2ª EDIÇÃO

CURSO DE FÉRIAS

SQL E PL/SQL BÁSICO

MATERA systems

Sumário

1. Introdução a Banco de Dados

1.1 O que é um banco de dados?

Banco de dados ou base de dados são coleções organizadas de informações a fim de dar suporte organizacional e computacional para qualquer grupo de pessoas, empresas ou software.

1.2 Porque precisamos de um banco de dados?

Quando estamos utilizando qualquer software toda informação que é vista trabalha de modo variável e é mantida na memória RAM do dispositivo, que possui tamanho limitado e alto custo, além disto a memória perde todos os dados ao descarregar a energia contida e então não existe persistência, chegando ao ponto que surge a necessidade de armazenar em discos.

1.3 O que pode ser um banco de dados?

Tudo que pode armazenar informações pode ser considerado um banco de dados, um arquivo de texto(txt), arquivo do excel (csv) ou qualquer local para armazenar informações.

	A	B	C
1	First Name	Last Name	Email Address
2	Jane	Doe	jane@gmail.com
3	John	Doe	john@gmail.com
4	Chris	Perkins	cperk@gmail.com
5	Eva	Davis	eva@gmail.com
6	Mitchell	Tarver	mitch@gmail.com
7	Nathan	Woodward	nathanwoodward@gmail.com

1.4 O que é um SGBD?

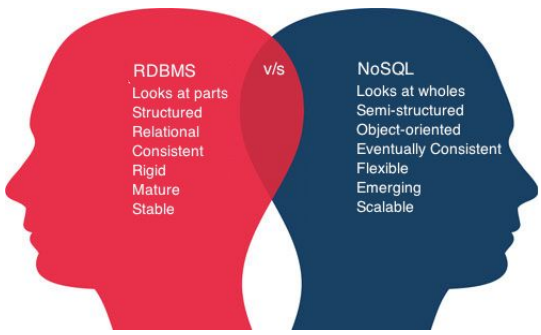
Um Sistema Gerenciador de Banco de Dados (SGBD) do inglês Data Base Management System (DBMS) é uma estrutura de software contendo uma estrutura e gerenciamento de informações de modo organizado.

1.5 Tipos de bancos de dados

Os dois principais bancos de dados são subdivididos entre relacionais e não relacionais.

- Os bancos de Dados relacionais armazenam informações em formatos de tabelas com linhas e colunas aplicando regras para garantir a integridade e normalização (utilizado no decorrer do curso).

- Os Bancos de dados não relacionais (NoSQL) possuem um foco diferenciado, escalabilidade e preço, e trabalham de modo diferente de tratar informações, MariaDb e MongoDB são dois grandes exemplos.



1.6 O que é SQL (Structured Query Language)?

Structured Query Language, ou Linguagem de Consulta Estruturada ou SQL, é a linguagem de pesquisa declarativa padrão para banco de dados relacional.

SQL é a estrutura base para todos bancos de dados relacionais em que através de uma interface conectada no banco podemos usar comandos do tipo: SELECT, INSERT, UPDATE e DELETE.

2. Comandos DDL

São comandos utilizados para definirem as estruturas de dados, como as tabelas que compõem um banco de dados, os cinco tipos básicos de instruções DDL são:

CREATE: cria uma estrutura de banco de dados. Por exemplo, CREATE TABLE é usada para criar uma tabela; outro exemplo é CREATE USER, usada para criar um usuário do banco de dados.

ALTER: modifica uma estrutura de banco de dados. Por exemplo, ALTER TABLE é usada para modificar uma tabela.

DROP: remove uma estrutura de banco de dados. Por exemplo, DROP TABLE é usada para remover uma tabela.

RENAME: muda o nome de uma tabela.

TRUNCATE: exclui todas as linhas de uma tabela.

2.1 Tipos de Dados

Cada valor manipulado pelo Oracle Database possui um tipo de dados. Quando se cria uma tabela é necessário informar qual o tipo de dados de suas colunas, assim como, quando se cria uma procedure ou function são especificados os tipos de dados de seus parâmetros, na tabela 1 são apresentados os tipos de dados aceitos pelo Oracle.

