GRADUAÇÃO

UniCesumar

Banco de Dados II

Prof. Msc. Aparecido Vilela Junior

aparecido.vilela@unicesumar.edu.br

Colunas Virtuais

Colunas virtuais

A habilidade para criar colunas virtuais é uma nova função do Oracle 11g.

Fornece a habilidade para definir uma coluna que contém dados derivados, dentro do banco de dados.

- Valores derivados para as colunas virtuais são calculados definindo um grupo de expressões ou funções que são associadas com as colunas virtuais.
- O ponto forte sobre as colunas virtuais é que elas não consumem qualquer espaço de armazenamento, visto que elas são computadas na execução.
- Pode-se utilizar as colunas virtuais praticamente em qualquer lugar que você usaria uma coluna normal.
  - Consultá-las;
  - Criar Índices;
  - Coletar Estatísticas;

Colunas virtuais

Os valores destas colunas não são armazenados fisicamente) mais conhecido entre os desenvolvedores de aplicações como "campos calculados".

Agora, teremos a opção de criar os campos calculados diretamente na tabela e assim, não ficar precisando mais gerar estas informações de forma codificada em nossas aplicações.

Em resumo, o Oracle fará automaticamente para nós o cálculo baseado em uma expressão definida na coluna em questão

Colunas virtuais

Existem algumas restrições que são:

Você não pode escrever em uma coluna virtual;

Não há suporte para as tabelas de índice organizado, externa, objeto, grupo ou temporárias;

Não há suporte para os tipos de dados atendidos pelo Oracle, tipos definidos pelos usuários, LOBs ou LONG RAWs.

Colunas virtuais - Exemplo


```
CREATE TABLE virtual (
cod             NUMBER,
nome            VARCHAR2(60),
sobrenome   VARCHAR2(60),
salario         NUMBER(9,2),
comissao1   NUMBER(3),
comissao2   NUMBER(3),
salario1    AS (ROUND(salario*(1+comissao1/100),2)),
CONSTRAINT pk_virtual PRIMARY KEY (cod));
```

Colunas virtuais – Exemplo02

```
CREATE TABLE t
 ( n1 INT
 , n2 INT
 , n3 INT GENERATED ALWAYS AS (n1 + n2) VIRTUAL);
```

Podemos ver que a coluna virtual é gerada a partir de uma simples expressão envolvendo as outras colunas na nossa tabela.

Note que a palavra-chave VIRTUAL é opcional.

Colunas Virtuais - Exercício

Crie uma tabela com uma coluna virtual, que traga o codigo do aluno, as notas do aluno (nota1 e nota2), utilize a coluna virtual para calcular a média, utilizando uma função.

Obs: na carga de uma função é interessante utilizar na função a cláusula RETURN <TIPO> DETERMINISTIC AS

Utilizando a tabela acima, crie também uma coluna virtual para apresentar o status do aluno (média <4 'Reprovado', média >= 4 e < 7 'Exame', media >= 7 'Aprovado'. Obs utilize CASE WHEN - ELSE

Resposta - 01

```
CREATE FUNCTION func_media(
p1 IN NUMBER,
p2 IN NUMBER ) RETURN NUMBER DETERMINISTIC AS
BEGIN
    RETURN (p1 + p2)/2;
END func_media;

CREATE TABLE NOTAS2 (
COD_ALUNO INTEGER,
NOTA1     NUMBER DEFAULT 0,
NOTA2     NUMBER DEFAULT 0,
MEDIA AS (func_media(nota1,nota2)))
```

Resposta 01

```
INSERT INTO NOTAS2 (COD_ALUNO, NOTA1, NOTA2) VALUES (1,80,40);
INSERT INTO NOTAS2 (COD_ALUNO, NOTA1, NOTA2) VALUES (2,60,60);
INSERT INTO NOTAS2 (COD_ALUNO, NOTA1, NOTA2) VALUES (3,90,80);
INSERT INTO NOTAS2 (COD_ALUNO, NOTA1, NOTA2) VALUES (4,85,75);
INSERT INTO NOTAS2 (COD_ALUNO, NOTA1, NOTA2) VALUES (5,40,30);

SELECT * FROM NOTAS2
```

