
Patel	Stock Clerk
795	
Dancs	Stock Clerk
860	
Schwartz	Stock Clerk
1100	

12 rows selected.

Nos exemplos anteriores vemos que retornavam só um único valor para comparação na clausula **where**, neste caso agora há

o retorno de mais de um valor para a comparação na clausula where, mas para que ocorra a comparação com mais de um valor temos que usar o **IN** em vez do "=" no exemplo abaixo ocorre um erro:

```
SQL> select last_name, title
  2     from s_emp
  3     where dept_id =
  4           (select id from s_dept
  5             where name = 'finance or region_id = 2);
select last_name, title
*
```

ERROR at line 1:  
ORA-01756: quoted string not properly terminated

Agora usando o **IN** na clausula **where** poderá o dept\_id ser comparado com as duas condições, o select trará os registros que na tabela **s\_dept** que tenham o **nome** igual a '**Finace**' ou que a **region\_id** seja igual a 2.

```
SQL> select last_name,first_name,title
  2     from s_emp
  3     where dept_id in
  4           (select id
  5             from s_dept
  6             where name = 'Finance' or region_id =2);
```

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

O uso do **having** em **subquery**:

Neste exemplo estamos querendo selecionar o **dept\_id** e a média dos salários de **s\_emp**, grupados pelo **dept\_id**, com a condição de que a média dos salários de **s\_emp** seja maior que a média dos salários do **dept 32** para isso usamos o **HAVING**.

Vamos ao Exemplo:

```
SQL> select dept_id,avg(salary)
  2   from    s_emp
  3   group by dept_id
  4   having  avg(salary)>
  5           (select avg(salary)
  6            from s_emp
  7            where dept_id = 32);
```

```
DEPT_ID AVG(SALARY)
-----
          33          1515
          50          2025
```

Agora dentro do **group by** estamos usando o **having** e dentro da **subquery** selecionando o menor valor da média da tabela **s\_emp** grupado por **title**

```
SQL>
  1 select title,avg(salary)
  2   from    s_emp
  3   group by title
  4   having  avg(salary) =
  5           (select min(avg(salary))
  6            from s_emp
  7*          group by title)
```

```
TITLE                                AVG(SALARY)
-----
Stock Clerk                            949
```

```
SQL> select min(avg(salary))
  2   from s_emp
```

```
3 group by title;  
MIN(AVG(SALARY))  
-----  
949
```

**FINAL DE CAPÍTULO**

Finalidade do capítulo é tratar o **RENAME**, **TRUNCATE**.

O **RENAME** é usado para renomear uma tabela, view etc. observe a sintaxe abaixo:

```
SQL> RENAME s_ord TO S_ORDER  
      2  ;
```

Table renamed.

O CMDO **TRUNCATE** é responsável por deletar os registros de uma tabela da mesma forma do **DELETE** com a diferença que o **TRUNCATE** libera espaço na tabela, para usar o CMDO **TRUNCATE** tem que ser dono da tabela ou ter permissão para isto. Se houver **constraints** o cmdo truncate não funciona, tem que usar o **DISABLE constraints** para desabilitar as **constraints**.

```
SQL> TRUNCATE TABLE S_ITEM;
```

Table truncated.

## **FINAL DE CAPÍTULO**

Finalidade do capítulo é tratar sobre o comando **UPDATE**.

O CMDO **UPDATE** serve para fazermos alterações em registros dentro de nossa tabela observe os exemplos abaixo:

Estamos vendo o nr do dept onde o id é igual a 2:

```
SQL> select dept_id
      2   from
      3   S_EMP
      4   where id = 2;
```

```
      DEPT_ID
-----
           41
```

Agora todos que tiverem nr do id igual a 2 passará a ter o id do dept igual a 10.

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

1 row updated.

Verificando se realmente se concretizou o **UPDATE**

```
SQL> select dept_id
      2   from
      3   S_EMP
      4   where id = 2;
```

```
      DEPT_ID
-----
           10
```

Estamos selecionado dept\_id, salary de s\_emp onde o id for igual a 1.

```
SQL> select dept_id,salary
      2   from s_emp
      3   where id = 1;
```

```
      DEPT_ID      SALARY
-----

```

50            2500

Agora vamos alterar os valores de **dept\_id** para 32 e de **salary** para 2550 onde o **id** for igual a 1.

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

1 row updated.

