


CURSO DE

# FÉRIAS



**SQL E PL/SQL  
BÁSICO**

## Sumário

1. Introdução a banco de dados.	2
2. Comandos DDL	3
3. Comandos DML.	7
4. Comandos DML II.	8
5. PLSQL.	14
6. Procedures e Functions	18
7. SubQuerys.	19
8. Mais Objetos Oracle	20
9. Packages	22
10. Ambiente e Resoluções.	23

# 1. Introdução a banco de dados.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

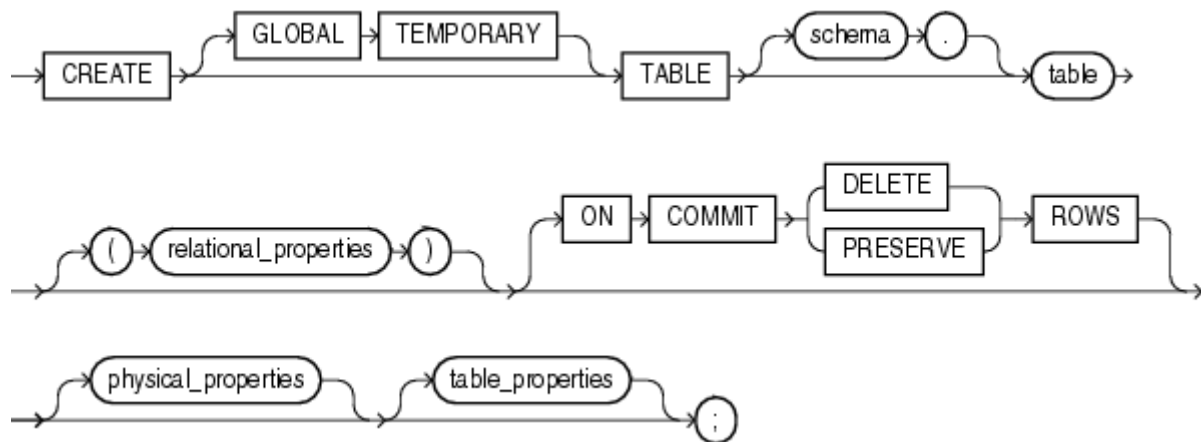
### Tipos de dados

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(comprimento_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

### 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 mantêm 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 times
( id_time      number          NOT NULL,
  nome         varchar2(400) NOT NULL
) TABLESPACE treinamento;
```

## 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 nessa 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.

```
CREATE TABLE time
( id_time      number          NOT NULL,
  nome         varchar2(400) NOT NULL,
  ind_ativo    varchar2(1),
  id_tecnico   number          NOT NULL
) TABLESPACE treinamento;

--
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';
comment on column time.ind_ativo IS 'Indica se o time está ativo ou não';
--
ALTER TABLE time MODIFY ( ind_ativo NOT NULL );
--
ALTER TABLE time ADD (CONSTRAINT ck_time_ativo
  CHECK (UPPER(ind_ativo)='S' or
        UPPER(ind_ativo)='N') );
--
ALTER TABLE time ADD CONSTRAINT uk_nome_time UNIQUE ( nome );
--
CONSTRAINT fk_id_tecnico FOREIGN KEY (id_tecnico) REFERENCES tecnico ( id_tecnico ),
CONSTRAINT pk_id_time PRIMARY KEY ( id_time );
```

## Comentários

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

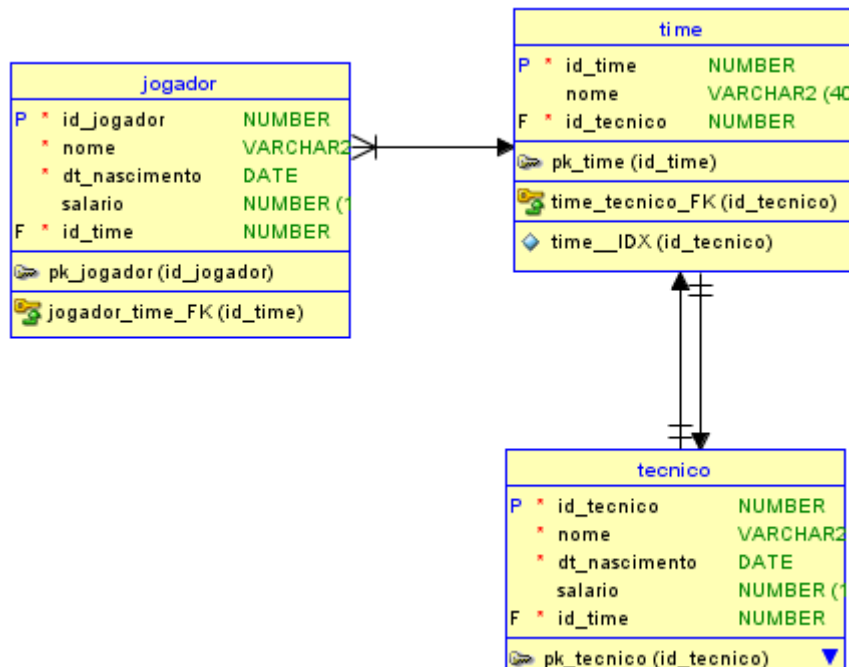
Exemplos:

```
comment on table times IS '[CADASTRO] Tabela para armazenamento de times';
comment on column times.id_time IS 'Código identificador do time';
comment on column times.nome IS 'Nome do time';
```

**Exercícios** (utilizar o DER abaixo):

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.

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

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

**Select \* from time where nome = 'BARCELONA' order by nome**

- O **COMMIT** é um comando utilizado no controle transacional, faz com quem 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.
- A instrução **INSERT** é utilizada para inserir dados no banco de dados.

**insert into time (id\_time, nome) values (1,'BARCELONA')**

- 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**.
- Este objeto possui dois métodos pré-definidos:
  1. nextval: retorna o próximo valor da sequence.
  2. Curval: retorna o último número gerado pela sequence.

[illegible]

```
/*exemplo de utilização de sequences*/
select seq_jogador.nextval from dual
```

- Exercícios:
  - Inserir 2 times.
  - Inserir 2 técnicos.
  - Inserir 11 jogadores em um time.
  - Listar todos jogadores de um determinado time.
  - Listar todos times.
  - Listar técnicos com mais de 40 anos.
  - Inserir os jogadores existentes para o outro time (select insert com sequence).
- A instrução UPDATE é utilizada para alterar dados já existentes no banco de dados.
  - **update** time **set** nome = 'BARCELONA FUTEBOL' **where** id\_time = 1
- Exercícios:
  - Inserir um time novo.
  - Alterar todos jogadores de um time para o novo time.
  - Aumentar em 10% o salário de todos jogadores do novo time.
  - Aumentar o salário de todos técnicos em 20%.
- A instrução DELETE é utilizada para remover dados no banco de dados.
  - **delete** times **where** id\_time = 1
- Exercícios:
  - Inserir um time novo.
  - Inserir 3 jogadores extras no time novo.
  - Alterar o salário de 3 jogadores para valores acima de R\$ 100.000,00.
  - Remover jogadores do novo time com salários superiores R\$ 100.000,00.
  - Remover times que estejam sem jogadores e técnicos.



## 4. Comandos DML II

### Junções de Dados e Apelidando no Oracle

Quando queremos juntar mais de uma tabela, podemos utilizar um dos seguintes comandos:

Considerando as figuras abaixo

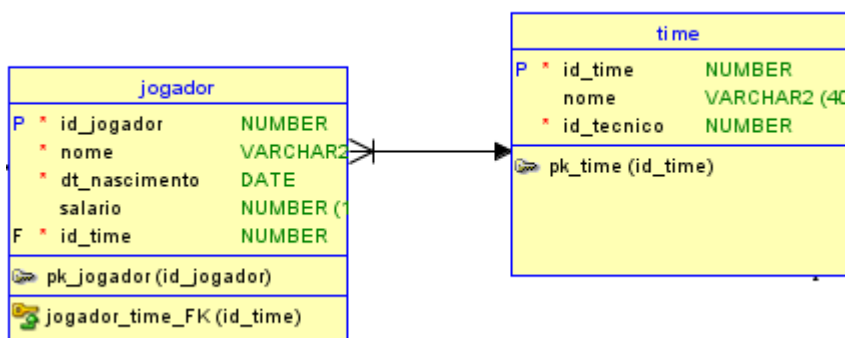


Figura 4.1: Jogador - Time

Jogador					Time	
id_jogador	nome	dt_nascimento	salario	time_id_time	id_time	nome
1	Ricardo Oliveira	06/05/80	50000	1	1	SANTOS
2	Vagner Love	11/06/84	40000	2	2	CORINTHIANS
3	Jadson	05/10/83	60000	2	3	ATLÉTICO
4	LUCAS PRATTO	04/06/88	20000	3	4	SPORT
5	Andre	27/09/90	30000	4	5	CORITIBA
6	Henrique	27/05/91	15000	5	6	ASSIS
7	Joao da silva	11/11/60	0			

Figura 4.2: Figura 4.2: Consultas

#### A. 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

Ex:

```

select j.nome, t.nome
from jogador j, time t
  
```

*retornaria*

i.nome	f.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

## B. Inner join

Quando queremos juntar duas ou mais tabelas, que internamente, tenham valores correspondentes. No exemplo Jogador X Time temos o **id\_time** no lado de Jogador e **id\_time** no lado de Time.

Ex:

```
select j.nome, t.nome
from jogador j, time t
where t.id_time = j.time_id_time
```

*retornaria*

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

## C. 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.

Ex:

```
select j.nome, t.nome
from jogador as j, time as t
where j.time_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

#### D. 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.

Ex:

```

Select j.nome, t.nome
from jogador as j, time as t
where j.time_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

#### E. 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

Ex:

```

select j.nome, t.nome
from jogador as j, time as t
where t.id_time = j.time_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

## 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 ordenados pelo nome. Ex.

```
select nome from jogador order by nome asc
```

- Exercícios:
  - Selecione os Times em ordem crescente.
  - Selecione os nomes de jogadores e seus respectivos nomes dos times ordenados(asc) pela data de nascimento dos jogadores.

## Funções de Agrupamento e agregação

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

Função	Ação
COUNT	Conta o número de linhas afetadas pelo comando.
SUM	Faz o somatório do valor das colunas especificadas.
AVG	Calcula a média aritmética dos valores das colunas.
MIN	Pega o menor valor da coluna de um grupo de linhas.
MAX	Pega o maior valor da coluna de um grupo de linhas.

```
select count(1) as N_jogadores, t.nome
from jogador as j , time as t
where t.id_time = j.time_id_time
group by t.nome
```

N_jogadores	t.nome
1	SANTOS
2	CORINTHIANS
1	ATLÉTICO
1	SPORT
1	CORITIBA

- Exercícios:
  - Gere uma consulta retornando a folha de pagamento de cada time.
  - Gere uma consulta retornando a média salarial de cada time.
  - Gere uma consulta que retorne o menor salário de cada time.
  - Gere uma consulta que retorne o maior salário de cada time.

## 5. PLSQL.

### 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.

### 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).

### Diferenças da Sintaxe SQL e PLSql

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.

### 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;
```

- Exercícios:
  - Criar Blocos Anônimos que:
    - Gere uma saída DBMS básica ('Hello DBMS') utilizando o pacote da Oracle DBMS\_OUTPUT.PUTLINE('text').
    - Gerar uma saída DBMS contendo as informações de um time e do seu técnico.
    - Gerar um jogo composto de dois times diferentes e escalar os jogadores participantes.
    - Marcar alguns gols para o jogo gerado respeitando o placar definido no jogo.

## 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.

*--line comments.*

*/\* block comments \*/*

*/\**

*block comments*

*here*

*\*/*

- Exercícios:
  - Incluir comentários nos blocos anônimos anteriores sem alterar o funcionamento.

## Desvios Condicionais

Desvios Condicionais são os populares “IF” das linguagens de programação?

Resposta, sim mas não somente pois Desvios condicionais são quaisquer operações utilizadas para tomada de decisão, estando dentro de um bloco lógico e até uma consulta SQL.

```
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;

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;

select decode(nome,
    null,
    'Técnico sem nome',
```



```
'chuck norris',  
' !!!!! ' ,  
nome)  
from tecnico
```

- Exercícios:
  - 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”.
  - 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’.

## 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.

```
BEGIN  
    --codigoQuePodeHaverExcessoes;  
EXCEPTION  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('Erro ao executar operacoes. Erro : ' || SQLERRM);  
END;
```

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

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.

## Cursors.

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

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

Loop Básico(Entro no laço sempre).

loop

```
--imprimo via dbms a variável vnContador
dbms_output.put_line(vnContador);
--incremento a variável vnContador
vnContador := vnContador + 1;
--forco sair do laço de repeticao caso atenda a minha condição (vnContador = 20)
exit when vnContador = 20;
end loop;
```

*While Loop (Entro no laço somente se atender a condição).*

```
while vnContador < 10 loop
    --imprimo via dbms a variável vnContador
    dbms_output.put_line(vnContador);
    --incremento a variável vnContador
    vnContador := vnContador + 1;
end loop;
```

### **Cursor Básico**

```
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;
```

- Exercícios:
  - Criar um laço de repetição que imprima os números de 0 a 100 via DBMS.
  - Criar um laço de repetição que imprima os números de 0 a 100 **pares** via DBMS.
  - Percorrer todos os jogos e inserir um gol para cada jogador.

## 6. Procedures e Functions

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.

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

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.

*Estrutura de uma procedure:*

```
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
  --codigo Logico Aqui;
END; --ou END NOME_DA_PROCEDURE;

CREATE OR REPLACE FUNCTION NOME_DA_FUNCTION(PARAM_ENTRADA NUMBER, PARAM_ENTRADA2 VARCHAR2) RETURN VARCHAR2 AS
  --VARIABLES
BEGIN
  --codigo lógico aqui.
end; --ou end NOME_DA_FUNCTION
```

- Exercícios:
  - Criar uma procedure insere cartões, recebendo somente a descrição do cartão.
  - Criar uma function que retorne a quantidade de gols de um jogador específico, sendo este o parâmetro de entrada.

- Criar uma consulta que retorne os times, os jogadores do time e a quantidade de gols de cada jogar (utilizar function criada).

## 7. 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 subquerys na funcionalidade “exists” que é definida nas condicionais das consultas (where, and).

