DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Relatório Projeto BDDAD

Funções e Procedimentos PL/SQL para Base de Dados

Turma 2DB
1180017 - Gabriel Pelosi
1171250 – Rogério Alves
1151352 – João Mata
1181845- Gonçalo Ribeiro

Docente
Angelo Manuel Rego E Silva Martins

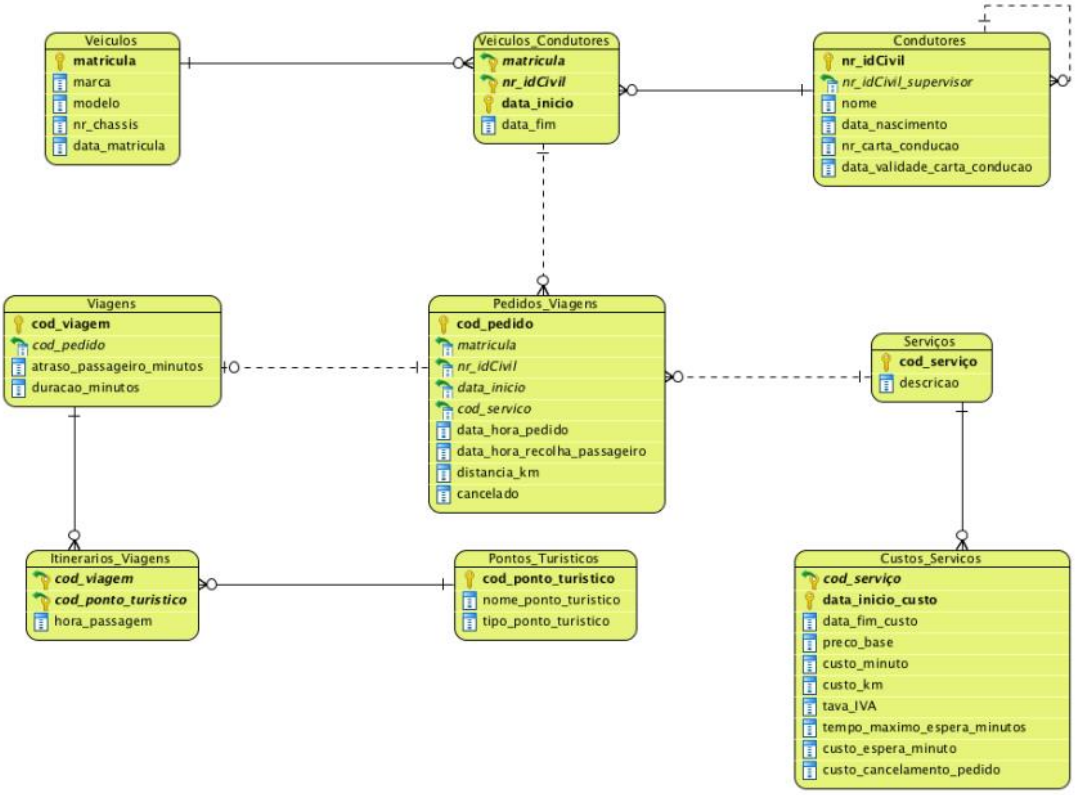
Unidade Curricular
Base de Dados (BDDAD)

Porto, 09 de Novembro 2019

isep Instituto Superior de Engenharia do Porto

Introdução

Este projeto foi desenvolvido por alunos do 2º ano do curso de Engenharia Informática do ISEP relativamente a diciplina Base de Dados,que aborda a linguagem de consulta estruturada PL/SQL. O trabalho prático teve como objetivo criar funções e procedimentos para realizar consultas e obter informações relativas as tabelas criadas e preenchidas na primeira parte do enunciado, isso tudo através do Oracle SQL developer.