Verificando a concretização da mudança.

```
SQL> select dept_id,salary
      2 from s_emp
      3 where id = 1;
```

DEPT_ID	SALARY
32	2550

Quando não usamos a clausula **where** no **UPDATE** fazemos alterações em todos os registros da tabela, no caso estamos mudando todos os campos de **commission\_pct** para 10 sem distinção.

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

26 rows updated.

Neste exemplo estamos tentando fazer uma alteração em um valor que é uma **FOREIKEY**, e o valor que queremos adicionar não existe na tabela de origem da FK portanto irá ferir uma constraint.

```
SQL> UPDATE S_EMP
      2 SET DEPT_ID = 60
      3 WHERE DEPT_ID = 10;
```

```
UPDATE S_EMP
      *
```

ERROR at line 1:

ORA-02291: integrity constraint (GUIMA.S\_EMP\_DEPT\_ID\_FK) violated - parent key not found

Agora no próximo exemplo estamos verificando quais são os **dept\_id** que fazem relação com s\_emp.

```
SQL> select dept_id  
2   from s_emp;
```

```
DEPT_ID  
-----  
32  
41  
31  
10  
50  
41  
42  
43  
44  
45  
31  
32  
33  
34  
35  
41  
41  
42  
42  
43  
43
```

26 rows selected.

Agora vamos fazer a alteração de dept\_id passando todos que são 10 para 40 e não irá violar a constraint porque o valor 40 existe na tabela **DEPT**. como se pode observar no select anterior.

```
SQL> UPDATE S_EMP  
2   SET DEPT_ID = 41  
3   WHERE DEPT_ID = 10;
```

1 row updated.

Finalidade do capítulo é tratar sobre **VIEW**.

Uma VIEW é como se fosse uma janela que dá acesso aos dados da tabela, só que com restrições. No exemplo abaixo estamos criando uma VIEW usando uma SUBQUERY, trazendo dados específicos de uma tabela. Vamos ao exemplo:

```
SQL> CREATE VIEW empvu45 2 AS SELECT id,last_name,title
3 FROM s_emp
4 WHERE dept_id = 45;
```

View created.

Estamos agora realizando um **select** em nossa VIEW:

```
SQL> select * from
2 empvu45;
```

ID	LAST_NAME	TITLE
10	Havel	Warehouse Manager
24	Dancs	Stock Clerk
25	Schwartz	Stock Clerk

Estamos fazendo um **select** na tabela **S\_EMP**, referente ao **dept\_id** 45 e vemos que nossa **VIEW** é idêntica.

```
SQL> select id,last_name,title
2 FROM s_emp
3 WHERE dept_id = 45;
```

ID	LAST_NAME	TITLE
10	Havel	Warehouse Manager
24	Dancs	Stock Clerk
25	Schwartz	Stock Clerk

Criando uma **VIEW** usando **ALIASES**:

```
SQL> CREATE VIEW salvu41
```

```

2 AS SELECT id,first_name PRIMEIRO,
3 last_name ULTIMO, salary SALARIO_MENSAL
4 FROM
5 s_emp
6 WHERE dept_id = 41;

```

View created.

Vendo a VIEW criada:

```

SQL> desc   salvu41
Name                               Null?      Type
-----
ID                                   NOT NULL  NUMBER(7)
PRIMEIRO
VARCHAR2(25)
ULTIMO                               NOT NULL  VARCHAR2(25)
SALARIO_MENSAL
NUMBER(11,2)

```

Mais um exemplo de como criar uma **VIEW**:

Estamos criando uma VIEW chamada "**dept\_sum\_vu**" com os nomes de colunas criados por eu da forma que achar melhor, observe que está sendo realizado uma **SUBQUERY** e um **JOIN** dentro da **SUBQUERY** entre as tabelas **s\_emp** e **s\_dept**, onde pega-se o menor salário função **MIN**, o maior salário(função **MAX**) a média dos salários(função **AVG**) com a condição estabelecida da clausula **WHERE** que não é mais que a ligação do **JOIN**, e por último nossa **VIEW** virá agrupada pelo **NOME** de **s\_dept**.

```

SQL> EDIT
Wrote file afiedt.buf
 1 create VIEW dept_sum_vu
 2 (name, minsal,maxsal, avgsal)
 3 AS SELECT d.name, MIN (e.salary),
 4 MAX(e.salary),AVG(e.salary)
 5 FROM s_emp e,s_dept d
 6 WHERE e.dept_id = d.id
 7* GROUP BY d.name
SQL> /

```

View created.

