**Manual do iniciante 3.0-SQL DEVELOPER**

**Comandos do banco de dados**

**Comandos citados**

**Nomenclatura criada para teste**

**TABLESPACE**

Espaço criados dentro do banco de dados como setores para organizar informações armazenado a TABLESPACE em pelo menos um arquivo.

Caminho dos arquivos de dados padrão do **ORACLE XE** é:

**C:\ORACLEXE\APP\ORACLE\ORADATA\XE**

Para criar uma **TABLESPACE** utiliza o comando **CREATE TABLE SPACE** definido o nome e o local onde deseja salvar o arquivo após **DATAFILE** dentro de aspas simples com o nome desejado do arquivo seguido de o tipo de arquivo (.DBF) me seguida definir o tamanho inicial do arquivo em megabytes no **SIZE** e de quanto em quantos megabytes ele vai crescer em **AUTOEXTEND ON NEXT** e seu tamanho em megabyte no **MAXSIZE.**

**CREATE TABLESPACE TS\_ESTOQUE**

**DATAFILE 'C:\ORACLEXE\APP\ORACLE\ORADATA\XE\TS\_ESTOQUE.DBF'**

**SIZE 50M AUTOEXTEND ON NEXT 10M MAXSIZE 200M;**

Para movimentar uma **TABLESPACE** de local primeiro tem que desativa com o comando **ALTER** **TABLESPACE** e colocá-la no modo off-line.

**ALTER TABLESPACE TS\_ESTOQUE OFFLINE;**

Em seguida utilizando o mesmo comando renomeia o caminho descrevendo o caminho antigo e o caminho novo.

**ALTER TABLESPACE TS\_ESTOQUE RENAME DATAFILE**

**'C:\ORACLEXE\APP\ORACLE\ORADATA\XE\TS\_ESTOQUE.DBF'**

**TO**

**‘C:\USUARIOS\JAILTONP\TS\_ESTOQUE.DBF’;**

Após definir novo local reativa a TABLESPACE com o comando

**ALTER TABLESPACE TS\_ESTOQUE ONLINE;**

**USUÁRIO**

Para criação do usuário utiliza o comando

**CREATE USER JAILTONP --nomenclatura do LOGIN**

**IDENTIFIED BY JAILTONP1 --definição da senha de acesso**

**DEFAULT TABLESPACE TS\_BENS --TABLESPACE o qual deseja do acesso**

**;**

**GRANTS**

Os acessos são dados através de GRANTS acesso esses de criação de banco de dados, tabelas, UPDATE, DELETE e ETC ...

Criação de seção

**GRANT**=permissão **CREATE SESSION**= criação de seção **TO**=para **JAILTONP**=LOGIN **WITH ADMIN OPTION**=podendo passa adiante a permissão

**GRANT CREATE SESSION TO JAILTONP WITH ADMIN OPTION;**

Permissão total na TABLESPACE

**GRANT UNLIMITED TABLESPACE TO JAILTONP WITH ADMIN OPTION;**

Permissão para criação de tabelas, procedure, view, triggers, sequence...

**GRANT CREATE TABLE, CREATE PROCEDURE TO JAILTONP WITH ADMIN OPTION;**

Permissão para projetar alterar inserir e deletar na tabela usar

**GRANT SELECT, INSERT, DELETE, UPDATE ON TB\_ALUNO TO JAILTONP WITH ADMIN OPTION;**

Podendo também da permissão especifica a coluna em INSERT, DELETE, UPDATE.

**GRANT UPDATE (NOME,NASCIDO) ON TB\_ALUNO TO JAILTONP WITH ADMIN OPTION;**

Para consultar permissões verificar na tabela DBA\_SYS\_PRIVS;

**SELECT \* FROM DBA\_SYS\_PRIVS;**

**REVOKE**

REVOKE funciona da maneira inversa do GRANT revogando as permissões determinadas exemplo de REVOKE.

