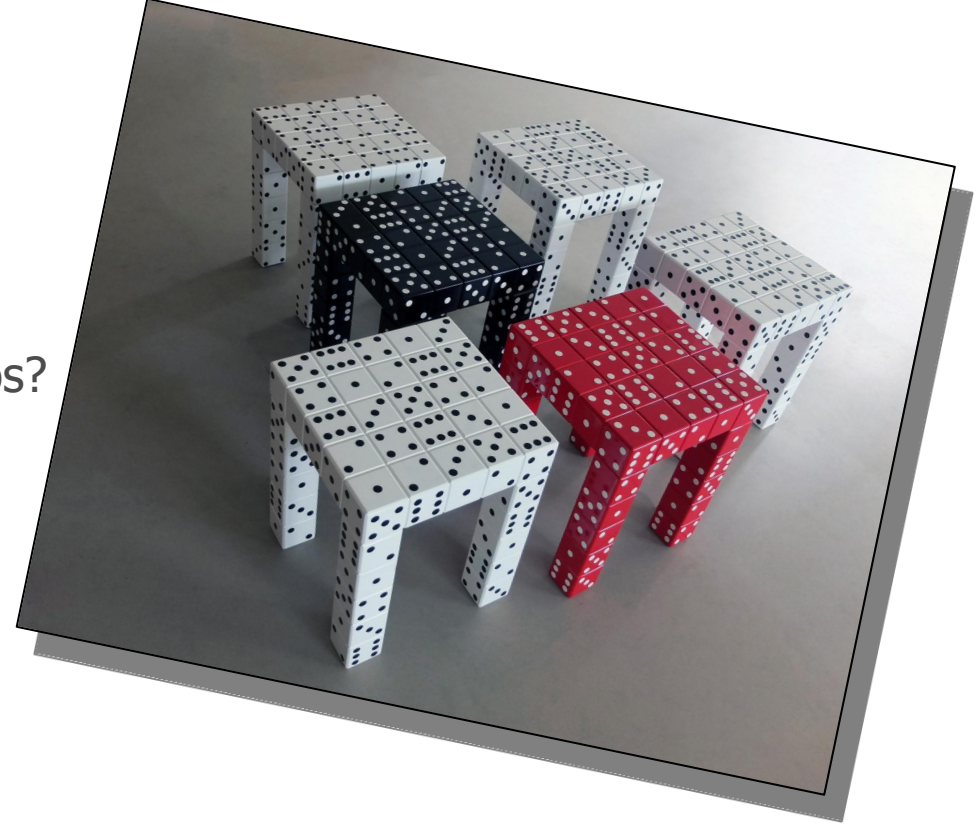
Curso de Férias
SQL e PL/SQL Básico

MATERA systems

1. Introdução a banco de dados.
2. Comandos DDL
3. Comandos DML.
4. Comandos DML II.
5. PLSQL.
6. Procedures e Functions.
7. SubQuerys.
8. Mais Objetos Oracle.
9. Packages.

Introdução a banco de dados.

- O que é um banco de dados?
- Porque precisamos de um banco de dados?
- O que pode ser um banco de dados?



MATERA systems

Introdução a banco de dados.

- O que é um SGBD?
- Tipos de bancos de dados.
- O que é SQL (Structured Query Language).

MATERA systems

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.

Comandos DDL

Tipos de dados.

Cada valor manipulado pelo Oracle Database possui um tipo de dados.

Tipo	Descrição
VARCHAR2(comprimento_máximo)	Carácter de tamanho variável, podendo atingir o tamanho máximo de até 32767 bytes.
NUMBER [precisão, escala]	Tipo numérico fixo e de ponto flutuante.
DATE	Tipo para acomodar data e hora

Comandos DDL

- Criação de tabelas

Os dados são armazenados em estruturas chamadas tabelas, abaixo é apresentada a composição do comando create table.

```
create table time
(
      id_time     number          not null,
      nome        varchar2(400) not null
);
```


MATERA systems

Comandos DDL

- Constraints

Constraints são objetos 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.