Este é o formato de nossa **VIEW**, verifique o nome das colunas correspondentes a **SUBQUERY** dentro do **CREATE VIEW**, e agrupados por nome.

```
SQL> desc dept_sum_vu
Name                                         Null?      Type
-----
NAME                                         NOT NULL
VARCHAR2(25)
MINSAL                                       NUMBER
MAXSAL                                       NUMBER
AVGSAL                                       NUMBER
```

Agora vamos fazer um select em **s\_dept** e **s\_emp** com um **JOIN** e depois vamos comparar os dados obtidos com o resultado obtido de nossa **VIEW**:

```
Wrote file afiedt.buf
 1  SELECT d.name, MIN  (e.salary),
 2  MAX(e.salary),AVG(e.salary)
 3  FROM s_emp e,s_dept d
 4  WHERE e.dept_id = d.id
 5* GROUP BY d.name
SQL> /
```

NAME	MIN(E.SALARY)	MAX(E.SALARY)	AVG(E.SALARY)
Administration	1550	2500	2025
Finance	1450	1450	1450
Operations	825	1540	1144.7667
Sales	874.5	1525	1379.2143

Agora estamos fazendo um select na **VIEW** criada usando todas as colunas especificadas em sua criação veja que os dados trazidos são os mesmos da tabela **s\_emp** e **s\_dept** submetida em um select com join:

```
SQL> EDIT
Wrote file afiedt.buf
 1  select NAME,MINSAL,MAXSAL,AVGSAL
 2  FROM
 3* DEPT_SUM_VU
SQL> /
```

NAME	MINSAL	MAXSAL
AVGSAL		

```

-----
Administration          1550          2500
2025
Finance                  1450          1450
1450
Operations               825           1540
1144.7667
Sales                    874.5         1525
1379.2143

```

Estamos agora criando uma **VIEW** com a condição **WITH CHECK OPTION CONSTRAINT** que tem a finalidade de não permitir alteração na **VIEW** em relação a clausula **WHERE**.

```

SQL> edit
Wrote file afiedt.buf
  1  create or replace VIEW empvu41
  2  AS SELECT          *
  3  FROM              s_emp
  4  WHERE             dept_id = 41
  5* WITH CHECK OPTION CONSTRAINT empvu41_ck
SQL> /

```

View created.

Tentamos fazer uma alteração na **VIEW empvu41**, não será permitida a alteração nos moldes sugeridos abaixo ou seja a alteração no nr do departamento.

```

SQL> UPDATE empvu41
  2  SET DEPT_ID = 42
  3  WHERE ID = 16;
UPDATE empvu41
      *
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation.

```

Criando uma **VIEW** e determinando que ela seja somente de leitura.

```

SQL> CREATE OR REPLACE VIEW empvu45
  2  (id_number, employee, job)
  3  AS SELECT id,last_name,title
  4  FROM S_EMP
  5  WHERE dept_id = 45
  6  WITH READ ONLY;

```

View created.

Estamos tentando deletar registros da VIEW empvu45 só que ela foi criada com a opção **WITH READ ONLY**, ou seja somente leitura.

```
SQL> DELETE FROM empvu45
      2  WHERE ID_NUMBER = 10;
DELETE FROM empvu45
      *
```

```
ERROR at line 1:
ORA-01752: cannot delete from view without exactly one key-
preserved table
```

Para saber as **VIEWS** existentes vamos a **USER\_VIEWS** que pertence ao dicionário de dados e então podemos pesquisar veja só:

```
SQL> describe user_views;
```

Name	Null?	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG

-- selecionando as VIEWS existentes:

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

```
VIEW_NAME      TEXT_LENGTH TEXT
-----
EMPVU41 194 SELECT"ID","LAST_NAME","FIRST_NAME","USE
RID","START_DATE","COMMENTS","MANAG

EMPVU45      74 SELECT id,last_name,title
              FROM  S_EMP
              WHERE   dept_id = 45
              WITH READ ONLY
```

Para deletar uma **VIEW** usamos o cmdo **DROP**:  
Quando deletamos uma **VIEW** não alteramos em nada a tabela.

```
SQL> DROP VIEW EMPVU45 ;
```

View dropped.

**FINAL DE CAPÍTULO**

Finalidade do capítulo é o uso da clausula **WHERE**.