Resposta - 02

```
ALTER TABLE NOTAS2
ADD STATUS VARCHAR2(20) AS
(CASE WHEN (FUNC_MEDIA(NOTA1,NOTA2) < 40) THEN 'REPROVADO'
     WHEN((FUNC_MEDIA(NOTA1,NOTA2) >=40) AND
    (FUNC_MEDIA(NOTA1,NOTA2) <70)) THEN 'EXAME'
                          ELSE 'APROVADO'
                     END)
/
```

Pivot | Unpivot Table

Com esse novo recurso do comando select podemos facilmente agregar linhas de uma tabela em formato de colunas (pivot) ou mesmo transformar dados organizados em colunas de uma tabela para serem retornados em forma de linhas (unpivot).

Pivot | Unpivot Table

```
SELECT * FROM Tabela
     PIVOT
    (sum(coluna)
FOR coluna IN (vl1,vl2,vl3,vl4, vn));
```

Pivot | Unpivot Table

```
SELECT * FROM tabela_unpivot
   UNPIVOT (valor FOR couna in vl1,vl2,vl3,vl4));
```

Exercício

Crie o Pivotamento da Tabela Funcionários trazendo por Departamento os cargos existentes.

Insira o resultado do Pivotamento acima em uma tabela, com o nome de TabPivoCargo

Utilize o UnPivot sobre a tabela acima criada para trazer os dados no formato linhas/colunas.

Resposta

```
select * from (
      select CD_DEPTO, NR_CARGO
      from FUNC F
   )
   pivot
   (
      count(NR_CARGO)
      for NR_CARGO in (54,42,43,53)
   )
   order by CD_DEPTO
```

Resposta

```
CREATE TABLE PIVOCARGO AS
    select * from (
      select CD_DEPTO, NR_CARGO
      from FUNC F
    )
    pivot
    (
      count(NR_CARGO)
      for NR_CARGO in (54,42,43,53)
    )
    order by CD_DEPTO
```

Respotas UNPIVOT

```
select *
    from PIVOCARGO
  unpivot
  (
     QTDE
     for NR_CARGO in ("54","42","43","53")
  )
  order by CD_DEPTO, NR_CARGO
```

ÍNDICE

Índices (index) são estruturas inseridas no BD com o objetivo de melhorar o desempenho de acesso às tabelas;

Tem a função de reduzir o I/O em disco localizando rapidamente os dados;

O ORACLE, cria automaticamente um índice do tipo Unique ao criar uma chave primária, o qual recebe o mesmo nome da constraint

ÍNDICE

Como são criados os índices:

- Automaticamente: quando você define uma primary key ou uma constraint unique, na definição de uma tabela
- Manualmente: Usuários podem criar índices não-únicos, para aumentar o tempo de acesso às linhas (registros)

ÍNDICE

Índices são utilizados durante comandos SELECT (order by e group by), portanto quando for composto, colocar primeiro a coluna mais usada;

Comando para a criação de Índices:

```
CREATE [UNIQUE] INDEX índice
ON TABELA (coluna) [ASC | DESC] [,coluna...]);
```

ÍNDICE

Opções	Descrição
UNIQUE	Informa que o índice em questão não aceitará valores repetidos
ASC	Especifica a classificação Ascendente (default)
DESC	Especifica a classificação descendente

ÍNDICE

Quando criar um índice ?

