Curso de Férias
SQL e PL/SQL Básico

MATERA
systems

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

MATERA systems

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
);
```

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. Existem alguns tipos de constraints no Oracle, a seguir elas são apresentadas:

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 indice único, caso ainda não exista para a coluna em causa.

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.

```
alter table TIME add constraint pk_time primary key (ID_TIME);
alter table jogador add constraint fk_time foreign key (id_time) references time(id_time);
```

Comandos DDL

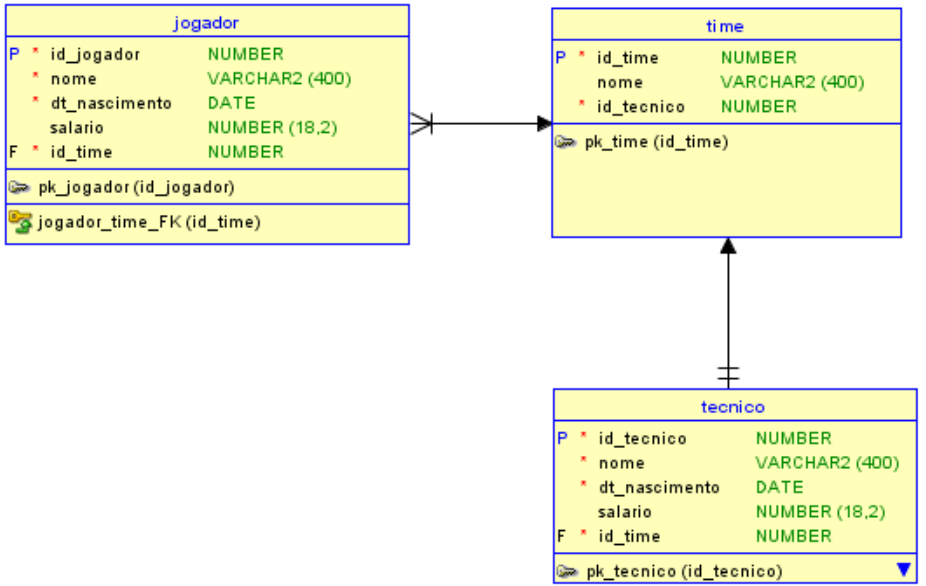
- Comentários

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 (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.

Comandos DML

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

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

SEQUENCE.

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

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

/*exemplo de utilização de sequences*/
select seq_jogador.nextval 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'
```

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

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

```
update time set nome = nome || ' COMPLEMENTO'
where id_time = 1
```

```
update time set nome = 'BARCELONA FUTEBOL', SEGUNDO_NOME = 'SEM SEGUNDO NOME'
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%.

Comandos DML

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

```
delete times where id_time = 1

delete jogador where salario = 10000
```

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.

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).

/ PLSQL JOIN */*

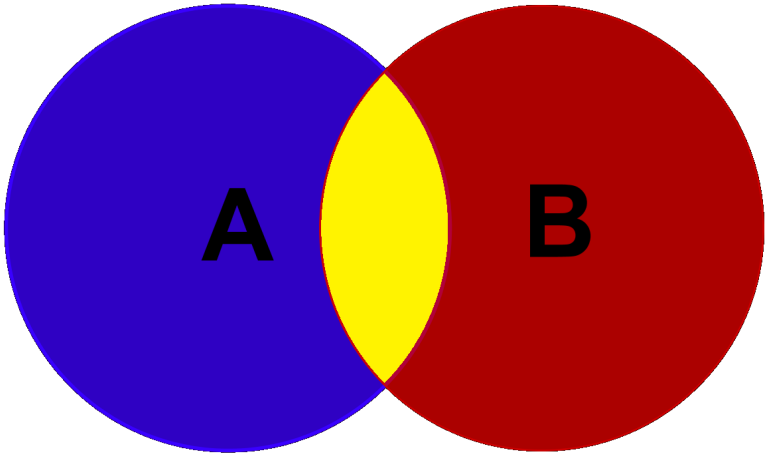
```
SELECT JOG.NOME     NOME,
       EQU.NOME AS NOME_DA_EQUIPE
FROM   JOGADOR JOG, EQUIPE    EQU
WHERE  JOG.ID_EQUIPE = EQU.ID_EQUIPE;
```

/ 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;
```