Comandos DDL

Primary key

- Cada tabela pode ter, no máximo, uma constraint de primary key (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.

```
alter table TIME add constraint pk_time primary key (ID_TIME);
```

Comandos DDL

Foreign Key

- A foreign key (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.

```
alter table JOGADOR add constraint fk_time foreign key (id_time) references time(id_time);
```

Comandos DDL

- Comentários

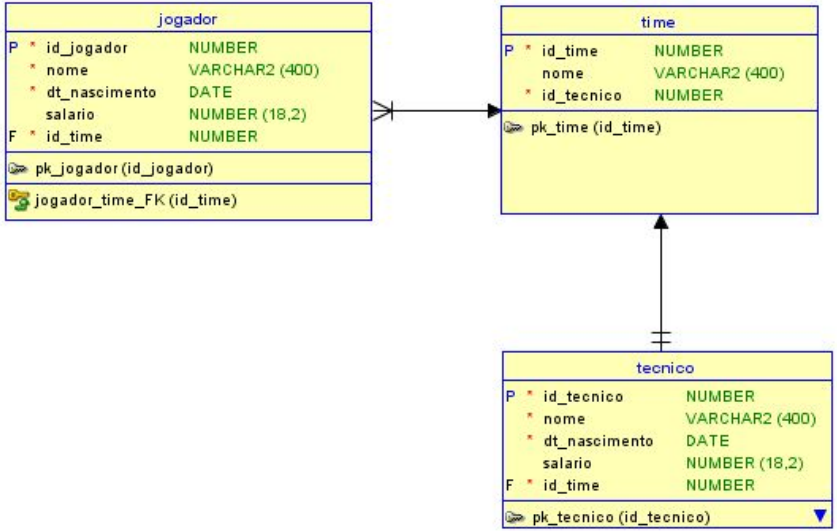
Ao criar uma tabela, é possível definir comentários para a tabela e colunas, isso auxilia no entendimento do objetivo da tabela e colunas.

```
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.';
```

Comandos DDL

- Exercícios:
  - Criar as tabelas time, técnico e jogador;
  - Definir constraints para as tabelas;
  - Criar comentários para as tabelas e as colunas;



MATERA systems

Comandos DML

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

- Manipulações como Inserção, alteração e deleção de registros.

MATERA systems

Comandos DML

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

```
Select      id_time, nome
from        time
where       nome = 'BARCELONA'
order by nome;
```

Comandos DML

- Comando ***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.

```
insert into time (id_time, nome) values (1,'BARCELONA');
--
commit;
--
select nome
from   time;
```

Comandos DML

- 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*.

```
delete time;
--
rollback;
--
select nome
from   time;
```

Comandos DML

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

```
insert into time (id_time, nome)
values (1,'BARCELONA');
```

```
insert into time
values (1,'BARCELONA');
```

Comandos DML

SEQUENCE.

Sequences é um tipo de objeto utilizado para incrementar valores. A Oracle disponibiliza este objeto para facilitar o controle dos valores já que, é controlado pelo db e você só precisa consumi-lo.

```
create sequence seq_jogador
minvalue 1
maxvalue 999999999999999999999999999
start with 1
increment by 1
nocache;
```

CACHE n / NOCACHE :
especifica quantos valores são pré-alocados pelo Oracle Server e mantidos na memória (por padrão, o Oracle Server armazena 20 valores em cache).

MATERA systems

Comandos DML

SEQUENCE.
Este objeto possui duas funções expostas já pré-definidas.

nextval que incrementa e retorna o próximo valor.
currval que retorna o último valor gerado.

```
insert into time (id_time, nome)
values (seq_jogador.nextval,'BARCELONA');
```

```
select seq_jogador.currval
from dual;
```

Comandos DML

- 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).

Comandos DML

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

```
update time
set nome = 'BARCELONA FUTEBOL'
where id_time = 1;
```

```
update time
set nome = 'BARCELONA FUTEBOL ALTERADO'
where nome = 'BARCELONA FUTEBOL';
```

Comandos DML

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

```
update time
set nome = 'BARCELONA' || 'FUTEBOL'
where id_time = 1;
```

```
update time
set nome = nome || 'FUTEBOL'
where id_time = 1;
```

Comandos DML

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

```
update time
set nome = 'BARCELONA' ,
segundo_nome = 'SEGUNDO NOME DO TIME'
where id_time = 1;
```

Comandos DML

- 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%.