A clausula **where** tem a função de dar condições para o **select** onde especifica a pesquisa, neste primeiro exemplo temos um select somente no departamento numero 42. SQL> SELECT

```
LAST_NAME, DEPT_ID, SALARY
 2 FROM S_EMP
 3 WHERE DEPT_ID = 42;
```

LAST_NAME	DEPT_ID	SALARY
Menchu	42	1250
Nozaki	42	1200
Patel	42	795

Neste próximo caso vemos o exemplo de que para fazermos uma pesquisa, temos que colocar o nome ou numero a ser pesquisado como mesmo formato do que se encontra no banco, no caso abaixo ocorreu erro por que o nome colocado é diferente do que está no banco.

```
SQL> SELECT FIRST_NAME, LAST_NAME, TITLE
 2 FROM S_EMP
 3 WHERE LAST_NAME = 'MAGEE';
```

no rows selected

Agora com a correção do nome temos a pesquisa com sucesso.

```
SQL> SELECT FIRST_NAME, LAST_NAME, TITLE
 2 FROM S_EMP
 3 WHERE LAST_NAME = 'Magee';
```

FIRST_NAME	LAST_NAME	TITLE
Colin	Magee	Sales Representative

Usando a clausula **where** com o comando **between ... and ...**, que tem a finalidade de trazer valores delimitados dentro de um determinado espaço, no exemplo abaixo o comando traz uma seleção que está entre 09-may-91 and 17-jun-91.

```
SQL> SELECT FIRST_NAME, LAST_NAME, START_DATE
2 FROM S_EMP
3 WHERE START_DATE BETWEEN '09-MAY-91'
4 AND '17-JUN-91';
```

FIRST_NAME	LAST_NAME	START_DAT
Midori	Nagayama	17-JUN-91
Alexander	Markarian	26-MAY-91
Sylvie	Schwartz	09-MAY-91

Agora vamos fazer um **desc** na tabela **s\_dept**:

```
SQL> DESC S_DEPT;
Name                               Null?      Type
-----
ID                                  NOT NULL  NUMBER(7)
NAME                                NOT NULL
VARCHAR2(25)
REGION_ID                            NUMBER(7)
```

Agora vamos fazer uma seleção onde **region\_id** seja (1,3) e somente estes.

```
SQL> SELECT ID, NAME, REGION_ID
2 FROM S_DEPT
3 WHERE REGION_ID IN (1,3);
```

ID	NAME	REGION_ID
10	Finance	1
31	Sales	1
33	Sales	3
41	Operations	1
43	Operations	3
50	Administration	1

6 rows selected.

No exemplo abaixo o uso comando **null** na clausula **where**, especifica e traz o **s\_customer** onde **sales\_rep\_id** for nulo ou seja onde o customer não tiver **sales\_rep\_id**. Mas neste exemplo abaixo não cumpre a finalidade, apesar de estar correto.

```
SQL> select ID,NAME,CREDIT_RATING
  2   FROM S_CUSTOMER
  3   WHERE SALES_REP_ID = NULL;
```

no rows selected

Forma errada de se pesquisar por um valor nulo:

```
SQL> SELECT ID,NAME,CREDIT_RATING
  2   FROM S_CUSTOMER
  3   WHERE SALES_REP_ID = ' ';
```

ERROR:

ORA-01722: invalid number

no rows selected

Veja a maneira correta de pesquisar usando a clausula **where** com a condição de campos nulos.

```
SQL>  SELECT ID,NAME,CREDIT_RATING
  2   FROM S_CUSTOMER
  3   WHERE SALES_REP_ID IS NULL;
```

ID	NAME	CREDIT_RA
207	Sweet Rock Sports	GOOD

/

O exemplo abaixo demonstra o uso do comando **and** dentro da clausula **where**, onde dentro da tabela **s\_emp** será pesquisado os registros que sejam do departamento **41** e com o nome do **title " stock clerk"**.

```
SQL> SELECT LAST_NAME,SALARY,DEPT_ID,TITLE
  2   FROM S_EMP
  3   WHERE DEPT_ID = 41
  4   AND TITLE = 'Stock Clerk';
```

LAST_NAME	SALARY	DEPT_ID	TITLE
-----------	--------	---------	-------

-----

Maduro Clerk	1400	41 Stock
Smith Clerk	940	41 Stock

Este exemplo mostra o uso do comando "OR" dentro da clausula **WHERE** onde no caso a pesquisa é feita na tabela "**S\_EMP**" trazendo todos os registros que contenham o **dept\_id** igual a **41** como também todos aqueles que possuam o **title** igual a "**Stock Clerk**".