Tipo de Dados	Descrição
VARCHAR2(compriment o_máximo)	Carácter de tamanho variável, podendo atingir o tamanho máximo de até 32767 bytes.
CHAR [comprimento_máximo]	Carácter fixo com o tamanho máximo de 32767 bytes. Se o tamanho não for especificado, o default é 1.
NUMBER [precisão, escala]	Tipo numérico fixo e de ponto flutuante.
BINARY_INTEGER	É o tipo básico para inteiros entre -2.147.483.647 e

	2.147.483.647. Utiliza funções da biblioteca interna para executar a aritmética.
LONG	Carácter de tamanho variável podendo Ter até 32760 bytes.
DATE	Tipo para acomodar data e hora
BOOLEAN	Armazena três valores lógicos possíveis, TRUE, FALSE e NULL.
LONG RAW	Tipo binário variável de até 32760 bytes.
ROWID	Utilizado para armazenar endereços físicos das linhas
UROWID	Utilizado para armazenar endereços físicos e lógicos

2.2 Criação de Tabelas

Os dados são armazenados em estruturas chamadas tabelas, abaixo é apresentada a composição do comando *create table*. Na criação de uma tabela, é necessário informar qual o tipo de tabela será criada, caso não seja informado o tipo, por padrão é criada uma tabela em que os dados armazenados se mantém armazenados (caso sejam efetivadas as alterações) após o término da transação, em seguida deve-se informar o esquema que a tabela pertencerá, e os campos da tabela.

Exemplo:

```
CREATE TABLE time(
  id_time NUMBER NOT NULL,
  nome VARCHAR2(400) NOT NULL
) TABLESPACE treinamento;
```

2.3 Constraints

Constraints Oracle são fundamentais para a escalabilidade, flexibilidade e integridade dos dados armazenados em um banco de dados. Elas aplicam regras específicas para os dados, garantem que os dados estejam em conformidade com os requisitos definidos. Existem alguns tipos de constraints no Oracle, a seguir elas são apresentadas:

Not NULL: Poderá designar qualquer coluna como NOT NULL (algo muito comum em colunas de IDs), e o que isto na prática quer dizer, é que se qualquer operação SQL deixar um valor NULL nesta coluna, então a base de dados Oracle retornará um erro.

Check: É uma constraint mais genérica, tratando-se de uma expressão booleana que avalia se algo é TRUE ou FALSE. Se a constraint check é avaliada e é obtido FALSE, então o statement SQL retornará um erro.

Unique: Este tipo de constraint previne que uma coluna ou um conjunto de colunas tenham valores não únicos (também muito comum em IDs), evitando assim, que sejam introduzidos valores repetidos, ou até modificados para valores repetidos.

Foreign Key: A foreign key (em português chave estrangeira) é definida para uma tabela (conhecida como filha) que tem um relacionamento com outra tabela (conhecida como pai). O valor guardado na foreign key deverá ser o mesmo presente na primary key respectiva.

Primary key: Cada tabela pode ter, no máximo, uma constraint de primary key (em português chave primária). A primary key pode ter mais que uma coluna da tabela. A constraint de primary key força que cada chave primária só pode ter um valor único, impondo em simultâneo a constraint unique e NOT NULL. Uma primary key vai criar um índice único, caso ainda não exista para a coluna em causa.

2.4 Comentários

Ao criar uma tabela, é possível definir comentários para a tabela e colunas, isso auxilia no entendimento do objetivo da tabela e colunas.

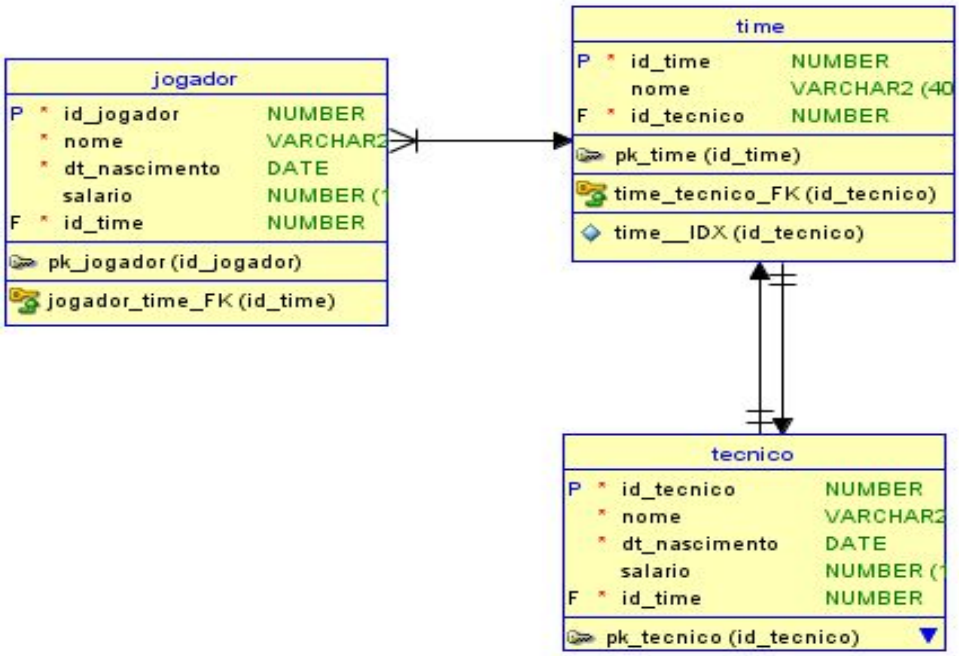
Exemplos:

```
COMMENT ON TABLE time IS '[CADASTRO] Tabela para armazenamento de times';
COMMENT ON COLUMN time.id_time IS 'Código identificador do time';
COMMENT ON COLUMN time.nome IS 'Nome do time';
```

2.5 Exercícios

1. Criar as tabelas time, técnico e jogador;
2. Definir constraints para as tabelas;

3. Criar comentários para as tabelas e as colunas;



3. Comandos DML I

Data Manipulation Language (DML) são utilizados para o gerenciamento de dados dentro de objetos do banco de dados.