MATERA systems

Comandos DML

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

```
delete time
where id_time = 1;
```

```
delete jogador
where salario>= 100000;
```

Comandos DML

- 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.

MATERA systems

Comandos DML II

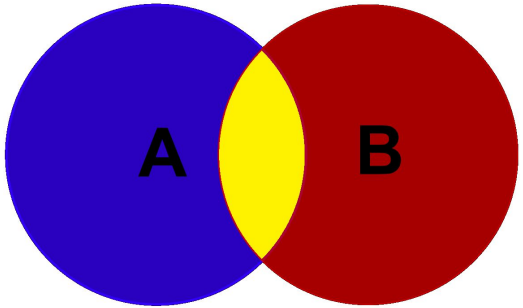
- Junções de Dados e Apelidos
  - INNER JOIN
  - LEFT JOIN
  - RIGHT JOIN

Comandos DML II | Inner Join

Quando queremos juntar duas ou mais tabelas, que internamente, tenham valores correspondentes (parte amarela).

SQL INNER JOIN

```
SELECT JOG.NOME NOME,
       EQU.NOME AS NOME_DA_EQUIPE
FROM   JOGADOR JOG INNER JOIN EQUIPE EQU
                   ON JOG.ID_EQUIPE = EQU.ID_EQUIPE;
```



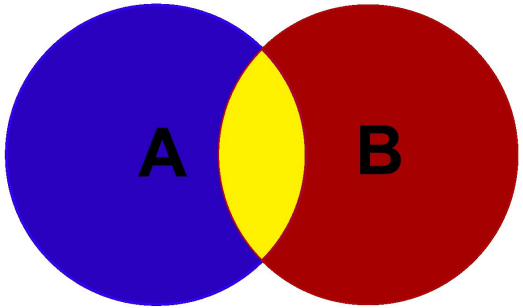
MATERA systems

Comandos DML II | Inner Join

Quando queremos juntar duas ou mais tabelas, que internamente, tenham valores correspondentes (parte amarela).

SQL JOIN

```
SELECT JOG.NOME NOME,
       EQU.NOME AS NOME_DA_EQUIPE
FROM   JOGADOR JOG JOIN EQUIPE EQU
                   ON JOG.ID_EQUIPE = EQU.ID_EQUIPE;
```



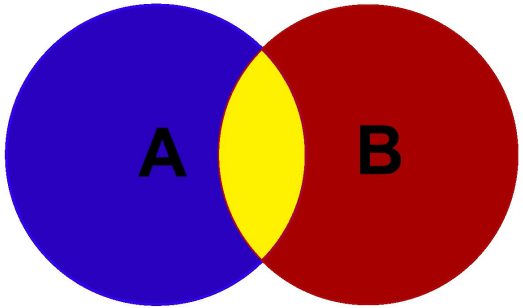
MATERA systems

Comandos DML II | Inner Join

Quando queremos juntar duas ou mais tabelas, que internamente, tenham valores correspondentes (parte amarela).

PL/SQL JOIN

```
SELECT JOG.NOME NOME,
       EQU.NOME AS NOME_DA_EQUIPE
FROM   JOGADOR JOG,  EQUIPE    EQU
WHERE JOG.ID_EQUIPE = EQU.ID_EQUIPE;
```



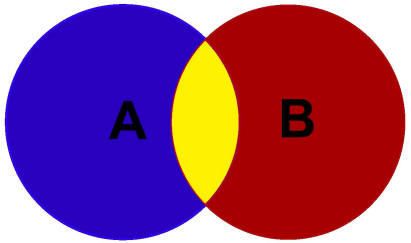
MATERA systems

Comandos DML II | Left Join

- **É utilizado para selecionar todos os itens de uma tabela A com uma tabela B mesmo que A não esteja relacionado com a tabela B (Parte Azul).**

```
SELECT JOG.NOME NOME, EQU.NOME AS NOME_DA_EQUIPE
FROM  JOGADOR JOG LEFT JOIN EQUIPE EQU
                          ON JOG.ID_EQUIPE = EQU.ID_EQUIPE;
```

```
SELECT JOG.NOME    NOME,
          EQU.NOME AS NOME_DA_EQUIPE
FROM   JOGADOR JOG, EQUIPE    EQU
WHERE JOG.ID_EQUIPE = EQU.ID_EQUIPE(+);
```