**REVOKE UPDATE (NOME, CPF) ON TB\_CLIENTE FROM USUARIO WITH ADMIN OPTION;**

**TABELAS**

**TIPOS**

**VACHAR2** é tipo STRING usado no Oracle PL\SQL DEVELOPER ou SQL DEVELOPER resumindo o VARCHAR dos outros bancos aceitando também o uso do VARCHAR.

**NUMBER** é tipo numérico fracionado da ORACLE seguindo o mesmo padrão do **NUMERIC** do SQL SERVE ou **FLOAT** do MySQL.

**CONSTRAINT**

**CONSTRAINTS** são regras atribuída a colunas como **NOT NULL** onde definir que a coluna não pode receber valor nulo, **UNIQUE** definir que a coluna não repetirá o mesmo valor, **PRIMARY KEY** definir que aquela coluna será a coluna de identificação chave primaria onde não recebera valor nulo nem iguais, **FOREIGN KEY** definir que aquela coluna é uma coluna que faz referência a uma **PRIMARY** **KEY** em outra tabela ou até mesmo na mesma tabela.

**RENAME COLUMN**

Para alterar o nome de uma tabela faz uso dentro do ALTER TABLE do comando RENAME COLUMN nome da coluna e TO para nome desejado.

**ALTER TABLE TB\_PRODUTOS RENAME COLUMN SERIE TO SN;**

**SEQUENCE**

A **SEQUENCE** é a atribuição de números de formas automáticas tendo a mesma função que o **IDENTITY** do SQL SERVE ou o **AUTO\_INCREMENT** do MySQL sendo que o SEQUENCE ao invés de ser apenas uma CONSTRAINT ele é um objeto independente podendo ser usado por mais de uma tabela.

Comando de criação de SEQUENCE.

**CREATE SEQUENCE SQ\_SETOR**

**START WITH 1 =** determina a partir de qual número começara a incrementa

**INCREMENT BY 1** = determina os passos que será incrementado de 1 em 1 ou de 2 em 2**;**

Maneira de usar

**INSERT INTO TB\_SETOR VALUES (SQ\_SETOR.NEXTVAL,'RH');**

**OPERADORES**

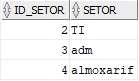
Clausura **WHERE** serve para determinar regras de projeção exemplo na tabela setor buscar onde o ID\_SETOR seja igual a dois.

**SELECT \* FROM TB\_SETOR WHERE ID\_SETOR = 2;**



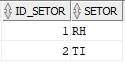
Operador **BETWEEN** serve para buscar valores entre o que for determinado podendo ser datas ou números, exemplo buscar onde ID\_SETOR esteja entre 2 e 3.

**SELECT \* FROM TB\_SETOR WHERE ID\_SETOR BETWEEN 2 AND 4;**



Operador **IN** serve para buscar vários valores variados de uma vez só no exemplo buscar onde SETOR seja RH ou TI.

**SELECT \* FROM TB\_SETOR WHERE SETOR IN ('RH','TI');**



Operador **LIKE** busca dados com parte das informações no exemplo buscar onde SETOR seja ALMOXARI levando em consideração que se não for definido valor antes ou depois do “%” ele trará qualquer coisa que existir.

**SELECT \* FROM TB\_SETOR WHERE SETOR LIKE 'ALMOXARI%';**

**SELECT \* FROM TB\_SETOR WHERE SETOR LIKE '%XARI%';**

**SELECT \* FROM TB\_SETOR WHERE SETOR LIKE '%XARIFADO';**



Operador **ISNULL** busca onde os dados são nulos, vazios.

**SELECT \* FROM TB\_SETOR WHERE SETOR IS NULL;**



Operador **AND** junta dois operadores ou mais em uma condição onde quando os dois ou mais indicadores forem verdadeiros tara valor.

