**SEQUENCE**

Sequence: Quando uma aplicação necessita utilizar valores numéricos sequenciais em uma tabela faz-se o uso de sequence.

[Create sequence nome\_da\_seqüência] [increment by N] [start with N] [maxvalue ] or [minvalue ] [cycle | nocycle] [cache N | nocache]

create sequence idn\_emp increment by 1 start with 100 maxvalue 999 nocycle

O nextval retorna o próximo número da seqüência.

Os comandos a seguir mostram a utilização da pseudocoluna nextval.

insert into emp (n\_emp, nome\_emp) values(idn\_emp.nextval,'MARCELO')

Quando a inserção de um número de empregado já existe, automaticamente a sequente “pega” o próximo valor.

**O uso de uma sequence por mais de uma tabela -** A sequence criada não necessariamente fica atrelada a tabela emp. Para verificar que a sequence pode ser utilizada por mais de uma tabela, realize os seguintes comandos:

alter table dep modify n\_dep number(4); insert into dep (n\_dep, nome\_dep) values (idn\_emp.nextval,'PRODUÇÃO') ; select \* from dep; delete from dep where n\_dep = 107 ( ou o código que foi gerado pela sequence)

**Usando o currval -** O Currval retorna o valor corrente conforme apresentado no comando abaixo.

select IDN\_EMP.currval from dual

Para eliminar uma sequence utilizaremos o comando drop. O comando abaixo mostra um exemplo da utilização do comando.

drop sequence idn\_emp

**Conclusões sobre a sequence -** Vantagens:

* Pode-se utilizar apenas uma sequence para várias tabelas do banco.
* O sequence ajuda na organização dos registros.

Desvantagens

* Caso haja uma falha no sistema será perdida toda a numeração seqüencial.
* Caso você execute o comando DELETE na sua tabela, o número da sequencia que você deletou será perdido, permanecendo um "buraco" na sua coluna.

**TRIGGER**

Gatilhos ou Triggers são sub-programas disparados automática e implicitamente sempre que ocorrer um evento associado a uma tabela, podendo ser:  Insert  Update  Delete  Os gatilhos são disparados antes (before) ou depois (after) de ocorrer um destes eventos

 É muito utilizado para ajudar a manter a consistência dos dados ou para propagar alterações em um determinado dado de uma tabela para outras.

 Um bom exemplo é um gatilho criado para controle de quem alterou a tabela, nesse caso, quando a alteração for efetuada, o gatilho é "disparado" e grava em uma tabela de histórico de alteração, o usuário e data/hora da alteração

CREATE OR REPLACE TRIGGER nome\_trigger (BEFORE / AFTER) DELETE OR INSERT OR UPDATE OF (col1, coln) ON nome\_da\_tabela FOR EACH ROW DECLARE variáveis, constantes, etc BEGIN END;

CREATE OR REPLACE TRIGGER TG\_BI\_EMP BEFORE INSERT ON EMP FOR EACH ROW BEGIN   
 IF :NEW.N\_EMP IS NULL THEN  
 SELECT IDN\_EMP.NEXTVAL   
 INTO :NEW.N\_EMP FROM DUAL;   
 END IF;   
END;

INSERT INTO EMP (NOME\_EMP, N\_DEP) VALUES ('JOHN', 20);

**BLOCOS PL SQL**

• Alterar, inserir excluir e pesquisar dados no BD • Criar variáveis e constantes herdando o tipo de dados e o tamanho de colunas de tabelas • Criar registros para guardar o resultado de campos de tabelas, herdando o tipo de dados e o tamanho de colunas de tabelas • Tratar Erro

• Portabilidade: Qualquer computador que execute o SGBD Oracle pode executar uma aplicação PL/SQL; • Integração com o SGBD: Como as variáveis podem herdar tipos de dados e tamanho de colunas de tabelas, alterações feiras no BD refletirão automaticamente no bloco PL/SQL sem qualquer alteração nele; • Capacidade Procedural: Comandos de repetição, controle de fluxo e tratamento de erros; • Produtividade: Desenvolvimento de procedures, functions e triggers de bancos de dados;