Vamos

```
SQL> SELECT LAST_NAME, SALARY, DEPT_ID, TITLE
      2 FROM S_EMP
      3 where DEPT_ID = 41
      4 OR TITLE = 'Stock Clerk';
```

LAST_NAME	SALARY	DEPT_ID	TITLE
Ngao Operations Urguhart Warehouse Manager	1450 1200	41	VP,
Maduro Clerk	1400	41	Stock
Smith Clerk	940	41	Stock
Nozaki Clerk	1200	42	Stock
Patel Clerk	795	42	Stock
Newman Clerk	750	43	Stock
Markarian Clerk	850	43	Stock
Chang Clerk	800	44	Stock
Patel Clerk	795	34	Stock
Dancs Clerk	860	45	Stock
Schwartz Clerk	1100	45	Stock

12 rows selected.

Nos exemplos abaixo temos como usar o **AND** e o **OR** juntos dentro de uma clausula **where**, no primeiro exemplo o select traz dentro da tabela s\_emp todos registros onde o salário seja maior ou igual a 1000 mas somente no departamento 44 e traz todos os registros do departamento 42, independente da condição de salário.

Vamos ao exemplo:

```
SQL> select last_name,salary,dept_id
2   from s_emp
3   where salary >= 1000
4   and dept_id = 44
5   or dept_id = 42;
```

LAST_NAME	SALARY	DEPT_ID
Menchu	1250	42
Catchpole	1300	44
Nozaki	1200	42
Patel	795	42

Com o uso do parênteses nós podemos no exemplo abaixo selecionar todos os registros do departamento 44 e 42 que recebam salário maior ou igual a 1000, diferente do exemplo anterior, portanto o uso do parênteses determina que a condição da clausula **WHERE** do salário sirva par os dois.

```
SQL> select last_name,salary,dept_id
2   from s_emp
3   where salary >= 1000
4   and (dept_id = 44
5   or dept_id = 42);
```

LAST_NAME	SALARY	DEPT_ID
Menchu	1250	42
Catchpole	1300	44
Nozaki	1200	42

Na clausula **where** usamos o operador **like** que serve no geral para trazer valores aproximados ou parecidos na pesquisa, podemos ver abaixo os exemplos:

No primeiro caso abaixo o operador **like** esta pesquisando o **last\_name** da tabela **s\_emp** que se inicia com a letra (m).

```
SQL> SELECT LAST_NAME
      2 FROM S_EMP
      3 WHERE LAST_NAME LIKE 'M%'
      4 ;
```

```
LAST_NAME
-----
Menchu
Magee
Maduro
Markarian
```

Agora no exemplo abaixo o comando **like** faz uma pesquisa no **start\_date** da tabela **s\_emp** onde a data termina em **91**.

```
SQL> SELECT LAST_NAME, START_DATE
      2 FROM S_EMP
      3 WHERE START_DATE LIKE '%91';
```

LAST_NAME	START_DAT
Nagayama	17-JUN-91
Urguhart	18-JAN-91
Havel	27-FEB-91
Sedeghi	18-FEB-91
Dumas	09-OCT-91
Nozaki	09-FEB-91
Patel	06-AUG-91
Newman	21-JUL-91
Markarian	26-MAY-91
Dancs	17-MAR-91
Schwartz	09-MAY-91

11 rows selected.

No exemplo abaixo é feito uma seleção na tabela **s\_emp** onde a segunda letra do nome começa com **a**, não deu certo no caso abaixo porque a letra digitada é maiúscula e na tabela a ser pesquisada as segundas letras são minúsculas.

```
SQL> SELECT LAST_NAME, START_DATE
 2   FROM S_EMP
 3   WHERE LAST_NAME LIKE '_A%';
```

no rows selected

Mas neste agora temos a pesquisa concluída satisfazendo a condição da segunda letra da clausula where, observe o uso do hífen.

```
SQL> SELECT LAST_NAME, START_DATE
 2   FROM S_EMP
 3   WHERE LAST_NAME LIKE '_a%';
```

LAST_NAME	START_DATE
Nagayama	17-JUN-91
Catchpole	09-FEB-92
Havel	27-FEB-91
Magee	14-MAY-90
Maduro	07-FEB-92
Patel	06-AUG-91
Markarian	26-MAY-91
Patel	17-OCT-90
Dancs	17-MAR-91

9 rows selected.