- Uma coluna contém uma grande escala de valores;
- Uma coluna contém um grande número de valores NULL;
- Uma ou mais colunas são frequentemente usadas na cláusula WHERE ou JOIN CONDITION;
- A tabela é grande e a maioria das pesquisas, tem a expectativa de recuperar menos de 2 a 4% das linhas

ÍNDICE

Quando NÃO criar um índice:

- A tabela é pequena;
- As colunas não são frequentemente usadas como uma condição nas pesquisas;
- Muitas pesquisas tem a expectativa de recuperar mais que 2 a 4% de linhas;
- A tabela é atualizada frequentemente;
- As colunas indexadas são refenciadas como parte de uma expressão

ÍNDICE

O dicionário de dados USER_INDEXES, contém o nome dos índices.

A tabela USER_IND_COLUMNS contém o nome do índice, o nome da tabela e o nome da coluna:

```
SELECT IC.INDEX_NAME, IC_COLUMN_NAME,
        IC.COLUMN_POSITION COL_POS, IX.UNIQUENESS
FROM USER_INDEXES IX, USER_IND_COLUMNS IC
WHERE IC.INDEX_NAME = IX.INDEX_NAME
AND  IC.TABLE_NAME = 'EMP'
```

ÍNDICE

A criação de índices também pode ser baseada em funções de expressão.

O índice de funções é construído de colunas de tabelas, constantes, funções SQL e funções definidas pelo usuário;

```
CREATE INDEX upper_emp_name_idx
ON EMP (UPPER(ENAME));
```

ÍNDICE

Facilita o processamento de consultas como:

SELECT * FROM emp WHERE UPPER(ename) = 'KING';

Para garantir que o Oracle use o índice em vez de desempenhar uma análise em toda a tabela, certifique-se de que o valor da função não seja nulo em consultas subseqüentes. Por exemplo, a instrução abaixo certamente usará o índice, mas sem a cláusula WHERE o Oracle executará uma análise em toda a tabela.

```
SELECT * FROM emp
WHERE UPPER (ename) IS NOT NULL
ORDER BY UPPER (ename);
```

O Oracle trata os índices com colunas marcadas como DESC como índices baseados em função. As colunas marcadas como DESC são classificadas em ordem decrescente.

ÍNDICE

Para remover um índice, do dicionário de dados do BD, utilizamos o comando DROP.

```
DROP INDEX Nome_Índice
```

BULK BINDS

- **A associação de valores a variáveis de PL/SQL em comandos de SQL é chamada Bind.**
- **A associação de uma coleção inteira de uma única vez é chamada Bulk Bind.**
- **O objetivo de um Bulk Bind é aumento de performance.**

BULK BINDS

- DECLARE
  - TYPE L_NUM IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
  - WCD_LISTA L_NUM;
- BEGIN
  - FOR I IN 1..2000 LOOP
    - WCD_LISTA(I) := I;
  - END LOOP;
- FOR I IN 1..2000 LOOP
  - DELETE FROM FUNC WHERE CD_MAT = WCD_LISTA(I);
- END LOOP;
- END;

BULK BINDS

- **No exemplo, o comando Delete é executado 2.000 vezes; isto significa que o PL/SQL (Engine) envia um comando SQL Delete para cada valor da lista.**
- **Cada vez que a PL/SQL Engine tem necessidade de estabelecer uma interrupção para acionar a SQL Engine, ocorre um overhead.**
- **Sendo assim, quando trabalhamos com coleções (Nested Tables, Index-By, Varray e Host Arrays) e utilizamos iterações (loops) usando elementos destas coleções como variáveis Bind (em comandos de SQL), adicionamos um overhead à nossa execução.**
- **Se a iteração afetar cinco ou mais linhas do banco de dados, o uso de Bulk Binds pode aumentar a performance consideravelmente.**

BULK BINDS - FORALL

- DECLARE
  - TYPE L_NUM IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
  - WCD_LISTA L_NUM;
- BEGIN
  - FOR I IN 1..2000 LOOP
  - WCD_LISTA(I) := I;
- END LOOP;
- FORALL I IN 1..2000
  - DELETE FROM FUNC WHERE CD_MAT = WCD_LISTA(I);
- END;

BULK BIND – FORALL (Exemplo)

- No exemplo, a coleção Wcd_Lista é passada inteira de uma única vez para a SQL Engine.