```
--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)
```

- Exercícios:
  - Criar uma consulta que retorne somente os cartões que já foram aplicados em algum jogo.
  - Reduzir em 10% o salário de todos os jogadores que receberam pelo menos um cartão.

## 8. Mais Objetos Oracle

### 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.
```

```
create or replace trigger nome_da_trigger  
after insert or update or delete on gol  
for each row  
begin  
--codigo lógico aqui;
```

```
end;
```

- Exercícios:
  - Criar uma trigger que depois de inserir algum gol ele contabilize mais um ao jogo para aquele determinado time.
  - Alterar a trigger criada anteriormente para que ela subtraia um gol do jogo caso o registro do gol tenha sido apagado.
  - Criar uma tabela de log de jogador para que armazene todos os dados antes de qualquer alteração.
  - Criar uma trigger que antes de realizar qualquer alteração em um jogador ele guarde no log o estado anterior.

## 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.

```
Create or replace view vTime as  
select * from time;
```

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

```
select * from vTime;  
select * from time;
```

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

## 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.

## 9. 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.

```
create or replace package campeonato is
  /*variáveis globais
  cursors globais
  */
  procedure adicionarJogador(pisNome varchar2);
end;
```

```
create or replace package body campeonato is
  /*variáveis privados
  cursors privados
  */
  procedure adicionarJogador(pisNome varchar2) is
  begin
    /*codigo que insere o jogador*/
    null;
  end;
```

```
end;
```

### *Ambiente e Resoluções.*

```
--cartao
CREATE TABLE
cartao
( id_cartao NUMBER NOT NULL , descricao VARCHAR2 (400) ) ;
ALTER TABLE cartao ADD CONSTRAINT pk_cartao PRIMARY KEY ( id_cartao ) ;

--gol
CREATE TABLE gol
( id_gol NUMBER NOT NULL , dh_gol DATE , id_jogo_jogador NUMBER NOT NULL ) ;
ALTER TABLE gol ADD CONSTRAINT gol_PK PRIMARY KEY ( id_gol ) ;

--jogador
CREATE TABLE jogador
( id_jogador NUMBER NOT NULL , nome VARCHAR2 (400) NOT NULL , dt_nascimento DATE NOT
NULL ,
salario NUMBER (18,2) , id_time NUMBER NOT null ) ;
ALTER TABLE jogador ADD CONSTRAINT pk_jogador PRIMARY KEY ( id_jogador ) ;

--jogo
CREATE TABLE jogo
( id_jogo NUMBER NOT NULL , id_time_a NUMBER , id_time_b NUMBER , nr_gol_a NUMBER ,
nr_gol_b NUMBER , dh_inicio DATE , dh_fim DATE ) ;
ALTER TABLE jogo ADD CONSTRAINT jogo_PK PRIMARY KEY ( id_jogo ) ;

--jogo cartao jogador
CREATE TABLE jogo_cartao_jogador
( id_jogo_cartao_jogador NUMBER NOT NULL , dh_penalidade DATE , id_cartao
NUMBER NOT NULL ,
id_jogo_jogador NUMBER NOT NULL ) ;
ALTER TABLE jogo_cartao_jogador ADD CONSTRAINT pk_jogo_cartao_jogador PRIMARY KEY
( id_jogo_cartao_jogador ) ;

--jogo jogador
CREATE TABLE jogo_jogador
(
id_jogo_jogador NUMBER NOT NULL , id_jogo NUMBER NOT NULL , id_jogador NUMBER NOT NULL
) ;
ALTER TABLE jogo_jogador ADD CONSTRAINT pk_jogo_jogador PRIMARY KEY ( id_jogo_jogador ) ;

--tecnico
CREATE TABLE tecnico
(
id_tecnico NUMBER NOT NULL , nome VARCHAR2 (400) NOT NULL , dt_nascimento DATE NOT NULL
,
salario NUMBER (18,2) , id_time NUMBER NOT NULL ) ;
ALTER TABLE tecnico ADD CONSTRAINT pk_tecnico PRIMARY KEY ( id_tecnico ) ;

--time
CREATE TABLE TIME
( id_time NUMBER NOT NULL , nome VARCHAR2 (400) ) ;

--foreign keys
ALTER TABLE TIME ADD CONSTRAINT pk_time PRIMARY KEY ( id_time ) ;
ALTER TABLE gol ADD CONSTRAINT gol_jogo_jogador_FK FOREIGN KEY ( id_jogo_jogador ) REFERENCES jogo_jogador
( id_jogo_jogador ) ;
ALTER TABLE jogador ADD CONSTRAINT jogador_time FK FOREIGN KEY ( id_time ) REFERENCES TIME ( id_time ) ;
ALTER TABLE jogo_cartao_jogador ADD CONSTRAINT jogojcartaojogador_cartao_FK FOREIGN KEY ( id_cartao )
REFERENCES cartao ( id_cartao ) ;
ALTER TABLE jogo_cartao_jogador ADD CONSTRAINT jogcartajog_jogojogador_FK FOREIGN KEY ( id_jogo_jogador )
REFERENCES jogo_jogador ( id_jogo_jogador ) ;
ALTER TABLE jogo_jogador ADD CONSTRAINT jogo_jogador_jogador_FK FOREIGN KEY ( id_jogador ) REFERENCES jogador
```



```
( id_jogador ) ;
ALTER TABLE jogo_jogador ADD CONSTRAINT jogo_jogador_jogo_FK FOREIGN KEY ( id_jogo ) REFERENCES jogo
( id_jogo ) ;
ALTER TABLE tecnico ADD CONSTRAINT tecnico_time_FK FOREIGN KEY ( id_time ) REFERENCES TIME ( id_time ) ;
```

