Apostila Oracle

Ricardo Terra
rterrabh [at] gmail.com

CV

Nome: Ricardo Terra

Email: rterrabh [at] gmail.com

www: ricardoterra.com.br

Twitter: rterrabh

Lattes: lattes.cnpq.br/ 0162081093970868

Ph.D. (UFMG/UWaterloo),

Post-Ph.D. (INRIA/Université Lille 1)

Background

Acadêmico: UFLA (desde 2014), UFSJ (1 ano), FUMEC (3 anos), UNIPAC (1 ano), FAMINAS (3 anos)

Profissional: DBA Eng. (1 ano), Synos (2 anos), Stefanini (1 ano)

Aula 01

Instalando o servidor *Oracle XE*

- Baixando o *Oracle XE*
  - Registrar-se na *Oracle*
  - *Ir ao endereco* *http://www.oracle.com/technology/software/products/database/xe/index.html*
- Instalando o *Oracle XE*
  - Depois que já baixar o instalador do *Oracle*, basta executá-lo e seguir os passos
    - Você deverá registrar o cadastro de uma senha para o usuário *system*
  - Dica:
    - Geralmente o próprio instalador já o configura como serviço do *Windows*, porém, como o mesmo é muito pesado, vá em *Serviços* e configure o serviço ***OracleXETNSListener*** e ***OracleServiceXE*** para iniciar manualmente
    - Assim, quando você precisar utilizar o banco de dados, basta ir na opção *Start Database* disponível no menu iniciar

Instalando o cliente *Oracle XE*

- Baixando o *Oracle SQL Developer*
  - Registrar-se na *Oracle*
  - *Ir ao endereco* *http://www.oracle.com/technology/software/products/sql/index.html*
  - Este é um cliente gráfico gratuito e muito utilizado

- Baixando o *SQL*Plus*
  - O *SQL*Plus* é largamente utilizado por DBAs e desenvolvedores para a interação com a base de dados. Usando o *SQL*Plus*, você pode executar todas as instruções SQL e programas PL/SQL, formatar resultados de consultas e administrar a base de dados
  - Ir ao endereço http://www.oracle.com/technology/software/products/database/oracle10g/htdocs/10201winsoft.html
  - Baixar *Oracle Database 10g Client Release 2 (10.2.0.1.0)*
    - Instalar pela opção *Runtime*.

Configurando o *SQL*Plus*

- Configurando o *SQL*Plus*
  - Depois de instalado, basta entrar no **Net Configuration Assistant**.
    - Selecione *Configuração do Nome do Serviço de Rede Local*
    - Selecione *Adicionar*
    - Em *nome do serviço:* ***xe***
    - Selecione *TCP*
    - Em *nome do host* deve ser inserido o *IP* ou *hostname* onde encontra-se instalado o *Oracle*
    - Realize o teste
      - Talvez seja necessário alterar o *login* para o teste
  - Com isto, o *SQL*Plus* já estará funcionando perfeitamente.

Arquitetura do *Oracle*

- Algumas arquiteturas de hardware permitem múltiplos computadores compartilharem os mesmos dados, softwares ou periféricos. O *Oracle* permite tirar proveito dessa característica através da execução de **múltiplas instancias que compartilham um único banco de dados**. Assim os usuários de múltiplas maquinas podem acessar o mesmo banco de dados com uma melhoria da performance.

- Quando o *Oracle* é iniciado um grupo de *buffers* de memória denominado **System Global Área (SGA)** é alocado e alguns processos que permanecem em *background* são inicializados. A combinação dos *buffers* com os processos em *background* formam uma **instância**.

- **SGA** é um grupo de buffers de memória compartilhados que são alocados pelo *Oracle* para uma instância.

- **Processos em *background*** executam tarefas distintas assincronamente em benefício de todos os usuários de um banco de dados.

Arquitetura do *Oracle*

- ***Tablespace*** é uma sub-divisão lógica de um banco de dados utilizado para agrupar estruturas lógicas relacionadas.

- As *tablespaces* apenas especificam a localização de armazenamento do banco de dados e são armazenadas fisicamente em ***datafiles***, que alocam imediatamente o espaço especificado na sua criação.

- A primeira *tablespace* criada pelo *Oracle* é a SYSTEM.

- Existe um relacionamento "um para muitos" entre os bancos e as *tablespaces* e um relacionamento "um para muitos" entre as *tablespaces* e os *datafiles*. A qualquer momento um *datafile* pode ser incluído em uma *tablespace.*

- Um banco de dados pode ter vários **usuários**, cada qual com seu **esquema**, que nada mais é do que uma coleção lógica de objetos de banco de dados, como tabelas e índices. Por sua vez, esses objetos referem-se às estruturas físicas dos dados, que são armazenados nos *datafiles* das ***tablespaces***.

Usuários do Oracle

- Usuários
  - SYS
    - Usuário que é conhecido como o proprietário do dicionário. Ele possui todas as tabelas bases e visões de acesso ao usuário de um dicionário de dados.
    - Nenhum usuário deve alterar (UPDATE, DELETE ou INSERT) qualquer linha ou objetos de esquema contidos no esquema SYS, pois tal ação pode comprometer a integridade dos dados.
  - SYSTEM
    - Usuário do DBA.
    - Pode-se dizer que o SYSTEM é um pouco mais “ fraco” que o SYS.
    - Responsável pela criação de tabelas e visões adicionais que exibem informações administrativas e/ou são utilizadas pelas ferramentas da *Oracle*.
    - Nunca deve-se criar tabelas no esquema SYSTEM de interesse de usuários particulares.
  - HR
    - Esquema de exemplo.

Criando seu próprio usuário

- Para criar seu próprio usuário, basta:

  create user <*nome_do_usuario*> identified by <*senha_do_usuario*>;

  grant connect, resource, create view, create sequence, create synonym,
  create trigger, create procedure to <*nome_do_usuario*>;

- Veremos maiores detalhes sobre as instruções acima nas aulas posteriores.

Tabela *dual*

- A tabela *dual* é uma tabela que é criada pelo *Oracle* junto com o dicionário de dados.
- Ela consiste exatamente de uma única coluna cujo nome é DUMMY e um registro cujo valor é “X”.

```
SQL> desc dual
Name                 Null?    Type
-------------------- -------- ---------------
DUMMY                         VARCHAR2(1)
```

```
SQL> select * from dual;
D
-
X
```

- O proprietário da tabela *dual* é o SYS, mas ela pode ser acessada por qualquer usuário.

Tabela *dual*

- A tabela *dual* é a tabela predileta para selecionar uma *pseudo*-coluna (tal como *sysdate*)

```
SQL> select sysdate from dual
SYSDATE
---------
15-FEB-08
```

- Mesmo sendo possível excluir o único registro ou inserir registros adicionais, realmente isto não deve ser feito.

Comandos importantes

- SPOOL <*caminho_do_arquivo*>
  - Grava toda a saída a partir deste comando no arquivo indicado até que seja inserido o seguinte comando:
    - SPOOL off

Comandos importantes

- SHOW <*opcao*>
  - ALL
    - Exibe todas as informações do ambiente
  - SGA
    - Exibe o tamanho atual da SGA (*System Global Area*)
  - RELEASE
    - Exibe o número da versão atual do *Oracle*
  - USER
    - Exibe o usuário corrente

Comandos importantes

- PASSWORD <*usuario*>
  - Altera a senha de um dado usuário.
- CONNECT <*usuario*>/<*senha*>@<*nome_do_servico*>
  - Conecta em dado usuário.
- DISCONNECT
  - Comita alterações pendentes na base de dados e desconecta o usuário do *Oracle,* porém não encerra o *SQL*Plus*.
- EXIT ou QUIT
  - Desconecta o usuário do *Oracle* e encerra o *SQL*Plus.*

Comandos importantes

- / (barra)
  - Executa o comando SQL que está no *buffer*.
- EDIT ou EDIT <*nome_do_arquivo*>
  - Edita o *buffer* ou um arquivo
- @<*caminho_do_arquivo*>
  - Executa todo o arquivo de *script*
  - Dica:
    - Caso queira inserir um comentário em um *script* basta inseri-lo entre ***/**** e ****/***.

Comandos importantes

- SET TIME ON
  - Exibe a hora no *prompt*.
- SET LINESIZE <*tamanho*>
  - Redimensiona o tamanho da linha a ser exibida.
  - Geralmente 100.
- SET PAGESIZE <*tamanho*>
  - Redimensiona o tamanho da página a ser exibida.
  - Geralmente 40.
- PAUSE <*texto*>
  - Exibe a mensagem e espera até que o usuário digite ENTER.
- PROMPT <*texto*>
  - Exibe a mensagem.

Comandos importantes

- select sysdate from dual;
  - Exibe a data.
- select systimestamp from dual;
  - Exibe a data e a hora.
- select TABLE_NAME from USER_TABLES;
  - Exibe a listagem de todas as tabelas do usuário ativo.
- DESC <*nome_da_tabela*> ou DESCRIBE <*nome_da_tabela*>
  - Descreve a estrutura da tabela.

Comandos importantes

- CLEAR <*opcao*>
  - screen
    - Limpa a tela.
  - buffer
    - Limpa o *buffer.*

- Dica interessante:
  - A seta para cima não volta aos comandos, para isto, no *SQL*Plus*, basta selecionar o texto que você deseja e, antes de soltar o botão esquerdo, aperte o botão direito. Assim, ele copiará o texto que você selecionou.

Utilizando parametrização

- Às vezes é necessária a parametrização de algum comando e isto pode ser facilmente realizado.

SQL> select *&campo* from dual
Enter value for campo: **sysdate**
old 1: select &campo from dual
new 1: select sysdate from dual
SYSDATE

15-FEB-08

Aula 02

Categorias de Instruções SQL

- DML (***D**ata **M**anipulation **L**anguage*)
  - Utilizada para acesso, criação, alteração e exclusão de dados em existentes estruturas do banco de dados.
    - *SELECT, INSERT, UPDATE, DELETE, EXPLAIN PLAN* e *LOCK TABLE* são as instruções de DML.
  - Também conhecida como LMD *(tradução desnecessária)*

- DDL (***D**ata **D**efinition **L**anguage*)
  - Utilizada para criar, alterar ou destruir objetos do banco de dados e seus privilégios.
    - *CREATE, ALTER, DROP, RENAME, TRUNCATE, GRANT, REVOKE, AUDIT, NOAUDIT e COMMENT* são as instruções de DDL.
  - Também conhecida como LDD *(tradução desnecessária)*

Categorias de Instruções SQL

- Controle de Transação (*Transaction Control*)
  - Utilizada para agrupar um conjunto de instruções como uma única transação.
    - *COMMIT, ROLLBACK, SAVE POINT, SET TRANSACTION* são as instruções de controle de transação.
- Controle de Sessão (*Session Control*)
  - Utilizada para controlar as propriedades de uma sessão de um usuário.
    - *ALTER SESSION, SET ROLE* são as instruções de controle de sessão.
- Controle de Sistema (*System Control*)
  - Utilizada para controlar as propriedades do banco de dados.
    - *ALTER SYSTEM* é a única instrução de controle de sistema.

Literais

- Literais são valores que representam uma constante. Existem quatro tipos de literais:
  - Texto (ou caractere)
    - Entre aspas simples.
      - Ex.: 'Firefox', 'Maria José'
  - Inteiro
    - Número sem casas decimais.
      - Ex.: -2 43 0
  - Número
    - Números inteiros e números decimais.
      - Ex.: 32 -2,55 3,14159 23e-10
  - Intervalo
    - Especifica um período de tempo em termos de anos e meses ou de dias e segundos.

Como trabalhar com outro esquema

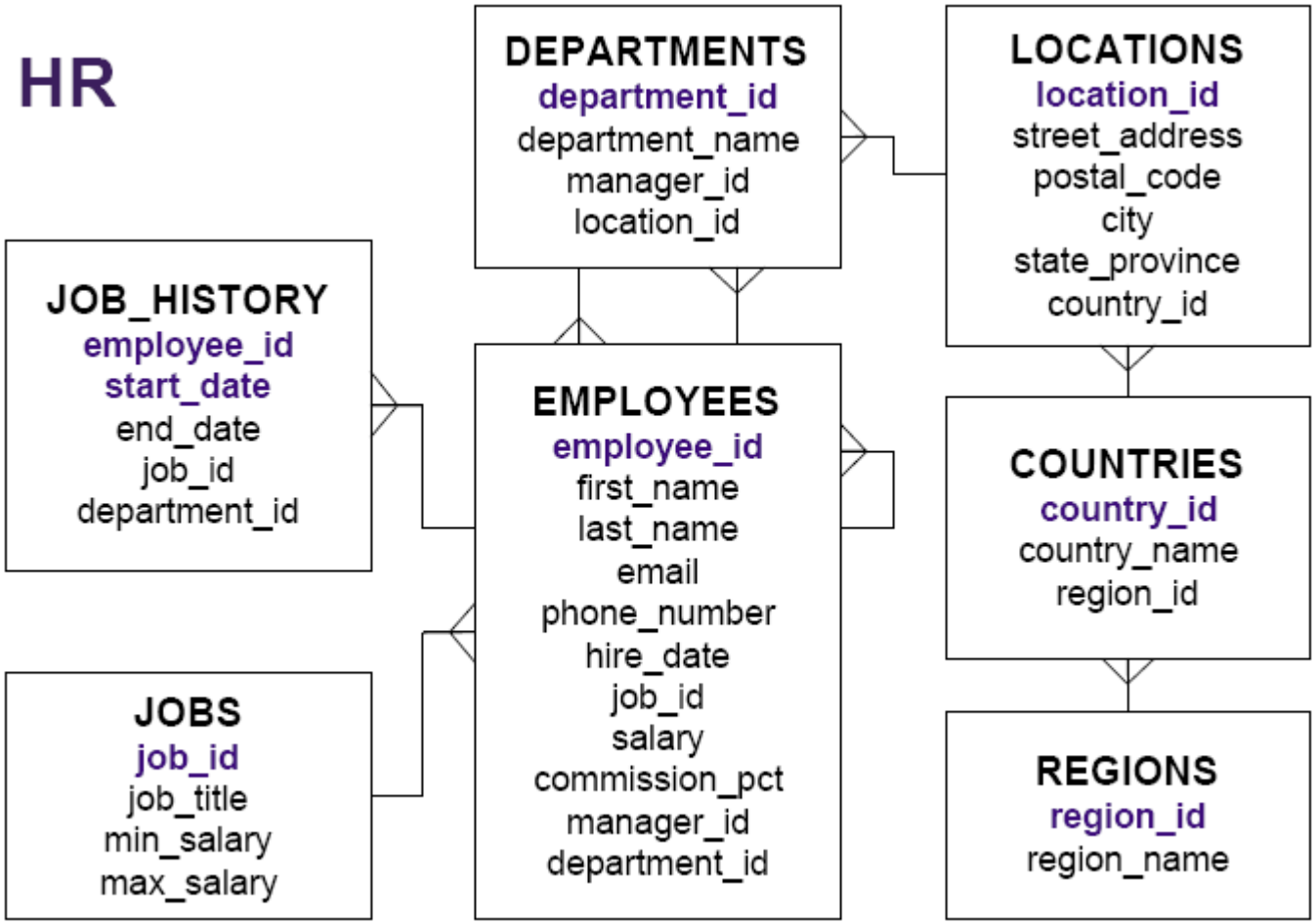
- Como já foi visto, o usuário no *Oracle* tem o seu próprio esquema.
- Na maioria das aulas irei criar um esquema que será utilizado em nosso aprendizado. O esquema de hoje será um do próprio *Oracle* denominado **HR**.
- O usuário **HR** vem bloqueado. Para desbloqueá-lo deve realizar o seguinte comando utilizando o usuário **SYSTEM**:
  - alter user HR identified by HRPASSWORD ACCOUNT UNLOCK;
- Para permitir um usuário a acessar tabelas de um outro usuário deve-se realizar o seguinte comando utilizando o usuário **SYSTEM**:
  - grant select on <*usuário_origem*>.<*tabela*> to <*usuário_destino*>;
    - Ex.:
      - grant select on HR.DEPARTMENTS to TERRA;

Como trabalhar com outro esquema

- As tabelas não pertencem ao seu usuário, mas sim, ao usuário **HR**, portanto para verificar as tabelas, que possui acesso, de um outro esquema deve ser realizado o seguinte comando:
  - select TABLE_NAME from ALL_TABLES where OWNER = '**HR**';

- Portanto, para acessar, descrever, enfim, qualquer ação nas tabelas deste esquema deve-se colocar o nome do esquema antes da tabela:
  - select * from **HR.**DEPARTMENTS;
  - desc **HR.**DEPARTMENTS;

Esquema HR


HR
DEPARTMENTS
department_id
department_name
manager_id
location_id
LOCATIONS
location_id
street_address
postal_code
city
state_province
country_id
JOB_HISTORY
employee_id
start_date
end_date
job_id
department_id
EMPLOYEES
employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
commission_pct
manager_id
department_id
COUNTRIES
country_id
country_name
region_id
JOBS
job_id
job_title
min_salary
max_salary
REGIONS
region_id
region_name

Instruções Básicas de SQL

- O SQL é a linguagem padrão ANSI para acesso a banco de dados relacionais e engloba tanto uma DDL quanto uma DML.

- É uma linguagem baseada em conjunto, isto é, com um simples comando recuperamos um conjunto de registros, sem precisarmos efetuar leituras registro a registro.

Select

- Comando para seleção (recuperação) de dados em uma tabela.
  - Uma sintaxe básica (veremos *order by* ainda hoje *e where* futuramente):
    - select <*lista_de_colunas*> from <*TABELA*>
      where <*condições*>
      order by <*lista_de_colunas*> <*asc*|*desc*>
  - Na lista de colunas, deverá ser citada cada uma das colunas separadas por vírgula (,).
    - select JOB_TITLE, MIN_SALARY, MAX_SALARY from HR.JOBS;
  - A lista de colunas pode ser substituida por asterisco (*), que indica a exibição de todas as colunas.
    - select * from HR.JOBS;

Select

- Em algumas situações o nome do cabeçalho de uma coluna pode apresentar pouco significado. Nestas situações pode ser utilizado um ‘apelido’ para a coluna.

- O ‘apelido’ é definido logo em seguida ao nome da coluna com um espaço ou utilizando a palavra **AS**. É recomendável inserir o nome do ‘apelido’ entre aspas duplas (").
  - Caso o apelido tenha espaço, é obrigatório o uso de aspas duplas (").

- Exemplo:
  - select JOB_TITLE AS “CARGO", MIN_SALARY AS "SALÁRIO BASE" from HR.JOBS;

    equivalente a
  - select JOB_TITLE “CARGO", MIN_SALARY "SALÁRIO BASE" from HR.JOBS;

Select distinct

- Em algumas situações o resultado pode apresentar repetição.
- Buscar os códigos de departamento dos empregados, por exemplo, retornará código de departamento repetido caso exista mais de um empregado que trabalhe em um mesmo departamento.
  - Observe:
    - select DEPARTMENT_ID from HR.EMPLOYEES order by 1;
      - Veremos em um futuro próximo o que quer dizer “order by 1”
- Para evitar registro repetidos basta inserir a palavra DISTINCT.
  - select distinct DEPARTMENT_ID from HR.EMPLOYEES order by 1;

Select distinct

- Esta unicidade não é aplicada somente à primeira coluna, mas sim, a toda linha. Observe fazendo a seguinte consulta:
  - select distinct DEPARTMENT_ID, JOB_ID from HR.EMPLOYEES order by 1;

- Observe que não houve um único registro com o mesmo código de departamento e código do cargo.

Ordenando

- **Ordenação (ORDER BY)**
  - A expressão ORDER BY é utilizada quando se pretende exibir os registros em uma determinada ordem, seja esta crescente (ASC - default) ou decrescente (DESC). Caso a expressão não esteja presente, os registros serão exibidos na ordem em que foram inseridos na tabela.
  - Os campos que constam na expressão ORDER BY não devem obrigatoriamente estarem presentes na expressão SELECT. Caso estejam presentes, pode-se utilizar números indicando que a ordenação será feira por determinado campo de acordo com a ordem do *select*.
  - Não é necessário possuir um índice fisicamente criado e composto pelos campos da ordenação para usar o ORDER BY. No entanto, na existência do índice, o comando será executado com melhor desempenho.

Ordenando

- Deve ser a última instrução do *select* e tem a seguinte sintaxe:
  - select ... order by <*lista_de_colunas*> <*asc|desc*> <*nulls first|nulls last*>
    - A lista de colunas pode referenciar à posição da coluna na lista de colunas do *select*, caso a coluna esteja sendo exibida.

- Exemplos:
  - select FIRST_NAME, SALARY from HR.EMPLOYEES order by SALARY desc

    equivale a

  - select FIRST_NAME, SALARY from HR.EMPLOYEES order by 2 desc
    - Observe que caso a ordenação seja feita por uma coluna que não esteja na lista de colunas do *select*, esta sintaxe não é possível.

Ordenando

- Mais exemplos:
  - Listagem do código do departamento e do nome do empregado ordenados ascendentemente pelo código do departamento:
    - select DEPARTMENT_ID, FIRST_NAME from HR.EMPLOYEES
      order by DEPARTMENT_ID asc
  - Agora ordenados descendentemente:
    - select DEPARTMENT_ID, FIRST_NAME from HR.EMPLOYEES
      order by DEPARTMENT_ID desc
  - Agora ordenados descendentemente pelo código do departamento e ascedentemente pelo nome do empregado:
    - select DEPARTMENT_ID, FIRST_NAME from HR.EMPLOYEES
      order by DEPARTMENT_ID desc, FIRST_NAME asc

Ordenando

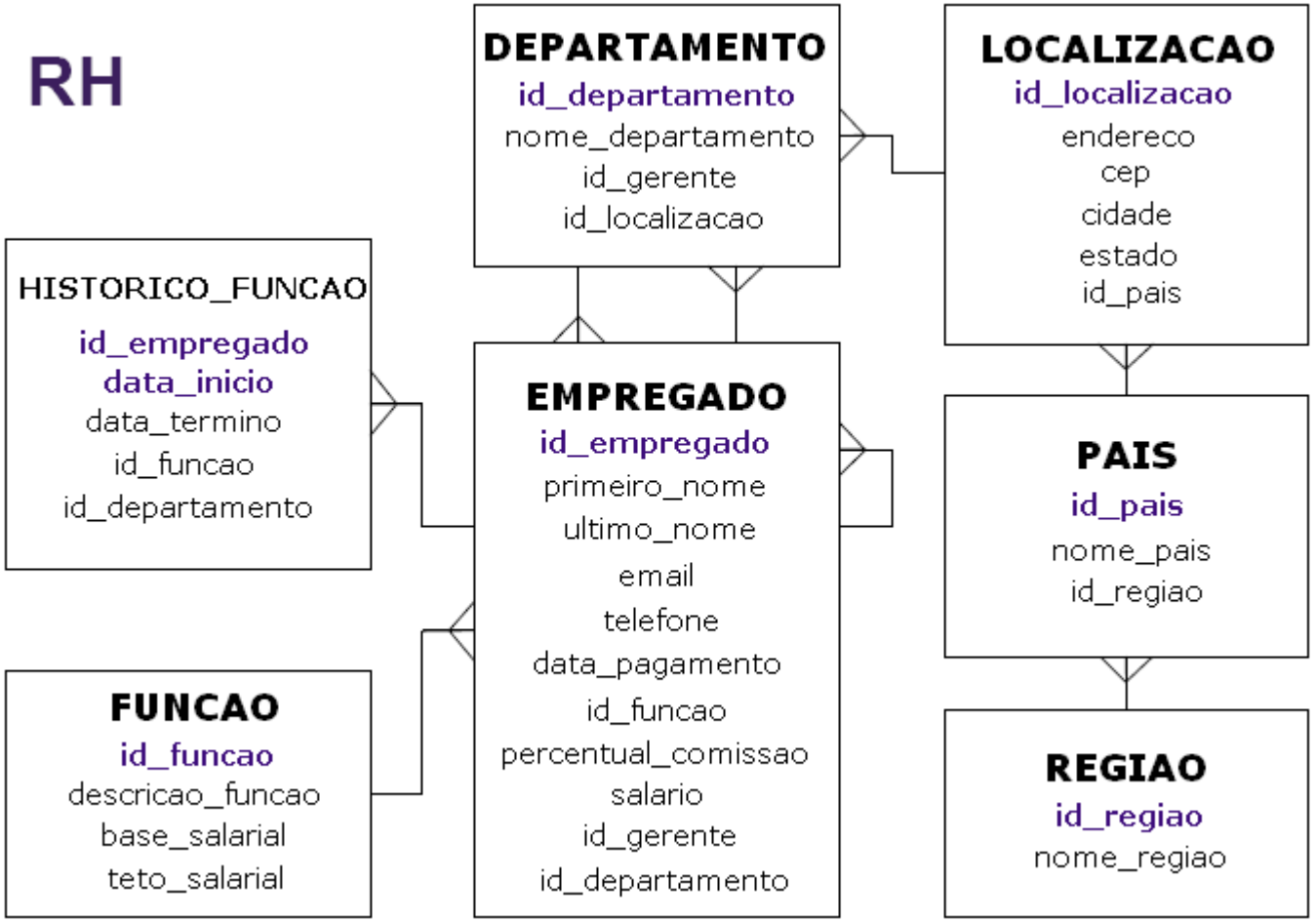
- Ainda mais exemplos:
  - Listagem do código do departamento e do nome do empregado ordenados ascendentemente pelo código do departamento, porém com a exibição de departamentos nulos no início:
    - select DEPARTMENT_ID, FIRST_NAME from HR.EMPLOYEES
      order by DEPARTMENT_ID asc nulls first;
  - Agora com os departamentos nulos no final:
    - select DEPARTMENT_ID, FIRST_NAME from HR.EMPLOYEES
      order by DEPARTMENT_ID asc nulls last;