3.1 Transações

O COMMIT é um comando utilizado no controle transacional, faz com que o dado inserido, alterado ou removido seja realmente persistido(salvo) no banco de dados.

O ROLLBACK é um comando utilizado também no controle transacional, ele desfaz as alterações de dados realizadas desde o início da Rotina, Checkpoint(savepoint) ou último COMMIT.

Exemplos:

```
commit;
rollback;
```

3.2 Sequence

Durante a instrução INSERT precisamos definir a chave primaria de uma tabela, como este tipo de objeto restringe-se a um identificador único de uma determinada tabela se torna verboso a cada insert escolher um número único não utilizado, então utilizamos de um objeto capaz de fazer isto no Oracle a SEQUENCE.

Exemplo de criação:

```
create sequence seq_jogador
minvalue 1 maxvalue 999999999999999999999999999
start with 1 increment by 1
no cache;
```

Este objeto possui dois métodos pré-definidos:

1. nextval: retorna o próximo valor da sequence.
2. Curval: retorna o último número gerado pela sequence.

Exemplos de utilização:

```
SELECT seq_jogador.NEXTVAL FROM DUAL;
SELECT seq_jogador.CURVAL FROM DUAL;
```

3.3 Consulta de Registros

A instrução SELECT é utilizada para recuperar os dados do banco de dados.

```
SELECT * FROM time WHERE nome = 'BARCELONA';
```

3.4 Inclusão de Registros

A instrução INSERT é utilizada para inserir dados no banco de dados.

```
INSERT INTO time (id_time, nome) VALUES (1,'BARCELONA');
```

3.5 Atualização de Registros

A instrução UPDATE é utilizada para alterar dados já existentes no banco de dados.

```
INSERT INTO time (id_time, nome) VALUES (1,'BARCELONA');
```

3.6 Exclusão de Registros

A instrução DELETE é utilizada para remover dados no banco de dados.

```
DELETE time WHERE id_time = 1;
```

Também temos a instrução TRUNCATE TABLE que também é utilizada para apagar os registros da tabela porém com algumas diferenças do DELETE, primeiro o “truncate “ é um comando DDL então ele possui “commit implícito” e segundo ele não permite escolher os registros que serão apagados, apagando todos registros da tabela sempre.

```
TRUNCATE TABLE nome_da_tabela;
```

3.7 Ordenações

Utilizamos as ordenações para ordenar os resultados de uma consulta. Podemos ordenar por ordem crescente(asc) ou decrescente(desc). No exemplo, vamos selecionar os jogadores ordenador pelo nome.

Exemplo:

```
SELECT nome FROM jogador ORDER BY nome ASC;
SELECT nome FROM jogador ORDER BY nome DESC;
```

3.8 Apelidando no Oracle

Muitas vezes o nome das tabelas e/ou colunas não são totalmente auto-explicativa, tornando a leitura de uma consulta SQL trabalhosa e cansativa, então utiliza se os “apelidos” para facilitar nestes

pontos.

- Para tabelas, basta apenas colocar o nome do “apelido” em seguida da tabela.
- Para colunas, basta apenas colocar o nome do “apelido” em seguida da tabela, porém é comum em outros bancos de dados também utilizar “AS”.

Exemplo:

```
SELECT jog.id_jogador,
       jog.nome
FROM jogador jog;
--
SELECT jog.id_jogador ID_DO_JOGADOR,
       jog.nome NOME_DO_JOGADOR
FROM jogador jog;
--
SELECT jog.id_jogador AS ID_DO_JOGADOR,
       jog.nome AS NOME_DO_JOGADOR
FROM jogador jog;
```

3.9 Exercícios

1. Inserir 2 times.
2. Inserir 2 técnicos.
3. Inserir 11 jogadores em um time.
4. Alterar todos jogadores de um time para o novo time.
5. Aumentar em 10% o salário de todos jogadores do novo time.
6. Aumentar o salário de todos técnicos em 20%.
7. Listar todos jogadores de um determinado time.
8. Listar todos times.
9. Listar técnicos com mais de 40 anos.
10. Inserir os jogadores existentes para o outro time (select insert com sequence).
11. Inserir um time novo.
12. Inserir 3 jogadores extras no time novo.
13. Alterar o salário de 3 jogadores para valores acima de R$ 100.000,00.
14. Remover jogadores do novo time com salários superiores R$ 100.000,00.
15. Remover times que estejam sem jogadores e técnicos.