### 3.

```
-- Criação Sequence
CREATE SEQUENCE SEQ_JOGADOR
MINVALUE 1
START WITH 1
INCREMENT BY 1
NOCACHE;

-- Inserir 2 times.
INSERT INTO TIME (ID_TIME, NOME) VALUES (1, 'BARCELONA');

INSERT INTO TIME (ID_TIME, NOME) VALUES (2, 'REAL MADRID');
COMMIT;

-- Inserir 2 técnicos.
INSERT INTO TECNICO (ID_TECNICO, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (1, 'JOSE MOURINHO', '23/06/1970', 300000, 1);

INSERT INTO TECNICO (ID_TECNICO, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (2, 'ZICO', '14/02/1960', 70000, 2);
COMMIT;

-- Inserir 11 jogadores em um time.
INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'RONALDINHO', '02/04/1980', 500000, 1);

--jogo jogador
CREATE TABLE jogo_jogador
(
    id_jogo_jogador NUMBER NOT NULL ,    id_jogo          NUMBER NOT NULL ,    id_jogador        NUMBER NOT NULL ) ;
ALTER TABLE jogo_jogador ADD CONSTRAINT pk_jogo_jogador PRIMARY KEY ( id_jogo_jogador ) ;

--tecnico
CREATE TABLE tecnico
(
    id_tecnico      NUMBER NOT NULL ,    nome              VARCHAR2 (400) NOT NULL ,    dt_nascimento    DATE NOT NULL ,
    salario         NUMBER (18,2) ,    id_time           NUMBER NOT NULL ) ;
ALTER TABLE tecnico ADD CONSTRAINT pk_tecnico PRIMARY KEY ( id_tecnico ) ;

--time
CREATE TABLE TIME
(
    id_time         NUMBER NOT NULL ,    nome              VARCHAR2 (400) ) ;

--foreign keys
ALTER TABLE TIME ADD CONSTRAINT pk_time PRIMARY KEY ( id_time ) ;
ALTER TABLE gol ADD CONSTRAINT gol_jogo_jogador_FK FOREIGN KEY ( id_jogo_jogador ) REFERENCES jogo_jogador ( id_jogo_jogador ) ;
ALTER TABLE jogador ADD CONSTRAINT jogador_time_FK FOREIGN KEY ( id_time ) REFERENCES TIME ( id_time ) ;
ALTER TABLE jogo_cartao_jogador ADD CONSTRAINT jogojcartaojogador_cartao_FK FOREIGN KEY ( id_cartao ) REFERENCES cartao ( id_cartao ) ;
ALTER TABLE jogo_cartao_jogador ADD CONSTRAINT jogcartajog_jogojogador_FK FOREIGN KEY ( id_jogo_jogador )
REFERENCES jogo_jogador ( id_jogo_jogador ) ;
ALTER TABLE jogo_jogador ADD CONSTRAINT jogo_jogador_jogador_FK FOREIGN KEY ( id_jogador ) REFERENCES jogador ( id_jogador ) ;
ALTER TABLE jogo_jogador ADD CONSTRAINT jogo_jogador_jogo_FK FOREIGN KEY ( id_jogo ) REFERENCES jogo ( id_jogo ) ;
ALTER TABLE tecnico ADD CONSTRAINT tecnico_time_FK FOREIGN KEY ( id_time ) REFERENCES TIME ( id_time ) ;

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'CRISTIANO RONALDO', '02/06/1986', 700000, 1);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'GUERREIRO', '28/03/1987', 200000, 1);
```

```

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'ROBINHO', '13/05/1988', 400000, 1);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'PIRLO', '08/05/1978', 300000, 1);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'JULIO CESAR', '03/06/1985', 60000, 1);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'NEYMAR JR', '18/08/1991', 700000, 1);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'SERGIO RAMOS', '04/08/1984', 200000, 1);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'FABREGAS', '17/11/1988', 100000, 1);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'PELE', '09/09/1950', 400000, 1);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'MARADONA', '06/06/1966', 100000, 1);

COMMIT;

-- Listar todos jogadores de um determinado time.
SELECT NOME
FROM JOGADOR
WHERE ID_TIME = 1;

-- Listar todos times.
SELECT *
FROM TIME;

-- Listar técnicos com mais de 40 anos.
SELECT NOME
FROM TECNICO
WHERE DT_NASCIMENTO < '01/01/1976';

-- Inserir os jogadores existentes para o outro time (select insert).
INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
SELECT SEQ_JOGADOR.NEXTVAL, NOME, DT_NASCIMENTO, SALARIO, 2
FROM JOGADOR;

COMMIT;

-- Inserir um time novo.
INSERT INTO TIME (ID_TIME, NOME)
VALUES (3, 'MILAN');
COMMIT;

-- Alterar todos jogadores de um time para o novo time
UPDATE JOGADOR
SET ID_TIME = 3
WHERE ID_TIME = 1;
COMMIT;

-- Aumentar em 10% o salario de todos jogadores do novo time
UPDATE JOGADOR
SET SALARIO = SALARIO * 1.10
WHERE ID_TIME = 3;
COMMIT;

-- Aumentar o salario de todos técnicos em 20%.
UPDATE TECNICO
SET SALARIO = SALARIO * 1.20;
COMMIT;

-- Inserir um time novo
INSERT INTO TIME (ID_TIME, NOME)
VALUES (4, 'INTERNAZIONALE');
COMMIT;

-- Inserir 3 jogadores extras no time novo.