Limitando o número de resultados

- As vezes você não está interessado em ver o resultado como um todo, mas sim, alguma parte do resultado. Para isto, o *Oracle* utiliza de uma *pseudo*-coluna conhecida como ROWNUM a partir do resultado.
- Por exemplo, a seguinte consulta retorna o nome e o salário de todos os empregados ordenados do maior salário para o menor.
  - select FIRST_NAME, SALARY from HR.EMPLOYEES order by SALARY desc;
- Mas o interesse é obter somente a lista dos TOP 3. Portanto o uso de ROWNUM é indispensável:
  - ```
    select * from (
        select FIRST_NAME, SALARY from HR.EMPLOYEES order by SALARY desc
    ) where ROWNUM <= 3;
    ```
- A princípio parece ser difícil já que outros SGBDs fazem de um modo muito mais fácil, porém estamos trabalhando com um SGBD de alta robustez.
    ```

Aula 03

Esquema RH (tradução do HR)



Restrigindo

- Existem vários operadores que pode ser usados para a restrição do número de resultados.
  - **Operadores de comparação**
    - Equalidade: =
    - Diferença: <> != ^=
    - Demais:
      - > < >= <=
      - ANY
      - SOME
      - ALL
  - **Lógicos:**
    - AND OR NOT
      - A utilização dos operadores lógicos utiliza a precedência de parênteses como em linguagens de programação.
  - **Outros operadores:**
    - IN
      - NOT IN
    - BETWEEN
    - EXISTS
    - IS NULL
      - IS NOT NULL
    - LIKE

Restrigindo

- Operadores de comparação
  - São operadores que retornam um booleano como resultado.
  - Permite que realizemos consultas restringindo pela equalidade, diferença, relações de maior, maior ou igual, menor e menor ou igual.
    - O oracle acrescenta os operadores SOME, ANY e ALL.

Restrigindo

- Um exemplo de uso da equalidade é a busca dos empregados de um determinado departamento, por exemplo:
  - select * from RH.EMPREGADO where ID_DEPARTAMENTO **=** 90
- Para retornar os que não estão em um departamento basta:
  - select * from RH.EMPREGADO where ID_DEPARTAMENTO **<>** 90

    equivalente a
  - select * from RH.EMPREGADO where ID_DEPARTAMENTO **!=** 90

    equivalente a
  - select * from RH.EMPREGADO where ID_DEPARTAMENTO **^=** 90

Restrigindo

- Um exemplo de uso da relação de maior, maior ou igual, menor ou menor ou igual é o busca de um empregado que possui um salário maior que um certo valor e menor do que outro.
  - select * from RH.EMPREGADO where SALARIO **>** 13000
    - Busca os que possuem salário maior que 13.000
  - select * from RH.EMPREGADO where SALARIO **>=** 13000
    - Busca os que possuem salário maior ou igual a 13.000
  - select * from RH.EMPREGADO where SALARIO **<** 13000
    - Busca os que possuem salário menor que 13.000
  - select * from RH.EMPREGADO where SALARIO **<=** 13000
    - Busca os que possuem salário menor ou igual a 13.000

Restrigindo

- ANY ou SOME
  - São utilizados para comparar um valor para cada valor em uma lista ou *subquery.*
  - **Devem** ser precedidos por um dos operadores de comparação.
    - **=**, **<>**, **!=**, **^=**, **<**, **<=**, **>**, ou **>=**
  - Para retornar os usuário que estão em uma determinada lista de departamentos utilizamos:
    - select * from RH.EMPREGADO where ID_DEPARTAMENTO **= ANY** (90,100)

      equivalente a
    - select * from RH.EMPREGADO where ID_DEPARTAMENTO **= SOME** (90,100)

Restrigindo

- ALL
  - São utilizados para comparar um valor para todos os valores em uma lista ou *subquery.*
  - Assim como SOME e ANY, **devem** ser precedidos por um dos operadores de comparação.
  - Para retornar os usuário que **não** estão nos departamentos de uma determinada lista utilizamos:
    - select * from RH.EMPREGADO where ID_DEPARTAMENTO **!= ALL** (10,30,50)

Restrigindo

- Operadores lógicos
  - Os operadores lógicos são utilizados para combinar os resultados de duas condições de comparação para produzir um único resultado (AND e OR) ou para inverter o resultado de uma simples comparação (NOT).
  - NOT
    - Às vezes, no português falado dizemos, por exemplo, que queremos todos os funcionários que **não** estão no departamento 30.
    - A consulta abaixo faz exatamente isto:
      - select * from RH.EMPREGADO where **not** ID_DEPARTAMENTO = 30
    - E poderia ser feito, de forma equivalente, como abaixo:
      - select * from RH.EMPREGADO where ID_DEPARTAMENTO **<>** 30

Restrigindo

- Operadores lógicos
  - AND
    - Às vezes, no português falado dizemos, por exemplo, que queremos todos os funcionários que estão no departamento 30 e que ganham mais de 3.000.
    - A consulta abaixo faz exatamente isto:
      - select * from RH.EMPREGADO where
        ID_DEPARTAMENTO = 30 **AND** SALARIO > 3000
  - OR
    - Existem outros casos que queremos todos os funcionários que estão no departamento 30 ou que ganham mais de 17.000.
    - A consulta abaixo faz exatamente isto:
      - select * from RH.EMPREGADO where
        ID_DEPARTAMENTO = 30 **OR** SALARIO > 17000

Restrigindo

- IN e NOT IN
  - O operador IN e NOT IN são utilizados para testar uma condição de um grupo de valores.
  - IN é equivalente ao operador =ANY, pois ambos retornam verdadeiro se um valor existe em uma lista ou em uma *subquery*.
  - NOT IN é equivalente ao operador !=ALL, pois ambos retornam verdadeiro se o valor não existe em uma lista ou em uma *subquery*.
  - Vamos ver um exemplo no próximo *slide* e um comparativo com o ANY e SOME.

Restrigindo

- IN
  - Para retornar os usuário que estão em uma determinada lista de departamentos utilizamos:
    - select * from RH.EMPREGADO where ID_DEPARTAMENTO **IN** (90,100)

      equivalente a

    - select * from RH.EMPREGADO where ID_DEPARTAMENTO **= ANY** (90,100)

- NOT IN
  - Para retornar os usuário que **não** estão nos departamentos de uma determinada lista utilizamos:
    - select * from RH.EMPREGADO where ID_DEPARTAMENTO **NOT IN** (10,30,50)

      equivalente a

    - select * from RH.EMPREGADO where ID_DEPARTAMENTO **!= ALL** (10,30,50)

Restrigindo

- BETWEEN
  - O operador BETWEEN permite testar se um determinado campo assume o valor dentro de um intervalo especificado. É utilizado por ser mais prático que o teste de >= e <=.
  - Para retornar os usuário que recebem entre 5.000 e 10.000 utilizamos:
    - select * from RH.EMPREGADO where SALARIO **between 5000 and 10000**
  - Caso queíramos os que não se enquadram neste intervalo, basta acrescentar o operador lógico NOT:
    - select * from RH.EMPREGADO where **not** SALARIO between 5000 and 10000

Restrigindo

- EXISTS
  - O operador EXISTS é sempre seguido por uma *subquery* em parênteses. Este operador retorna verdadeiro se o resultado da *subquery* retorna pelo menos um registro.

  - O exemplo abaixo retorna os empregados que trabalham no departamento de *marketing*. Pode não ficar claro o entendimento desde operador, porém o abordaremos mais detalhadamente ao falarmos de *subqueries*.

```
select * from RH.EMPREGADO e
   where EXISTS (
      select 1 from RH.DEPARTAMENTO d
         where d.ID_DEPARTAMENTO = e.ID_DEPARTAMENTO AND
            d.NOME_DEPARTAMENTO = 'Marketing'
   )
```

Restrigindo

- IS NULL e IS NOT NULL
  - Para encontrar valores nulos, você deve utilizar o operador IS NULL. Os operadores ‘=‘ ou ‘!=‘ **não** funcionam com valores nulos.
  - A literal NULL é a representação para campos sem valores (é bem diferente um campo ser nulo e ser zero). O nulo significa que nenhum valor se aplica.
    - No momento da criação da tabela é que se define se o campo pode ou não conter valores nulos.
  - Para retornar os usuários que não estão lotados em nenhum departamento utilizamos:
    - select * from RH.EMPREGADO e where ID_DEPARTAMENTO **is NULL**
  - Caso queíramos os que estejam lotados em um departamento:
    - select * from RH.EMPREGADO e where ID_DEPARTAMENTO **is not NULL**

Restrigindo

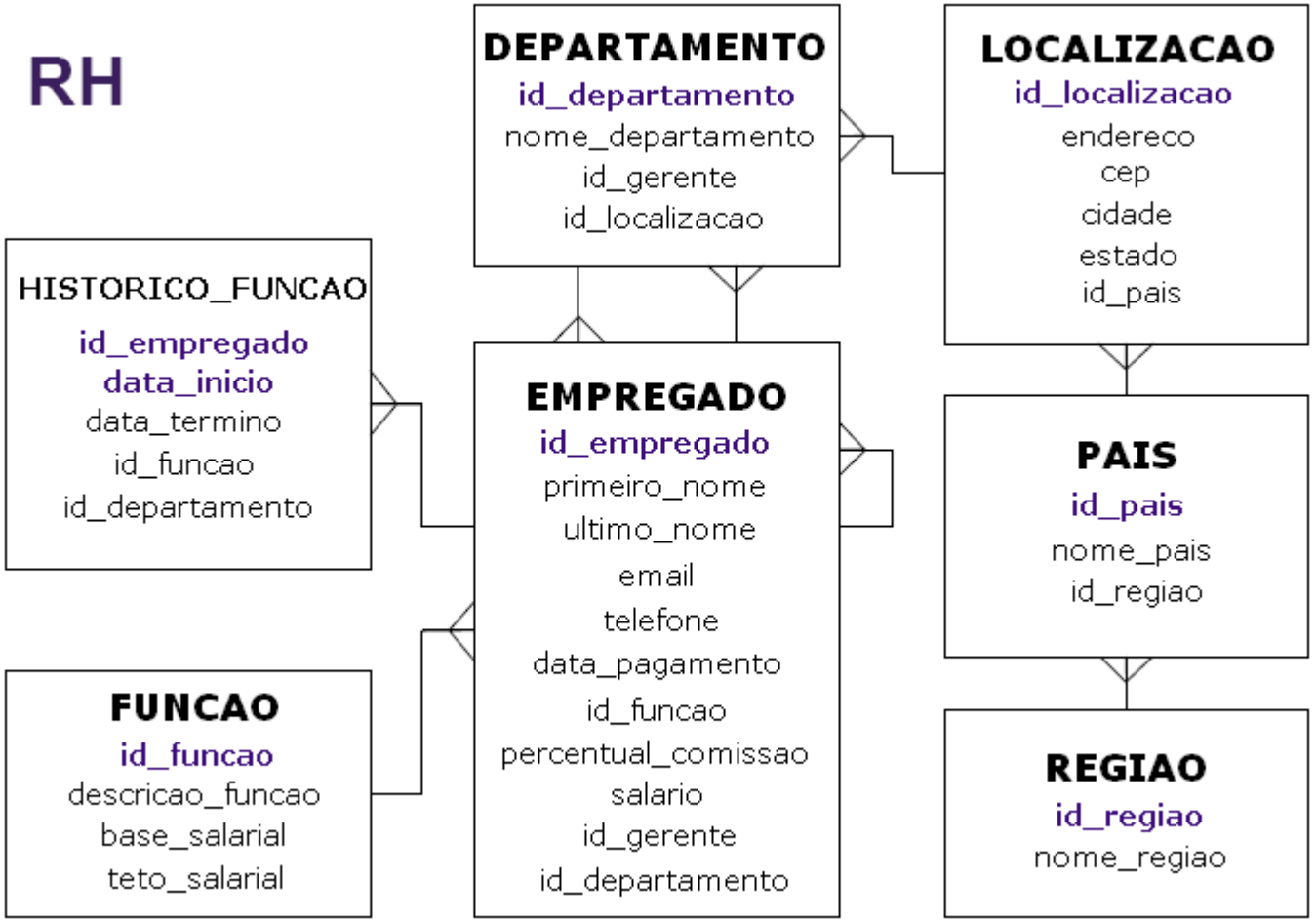
- LIKE
  - O operador LIKE é usado quando se deseja obter colunas de um registro que sigam um determinado padrão pré-especificado. Quando se quer saber o nome de todos os funcionários cujo nome começa com ‘João’ ou termina com ‘Silva’, utiliza-se o operador LIKE.
  - Comparando com MS-DOS:
    - % funciona como o * (indica quaisquer caracteres)
    - _ funciona como o ? (indica que naquela posição pode qualquer caractere)
  - Exemplo:
    - select * from RH.EMPREGADO where PRIMEIRO_NOME like 'Alex%'
      - Seleciona os empregados cujo nome inicia-se com ‘Alex’.
    - select * from RH.EMPREGADO where PRIMEIRO_NOME like '_o%'
      - Seleciona os empregados cujo nome possua a letra ‘o’ na 2º posição.

Restrigindo

- Restringindo pelo valor de uma data
  - No *Oracle*, '07-06-1994' não é uma data, mas sim, uma *string*. Portanto, como representar uma data para utilizar os operadores aprendidos?
  - Basta utilizar a função **to_date** que veremos mais à frente, porém segue um exemplo para já ir experimentando.
  - Selecionar os empregados que foram pagos em 07 de junho de 1994.
    - select * from RH.EMPREGADO
      where DATA_PAGAMENTO = to_date('07/06/1994', 'dd/mm/yyyy')
  - Selecionar os empregados que já foram pagos.
    - select * from RH.EMPREGADO
      where DATA_PAGAMENTO < sysdate

Aula 04

Esquema RH (tradução do HR)



Tipos de dados no Oracle

- No oracle, existem vários tipos de dados. Observe:

Categoria	Tipos de dados
Caractere	CHAR, NCHAR, VARCHAR2, NVARCHAR2
Número	NUMBER
Dados brutos alfanuméricos e binários	LONG, LONG RAW, RAW
Data e hora	DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND
Objetos largos	CLOB, NCLOB, BCLOB, BFILE
Identificadores de linha	ROWID, UROWID

  - Nos *slides* seguintes, veremos, em detalhes, os mais utilizados.

Tipos de dados

- CHAR(<*size*>)
  - É uma string alfanumérica de tamanho fixo, que possui um tamanho máximo em bytes.
  - Seu tamanho deve ser de 1 à 2000 bytes. O valor padrão é 1.
  - Dados armazenados em um tipo *char* são completados com espaços até completar o tamanho máximo.
    - Ex.: Gravar "Olá" em um char(10), ficaria mais ou menos assim: "Olá "
  - O Oracle garante que todo dado dentro deste tipo terá sempre o mesmo tamanho.
    - Se for menor, é completado com espaços.
    - Se for maior, um erro é levantado.

Tipos de dados

- VARCHAR2(<*size*>)
  - É uma string alfanumérica de tamanho variável, que possui um tamanho máximo em bytes.
  - Seu tamanho deve ser de 1 à 4000 bytes e não há valor padrão de tamanho.
  - Dados armazenados em um tipo *varchar2* somente requerem o espaço necessário para armazenar o dado.
    - Isto é, um coluna *varchar2(4000)* vazia gasta o mesmo espaço que uma coluna *varchar(1)* vazia.

Tipos de dados

- NUMBER(<*p*>,<*s*>)
  - O tipo NUMBER armazena números com uma precisão de *p* dígitos tendo uma escala de *s* dígitos reservados para as casas decimais.
  - A precisão e a escala de dígitos são opcionais, porém se não informadas o Oracle assume o valor máximo possível.
  - Exemplos:
    - NUMBER(6,2) → -9999.99 a 9999.99
    - NUMBER(2,0) → -99 a 99
  - Se o número a ser inserido é superior à capacidade, um erro é levantando. Porém, caso a parte decimal informada é superior a suportada pelo tipo, o Oracle realiza o arrendondamento.
    - Ex.:
      - 9.555 em um NUMBER(3,2) ficará 9.56
      - 99.0123 em um NUMBER(4,2) ficará 99.01
      - 100 em um NUMBER(4,2) gerará erro.

Tipos de dados

- DATE
  - É o tipo utilizado para armazenar informação de data e hora.
  - Este tipo pode ser convertido para outras formas de visualização, mas existe várias funções e propriedades especiais que permite sua manipulação e cálculos simples.
  - Um tipo data ocupa 7 bytes. As seguintes informações são armazenadas em cada tipo DATE:
    - Século
    - Ano
    - Mês
    - Dia
    - Hora
    - Minuto
    - Segundo

Concatenação

- Concatenação
  - Antes de iniciar com funções, vamos falar um pouco sobre o operador de concatenação.
  - Ele é representado por duas barras verticais (||) e é utilizado para concatenar ou juntar duas strings.
  - Exemplos:
    - select 'Eu ' || 'amo ' || 'Oracle' as "FRASE" from DUAL;
    - select PRIMEIRO_NOME || ' ' || ULTIMO_NOME as "NOME COMPLETO" from RH.EMPREGADO

Expressões

- Expressões
  - Antes de iniciar com funções, vamos falar um pouco sobre expressões.
  - Uma expressão é uma combinação de um ou mais valores, operadores e funções SQL que resultam em um único valor.
  - Exemplos:
    - 5+6 → 11
    - mod(4*2, 3) → 2
    - upper('José ' || 'oliveira') → 'JOSÉ OLIVEIRA'
    - sqrt(mod((4*4)+3, 5)) → 2

Expressão CASE

- Expressão CASE
  - A expressão CASE pode ser usada como uma derivação da lógica if.. then.. else no SQL.
  - Sintaxe:
    - CASE <expressao>
      WHEN <valor de comparação> THEN <valor de retorno>
      ELSE <valor de retorno>
      END
  - A cláusula WHEN pode se repetir.
  - A cláusula ELSE é opcional.
  - No *slide* seguinte, veremos um exemplo.

Expressão CASE

- Expressão CASE
  - Retorna o nome do país, o código da região e o continente. Este último é realizado a partir de uma expressão CASE pelo código da região.

```
select NOME_PAIS , ID_REGIAO,
  CASE ID_REGIAO
    WHEN 1 THEN 'Europa'
    WHEN 2 THEN 'America'
    WHEN 3 THEN 'Asia'
    ELSE 'Outro'
  END as "Continente"
from RH.PAIS
where upper(NOME_PAIS) like 'B%'
```

Funções

- O Oracle criou várias funções que podem ser chamadas a partir de instruções SQL.

- O Oracle possui cinco classes principais de funções:
  - Funções de única linha (serão abordadas nesta aula)
  - Funções de agregação (serão abordadas nas próximas aulas)
  - Funções analíticas
  - Funções de referência à objetos
  - Funções definidas pelo programador

Funções de única linha (single-row funcions)

- Funções de uma única linha
  - Trabalha em expressões derivadas de colunas ou literais.
  - São executadas uma vez para cada linha recuperada.

- Existem vários tipos: funções textuais, funções numéricas, funções de data, funções de conversão e funções diversas.

- Todas as funções de uma única linha pode ser incorporadas no SQL (e, certamente, em PL/SQL).
  - Elas são utilizadas no SELECT, WHERE e ORDER BY de instruções SELECT.

Funções de única linha de valores nulos

- Existem as funções NVL e NVL2

- NVL (x1, x2)
  - Possui dois argumentos nos quais ambos são expressões.
  - A função NVL retorna x2 se x1 é nulo.
    - Se x1 não é nulo, então x1 é retornado.

  - Ex.:
    - select PRIMEIRO_NOME || ' ' || ULTIMO_NOME as "NOME COMPLETO",
      NVL(PERCENTUAL_COMISSAO,0) AS "COMISSAO (%)"
      from RH.EMPREGADO order by 2 desc

Funções de única linha (single-row funcions)

- Veremos agora várias tabelas com a listagem de todas as funções de única linha utilizadas pelo Oracle.

- A idéia é passar uma visão sobre cada uma delas, porém a sintaxe destas funções deverá ser estudada nas bibliografias da disciplina.

Funções textuais de única linha (1/2)

Função	Descrição
ASCII	Retorna o código na tabela ASCII equivalente ao caractere informado.
CHR	Retorna o caractere dado o código da tabela ASCII.
CONCAT	Concatena duas *strings*, assim como o operador ||.
INITCAP	Retorna uma *string* com a primeira letra de cada palavra em maíuscula.
INSTR	Encontra a posição numérica inicial de uma *string* dentro de uma outra *string.*
INSTRB	Assim com INSTR, porém conta bytes ao invés de caracteres.
LENGTH	Retorna o tamanho de uma *string* em caracteres.
LENGTHB	Retorna o tamanho de uma *string* em bytes.
LOWER	Converte toda a *string* para minúscula.
LPAD	Complementa à esquerda uma *string* até um tamanho especificado usando um caractere específico.

Funções textuais de única linha (2/2)

Função	Descrição
LTRIM	Retira espaços à esquerda de uma *string*.
RPAD	Complementa à direita uma *string* até um tamanho especificado usando um caractere específico.
RTRIM	Retira espaços à direita de uma *string*.
REPLACE	Realiza pesquisa e substituição de *substring*.
SUBSTR	Retorna uma parte de uma *string* especificada pelo posição numérica dos caracteres.
SUBSTRB	Retorna uma parte de uma *string* especificada pelo posição numérica dos bytes.
SOUNDEX	Retorna a representação fonética de uma *string*.
TRANSLATE	Realiza pesquisa e substituição de caracteres.
TRIM	Retira espaços à direita e à esquerda de uma *string*.
UPPER	Converte toda a *string* para maiúscula.

Funções numéricas de única linha (1/3)

Função	Descrição
ABS	Retorna o valor absoluto.
ACOS	Retorna o arco-cosseno.
ASIN	Retorna o arco-seno.
ATAN	Retorna o arco-tangente.
ATAN2	Retorna o arco-tangente. Recebe dois parâmetros.
BITAND	Retorna o resultado de uma operação AND nos bits de dois parâmetros.
CEIL	Retorna o próximo inteiro superior.
COS	Retorna o cosseno.
COSH	Retorna o cosseno hiperbólico.
EXP	Retorna a base de um logaritmo natural elevada a uma potência.

Funções numéricas de única linha (2/3)

Função	Descrição
FLOOR	Retorna o próximo inteiro inferior.
LN	Retorna o logaritmo natural.
LOG	Retorna o logaritmo.
MOD	Retorna o módulo (resto) de uma operação de divisão.
POWER	Retorna um número elevado a uma potência.
ROUND	Arredonda o número.
SIGN	Retorna o signal: negativo, positivo ou zero.
SIN	Retorna o seno.
SINH	Retorna o seno hiperbólico.
SQRT	Retorna a raiz quadrada de um número

Funções numéricas de única linha (3/3)

Função	Descrição
TAN	Retorna a tangente.
TANH	Retorna a tangente hiperbólica.
TRUNC	Retorna o número truncado.

Funções de data de única linha (1/2)

Função	Descrição
ADD_MONTHS	Adiciona um número de meses a uma data.
CURRENT_DATE	Retorna a data corrente.
CURRENT_TIMESTAMP	Retorna a data e hora corrente.
DBTIMEZONE	Retorna o fuso horário do banco de dados.
EXTRACT	Retorna um componente de uma data.
FROM_TZ	Retorna uma data e hora com o fuso horário dado.
LAST_DAY	Retorna o último dia do mês.
LOCALTIMESTAMP	Retorna a data e hora corrente do fuso horário da sessão.
MONTHS_BETWEEN	Retorna o número de meses entre duas datas.
NEW_TIME	Retorna a data e hora em um diferente fuso horário.

Funções de data de única linha (2/2)

Função	Descrição
NEXT_DAY	Retorna o próximo dia da semana de um dada data.
ROUND	Arredonda a data/hora.
SESSIONTIMEZONE	Retorna o fuso horário do sessão corrente.
SYS_EXTRACT_UTC	Retorna o UTC (GMT) de uma data/hora.
SYSDATE	Retorna a data corrente.
SYSTIMESTAMP	Retorna a data e hora corrente.
TRUNC	Trunca uma data para uma dada granularidade.
TZ_OFFSET	Retorna a diferença da UTC de uma determinada localização.

Funções de conversão de única linha (1/2)

Função	Descrição
ASCIISTR	Converte caracteres para ASCII.
BIN_TO_NUM	Converte uma string de bits para um número.
CAST	Converte tipos.
CHARTOROWID	Converte um caractere para um tipo ROWID.
COMPOSE	Converte para UNICODE.
CONVERT	Converte de um conjunto de caracteres para outro.
DECOMPOSE	Decompõe um string UNICODE.
HEXTORAW	Converte um hexadecimal para um raw.
NUMTODSINTERVAL	Converte um número para um intervalo de dias por segundo.
NUMTOYMINTERVAL	Converte um número para um intervalo de anos por mês.

Funções de conversão de única linha (2/2)

Função	Descrição
RAWTOHEX	Converte um raw para um hexadecimal.
ROWIDTOCHAR	Converte um ROWID para um caractere.
TO_CHAR	Converte e formata uma data em uma *string*.
TO_DATE	Converte uma *string* para uma data, especificando o formato.
TO_DSINTERVAL	Converte uma *string* para um intervalo dias por segundo.
TO_MULTIBYTE	Converte um caractere de um único byte para seu correspondente multibyte.
TO_NUMBER	Converte uma string numérica para um número, especificado o formato.
TO_SINGLE_BYTE	Converte um caractere multibyte para seu correspondente único byte.
TO_YMINTERVAL	Converte uma *string* para um intervalo anos por mês.
UNISTR	Converte UCS2 Unicode.

Funções diversas de única linha (1/2)

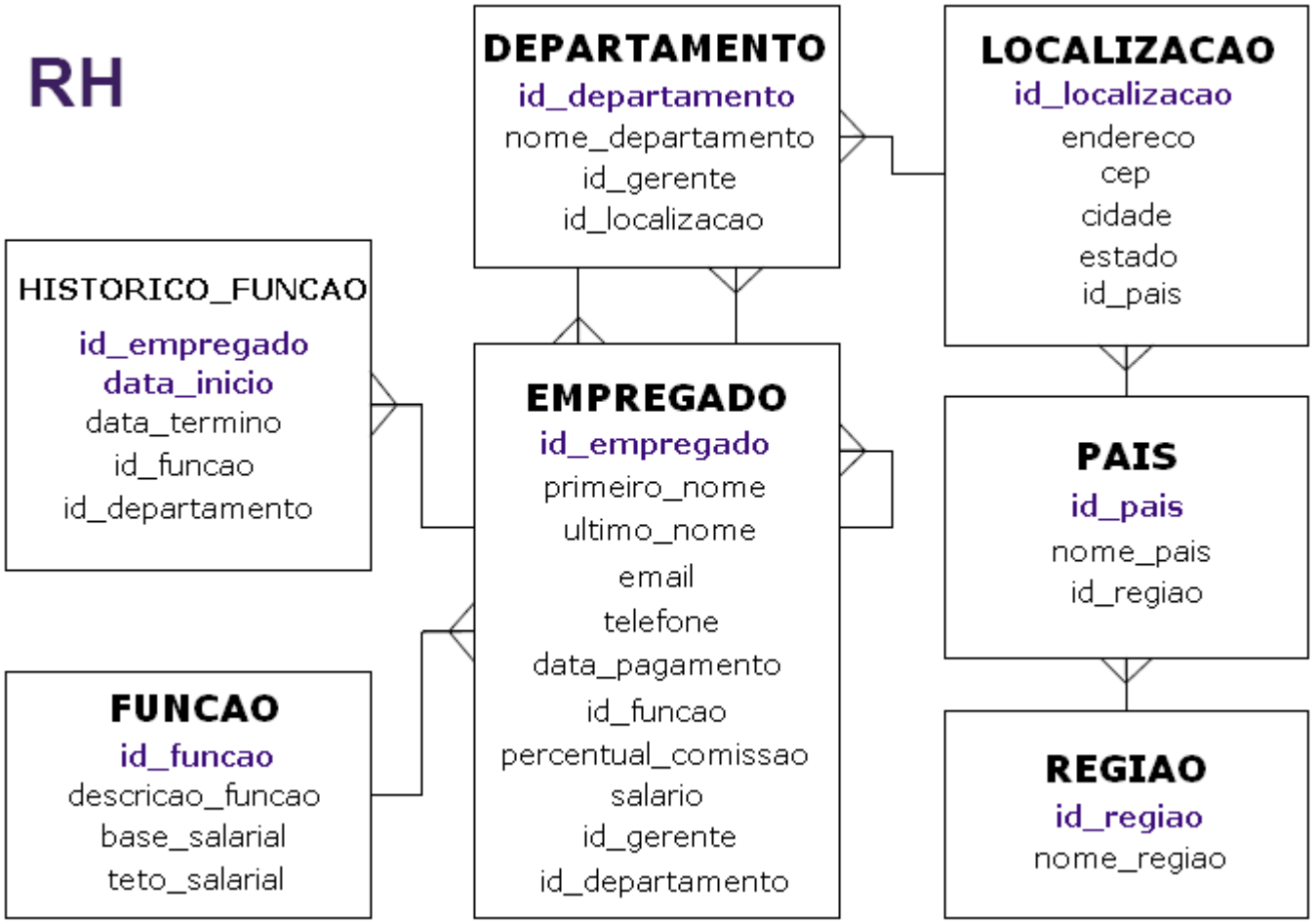
Função	Descrição
BFILENAME	Retorna o localizador BFILE para o especificado diretório e arquivo.
COALESCE	Retorna o primeiro não-nulo em uma lista.
DECODE	Instrução *case* em uma única linha (uma função if.. então.. else)
DUMP	Retorna uma *substring* raw na codificação especificada.
EMPTY_BLOB	Retorna um localizador BLOB vazio.
EMPTY_CLOB	Retorna um localizador CLOB vazio.
GREATEST	Organiza os argumentos e retorna o maior.
LEAST	Organiza os argumentos e retorna o menor.
NULLIF	Retorna NULL se duas expressões são iguais.
SYS_CONNECT_BY_PATH	Retorna valores da raiz até os nodos de uma consulta usando CONNECT BY.

Funções diversas de única linha (2/2)

Função	Descrição
SYS_CONTEXT	Retorna vários atributos de sessão, tais como endereço IP, terminal e usuário corrente.
UID	Retorna o ID numérico do usuário da sessão corrente.
USER	Retorna o nome do usuário da sessão corrente.
USERENV	Não mais utilizado. Substituído por SYS_CONTEXT.
VSIZE	Retorna o tamanho interno em bytes de uma expressão.

Aula 05

Esquema RH (tradução do HR)



Fundamentos de funções de agregação

- Funções de agregação (*group functions*)
  - Também conhecidas como **funções de agrupamento** e retornam um valor baseado no número de entradas.
  - O número exato de entradas não está determinado até que a consulta tenha sido executada e todas as linhas tenham sido trazidas. Isto difere das funções de uma única linha, em que o número de entradas já é conhecido antes que a consulta seja executada.
  - Devido a diferença acima, funções de agregação possuem requisitos e comportamentos ligeiramente diferentes que funções de uma única linha.

Fundamentos de funções de agregação

- Valores Nulos
  - Funções de agregação não processam valores nulos e nunca retornam um valor nulo, mesmo quando somente valores nulos são avaliados.
    - Ex.:
      - Uma contagem (COUNT) ou uma soma (SUM) de valores nulos irá retornar 0 (zero).
- Aplicabilidade
  - A maioria das funções de agregação podem ser aplicadas em todos os valores (ALL) ou somente em valores distintos (DISTINCT).
  - Ex.:
    - Soma de todos os salários
      - select sum(SALARIO) from RH.EMPREGADO

        ou
      - select sum(**all** SALARIO) from RH.EMPREGADO
    - Soma de todos os salários diferentes
      - select sum(**distinct** SALARIO) from RH.EMPREGADO

Fundamentos de funções de agregação

- Observe a consulta simples abaixo e o seu resultado:
  - select PRIMEIRO_NOME || ' ' || ULTIMO_NOME as "NOME COMPLETO", SALARIO
    from RH.EMPREGADO where ID_DEPARTAMENTO = 90 order by 1