CRIANDO UM FUNÇÃO

create or replace function "MEDIA\_POND" (nota1 in NUMBER, peso1 in NUMBER, nota2 in NUMBER, peso2 in NUMBER)  
return NUMBER  
Is mp NUMBER;  
begin  
mp:=(nota1\*peso1 + nota2\*peso2) / (peso1 + peso2);   
return mp;  
end;  
select media\_pond (10,1,5,3) as mp from dual;

ANALISANDO O CÓDIGO FONTE CREATE OR REPLACE FUNCTION “MEDIA\_POND”

Estamos criando ou substituindo uma função cujo nome é MEDIA\_POND. (nota1 in NUMBER, peso1 in NUMBER, ...) É a declaração de parâmetros de entrada, no momento do uso da função deverão ser fornecidos quatro números. return NUMBER É o valor de retorno da função, ao final do seu processamento ela deve retornar a quem chamou um número. Is Indica que a escrita da função irá começar. mp NUMBER; Declaração de uma variável de escopo local do tipo número para ser usada durante o processo de calculo. mp:=(nota1\*peso1 + nota2\*peso2) / (peso1 + peso2); Armazena temporariamente na variável numérica “mp” o valor calculado.

**Controle de Fluxo no PL/SQL**

Temos no PL/SQL três formas de fluxo: • A sequencial ou natural que executa os comandos de cima para baixo e da esquerda para direita, sendo o “;” o caractere que determina o fim de um comando. • A outra forma é de decisão onde escolhemos fazer ou não determinado passo ou executar um passo se verdadeiro ou outro passo se falso. • E por fim temos a repetição que permite voltar no fluxo de execução fazendo que determinados passos sejam repetidos conforme uma condição de saída

Dado um número a função retorna em varchar2 se o número é PAR ou ÍMPAR:   
create or replace function "TESTE" (num in number)   
return VARCHAR2 is   
begin   
 if num mod 2 = 0 then   
 return 'PAR';   
 else   
 return 'IMPAR';   
 end if;   
end;   
select TESTE (22) as "PAR OU IMPAR" FROM DUAL;

**CURSORES**

• Cursor no PL/SQL é uma forma de tratar um resultado de consulta de forma personalizada. Podemos trabalhar registro a registro aplicando uma lógica própria.

Desejamos ler registro a registro a tabela FUN e mostrar os funcionários e seus salários, informando ainda se o salário do mesmo é acima ou abaixo da média:

DECLARE   
 CURSOR <NOME-CURSOR> IS <CONSULTA>;   
 <VARIAVEL-REGISTRO> <NOME-CURSOR>%ROWTYPE;   
BEGIN   
 OPEN <NOME-CURSOR>;   
 LOOP   
 FETCH <NOME-CURSOR> INTO <VARIAVEL-REGISTRO>;   
 EXIT WHEN <NOME-CURSOR>%NOTFOUND;   
 /\* LÓGICA PERSONALIZADA   
 – INSERIR, ALTERAR EM OUTRA TABELA;   
 - FAZER ALGUM CALCULO, EXPRESSÕES; \*/   
 END LOOP;   
 CLOSE <NOME-CURSOR>;   
END;

Dada a seguinte tabela com os seguintes registros: CREATE TABLE Fun( idFun INTEGER, nomeFun VARCHAR2(100), salario DECIMAL(8,2), CONSTRAINT pk\_fun PRIMARY KEY (idFun) );

INSERT INTO Fun VALUES (1, 'Maria', 2000.50); INSERT INTO Fun VALUES (2, 'Carolina', 3000.25); INSERT INTO Fun VALUES (3, 'Sergio', 2500.00);

CRIANDO UM CURSOR COM UMA PROCEDURE

create or replace PROCEDURE LISTAFUN is CURSOR c\_fun is SELECT \* FROM Fun;

r\_fun fun%ROWTYPE;  
media NUMBER;