Resultados e Justificações

Exercício 1

Para resolver o exercício 1, foi criada uma função que recebia um período(data inicio e data fim), um tipo de servico e um numero inteiro(n) como parameto,no seu primeiro bloco, foi declarado o cursor, que será retornado, no segundo bloco (“beguin”) as excecoes foram criadas e o cursor foi preenchido com a informacao solicitada no enunciado e logo em seguida, retornado.

```
create or replace function funcTopServico(tipo_servico
servicos.cod_servico%TYPE, periodoI date, periodoF date, n int)
return SYS_REFCURSOR as
   id_cur SYS_REFCURSOR;
begin
   --exception check variables--
   IF tipo_servico IS NULL OR periodoI IS NULL OR periodoF IS NULL THEN RAISE
no_data_found;
   END IF;
   IF n <= 0 THEN RAISE no_data_found; END IF;
   IF periodoF < periodoI THEN RAISE no_data_found; END IF;
   ------------------
      OPEN id_cur FOR
      SELECT DISTINCT
         c.nr_idcivil
      FROM
         condutores c, custos_servicos cs, pedidos_viagens pv, viagens v, servicos
s
      WHERE  s.cod_servico = tipo_servico
      AND (pv.data_inicio BETWEEN periodoI AND periodoF )
      GROUP BY c.nr_idcivil
```

```
    ORDER BY sum(
      CASE
        WHEN pv.cancelado = 'nao' THEN
          cs.preco_base+(cs.custo_min *
v.duracao_minutos)+(cs.custo_km*pv.distancia_km)
            +(cs.custo_espera_minuto*v.atraso_passageiro_minutos)
            + (cs.preco_base+(cs.custo_min *
v.duracao_minutos)+(cs.custo_km*pv.distancia_km)

+(cs.custo_espera_minuto*v.atraso_passageiro_minutos))*cs.taxa_iva
        ELSE
          cs.custo_cancelamento_pedido
      END
      )
    fetch first n rows only;
    return id_cur;
    EXCEPTION WHEN no_data_found THEN return NULL;
      end funcTopServico;
```

Exercício 2

Para resolver o exercício 2, foi criada uma funcao que não recebe paraetos de entrada e retorna um valor booelano. No seu primeir bloco de execucao, são craidas as variaveis e um cursor, no bloco “beguin”, a funcao busca as maticulas cujo apresentam uma sobreposicao temporal e quardam essa quantidade em uma variavel(maticulas_sobre), caso pelo menos uma seja adicionada, a funcao retorna true, caso nenhuma seja adicionada, retorna false.

```
create or replace function funcSobreposicoesVeiculosCondutores return
boolean  as
  bool boolean;
  inicio date;
  fim date;
  matricula_t veiculos_condutores.matricula%type;
  matricula_sobre veiculos_condutores.matricula%type;
```

```
   cursor cursors is
      select distinct vc.matricula, vc.data_inicio, vc.data_fim from
veiculos_condutores vc;
begin
   open cursors;
   --matricula_sobre := 'AA-AA-AA';
   bool :=FALSE;
loop
   fetch cursors into matricula_t, inicio, fim;

  select count(matricula) into matricula_sobre
  from veiculos_condutores
  where matricula=matricula_t
  and data_inicio BETWEEN inicio and fim or data_fim BETWEEN inicio and fim;
      if matricula_sobre >1  then
         bool:= true;
      end if;
   exit when cursors%notfound;
end loop;

  return bool;
end;
```