**SELECT \* FROM TB\_FUNCIONARIO WHERE ATIVOS = ‘SIM’ AND FILIAL = 001**

Operador **OR** junta dois operadores ou mais em uma condição onde quando uma das duas forem verdadeiros tara valor.

**SELECT \* FROM TB\_FUNCIONARIO WHERE ATIVOS = ‘SIM’ OR FILIAL = 001**

Operador **NOT** inverte as condições dos outros operadores.

**SELECT \* FROM TB\_SETOR WHERE SETOR IS NOT NULL; ------**que não seja nulo

**SELECT \* FROM TB\_SETOR WHERE SETOR NOT IN ('RH','TI');** que não seja RH ou TI

**FUNÇÕES**

**LENGTH**

Função é utilizada para contar a quantidade de caracteres que contém dentro da célula podendo ser usado no WHERE.

**SELECT LENGTH('CONTAGEM') FROM DUAL;**



**UPPER**

Função usada para deixa todos caracteres da coluna em maiúsculas

**SELECT UPPER('maxlet') FROM DUAL;**



**LOWER**

Função usada para deixa todos caracteres da coluna em minúsculas

**SELECT LOWER('DIMIN') FROM DUAL;**



**INITCAP**

Função usada para deixa todas palavras da coluna coma a primeira letra em maiúscula.

**SELECT INITCAP ('LUIZ jailton') FROM DUAL;**



**DECODE**

Função usada para trocar valores da projeção de acordo com que é recebido exemplo se valor for ‘**SIM’** DECODE para ‘**ACEITO’**.

Exemplo DECODE (COLUNA, VALOR DA COLUNA, VALOR DESEJADO).

**SELECT DECODE (CONTRATO, ’SIM’, ‘ACEITO’, ‘NÃO’, ‘RECUSADO’) FROM TB\_FORMULARIO;**

**NVL**

Função usada para trocar valores da projeção quando o valor é nulo exemplo se valor for **NULL** troquepara ‘**VAZIO’**. NLV (COLUNA, ‘VALOR DESEJADO’).

**SELECT NVL (SALDO, ’LISO’) FROM TB\_CONTA**

**SUBSTR**

Função projetar os dados de acordo com as posições definido exemplo

**SUBSTR (COLUNA, COMEÇA, TERMINAR)** podendo usar apenas o começa a partir.

**SELECT**

**F.NOME,**

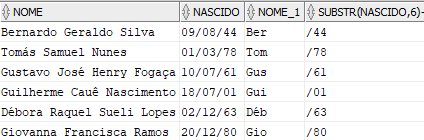
**F.NASCIDO,**

**SUBSTR (NOME,1,3) AS "NOME",**

**SUBSTR (NASCIDO, 3)**

**FROM**

**TB\_FUNCIONARIOS F;**



**DISTINCT**

Utiliza para quando os valores da projeção são exatamente iguais nas colunas declarada para trazer apenas uma linha.

**SELECT PRODUTO, SN FROM TB\_PRODUTO;**

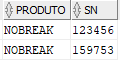
****

**SELECT DISTINCT PRODUTO FROM TB\_PRODUTO;**



Caso as colunas declaradas não tenha os mesmos valores será projetado as linhas de valores divergentes

**SELECT DISTINCT PRODUTO, SN FROM TB\_PRODUTO;**



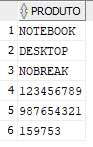
**UNION ALL**

**UNION ALL** serve para junta query onde as projeções das colunas são do mesmo tipo não sendo necessário ter a mesma quantidade de caracteres na coluna declarado ou ter o mesmo nome.

**SELECT PRODUTO FROM TB\_PRODUTO** PRODUTO VARCHAR2(20)

**UNION ALL**

**SELECT SN FROM TB\_PRODUTO;** SN VARCHAR2(21)



**UNION**

**UNION** serve para junta duas queries trazendo apenas os valores não repetidos onde as projeções de das colunas são do mesmo tipo não sendo necessário ter a mesma quantidade de caracteres na coluna declarado ou ter o mesmo nome.