- CREATE TABLE LISTA
- (CD_LISTA NUMBER,
- TX_LISTA VARCHAR2(200));

BULK BIND – FORALL (Exemplo)

```
DECLARE
    TYPE L_NUM IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
    TYPE L_CHAR IS TABLE OF VARCHAR2(200) INDEX BY BINARY_INTEGER;
    WCD_LISTA L_NUM;
    WTX_LISTA L_CHAR;
    T1 NUMBER;
    T2 NUMBER;
    T3 NUMBER;
    MSG VARCHAR2(2000);
BEGIN
    — Preenche as tabelas em memória —
FOR I IN 1..20000 LOOP
    WCD_LISTA(I) := I;
    WTX_LISTA(I) := 'TEXTO DA LINHA ' | | TO_CHAR(I);
    END LOOP;
    — Inclui as linhas na tabela usando For Loop
    SELECT TO_CHAR(CURRENT_TIMESTAMP, 'SSSSS') INTO T1 FROM DUAL;
    FOR I IN 1..20000 LOOP
        INSERT INTO LISTA VALUES (WCD_LISTA(I), WTX_LISTA(I));
    END LOOP;
```

BULK BIND – FORALL (Exemplo)

```
— Inclui as linhas na tabela usando ForAll
SELECT TO_CHAR(CURRENT_TIMESTAMP, 'SSSSS') INTO T2 FROM DUAL;
FORALL I IN 1..20000
    INSERT INTO LISTA VALUES(WCD_LISTA(I),WTX_LISTA(I));
SELECT TO_CHAR(CURRENT_TIMESTAMP, 'SSSSS') INTO T3 FROM DUAL;
MSG := 'Tempo para Insert com For Loop = ' || TO_CHAR(T2 - T1) || CHR(10) ||
'Tempo para Insert com ForAll = ' || TO_CHAR(T3 - T2);
DBMS_OUTPUT.PUT_LINE(MSG);
END;
/
```

BULK BIND – FORALL (Exemplo)

- Como características ou restrições, temos:
  - O comando SQL presente na sintaxe pode ser Insert, Update ou Delete, fazendo referência a elementos da coleção (a SQL Engine executa um comando Select para cada índice no intervalo).
  - As fronteiras (<valor inferior> e <valor superior>) devem especificar um intervalo válido de índices numéricos consecutivos (não precisa ser a coleção inteira, pode ser parte da coleção desde que seja consecutiva).
  - Todos os elementos da coleção no intervalo especificado devem existir.
  - O subscrito da coleção não pode ser uma expressão, e portanto o texto Where cd_mat = wcd_lista(I + 1) é inválido.
  - Antes de cada comando SQL executado pelo ForAll, é criado um SavePoint implícito. Desta forma, se no comando ForAll de um determinado programa a terceira execução (ou a utilização do terceiro índice da lista) falha, a primeira e segunda execuções do comando SQL associado não são desmanchadas; apenas a partir do terceiro índice a ação falha.

CLÁUSULA BULK COLLECT

- A cláusula Bulk Collect indica à SQL Engine para preparar toda a coleção de saída (Bulk Bind) antes de retorná-la para a PL/SQL Engine.
- Esta cláusula pode ser usada junto com a cláusula INTO nos comandos Select Into, Fetch Into e Returning Into.
- Na Listagem a seguir, utilizamos a cláusula Bulk Collect Into em um comando Select.
- Observe que a coleção (Index-By Table) não precisou ser inicializada.
- A SQL Engine realiza esta operação implicitamente.