```

```

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'BEBETO', '06/04/1976', 30000, 4);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'ROMARIO', '05/07/1970', 70000, 4);

INSERT INTO JOGADOR (ID_JOGADOR, NOME, DT_NASCIMENTO, SALARIO, ID_TIME)
VALUES (SEQ_JOGADOR.NEXTVAL, 'RIVELINO', '09/06/1950', 200000, 4);
COMMIT;

-- Alterar o salario de 3 jogadores para valores acima de R$ 100.000,00
UPDATE JOGADOR
SET SALARIO = 120000
WHERE ID_JOGADOR = 6;

UPDATE JOGADOR
SET SALARIO = 110000
WHERE ID_JOGADOR = 34;

UPDATE JOGADOR
SET SALARIO = 115000
WHERE ID_JOGADOR = 35;
COMMIT;

-- Remover jogadores do novo time com salário superior a R$ 100.000,00
DELETE
FROM JOGADOR
WHERE SALARIO > 100000 AND ID_TIME = 4;
COMMIT;

```

## 4.

```

-- Selecione os nome dos times por ordem crescente
--
select nome from time order by nome asc;

-- Selecione os nomes de jogadores e seus respectivos nomes de times ordenado(asc) pela data de nascimento
dos jogadores.

select j.nome,
       t.nome
from   jogador j,
       time t
where  j.id_time = t.id_time
order by j.dt_nascimento;

-- Qual é a folha de pagamento de cada time?

select t.id_time,
       t.nome,
       sum(j.salario) as salario_total
from   time t,
       jogador j
where  t.id_time = j.id_time
group by t.id_time,
         t.nome;

-- faça a média de salário/ jogador de cada time
select t.id_time,
       t.nome,
       avg(j.salario) as media_salario
from   time t,
       jogador j
where  t.id_time = j.id_time
group by t.id_time,
         t.nome;

-- Faça uma consulta que retorne o menor salário de cada time.
select t.id_time,
       t.nome,

```

```

        min(j.salario) as salario
from   time    t,
        jogador j
where  t.id_time = j.id_time
group  by t.id_time,
        t.nome;
-- faça uma consulta que retorne o maior salário de cada time.
select t.id_time,
        t.nome,
        max(j.salario) as salario
from   time    t,
        jogador j
where  t.id_time = j.id_time
group  by t.id_time,
        t.nome;