**SELECT PRODUTO FROM TB\_PRODUTOS**

**UNION**

**SELECT PRODUTO FROM TB\_PRODUTOS;**



**TRIM**

Função **TRIM** serve para retira espaços das palavras que existir antes ou depois podendo usar a variação **LTRIM** para retirar apenas espaços a esquerda ou **RTRIM** para retirar espaço a direita

**SELECT TRIM (COLUNA) FROM TB\_ARQUIVOS;**

**RPAD-LPAD**

Definir a quantidade de caracteres deve ser exibido caso a projeção não alcance o valor desejado preencher com algum valor definido após a quantidade a direita usando **RPAD** ou a esquerda usando **LPAD,** LPAD(COLUNA, QUANTIDADE, VALOR).

**SELECT LPAD(SEXO,10,'\*') FROM TB\_SEXO;**



**SELECT RPAD(SEXO,10,'\*') FROM TB\_SEXO;**



**DUMP**

Utiliza para que possa saber os dados de tipo, quantidade de caráteres e a posição ASCII de cada caractere

**SELECT DUMP ('VAL') AS "PR1”, DUMP (12) AS"PR2" FROM DUAL;**



**TRANSLATE**

Utiliza para transforma valores de uma projeção ou variável independente de sua posição Exemplo palavra ‘ALUNO 02’ desejo transforma o A em O e o 2 em 1, (COLUNA,VALOR PROJETADO, VALOR NOVO)

**SELECT TRANSLATE ('ALUNO 02','2O','1A') AS "TR1" FROM DUAL;**



**NULLIF**

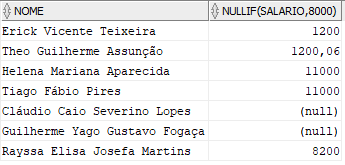
Utiliza para comparar valores e caso o valor comparado seja igual do parâmetro devolver o valor nulo **NULL** caso seja diferente trazer o valor real da projeção sem alterações.

**SELECT**

**NOME,**

**NULLIF (SALARIO,8000)** –todo salario que seja igual a 8000 deve vim nulo

**FROM TB\_FUNCIONARIO;**



**ABS**

Função ABS serve para converter números negativos para valor positivo exemplo de **-2** para **2;**

**SELECT ABS (-2.2) FROM DUAL;**



**TRUNC**

Função **TRUNC** serve para determinar a quantidade de casas decimais será usada exemplo o uso apenas do **TRUNC** retira todas as casas decimais, porém podemos definir a quantidade de casas decimais que desejamos usando parâmetro em seguida.

**SELECT**

**TRUNC (2.2),**

**TRUNC (10.123456,2)**

**FROM DUAL;**



**ROUND**

ROUND serve como para arredondar valores exemplos

Parâmetro negativo serve para arredondar números inteiros fazendo uso do **ROUND (VALOR,**

**-NÚMERO DE CASA QUE DESEJA ARREDONDAR)**

**SELECT ROUND (16,-1) FROM DUAL;**

****

**SELECT ROUND (14,-1) FROM DUAL;**

****

**SELECT ROUND (114,-2) FROM DUAL;**



Parâmetro positivo serve para arredondar números decimais fazendo uso do **ROUND (VALOR,**

**NÚMERO DE CASA QUE DESEJA ARREDONDAR)**

**SELECT ROUND (114.55,1) FROM DUAL;**



Caso não use os parâmetros de casas desejadas ira truncar o número e arredonda para valor mais próximo.

**SELECT ROUND (114.55) FROM DUAL;**



**SIGN**

Utilizado para valores numéricos para codificar em **1, -1,0, NULL** quando o valor é positivo retorna valor **1**, quando valor é negativo retorna valor **-1**, valores nulos retorno **NULL** e para valores 0 retornos **0**.