/ SQL JOIN (ANSI)*/*

```
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 | Left Join

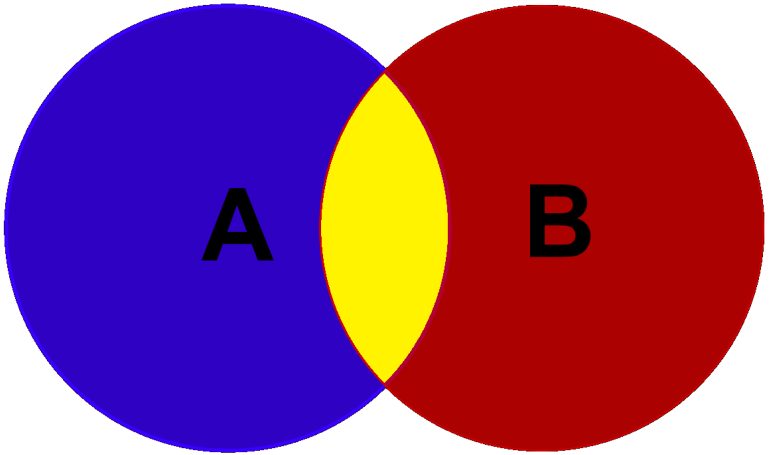
Serve para selecionar todos os itens de uma tabela A com uma tabela B mesmo que A não esteja relacionado com a tabela B (Parte Azul).

/ PLSQL LEFT JOIN */*

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

/ SQL JOIN (ANSI)*/*

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



MATERA systems

Comandos DML II | Right Join

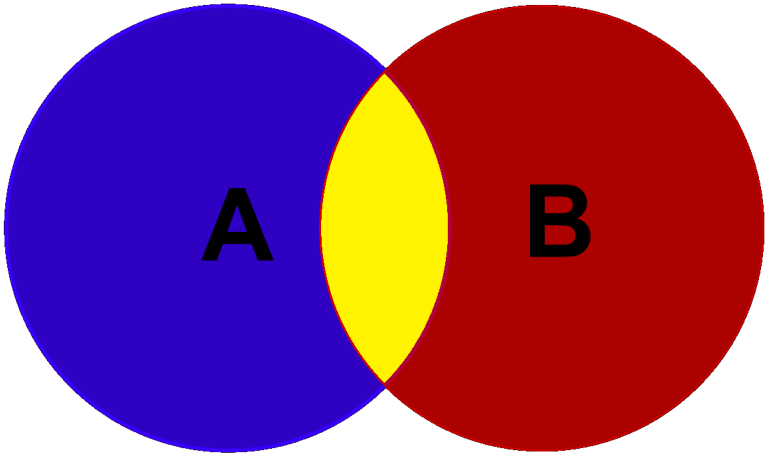
Funciona como o left outer join, mas ao contrário. (Parte Vermelha).

/ PLSQL LEFT JOIN */*

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

/ SQL JOIN (ANSI)*/*

```
SELECT JOG.NOME     NOME,
       EQU.NOME AS NOME_DA_EQUIPE
FROM   JOGADOR JOG RIGHT JOIN EQUIPE EQU ON 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, data_nascimento as data_nasc from jogador order by nome asc

select nome, data_nascimento as data_nasc from jogador order by nome, data_nascimento asc

select nome, data_nascimento as data_nasc from jogador order by 2, 1 desc

select nome, data_nascimento as data_nasc from jogador order by data_nasc desc
```

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.

Comandos DML II | Agrupamentos de Dados

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

- Com a utilização e grupos podemos utilizas 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 as j,
       time    as 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.

PLSQL

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

PLSQL | Blocos Anonimos

- **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).**

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

PLSQL | Blocos Anonimos

- **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.**

PLSQL | Comentários

```
--line comments.