```
NOME COMPLETO                                SALARIO
-------------------------------------------- ----------
Lex De Haan                                       17000
Neena Kochhar                                     17000
Steven King                                       24000
```

- Observe a consulta utilizando função de agregação abaixo e o seu resultado:
  - select avg(SALARIO) "MEDIA", avg(ALL SALARIO) "MEDIA",
    avg(DISTINCT SALARIO) "MEDIA DISTINTA",
    count(SALARIO) "CONTAGEM", count(DISTINCT SALARIO) "CONTAGEM DISTINTA",
    sum(SALARIO) "SOMA", sum(DISTINCT SALARIO) "SOMA DISTINTA" from TABELA_ACIMA

```
     MEDIA      MEDIA MEDIA DISTINTA   CONTAGEM CONTAGEM DISTINTA       SOMA SOMA DISTINTA
---------- ---------- -------------- ---------- ----------------- ---------- -------------
19333.3333 19333.3333          20500          3                 2      58000         41000
```

Funções de agregação

- Usando funções de agregação
  - Assim como funções de uma única linha, o Oracle oferece uma grande variedade de funções de múltiplas linhas. Estas funções aparecem no SELECT ou nas cláusulas HAVING de instruções SELECT.
  - Quando utilizada em um SELECT, geralmente requer uma cláusula GROUP BY, porém quando a mesma não é especificada, o agrupamente padrão é por todo o resultado.
  - Funções de agregação **NÃO PODEM** aparecer na cláusula **WHERE** de uma instrução SELECT.
  - Nos próximos *slides* veremos as seguintes funções de agregação:
    - AVG
    - COUNT
    - MAX
    - MIN
    - SUM

Funções de agregação

- AVG (média aritmética):
  - Retorna a média dos valores de uma determinada coluna. Sintaxe:
    - avg([ALL | DISTINCT] <nome-da-coluna>)
  - Ex.:
    - select avg(SALARIO) from EMPREGADO where ID_DEPARTAMENTO=30
      - Seleciona a média salarial dos empregados do departamento 30

- SUM (soma de valores):
  - Retorna a soma dos valores de uma determinada coluna. Sintaxe:
    - sum([ALL | DISTINCT] <nome-da-coluna>)
  - Ex.:
    - select sum(SALARIO) from EMPREGADO
      - Seleciona a soma dos salários de todos os empregados

Funções de agregação

- MAX (maior valor):
  - Retorna o maior valor de uma determinada coluna. Sintaxe:
    - max([ALL | DISTINCT] <nome-da-coluna>)
  - Exemplo:
    - select max(SALARIO) from EMPREGADO
      - Seleciona o maior salário de todos os empregados

- MIN (menor valor):
  - Retorna o menor valor de uma determinada coluna. Sintaxe:
    - min([ALL | DISTINCT] <nome-da-coluna>)
  - Exemplo:
    - select min(SALARIO) from EMPREGADO
      - Seleciona o menor salário de todos os empregados

Funções de agregação

- COUNT (contagem)
  - Retorna o número de linhas ou de colunas do resultado. Pode vir a ser distinta com a utilização do operador DISTINCT. Sintaxe:
    - count(*)
      - Retorna o número de linhas do resultado.
    - count(<nome-da-coluna>) ou count(ALL <nome-da-coluna>)
      - Retorna o número de colunas do resultado.
    - count(DISTINCT <nome-da-coluna>)
      - Retorna o número de colunas distintas do resultado.
  - Exemplo:
    - select count(*) from EMPREGADO
      - Seleciona o número de empregados.
    - select count(PERCENTUAL_COMISSAO) from EMPREGADO
      - Seleciona o número de empregados que possuem percentual de comissão, pois valores nulos não são avaliados.
    - select count(DISTINCT ULTIMO_NOME) from EMPREGADO
      - Seleciona o número dos sobrenomes distintos de empregados.

Agrupando dados com GROUP BY

- Agrupando os dados utilizando GROUP BY
  - As funções de agregação são úteis, porém seu uso comum é vinculá-las a uma cláusula GROUP BY.
  - A cláusula GROUP BY como o próprio nome diz trabalha nos dados que estão agrupados. Como já visto, quando a mesma não é especificada, o agrupamente padrão é por todo o resultado.
  - Por exemplo, seria interessante buscar o menor e o maior salário por cada departamento, a média salarial por função, o número de países por cada região, entre outros. Para isto, devemos aplicar um função de agregação e agrupá-la por algum campo.

Agrupando dados com GROUP BY

- Buscar o maior e o menor salário por cada departamento
  - select ID_DEPARTAMENTO, MIN(SALARIO), MAX(SALARIO) from RH.EMPREGADO group by ID_DEPARTAMENTO order by 1 nulls first

```
ID_DEPARTAMENTO MIN(SALARIO) MAX(SALARIO)
--------------- ------------ ------------
                        7000         7000
             10         4400         4400
             20         6000        13000
             30         2500        11000
             40         6500         6500
             50         2100         8200
             60         4200         9000
             70        10000        10000
             80         6100        14000
             90        17000        24000
            100         6900        12000
            110         8300        12000
```

Agrupando dados com GROUP BY

- Buscar a média salarial por função.
  - select ID_FUNCAO, AVG(SALARIO) from RH.EMPREGADO group by ID_FUNCAO order by 1 nulls first

```
ID_FUNCAO  AVG(SALARIO)
---------- ------------
AC_ACCOUNT         8300
AC_MGR            12000
AD_ASST            4400
AD_PRES           24000
...
SA_MAN            12200
SA_REP             8350
SH_CLERK           3215
ST_CLERK           2785
ST_MAN             7280
```

Agrupando dados com GROUP BY

- Buscar o número de países por região
  - select ID_REGIAO, count(ID_PAIS) from RH.PAIS group by ID_REGIAO order by 1

```
 ID_REGIAO COUNT(ID_PAIS)
---------- --------------
         1              8
         2              5
         3              6
         4              6
```

Limitando com HAVING

- Limitando dados agrupados com HAVING
  - Funções de agregação não podem ser usadas na cláusula WHERE, porém em alguns casos você deseja agrupar os dados e aplicar um certo filtro no agrupamento dos dados.
  - Este filtro não pode ser feito na cláusula WHERE, pois ele é um filtro sobre as linhas agrupadas.
  - Por exemplo, exibir a média salarial dos empregados de cada departamento cuja média salarial seja superior a 8.000,00. O filtro não é sobre a tabela de DEPARTAMENTO, mas sim, sobre o agrupamento feito sobre esta tabela.

Limitando com HAVING

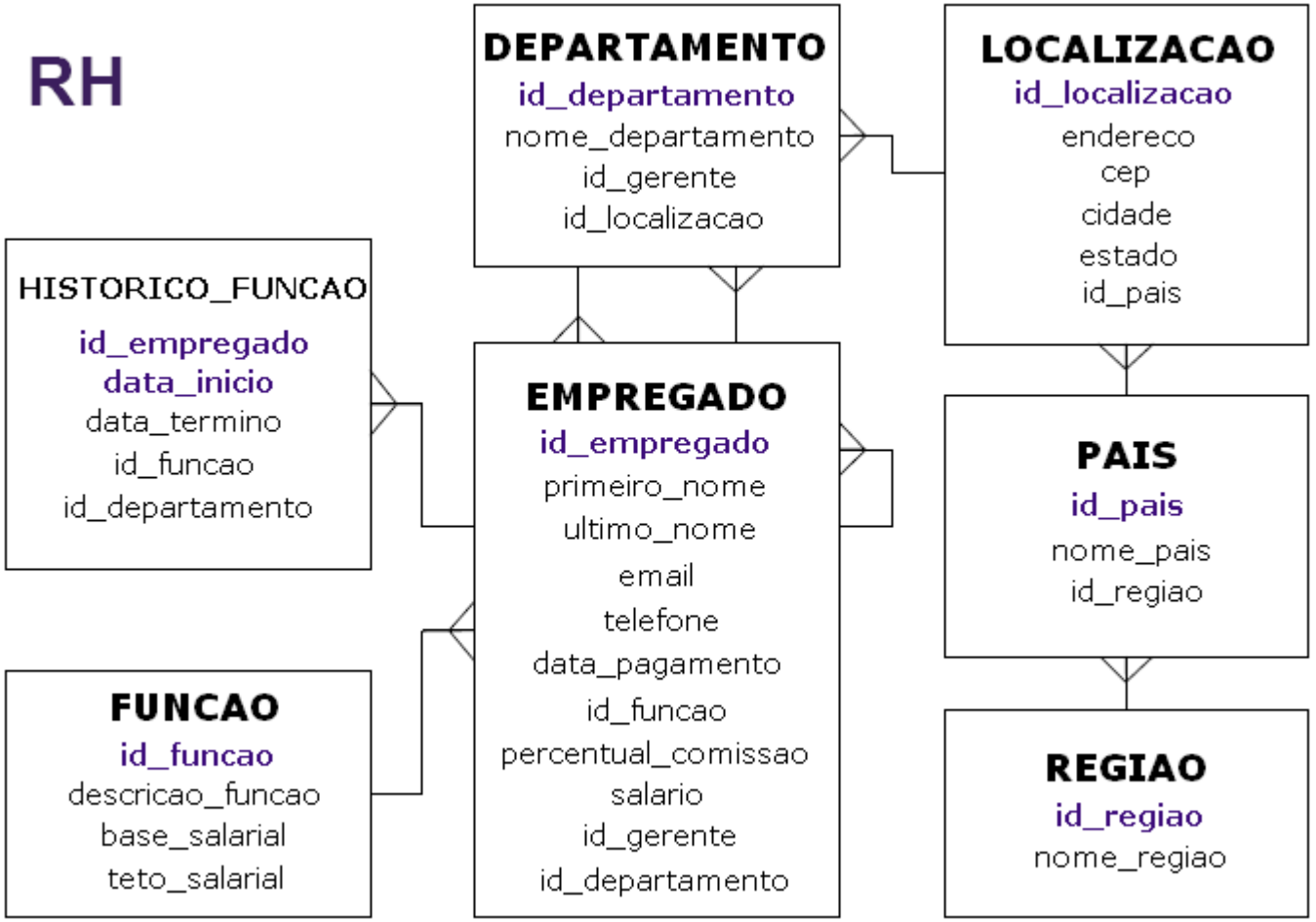
- Buscar a média salarial dos empregados que não possuem percentual de comissão de cada departamento cuja média salarial seja superior ou igual a 10.000,00.
  - select ID_DEPARTAMENTO, avg(SALARIO) from RH.EMPREGADO
    where PERCENTUAL_COMISSAO is null
    group by ID_DEPARTAMENTO
    having avg(SALARIO) >= 10000.00
    order by avg(SALARIO)

```
ID_DEPARTAMENTO AVG(SALARIO)
--------------- ------------
             70        10000
            110        10150
             90   19333.3333
```

  - Observe como a ordenação pode ser feita pelo resultado da função de agregação.

Aula 06

Esquema RH (tradução do HR)



Consultas em múltiplas tabelas

- Um banco de dados possui várias tabelas que armazenam dados. Nós já aprendemos a escrever consultas simples que selecionam informações de uma única tabela.
- A capacidade de junção de duas ou mais tabelas e o acesso às informações é a principal força dos banco de dados relacionais.
- Usando a instrução SELECT podemos escrever consultas avançadas que satisfazem os requisitos dos usuários.
- O foco deste material é vermos como escrever consultas avançadas que satisfazem os requisitos dos usuários.

Consultas em múltiplas tabelas

- A instrução SELECT possui as seguintes cláusulas obrigatórias:
  - SELECT
    - possui uma lista de colunas, expressões, funções e afins.
  - FROM
    - diz em qual(is) tabela(s) irá buscar as informações desejadas.
      - Até então, aprendemos a buscar informações de uma única tabela, porém isto agora não mais será uma limitação.

Consultas em múltiplas tabelas

- Para consultar dados de mais de uma tabela, você precisa identificar colunas comuns entre as duas tabelas.
  - Um exemplo seria a coluna ID_DEPARTAMENTO na tabela de EMPREGADO que corresponde a coluna de mesmo nome (ID_DEPARTAMENTO) na tabela de DEPARTAMENTO.

- Na cláusula WHERE, você define a relação entre as tabelas listadas na cláusula FROM usando operadores de comparação.
  - Normalmente o operador de igualdade (=) é o utilizado.

- Veremos que ao invés de utilizarmos o WHERE, podemos indicar a correspondência entre duas tabelas utilizando a cláusula JOIN.
  - A consulta entre múltiplas tabelas sem correspondência ou colunas em comum é conhecida como produto cartesiano e será abordada à frente.

Junções Simples

- Uma junção simples é conhecida como *inner join*, pois ela retorna somente as linhas que satisfazem as condições de junção.
  - Um exemplo seria a busca do código e nome dos departamentos e suas respectivas cidades. Esta informação não consta em uma única tabela, bastando uma junção simples.
    - select RH.DEPARTAMENTO.ID_DEPARTAMENTO,
      RH.DEPARTAMENTO.NOME_DEPARTAMENTO,
      RH.LOCALIZACAO.CIDADE
      from RH.DEPARTAMENTO, RH.LOCALIZACAO
      where RH.DEPARTAMENTO.ID_LOCALIZACAO = RH.LOCALIZACAO.ID_LOCALIZACAO
  - O resultado é algo como abaixo:

```
ID_DEPARTAMENTO NOME_DEPARTAMENTO              CIDADE
--------------- ------------------------------ ------------------------------
             10 Administration                 Seattle
             20 Marketing                      Toronto
            ... ...                            ...
            270 Payroll                        Seattle
```

Junções Complexa

- Uma junção complexa é uma junção simples que contém na cláusula WHERE, além da condição de junção, uma outra condição para filtrar as linhas retornadas.
  - Um exemplo seria a busca do código e nome dos departamentos e suas respectivas cidades fora dos Estados Unidos.
    - select RH.DEPARTAMENTO.ID_DEPARTAMENTO,
      RH.DEPARTAMENTO.NOME_DEPARTAMENTO,
      RH.LOCALIZACAO.CIDADE
      from RH.DEPARTAMENTO, RH.LOCALIZACAO
      where RH.DEPARTAMENTO.ID_LOCALIZACAO = RH.LOCALIZACAO.ID_LOCALIZACAO
      and RH.LOCALIZACAO.ID_PAIS != 'US'
  - O resultado é algo como abaixo:

```
ID_DEPARTAMENTO NOME_DEPARTAMENTO              CIDADE
--------------- ------------------------------ ------------------------------
             20 Marketing                      Toronto
            ... ...                           ...
             80 Sales                          Oxford
```

Criando *aliases* para tabelas

- Antes de iniciar nosso estudo sobre consultas em múltiplas tabelas, é indispensável o conhecimento sobre “apelidar” tabelas.

- Assim como em colunas, tabelas também pode ter “apelidos”.

- Observe a consulta do *slide* anterior reescrita usando “apelidos”, como o seguinte:

```
select d.ID_DEPARTAMENTO, d.NOME_DEPARTAMENTO, l.CIDADE
        from RH.DEPARTAMENTO d, RH.LOCALIZACAO l
where d.ID_LOCALIZACAO = l.ID_LOCALIZACAO
        and l.ID_PAIS != 'US'
```

- Apelidos de tabela aumentam a legibilidade da consulta. Eles são muito utilizados para reduzir grandes nomes de tabelas em curtos apelidos.

Sintaxe ANSI

- A diferença entre a sintaxe de junção tradicional do Oracle e a sintaxe ANSI/ISO SQL1999 é que na ANSI, o tipo de junção é especificado explicitamente na cláusula FROM.

- Usar a sintaxe ANSI é mais clara e é mais recomendada do que a sintaxe de junção tradicional do Oracle. As junções simples podem ter as seguintes formas:
  - NATURAL JOIN
    - <nome-da-tabela> NATURAL [INNER] JOIN <nome-da-tabela>
  - JOIN ... USING
    - <nome-da-tabela> [INNER] JOIN <nome-da-tabela> USING (<colunas>)
  - JOIN ... ON
    - <nome-da-tabela> [INNER] JOIN <nome-da-tabela> ON <condição>

NATURAL JOIN

- No *natural join*, a junção é realizada por todas as colunas que possuem o mesmo nome em ambas as tabelas.

- Neste tipo de junção não se deve qualificar o nome das colunas com o nome da tabela ou com o "apelido" da tabela.

- Sintaxe:
  - <nome-da-tabela> NATURAL [INNER] JOIN <nome-da-tabela>

- Vamos refazer o exemplo da busca do código e nome dos departamentos e suas respectivas cidades.
  - select ID_DEPARTAMENTO, NOME_DEPARTAMENTO, CIDADE
    from RH.DEPARTAMENTO **natural join** RH.LOCALIZACAO

JOIN ... USING

- Neste tipo de junção, a junção é realizada pelas colunas indicadas e que possuem o mesmo nome em ambas as tabelas.
- A cláusula USING especifica os nomes das colunas que devem ser utilizadas para realizar a junção das tabelas.
- Também neste tipo de junção não se deve qualificar o nome das colunas com o nome da tabela ou com o “apelido” da tabela.
- Sintaxe:
  - <nome-da-tabela> [INNER] JOIN <nome-da-tabela> USING (<colunas>)

JOIN ... USING

- Vamos refazer o exemplo da busca do código e nome dos departamentos e suas respectivas cidades.
  - select ID_DEPARTAMENTO, NOME_DEPARTAMENTO, CIDADE
    from RH.DEPARTAMENTO **join** RH.LOCALIZACAO
    using (ID_LOCALIZACAO)

JOIN ... ON

- Quando não se tem nomes de colunas comuns entre tabelas para fazer uma junção ou deseja-se especificar condições de junções arbitrárias, este é o tipo de junção a ser utilizado.

- Neste tipo de junção pode-se e deve-se qualificar o nome das colunas com o nome da tabela ou com o “apelido” da tabela.

- Sintaxe:
  - <nome-da-tabela> [INNER] JOIN <nome-da-tabela> ON <condição>

JOIN ... ON

- Vamos refazer o exemplo da busca do código e nome dos departamentos e suas respectivas cidades.
  - select d.ID_DEPARTAMENTO, d.NOME_DEPARTAMENTO, l.CIDADE
    from RH.DEPARTAMENTO d **join** RH.LOCALIZACAO l
    on (d.ID_LOCALIZACAO = l.ID_LOCALIZACAO)

Junções em múltiplas tabelas

- Uma junção em múltiplas tabelas é uma junção em mais de duas tabelas.

- Na sintaxe ANSI, as junções são realizadas da esquerda para a direita. A primeira condição de junção pode referenciar colunas somente da primeira e segunda tabelas; a segunda condição de junção da primeira, segunda e terceira e assim por diante.

Junções em múltiplas tabelas

- Vamos fazer o exemplo da busca do primeiro nome do empregado, nome do seu departamento, nome da cidade do seu departamento.
  - select e.PRIMEIRO_NOME, d.NOME_DEPARTAMENTO, l.CIDADE
    from RH.EMPREGADO e
    JOIN RH.DEPARTAMENTO d
    on (e.ID_DEPARTAMENTO = d.ID_DEPARTAMENTO)
    JOIN RH.LOCALIZACAO l
    on (d.ID_LOCALIZACAO = l.ID_LOCALIZACAO)

- Perguntas:
  - Será que poderia utilizar NATURAL JOIN ou JOIN ... USING ao invés de JOIN ... ON?
  - Outra pergunta: Será que poderia misturar NATURAL JOIN, JOIN ... USING, JOIN ... ON e, até mesmo, junção tradicional em uma mesma consulta?

Junções cartesianas

- Uma junção cartesiana ocorre quando os dados são selecionados de duas ou mais tabelas e não há uma condição de junção especificada.

- Assim, é feita a junção de cada linha da primeira tabela com todas as linhas da segunda tabela.
  - Se a primeira tabela tem 5 linhas e a segunda tabela tem 10 linhas, será gerado um resultado de 50 linhas. Se for adicionada uma nova tabela de 10 linhas sem ainda especificar uma condição de junção, o resultado terá 500 linhas.

- Para se evitar um produto cartesiano, deve existir pelo menos **n-1** condições de junções para a junção de **n** tabelas.

Junções cartesianas

- Um exemplo seria buscar o nome da região e o nome do país daqueles países que começam com ‘I’.
  - select r.NOME_REGIAO, p.NOME_PAIS
    from RH.REGIAO r, RH.PAIS p
    where p.NOME_PAIS like 'I%'

    ou

    select r.NOME_REGIAO, p.NOME_PAIS
    from RH.REGIAO r **cross join** RH.PAIS p
    where p.NOME_PAIS like 'I%'

  - Observe o resultado no próximo *slide*.

Junções cartesianas