**SELECT SIGN (COLUNA\_NUMERICA) FROM DUAL;**

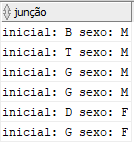
**CONCATENADOR**

|| serve para juntas as projeções fazendo mesma função que o sinal de mais (+) em outros bancos de dados exemplo de uso caso deseje juntar a projeção com alguma palavra ou outra coluna para que seja projetado em uma coluna só.

**SELECT**

**'INICIAL: '||SUBSTR(NOME,1,1)||' SEXO: '||SEXO AS "JUNÇÃO"**

**FROM TB\_FUNCIONARIOS;**



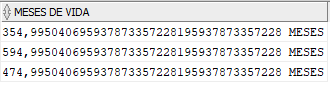
**MONTHS\_BETWEEN**

**MONTHS\_BETWEEN** serve para comparar um intervalor de datas dado como resultado o total de meses em número real.

**SELECT**

**MONTHS\_BETWEEN (SYSDATE, NASCIDO)||' MESES' AS "MESES DE VIDA"**

**FROM TB\_ALUNOS;**



Podendo usar **TRUNC** para trucar o valor numérico para valor inteiro.

**SELECT**

**TTRUC (MONTHS\_BETWEEN (SYSDATE, NASCIDO)) ||' MESES' AS "MESES DE VIDA"**

**FROM TB\_ALUNOS;**



**NEXT\_DAY**

Utiliza para determinar quando será o próximo dia da semana usando para metros de dias da semana exemplo segunda-feira=2 terça-feira=3...

**NEXT\_DAY (**data de parâmetro, dia da semana desejado**)**

**SELECT TO\_CHAR (SYSDATE,'DAY DD/MON/YYY') AS "DATA" FROM DUAL;**



**SELECT TO\_CHAR (NEXT\_DAY (SYSDATE,1),'DAY DD/MON/YYY') AS "DATA" FROM DUAL;**



**ADD\_MONTHS**

Função usada para projetar meses seguintes ou passado de acordo com o parâmetro determinado **ADD\_MONTHS (**data de parâmetro, quantidade positivo para meses à frente negativos para meses atrás**)**

**SELECT TO\_CHAR (SYSDATE, 'DAY DD/MON/YYYY') FROM DUAL;**

****

**SELECT TO\_CHAR (ADD\_MONTHS (SYSDATE, -2), 'DAY DD/MON/YYYY') FROM DUAL;**

****

**RANK-DENSE\_RANK**

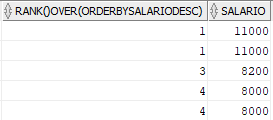
RANK serve para classificar as TUPLAS por determinada colunas exemplo em uma tabela de funcionários eu quero classificá-los pelo salário então faz uso do comando **RANK () OVER (ORDER BY COLUNA DESC);** onde determino o **RANK** jugando pela a ordenação da coluna desejada com ordem **DESC** degressiva onde quando duas tem o mesmo valor recebe o mesmo rank pulando o seguinte a qual iria ocupa.

**SELECT**

**RANK () OVER (ORDER BY SALARIO DESC),**

**SALARIO**

**FROM TB\_FUNCIONARIO;**



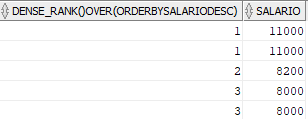
No caso do DENSE\_RANK funciona da mesma forma, porém caso exista valores iguais ele não sal o RANK seguinte.