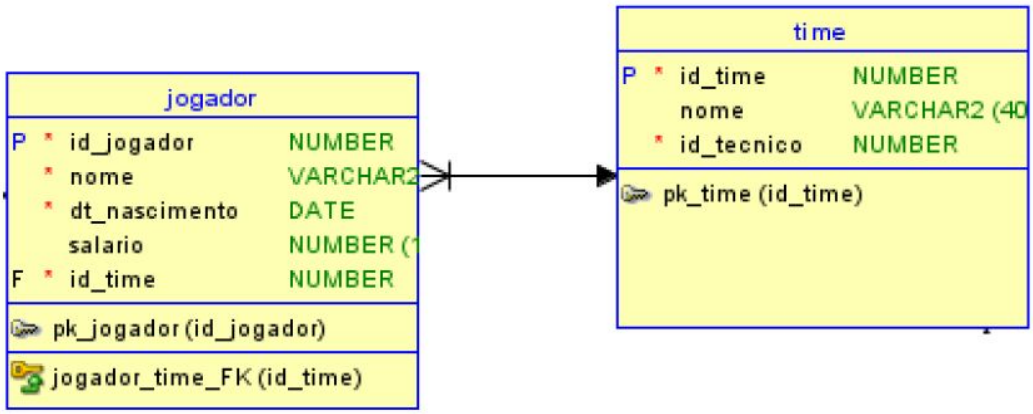
16. Selecione os Times em ordem crescente.
17. Selecione os nomes de jogadores e seus respectivos nomes dos times ordenado(asc) pela data de nascimento dos jogadores.

4. Comandos DML II

4.1 Junções de Dados

Quando queremos unir resultados de mais de uma tabela.

Considerando as figuras abaixo:



Time	
id_time	nome
1	SANTOS
2	CORINTHIANS
3	ATLÉTICO
4	SPORT
5	CORITIBA
6	ASSIS

Jogador		
id_jogador	nome	id_time
1	Ricardo Oliveira	1
2	Vagner Love	2
3	Jadson	2
4	LUCAS PRATO	3
5	Andre	4
6	Henrique	5
7	Joao da silva	

4.1.1 Cross Join

Quando queremos juntar duas ou mais tabelas por cruzamento, isso significa, todas as linhas da tabela Jogador que estão relacionadas com a tabela Time ou o inverso, Time com jogador

Exemplo:

```
SELECT j.nome, t.nome FROM jogador j, time t;
```

retornaria:

j.nome	t.nome
Ricardo Oliveira	SANTOS
Ricardo Oliveira	CORINTHIANS

Ricardo Oliveira	ATLÉTICO
Ricardo Oliveira	SPORT
Ricardo Oliveira	CORITIBA
Vagner Love	SANTOS
Vagner Love	CORINTHIANS
Vagner Love	ATLÉTICO
Vagner Love	SPORT
Vagner Love	CORITIBA
Jadson	SANTOS
Jadson	CORINTHIANS
Jadson	ATLÉTICO
Jadson	SPORT
Jadson	CORITIBA
LUCAS PRATTO	SANTOS
LUCAS PRATTO	CORINTHIANS
LUCAS PRATTO	ATLÉTICO
LUCAS PRATTO	SPORT
LUCAS PRATTO	CORITIBA
Andre	SANTOS
Andre	CORINTHIANS
Andre	ATLÉTICO
Andre	SPORT

Andre	CORITIBA
Henrique	SANTOS
Henrique	CORINTHIANS
Henrique	ATLÉTICO
Henrique	SPORT
Henrique	CORITIBA

4.1.2 Inner Join

Quando queremos juntar duas ou mais tabelas, que internamente, tenha valores correspondentes. No exemplo Jogador X Time temos o id_time no lado de Jogador e id_time no lado de Time.

Exemplo:

```
SELECT j.nome, t.nome
FROM jogador j, time t
WHERE t.id_time = j.id_time;
```

retornaria

j.nome	t.nome
Ricardo Oliveira	SANTOS
Vagner Love	CORINTHIANS
Jadson	CORINTHIANS
LUCAS PRATTO	ATLÉTICO
Andre	SPORT

Henrique	CORITIBA

4.1.3 Left Outer Join

Serve para selecionar todos os itens de uma tabela A com uma tabela B mesmo que A não esteja relacionado com B. No exemplo, vamos selecionar todos os jogadores, mesmo os que não possuem time relacionado.