Exercício 3

Para resolver o exercício 3, foi criada uma função chamada FUNCOBTERINFOSEMANALVEICULOS, a qual recebe uma data por parametro e retorna um cursor com todas as informações relativas aos veiculos na base de dados em que o numero de viagens e quiloemtros e tempo gasto são relativos a data passada por parametro, ou seja, a função apenas busca as informacoes dos veiculos de uma ou mais semanas recebidas por parametro.

```
create or replace FUNCTION FUNCOBTERINFOSEMANALVEICULOS(dat DATE)
RETURN SYS_REFCURSOR AS
  matricula  veiculos.matricula%type;
   dataInicial DATE;
    dataFinal DATE;
    nr_viagens number;
    nr_kilometros number;
    tempo_gasto number;



   cursors   SYS_REFCURSOR;
begin
dataInicial := TRUNC(dat, 'iw') ;
dataFinal := TRUNC(dat, 'iw')+ 7 - 1/86400 ;


OPEN cursors for SELECT vc.matricula, vc.data_inicio , vc.data_fim,
count(pv.cod_pedido) as numero_viagens ,sum(pv.distancia_km) as
distancia_total, sum(v.duracao_minutos) as duracao_viagens  FROM
veiculos_condutores vc, pedidos_viagens pv, viagens v where
pv.matricula=vc.matricula and v.cod_pedido=pv.cod_pedido and
pv.data_hora_pedido BETWEEN dataInicial and dataFinal group by vc.matricula,
vc.data_inicio,  vc.data_fim;


  RETURN cursors;
END ;
```

Exercício 4

```
Create OR REPlACE PROCEDURE procAtualizarCustosServico(data_up in DATE,
percen in number) AS
custo Number;
data_ultimo_update DATE;
upd_per int;
```

```
cod_servico_maior_vol_neg servicos.cod_servico%type ;
data_inicio_ser_maior_vel custos_servicos.data_inicio_custo%type;
time_exception EXCEPTION;
preco_base_up custos_servicos.preco_base%type;
custo_minuto_up custos_servicos.custo_min%type;
custo_espera_minuto_up custos_servicos.custo_espera_minuto%type;
custo_cancelamento_pedido_up
custos_servicos.custo_cancelamento_pedido%type;
```

*Aqui declarei as variáveis e os seus tipos, agora vou fazer uma pesquisa sql para me dar o cod do serviço que teve um maior volume de negócios na semana que quero analizar e ponho o código na variável cod_servico_maior_vol_neg,custo.

```
 begin

select  cod_servico as  "Cod_servico",   NVL((select sum(
cs.preco_base+(cs.custo_min *
v.duracao_minutos)+(cs.custo_km*pv.distancia_km)
+(cs.custo_espera_minuto*v.atraso_passageiro_minutos)+
(cs.preco_base+(cs.custo_min *
v.duracao_minutos)+(cs.custo_km*pv.distancia_km)
+(cs.custo_espera_minuto*v.atraso_passageiro_minutos))*cs.taxa_iva)
from  custos_servicos cs, pedidos_viagens pv, viagens v
where  EXTRACT(YEAR FROM pv.data_hora_pedido)BETWEEN  EXTRACT(YEAR
FROM(data_up)) and  (EXTRACT(YEAR FROM(data_up))-1) and pv.cod_servico =
a.cod_servico and cs.cod_servico=a.cod_servico and
v.cod_pedido=pv.cod_pedido and pv.data_hora_pedido>= sysdate - interval '1'
year), 0 ) as "custo_total" into  cod_servico_maior_vol_neg,custo
from servicos a
```

```
where rownum<=1
order by "custo_total" desc;
```

*Aqui vou selecionar a data_inicio_custo do serviço obtido anteriormente e guardar na variável data_ultimo_update

```
select cs.data_inicio_custo into data_ultimo_update from custos_servicos cs, servicos s where cs.cod_servico = s.cod_servico and cod_servico_maior_vol_neg=s.cod_servico ;
```

*Aqui verifico se o serviço já teve o seu custo atualizado á mais de 6 meses, se for verdade, atualizo o serviço, se o custo do serviço foi atualizado á menos de 6 meses levanto uma exceção.

```
if MONTHS_BETWEEN (current_date, data_ultimo_update ) < 6 then
raise time_exception;
else
```

*Aqui faço o update dos preços da tabela custos_serviços e actualizo também o atributo data_inicio_custo para a data actual

```
  update Custos_servicos
 SET preco_base=preco_base+((percen/100)*preco_base),
custo_min=custo_min+((percen/100)*custo_min),
custo_espera_minuto=custo_espera_minuto+(custo_espera_minuto*(percen/100)),
custo_cancelamento_pedido=custo_cancelamento_pedido+(custo_cancelamento_pedido*(percen/100)), data_inicio_custo=(CURRENT_DATE)
 where  cod_servico= cod_servico_maior_vol_neg;
```