/* block comments */

/*

block comments
here

*/
```

PLSQL | Comentários

Exercícios:

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

PLSQL | Desvios Condicionais

```
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;
```

```
eclare
  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
```


MATERA systems

PLSQL | Desvios Condicionais

- 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’.

PLSQL | 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.

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.

PLSQL | Cursores

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

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

PLSQL | Cursores

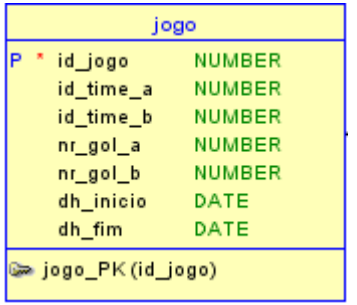
```
while vnContador < 10 loop
  --imprimo via dbms a variável vnContador
  dbms_output.put_line(vnContador);
  --incremento a variável vnContador
  vnContador := vnContador + 1;
end loop;
```

PLSQL | Cursores

```
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 d
               o cursor que estou percorrendo
            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

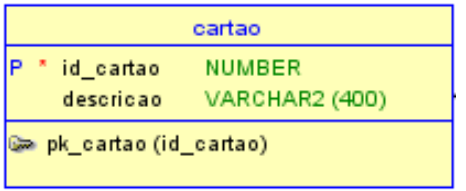
- Stored procedures, e stored functions, são objetos do banco de dados Oracle, compostos de códigos PL/SQL e SQL, logicamente agrupados, com a finalidade de executarem uma atividade específica.
- Diferente de um bloco anônimo após definidas, são compiladas pelo Oracle e armazenadas no banco de dados a fim de que possam ser acessadas e executadas em qualquer momento.

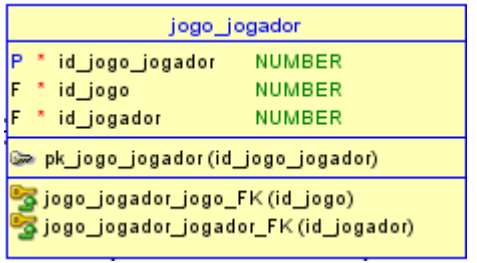
```
CREATE OR REPLACE PROCEDURE NOME_DA_PROCEDURE
(
  PARAM_ENTRADA  NUMBER,
  PARAM_ENTRADA2 IN VARCHAR2,
  PARAM_SAIDA    OUT NUMBER,
  PARAM_ENT_SAID IN OUT VARCHAR2
) AS
  --variaveis;
BEGIN

  --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
  --VARIAVEIS
BEGIN

  --codigo lógico aqui.

end; --ou end NOME_DA_FUNCTION
```

Procedures & Functions

- 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).





MATERA systems

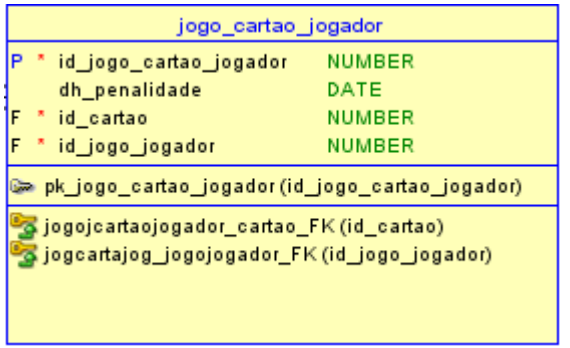
SubQuerys

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

```
--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 )
```

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, 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.

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;
```

Triggers

```
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 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;
```

Triggers

- Exercícios:
  - Criar uma trigger que *depois* de inserir algum gol ele contabilize mais um ao jogo para aquela determinada equipe.
  - 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

```
Create or replace view vequipe as
select *
from   equipe;
```

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

```
select * from vEquipe;

select * from equipe;
```

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 e exceptions.

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.

Packages

```
create or replace package campenato is

  /*variáveis globais
   cursors globais
  */
   procedure adicionarJogador(pisNome varchar2);

end;
```

```
create or replace package body campenato is

  /*variáveis privados
   cursors privados
  */
  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