Exemplo:

```
SELECT j.nome, t.nome
FROM jogador j, time t
WHERE j.id_time = t.id_time(+);
```

retornaria

j.nome	t.nome
Ricardo Oliveira	SANTOS
Vagner Love	CORINTHIANS
Jadson	CORINTHIANS
LUCAS PRATTO	ATLÉTICO
Andre	SPORT
Henrique	CORITIBA
Joao da silva	null

4.1.4 Right Outer Join

Funciona como o left outer join, mas ao contrário. No exemplo, vamos selecionar todos os Times, mesmo os que não possuem Jogador relacionado.

Exemplo:

```
SELECT j.nome, t.nome
FROM jogador j, time as t
WHERE j.id_time(+) = t.id_time;
```

retornaria

j.nome	t.nome
Ricardo Oliveira	SANTOS
Vagner Love	CORINTHIANS
Jadson	CORINTHIANS
LUCAS PRATTO	ATLÉTICO
Andre	SPORT
Henrique	CORITIBA
null	ASSIS

4.1.5 Outer Full Join

Nesse caso seria a junção dos caso INNER JOIN, LEFT OUTER JOIN E RIGTH OUTER JOIN. No exemplo, vamos selecionar todos os jogadores e times independente de relacionamento entre as tabelas

Exemplo:

```
SELECT j.nome, t.nome
FROM jogador j, time t
```

```
WHERE t.id_time = j.id_time;
```

retornaria

j.nome	t.nome
Ricardo Oliveira	SANTOS
Vagner Love	CORINTHIANS
Jadson	CORINTHIANS
LUCAS PRATTO	ATLÉTICO
Andre	SPORT
Henrique	CORITIBA
Joao da silva	null
null	ASSIS

4.2 Funções de Agregação

Utilizamos agrupamento para juntar os dados equivalentes com a palavra **group by**. As Funções de Agregação são utilizadas para manipular os dados agrupados.

4.2.1 Count

Conta o número de linhas afetadas pelo comando.

Exemplo:

```
SELECT COUNT(1) quantidade_jogadores,
```

```
      t.nome
FROM   jogador j,
       time t
WHERE t.id_time = j.id_time
GROUP BY t.nome
```

4.2.2 Sum

Faz o somatório do valor das colunas especificadas.

Exemplo:

```
SELECT SUM(j.salario) salario,
       t.nome
FROM   jogador j,
       time t
WHERE t.id_time = j.id_time
GROUP BY t.nome;
```

4.2.3 Avg

Calcula a média aritmética dos valores das colunas.

Exemplo:

```
SELECT AVG(t.salario) salario,
       t.nome
FROM   jogador j,
       time t
WHERE t.id_time = j.id_time
GROUP BY t.nome;
```

4.2.4 Min

Pega o menor valor da coluna de um grupo de linhas.

Exemplo:

```
SELECT MIN(t.salario) salario,
    j.nome
FROM   jogador j;
```

4.2.5 Max

Pega o maior valor da coluna de um grupo de linhas.

Exemplo:

```
SELECT MAX(t.salario) salario,
    t.nome
FROM   jogador j;
```

4.3 Exercícios

1. Gere uma consulta retornando a folha de pagamento de cada time.
2. Gere uma consulta retornando a média salarial de cada time.
3. Gere uma consulta que retorne o menor salário de cada time.
4. Gere uma consulta que retorne o maior salário de cada time.

5. PLSQL

5.1 O que é PL/SQL?

Oracle PL/SQL (Procedural Language/SQL) é uma extensão da linguagem SQL (Structured Query Language) que tem por objetivo, auxiliar as tarefas de programação no ambiente Oracle de banco de dados, trazendo ao SQL características procedurais não suportadas no padrão ANSI.

Programar em PL/SQL, significa ter a disposição, um ambiente procedural desenvolvido para aplicações de bancos de dados, beneficiando-se do controle transacional inerente das aplicações deste tipo.

5.2 As vantagens do PL/SQL

- Capacidade Procedural.
- Aumento de performance de procedimentos.
- Ganho de produtividade.
- Portabilidade entre Oracle Servers (tipos diferentes).
- Integração direta com o RDBMS Oracle (relational database management system).
- Facilidade para desenvolvimento particionado (Cliente X Server).

5.3 Diferenças da Sintaxe SQL e PL/SQL

Conforme definido, PL/SQL é uma extensão ao SQL trazendo a esta linguagem padrão ANSI, capacidade procedural e, portanto, a integração entre as duas transcorre de forma bastante natural, diferentemente da programação com SQL através de linguagens tradicionais de 3a geração.

5.4 Blocos Anônimos

Um bloco PL/SQL que é utilizado para realizar alguma lógica que não é definido ou nomeado como procedure, function, trigger ou outro objeto nativo do Oracle é comumente chamado de Bloco Anônimo.

Composição de um bloco Anônimo.

- Declarativa (opcional).
- Executável (obrigatória).
- Manipulação de Exceções e Erros (opcional).