BEGIN   
SELECT AVG(salario) INTO media FROM Fun;   
DBMS\_OUTPUT.PUT\_LINE('Média=' || to\_char(media));   
DBMS\_OUTPUT.PUT\_LINE('Lista de Funcionários');   
OPEN c\_fun;   
LOOP   
 FETCH c\_fun INTO r\_fun;   
 EXIT WHEN c\_fun%NOTFOUND;   
 if r\_fun.salario>=media THEN   
 DBMS\_OUTPUT.PUT\_LINE(r\_fun.nomeFun || ' - ' || to\_char(r\_fun.salario) || ' - Salário maior ou igual a média');   
 ELSE   
 DBMS\_OUTPUT.PUT\_LINE(r\_fun.nomeFun || ' - ' || to\_char(r\_fun.salario) || ' - Salário menor que a média');   
 END IF;   
 END LOOP;   
 CLOSE c\_fun;   
END LISTAFUN;

RODANDO A PROCEDURE COM O CURSOR

BEGIN LISTAFUN; END;

**SELECT INTO**

O resultado de um comando SELECT que retorna várias colunas (mas apenas uma linha) pode ser atribuído a uma variável registro, a uma variável tipo-linha, ou a uma lista de variáveis escalares. É feito através de SELECT INTO destino expressões\_de\_seleção FROM ...;

onde destino pode ser uma variável registro, uma variável linha, ou uma lista separada por vírgulas de variáveis simples e campos de registro/linha.

A expressões\_de\_seleção e o restante do comando são os mesmos que no SQL comum.

Se for utilizado como destino uma linha ou uma lista de variáveis, os valores selecionados devem corresponder exatamente à estrutura do destino, senão ocorre um erro em tempo de execução.

Quando o destino é uma variável registro, esta se configura automaticamente para o tipo linha das colunas do resultado da consulta.

Estrutura de um Bloco PL/SQL

• Área de declaração, • Área de comandos • Área de exceções

Estrutura de um Bloco PL/SQL

DECLARE   
 declarações   
BEGIN   
 estruturas executáveis, comandos e outros blocos PL/SQL   
 BEGIN   
 EXCEPTION   
 tratamento de exceções (pode conter outros blocos)   
 END ;   
END;

EXEMPLO

DECLARE   
 NOME EMP.NOME\_EMP%TYPE;   
 NUMEROEMPREGADO NUMBER;   
BEGIN  
 NUMEROEMPREGADO := &NUMEROEMPREGADO;   
 SELECT NOME\_EMP  
 INTO NOME  
 FROM EMP  
 WHERE N\_EMP = NUMEROEMPREGADO;  
 IF NOME IS NOT NULL THEN  
 DBMS\_OUTPUT.PUT\_LINE ('NOME DO EMPREGADO ' || NOME);  
 END IF;  
 EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('NÃO EXISTE EMPREGADO COM O NUMERO '|| NUMEROEMPREGADO||'. POR FAVOR VERIFIQUE.');  
 WHEN TOO\_MANY\_ROWS THEN DBMS\_OUTPUT.PUT\_LINE('MAIS DE UM EMPREGADO COM O NUMERO '||NUMEROEMPREGADO||' FOI ENCONTRADO, POR FAVOR VERIFIQUE.');  
 END;

DECLARE   
 DEPARTAMENTO EMP.N\_DEP%TYPE;  
 NUMEROEMPREGADO NUMBER;  
 REG\_EMP EMP%RowType  
 BEGIN  
 NUMEROEMPREGADO := &NUMEROEMPREGADO;  
 SELECT \*  
 INTO REG\_EMP  
 FROM EMP  
 WHERE REG\_EMP.N\_EMP = NUMEROEMPREGADO;  
 IF REG\_EMP.COM IS NOT NULL THEN  
 DBMS\_OUTPUT.PUT\_LINE (‘COMISSÃO DO EMPREGADO ' || REG\_EMP.COM);  
 END IF;  
 EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('NÃO EXISTE EMPREGADO COM O NUMERO '|| NUMEROEMPREGADO||'. POR FAVOR VERIFIQUE.');   
 WHEN TOO\_MANY\_ROWS THEN DBMS\_OUTPUT.PUT\_LINE('MAIS DE UM EMPREGADO COM O NUMERO '||NUMEROEMPREGADO||' FOI ENCONTRADO, POR FAVOR VERIFIQUE.');  
 END;