*Aqui ponho os novos preços em variáveis.

```
select preco_base,
custo_min,custo_espera_minuto,custo_cancelamento_pedido into
preco_base_up,
custo_minuto_up,custo_espera_minuto_up,custo_cancelamento_pedido_up
from Custos_servicos cs where cs.cod_servico=cod_servico_maior_vol_neg ;
```

*Aqui imprimo os novos preços.

```
 dbms_output.put_line('Preços atualizados :--preco base--'||preco_base_up||'-
-custo_minuto--'||custo_minuto_up||'--custo_espera_minuto--
'||custo_espera_minuto_up||'--custo_cancelamento_pedido--
'||custo_cancelamento_pedido_up );




end if;
*Se a exceção for chamada o procedimento salta para este código
EXCEPTION
   when  time_exception THEN


                    dbms_output.put_line('NAO PODE SER ACTUALIZADO VISTO
QUE FOI ACTUALIZADO Á MENOS DE 6 MESES');
 end;
```

Exercício 5

```
Create OR REPlACE PROCEDURE procDetetarAssociacoes AS
inicio date;
fim date;
matricula_t veiculos_condutores.matricula%type;
matricula_sobre veiculos_condutores.matricula%type;
correction_not_sucesfull EXCEPTION;
dummy boolean;
*Declaro variáveis e seus tipos e o cursor vai apontar para atributos da tabela
veículos_condutores.
cursor cursors is
 select  vc.matricula as m, vc.data_inicio as di, vc.data_fim as df from
veiculos_condutores vc;
 cursor cursors2 is
```

```
 select  vc.matricula from veiculos_condutores vc where   vc.data_inicio
BETWEEN inicio and fim or vc.data_fim BETWEEN inicio  and fim;
```

* cursors2 vai apontar para o atributo matricula da tabela veículos_condutores que estejam dentro das datas para ver se há sobreposiçao.

```
begin
open cursors;



loop
```

*primeiro loop para carregar para variáveis a informação de datas e matricula de veículos condutores

```
fetch cursors into matricula_t, inicio, fim;



open cursors2;
 loop
```

*segundo loop , o fetch para matricula_sobre so vai ocorrer se existir sobreposição pois é assim que o cursor 2 esta defenido

```
  fetch cursors2 into matricula_sobre ;
```

*se houver uma matricula com sobreposição ele actualiza a data para corrigir

```
  update Veiculos_condutores
 SET data_fim=data_inicio where matricula = matricula_sobre;
 exit when cursors2%notfound;
end loop;



  exit when cursors%notfound;
```

```
 close cursors2;
end loop;
*chama a função do ex 2 para verificar se a sobreposição foi corrigida, se não foi levanta exepçao
dummy := funcSobreposicoesVeiculosCondutores();
if  dummy = true then
raise correction_not_sucesfull;
end if;
EXCEPTION
   when  correction_not_sucesfull THEN

                    dbms_output.put_line('nao foi possivel resolver a sobreposicao');

end;
```

Exercício 6

```
create or replace PROCEDURE procGuardarInformacaoSemanal(
            date_in DATE)
AS

   curs1 SYS_REFCURSOR;
```

- Aqui define o curs2 para apontar para os atributos matricula data_inicio e data_fim da tabela veículos_condutores

```
   CURSOR curs2 is   select matricula, data_inicio, data_fim from veiculos_condutores  ;
```

- Aqui define o curs3 para apontar para os atributos matriculae nr_viagens da tabela resumos_veiculos

```
    CURSOR curs3 is   select resumos_veiculos_matricula,
resumos_veiculos_nr_viagens from resumos_veiculos  ;
```