Exemplo de bloco anônimo sem ação lógica.

```
DECLARE
-- Local variables here
BEGIN
-- Logical code here
  null;
END;
```

5.5 Comentários

Comentários são textos colocados no meio dos códigos que não afetam seu funcionamento.

Seguindo as boas práticas do mundo de desenvolvimento computacional todo código deve ser o mais claro e intuitivo possível (Clean Code), tornando os comentários pouco utilizados no dia-a-dia, porém há momentos que se torna essencial em um código para ser tornar de fácil de se ler e de se manter.

Exemplos:

```
--line comments.
/* block comments */
/*
block comments
here
*/
```

5.6 Desvios Condicionais

Um desvio condicional acontece quando o programa precisa "tomar uma decisão", ou seja, seguir diferentes caminhos de acordo com determinada condição. São os populares “IF” das linguagens de programação, mas não são somente isso. Desvios condicionais são quaisquer operações utilizadas para tomada de decisão, *estando dentro de um bloco lógico e até uma consulta SQL*.

5.6.1 IF/ELSE IF/ELSE

Exemplos:

```
IF (condição) THEN
  /* comandos aqui */
END IF;

IF (condição) THEN
  /* comandos aqui */
ELSE
  /* comandos aqui */
END IF;

IF (condição) THEN
/* comandos aqui */
ELSIF (condição2) THEN
  /* comandos aqui */
ELSIF (condição3) THEN
  /* comandos aqui */
ELSE
  /* comandos aqui */
END IF;
```

5.6.2 CASE

Exemplo:

```
DECLARE
  vnNumero  number(1) := 1;
  vNRetorno number;
BEGIN
  varialveDeRetorno := CASE
                       WHEN vnNumero = 1 then
                        11
                       WHEN vnNumero = 2 then
                        22
                       ELSE
                        33
                      END;

  dbms_output.put_line(varialveDeRetorno);
END;
```

5.6.3 DECODE

Exemplo:

```
SELECT DECODE(nome,
              null,
              'Técnico sem nome',
              'chuck norris',
              '!!!!!!',
              nome)
FROM tecnico;
```

5.7 Exceptions

Quando um bloco PL/SQL é executado, isto se faz após o parágrafo BEGIN, onde comandos são executados, na ocorrência de algum erro ou anormalidade, uma exceção é levantada (raise

exception) e o fluxo normal do programa é desviado para o parágrafo declarado na área de EXCEPTION. Dentro da área de EXCEPTION, blocos PL/SQL podem ser executados ou um desvio para fora do bloco principal poderá ocorrer, terminando a execução da rotina ou procedimento.

Exemplo:

```
BEGIN
  -- Código que pode gerar excessões;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Erro ao executar operações. Erro :' || SQLERRM);
END;
```

Exceptions mais utilizadas:

1. OTHERS, qualquer erro disparado pode ser tratado por este.
2. TOO_MANY_ROWS, quando uma consulta retorna mais uma linha (erro cartesiano).
3. NO_DATA_FOUND, quando uma consulta não retorna nenhuma informação.

5.8 Cursores

Cursores comumente são vetores que apontam para determinados “lugares” da memória, no caso do Oracle Registros do banco de dados.

Quando trabalhamos com cursores estamos implicitamente trabalhando com laços de repetições(loops) então veremos os laços básicos e os cursores mais utilizados.

5.8.1 Loop Básico

Neste tipo de “laço de repetição” sempre inicia execução.

Exemplo:

```
LOOP
  --imprime via dbms a variável vnContador
```

```
 DBMS_OUTPUT.put_line(vnContador);

 --incremento a variável vnContador
 vnContador := vnContador + 1;

 --força a saída do laco de repeticao caso atenda a condição
 EXIT WHEN vnContador = 20;

END LOOP;
```

5.8.2 While Loop

Neste tipo de “laço de repetição” só se inicia somente se atender a condição determinada.

Exemplo:

```
WHILE vnContador < 10 LOOP

        --imprimo via dbms a variável vnContador
        DBMS_OUTPUT.put_line(vnContador);

        --incremento a variável vnContador
        vnContador := vnContador + 1;

END LOOP;
```

5.8.3 Cursor Básico

Exemplo:

```
DECLARE
  --declaração do cursor
  CURSOR vcCursor is
```

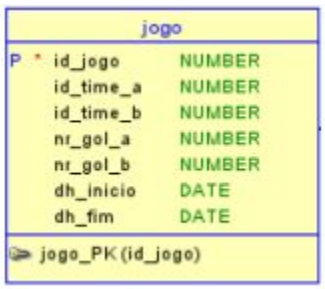
```
   SELECT * FROM time;

BEGIN

  --percorro o cursor selecionado
  For meuCursor in vcCursor loop
  --imprimo via dbms a o registro atual da coluna nome do cursor que estou percorrendo

   DBMS_OUTPUT.put_line(meuCursor.nome);
  --
  END LOOP;

END;
```

5.9 Exercícios

1. Criar blocos anônimos que gerem uma saída DBMS básica (‘Hello DBMS’) utilizando o pacote da Oracle DBMS_OUTPUT.PUTLINE(‘text’).