DECLARAÇÕES: Pode-se declarar os seguintes tipos: – Variáveis – Constantes – Cursores – Estruturas – Tabelas

CONSTANTES : • Para declarar-se constantes, utiliza-se a palavra reservada CONSTANT • Deve aparecer antes do tipo de dado de um valor • Deve ser atribuído um valor para a constante ou a palavra reservada DEFAULT seguida de um valor

Exemplos:   
NR\_LIDOS CONSTANT NUMBER(4) :=15;   
NR\_TENTATIVA NUMBER(4) := 0;   
NR\_LIDOS2 CONSTANT NUMBER(4) DEFAULT 15  
NR\_TENTATIVA2 NUMBER(4) DEFAULT 0; CD\_CLIENTE   
NUMBER(4) NOT NULL :=1;

Herança de Tipos de Dados e Tamanho

• Para herdar o tipo de uma coluna de uma tabela:   
Nome\_da\_variavel Nome\_da\_tabela.Nome\_da\_Coluna%Type;   
• Para herdar o tipo do registro (linha inteira):  
Nome\_da\_variavel Nome\_da\_tabela%RowType;   
• Para herdar o tipo de uma variável previamente declarada:   
Nome\_da\_variavel Nome\_da\_variavel%Type;

EXEMPLO   
DECLARE   
REG\_ITEM ITEM\_NOTA\_FISCAL%ROWTYPE;   
AUX\_CD\_PRODUTO CONSTANT PRODUTO.CD\_PRODUTO%TYPE := 1;   
VL\_TOTAL PRODUTO.VL\_CUSTO\_MEDIO%TYPE;