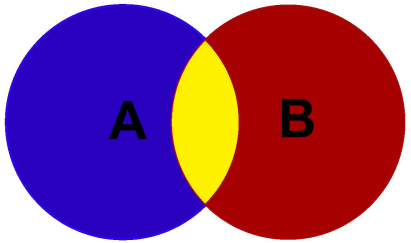
MATERA systems

Comandos DML II | Right Join

Funciona como o left outer join, mas ao contrário. (Parte Vermelha).

```
SELECT JOG.NOME NOME, EQU.NOME AS NOME_DA_EQUIPE
FROM  JOGADOR JOG RIGHT JOIN EQUIPE EQU
                                ON JOG.ID_EQUIPE = EQU.ID_EQUIPE;
```

```
SELECT JOG.NOME    NOME,
          EQU.NOME AS NOME_DA_EQUIPE
FROM   JOGADOR JOG, EQUIPE    EQU
WHERE JOG.ID_EQUIPE(+) = EQU.ID_EQUIPE;
```



MATERA systems

Comandos DML II | Ordenações

Utilizamos as ordenações para ordenar os resultados de uma consulta.
Podemos ordenar por ordem crescente(*asc*) ou decrescente(***desc***).

```
SELECT NOME, DT_NASCIMENTO AS DATA_NASC
FROM JOGADOR
ORDER BY NOME ASC
```

```
SELECT NOME, DT_NASCIMENTO AS DATA_NASC
FROM JOGADOR
ORDER BY NOME, DT_NASCIMENTO ASC
```

Comandos DML II | Ordenações

Utilizamos as ordenações para ordenar os resultados de uma consulta.
Podemos ordenar por ordem crescente(*asc*) ou decrescente(***desc***).

```
SELECT NOME, DT_NASCIMENTO AS DATA_NASC
FROM JOGADOR
ORDER BY 2, 1 ASC
```

```
SELECT NOME, DT_NASCIMENTO AS DATA_NASC
FROM JOGADOR
ORDER BY DATA_NASC ASC
```

Comandos DML II | Ordenações

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

MATERA systems

Comandos DML II | Agrupamentos de Dados

Utilizamos agrupamento para juntar os dados equivalentes com a palavra group by.

- Com a utilização e grupos podemos utilizar as funções de agregação, que permitem realizar cálculos sobre o resultado da consulta retornada.
- Todas colunas selecionadas que não estão sendo utilizadas em algum tipo de função de agregação deverão estar declaradas no “group by”.

```
SELECT T.NOME
FROM   JOGADOR J, TIME T
WHERE  T.ID_TIME = J.TIME_ID_TIME
GROUP  BY T.NOME
```

Comandos DML II | Funções de Agregação

As **Funções de Agregação** são utilizadas para manipular os dados agrupados.

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

```
SELECT COUNT(*), T.NOME
FROM   JOGADOR AS J,
       TIME    AS T
WHERE  T.ID_TIME = J.TIME_ID_TIME
GROUP  BY T.NOME
```

Comandos DML II | Funções de Agregação

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

MATERA systems

PLSQL

- O que é PL/SQL ?
- As vantagens do PL/SQL
- Diferenças da Sintaxe SQL e PLSql

PLSQL | 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.**

```
1  declare
2    -- Local variables here
3    i integer;
4  begin
5    -- Test statements here
6  end;
```

PLSQL | Blocos Anônimos

- Composição de um bloco Anônimo.
  - Declarativa (opcional).
  - Executável (obrigatória).
  - Manipulação de Exceções e Erros (opcional).
  - Finalização do bloco.

```
declare
  --
  vnNumber number;
  --
begin
  --
  null;
  --
exception
  when other then
    --
    dbms_output.put_line('error');
    --
end;
```

PLSQL | Blocos Anonimos

- Exercícios:
  - Criar Blocos Anônimos que:
    - Gere uma saída DBMS básica (‘Hello DBMS’) utilizando o pacote da Oracle DBMS_OUTPUT.PUT_LINE(‘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.