```
NOME_REGIAO               NOME_PAIS
------------------------- ----------------------------------------
Europe                    Israel
Americas                  Israel
Asia                      Israel
Middle East and Africa    Israel
Europe                    India
Americas                  India
Asia                      India
Middle East and Africa    India
Europe                    Italy
Americas                  Italy
Asia                      Italy
Middle East and Africa    Italy
```

- Observe que não foi citada nenhuma condição de junção e, portanto, cada uma das regiões será relacionada a todos os países.
  - Isto faz sentido? Por que?

- Existe um exemplo em que o produto cartesiano deva ser usado?

OUTER JOINS

- Até agora, nos vimos *inner joins*, que retornam somente as linhas correspondentes entre ambas as tabelas, e produtos cartesianos, que retornam uma combinação de todas as linhas de ambas as tabelas.

- Em alguns casos, deseja-se ver os dados de uma tabela, mesmo que não haja uma linha correspondente na tabela de junção.

- Para o problema acima, existe a junção do tipo *outer join*. Ele retorna valores baseados nas condições de *inner join*, como trás também linhas não relacionadas de um ou dos dois lados das tabelas.

OUTER JOINS

- Na sintaxe tradicional do Oracle, o símbolo + envolvido por parênteses, denota um *outer join* na consulta.

- Digite (+) próximo a coluna da tabela onde pode não haver linhas correspondentes.

- Um exemplo é buscar a lista de nomes de todos os países e cidades em que possuem departamentos localizados. Caso o país não tenha nenhum departamento localizado, deseja-se exibí-lo mesmo assim.
  - select p.NOME_PAIS, l.CIDADE from RH.PAIS p, RH.LOCALIZACAO l
    where p.ID_PAIS = l.ID_PAIS **(+)**

OUTER JOINS

- O exemplo da página anterior pode ser reescrito das seguintes formas:
  - select p.NOME_PAIS, l.CIDADE
    from RH.PAIS p **left outer join** RH.LOCALIZACAO l
    on p.ID_PAIS = l.ID_PAIS

  - select p.NOME_PAIS, l.CIDADE
    from RH.PAIS p **left outer join** RH.LOCALIZACAO l
    using (ID_PAIS)

  - select p.NOME_PAIS, l.CIDADE
    from RH.PAIS p **natural left outer join** RH.LOCALIZACAO l

OUTER JOINS

- Um exemplo inverso é buscar a lista de todas as cidades que possuem departamentos e os seus países. Caso a cidade não esteja vinculada a um país, deseja-se exibí-la mesmo assim.
  - select p.NOME_PAIS, l.CIDADE from RH.PAIS p, RH.LOCALIZACAO l
    where p.ID_PAIS **(+)** = l.ID_PAIS

OUTER JOINS

- O exemplo da página anterior pode ser reescrito das seguintes formas:
  - select p.NOME_PAIS, l.CIDADE
    from RH.PAIS p **right outer join** RH.LOCALIZACAO l
    on p.ID_PAIS = l.ID_PAIS

  - select p.NOME_PAIS, l.CIDADE
    from RH.PAIS p **right outer join** RH.LOCALIZACAO l
    using (ID_PAIS)

  - select p.NOME_PAIS, l.CIDADE
    from RH.PAIS p **natural right outer join** RH.LOCALIZACAO l

OUTER JOINS

- Um exemplo que envolva os dois lados, isto é, inverso é buscar a lista de todas as cidades que possuem departamentos e os todos os países. Caso a cidade não esteja vinculada a um país ou caso o país não tenha nenhum departamento localizado, deseja-se exibí-lo mesmo assim.
  - select p.NOME_PAIS, l.CIDADE from RH.PAIS p, RH.LOCALIZACAO l
    where p.ID_PAIS **(+)** = l.ID_PAIS **(+)**
    - **ERRO.** Não funciona. Deve-se utilizar o **full outer join.**
  - select p.NOME_PAIS, l.CIDADE
    from RH.PAIS p **full outer join** RH.LOCALIZACAO l
    using (ID_PAIS)
    - Se preferir, pode-se utilizar o NATURAL JOIN ou JOIN ... ON sem problemas.

SELF JOINS

- Um *self-join* é uma junção na própria tabela. Certamente um resultado de um auto-relacionamento.

- Um exemplo claro é o ID_GERENTE na tabela EMPREGADO que referencia o ID_EMPREGADO na própria tabela. Portanto, como fazer a junção, isto é, buscar o nome completo dos gerentes e de seus respectivos subordinados.
  - select **e**.PRIMEIRO_NOME || ' ' || **e**.ULTIMO_NOME as "NOME EMPREGADO",
    g.PRIMEIRO_NOME || ' ' || **g**.ULTIMO_NOME as "NOME GERENTE"
    from RH.EMPREGADO **e** INNER JOIN RH.EMPREGADO **g**
    ON **e**.ID_GERENTE = **g**.ID_EMPREGADO
    order by 1, 2

- Em um *self-join* pode ser utilizado NATURAL JOIN? E JOIN ... USING?

Aula 06 - Complementar

Esquema HOMEM x MULHER

Esquema HOMEM x MULHER

- Tabela HOMEM

```
  ID_HOMEM NOME_HOMEM                      ID_MULHER
---------- ------------------------------ ----------
        10 Anderson
        20 Jander                                  1
        30 Rogério                                 2
```

- Tabela MULHER

```
 ID_MULHER NOME_MULHER
---------- ------------------------------
         1 Edna
         2 Stefanny
         3 Cássia
```

INNER JOINS

- Selecionar os casamentos:
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h, MULHER m
    where h.ID_MULHER = m.ID_MULHER

- Resultado:

```
NOME_HOMEM                     NOME_MULHER
------------------------------ ------------------------------
Jander                         Edna
Rogério                        Stefanny
```

- Também pode ser feito com NATURAL JOIN, JOIN ... USING e JOIN ... ON conforme poderá ser visto no próximo *slide*.

INNER JOINS

- NATURAL JOIN
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h natural join MULHER m
- JOIN ... USING
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h join MULHER m
    USING (ID_MULHER)
- JOIN ... ON
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h join MULHER m
    ON (h.ID_MULHER = m.ID_MULHER)

Produto Cartesiano

- Simular todos os casamentos possíveis:
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h, MULHER m

    ou
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h cross join MULHER m

Produto Cartesiano

- Resultado:

```
NOME_HOMEM                     NOME_MULHER
------------------------------ ------------------------------
Anderson                       Edna
Anderson                       Stefanny
Anderson                       Cássia
Jander                         Edna
Jander                         Stefanny
Jander                         Cássia
Rogério                        Edna
Rogério                        Stefanny
Rogério                        Cássia
```

OUTER JOINS

- Selecionar os casamentos, caso não haja **homens** casados também é desejável exibí-los:
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h, MULHER m
    where h.ID_MULHER = m.ID_MULHER (+)

- Resultado:

```
NOME_HOMEM                     NOME_MULHER
------------------------------ ------------------------------
Anderson
Jander                         Edna
Rogério                        Stefanny
```

OUTER JOINS

- Selecionar os casamentos, caso não haja **mulheres** casadas também é desejável exibí-las:
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h, MULHER m
    where h.ID_MULHER (+) = m.ID_MULHER

- Resultado:

```
NOME_HOMEM                     NOME_MULHER
------------------------------ ------------------------------
Jander                         Edna
Rogério                        Stefanny
                               Cássia
```

OUTER JOINS

- Além da símbolo (+), um *outer join* pode ser feito por NATURAL OUTER JOIN, OUTER JOIN ... USING e OUTER JOIN ... ON.

- Casamentos e todos os **homens** mesmo que não estejam casados.
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h **natural left outer join** MULHER m

- Casamentos e todas as **mulheres** mesmo que não estejam casadas.
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h **natural right outer join** MULHER m

- Como exercício refaça as consultas acima usando OUTER JOIN ... USING e OUTER JOIN ... ON.

OUTER JOINS

- E se deseja-se selecionar todos casamentos e caso não hajam **homens** ou **mulheres** casados também é desejável exibí-los:
  - select h.NOME_HOMEM, m.NOME_MULHER
    from HOMEM h **natural full outer join** MULHER m

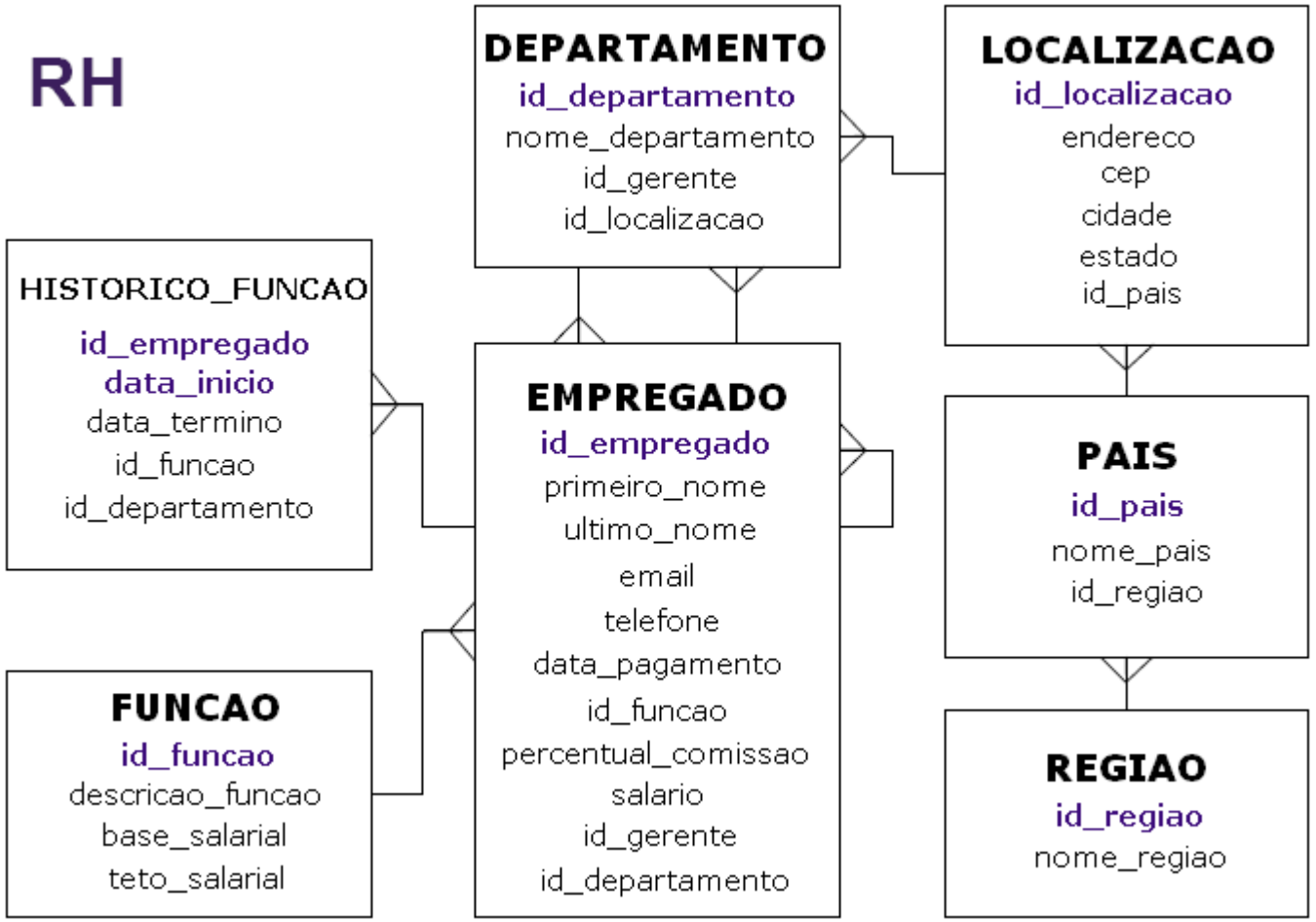
- Resultado:

```
NOME_HOMEM                     NOME_MULHER
------------------------------ ------------------------------
Anderson
Jander                         Edna
Rogério                        Stefanny
                               Cássia
```

- Como exercício refaça a consulta acima usando OUTER JOIN ... USING e OUTER JOIN ... ON. Observe que não é possível com (+).

Aula 07

Esquema RH (tradução do HR)



Operadores de Conjunto

- Operadores de conjuntos *(set operators)* podem ser usados para selecionar dados de tabelas múltiplas.

- Eles simplesmente combinam resultados de duas consultas em uma única consulta. Estas consultas são conhecidas como consultas compostas *(compound queries).*

Operadores de Conjunto

- São os operadores de conjunto:

Operador	Descrição
UNION	Retorna todas as linhas não repetidas selecionadas por cada consulta.
UNION ALL	Retorna todas as linhas, inclusive duplicadas, selecionadas por cada consulta.
INTERSECT	Retorna linhas selecionadas por ambas as consultas.
MINUS	Retorna linha únicas selecionadas pela primeira consulta, porém que não foram selecionadas na segunda consulta.

Operadores de Conjunto

- Para entender melhor vamos tomar como base as seguintes consultas:
  - Selecionar o último nome dos empregados que trabalham no departamento 90.
    - select e.ULTIMO_NOME
      from RH.EMPREGADO e
      where e.ID_DEPARTAMENTO = 90
  - Resultado:

```
ULTIMO_NOME
-------------------------
King
Kochhar
De Haan
```

  - Selecionar o último nome dos empregados que começam com ‘K’.
    - select e.ULTIMO_NOME
      from RH.EMPREGADO e
      where e.ULTIMO_NOME like 'K%'
  - Resultado:

```
ULTIMO_NOME
-------------------------
King
Kochhar
Khoo
Kaufling
King
Kumar
```

UNION

- Une todas as linhas não repetidas de cada consulta.
  - select e.ULTIMO_NOME from RH.EMPREGADO e
    where e.ID_DEPARTAMENTO = 90

    UNION

    select e.ULTIMO_NOME from RH.EMPREGADO e
    where e.ULTIMO_NOME like 'K%‘

  - Resultado:

```
ULTIMO_NOME
-------------------------
De Haan
Kaufling
Khoo
King
Kochhar
Kumar
```

UNION ALL

- Une todas as linhas, inclusive duplicadas, de cada consulta.
  - select e.ULTIMO_NOME from RH.EMPREGADO e
    where e.ID_DEPARTAMENTO = 90

    UNION ALL

    select e.ULTIMO_NOME from RH.EMPREGADO e
    where e.ULTIMO_NOME like 'K%‘

  - Resultado:

```
ULTIMO_NOME
-------------------------
King
Kochhar
De Haan
King
Kochhar
Khoo
Kaufling
King
Kumar
```

INTERSECT

- Faz a interseção entre as linhas de cada consulta, isto é, retorna linhas comuns de ambas as consultas.
  - select e.ULTIMO_NOME from RH.EMPREGADO e
    where e.ID_DEPARTAMENTO = 90

    INTERSECT

    select e.ULTIMO_NOME from RH.EMPREGADO e
    where e.ULTIMO_NOME like 'K%'

  - Resultado:

```
ULTIMO_NOME
-------------------------
King
Kochhar
```

MINUS

- Realiza a subtração entre as linhas da primeira consulta com as linhas da segunda consulta, isto é, retorna as linhas da primeira consulta que não aparecem na segunda consulta.
  - select e.ULTIMO_NOME from RH.EMPREGADO e
    where e.ID_DEPARTAMENTO = 90

    MINUS

    select e.ULTIMO_NOME from RH.EMPREGADO e
    where e.ULTIMO_NOME like 'K%'

  - Resultado:

```
ULTIMO_NOME
-------------------------
De Haan
```

SUBQUERIES

- Uma *subquery* é um consulta dentro de uma consulta.
- Quando você tem várias *subqueries*, a *subquery* mais interna é avaliada primeiramente.
- Podem ser usadas com todas as instruções DML.

SUBQUERIES

- Existem vários tipos de *subqueries* tais como:
  - nested subquery
    - Tradução: *subquery* aninhada
    - São *subqueries* utilizadas na cláusula WHERE.
  - correlated subquery
    - Tradução: *subquery* correlacionada
    - São *subqueries* que utilizam colunas da consulta "pai". Para cada linha processada da consulta "pai", a *correlated subquery* é avaliada novamente.
  - scalar subquery
    - Tradução: *subquery* escalar
    - São *subqueries* que podem ser usadas em qualquer lugar onde um nome de uma coluna ou um expressão podem ser utilizadas.

- Vamos ver alguns exemplos de *subqueries*.

SUBQUERIES

- Buscar o primeiro e o último nome e o salário do empregado com o maior salário.
  - select e.PRIMEIRO_NOME, e.ULTIMO_NOME, e.SALARIO
    from RH.EMPREGADO e
    where e.SALARIO =
    (select max(SALARIO) from RH.EMPREGADO)

  - Resultado:

```
PRIMEIRO_NOME        ULTIMO_NOME                    SALARIO
-------------------- ------------------------- ----------
Steven               King                            24000
```

  - Como fazer isto sem utilizar *subquery?*

SUBQUERIES

- Buscar o primeiro e o último nome dos empregados e o código do departamento cujo nome do departamento comece com a letra 'A'.
  - select e.PRIMEIRO_NOME, e.ULTIMO_NOME, e.ID_DEPARTAMENTO
    from RH.EMPREGADO e
    where e.ID_DEPARTAMENTO IN
    (select ID_DEPARTAMENTO from RH.DEPARTAMENTO
    where NOME_DEPARTAMENTO like 'A%')
  - Resultado:

```
PRIMEIRO_NOME        ULTIMO_NOME                    ID_DEP
-------------------- ------------------------- ----------
Jennifer             Whalen                             10
Shelley              Higgins                           110
William              Gietz                             110
```

  - Como fazer isto sem utilizar *subquery?*

SUBQUERIES

- Buscar o código do departamento, o primeiro e o último nome e o salário do empregado com o maior salário do seu departamento.
  - select e.ID_DEPARTAMENTO,
    e.PRIMEIRO_NOME, e.ULTIMO_NOME, e.SALARIO
    from RH.EMPREGADO e
    where e.SALARIO =
    (select max(ei.SALARIO) from RH.EMPREGADO ei
    where ei.ID_DEPARTAMENTO = e.ID_DEPARTAMENTO)
  - Resultado no próximo *slide*.

SUBQUERIES

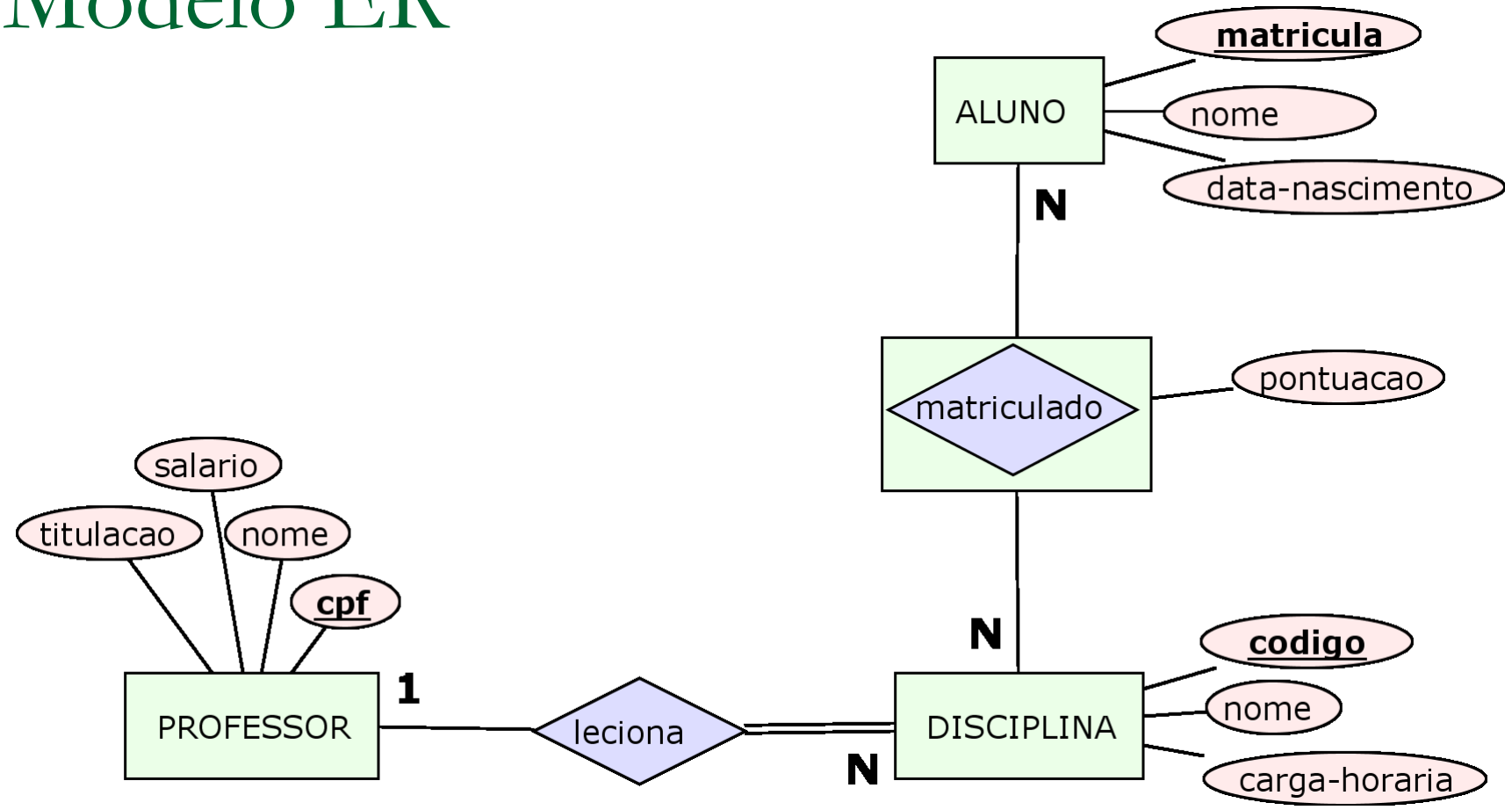
- Resultado:

ID_DEP	PRIMEIRO_NOME	ULTIMO_NOME	SALARIO
90	Steven	King	24000
60	Alexander	Hunold	9000
100	Nancy	Greenberg	12000
30	Den	Raphaely	11000
50	Adam	Fripp	8200
80	John	Russell	14000
10	Jennifer	Whalen	4400
20	Michael	Hartstein	13000
40	Susan	Mavris	6500
70	Hermann	Baer	10000
110	Shelley	Higgins	12000

- Como fazer isto sem utilizar *subquery?*

Aula 08

Esquema Acadêmico Simples
Modelo ER



Esquema Acadêmico Simples
Modelo Relacional (abstraindo domínios)

PROFESSOR (cpf, nome, salario, titulacao)

DISCIPLINA (codigo, nome, carga_horaria, cpf_professor)

DISCIPLINA [cpf_professor] ← PROFESSOR [cpf]

ALUNO (matricula, nome, data_nascimento)

ALUNO_DISCIPLINA (matricula_aluno, codigo_disciplina, pontuacao)

ALUNO_DISCIPLINA [matricula_aluno] ← ALUNO [matricula]

ALUNO_DISCIPLINA [codigo_disciplina] ←DISCIPLINA [codigo]

Instruções DML

- DML *(Data Manipulation Language)* é um subconjunto do SQL que é empregado para manipular dados.

Instrução	Propósito
INSERT	Adicionar linhas à uma tabela.
UPDATE	Modificar os valores armazenados em uma tabela.
MERGE	Atualizar ou inserir linhas de uma tabela em outra.
DELETE	Remover linhas de uma tabela.
SELECT FOR UPDATE	Bloqueia outras sessões de realizar DML nas linhas selecionadas.
LOCK TABLE	Bloqueia outras sessões de realizar DML em uma tabela.

  - Nos *slides* seguintes, abordaremos cada instrução acima.

Insert

- A instrução INSERT é usada para adicionar linha em uma ou mais tabelas.
  - As linhas podem ser adicionada com valores especificados ou podem ser criadas por dados existentes utilizando uma *subquery*.
- Sintaxes:
  - insert into <TABELA> (<lista-de-colunas>) values (<lista-de-valores>)
  - insert into <TABELA> values (<lista-de-valores>)
    - A lista de colunas é opcional, porém só funciona se na lista-de-valores forem inseridas todas as colunas da tabela na ordem de sua criação.
  - insert into <TABELA> [<lista-de-colunas>] **SELECT**...
    - Também pode ser inserido em uma tabela o resultado de uma consulta. Muito utilizada em migração de dados.
    - Lista de colunas é opcional.

Insert

- Exemplos:
  - insert into PROFESSOR (CPF,NOME,SALARIO,TITULACAO)
    values (11111111111, 'RICARDO TERRA', 1.99, 'ESPECIALISTA');
  - insert into PROFESSOR
    values (11111111111, 'RICARDO TERRA', 1.99, 'ESPECIALISTA');
  - insert into PROFESSOR (CPF,NOME,SALARIO,TITULACAO)
    select cpf_cliente, nome_cliente, 1000.00, 'GRADUADO' from CLIENTE
  - insert into PROFESSOR
    select cpf_cliente, nome_cliente, 1000.00, 'GRADUADO' from CLIENTE

Update

- A instrução UPDATE é usada para modificar linhas existentes em uma tabela.

- Sintaxe:
  - update <TABELA>
    set <coluna1> = <valor1>, <coluna2> = <valor2>, ...
    [where <condições>]

- Exemplos:
  - update PROFESSOR set SALARIO = 1.1*SALARIO

  - update PROFESSOR set SALARIO = 1.1*SALARIO
    where upper(NOME) like ' %TERRA%'

Delete

- A instrução DELETE é usada para remover linhas de uma tabela.

- Sintaxe:
  - delete [from] <TABELA>
    [where <condições>]

- Exemplos:
  - delete from PROFESSOR
    - exclui todos os professores

  - delete from PROFESSOR
    where SALARIO > 1.99
    - exclui os professores cujo salário seja superior a 1.99

Truncate

- Quando se deseja apagar todas as linhas de uma tabela, deve ser considerado o uso da instrução TRUNCATE.
- Assim como na instrução DELETE sem a cláusula WHERE, TRUNCATE irá remover todas as linhas e uma tabela.
- Entretanto, TRUNCATE não é uma DML, mas sim, uma DDL e, por isto, possui características diferentes de uma instrução DELETE que é uma instrução DML.
- Um usuário só poderá realizar o TRUNCATE de uma tabela de um outro esquema, se tiver permissão de DROP TABLE naquela tabela.

Truncate

- Para sua execução não poderá haver FKs ativas.
- Sintaxe:
  - truncate table ALUNO;
- Exemplo:
  - alter table ALUNO_DISCIPLINA disable constraint fk_aluno;
    truncate table ALUNO;
    - Depois não esqueça de reativar as FKs:
      - alter table ALUNO_DISCIPLINA enable constraint fk_aluno;

Truncate

- Algumas diferenças do TRUNCATE para o DELETE:
  - O TRUNCATE é mais rápido tanto em largas ou pequenas tabelas. O DELETE gerará informação de recuperação (como um desfazer), ao contrário do TRUNCATE que não gera este tipo de informação.
  - Não ativa *triggers*.
  - Com o TRUNCATE, o armazenamento da tabela e todos os índices são zerados.

Truncate

- Mesmo sendo uma instrução DDL, realizar o TRUNCATE de uma tabela é diferente de removê-la (DROP TABLE) e criá-la (CREATE TABLE) novamente.

- O TRUNCATE não faz:
  - Invalidar objetos dependentes.
  - Exclui índices, *triggers* ou restrições de integridade referencial.
  - Requer que privilégios sejam garantidos novamente.

Transação

- Uma transação representa uma unidade atômica de trabalho.
  - Todas as operações são aplicadas ou nenhuma delas.

- Uma transferência monetária é um exemplo tradicional de transação. Uma conta deve ser debitada e a outra creditada. Caso uma das operações apresente problema, nada é feito, senão todas as operações são efetuadas.

Transação

- Por padrão, uma transação é aberta logo que a primeira instrução seja efetuada. Ao criar um *script* é altamente recomendável inserir todas as operações em um bloco BEGIN e END.
  - O *SQL*Plus* possui a configuração AUTOCOMMIT ativada, para desativá-la, basta:
    - SET AUTOCOMMIT OFF

- Um bloco inicia com BEGIN e o termina com END.
  - A instrução COMMIT confirma todas as alterações.
  - A instrução ROLLBACK desfaz todas as alterações.

Transação

- Exemplo:

```
BEGIN
   update CONTA set SALDO - 50 where NUMERO = 100;

   update CONTA set SALDO + 50 where NUMERO = 200;

   COMMIT;

   EXCEPTION WHEN OTHERS THEN ROLLBACK;
END;
```

  - Caso as atualizações não gerem problemas, todas as operações serão confirmadas pelo COMMIT. Porém, caso haja qualquer problema, nenhuma alteração será realizada.

Selecionando linhas FOR UPDATE

- A instrução SELECT FOR UPDATE é utilizada para bloquear linhas específicas, impedindo que outras sessões alterem ou excluam estas linhas.
- Quando uma linha está bloqueada, outras sessões somente podem selecionar estas linhas, mas não podem alterar nem tampouco bloquear estas linhas.
- A sintaxe é similar a de um SELECT, porém com a inserção da expressão FOR UPDATE no final.
- O bloqueio permanece, independente de qualquer coisa, até que a transação encerre com um COMMIT ou com um ROLLBACK.

Selecionando linhas FOR UPDATE

- Exemplo:

```
declare l_professor PROFESSOR%rowtype;
BEGIN
   select * into linha_professor from PROFESSOR
      where TITULACAO = 'ESPECIALISTA' FOR UPDATE;

    --Várias outras operações

   COMMIT;

   EXCEPTION WHEN OTHERS THEN ROLLBACK;
END;
```

  - Nenhuma modificação ou exclusão nos professores especialistas pode ser feita por outra sessão até que haja um COMMIT ou ROLLBACK.

Bloqueando uma tabela

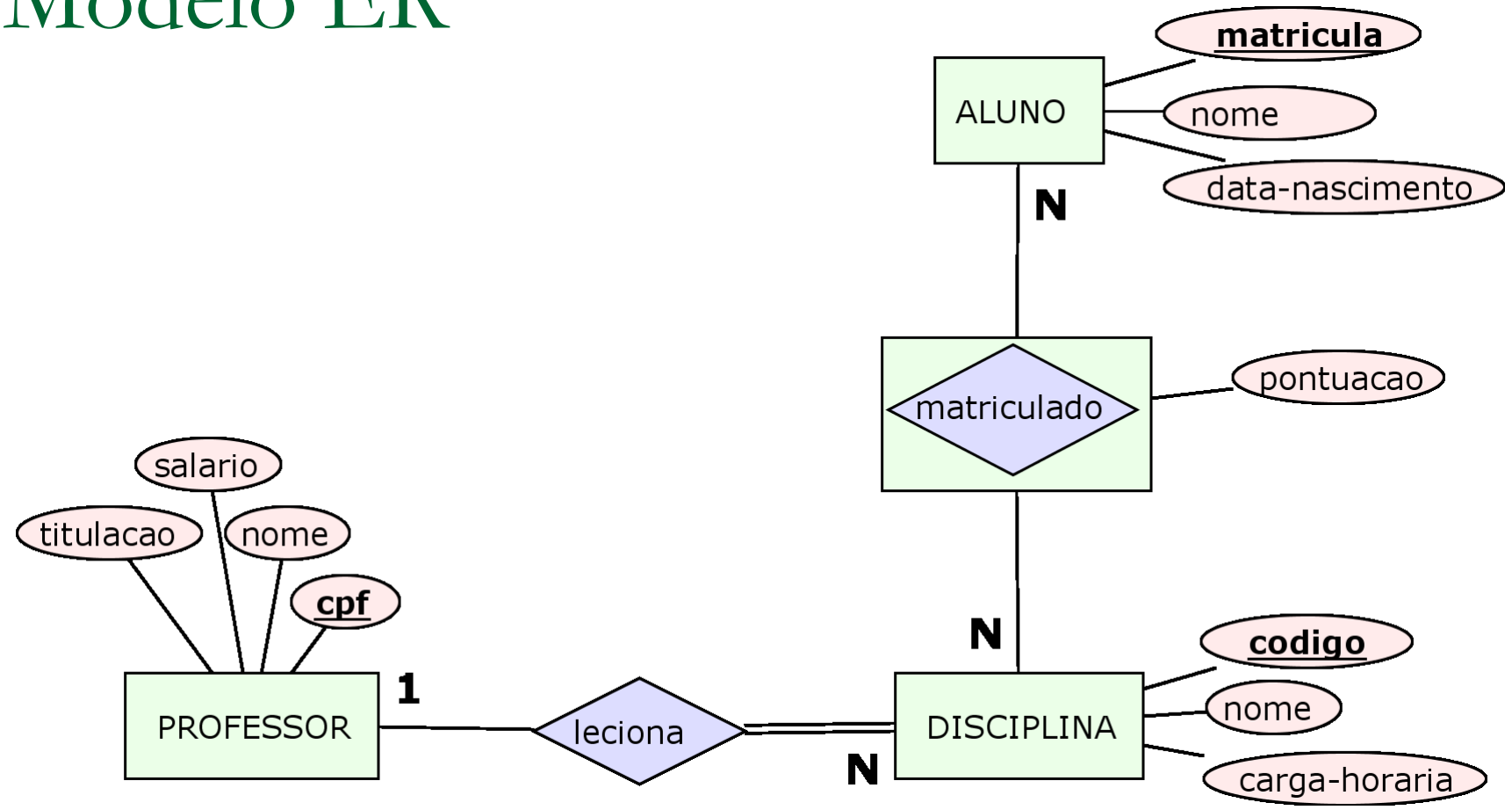
- A instrução LOCK é usada para bloquear toda uma tabela, impedindo outras sessões de executar a maioria ou todas as DML na tabela bloqueada.

- Sintaxe:
  - LOCK TABLE professor IN EXCLUSIVE MODE
    - O EXCLUSIVE MODE impede qualquer outra sessão de adquirir qualquer compartilhamento ou bloqueio nesta tabela.
    - Além do EXCLUSIVE MODE, existem diversos modos de bloquear uma tabela.

- Deadlocks
  - Ocorre quando duas transações aguardam um desbloqueio e cada uma delas está esperando o desbloqueio da outra. Oracle detecta?

Aula 09

Esquema Acadêmico Simples
Modelo ER



Esquema Acadêmico Simples
Modelo Relacional

PROFESSOR (cpf, nome, salario, titulacao)

dom(cpf) = numérico(11) NN

dom(nome) = alfabético(60) NN

dom(salario) = numérico(9,2) NN

dom(titulacao) = alfabético(40) NN

ALUNO (matricula, nome, data_nascimento)

dom(matricula) = numérico(6) NN

dom(nome) = alfabético(60) NN

dom(data_nascimento) = data NN

Esquema Acadêmico Simples
Modelo Relacional

DISCIPLINA (codigo, nome, carga_horaria, cpf_professor)

dom(codigo) = numérico(3) NN

dom(nome) = alfanumérico(40) NN

dom(carga_horaria) = numérico(3) NN

dom(cpf_professor) = numérico(11) NN

DISCIPLINA [cpf_professor] ← PROFESSOR [cpf]

ALUNO_DISCIPLINA (matricula_aluno, codigo_disciplina, pontuacao)

dom(matricula_aluno) = numérico(6) NN

dom(codigo_disciplina) = numérico(3) NN

dom(pontuacao) = numérico(3)

ALUNO_DISCIPLINA [matricula_aluno] ← ALUNO [matricula]

ALUNO_DISCIPLINA [codigo_disciplina] ←DISCIPLINA [codigo]

Tipos de dados

- Os tipos de dados existentes na linguagem SQL variam de acordo com a versão e o fabricante. O conjunto de tipos de dados evoluem junto com a evolução da informática. A primeira versão, surgida por volta de 1970, não possuia tipos de dados para armazenamento de informações multimídia como som, imagem, vídeo; tão comuns nos dias de hoje. A maioria dos SGBDs incorporaram esses novos tipos de dados às suas versões da linguagem SQL.

- Os principais tipos de dados do Oracle que iremos utilizar estão citados no próximo slide.
  - A explicação detalhada destes tipos pode ser encontrada na Aula 04.

Tipos de dados

- CHAR(<*size*>)
  - É uma string alfanumérica de tamanho fixo, que possui um tamanho máximo em bytes (2000).
- VARCHAR2(<*size*>)
  - É uma string alfanumérica de tamanho variável, que possui um tamanho máximo em bytes (4000).
- NUMBER(<*p*>,<*s*>)
  - O tipo NUMBER armazena números com uma precisão de *p* dígitos tendo uma escala de *s* dígitos reservados para as casas decimais.
- DATE
  - É o tipo utilizado para armazenar informação de data e hora.

Criando tabelas

- Você pode pensar em uma tabela como uma planilha com linhas e colunas. Esta é a estrutura que armazena dados em uma base de dados relacional.

- A tabela é criada com um nome para identificá-la e com colunas bem definidas com nomes e propriedades válidas, tais como tipo de dado e tamanho.

Criando tabelas

- A instrução CREATE TABLE é uma instrução com várias opções.
  - Exemplo da sintaxe mais simples de criação de tabela:

```
create table ALUNO (
    MATRICULA           number(6),
    NOME                varchar2(60),
    DATA_NASCIMENTO     date
)
```

- Você deve especificar o nome da tabela após o **create table**. Como não informou o usuário da tabela, o Oracle entende como sendo o seu.

- As definições de colunas estão delimitadas por um parênteses. A tabela criada acima possui três colunas, cada qual com seu nome, tipo de dado e tamanho.

Criando tabelas

- Algumas recomendações e regras para nomenclatura de nomes de tabelas e colunas:
  - Tente fazer nomes de tabelas e colunas o mais descritivo possível.
  - Nomes de tabelas e colunas são identificadores e podem:
    - ter no máximo 30 caracteres;
    - deve começar com letra e pode conter números;
    - também aceitam os caracteres **$** (cifrão), **_** (traço inferior) e **#** (sustenido).
  - Os nomes não são sensíveis a caixa, isto é, a tabela ALUNO pode ser referenciada como Aluno ou aluno ou AlUnO.
    - O *Oracle* converte tudo para maiúscula e grava no dicionário de dados.
    - Porém, se você cria a tabela ou coluna com o nome em aspas duplas ("), o *Oracle* a trata como sensível a caixa.

Criando tabelas

- Exemplo:
  - create table Abc (
    COL1 number
    );

    select * from ABC;

  - create table "Abc" (
    COL1 number
    );

    select * from ABC; --Não funciona

    select * from Abc; --Não funciona

    select * from "Abc";

Criando tabelas: valor padrão

- Na criação ou alteração de uma tabela, pode-se especificar valores padrões para colunas.
- O valor padrão especificado será usado quando você não especificar o valor para aquela coluna no momento de inserção de um registro.
- O valor padrão especificado deve seguir o tipo de dado e o tamanho da coluna em questão.
- Se o valor padrão não é explicitado, o valor padrão é implicitamente NULL.
- Geralmente constantes ou: SYSDATE, USER, USERENV e UID.

Criando tabelas: valor padrão

- Exemplo:

```
create table ALUNO (
    MATRICULA           number(6),
    NOME                varchar2(60)      DEFAULT 'ANÔNIMO',
    DATA_NASCIMENTO     date              DEFAULT SYSDATE
);

insert into ALUNO (MATRICULA) values (1);

select * from ALUNO;
```

```
 MATRICULA NOME                           DATA_NASC
---------- ------------------------------ ---------
         1 ANÔNIMO                        25-APR-08
```

Criando tabelas: anulabilidade

- Uma coluna pode ser obrigatória ou opcional.
- Se a coluna é obrigatória deve ser declarada como **NOT NULL** e, caso seja opcional, deve ser declarada como **NULL**. Se não for declarado, NULL será admitido.
- Exemplo:

```
create table ALUNO (
    MATRICULA           number(6)       NOT NULL,
    NOME                varchar2(60)    NOT NULL,
    DATA_NASCIMENTO     date            NULL
);
```

  - Neste caso a matrícula e o nome são obrigatórios, porém a data de nascimento é opcional.

Criando tabelas: chave primária

- Uma tabela sempre possui uma chave primária.
- Uma tabela só possui uma chave primária (pode ser mais de uma coluna) que é única e nunca nula.
- Existem várias sintaxes para criação de uma chave primária, porém vamos aprender a criá-la:
  - junto com a instrução **create table**;
  - utilizando a instrução **alter table**.

Criando tabelas: chave primária

- Exemplo da criação de uma chave primária junto ao **create table**:

```
create table ALUNO (
    MATRICULA           number(6)       NOT NULL,
    NOME                varchar2(60)    NOT NULL,
    DATA_NASCIMENTO     date            NULL,
    CONSTRAINT pk_aluno PRIMARY KEY (MATRICULA)
);
```

- Caso seja mais de uma coluna, basta separá-las por vírgula:
  - CONSTRAINT pk_tabela PRIMARY KEY (COLA, COLB, COLC)

Criando tabelas: chave primária

- Exemplo da criação de uma chave primária utilizando **alter table**:

```
create table ALUNO (
    MATRICULA           number(6)       NOT NULL,
    NOME                varchar2(60)    NOT NULL,
    DATA_NASCIMENTO     date            NULL
);

alter table ALUNO add
    CONSTRAINT pk_aluno PRIMARY KEY (MATRICULA);
```

  - Isto é, você cria a tabela sem chave primária e depois a altera inserido a restrição de chave primária.

Criando tabelas: comentários

- O propósito da tabela ou da coluna pode ser documentado na base de dados usando a instrução COMMENT.
- Exemplos:

```
COMMENT ON TABLE ALUNO IS
    'Tabela de Armazenamento de Alunos';

COMMENT ON COLUMN ALUNO.MATRICULA IS
    'Matrícula Única do Aluno';

COMMENT ON COLUMN ALUNO.NOME IS
    'Nome Completo do Aluno';

COMMENT ON COLUMN ALUNO.DATA_NASCIMENTO IS
    'Data de Nascimento do Aluno';
```

Criando tabelas: comentários

- Para ver os comentários de uma tabela:
  - select TABLE_NAME, **COMMENTS**
    from **USER_*TAB*_COMMENTS**
    where TABLE_NAME = 'ALUNO';

```
TABLE_NAME   COMMENTS
------------ --------------------------------
ALUNO        Tabela de Armazenamento de Alunos
```

Criando tabelas: comentários

- Para ver os comentários das colunas de uma tabela:
  - select TABLE_NAME, **COLUMN_NAME**, **COMMENTS**
    from **USER_*COL*_COMMENTS**
    where TABLE_NAME = 'ALUNO';

```
TABLE_NAME      COLUMN_NAME         COMMENTS
--------------- ------------------- ---------------------------
ALUNO           MATRICULA           Matrícula Única do Aluno
ALUNO           NOME                Nome Completo do Aluno
ALUNO           DATA_NASCIMENTO     Data de Nascimento do Aluno
```

Criando tabelas: a partir de outra tabela

- Você pode criar uma tabela utilizando uma consulta em uma ou mais tabelas ou visões existentes.
- O tipo de dado e o tamanho de cada coluna será determinado pelo resultado da consulta.
- Além do mais, os dados já virão populados na nova tabela.
- Sintaxe:
  - **CREATE TABLE** <nome-da-tabela> **AS SELECT** <consulta>

Criando tabelas: a partir de outra tabela

- Exemplo:
  - **CREATE TABLE** RH.REGIAO **AS**

    select REGION_ID as "ID_REGIAO", REGION_NAME as "NOME_REGIAO" from HR.REGIONS

- Quando você faz este tipo de criação, somente as restrições de anulabilidade (not null) são copiadas para a nova tabela, isto é, outras restrições e definições de valores padrões não são copiados.

Destruindo tabelas

- Destruir uma tabela é simples, bastando utilizar a instrução DROP TABLE.
- Sintaxe:
  - DROP TABLE <tabela> [CASCADE CONSTRAINTS]
- Quando você destroi uma tabela, os dados e as definições dela são removidas. Os índices, restrições, gatilhos (*triggers*) e privilégios também são destruídos.
- Uma vez destruída, a ação não pode ser desfeita.

Destruindo tabelas

- Caso existam restrições de integridade referencial de outras tabelas que referenciam a uma chave primária ou única desta tabela, você deverá especificar o CASCADE CONSTRAINTS.

- Em outras palavras, a tabela PROFESSOR não pode ser removida, pois a tabela de DISCIPLINA referencia à chave primária (CPF_PROFESSOR) desta tabela.

```
SQL> drop table PROFESSOR;
drop table PROFESSOR
           *
ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys

SQL> drop table PROFESSOR cascade constraints;
Table dropped.
```

Aula 10

Gerenciando restrições

- Restrições são criadas em um banco de dados para garantir uma regra de negócio no banco de dados e para especificar relacionamentos entre várias tabelas.

- Regras de negócio também podem ser garantidas pela utilização de gatilhos (*triggers)* e pelo código da aplicação.

- Restrições de integridade previnem que dados inconsistentes entrem no banco de dados.

Gerenciando restrições

- Oracle suporta cinco tipos de restrições de integridade:

NOT NULL	Previne valores nulos em colunas. Por padrão, o Oracle permite valores NULL em qualquer coluna.
PRIMARY KEY	Identifica unicamente cada linha de uma tabela e previne valores nulos. Uma tabela pode somente possuir uma única chave primária.
FOREIGN KEY	Estabelece um relacionamento pai-filho entre tabelas por utilização de colunas comuns. Uma chave estrangeira em uma tabela referencia a uma chave primária ou única de uma outra tabela.
UNIQUE	Garante a unicidade de valores para as colunas especificadas.
CHECK	Verifica se a condição especificada é satisfeita.

Criando restrições

- Existem várias sintaxes para criação de uma chave estrangeira, porém vamos aprender a criá-la:
  - junto com a instrução **create table**;
  - utilizando a instrução **alter table**.

- Se você não fornece um nome para as restrições, Oracle atribui um nome único gerado automaticamente que começa com "SYS_".
  - O nome é fornecido logo em seguida a palavra-chave CONSTRAINT conforme será visto na criação de restrições a frente.

Criando restrições: chave estrangeira

- Uma restrição de chave estrangeira *(foreign key)* garante que uma ou mais colunas em uma tabela tenham valores não-nulos correspondentes em alguma chave primária ou única no banco de dados.

- Isto é, uma chave estrangeira é uma ou mais colunas em uma tabela (tabela filha) que referenciam a uma chave primária ou uma chave única em uma outra tabela (tabela pai).

- Os tipos da coluna na tabela pai e na tabela filho devem coincidir, isto é, serem idênticos.

Criando restrições: chave estrangeira

- Exemplo da criação de uma chave estrangeira junto ao **create table**:

```
create table DISCIPLINA(
    CODIGO number(3) not null PRIMARY KEY,
    NOME varchar2(40) not null,
    CARGA_HORARIA number(3) not null,
    CPF_PROFESSOR number(11) not null,
    CONSTRAINT fk_professor
            FOREIGN KEY (CPF_PROFESSOR)
                    REFERENCES PROFESSOR(CPF)
);
```

- Caso seja mais de uma coluna, basta separá-las por vírgula:
  - … FOREIGN KEY (COLA,COLB) REFERENCES TABLE(COLA,COLB)

Criando restrições: chave estrangeira

- Exemplo da criação de uma chave estrangeira utilizando **alter table**:

```
create table DISCIPLINA(
    CODIGO number(3) not null PRIMARY KEY,
    NOME varchar2(40) not null,
    CARGA_HORARIA number(3) not null,
    CPF_PROFESSOR number(11) not null
);

ALTER TABLE DISCIPLINA ADD CONSTRAINT fk_professor
        FOREIGN KEY (CPF_PROFESSOR)
            REFERENCES PROFESSOR(CPF);
```

  - Isto é, você cria a tabela sem chave estrangeira e depois a altera inserido a restrição de chave estrangeira.

Criando restrições: chave estrangeira

- Independente da sintaxe de criação, existe uma cláusula opcional a ser inserida na sintaxe da restrição:
  - [ON DELETE {CASCADE | SET NULL}]

- Esta cláusula especifica a ação que deve ser tormada quando um registro na tabela pai é excluído e ainda existem registros filhos que o referenciam.
  - Você pode deletar todos os registros filhos (CASCADE) ou alterar a coluna de chave estrangeira para NULL (SET NULL).
  - Se a cláusula é omitida, o Oracle não permite a exclusão de um registro se existirem registros filhos. Você deve excluir os registros filhos primeiramente antes de excluir o registro pai. É o mais utilizado.

Criando restrições: chave estrangeira

- Exemplo 1:

```
alter table CIDADE add constraint FK_ESTADO
    foreign key (ID_ESTADO) references ESTADO (ID)
        ON DELETE CASCADE;
```

- Exemplo 2:

```
alter table CIDADE add constraint FK_ESTADO
    foreign key (ID_ESTADO) references ESTADO (ID)
        ON DELETE SET NULL;
```

- O que acontecerá no primeiro exemplo? E no segundo?

Criando restrições: chave única

- Uma restrição de chave única protege uma ou mais colunas em uma tabela, garantindo que registros tenham valores repetidos.

Criando restrições: chave única

- Exemplo da criação de uma chave única junto ao **create table**:

```
create table DISCIPLINA(
    CODIGO number(3) not null PRIMARY KEY,
    NOME varchar2(40) not null,
    CARGA_HORARIA number(3) not null,
    CPF_PROFESSOR number(11) not null,
    CONSTRAINT fk_professor
            FOREIGN KEY (CPF_PROFESSOR)
                    REFERENCES PROFESSOR(CPF),
    CONSTRAINT un_nome_disciplina UNIQUE (nome)
);
```

- Caso seja mais de uma coluna, basta separá-las por vírgula.

Criando restrições: chave única

- Exemplo da criação de uma chave estrangeira utilizando **alter table**:

```
create table DISCIPLINA(
    CODIGO number(3) not null PRIMARY KEY,
    NOME varchar2(40) not null,
    CARGA_HORARIA number(3) not null,
    CPF_PROFESSOR number(11) not null
);

ALTER TABLE DISCIPLINA ADD CONSTRAINT un_nome_disciplina
            UNIQUE (NOME);
```

  - Isto é, você cria a tabela sem chave única e depois a altera inserido a restrição de chave única.

Criando restrições: verificação

- Uma restrição de verificação *(check)* especifica uma condição booleana que deve ser sempre satisfeita.

- Pode referir aos valores das colunas, porém não pode realizar consultas, nem utilizar funções do ambiente (tal como SYSDATE, USER, etc) e pseudo-colunas (tal como ROWNUM, etc).

Criando restrições: verificação

- Exemplo da criação de uma restrição de verificação junto ao **create table**:

```
create table DISCIPLINA(
    CODIGO number(3) not null PRIMARY KEY,
    NOME varchar2(40) not null,
    CARGA_HORARIA number(3) not null,
    CPF_PROFESSOR number(11) not null,
    CONSTRAINT fk_professor
            FOREIGN KEY (CPF_PROFESSOR)
                    REFERENCES PROFESSOR(CPF),
    CONSTRAINT un_nome_disciplina UNIQUE (nome),
    CONSTRAINT ck_carga_horaria
            CHECK (carga_horaria between 80 and 160)
);
```

Criando restrições: verificação

- Exemplo da criação de uma restrição de verificação utilizando **alter table**:

```
create table DISCIPLINA(
    CODIGO number(3) not null PRIMARY KEY,
    NOME varchar2(40) not null,
    CARGA_HORARIA number(3) not null,
    CPF_PROFESSOR number(11) not null
);
```

```
ALTER TABLE DISCIPLINA ADD CONSTRAINT ck_carga_horaria
        CHECK (carga_horaria between 80 and 160)
```

  - Isto é, você cria a tabela sem restrição de verificação e depois a altera inserido a restrição de verificação.

Adiando verificação de restrições

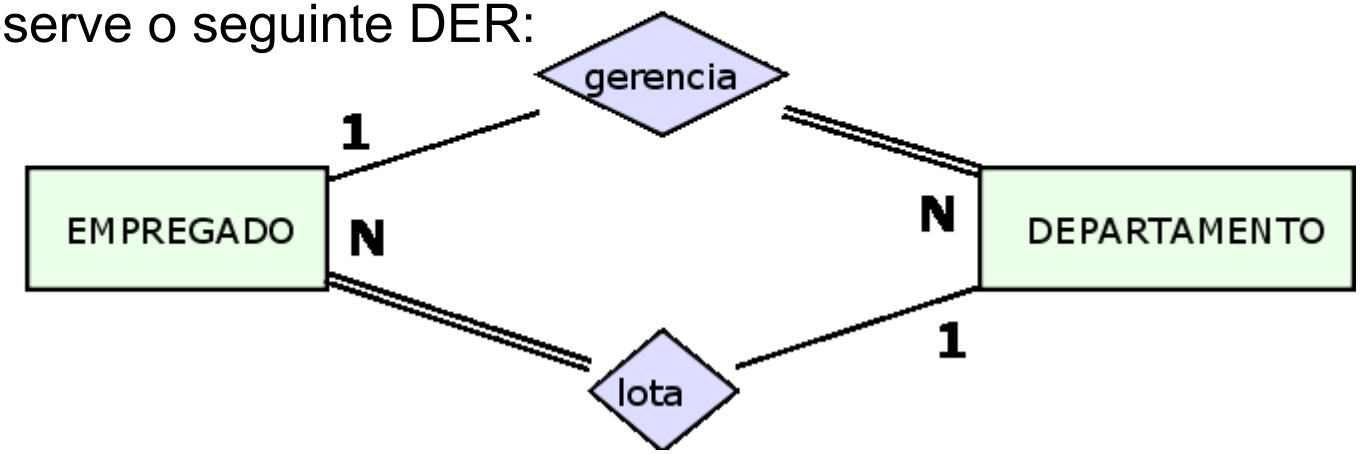
- Por padrão, o Oracle verifica se um dado conforma com a restrição logo que a instrução é executada.

- Oracle permite alterar este comportamento se a restrição é criada com a cláusula DEFERRABLE que modifica o comportamento da verificação de restrições.
  - INITIALLY IMMEDIATE indica que um dado é verificado logo após a instrução ser executada. É o padrão.
  - INITIALLY DEFERRED indica que a restrição deve ser verificada no final da transação, ie, no momento do *commit*.

- Aplica-se às restrições de chave primária ou de chave estrangeira.

Adiando verificação de restrições

- Observe o seguinte DER:



  - Um departamento só existe se possuir um empregado que o gerencie.
  - Um empregado só existe se possuir um departamento que o lote.

- Enfim, para incluir um empregado deve-se vinculá-lo a um departamento e para incluir um departamento deve-se indicar quem o gerencia. E se o banco de dados estiver vazio, como faremos para populá-lo?

Adiando verificação de restrições

- Observe a seguinte DML:

```
create table DEPARTAMENTO(
    CODIGO number(3) not null PRIMARY KEY,
    NOME varchar2(60) not null,
    CPF_GERENTE number(11) not null
);
create table EMPREGADO(
    CPF number(11) not null PRIMARY KEY,
    NOME varchar2(60) not null,
    CODIGO_DEPTO number(3) not null
);
```

Adiando verificação de restrições

- Observe a seguinte DML:

```
alter table DEPARTAMENTO add constraint fk_depto_gerente
    FOREIGN KEY (CPF_GERENTE) references EMPREGADO(CPF)
      INITIALLY DEFERRED DEFERRABLE;

alter table EMPREGADO add constraint fk_emp_depto
    FOREIGN KEY (CODIGO_DEPTO) references DEPARTAMENTO(CODIGO)
      INITIALLY DEFERRED DEFERRABLE;
```

Adiando verificação de restrições

- Observe o bloco transacional abaixo:

```
BEGIN
    insert into DEPARTAMENTO (CODIGO, NOME, CPF_GERENTE)
        values (1,'Departamento Qualquer',12345678901);

    insert into EMPREGADO (CPF, NOME, CODIGO_DEPTO)
        values (12345678901,'Gerente do Departamento',1);

    COMMIT;
END;
```

- Funcionará? Funcionaria sem a utilização da cláusula DEFERRABLE? Por quê?

Ativando e desativando restrições

- Quando se cria uma restrição, ela está automaticamente ativa (a não ser que você a crie utilizando a cláusula DISABLE).
- Você pode desativar uma restrição utilizando a cláusula DISABLE na instrução ALTER TABLE.
- Você pode desativar qualquer restrição especificando a cláusula DISABLE seguida do nome da restrição.
  - Pode-se também especificar a palavra-chave UNIQUE e o nome da(s) coluna(s) ou também somente especificando a palavra-chave PRIMARY KEY.
- Para desativar todas as chaves estrangeiras vinculadas a uma chave primária ou única, basta especificar CASCADE.
  - Mas CASCADE não funciona para a ativação.

Ativando e desativando restrições

- Exemplos de desativação:
  - ALTER TABLE professor **DISABLE** CONSTRAINT ck_salario;
    - Desabilita a verificação de salário.
  - ALTER TABLE disciplina **DISABLE** CONSTRAINT un_nome_disciplina;
    - Desabilita a restrição de unicidade do nome da disciplina.
  - ALTER TABLE disciplina **DISABLE** UNIQUE (NOME);
    - Idem anterior.
  - ALTER TABLE disciplina **DISABLE** PRIMARY KEY CASCADE;
    - Desabilita a verificação da chave primária da tabela DISCIPLINA e de todas as chaves estrangeiras vinculadas.

Ativando e desativando restrições

- Exemplos de ativação:
  - ALTER TABLE professor **ENABLE** CONSTRAINT ck_salario;
    - Ativa a verificação de salário.
  - ALTER TABLE disciplina **ENABLE** CONSTRAINT un_nome_disciplina;
    - Ativa a restrição de unicidade do nome da disciplina.
  - ALTER TABLE disciplina **ENABLE** UNIQUE (NOME);
    - Idem anterior.
  - ALTER TABLE disciplina **ENABLE** PRIMARY KEY;
    - Ativa a verificação da chave primária da tabela DISCIPLINA, mas não das chaves estrangeiras vinculadas. O CASCADE não funciona na ativação.

Destruindo restrições

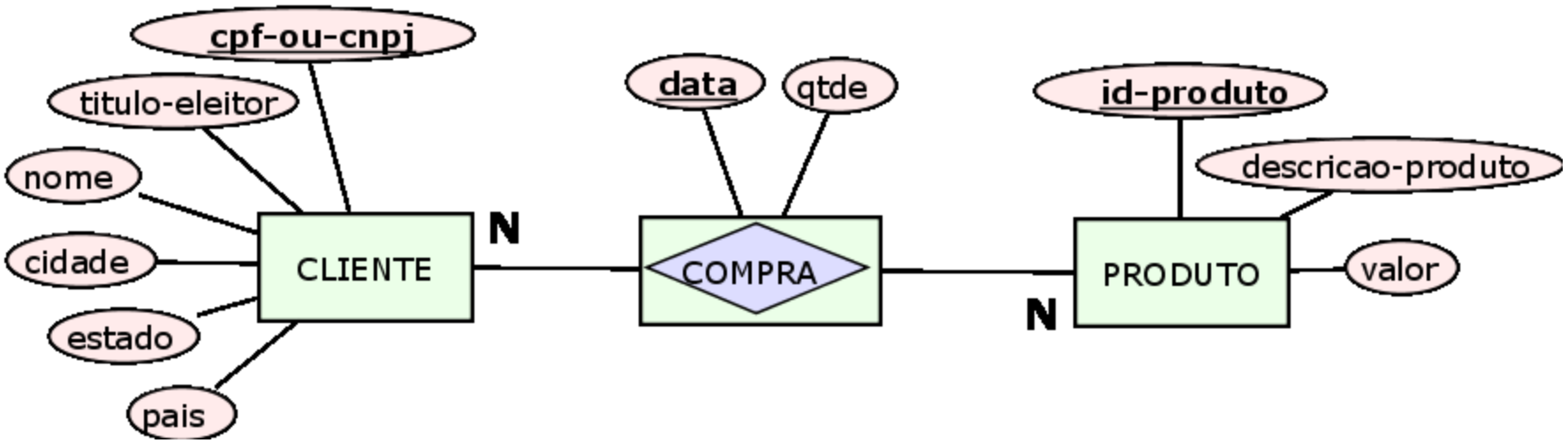
- Restrições são destruídas usando a instrução ALTER TABLE. Qualquer restrição pode ser destruída pela espeficação do seu nome.
  - ALTER TABLE disciplina DROP CONSTRAINT ck_carga_horaria;

- Restrições de chave única podem ser destruídas pela especificação do nome da coluna única e pode também destruir todas as chaves estrangeiras vinculadas pela utilização da cláusula CASCADE.
  - ALTER TABLE disciplina DROP UNIQUE (NOME) CASCADE;

- O mesmo pode ser feito com a chave primária:
  - ALTER TABLE disciplina DROP PRIMARY KEY CASCADE;

Aula 11

Esquema Venda
Modelo ER



Esquema Venda
Modelo Relacional

CLIENTE (cpf-ou-cnpj , titulo-eleitor , nome , cidade , estado , pais)

dom(cpf-ou-cnpj) = numérico(14) NN

dom(titulo-eleitor) = numérico(8) NN, com título de eleitor único

dom(nome) = alfabético(60) NN

dom(cidade) = alfabético(60) NN

dom(estado) = alfabético(2) NN, deve ser “MG”, “RJ” ou “SP”

dom(pais) = albabético(60)

PRODUTO (id-produto , descricao-produto , valor)

dom(id-produto) = numérico(2) NN

dom(descricao-produto) = alfabético(30) NN, com descrição única

dom(valor) = numérico(4,2) NN

Esquema Venda
Modelo Relacional

COMPRA (cpf-ou-cnpj , id-produto , data , qtde)

dom(cpf-ou-cnpj) = numérico(14) NN

dom(id-produto) = numérico(2) NN

dom(data) = data NN

dom(qtde) = numérico(1) NN, com o valor “1” como padrão

COMPRA [cpf-ou-cnpj] $\leftarrow^{r}$ CLIENTE [cpf-ou-cnpj]

COMPRA [id-produto] $\leftarrow^{r}$ PRODUTO [id-produto]

Renomeando tabelas

- A instrução RENAME é usada para renomear uma tabela e outros objetos, como visões, sinônimos ou sequências.

- Sintaxe:
  - RENAME *nome-antigo* TO *novo-nome;*

- Quando você renomeia uma tabela, o Oracle automaticamente transfere as restrições de integridade, índices e permissões da tabela antiga para a nova tabela.

- Exemplo:
  - RENAME funcionario TO empregado;

Modificando Tabelas

- Após a criação de uma tabela, existem vários motivos pelos quais você tenha que modificá-la.

- Você pode modificar uma tabela para alterar a definição de alguma coluna ou seu valor padrão, adicionar uma nova coluna, remover uma coluna existente ou, até mesmo, renomear colunas.

Renomeando colunas

- A instrução ALTER TABLE é utilizada para renomear colunas.

- Sintaxe:
  - **ALTER TABLE** nome-tabela
    RENAME COLUMN *nome-antigo* **TO** *novo-nome;*

- Exemplo:
  - **ALTER TABLE** empregado
    RENAME COLUMN pnome **TO** primeiro_nome;

Adicionando colunas

- Às vezes pode ser necessária adicionar colunas a uma tabela já existente.

- Sintaxe:
  - **ALTER TABLE** nome-tabela
    ADD *definição-da-coluna;*

- Quando uma nova coluna é adicionada, ela sempre é inserida como a última coluna da tabela.

- Exemplo:
  - **ALTER TABLE** empregado
    ADD *salario number(8,2) null;*

Adicionando colunas

- Pode-se adicionar várias colunas ao mesmo tempo, bastando apenas demarcar as colunas por parênteses e separar cada uma das colunas por vírgula.

- Exemplo:
  - **ALTER TABLE** empregado

    ADD (*salario number(8,2) null* **,** *bonus number(6,2) null* **)**

- Para as linhas existentes, o valor padrão da coluna será NULL.
  - Isto pode resultar em algum problema?

Adicionando colunas

- Se a coluna for NOT NULL, como pode-se preencher as linhas existentes com valor NULL?
  - Uma solução é adicionar a coluna com um valor padrão (o salário mínimo, por exemplo) que será preenchido para todas as linhas existentes, atualizar o salário dos empregados que não ganham o salário mínimo e.
    - **Ex.:**
      - **ALTER TABLE** empregado

        ADD salario number(8,2) DEFAULT 415.00 not null;

        ... (série de atualizações) ...

        ALTER TABLE empregado

        MODIFY salario DEFAULT NULL;

Adicionando colunas

- Uma outra solução seria adicioná-la como NULL, atualizar o salário de todos os empregados (colocando valores não-nulos) e então modificar a coluna para NOT NULL.
  - **Ex.:**
    - **ALTER TABLE** empregado

      ADD salario number(8,2) null;

      ... (série de atualizações) ...

      ALTER TABLE empregado

      MODIFY salario not null;

- Quais das soluções apresentadas você considera mais "adequada"?

Modificando colunas

- Às vezes pode ser necessária modificar colunas de uma tabela já existente.

- Sintaxe:
  - **ALTER TABLE** nome-tabela
    MODIFY nome-da-coluna *novos-atributos;*

- Se você omitir qualquer parte da definição da coluna (tipo de dado, valor padrão ou anulabilidade) estas partes não serão modificadas.

- Assim como na adição de colunas, você pode modificar várias colunas na mesma instrução usando a delimitação por parênteses e separação por vírgula.

Modificando colunas

- Exemplo:
  - Alterando o tipo de dado:
    - ALTER TABLE empregado **MODIFY salario number(10,2);**
  - Alterando a anulabilidade:
    - ALTER TABLE empregado **MODIFY salario null;**
  - Alterando o valor padrão:
    - ALTER TABLE empregado **MODIFY salario default 430.00;**
  - Alterando o tipo de dado, a anulabilidade e o valor padrão:
    - ALTER TABLE empregado
      MODIFY salario number(10,2) default 430.00 null;

Modificando colunas

- Existem algumas regras a serem seguidas para a alteração das definições de colunas:
  - Você pode aumentar o tamanho de uma coluna texto e a precisão de uma coluna numérica. Se a coluna for CHAR e a tabela possuir diversas linhas irá demandar um tempo superior, pois todos as linhas terão que serem preenchidas com espaços em branco.
  - Você pode reduzir o tamanho de uma coluna texto ou numérica **se** todos os dados couberem neste novo tamanho.
  - Para a modificação do tipo de dado, todos os valores da coluna devem estar NULL, exceto se a alteração for de CHAR para VARCHAR2 ou vice-versa.

Destruindo colunas

- Às vezes uma coluna não faz mais sentido e deseja-se retirá-la da tabela.

- Sintaxe:
  - **ALTER TABLE** nome-tabela
    DROP nome-da-coluna [CASCADE CONSTRAINTS]*;*

- A cláusula CASCADE CONSTRAINTS destrói também todas as restrições vinculadas a esta coluna.

- Assim como na adição e modificação de colunas, você pode destruir várias colunas na mesma instrução usando a delimitação por parênteses e separação por vírgula.

Destruindo colunas

- Destruir uma coluna de uma tabela com muitos registros pode levar um tempo considerável.

- O Oracle oferece uma sintaxe de desativação temporária de uma coluna, para que em uma melhor momento a coluna seja realmente destruida.

- Sintaxe:
  - **ALTER TABLE** nome-tabela
    SET UNUSED COLUMN nome-da-coluna
    [CASCADE CONSTRAINTS];

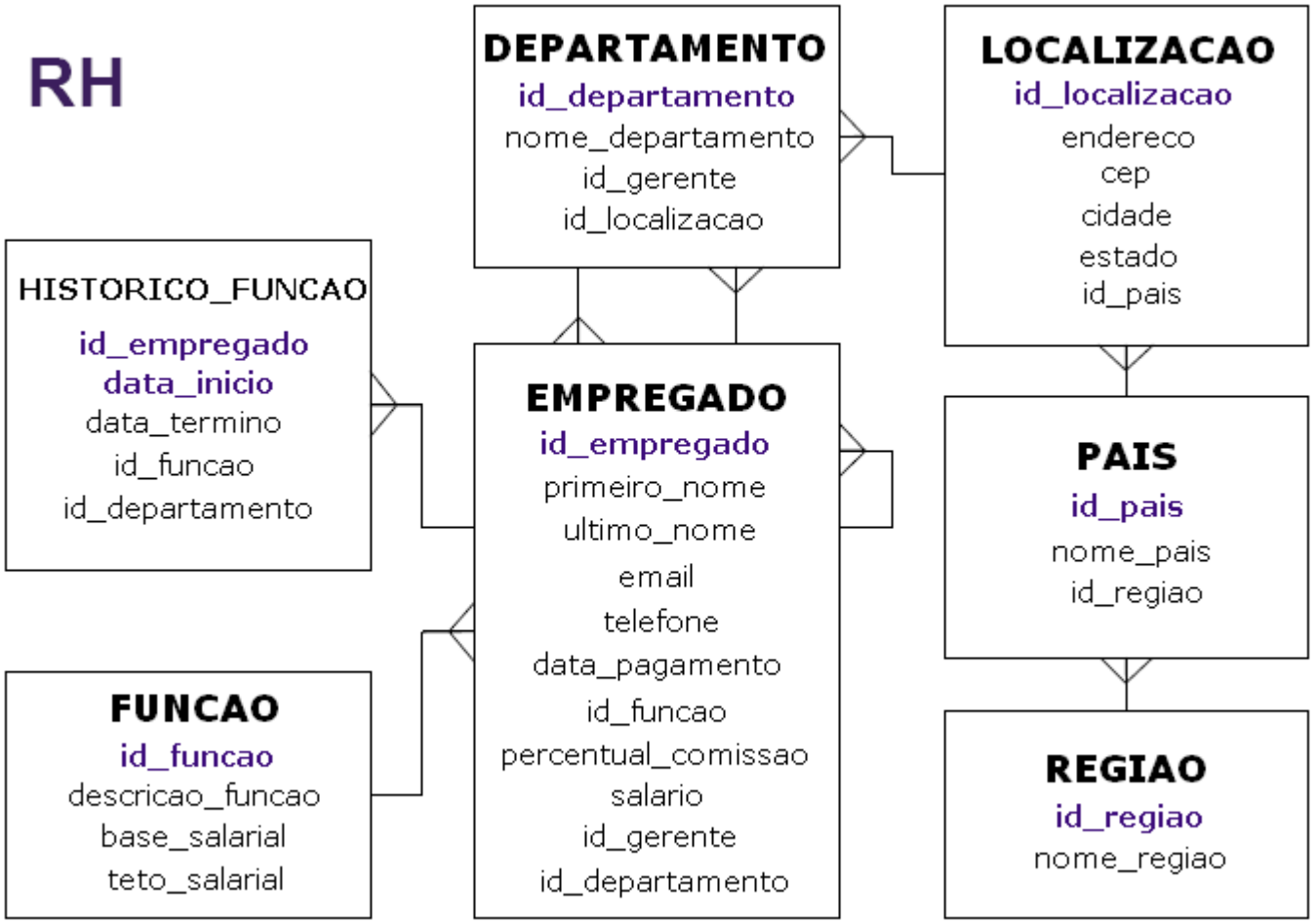
Destruindo colunas

- Exemplo:
  - **ALTER TABLE** empregado
    SET UNUSED COLUMN *titulo_eleitor;*

- Para realmente efetivar a destruição da coluna, deve-se fazer a seguinte instrução:
  - **ALTER TABLE** empregado **DROP UNUSED COLUMNS**;

Aula 12

Esquema RH (tradução do HR)



Criando visões

- Uma visão é uma representação lógica de dados de uma ou mais tabelas ou visões. Pode-se pensar em uma visão como uma consulta (*query*) armazenada no banco de dados.

- As tabelas que a visão referencia são conhecidas como *tabelas base*.

- Uma visão pode ser considerada uma consulta armazenada ou mesmo uma tabela virtual (não existe fisicamente).
  - Somente a consulta é armazenada no dicionário de dados, os dados atuais não são copiados de forma alguma (com exceções da visão materializada que será abordada à frente).
  - Isto indica que visão convencional não ocupa qualquer espaço de armazenamento, somente um espaço no dicionário de dados.

Criando visões

- Exemplo:
  - O departamento 90 é o departamento *Executivo*, portanto segue uma visão simplificada sobre os seus funcionários:
    - **create view EXECUTIVO as**

```
select PRIMEIRO_NOME || ' ' || ULTIMO_NOME as "NOME", EMAIL
from RH.EMPREGADO
where ID_DEPARTAMENTO = 90;
```

  - Basta a seguinte consulta para retornar todos os executivos:
    - select * from EXECUTIVO;

```
NOME                                EMAIL
----------------------------------- -------------------------
Steven King                         SKING
Neena Kochhar                       NKOCHHAR
Lex De Haan                         LDEHAAN
```

Criando visões

- Visões podem ser baseadas em consultas mais sofisticadas, envolvendo junções, funções, agrupamentos, *subqueries,* etc.
  - Observe a seguinte visão. Ela retorna a quantidade de funcionários por função de cada departamento daquelas funções que possuem mais de dois funcionários ordenadas pelo departamento, depois pela função e depois pela quantidade é uma consulta muito utilizada.
    - **create view REL_DEPTO_FUNCAO as**

```
select d.NOME_DEPARTAMENTO as "DEPARTAMENTO",
       f.DESCRICAO_FUNCAO as "FUNÇÃO",
       to_char(count(e.ID_EMPREGADO),'000') as "QUANTIDADE"
from RH.DEPARTAMENTO d inner join RH.EMPREGADO e
    using (ID_DEPARTAMENTO)
    inner join RH.FUNCAO f using (ID_FUNCAO)
group by d.NOME_DEPARTAMENTO, f.DESCRICAO_FUNCAO
having count(e.ID_EMPREGADO) > 2 order by 1, 2, 3
```

Criando visões

- Como é uma consulta largamente utilizada, agora basta realizar a seguinte consulta que retornará o resultado esperado:
  - select * from REL_DEPTO_FUNCAO;