“Sistemas de processamento de transação são sistemas com grandes bancos de dados e centenas de usuários executando transações concorrentes no banco de dados” (ELMASRI e NAVATHE, 2005, p. 397)   
λ Uma transação é uma sequência de operações tratadas como um bloco único e indivisível. A execução parcial não é permitida.   
 λ O final da transação é indicado pelos comandos:   
λ A transação envolve as seguintes operações no banco de dados:  
 λ Inserção λ Alteração λ Exclusão   
λ Estas operações são executadas utilizando-se a linguagem SQL (Structure Query Language – Linguagem Estruturada de Consulta)   
λ Em linguagem SQL, as operações envolvidas na transação são, respectivamente: Insert, Update e Delete.   
λ Os problemas mais comuns em transações são:   
λ Atualização perdida: Ocorre quando duas transações são emitidas quase que simultaneamente. A segunda transação utiliza os dados antes que a primeira tenha efetivado as mudanças. Quando a segunda atualiza o Banco de Dados, as modificações da primeira transação são perdidas.   
λ Atualização Temporária: Ocorre quando há um erro em uma transação e uma segunda acessa os dados antes que os valores originais sejam restaurados, usando assim dados incorretos.   
• Os problemas mais comuns em transações são: – Sumário Incorreto: Ocorre quando uma transação usa a função de sumarização enquanto uma outra ainda atualiza os dados – Leitura sem repetição: Ocorre quando uma transação deve ler um dado duas vezes e entre as leituras uma outra transação efetua uma modificação nele. 6 28/09/2016   
• Algumas possibilidade para ocorrência de falhas durante uma transação: – Falha do computador (hardware, software ou rede) – Erro de transação ou sistema: Erros de lógica de programação, valores de parâmetros, etc – Erros locais ou condições de exceções detectadas pela transação – Falha de disco (de leitura ou gravação) – Problemas físicos e catástrofes: falta de energia, fogo, furto, etc   
• São possíveis estados de uma transação: – begin\_transaction: Marca o início de uma transação, passando ao estado ativo – read ou write: São leituras ou gravações feitas enquanto a transação se encontra no estado ativo. Ao encerrar-se, passa ao estado de efetivação parcial – commit\_transaction: Efetua no banco de dados as modificações feitas e não permite que sejam desfeitas, passando ao estado de efetivação – rollback: Operação contrária do COMMIT, retornando o banco de dados ao estado anterior ao início da transação, passando a transação ao estado de falha – end\_transaction: Marca o fim da transação, passando ao estado encerrado   
• Propriedades das transações – ACID – Atomicidade: Uma transação deve ser executada por completo ou não ser executada. Quando há uma falha, as técnicas de recuperação devem desfazer totalmente as modificações feitas – Consistência: Um Banco de Dados é consistente quando satisfaz a restrições específicas, assim, deve continuar sendo consistente ao término de uma transação, cuja responsabilidade é do programador ou do SGBD – Isolamento: Um transação não pode sofrer interferência de transações concorrentes – Durabilidade: As atualizações feitas por uma transação devem persistir no Banco de Dados, independente da ocorrência de falhas.   
 • Para controlar a ordem de execução das transações é criado um plano de execução, que determina a seqüência das operações e das transações   
• O plano de execução pode basear-se na restaurabilidade ou na serialidade   
• A restaurabilidade garante que nenhuma transação T seja efetivada se existirem transações que ainda não efetivaram os dados lidos pela transação T 10   
• Na serialidade, as operações das transações são executadas consecutivamente, sem intercalações com operações de outras transações mas, desperdiça tempo de processamento de CPU   
• Conflito – Duas operações de um plano estão em conflito se pertencem a transações diferentes, tentem acessar um mesmo item do banco de dados e, ao menos uma delas é um operação de gravação   
• Para controlar transações concorrentes utiliza-se técnicas de bloqueio • O bloqueio (lock) indica o que pode ser aplicado em relação às operações que são executadas pelas transações. Deve haver um bloqueio para cada dado no banco de dados.  
 • Tipos de bloqueio – Bloqueios compartilhados – Bloqueios exclusivos 12 28/09/2016  
 • Operações de bloqueio – Read\_lock(X) – “read\_locked” – Write\_lock(X) – “write\_locked” – Unlock(X) – “unlocked”   
• Elmasri e Navathe (2205) definem regras para utilização de bloqueio compartilhado/exclusivo: – Antes de ler um item deve garantir a operação read\_lock ou write\_lock dele – Antes de escrever em um item, deve garantir a operação write\_lock dele – Após todas as operações de leitura e escrita deve garantir a operação de unlock dele   
• Elmasri e Navathe (2205) definem regras para utilização de bloqueio compartilhado/esclusivo: – Se uma operação já tem um bloqueio de leitura ou de escrita não fará a operação de read\_lock dele – Se uma operação já tem um bloqueio de leitura ou de escrita não fará a operação de write\_lock dele – Se uma operação NÃO tem um bloqueio de leitura ou de escrita não fará a operação de unlock dele   
• Deadlock – É a espera de uma transação por um item que está bloqueado por uma outra transação – Detecção do deadlock   
• Indicada para transações que tem poucos acessos aos mesmos itens simultaneamente   
• Quando o deadlock é detectado, algumas transações que o causam devem ser abortadas   
• O uso de timeouts pode ser utilizado para prevenir os deadlocks

Exercícios Trigger e Sequence

create sequence idn\_emp increment by 1 start with 100 maxvalue 999 nocycl  
select \* from user\_sequences  
select \* from user\_tables