MATERA systems

PLSQL | Comentários

```
--line comments.

/* block comments */

/*

  block comments
  here

*/
```

PLSQL | Comentários

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

MATERA systems

PLSQL | Desvios Condicionais

```
IF (condição) THEN

   /* comandos aqui */

END IF;
```

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

PLSQL | Desvios Condicionais

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

PLSQL | Desvios Condicionais

```
declare
  vnNumero  number(1) := 1;
  vNRetorno number;

begin

  vNRetorno := case
                 when vnNumero = 1 then
                   11
                 when vnNumero = 2 then
                   22
                 else
                   33
               end;

  dbms_output.put_line(vNRetorno);

end;
```

PLSQL | Desvios Condicionais

```
select decode(nome, null,
                    'Técnico sem nome',
                    'chuck norris',
                    ' !!!!!! ',
                    nome)
from tecnico;
```

PLSQL | Desvios Condicionais

- Exercícios:
  - Criar bloco anônimo que gera um jogo composto de dois times diferentes e e validar para que não permita inserir um jogo sendo o mesmo time para “ambos os lados” (“if”).

  - 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’.

MATERA systems

PLSQL | Exceptions

Quando um bloco **PL/SQL** é executado ele tem um fluxo de vida, com execução TOP-DOWN porém com desvios de fluxos,
durante a execução pode ocorrer algum erro ou anormalidade.. então uma exceção é levantada (raise exception) e o fluxo normal do programa é desviado para o parágrafo declarado na área de *EXCEPTION (caso tenha sido declarado)*.

Dentro da tratativa EXCEPTION, podemos tratar a exceção da melhor maneira possível.

- Caso uma exceção não seja "tratada" a execução da rotina ou procedimento termina em falha abortando todo processo!

PLSQL | Exceptions

Exceptions mais utilizadas:

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

```
BEGIN

  --codigoQuePodeHaverExcessoes;

EXCEPTION
  WHEN OTHERS THEN

    DBMS_OUTPUT.PUT_LINE('Erro ao executar operacoes. Erro :' || SQLERRM);

END;
```

PLSQL | Exceptions

- 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.

MATERA systems

PLSQL | Cursores/Loops

Loops comumente são laços de repetições sendo a base de conhecimento para o entendimento de manipulação de cursores.

```
Loop
  --
  dbms_output.put_line(vnContador);
  --
  vnContador := vnContador + 1;
  --
  exit when vnContador = 20;
  --
end loop;
```

PLSQL | Cursores

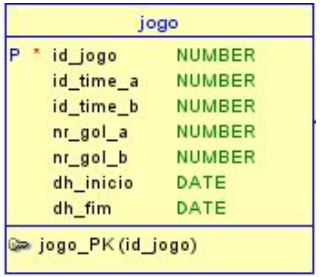
```
while vnContador < 10 loop
  --
  dbms_output.put_line(vnContador);
  --
  vnContador := vnContador + 1;
  --
end loop;
```

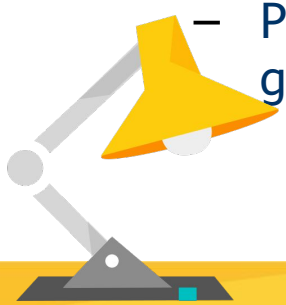
PLSQL | Cursores

```
Declare
  --
  cursor vcCursor is
    select *
    from time;
    --
Begin
  --
  For meuCursor in vcCursor loop
    --
    dbms_output.put_line(meuCursor.nome);
    --
  end loop;
  --
end;
```

PLSQL | Cursores

- 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.
  - Criar uma tabela de jogo e uma que marque os os gols do jogo.
  - Percorrer todos os jogos e inserir um gol para cada jogador.





MATERA systems

Procedures & 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.