```

## 5.

```

--Gere uma saída DBMS básica ('Hello DBMS') utilizando o pacote da Oracle DBMS_OUTPUT.PUTLINE('text').
DECLARE
BEGIN
    DBMS_OUTPUT.PUT_LINE('HELLO DBMS');
END;

--Gerar uma saída DBMS contendo as informações de um time e do seu técnico.
DECLARE
    vsNomeTime    varchar2(200);
    vsNomeTecnico varchar2(200);
BEGIN
    SELECT TEC.NOME,
           TIM.NOME
    into   vsNomeTecnico,
           vsNomeTime
    FROM   TECNICO TEC,
           TIME    TIM
    WHERE  TEC.ID_TIME = TIM.ID_TIME AND
           TEC.ID_TECNICO = 2;
    DBMS_OUTPUT.PUT_LINE('Tecnico ' || vsNomeTecnico || ', Time ' ||
                          vsNomeTime);
END;

--Gerar um jogo composto de dois times diferentes e escalar os jogadores participantes.
INSERT into jogo
(id_jogo,
 id_time_a,
 id_time_b,
 nr_gol_a,
 nr_gol_b,
 dh_inicio,
 dh_fim)
values
(1,
 2,
 3,
 0,
 0,
 sysdate,
 sysdate);

-- Criação Sequence
CREATE SEQUENCE SEQ_JOGO_JOGADOR MINVALUE 1 START
WITH 1 INCREMENT BY 1 NOCACHE;

insert into jogo_jogador
(id_jogo_jogador,
 id_jogo,

```

```

        id_jogador)
select SEQ_JOGO_JOGADOR.NEXTVAL,
       1,
       jog.id_jogador
from   jogador jog
where  jog.id_time in (2,
                      3);

--Marcar alguns gols para o jogo gerado respeitando o placar definido no jogo.
insert into gol
(id_gol,
 dh_gol,
 id_jogo_jogador)
values
(1,
 sysdate,
 1);
update jogo set nr_gol_a = 1 where id_jogo = 1;

--Incluir comentários nos blocos anônimos anteriores sem alterar o funcionamento.
DECLARE
BEGIN
    /*Comment*/
    DBMS_OUTPUT.PUT_LINE('HELLO DBMS');
END;

--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".
declare
    vnTimeA number := 2;
    vnTimeB number := 3;
begin
    if vnTimeA <> vnTimeB THEN
        INSERT into jogo
        (id_jogo,
         id_time_a,
         id_time_b,
         nr_gol_a,
         nr_gol_b,
         dh_inicio,
         dh_fim)
        values
        (1,
         vnTimeA,
         vnTimeB,
         0,
         0,
         sysdate,
         sysdate);
    END IF;
end;

--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'.
select jogo.id jogo,
       time_a.nome as time_A,
       decode(jogo.nr_gol_a,
              jogo.nr_gol_b,
              'EMPATE',
              jogo.nr_gol_a || ' - ' || jogo.nr_gol_b) AS PLACAR,
       time_b.nome as TIME_B
from   jogo,
       time time_a,
       time time_b
where  jogo.id_time_a = time_a.id_time and
       jogo.id_time_b = time_b.id_time;

--Criar bloco anônimo com algum comando dml "insert" que irá causar erro, e tratar com a exception OTHERS.

```



```

    psDescricao VARCHAR2
) AS
--variaveis
BEGIN
    insert into cartao(id_cartao,descricao) values (SEQ_CARTAO.nextval,psDescricao);
END;
--Criar uma function que retorne a quantidade de gols de um jogador específico, sendo este o
parâmetro de entrada.

CREATE OR REPLACE FUNCTION OBTEM_NR_GOLS(pIdJogador number) return number is
    vnNumeroDeGol number;
BEGIN
    select count(*)
    into    vnNumeroDeGol
    from    jogo_jogador jj,
           gol g
    where   jj.id_jogador = pIdJogador and
           jj.id_jogo_jogador = g.id_jogo_jogador;

    return vnNumeroDeGol;

END;
--Criar uma consulta que retorne os times, os jogadores do time e a quantidade de gols de cada jogar
(utilizar function criada).
select jog.nome as NOME_JGOADOR,
       T.NOME as NOME_TIME,
       OBTEM_NR_GOLS(jog.id_jogador)
from   jogo_jogador jj,
       jogador      jog,
       time         t
where  jj.id_jogador = jog.id_jogador and
       jog.id_time = t.id_time;

```

## 7.

--Criar uma consulta que retorne somente os cartões que já foram aplicados em algum jogo.

```

select *
from cartao cart
where exists ( select * from jogo_cartao_jogador jcj where jcj.cartao_id_cartao = cart.id_cartao )

```

--Reduzir em 10% o salário de todos os jogadores que receberam pelo menos um cartão

```

update jogador jog set salario = salario*0.9 where exists(select * from jogo_jogador jj where
jj.jogador_id_jogador=jog.id_jogador and exist(select * from jogo_cartao_jogador where
jcj.jogo_jogador_id_jogo_jogador=jj.id_jogo_jogador))