Select \* from emp  
insert into emp (codigo, nome) values(idn\_emp.nextval,'MARCELO');  
insert into emp (codigo, nome) values(idn\_emp.nextval,'ADRIANA');  
insert into emp (codigo, nome) values(idn\_emp.nextval,'ALEXANDRE');  
insert into emp (codigo, nome) values(idn\_emp.nextval,'ANA');

alter table dep modify n\_dep number(4)  
insert into dep (n\_dep, nome\_dep) values (idn\_emp.nextval,'PRODUÇÃO')  
select \* from dep  
delete from dep where n\_dep = 106

select IDN\_EMP.currval from dual

create sequence sqc\_restaurantenormal incremente by 1 start with 100 maxvalue 999 no cycle

create sequence sqc\_restaurantepar incremente by 2 start with 100 maxvalue 999 no cycle

create sequence sqc\_restauranteimpar incremente by 3 start with 101 maxvalue 999 no cycle

select \* from user\_sequences

CREATE SEQUENCE SQC\_EMP INCREMENT BY 1 START WITH 100 MAXVALUE 999 NO CYCLE  
CREATE OR REPLACE TRIGGER TG\_BI\_EMP  
BEFORE INSERT ON EMP FOR EACH ROW

BEGIN  
 IF :NEW.CODIGO IS NULL THEN  
 SELECT SQC\_EMP.NEXTVAL  
 INTO :NEW.CODIGO FROM DUAL;  
 END IF;  
END;  
INSERT INTO EMP (CODIGO, NOME) VALUES (null, 'JHON');  
SELECT \* FROM EMP

create sequence sqc\_dep increment by 1 start with 213 maxvalue 999 nocycle

CREATE OR REPLACE TRIGGER TG\_BI\_DEP   
BEFORE INSERT ON DEP FOR EACH ROW

BEGIN  
 IF :NEW.N\_DEP IS NULL THEN  
 SELECT SQC\_DEP.NEXTVAL  
 INTO :NEW.N\_DEP FROM DUAL;  
 END IF;  
END;

INSERT INTO DEP (N\_DEP, NOME\_DEP) VALUES (NULL, 'NUPSI');  
SELECT \* FROM DEP

create sequence sqc\_prof increment by 1 start with 666 maxvalue 999 nocycle

CREATE OR REPLACE TRIGGER TG\_BI\_PROF  
BEFORE INSERT ON PROFISSIONAL FOR EACH ROW  
BEGIN  
 IF :NEW.N\_CONTROLE IS NULL THEN  
 SELECT IDN\_EMP.NEXTVAL  
 INTO :NEW.N\_CONTROLE FROM DUAL;  
 END IF;  
END;  
INSERT INTO PROFISSIONAL VALUES (null,4,'AMARINDO','20/03/97','Chefe','Rua das Moitas',178,'Franca');  
SELECT \* FROM PROFISSIONAL

CREATE SEQUENCE SQC\_VACINA INCREMENT BY 2 START WITH 654 MAXVALUE 6545 NOCYCLE  
CREATE OR REPLACE TRIGGER TG\_VACINA   
BEFORE INSERT ON VACINA FOR EACH ROW

BEGIN  
 IF :NEW.ID IS NULL THEN  
 SELECT SQC\_VACINA.NEXTVAL  
 INTO :NEW.ID FROM DUAL;  
 END IF;  
END;

INSERT INTO VACINA VALUES (NULL,'TESTE','24/08/2016','25/08/2016')

SELECT \* FROM VACINA

CREATE SEQUENCE SQC\_CONTA INCREMENT BY 12 START WITH 0 MAXVALUE 1000 NOCYCLE

CREATE OR REPLACE TRIGGER TG\_CONTA BEFORE INSERT ON CONTA FOR EACH ROW

BEGIN  
 IF :NEW.NUMCONTA IS NULL THEN  
 SELECT SQC\_VACINA.NEXTVAL  
 INTO :NEW.NUMCONTA FROM DUAL;  
 END IF;  
END;

INSERT INTO CONTA VALUES (null,'teste','C','S')  
select \* from conta

Exercícios Função, Repetição e Seleção

Crie uma função que dado o número de minutos retorne o número de horas correspondente.