**SELECT**

**DENSE\_RANK () OVER (ORDER BY SALARIO DESC),**

**SALARIO**

**FROM TB\_FUNCIONARIO;**



Outra variação do RANK é classificação (PARTITION BY) do RANK por outra coluna exemplo

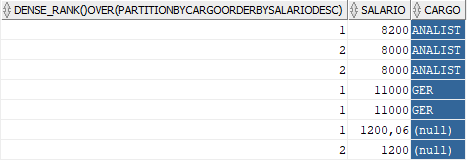
**SELECT**

**DENSE\_RANK () OVER (PARTITION BY CARGO ORDER BY SALARIO DESC),**

**SALARIO,**

**CARGO**

**FROM TB\_FUNCIONARIO;**



**Procedure**

Criação da proceure no DEVELOPER ocorre da mesma maneira dos outros bancos com diferença no modo que é declarado as variáveis tem que está entre parênteses e para variáveis que irá receber valores usar antes do tipo a clausula **IN** para as variáveis que irá devolver valores usa antes do tipo a clausula **OUT** e antes de declara uma variável dentro do código não precisa do uso do comando **DECLARE** apena a variável e seu tipo.

Uma diferencia no DEVELOPER é que nos **IF** sempre deve ser declaro o **THEN** antes das ações e ao finalizar o código deve fecha o **IF** com **END IF;** outro ponto é atribuição de valores que é feita através de “**: =** ” e nos **SELECT** atribuição de valores é feita através de **INTO** ao invés de usar apenas igual “**=**” entre a variável e a coluna sendo antes do **INTO** os que irão passar o valor e após os que irão receber o valor o valor.

Para printar uma informação na tela deve usar o comando **DBMS\_OUTPUT.PUT\_LINE ();** e dentro do parêntese a informação que deseja caso ao executar a procedure não tenha trago a informações na tela na página onde está executando execute o comando **SET SERVEROUTPUT ON;**

**CREATE OR REPLACE PROCEDURE PR\_SETOR (V\_SETOR IN VARCHAR2)**

**AS**

**V\_SETOR\_C VARCHAR2(20);** --declarando variável

**BEGIN**

**SELECT COUNT (SETOR) INTO V\_SETOR\_C FROM TB\_SETOR WHERE SETOR=UPPER (V\_SETOR);**

**IF (V\_SETOR\_C=0)**

**THEN**

**INSERT INTO TB\_SETOR VALUES (SQ\_SETOR.NEXTVAL, UPPER (V\_SETOR));**

**DBMS\_OUTPUT.PUT\_LINE ('SETOR '||UPPER(V\_SETOR)||' CADASTRADO');**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE ('ERRO: SETOR '||UPPER (V\_SETOR) ||' JÁ CADASTRADO');**

**END IF;**

**END PR\_SETOR;**

**/**

**TRIGGER**

As **TRIGGER** no **DEVELOPER** serve para acionar alguma ação sempre que houver umas das três opções **DELETE**, **UPDATE** ou **INSERT** na tabela vinculada antes ou depois de executa definido como **AFTER** para após e **BEFORE** para antes.

**CREATE OR REPLACE TRIGGER TG\_CARGOS**

**AFTER DELETE TB\_CARGOS**

**FOR EACH ROW**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE(‘AÇÃO NÃO PERMITIDA’);**

**ROLLBACK;**

**END;**

Para monitora mais de um tipo de ação e usasse a clausula **IF** para determina uma ação para cada tipo **DELETING**, **UPDATING** ou **INSERTING**, para que sejam coletados os valores antes das alterações utiliza a clausula **:OLD.** Seguido do nome da coluna da tabela que está sendo monitorada no caso dos novos valores **:NEW**.

**CREATE OR REPLACE TRIGGER TG\_CARGOS**

**AFTER DELETE TB\_CARGOS**

**FOR EACH ROW**

**BEGIN**

**IF DELETING**

**THEN**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE(‘AÇÃO NÃO PERMITIDA’);**

**ROLLBACK;**

**END;**

**ELSIF UPDATING**

**THEN**

**BEGIN;**

**INSERT INTO TB VALUES(:OLD.NOME,:OLD.SALARIO,:NEW.SALARIO)**

**END;**

**ELSIF INSERTING**

**THEN**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE(‘REGISTRADO’);**

**END;**

**END;**

**/**