```

## 8.

--Criar uma trigger que depois de inserir algum gol ele contabilize mais um ao jogo para aquele determinado time.

```

CREATE OR REPLACE TRIGGER tgGol
AFTER INSERT OR DELETE ON gol
FOR EACH ROW
declare
    vnIdJogo number;
    vnIdTime number;
    vnTimeA number;
    vnIdJogoJogador number;
begin
    vnIdJogoJogador:= nvl(:new.id_jogo_jogador,:old.id_jogo_jogador);
    -- busco o id do jogo , e o id do time
    select jj.id_jogo,

```

```

into      jog.id_time
into      vnIdJogo,
          vnIdTime
from      jogo_jogador jj,
          jogador      jog
where     jj.id_jogo_jogador = vnIdJogoJogador and
          jj.id_jogador = jog.id_jogador;

-- verifico se existe o time é o time A
select count(*)
into      vnTimeA
from      jogo
where     jogo.id_jogo = vnIdJogo and
          jogo.id_time_a = vnIdTime;

-- incrementa um gol para o time que fez o gol
if nvl(vnTimeA,0) > 0 then

    update jogo
    set     (nr_gol_a) =
            (nr_gol_a + 1)
    where   id_jogo = vnIdJogo;

else

    update jogo
    set     (nr_gol_b) =
            (nr_gol_b + 1)
    where   id_jogo = vnIdJogo;

end if;

end;

--Alterar a trigger criada anteriormente para que ela subtraia um gol do jogo caso o registro do gol tenha
sido apagado.
CREATE OR REPLACE TRIGGER tgGol
AFTER INSERT OR DELETE ON gol
FOR EACH ROW
declare
vnIdJogo      number;
vnIdTime      number;
vnTimeA       number;
vnIdJogoJogador number;
begin

vnIdJogoJogador := nvl(:new.id_jogo_jogador,:old.id_jogo_jogador);

-- busco o id do jogo , e o id do time
select jj.id_jogo,
       jog.id_time
into    vnIdJogo,
       vnIdTime
from    jogo_jogador jj,
       jogador      jog
where   jj.id_jogo_jogador = vnIdJogoJogador and
       jj.id_jogador = jog.id_jogador;

-- verifico se existe o time é o time A
select count(*)
into      vnTimeA
from      jogo
where     jogo.id_jogo = vnIdJogo and
          jogo.id_time_a = vnIdTime;

-- se é time A estamos inserindo incrementa um gol para o time A
if nvl(vnTimeA,0) > 0 AND INSERTING then

    update jogo

```



```

set      (nr_gol_a) =
          (nr_gol_a + 1)
where    id_jogo = vnIdJogo;

-- se é time A estamos deletando decrementa um gol para o time A
elsif nvl(vnTimeA,0) > 0 AND DELETING then

  update jogo
  set      (nr_gol_a) =
            (nr_gol_a - 1)
  where    id_jogo = vnIdJogo;

-- se é time B estamos inserindo incrementa um gol para o time B
elsif nvl(vnTimeA,0) = 0 AND INSERTING then

  update jogo
  set      (nr_gol_b) =
            (nr_gol_B + 1)
  where    id_jogo = vnIdJogo;

-- senao decremento um gol para o time B
else

  update jogo
  set      (nr_gol_b) =
            (nr_gol_B - 1)
  where    id_jogo = vnIdJogo;

end if;

end;

--Criar uma tabela de log de jogador para que armazene todos os dados antes de qualquer alteração.
create table JOGADOR_LOG
( id_jogador_log  number,
  id_jogador      NUMBER ,
  nome            VARCHAR2(400),
  dt_nascimento   DATE ,
  salario         NUMBER(18,2),
  id_time         NUMBER
);
alter table JOGADOR_LOG  add constraint PK_JOGADOR_log primary key (id_jogador_log);

-- Criação Sequence
CREATE SEQUENCE SEQ_JOGADOR_LOG
MINVALUE 1
START WITH 1
INCREMENT BY 1
NOCACHE;
--Criar uma trigger que antes de realizar qualquer alteração em um jogador ele guarde no log o estado
anterior.
CREATE OR REPLACE TRIGGER tgJogadorLog
AFTER UPDATE  ON jogador
FOR EACH ROW
declare

begin

  insert into jogador_log
  (id_jogador_log,
   id_jogador,
   nome,
   dt_nascimento,
   salario,
   id_time)
  values
  (seq_jogador_log.nextval,
   :old.id_jogador,
   :old.nome,

```

```
:old.dt_nascimento,  
:old.salario,  
:old.id_time);  
  
end;
```