create or replace function "horas"(minutos in NUMBER)  
return NUMBER  
Is hora NUMBER;

begin  
 hora:=minutos/60;  
 return hora;  
end;  
select from horas(120) as HORAS from dual;

Crie uma função que dado um salário retorne o número de salários mínimos que a pessoa ganha.

create or replace function "quant\_salarios"(salario in NUMBER)  
return NUMBER  
is res NUMBER;

begin  
 res:=salario/788;  
 return res;  
end;  
select from quant\_salarios(788) as SALARIO from dual;

Implemente uma função que retorne o fatorial de um número inteiro.

create or replace function "calc\_fatorial"(fat in NUMBER)  
return NUMBER  
is fatorial NUMBER;  
n NUMBER;

begin  
fatorial:=1;  
n:=fat;  
 loop  
 fatorial:= fatorial\*n;  
 exit when n=1;  
 n:=n-1;  
 end loop;   
 return fatorial;  
 end;  
select from calc\_fatorial(4) as FATORIAL from dual;

Implemente uma função que retorne a média entre os valores inteiros entre o número 1 até o valor passado como inteiro;

create or replace function "MEDIA\_INT" (numero in NUMBER)  
return NUMBER   
is n NUMBER;  
soma NUMBER;  
res NUMBER;  
aux NUMBER;

begin  
n:=numero;  
aux:=n;  
soma:=0;  
res:=0;  
 loop  
 soma:= soma+n;  
 exit when n=1;  
 n:=n-1;  
 end loop;   
 res:= soma/aux;  
return res;  
end;   
select from media\_int(5) as MEDIA from dual;

Suponha que a população do Brasil seja menor que a população da França e que o Brasil tenha taxa de crescimento anual de 4% ao ano e a França de 2%. Dado um ano implemente uma função que calcule o ano em que a população do Brasil será maior que a da França. Os parâmetros da função são a população do Brasil, população da França e o ano base para iniciar os cálculos.

create or replace function "POPULACAO"(br\_pop in NUMBER, fr\_pop in NUMBER, ano\_base in NUMBER)  
return NUMBER  
is  
pop\_br NUMBER;  
pop\_fr NUMBER;  
ano NUMBER;  
aux NUMBER;

begin  
 pop\_br:=br\_pop;  
 pop\_fr:=fr\_pop;  
 ano:=ano\_base;  
 aux:=ano\_base;   
 loop  
 pop\_br:=pop\_br\*0.04;  
 pop\_fr:=pop\_fr\*0.02;  
 exit when pop\_br>pop\_fr;  
 ano:=ano+1;  
 end loop;  
 return ano;  
end;  
select from populacao(100,400,2012) as POPULACAO\_MAIOR from dual;

Exercícios Select Into

• Crie um bloco PL/SQL para verificar se existe um departamento conforme o valor de entrada de uma variável

declare NUMDEP NUMBER;

DEPNUM DEP%ROWTYPE;

begin

NUMDEP := &NUMDEP;

select \* into DEPNUM

from DEP

where NUMDEP = DEP.N\_DEP;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('Departamento não existente');

WHEN TOO\_MANY\_ROWS THEN DBMS\_OUTPUT.PUT\_LINE('Departamento encontrado');

end;

• Crie um bloco PL/SQL para imprimir um os dados de um produto a partir de seu código

declare CPRODUTO PRODUTO%ROWTYPE;

PRODUTOCOD produto.codcategoria%TYPE;

PRODUTON PRODUTO.NOMPRODUTO%TYPE;

begin

PRODUTOCOD := &PRODUTOCOD;

select \*

into CPRODUTO

from PRODUTO

where PRODUTO.CODCATEGORIA = PRODUTOCOD;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN DBMS\_OUTPUT.PUT\_LINE('Não encontrado o nome do produto');

WHEN TOO\_MANY\_ROWS THEN DBMS\_OUTPUT.PUT\_LINE('Nome produto '||PRODUTON||'');  
END;