CLÁUSULA BULK COLLECT

- Bulk Collect com Select
- DECLARE
  - TYPE L_NUM IS TABLE OF LISTA.CD_LISTA%TYPE INDEX BY BINARY_INTEGER;
  - TYPE L_CHAR IS TABLE OF LISTA.TX_LISTA%TYPE INDEX BY BINARY_INTEGER;
  - WCD_LISTA L_NUM;
  - WTX_LISTA L_CHAR;
  - MSG VARCHAR2(2000);
- BEGIN
  - SELECT CD_LISTA,TX_LISTA
  - BULK COLLECT INTO WCD_LISTA,WTX_LISTA FROM LISTA;
  - MSG := 'Códigos lidos: ';
  - FOR I IN 20000..20005 LOOP
    - MSG := MSG || WCD_LISTA(I) || '; ';
  - END LOOP;
- END;

Uso do Bulk

- **Converta suas rotinas de carga de dados – copia de uma tabela para outra, com base em Bulk Collect.**

Resolvendo

- CREATE TABLE **EXER1 (**
- **COD_NUMBER** NUMBER**(5),**
  DESCRICAO VARCHAR2**(40),**
  CONSTRAINT **EXER1_PK** PRIMARY KEY **(COD_NUMBER));**
- **Inserindo os dados na tabela:**
- BEGIN
  INSERT INTO **EXER1**
  SELECT LEVEL**, DBMS_RANDOM.**STRING**('A', 40)** FROM **DUAL** CONNECT BY LEVEL **<= 100;**
- COMMIT**;**
  END**;**

Resolvendo

- **Vamos agora usar o Bulk Collect, colocando um limite de 70 registros. Isso possibilita colocar a cada loop apenas 70 registros na memória.**
- DECLARE
  CURSOR **CUR_EXER** IS
  SELECT **COD_NUMBER, DESCRICAO** FROM **EXER1;**

- **--CRIANDO O TIPO PARA A COLEÇÃO**
  TYPE **TROW_EXER1** IS TABLE OF **CUR_EXER%**ROW
  TYPE INDEX BY PLS_INTEGER**;**
- **--DECLARANDO A COLEÇÃO ROW_TAB19X**
- **ROW_EXER1 TROW_EXER1;**

Resolvendo

- BEGIN
  OPEN **CUR_EXER;**
  LOOP
    FETCH **CUR_EXER** BULK COLLECT
    INTO **ROW_EXER1** LIMIT **70;**
- EXIT WHEN **ROW_EXER1.**COUNT = **0;**

- FOR **I** IN1 **.. ROW_EXER1.**COUNT LOOP
  DBMS_OUTPUT.PUT_LINE('COD:' \|\| ROW_EXER1(I).COD_NUMBER \|\| ' DESCRICAO:' \|\| ROW_EXER1(I).DESCRICAO);
  END LOOP;
  END LOOP;
  CLOSE **CUR_EXER;**
  END;

Para Entregar -

- **Crie um bloco de PL/SQL que receba como parâmetro uma série de matrículas separadas por vírgula em um único parâmetro.**
- **Monte este número em um array e faça uma consulta ao banco de dados, obtendo o nome e salário de todos os funcionários que existam na lista.**
- **Use ForAll e Bulk Collect.**

```
DECLARE
 MATS           VARCHAR2(300) := '&MATRICULAS' | | ',';
 TYPE TAB       IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
 TYPE TABCHAR   IS TABLE OF VARCHAR2(50) INDEX BY BINARY_INTEGER;
 TABMAT         TAB;
 SALARIOS       TAB;
 NOMES          TABCHAR;
 POS            NUMBER;
 I              NUMBER := 1;
BEGIN
 POS := INSTR(MATS, ',');
 WHILE POS > 0 LOOP
  TABMAT(I) := TO_NUMBER(SUBSTR(MATS,1,POS-1));
   MATS := SUBSTR(MATS,POS+1);
   POS := INSTR(MATS, ',');
   I := I + 1;
 END LOOP;
 :MSG := '';
 IF I > 1 THEN
   FORALL J IN 1..TABMAT.LAST
     UPDATE FUNC SET VL_SAL = VL_SAL
      WHERE CD_MAT = TABMAT(J)
```

Tabelas Temporárias

Tabelas Temporárias

As tabelas temporárias estão disponíveis desde o 8i.