Quando optarmos por trazer somente os dados onde não contenham uma letra qualquer, usamos no comando **notlike**. A seguinte expressão : **NOT LIKE '%A%'**, no exemplo abaixo não traria o resultado desejado, porque as letras na tabela são minúsculas "a" e a utilizada foi maiúscula "A".

```
SQL> SELECT LAST_NAME
 2   FROM S_EMP
 3   WHERE LAST_NAME NOT LIKE '%A%';
```

LAST_NAME
Velasquez
Ngao
Nagayama
Quick-To-See

```
Ropeburn
Urguhart
Menchu
Biri
Catchpole
Havel
25 rows selected.
```

Agora nós estamos usando uma letra que está no mesmo formato dos dados na tabela e então a execução do **select** trará todos os nomes da tabela **s\_emp** que não tenham a letra "a".

Vamos ao exemplo:

```
SQL> SELECT LAST_NAME
      2 FROM S_EMP
      3 WHERE LAST_NAME NOT LIKE '%a%';
```

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

8 rows selected.

No exemplo abaixo estamos selecionando somente funcionários da tabela **s\_emp**, que receberam comissão, portanto na clausula **where** temos o comando condicional **"is not null"**.

```
SQL> SELECT LAST_NAME, TITLE, COMMISSION_PCT
      2 FROM S_EMP
```

```
3 WHERE COMMISSION_PCT IS NOT NULL;
```

LAST_NAME	TITLE
COMMISSION_PCT	
Magee	Sales Representative
10	
Giljum	Sales Representative
12.5	
Sedeghi	Sales Representative
10	
Nguyen	Sales Representative
15	
Dumas	Sales Representative
17.5	

Agora queremos todos os funcionários que não recebem comissão ou seja que possuam o campo **COMMISSION\_PCT** nulo.

```
SQL> SELECT LAST_NAME, TITLE, COMMISSION_PCT
2 FROM S_EMP
3 WHERE COMMISSION_PCT IS NULL;
```

LAST_NAME	TITLE
COMMISSION_PCT	
Velasquez	President
Ngao	VP, Operations
Nagayama	VP, Sales
Quick-To-See	VP, Finance
Ropeburn	VP, Administration
Urguhart	Warehouse Manager
Menchu	Warehouse Manager
Biri	Warehouse Manager
Catchpole	Warehouse Manager
Havel	Warehouse Manager

Maduro	Stock Clerk
Smith	Stock Clerk
Nozaki	Stock Clerk

20 rows selected.

**FINAL DE CAPÍTULO**

O Usuário inserindo valores em tempo de execução:  
O uso do '&' e do 'SET OFF', ' SET VVERIFY ON' E ETC.

A finalidade dos exemplos abaixo é demonstrar como fazer quando queremos que o usuário entre com valores, em tempo de execução. É sempre bom lembrar que estamos no SQL Plus e os artifícios de entrada não são tão amigáveis, mas servem para que em uma programação posterior, possamos usá-los para execução.

Primeiramente o usuário entra com dados em tempo de execução, para em seguida ser executada uma pesquisa através de um cmdo sql.

Vemos o uso do "&" para que o usuário entre com dados. No exemplo abaixo temos um pedido de entrada de um numero por isso é que o **numero\_do\_dept** não está entre aspas.

Podemos notar que aparece o "old" valor e o "new" valor isso é devido ao cmdo **set verify on**, se quisermos que não apareça, deve ser usado o **set verify off**.

Vamos ao Exemplo: No exemplo queremos selecionar o id,last\_name,salary da tabela s\_emp onde o dept\_id seja igual ao valor digitado pelo usuário através **&numero\_do\_dept**, que no exemplo será o nr 31.

```
SQL> set verify on
SQL> select id,last_name,salary 2 from
s_emp 3 where dept_id=&numero_do_dept;
Enter value for numero_do_dept: 31old 3: where
dept_id=&numero_do_deptnew 3: where dept_id=31 ID
LAST_NAME SALARY
-----
3 Nagayama 1400
11 Magee 1400
```

Agora não aparecerá o new e old valor por causa do **SET VERIFY OFF**.

```
SQL> set verify off
SQL> select id,last_name,salary
2 from s_emp
3 where dept_id=&numero_do_dept;
Enter value for numero_do_dept: 31 ID LAST_NAME
SALARY-----
3 Nagayama 1400
11 Magee 1400
```

Teremos um exemplo de como usuário pode entrar com dados quer seja do tipo caracter ou numérico, note que há o uso das aspas, temos que ter em mente que o formato a ser digitado especificamente no exemplo abaixo tem que ser idêntico ao que está na tabela, com maiúsculas e minúsculas pois não estamos usando nenhum comando para fazer a conversão.