```
DEPARTAMENTO         FUNÇÃO                    QUAN
-------------------- --------------------      ----
Finance              Accountant                005
IT                   Programmer                005
Purchasing           Purchasing Clerk          005
Sales                Sales Manager             005
Sales                Sales Representative      029
Shipping             Shipping Clerk            020
Shipping             Stock Clerk               020
Shipping             Stock Manager             005
```

Criando visões somente leitura

- Ao criar uma visão, instruções DML podem ser executadas nela. Por exemplo, pode-se inserir, atualizar, excluir dados de uma visão, o que irá modificar as tabelas base envolvidas.
  - Existem uma série de regras para que possa realizar instruções DML em uma visão.

- Porém, para evitar qualquer problema, pode-se criar uma visão somente leitura, bastando apenas utilizar a opção WITH READ ONLY.
  - As visões criadas com esta opção somente podem ser usadas em consultas, nenhuma outra instrução DML poderá ser utilizada.

Criando visões somente leitura

- Exemplo:
  - create view ASSALARIADO as
    select PRIMEIRO_NOME || ' ' || ULTIMO_NOME as "NOME",
    SALARIO as "SALARIO"
    from RH.EMPREGADO
    WITH READ ONLY

- A seguinte instrução não funcionará:
  - **update ASSALARIADO set SALARIO = 4000 where NOME = 'Sarah Bell'**

Criando visões materializadas

- Uma visão materializada é uma visão que armazena os resultados de uma consulta.
- Tem várias utilidades, mas, geralmente, são cópias locais de dados localizados remotamente ou são usadas para criar tabelas sumarizadas baseadas nos dados de uma certa tabela.
- Elas são cópias e devem possuir um mecanismo de atualização destes dados.
  - Geralmente a atualização ocorre de tempos em tempos com base na chave primária.

Criando visões materializadas

- Exemplo:
  - **create materialized view FUNCIONARIO**
    REFRESH FORCE
    START WITH SYSDATE
    NEXT SYSDATE + 1
    WITH PRIMARY KEY
    as select * from RH.EMPREGADO@*banco_dados_remoto*

- "@*banco_dados_remoto*" é um *dblink* para um outro SGBD Oracle.
  - Portanto, não funcionará. Foi apenas para dar mais realidade ao uso.
    - Para funcionar basta retirar o "@*banco_dados_remoto*"

Modificando visões

- Para alterar uma definição de uma visão, deve-se utilizar a instrução CREATE VIEW com a opção OR REPLACE.

- Quanto se usa a opção OR REPLACE, se a visão existir, ela será substituída pela nova definição. Caso contrário, uma nova visão será criada.

- Como exemplo, a visão assalariado tem formatação do salário:
  - create **OR REPLACE** view ASSALARIADO as

```
select PRIMEIRO_NOME || ' ' || ULTIMO_NOME as "NOME",
                to_char(SALARIO,'990,000.00') as "SALARIO"
from RH.EMPREGADO
```

Destruindo visões

- Para destruir uma uma visão, deve-se utilizar a instrução DROP VIEW.

- Ao destruir uma visão, a definição desta visão é removida do dicionário de dados e os privilégios e permissões da visão também serão removidos.
  - Outras visões que se referem a ela, tornarão-se inválidas.

- Exemplo:
  - **drop view ASSALARIADO**

Outras operações em visões

- Existem várias outras coisas que podem ser feitas em uma visão como:
  - Criar visões com erros (opção FORCE)
  - Criar restrições em visões
  - Utilizar a instrução ALTER VIEW para:
    - compilar ou recompilar
    - validar e invalidar
    - adicionar ou destruir restrições

Utilizando visões

- Uma visão pode ser utilizada na maioria dos locais onde uma tabela é utilizada, tal como em consultas e instruções DML.
  - Para verificar as visões do seu esquema basta:
    - select VIEW_NAME from USER_VIEWS
  - Para verificar as visões, que possui acesso, de um outro esquema deve ser realizado o seguinte comando:
    - select VIEW_NAME from ALL_VIEWS where OWNER = '<*nome-usuário*>'

Utilizando visões

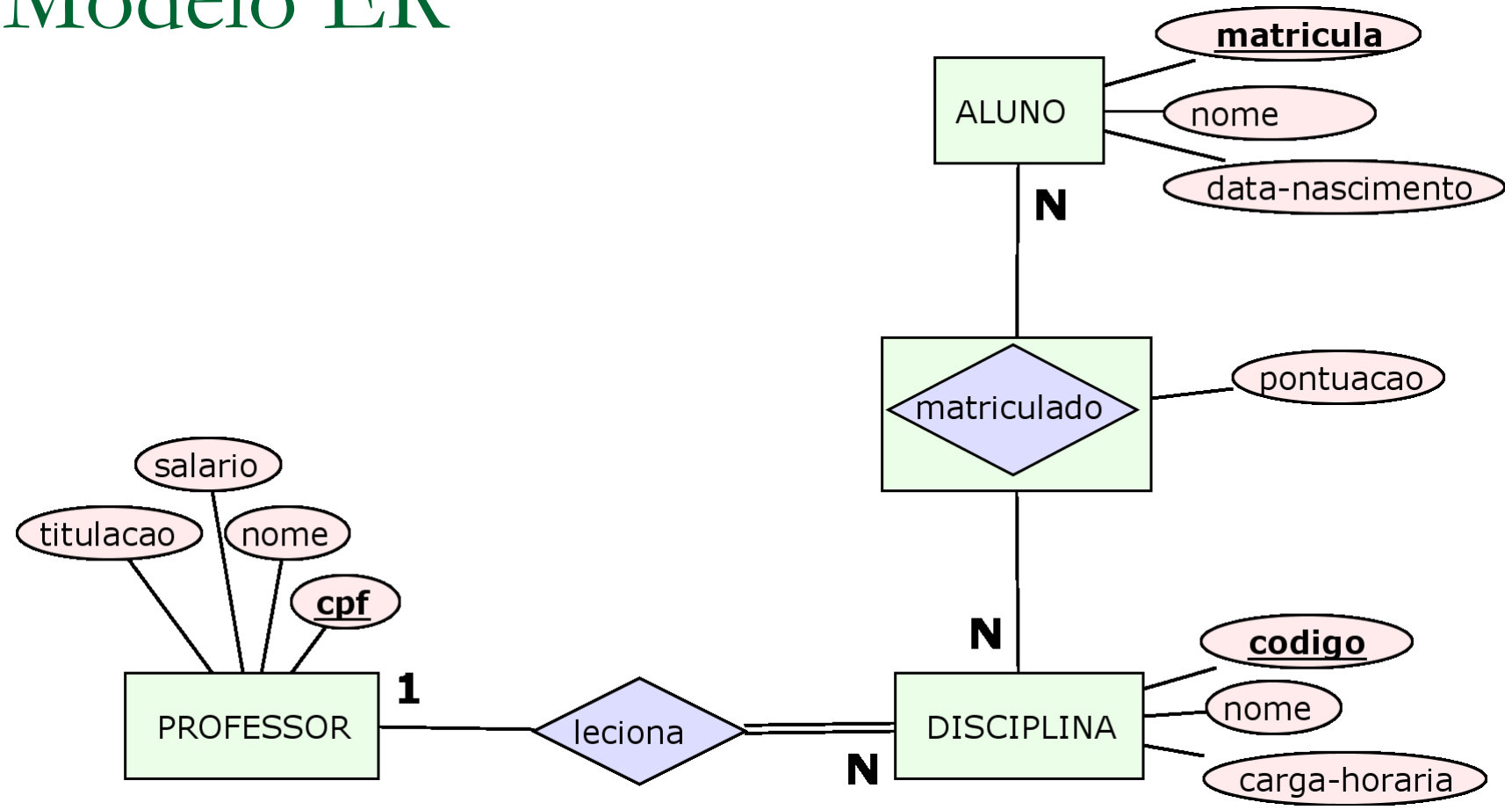
- Se certas condições são respeitadas, maioria das visões baseadas em uma única tabela e muitas das visões baseadas em junções podem ser utilizadas para inserir, atualizar e deletar dados da(s) tabela(s) base.
  - Para verificar quais instruções DML podem ser realizadas em uma dada visão, basta:
    - select COLUMN_NAME, UPDATABLE, INSERTABLE, DELETABLE
      from USER_UPDATABLE_COLUMNS
      where OWNER = '<*nome-usuário*>' and
      TABLE_NAME='<*nome-tabela*>'
- Todas as operações em visões afetam as tabelas base, contudo elas devem satisfazer toda restrição de integridade definida nas tabelas base.

Utilizando visões

- São utilizações comuns de visões:
  - **Representar um subconjunto de dados**
  - **Representar um superconjunto de dados**
  - **Ocultar junções complexas**
  - **Prover nomes mais entendíveis para colunas**
  - **Criação de uma camada entre a aplicação e a fonte de dados**

Aula 13

Esquema Acadêmico Simples
Modelo ER



Esquema Acadêmico Simples
Modelo Relacional

PROFESSOR (cpf, nome, salario, titulacao)

dom(cpf) = numérico(11) NN

dom(nome) = alfabético(60) NN

dom(salario) = numérico(9,2) NN

dom(titulacao) = alfabético(40) NN

ALUNO (matricula, nome, data_nascimento)

dom(matricula) = numérico(6) NN

dom(nome) = alfabético(60) NN

dom(data_nascimento) = data NN

Esquema Acadêmico Simples
Modelo Relacional

DISCIPLINA (codigo, nome, carga_horaria, cpf_professor)

dom(codigo) = numérico(3) NN

dom(nome) = alfanumérico(40) NN

dom(carga_horaria) = numérico(3) NN

dom(cpf_professor) = numérico(11) NN

DISCIPLINA [cpf_professor] ← PROFESSOR [cpf]

ALUNO_DISCIPLINA (matricula_aluno, codigo_disciplina, pontuacao)

dom(matricula_aluno) = numérico(6) NN

dom(codigo_disciplina) = numérico(3) NN

dom(pontuacao) = numérico(3)

ALUNO_DISCIPLINA [matricula_aluno] ← ALUNO [matricula]

ALUNO_DISCIPLINA [codigo_disciplina] ←DISCIPLINA [codigo]

Outros objetos do banco de dados

- Uma base de dados do Oracle contem muito mais do que simples tabelas e visões:
  - Sequências podem ser utilizadas para gerar chaves artificiais;
  - Sinônimos provêm apelidos (*alias*) para objetos;
  - Vários tipos de índices podem ser inseridos para melhorar o desempenho de consultas.

Sequências

- Uma sequência do Oracle é um gerador nominado de números sequênciais.
- Geralmente são utilizadas para chaves artificiais ou para ordenar linhas que de outra forma não teriam ordem.
- Assim como as restrições, as sequências existem somente no dicionário de dados.
- Elas podem ser configuradas para aumentar ou diminuir sem limites ou para repetir após alcançar um valor limite.

Criando sequências

- Sintaxe:
  - CREATE SEQUENCE [*esquema*].<*nome-da-sequência*>
    [START WITH integer]
    [INCREMENT BY integer]
    [MINVALUE integer | NOMINVALUE]
    [MAXVALUE integer | NOMAXVALUE]
    [CYCLE | NOCYCLE]
    [CACHE integer | NOCACHE]
    [ORDER | NOORDER]

Criando sequências

- **START WITH**
  - Define o valor inicial da sequência. Por padrão é MAXVALUE para sequências descendentes (-1) e MINVALUE para ascendentes (1).
- **INCREMENT BY**
  - Define a quantidade que aumenta ou diminui entre números. O padrão é 1. Para sequências descendentes, basta inserir um valor negativo.
- **MINVALUE**
  - Define o menor valor que a sequência irá gerar. O padrão é NOMINVALUE. (1 para sequências descendentes e -10^{26} para as ascendentes.
- **MAXVALUE**
  - Define o maior valor que a sequência irá gerar. O padrão é NOMAXVALUE. (-1 para sequências descendentes e 10^{27} para as ascendentes.

Criando sequências

- **CYCLE**
  - Configura a sequência para repetir números após alcançar o limite.
- **NOCYCLE**
  - Configura a sequência para **não** repetir números após alcançar o limite. Este é o padrão. Quando tentar gerar MAXVALUE+1, uma exceção será lançada.
- **CACHE**
  - Define o tamanho do bloco de números da sequência armazenados na memória. O padrão é 20.
- **NOCACHE**
  - Força o dicionário de dados a atualizar a sequência para cada número gerado, garantindo que nenhum espaço existirá entre os números gerados, porém o desempenho inferior.

Criando sequências

- **ORDER**
  - Garante que os números da sequência sejam gerados pela ordem de requisição.
- **NOORDER**
  - Não garante que os números da sequência sejam gerados pela ordem de requisição. Esta é a opção padrão.

Criando sequências

- Exemplo de uma sequência ascendente:
  - create sequence SQ_MATRICULA_ALUNO;

    corresponde a

  - create sequence SQ_MATRICULA_ALUNO
    START WITH 1 INCREMENT BY 1
    NOMINVALUE NOMAXVALUE
    NOCYCLE CACHE 20 NOORDER

- Exemplo de uma sequência descendente:
  - create sequence SQ_MATRICULA_ALUNO
    INCREMENT BY -1

Destruindo sequências

- Sintaxe:
  - drop sequence <*nome-da-sequência*>
- Exemplo:
  - drop sequence SQ_MATRICULA_ALUNO

Usando sequências

- Para acessar o próximo número na sequência, você simplesmente seleciona, usando a *pseudo*-coluna NEXTVAL.
  - Ex.:
    - select SQ_MATRICULA_ALUNO.NEXTVAL from dual;

- Para buscar o último número da sequência gerado na sua sessão, você seleciona a *pseudo*-coluna CURRVAL.
  - Ex.:
    - select SQ_MATRICULA_ALUNO.CURRVAL from dual;

  - Se você ainda não tiver gerado nenhum número da sequência, CURRVAL não funcionará.

Usando sequências

- Exemplo de usos:
  - Inserindo um novo aluno:
    - insert into ALUNO (matricula, nome, data_nascimento)
      values (**sq_matricula_aluno.nextval**, 'Aluno', sysdate)
  - Atualização de todas as matrículas:
    - update ALUNO set matricula = **sq_matricula_aluno.nextval**

Alterando sequências

- Você não pode simplesmente alterar a sequência e ajustar o seu NEXTVAL. Para isto existem as seguintes soluções:
  - Destruí-la e recriá-la.
    - Invalidará todos os objetos dependentes e perderá as permissões
  - Selecionar NEXTVAL até que a sequência alcançe o valor desejado.
  - Alterar a sequência ajustando o valor do INCREMENT BY para um valor muito grande, selecionar NEXTVAL e depois alterar a sequência para voltar ao INCREMENT BY ao valor original.
    - alter sequence SQ_MATRICULA_ALUNO increment by 1000;
      select SQ_MATRICULA_ALUNO.NEXTVAL from dual;
      alter sequence SQ_MATRICULA_ALUNO increment by 1;

Sequências

- Para ver as sequências existentes e suas propriedades, basta:
  - select * from USER_SEQUENCES;

- Para ver de um outro usuário:
  - select * from ALL_SEQUENCES
    where SEQUENCE_OWNER = '*<usuario>*';

Sinônimos

- Sinônimo é um apelido para um outro objeto do banco de dados.
- Sintaxe de criação:
  - create synonym <*nome-sinônimo*> for <[*esquema*].<*nome-objeto*>>
  - Ex.:
    - create synonym EMPREGADO for RH.EMPREGADO
- Sintaxe de destruição:
  - Ex.:
    - drop synonym EMPREGADO

Índices

- Os índices são estruturas de dados que podem oferecer um melhor desempenho na obtenção de linhas específicas do que o *full-table scan* (escaneamento completo da tabela) que é o padrão.

- Índices podem aumentar o desempenho de instruções SELECT, porém podem ser gargalos para operações de alterações de dados, pois devem ser atualizados de acordo com os dados.
  - Por isto devem ser utilizados com racionalidade e cautela.

- Pensamento:
  - Um bilhão de empregados. Crio os índices e depois insiro todos os empregados ou insiro todos os empregados e depois crio os índices?

Índices

- Os índices podem ser únicos (UNIQUE) ou não únicos.
- Os índices podem vincular mais de uma coluna. São conhecidos como índices concatenados.
- Pensamentos:
  - Qual possui melhor desempenho?
  - Se algum é melhor, posso sempre utilizá-lo?

Criando índices

- Sintaxe:
  - CREATE [UNIQUE] INDEX <*nome-índice*> ON
    <*nome-tabela*> (<*nome-coluna*> [, **<***nome-coluna***>...]**)

- Exemplo:
  - create index IX_DATA_NASCIMENTO on
    ALUNO (DATA_NASCIMENTO)

  - create **unique** index IX_NOME_UNICO on
    ALUNO (NOME)

Destruindo índices

- Sintaxe:
  - DROP INDEX <*nome-índice*>
- Exemplo:
  - drop index IX_DATA_NASCIMENTO
  - drop index IX_NOME_UNICO

Aula 14

Acesso de usuário e segurança

- O Oracle provê vários métodos para controle de acesso de usuários. Quando se cria um usuário, pode-se especificar como ele é autenticado e também ajustar vários de seus atributos.

- O modo inicial para controle de acesso de usuários é através de privilégios. Por garantir (*grant*) ou revogar (*revoke*) privilégios, você pode especificar o que os usuários podem fazer como certos objetos do banco de dados.

- Como os usuários usarão os recursos do sistema e diretivas de senha se fazem por um outro modo de controle conhecido como *profile* (perfil).

Criando e alterando usuários

- A instrução CREATE USER é empregada para a criação de usuários (também conhecidos como conta ou esquema) e pode, opcionalmente, designar certos atributos adicionais ao usuário.

- A instrução ALTER USER é utilizada para a realização de alterações de quaisquer atributos do usuário, porém o mesmo já deve estar criado.

Criando usuários

- A sintaxe para criação de usuários é:
  - CREATE USER <*nome-do-usuário*> IDENTIFIED BY <*senha-do-usuário*>

- O usuário acima é criado para ser autenticado no próprio Oracle. Porém existem outros modos de criação de usuário:
  - Autenticação externa: Validação de senha pelo sistema operacional ou pela rede.
  - Autenticação Global: Validação de senha pelo Serviço de Segurança do Oracle (*Oracle Security Service*), um serviço configurado e licenciado separadamente.

Designando atributos à usuários

- Atributos da conta são atribuidos com a própria instrução CREATE como com a instrução ALTER USER.
  - A instrução CREATE USER deve conter minimamente o usuário e a cláusula da senha.

- As instruções CREATE e ALTER USER podem:
  - Designar validação (senha)
  - Designar a *default tablespace* (espaço de tabelas padrão)
  - Designar a *temporary tablespace* (espaço de tabelas padrão)
  - Designar a quota de espaço disponível
  - Vincular a um *profile* (perfil)
  - Habilitar e desabilitar uma *role* (papel)
  - Ajustar expiração de senha
  - Travar e destravar usuários

Designando atributos à usuários

- Segue exemplos dos mais utilizados:
  - Alterando senha:
    - ALTER USER <*usuário*> IDENTIFIED BY <*senha*>
  - Vinculando a um *profile*:
    - ALTER USER <*usuário*> PROFILE <*nome-do-perfil*>
  - Vinculando a um *role:*
    - ALTER USER <*usuário*> DEFAULT ROLE <*nome-do-papel*>
      - Somente funciona com usuários administradores e só com a instrução ALTER USER.

Designando atributos à usuários

- Expirando senha:
  - ALTER USER <*usuário*> PASSWORD EXPIRE
- Travando um usuário:
  - ALTER USER <*usuário*> ACCOUNT LOCK
    - Para destravar, basta substituir LOCK por UNLOCK

Usando controle de licença de usuários

- O Oracle fornece vários tipos de licença de funcionamento. Para que se respeite a licença de uso adquirida, o Oracle inclui alguns mecanismos de controle.
- Você pode limitar o número de sessões permitidas a conectar simultaneamente e, também, o número de usuários que podem estar criados no SGBD.
  - Estes parâmetros que estão no *init.ora* podem ajudar nas restrições:
    - *licence_max_users*
    - *licence_sessions_warning*
    - *licence_max_sessions*
- Como administrador, para consultar os usuário do Oracle:
  - select USERNAME, PASSWORD, CREATED, PROFILE, ACCOUNT_STATUS from DBA_USERS;

Criando e usando *roles*

- Um *role* é um instrumento para administrar privilégios. Os privilégios podem ser garantidos a um *role* e este *role ser* garantido a um usuário ou, até mesmo, a um outro *role*.
  - Não servem para nenhum outro propósito a não ser um agrupamento de privilégios para facilitar a administração de privilégios.

- Sintaxe:
  - CREATE ROLE <nome-do-papel>;
    - Pode-se inserir uma senha para o *role*, se isto for feito ela deve ser ativada para seu funcionamento.

Criando e usando *roles*

- Exemplo:
  - CREATE ROLE permissao_aluno;

```
GRANT select on rh.empregado TO permissao_aluno;
GRANT select on rh.departamento TO permissao_aluno;
GRANT select on rh.funcao TO permissao_aluno;
GRANT select on rh.localizacao TO permissao_aluno;
GRANT select on rh.pais TO permissao_aluno;
GRANT select on rh.regiao TO permissao_aluno;
GRANT select on rh.historico_funcao TO permissao_aluno;