Elas são temporárias no que diz respeito aos dados que armazenam, não à definição da tabela propriamente dita.

O comando create global temporary table cria uma tabela desse tipo.

Tabelas Temporárias

Existem dois tipos diferentes de dados em uma tabela temporária:

- Temporários em relação à duração da transação
- Temporários em relação à duração da sessão.

A longevidade dos dados temporários é controlada pela cláusula on commit;

- on commit delete rows: remove todas as linhas da tabela temporária quando um comando commit ou rollback é emitido;
- on commit preserve rows mantém as linhas na tabela além do limite da transação.

Obs: quando a sessão do usuário termina, todas as linhas do usuário na tabela temporária são removidas.

Tabelas Temporárias

Para criar uma tabela temporária use o comando create global temporary table.

Para excluir automaticamente as linhas no final da transação, especifique on commit delete rows:

```
CREATE GLOBAL TEMPORARY TABLE TEMP_ESOFT (
NOME VARCHAR2(25), ENDERECO VARCHAR2(25),
CIDADE VARCHAR2(25))
ON COMMIT DELETE ROWS;
```

Tabelas Temporárias

Pode-se inserir linhas emTEMP_ESOFT durante o processamento da aplicação.

Quando efetuar um commit, o Oracle truncará TEMP_ESOFT.

Da perspectiva do DBA, precisamos saber se os desenvolvedores da aplicação estão usando estes recursos.

É necessário considerar o espaço requerido pelas tabelas temporárias durante o processamento.

As tabelas temporárias, em geral, são usadas para melhorar as velocidades de processamento de transações complexas.

Equilibrando o benefício de desempenho com os custos de espaços.

Pode-se criar índices em tabelas temporárias para melhorar ainda mais o desempenho do processamento, novamente com o custo do aumento do uso do espaço.

Tabelas Temporárias

Obs: As tabelas temporárias e seus índices não alocam espaço até que a primeira instrução insert ocorra.

Quando não são mais usadas, o espaço utilizado por elas é deslocado.

Expressões Regulares

Expressões Regulares

Expressões regulares foram criadas no Oracle 10g e agora a versão 11g trouxe algumas melhorias e uma nova função.

Estas expressões são uma poderosa ferramenta para se trabalhar com os seguintes tipos de dados: CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2 e VARCHAR2.

Expressões regulares são muito comuns em ferramentas UNIX para trabalhar com dados do tipo caractere.

Podem ser usados para encontrar, substituir, procurar por espaços em branco, etc., ou seja, qualquer tipo de manipulação de caracteres e agora, este tipo de operação foi trazida para o banco de dados.

Expressões Regulares

As funções são: REGEXP_LIKE, REGEXP_INSTR, REGEXP_SUBSTR, REGEXP_REPLACE e REGEXP_COUNT, este último sendo novo na versão 11g.

Podemos relacionar estas expressões com as funções mais conhecidas como LIKE, INSTR, SUBSTR e REPLACE, onde a diferença é que as expressões regulares são muito mais poderosas.

Para usar as expressões regulares, temos que conhecer os meta-caracteres

Expressões Regulares

GRADUAÇÃO UniCesumar

(. - ponto): usado para encontrar qualquer caractere na posição indicada. Exemplo: r.d – significa que quero encontrar qualquer padrão em que haja a letra “r” seguida de qualquer caractere e a letra “d” em seqüência, como “**rod**”, “**Rod**rigo”, “p**rod**uction”;

(^ - circunflexo): usado para apontar o início de uma linha. Exemplo: ^rod encontrará “Rodrigo”, “rod”;

($ - cifrão): usado para apontar o final de uma linha. Exemplo: rod$ encontrará “prod”, “rod”;

(* - asterisco): usado para encontrar múltiplos caracteres, similar ao (% - porcentagem) usado na função LIKE.