Vamos ao Exemplo: estamos selecionando id,last\_name,salary da tabela s\_emp onde o campo da coluna title seja exatamente igual ao nome digitado pelo usuário pela a opção '&job\_title'

```
SQL> select id,last_name,salary
      2   from s_emp
      3   where title = '&job_title';
```

Enter value for job\_title: Stock Clerk

ID	LAST_NAME	SALARY
16	Maduro	1400
17	Smith	940
18	Nozaki	1200
19	Patel	795
20	Newman	750
21	Markarian	850
22	Chang	800
23	Patel	795
24	Dancs	860
25	Schwartz	1100

10 rows selected.

Agora vamos ver como que o usuário poderá entrar com o nome da coluna e com a condição de pesquisa que ele deseja que seja estabelecida na cláusula **WHERE**. Neste caso temos um exemplo onde é requerida a entrada de um dado numérico.

Vamos ao Exemplo: estamos selecionando o id, uma coluna sugerida pelo usuário(logo que exista), referente a tabela s\_emp e como também definindo uma condição para tal pesquisa.SQL> select id,&nome\_coluna 2 from s\_emp 3 where &condição;

Enter value for nome\_coluna: LAST\_NAME

Enter value for condição: SALARY > 100

ID	LAST_NAME
----	-----------

- 1 Velasquez
- 2 Ngao
- 3 Nagayama
- 4 Quick-To-See
- 5 Ropeburn
- 6 Urguhart
- 7 Menchu
- 8 Biri

Outro exemplo em que o usuário entra com o valor da coluna e da condição **WHERE**.

```
SQL> select id,&nome_coluna 2 from s_ord 3 where
&condição;
Enter value for nome_coluna: date_orderedEnter value for
condição: total>30000
```

ID	DATE_ORDE
100	31-AUG-92
104	03-SEP-92
107	07-SEP-92
108	07-SEP-92
109	08-SEP-92
97	28-AUG-92

6 rows selected.

Construindo um script usando as opções para que o usuário possa entrar com dados e como também mostrar em tela somente o prompt.

O cmdo **set echo off** serve para que em tempo de execução não se exiba os comandos do sql. Já o **set echo on** serve para retornar a forma anterior.

O cmdo **accept** serve para preparar um prompt para receber um valor.

Neste exemplo também temos a criação de uma variável `v_name` que recebe valores.

Estamos preparando o ambiente para receber valores que serão armazenados dentro de uma variável, para após isto, serem feitas comparações dentro da cláusula **where**.

Estamos usando duas tabelas s\_dept e s\_region, o **and** é uma função onde complementa a cláusula **where**, e o **upper** no dpt.name está passando o conteúdo do nome do dept para maiúsculo para que seja efetuada a comparação com o nome digitado pelo o usuário que por sua vez recebe um **upper** que o transforma em maiúsculo, essa alternativa é feita porque não sabemos qual é o formato do dado na tabela.

Vamos ao Exemplo:

```
SET ECHO OFF
ACCEPT V_NAME PROMPT 'DÊ O NOME DO DEPARTAMENTO:'
SELECT DPT.NAME, REG.ID, REG.NAME " NOME DA REGIÃO"
FROM S_DEPT DPT, S_REGION REG
WHERE DPT.REGION_ID = REG.ID -- está fazendo o join entre as
tabelas
AND UPPER(DPT.NAME) LIKE UPPER('%&V_NAME%')
/
SET ECHO ON
```

Como estamos gerando um SCRIPT os comandos devem ficar guardados dentro de um arquivo que possua a extensão \*.SQL e preparado da forma acima. Uma vez no SQL se desejarmos executar o nosso SCRIPT temos que seguir o procedimento descrito abaixo, usando "@" e o nome do arquivo, ou " **START**" e o nome do arquivo.

```
SET ECHO ON
SQL> @TEST.SQL
SQL> SET ECHO OFF
DÊ O NOME DO DEPARTAMENTO:sales
old 4: AND UPPER(DPT.NAME) LIKE UPPER('%&V_NAME%')
new 4: AND UPPER(DPT.NAME) LIKE UPPER('%sales%')
```