- Aqui defino veriaveis que vou usar, e seus tipos

```
   ite number;
   counter number;

percentagem_v_activos number;
percentagem_v_inactivos number;

   cod_Matricula veiculos.matricula%type;
   cod_Data_Inicio veiculos_condutores.data_inicio%type;
   cod_Data_Fim veiculos_condutores.data_fim%type;
   cod_numero_viagens integer;
   cod_total_distancia integer;
   cod_total_duracao integer;

   dummy_nr_viagens number;
BEGIN
counter := 0;   *  Inicializo o counter a 0

  curs1 := FUNCOBTERINFOSEMANALVEICULOS(date_in);
```

- O curs1 vai ser o cursor retornado pela função do exercico3

```
     select count(*) into ite from veiculos_condutores;
```

- Ponho em ite o numero de linhas da tabela veículos_condutores

```
     OPEN curs2;
     LOOP
      EXIT WHEN counter =ite;
```

- Quando count = numero de linhas da tabela veículos_condutores acabo o loop

```
        FETCH curs2 INTO cod_Matricula, cod_Data_Inicio, cod_Data_Fim;
```

- Ponho nas variáveis cod_matricula, cod_data_inicio, cod_data_fim a informação do curs2

INSERT INTO resumos_veiculos VALUES (sysdate, cod_Data_Inicio, cod_Data_Fim, cod_Matricula, null, null, null);

- Insiro na tabela resumos_veiculos a informação das variáveis inicializadas anteriormente com 3 valores a null porque ainda não tenho essa informação e pode ser que não exista informação para todos os veículos_condutores

counter := counter+1; * incremento o contador

END LOOP;

CLOSE curs2;

counter := 0; * volto a igualar o counter a 0 para me servir para o próximo loop

LOOP

FETCH curs1 INTO cod_Matricula, cod_Data_Inicio, cod_Data_Fim, cod_numero_viagens, cod_total_distancia, cod_total_duracao;

*Ponho nas variáveis a informação do curs1, este cursor tem a informação que tiveram viagens nessa semana e é retornado pela função do ex3

update resumos_veiculos

SET resumos_veiculos_nr_viagens = cod_numero_viagens, resumos_veiculos_soma_km=cod_total_distancia,resumos_veiculos_soma_duracao= cod_total_duracao where resumos_veiculos_data_inicio = cod_Data_Inicio;

- Actualizo a informação da tabela resumos_veiculos dos veículos que tiveram viagens nessa semana

EXIT WHEN curs1%NOTFOUND;

counter := counter+1;

END LOOP;

- Calculo a percentage de veículos que fizeram ou não viagens

percentagem_v_activos := (counter/ite)*100;

percentagem_v_inactivos := 100-percentagem_v_activos;

- Escrevo no ecra

dbms_output.put_line('A percentagem de veiculos que realizaram serviços esta semana é de'||percentagem_v_activos|| 'e dos que nao realizaram é de '|| percentagem_v_inactivos);

counter :=0; * Volto a igualar counter a 0 para me servir para o próximo loop

```
OPEN curs3;
 LOOP
 EXIT WHEN counter =ite;
  FETCH curs3 INTO cod_Matricula, dummy_nr_viagens;
  if dummy_nr_viagens != null then
     dbms_output.put_line('o veiculo de matricula' || cod_Matricula|| ' realizou viagens ');
     else
      dbms_output.put_line('o veiculo de matricula' || cod_Matricula|| ' NAO realizou viagens ');
```

*Nteste loop o curs3 aponta as matriculas e para o numero de viagens da tabela resumos_veiculos (como já foi referido), se o numero de viagens não for null quer dizer que o veiculo realizou viagens nessa semana e escreve a matricula do veiculo e diz que realizou viagens, se não, escreve a matricula do veiculo e que não realizou viagens

```
end if;
counter :=counter+1;
end loop;

END;
```

Pontos a melhorar

- Divisão das tarefas
- Organização do trabalho
- comunicação
- proatividade
- Team work