Expressões Regulares - Exemplo

```
select nm_func, nm_sobrenome,dt_nasc
    from func
    where REGEXP_LIKE(TO_CHAR(DT_NASC,'YYYY'),'^197[0-5]$');
```

Retorne todos os funcionários (nome, sobrenome, data de nascimento), cujo ano de nascimento se inicie com 197(“^197”) e termine (“$”) com um elemento de 0 até 5 (“[0-5]$”).

Expressões Regulares - Exemplo

```
select nm_func, nm_sobrenome, dt_nasc
    from func
    where regexp_like(nm_sobrenome,'^p', 'i');
```

Outro exemplo de recuperação de registros dos funcionários, podemos ter a necessidade de recuperar em uma base qualquer os funcionários cujo sobrenome se iniciem com ‘p’ ou ‘P’, realmente um exemplo simples

Expressões Regulares – Desafio I

Recuperar os registros dos funcionários que se chamem Flávio ou Flavio, com Expressão Regular.

Resposta

```
select nm_func, nm_sobrenome, dt_nasc
   from func
   WHERE REGEXP_LIKE(nm_func,'^FL[aá]', 'i');
```

Expressões Regulares - Exemplo

Continuando com nossos exemplos, podemos trazer uma seleção de linhas em que o nome possua exatamente cinco letras:

```
select nm_func, nm_sobrenome, dt_nasc
   from func
   WHERE REGEXP_LIKE(nm_func,'^.....$','i');
```

Expressões Regulares - Exemplos

Podemos observar que o número de caracteres desejadas também poderia ser expresso da seguinte maneira, no bloco WHERE da nossa expressão:

```
REGEXP_LIKE(nome, '^.{5}$','i');
```

Expressões Regulares – Desafio II

Nesse momento vamos supor que queremos procurar dois nomes ao mesmo tempo, ou ‘joao’ ou ‘maria’:

Resposta

```
select nm_func, nm_sobrenome, dt_nasc
   from func
   WHERE
   REGEXP_LIKE(nm_func,'(joao|maria)','i');
```

Expressões Regulares - Exemplo

```
create table toys
(id number,
descr varchar2(20));

insert into toys values (1,'TOY1');
insert into toys values (2,'T1OY2');
insert into toys values (3,'1TOY1');
insert into toys values (4,'999');
insert into toys values (5,'888');
insert into toys values (6,'TOY2007');
COMMIT;
```

Expressões Regulares – Exemplo

REGEXP_LIKE

```
SELECT DESCR
FROM TOYS
WHERE REGEXP_LIKE(DESCR, '[^[:digit:]]');
```

Busca por registros cuja coluna DESCR contenha caracteres alfanuméricos.

Expressões Regulares – Exemplo REGEXP_COUNT.

```
select REGEXP_COUNT('Rodrigo Righetti', 'i', 2, 'c')
from dual;
```

procura pelo caractere “i”, iniciando sua busca na segunda posição e apenas caracteres minúsculos (case sensitive, a opção 'c' da expressão).

```
select REGEXP_COUNT('Ricardo Rezende', 'R', 1, 'c')
from dual;
```

procuramos por ocorrências do caractere “R”, iniciando a busca na primeira posição e apenas caracteres maiúsculos

Expressões Regulares – Exemplo REGEXP_REPLACE.

A função regexp_replace é semelhante a função REPLACE e TRANSLATE, exceto que ele substitui um padrão de cadeia, especificado com uma expressão regular, em vez de uma string literal.

```
SELECT REGEXP_REPLACE('Rodrigo     Righetti','( ){5,}', ' ')
FROM dual;
```

procuramos por 5 caracteres de espaço substituindo-os por apenas um caractere de espaço.

REGEXP_REPLACE - Exemplo

Uma troca comum, como uma string "bc" no final de uma cadeia, não pode ser facilmente realizado com um único comando de REPLACE ou TRANSLATE

O Regexp_replace lida com tal padrão corresponde facilmente.

```
Regexp_replace (Texto,'bc $ ', ' XY ')
```

Expressões Regulares – Desafio III

Utilize RegExp_Replace para listar na descrição somente os números. Utilize a tabela criada anteriormente (Toys).