2. Gerar uma saída DBMS contendo as informações de um time e do seu técnico.

3. Criar uma tabela que armazena os dados do jogo (seguindo exemplo do diagrama abaixo) e gerar um jogo composto de dois times diferentes e escalar os jogadores participantes.



4. Incluir comentários nos blocos anônimos anteriores sem alterar o funcionamento.

5. Altera bloco anônimo que gerar o jogo composto de dois times diferentes e adicionar quantos “if” forem necessários para que não permita inserir um jogo sendo o mesmo time para “ambos os lados”.

6. Criar uma consulta que retorne os jogos, times que estão participando, e placar, sendo que na coluna placar deve trazer o número de gols de cada time respeitando o seu lado na ordenação das

colunas (4 - 3) e caso seja o mesmo número de gols deve aparecer ‘empate’.

7. Criar bloco anônimo com algum comando dml “insert” que irá causar erro, e tratar com a exception OTHERS.

8. Criar bloco anônimo com alguma consulta que irá obter um valor e popular em uma variável e “forçar” um erro, e tratar com a exception TOO_MANY_ROWS.

9. Marcar alguns gols para o jogo gerado respeitando o placar definido no jogo.

10. Criar um laço de repetição que imprima os números de 0 a 100 via DBMS.

11. Criar um laço de repetição que imprima os números de 0 a 100 pares via DBMS.

12. Percorrer todos os jogos e inserir um gol para cada jogador.

6. Objetos Oracle

6.1 Procedures

Stored procedures, e stored functions, são objetos do banco de dados Oracle, compostos de códigos PL/SQL e SQL, logicamente agrupados, com a finalidade de executarem uma atividade específica.

Diferente de um bloco anônimo após definidas, são compiladas pelo Oracle e armazenadas no banco de dados a fim de que possam ser acessadas e executadas em qualquer momento.

Por este motivo, as procedures e functions, são compostas de código programável e possuem um nome, que as identifica univocamente dentro de um schema específico no dicionário de dados.

As procedures e functions são sempre criadas no schema de um usuário.

As procedures e functions são compostas de parâmetros de entrada e/ ou saída.

As procedures podem ser executadas por diversos ambientes, como por exemplo, dentro do código de outras procedures, de blocos PL/SQL anônimos, de database triggers, de linguagens 3GL, etc.

Exemplo:

```
CREATE OR REPLACE PROCEDURE NOME_DA_PROCEDURE
(
  PARAM_ENTRADA  NUMBER,
  PARAM_ENTRADA2 IN VARCHAR2,
  PARAM_SAIDA    OUT NUMBER,
  PARAM_ENT_SAID IN OUT VARCHAR2
) AS
  -- variaveis;
BEGIN
  -- código lógico aqui;
END;
```

6.2 Functions

Functions, retornam sempre um valor, seguindo esta lógica, comumente não possuem parâmetros de saída diferente das *Stored Procedures* recebem parâmetros de entrada e saída quando necessário.

Exemplo:

```
CREATE OR REPLACE FUNCTION NOME_DA_FUNCTION(PARAM_ENTRADA
NUMBER,       PARAM_ENTRADA2 VARCHAR2) RETURN VARCHAR2 AS
  -- variáveis
BEGIN
  -- código lógico
END;
```

6.3 Packages

Uma package é um objeto que fica armazenado no banco de dados Oracle e que agrupa um conjunto de stored procedures, stored functions, cursores, variáveis e exceptions, geralmente envolvidos numa mesma regra de negócios, com o objetivo de encapsular esses objetos.

Além da organização e facilidade a package deixa a possibilidade aberta de utilizar o polimorfismo, podendo haver mais de um objeto interno com mesmo nome mas com parâmetros diferentes os tornando distintos em sua chamada.

Uma Package é subdividida em duas partes principais:

Package Specification

Onde é feita a declaração da package e dos objetos públicos que a compõem.

Package Body

Onde ficam armazenadas as definições dos códigos dos objetos públicos declarados na package specification, onde ficam armazenadas as declarações e as definições dos objetos privados de uma package.

Exemplo:

```
CREATE OR REPLACE PACKAGE campeonato IS
  /* variáveis globais cursors globais */
    PROCEDURE insereTime(pinIdTime number);
```

```
END;


CREATE OR REPLACE PACKAGE BODY campeonato IS
 /* variáveis privados cursors privados */
 PROCEDURE insereTime(pinIdTime number) IS

 BEGIN
  /* código que insere o jogador */
  null;
 END;
```

6.4 SubQuerys

Subquery é um dos muitos termos para uma consulta sql dentro de outra consulta, outros autores costumam chamar como subselect.