NAME	ID	NOME DA REGIÃO
Sales	1	North America
Sales	2	South America
Sales	3	Africa / Middle East
Sales	4	Ásia
Sales	5	Europe

Podemos notar que no exemplo anterior foi mostrado o **OLD** e o **NEW** valores da variável, para que não mostre, temos que usar o **SET VERIFY OFF**, veja abaixo:

```
SET VERIFY OFF
SET ECHO OFF
ACCEPT V_NAME PROMPT 'DÊ O NOME DO DEPARTAMENTO:'
SELECT DPT.NAME, REG.ID, REG.NAME " NOME DA REGIÃO"
FROM S_DEPT DPT, S_REGION REG
WHERE DPT.REGION_ID = REG.ID
AND UPPER(DPT.NAME) LIKE UPPER('%&V_NAME%')
/
SET ECHO ON
```

```
SQL> START TEST.SQL
SQL> SET VERIFY OFF
SQL> SET ECHO OFF
DÊ O NOME DO DEPARTAMENTO:SALES
```

NAME	ID	NOME DA REGIÃO
-----		-----
Sales	1	North America
Sales	2	South America
Sales	3	Africa / Middle East
Sales	4	Ásia
Sales	5	Europe

Input truncated to 11 characters

**FINAL DE CAPÍTULO**

Finalidade do capítulo é tratar sobre exemplos diversos.

Vamos descrever o uso dos cmdos abaixo: **Concat**, usado para concatenar o conteúdo de `last_name` mais o conteúdo de `title` jogando dentro de " vice presidência" **upper** está transformando o conteúdo de `last_name` em maiúsculo, o cmdo **substr**(`title,3`) reduz as três primeiras posições do conteúdo da coluna `title` que no caso é "vp", o cmdo **like** pesquisa algo parecido com "vp" = 'vp%'.

```
SQL> SELECT CONCAT(UPPER(last_name),
  2     substr(title,3) " vice presidência"
  3   from s_emp
  4     where title LIKE 'VP%';
```

vice presidência

```
-----
NGAO, Operations
NAGAYAMA, Sales
QUICK-TO-SEE, Finance
ROPEBURN, Administration
```

Para colocarmos um valor caracter dentro de um campo que seja number, devemos usar a conversão " **TO\_CHAR**", funções e cmdos usados no exemplo abaixo:

"**NVL**" usado para manipulação de valores nulos.

"**TO\_CHAR**" usado no caso para a conversão de "manager\_id" que tem um formato numérico, a fim de prepará-lo para receber um valor que não é numérico.

"**is null**" função que indica o valor nulo na coluna.

```
SQL> select last_name,
  2     nvl(to_char(manager_id), 'não tem gerente')
  3   from s_emp
  4     where manager_id is null;
```

```
LAST_NAME      NVL(TO_CHAR(MANAGER_ID), 'NÃO TEM GERENTE')
```

```
-----
Velasquez      não tem gerente
```

Comandos utilizados no exemplo abaixo:

"**TO\_CHAR**" neste caso com a função de fazer a conversão da data default do sistema para a data desejada, onde os formatos são colocados da seguinte forma e com as seguintes funções: **Day** = dia por extenso, **Month** = nome do mês por extenso, **ddth** nome do dia em forma de numeral, **YYYY** = ano no padrão numeral, o cmdo "**NEXT DAY** ' tem a função de ir para uma próxima data que no caso faz referencia a próxima sexta feira. ( **next\_day**, '**friday**' ). O cmdo "**ADD\_MONTHS**" faz a soma de 6 meses a **data\_ordered** (date\_ordered,"6") o seis faz referencia a seis meses.

```
SQL> select to_char(next_day(add_months
  2                               (date_ordered,6), 'friday'),
  3                               'day,month ddth,yyyy') " próxima
sexta daqui a 6 meses"
  4   from s_ord
  5   order by date_ordered;
```

próxima sexta daqui a 6 meses

---

friday	,march	05th,1993
friday	,march	12th,1993

16 rows selected.

Faça do site <http://www.geocities.com/hollywood/makeup/2765/bs.htm> o melhor local para baixar apostilas grátis

Envie uma pequena contribuição para

Leonardo R Motta

Rua Caxinava nº 56

Jardim Umarizal

São Paulo - SP

Informe também que apostila gostaria de poder baixar gratuitamente no site

Será pesquisado livros sobre temas escolhido pela grande maioria e suas apostilas disponibilizadas gratuitamente