Procedures & Functions

```
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
  --
BEGIN
  --
  null;
  --
END;
```

Procedures & Functions

```
CREATE OR REPLACE FUNCTION NOME_DA_FUNCTION
(
  PARAM_ENTRADA    NUMBER,
  PARAM_ENTRADA2 VARCHAR2
) RETURN VARCHAR2 AS
  --
BEGIN
  --
  return ' retorno ';
  --
end;
```

Procedures & Functions

- Exercícios:
  - Criar uma procedure insere cartões, recebendo somente a descrição do cartão.
  - Criar uma function que retorna 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 jogador (utilizar function criada).

cartao		
P	* id_cartao	NUMBER
	descricao	VARCHAR2 (400)
pk_cartao (id_cartao)		

jogo_jogador		
P	* id_jogo_jogador	NUMBER
F	* id_jogo	NUMBER
F	* id_jogador	NUMBER
pk_jogo_jogador (id_jogo_jogador)		
jogo_jogador_jogo_FK (id_jogo)		
jogo_jogador_jogador_FK (id_jogador)		

MATERA systems

SubQuerys

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

Selecionar todos os times que não possuem jogadores.

```
select *
from   time t
where not exists (select *
                  from    jogador jog
                  where  jog.id_time = t.id_time );
```

SubQuerys

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

Selecionar todos os times que possuem jogadores.

```
select *
from   time t
where exists (select *
                from    jogador jog
                where  jog.id_time = t.id_time );
```

SubQuerys

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

Apago todos os 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 );
```

SubQuerys

- 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.

MATERA systems

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.

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

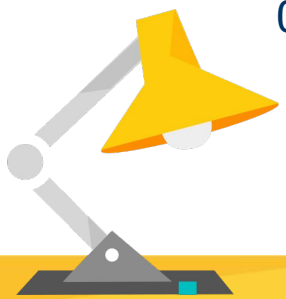
MATERA systems

Triggers

Teoricamente existem dois tipos de **Trigger** no Oracle a de linha e a de instrução.

- Trigger de linha > É executada no nível de linha ou seja executado uma vez para cada comando alteração na linha (**row level trigger**).
  - Ex: Caso você realize um UPDATE de 10 linhas, a trigger será executada 10 vezes.

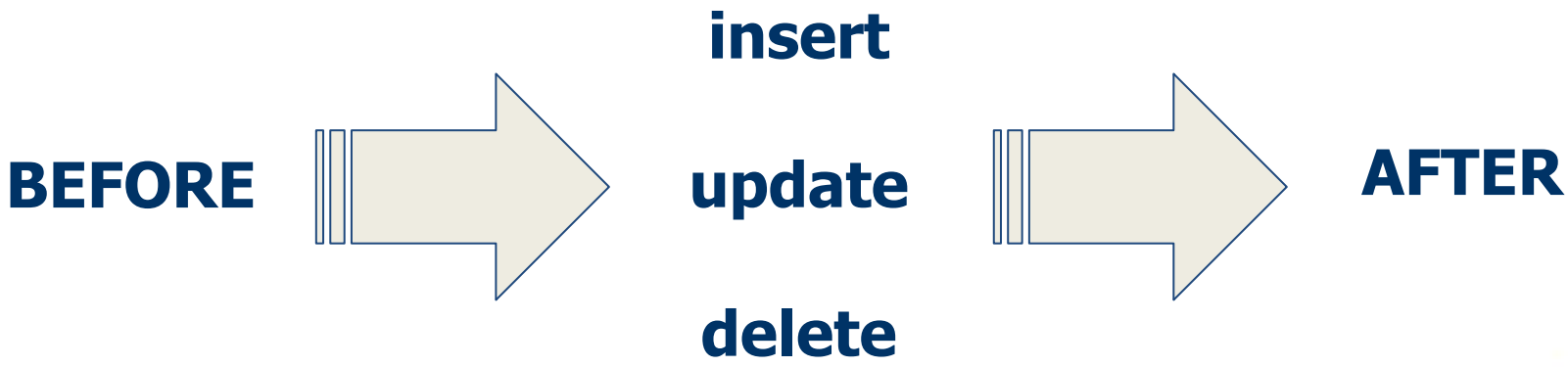
- Trigger de instrução > É executado a nível de instrução DML ou seja é executado apenas uma vez por comando DML(**statement level trigger**).
  - Ex: Caso você realize um DELETE de 100 linhas, a trigger será executada apenas 01 vez.



MATERA systems

Triggers

- As triggers possuem dois tipos eventos de disparo, havendo a possibilidade de utilizar sob qualquer evento de *manipulação de Dados*.
  - before > antes da manipulação.
  - after > depois da manipulação.



Triggers

- **Eventos temporais (Row level trigger)**

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.

MATERA systems

Triggers

- **Eventos temporais (Row level trigger)**

Estrutura básica de uma trigger.