A necessidade de uma subquery pode surgir com a necessidade de trazer campos adicionais sem realizar junções de tabelas na consulta, ou verificação de existência de dados, entre outras.

Vamos focar em subqueries na funcionalidade “exists” que é definida nas condicionais das consultas (where, and).

Exemplos:

```
--seleciono todos os times que não possuem jogadores
SELECT * FROM time tim WHERE NOT EXISTS
(SELECT * FROM jogador jog WHERE jog.id_time = tim.id_time);

--seleciono todos os times que possuem jogadores
SELECT * FROM time tim WHERE EXISTS
(SELECT * FROM jogador jog WHERE jog.id_time = tim.id_time);

--apago todos jogadores que não participaram de nenhum jogo
DELETE jogador jog WHERE NOT EXISTS
(SELECT * FROM jogo_jogador jjg WHERE jjg.id_jogador = jog.id_jogador);
```

6.5 Triggers

Trigger (Gatilho) é uma construção PL/SQL semelhante a uma procedure, possui nome e blocos com seções declarativas, executáveis e de manipulação de erros e exceptions.

Porém a grande diferença entre estes objetos é que uma procedure é executada de forma explícita através de uma aplicação, linha de comando ou outra construção PL/SQL, manipulando parâmetros com o programa chamador. Já um trigger, é sempre executado de forma implícita, como consequência da ocorrência de um evento de TRIGGER (a triggering event) e não aceita parâmetros.

Um evento de trigger, o qual chamamos de disparo do trigger, consiste numa

operação DML (Insert, Update ou Delete) sobre uma tabela do banco de dados.

Devemos sempre definir qual operação DML deve disparar a trigger, before (antes) ou after (depois) de alguma operação DML (insert or update or delete)

Quando estamos no bloco lógico de uma trigger podemos obter um valor de uma determinada coluna da tabela que implementamos a trigger, isto pode ser feito utilizando os comandos:

:old.nome_da_coluna --valor anterior do campo.
:new.nome_da_coluna --novo valor do campo.

ON INSERT

:old traz nulo afinal antes não existia o registro.

:new traz o novo valor.

ON UPDATE

:old traz o valor anterior ao update.

:new traz o novo valor.

ON DELETE

:old traz o valor anterior ao delete.

:new traz nulo já que está sendo removido o registro.

Exemplo:

```
CREATE OR REPLACE TRIGGER nome_da_trigger
  AFTER INSERT OR UPDATE OR DELETE ON gol
  FOR EACH ROW
BEGIN
  -- código lógico aqui;
END;
```

6.6 Views

View são representações lógicas de um ou muitas outras tabelas. Uma view é montada a partir de uma estrutura de tabelas, então ao consultar uma view seria como executar o sql propriamente dito.

A view serve como uma camada de abstração e controle, pois além de tornar o acesso a uma determinada tabela fácil se torna simples de liberar acesso a dados de uma determinada origem sem acessar a própria tabela.

Exemplo:

```
CREATE OR REPLACE VIEW vTime AS
SELECT * FROM time;
```

Então com este objeto de abstração criado podemos consultá-lo como uma tabela.

Exemplos:

```
SELECT * FROM vTime;
SELECT * FROM time;
```

Uma view suporta todas operações de insert, updates e deletes caso tenha seu acesso permitido (grants).

6.7 Views Materializadas

Uma view materializada (Snapshots) é muito semelhante a uma view comum, mas sua principal diferença é que enquanto a view executa a consulta no momento do select a view materializada ele atualiza os registros de tempo em tempo armazenando em cache, permitindo um melhor desempenho se tratando de servidores remotos.

6.8 Exercícios

1. Criar uma trigger que depois de inserir algum gol ele contabiliza mais um ao jogo para aquele determinado time.
2. Alterar a trigger criada anteriormente para que ela subtraia um gol do jogo caso o registro do gol tenha sido apagado.
3. Criar uma tabela de log de jogador para que armazene todos os dados antes de qualquer alteração.
4. Criar uma trigger que antes de realizar qualquer alteração em um jogador ele guarde no log o estado anterior.
5. Criar uma consulta que retorne somente os cartões que já foram aplicados em algum jogo.
6. Reduzir em 10% o salário de todos os jogadores que receberam pelo menos um cartão.
7. Criar uma procedure insere cartões, recebendo somente a descrição do cartão.
8. Criar uma function que retorne a quantidade de gols de um jogador específico, sendo este o parâmetro de entrada.
9. Criar uma consulta que retorne os times, os jogadores do time e a quantidade de gols de cada jogar (utilizar function criada).