Utilizar Expressão Regular para extrair somente números de um cpf (tabela Func)

Respostas

Utilize RegExp_Replace para listar na descrição somente os números.
Utilize a tabela criada anteriormente (Toys).

```
SELECT REGEXP_REPLACE(DESCR,'[^[:digit:]]')
    FROM TOYS
```

Utilizar Expressão Regular para extrair somente números de um cpf.
(Tabela Func)

```
SELECT REGEXP_REPLACE(nr_cpf,'[^[:digit:]]') FROM func;
```

REGEXP_INSTR

Objetivo: Ele procura uma string para uma determinada ocorrência de expressão regular em busca de uma posição de início especificada.

Retorna um inteiro que indica a posição na sequência em que a correspondência é encontrada.

Sintaxe: REGEXP_INSTR (SourceString, Pattern [, Posição, ocorrência, MatchOption])

REGEXP_INSTR – Desafio IV

Informe a posição do primeiro caracter numérico da expressão abaixo, utilizando REGEXP_INSTR;

‘AStringWithNumbers1234 etc’

```
Select RegExp_Instr('AStringWithNumbers1234 etc','[^[:alpha:]]')
From Dual;
```

Expressões Regulares – Exemplo REGEXP_COUNT.

```
select REGEXP_COUNT('Rodrigo Righetti', 'i', 2, 'c')
from dual;
```

procura pelo caractere “i”, iniciando sua busca na segunda posição e apenas caracteres minúsculos (case sensitive, a opção 'c' da expressão).

```
select REGEXP_COUNT('Ricardo Rezende', 'R', 1, 'c')
from dual;
```

procuramos por ocorrências do caractere “R”, iniciando a busca na primeira posição e apenas caracteres maiúsculos

Expressões Regulares – Exemplo REGEXP_REPLACE.

A função regexp_replace é semelhante a função REPLACE e TRANSLATE, exceto que ele substitui um padrão de cadeia, especificado com uma expressão regular, em vez de uma string literal.

```
SELECT REGEXP_REPLACE('Rodrigo     Righetti','( ){5,}', ' ')
FROM dual;
```

procuramos por 5 caracteres de espaço substituindo-os por apenas um caractere de espaço.

REGEXP_REPLACE - Exemplo

Uma troca comum, como uma string "bc" no final de uma cadeia, não pode ser facilmente realizado com um único comando de REPLACE ou TRANSLATE

O Regexp_replace lida com tal padrão corresponde facilmente.

```
Regexp_replace (Texto,'bc $ ', ' XY ')
```

Expressões Regulares – Desafio III

Utilize RegExp_Replace para listar na descrição somente os números. Utilize a tabela criada anteriormente (Toys).

Utilizar Expressão Regular para extrair somente números de um cpf (tabela Func)

Respostas

Utilize RegExp_Replace para listar na descrição somente os números.
Utilize a tabela criada anteriormente (Toys).

```
SELECT REGEXP_REPLACE(DESCR,'[^[:digit:]]')
    FROM TOYS
```

Utilizar Expressão Regular para extrair somente números de um cpf.
(Tabela Func)

```
SELECT REGEXP_REPLACE(nr_cpf,'[^[:digit:]]') FROM func;
```

REGEXP_INSTR

Objetivo: Ele procura uma string para uma determinada ocorrência de expressão regular em busca de uma posição de início especificada.

Retorna um inteiro que indica a posição na sequência em que a correspondência é encontrada.

Sintaxe: REGEXP_INSTR (SourceString, Pattern [, Posição, ocorrência, MatchOption])

REGEXP_INSTR – Desafio IV

Informe a posição do primeiro caracter numérico da expressão abaixo, utilizando REGEXP_INSTR;

‘AStringWithNumbers1234 etc’

```
Select RegExp_Instr('AStringWithNumbers1234 etc','[^[:alpha:]]')
From Dual;
```