```
create or replace trigger nome_da_trigger
  after insert or update or delete on tabela_com_gatilho
  for each row

begin

  --codigo lógico aqui;

end;
```

MATERA systems

Triggers

- **Eventos temporais (Row level trigger)**

Ex:

```
create or replace TRIGGER TG_EQUIPE_01
  BEFORE INSERT OR UPDATE OR DELETE ON EQUIPE
  FOR EACH ROW
  --
DECLARE
  vsNome EQUIPE.NOME%TYPE;
BEGIN

  IF INSERTING THEN

    :NEW.NOME := :NEW.NOME || ' (NEW)';

  ELSIF UPDATING AND :NEW.NOME IS NOT NULL THEN

    :NEW.NOME := :NEW.NOME || ' (UPDATED)';
```

■■■

```
  ELSIF UPDATING AND :NEW.NOME IS NOT NULL THEN

    :NEW.NOME := :NEW.NOME || ' (UPDATED)';

  ELSIF UPDATING AND :NEW.NOME IS NULL THEN

    :NEW.NOME := 'NULL (UPDATED)';

  ELSE

    vsNome := :OLD.NOME;

    INSERT INTO EQUIPE_LOG (ID_EQUIPE_LOG,
NOME)
    VALUES              (SEQ_EQUIP_LOG.NEXTVAL, vsNome);

  END IF;

END;
```


MATERA systems

Triggers

- Exercícios:
  - Criar uma trigger que *depois* de inserir algum gol ele contabiliza na tabela que guarda o placar.
  - Alterar a trigger criada anteriormente para que ela subtrai 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.

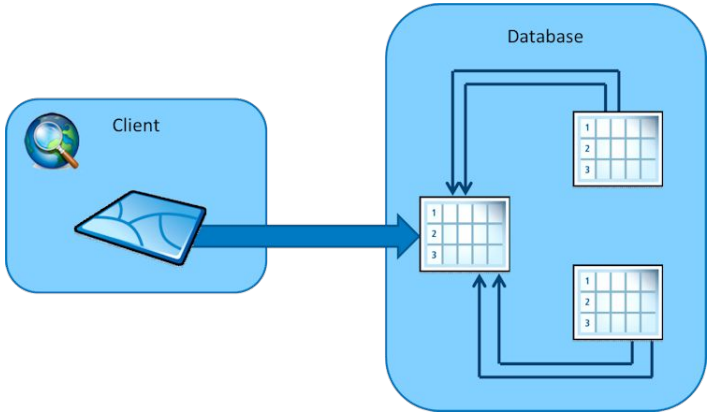
MATERA systems

Views

Views 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.

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



MATERA systems

Views

```
create or replace force view vEquipe as
select id_equipe,
       descricao
from   equipe;
```

```
select id_equipe, descricao
from   vEquipe;
```

```
select id_equipe, descricao
from   equipe;
```


MATERA systems

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.

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, exceptions e outros 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 com comportamento distintos em a partir da sua chamada.

Packages

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

Packages

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

Packages

```
create or replace package body campeonato is
  /* variáveis globais
      cursors globais
      procedures publicas
      functions publicas
  */
   procedure adicionarJogador(pisNome varchar2) is;
   --
   Begin
     --codigo que insere o jogador
     Null;
     --
   End;
--
end;
```

Contato

www.matera.com

Altieres de Matos
altieres.matos@matera.com
altitdb@gmail.com
Linkedin: https://goo.gl/kb3fyq

Junior Miqueletti
junior.miqueletti@matera.com
juniormiqueletti@gmail.com
Linkedin: https://goo.gl/ryjTG1

Thank you :D

MATERA systems