GRANT permissao_aluno TO aluno1, aluno2, aluno3, aluno4;
```

Criando e usando *profiles*

- *Profiles* permitem gerenciar grupos de usuários por limitar recursos de processamento ou políticas de senha.

- Todos os usuário possuem um *profile*. O *profile* padrão é admitido quando não é explicitado na criação de um usuário.

- As instruções CREATE PROFILE e ALTER PROFILE são responsáveis, respectivamente, por criar e alterar *profile*. Possuem praticamente a mesma sintaxe.
  - A única diferença é que a instrução ALTER PROFILE necessita que o *profile* já exista.

Criando e usando *profiles*

- Sintaxe:
  - [CREATE | ALTER] PROFILE <*nome-do-perfil*> LIMIT <*restrição*>

- Existem várias restrições, são alguns exemplos:
  - CONNECT_TIME
    IDLE_TIME
    CPU_PER_CALL
    CPU_PER_SESSION
    SESSIONS_PER_USER
    FAILED_LOGON_ATTEMPTS
    PASSWORD_LIFE_TIME
    PASSWORD_GRACE_TIME
    PASSWORD_LOCK_TIME
    PASSWORD_REUSE_TIME
    PASSWORD_VERIFY_FUNCTION
    PRIVATE_SGA
  - Mais detalhes podem ser encontrados nas páginas 482-490 da referência DAWES e THOMAS (2002).

Criando e usando *profiles*

- Exemplo:
  - CREATE PROFILE aluno LIMIT SESSIONS_PER_USER 1;

    ALTER PROFILE aluno LIMIT CONNECT_TIME 3;
    ALTER PROFILE aluno LIMIT IDLE_TIME 1;
    ALTER PROFILE aluno LIMIT FAILED_LOGIN_ATTEMPTS 3;
    ALTER PROFILE aluno LIMIT PASSWORD_LIFE_TIME 30;
    ALTER PROFILE aluno LIMIT PASSWORD_REUSE_TIME 90;

    CREATE USER alunox IDENTIFIED BY alunox PROFILE aluno;
    GRANT papel_usuario TO alunox;

Garantindo e revogando privilégios

- Privilégios permitem um usuário acessar objetos ou executar programas que são de um outro usuário ou executar operações de nível de sistema, como criar e destruir objetos.

- Privilégios podem ser garantidos (*grant*) para um usuário, para o usuário especial PUBLIC ou para um *role*. Uma vez garantidos, tais privilégios podem ser revogados *(revoke)*.

Garantindo e revogando privilégios

- O Oracle tem três tipos de privilégios:
  - Privilégios de objetos (*Object privileges)*
    - São permissões em objetos de esquemas, como tabelas, visões, funções e bibliotecas.
  - Privilégios de sistema *(System privileges)*
    - Habilita ao usuário a desenvolver operações de nível de sistema, tais como conexão, alterar sessão, criar tabelas ou criar usuários.
  - Privilégios de *role (Role privileges)*
    - São aqueles privilégios que um usuário possui por intermédio de um *role*.
- Nos próximos *slides* iremos ver quais são estas permissões, contudo mais detalhes podem ser encontrados nas páginas 464-473 da referência DAWES e THOMAS (2002).

Privilégios de objetos

- Privilégios de objetos:
  - ALTER
  - DELETE
  - EXECUTE
  - INDEX
  - INSERT
  - SELECT
  - UPDATE
  - ALL

  - Outros privilégios:
    - READ REFERENCE

Privilégios de sistema

- Privilégios de sistema:
  - CLUSTER
  - DATABASE
  - INDEX
  - PROCEDURE
  - PROFILE
  - ROLE
  - ROLLBACK SEGMENT
  - SEQUENCE
  - SESSION
  - SYNONYM
  - TABLE
  - TABLESPACE
  - USER
  - VIEW
  - Existem alguns privilégios especiais:
    - SYSDBA
    - SYSOPER
  - Outros privilégios:
    - ANALYSE
    - AUDIT
    - COMMENT
    - PRIVILEGE
    - ROLE

Atribuindo privilégios

- Quando se deseja atribuir um ou mais privilégios para um usuário ou um *role*, deve ser utilizada a instrução GRANT.

- SINTAXES:
  - GRANT <*privilegio-de-objeto*> TO <*usuário*|*role*|*PUBLIC*>
    [WITH GRANT OPTION]
  - GRANT <*privilegio-de-sistema*> TO <*usuário*|*role*|*PUBLIC*>
    [WITH ADMIN OPTION]

    - Garantir um privilégio para o usuário especial PUBLIC implica em garantir o privilégio para qualquer usuário que conectar o banco de dados.
    - Quando um privilégio é garantido, o seu efeito é imediato, não necessitando que um usuário desconecte e conecte novamente.

Atribuindo privilégios de objetos

- Exemplos de atribuição de privilégios de objetos:
  - GRANT SELECT ON rh.empregado TO joazinho

    GRANT UPDATE (ultimo_nome, salario) ON rh.empregado TO joazinho

    GRANT INSERT, UPDATE, DELETE ON rh.departamento TO joaozinho

Atribuindo privilégios de objetos

- Privilégios de objetos podem ser garantidos com WITH GRANT OPTION, que permite ao garantido garantir estes privilégios para qualquer outro usuário ou *role*.

Atribuindo privilégios de objetos

- Por exemplo, se o usuário RH conceder consulta a tabela FUNCAO ao Joãozinho usando WITH GRANT OPTION, Joãozinho pode conceder consulta a essa mesma tabela a Mariazinha:
  - conn rh/rh@xe;

    GRANT SELECT ON rh.funcao TO joaozinho WITH GRANT OPTION;

    conn joaozinho/joaozinho@xe;

    GRANT SELECT ON rh.funcao TO mariazinha;

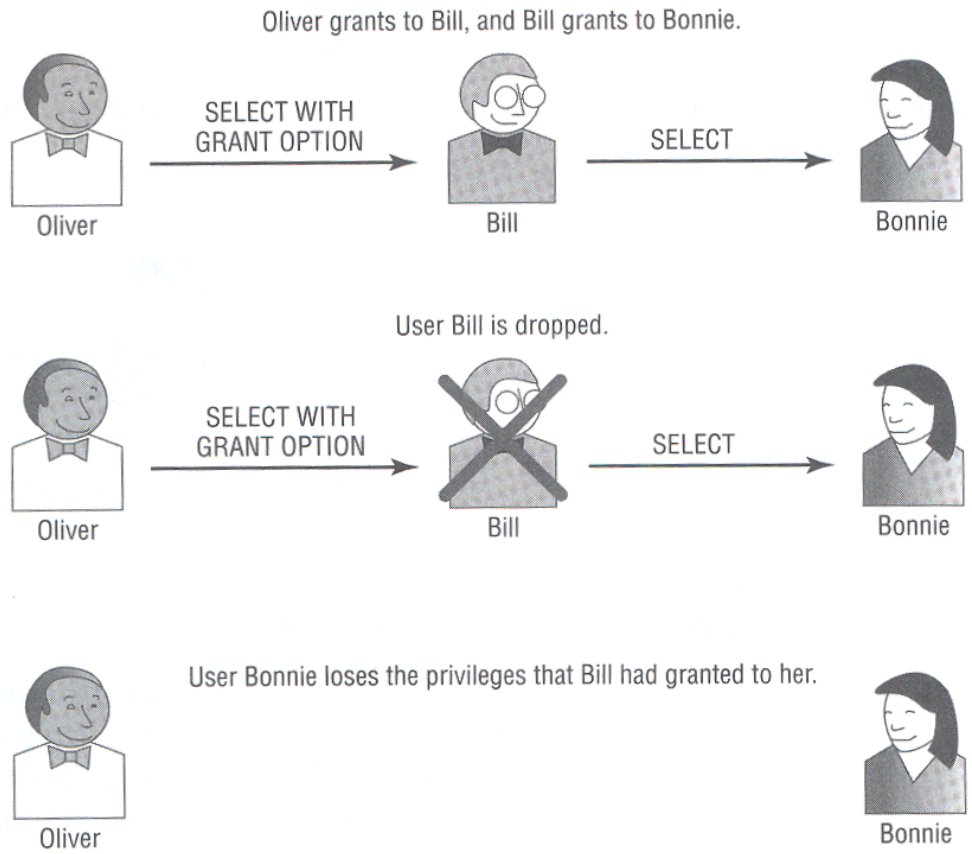
    conn mariazinha/mariazinha@xe;

    select * from rh.funcao;

  - Porém, se Joãozinho deixar de ser um usuário, Mariazinha pode perder a sua garantia de consulta.
    - A não ser que outro usuário também tenha concedido este privilégio a ela.

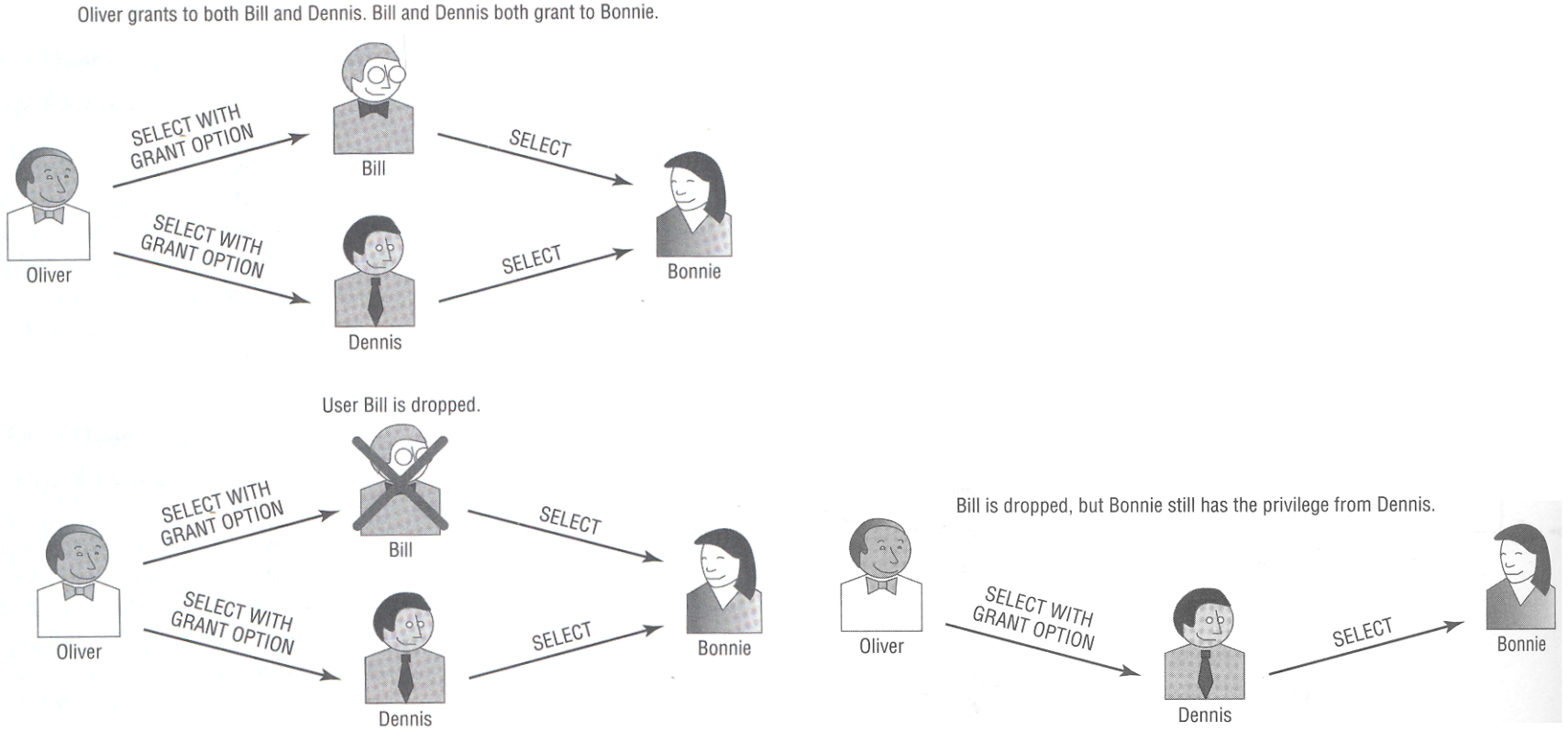
Atribuindo privilégios de objetos

- Entendimento 1:



Atribuindo privilégios de objetos

- Entendimento 2:



Atribuindo privilégios de sistema

- Exemplos de atribuição de privilégios de sistema:
  - GRANT CREATE SESSION TO joaozinho
    GRANT CREATE SEQUENCE TO joaozinho
    GRANT CREATE SYNONYM TO joaozinho
    GRANT CREATE TABLE TO joaozinho
    GRANT CREATE ANY TABLE TO joaozinho
    GRANT DROP TABLE TO joaozinho
    GRANT DROP ANY TABLE TO joaozinho
    GRANT CREATE PROCEDURE TO joaozinho
    GRANT EXECUTE ANY PROCEDURE TO joaozinho
    GRANT CREATE USER TO joaozinho
    GRANT DROP USER TO joaozinho
    GRANT CREATE VIEW TO joaozinho

Atribuindo privilégios de sistema

- Existe três *roles* que abrangem vários privilégios de sistema e facilitam em muito a concessão de várias permissões:
  - CONNECT
    - Contém somente o privilégio CREATE SESSION.
  - RESOURCE
    - Contém os seguintes privilégios:
      - CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE.
  - DBA
    - Contém os privilégios comuns de um administrador do banco de dados.

Atribuindo privilégios de sistema

- Privilégios de sistema podem ser garantidos com WITH ADMIN OPTION, que permite ao garantido garantir estes privilégios de sistema para qualquer outro usuário ou *role*.

Atribuindo privilégios de sistema

- Por exemplo, se o usuário RH conceder criação de visão ao Joãozinho usando WITH ADMIN OPTION, Joãozinho pode conceder criação de visão a Mariazinha:
  - conn rh/rh@xe;

    GRANT CREATE VIEW TO joaozinho WITH ADMIN OPTION;

    conn joaozinho/joaozinho@xe;

    GRANT CREATE VIEW TO mariazinha;

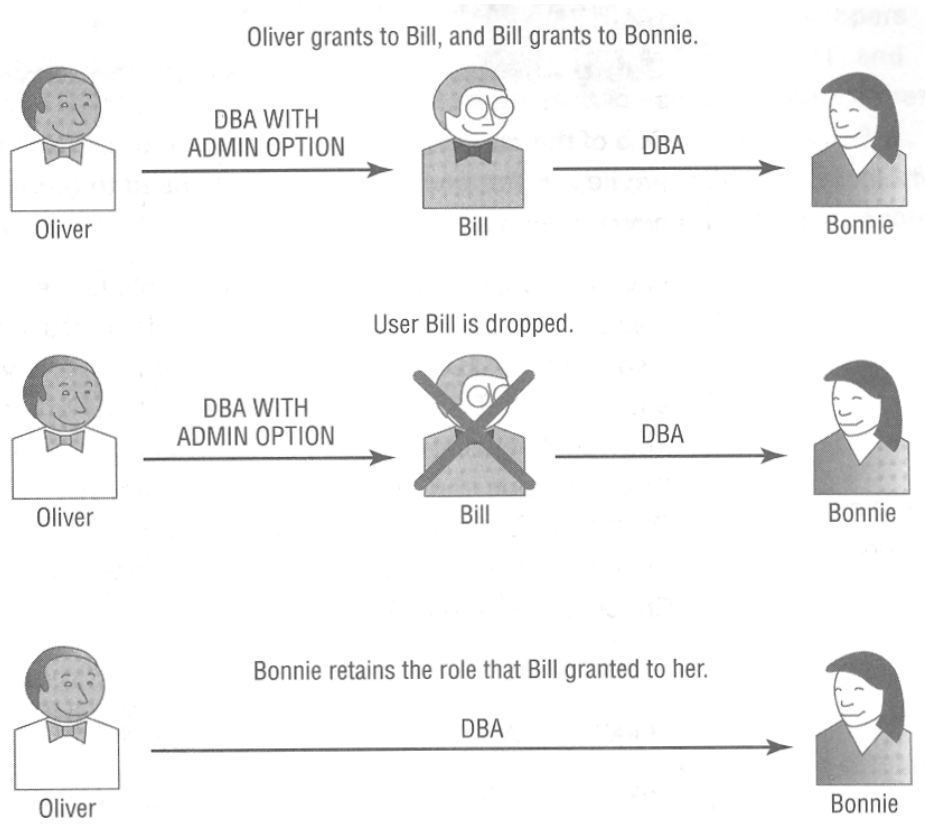
    conn mariazinha/mariazinha@xe;

    create view DUPLO as select * from dual;

  - Ao contrário do privilégio de objetos, que utiliza WITH GRANT OPTION, se Joãozinho deixar de ser um usuário, Mariazinha não perde a garantia de criação de visão.

Atribuindo privilégios de sistema

- Entendimento:



Revogando privilégios

- Quando se deseja revogar um ou mais privilégios para um usuário ou um *role*, deve ser utilizada a instrução REVOKE.

- SINTAXE:
  - REVOKE <*privilegio-de-objeto-ou-sistema*> FROM <*usuário*|*role*|*PUBLIC*>

Revogando privilégios

- Exemplos de revogação de privilégios de sistema:
  - REVOKE CREATE SESSION FROM joaozinho
  - REVOKE CREATE SEQUENCE FROM joaozinho
  - REVOKE CREATE SYNONYM FROM joaozinho
  - REVOKE CREATE TABLE FROM joaozinho
  - REVOKE CREATE ANY TABLE FROM joaozinho
  - REVOKE DROP TABLE FROM joaozinho
  - REVOKE DROP ANY TABLE FROM joaozinho
  - REVOKE CREATE PROCEDURE FROM joaozinho
  - REVOKE EXECUTE ANY PROCEDURE FROM joaozinho
  - REVOKE CREATE USER FROM joaozinho
  - REVOKE DROP USER FROM joaozinho
  - REVOKE CREATE VIEW FROM joaozinho

Aula 15

Sistema da Locadora: MR s/atributos

ATOR (id_ator , nome_real , nome_artistico , dt_nascimento)

FILME (id_filme , titulo , id_categoria)
FILME [id_categoria] ← CATEGORIA [id_categoria]

CATEGORIA (id_categoria , descricao)

ATOR_FILME (id_ator , id_filme)
ATOR_FILME [id_ator] ← ATOR [id_ator]
ATOR_FILME [id_filme] ← FILME [id_filme]

DVD (id_filme , numero)
DVD [id_filme] ←FILME [id_filme]

Sistema da Locadora: MR s/atributos

LOCACAO (id_filme , numero , id_cliente , dt_locacao , dt_devolucao_prevista, dt_devolucao_efetiva)

LOCACAO [id_filme , numero] ← DVD [id_filme , numero]

LOCACAO [id_cliente] ← CLIENTE [id_cliente]

CLIENTE (id_cliente , nome , telefone , id_endereco)

CLIENTE [id_endereco] ← ENDERECO [id_endereco]

TITULAR (id_titular , cpf)

TITULAR [id_titular] ← CLIENTE [id_cliente]

DEPENDENTE (id_dependente , id_titular)

DEPENDENTE [id_dependente] ← CLIENTE [id_cliente]

DEPENDENTE [id_titular] ← TITULAR [id_titular]

ENDERECO (id_endereco , logradouro , numero , complemento , cep , bairro , cidade , estado)

Visões, permissões e otimizações

- Hoje vamos trabalhar com visões, permissões e otimizações
- Com o uso de visões, conseguimos filtrar informações de um banco de dados. São utilizações comuns de visões:
  - Representar um subconjunto de dados
  - Representar um superconjunto de dados
  - Ocultar junções complexas
  - Prover nomes mais entendíveis para colunas
  - Criação de uma camada entre a aplicação e a fonte de dados
- Com o uso de permissões, podemos restringir o que cada usuário pode fazer com certos objetos do banco de dados
- Com o uso de índices, podemos melhorar o desempenho em várias consultas

Cronograma da aula

1. Criar três tipos de usuários:
   - Administrador, cliente e atendente
2. Criar todo o modelo do Sistema Locadora naquele que será o administrador, no caso, proprietário
   - Explicar o funcionamento usual em sistemas *web*
3. Criar visões aos usuários do tipo *cliente* e *atendente*
4. Garantir as devidas permissões ao cliente e ao atendente
5. Criar sinônimos
6. Pensar em otimização utilizando índices

Para o **cliente**

- Criação da visão somente leitura **VW_DADOS_CLIENTE**
  - Exibirá o ID_CLIENTE, CPF, NOME, TIPO, TELEFONE, ENDERECO e o NOME_TITULAR (este último caso seja dependente)
  - A própria aplicação fará o filtro pelo ID_CLIENTE
- Criação da visão somente leitura **VW_VINCULO_DEPENDENTE**
  - Exibirá o ID_CLIENTE, ID_DEPENDENTE e NOME_DEPENDENTE
  - A própria aplicação fará o filtro pelo ID_CLIENTE

Para o **cliente**

- Criação da visão somente leitura **VW_RELATORIO_FILMES**
  - Exibirá o NOME_FILME, CATEGORIA, NOME_ARTISTICO (caso não exista exibir o NOME_REAL), QTDE e QTDE_DISP

- Criação da visão somente leitura **VW_HISTORICO_LOCACOES**
  - Exibirá o NOME_USUARIO, TIPO_USUARIO, NOME_FILME, DT_LOCACAO, DT_DEV_PREVISTA, DT_DEV_EFETIVA e STATUS
    - STATUS pode ser ALUGADO, DEVOLUÇÃO PENDENTE, DEVOLVIDO e DEVOLVIDO COM ATRASO
  - A própria aplicação fará o filtro pelo ID_CLIENTE

Para o **cliente**

- As seguintes permissões devem ser concedidas:
  - Em todas as visões criadas para o cliente
    - R (*Retrieve*)
      - Pois somente poderá consultar

- Sinônimos para todas as visões criadas para o cliente foram criados:
  - `create SYNONYM VW_RELATORIO_FILMES FOR LOC_ADM.VW_RELATORIO_FILMES;`
  - `create SYNONYM VW_DADOS_CLIENTE FOR LOC_ADM.VW_DADOS_CLIENTE;`
  - `create SYNONYM VW_HISTORICO_LOCACOES FOR LOC_ADM.VW_HISTORICO_LOCACOES;`
  - `create SYNONYM VW_VINCULO_DEPENDENTE FOR LOC_ADM.VW_VINCULO_DEPENDENTE;`

Para o **atendente**

- Criação da visão somente leitura **VW_RELATORIO_FILMES_COMPLETO**
  - Exibirá o NOME_FILME, CATEGORIA, NOME_REAL, **NOME_ARTISTICO** QTDE e QTDE_DISP

- Criação da visão somente leitura **VW_SITUACAO_DVDS**
  - Exibirá o ID_FILME, NOME_FILME, NUMERO_DVD e STATUS
    - STATUS pode ser ALUGADO, DEVOLUÇÃO PENDENTE e DISPONÍVEL

- Criação de uma tabela **FUNCIONARIO**
  - Para informar os dados do atendente e para o mesmo se logar
  - Terá ID_FUNCIONARIO, CPF, NOME e SENHA
  - Pertencerá ao administrador

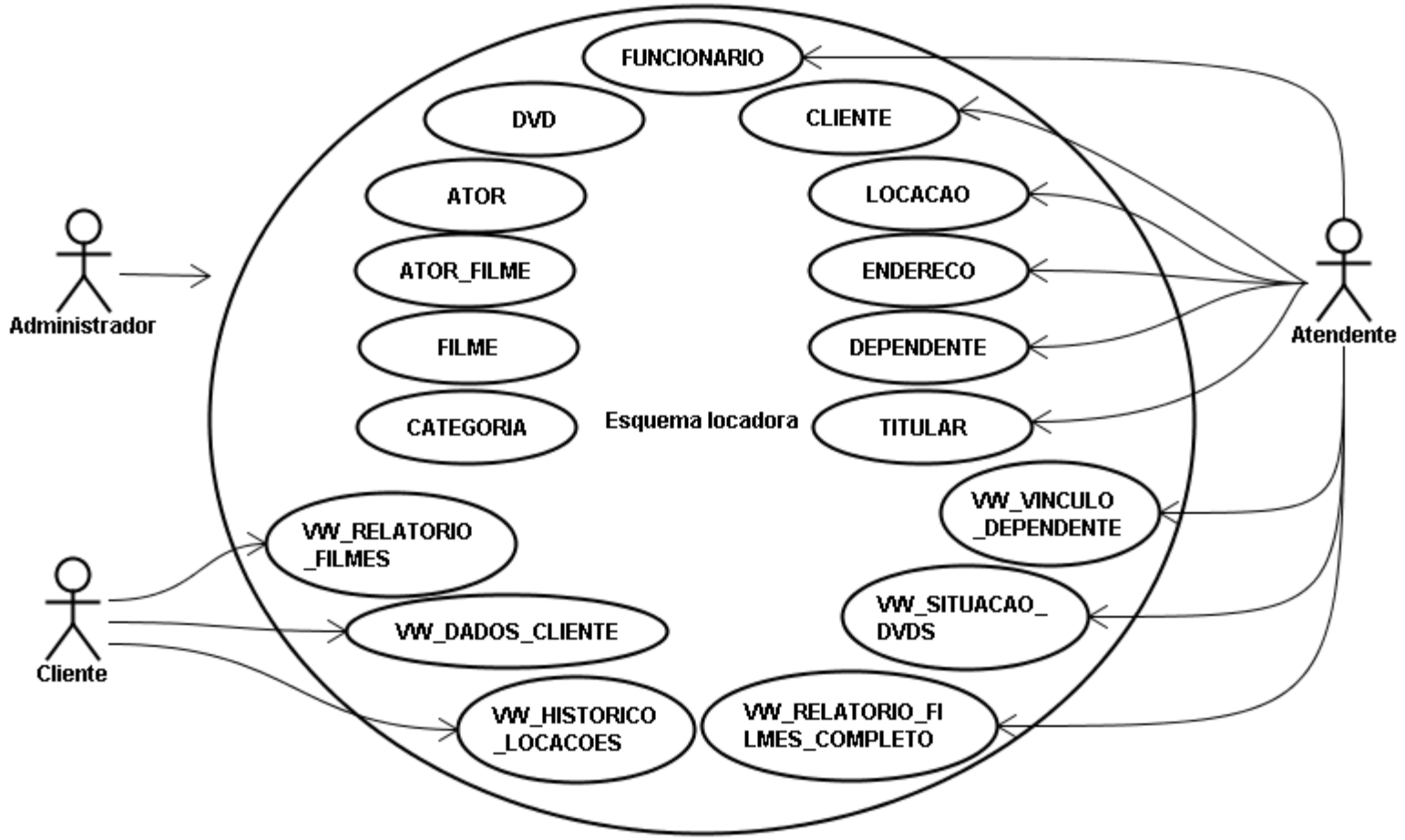
Para o **atendente**

- As seguintes permissões devem ser concedidas:
  - Em LOCACAO
    - CRUD (*Create, Retrieve, Update* e *Delete*)
      - Pois o atendente deve gerenciar as locações
  - Em FUNCIONARIO
    - R
      - Pois o atendente somente poderá consultar essa tabela para se logar
  - Em CLIENTE, TITULAR, DEPENDENTE e ENDERECO
    - CRUD
      - Pois o atendente deve gerenciar o cadastro de clientes
  - Em SQ_CLIENTE e SQ_ENDERECO
    - Acesso
      - Pois o atendente deve utilizar essas sequências para as atividades acima mencionadas

Para o **atendente**

- Sinônimos para todas as visões criadas para o atendentes foram criados:
  - `create SYNONYM VW_RELATORIO_FILMES_COMPLETO FOR LOC_ADM.VW_RELATORIO_FILMES_COMPLETO;`
  - `create SYNONYM VW_SITUACAO_DVDS FOR LOC_ADM.VW_SITUACAO_DVDS;`

Como ficou?


FUNCIONARIO
DVD
CLIENTE
ATOR
LOCACAO
ATOR_FILME
ENDERECO
FILME
DEPENDENTE
CATEGORIA
Esquema locadora
TITULAR
VW_VINCULO
_DEPENDENTE
VW_RELATORIO
_FILMES
VW_SITUACAO_
DVDS
VW_DADOS_CLIENTE
VW_HISTORICO
_LOCACOES
VW_RELATORIO_FI
LMES_COMPLETO
Administrador
Atendente
Cliente

Otimização

- Observando as visões criadas e as responsabilidades tanto do **cliente** quanto do **atendente**, quais índices seriam apropriados para criação e porque?
  - CPF e SENHA do FUNCIONÁRIO
    - `create index ix_login_funcionario on FUNCIONARIO(CPF,SENHA);`
  - ID_CLIENTE em LOCACAO
    - `create index ix_locacao_cliente on LOCACAO(ID_CLIENTE);`
  - ID_FILME e NUMERO em LOCACAO
    - `create index ix_locacao_dvd on LOCACAO(ID_FILME, NUMERO);`
  - ID_TITULAR em DEPENDENTE
    - `create index ix_dependente_titular on DEPENDENTE(ID_TITULAR);`
  - ...

Otimização

- ID_CATEGORIA em FILME
  - `create index ix_filme_categoria on FILME(ID_CATEGORIA);`
- NOME em CLIENTE
  - `create index ix_nome_cliente on CLIENTE(NOME);`

Aula 16

Triggers

- Triggers são procedimentos que podem ser gravados em Java, PL/SQL ou C. São executados (ou disparados) implicitamente quando uma tabela é modificada, um objeto é criado ou ocorrem algumas ações de usuário ou de sistema de banco de dados

- As *triggers* são similares as *stored procedures* diferenciando, apenas, na maneira como são chamadas. A *trigger* é executada implicitamente quando ocorre algum evento enquanto a *stored procedure* deve ser executado explicitamente

Triggers

- Uma trigger é composta por quatro partes:
  - Momento
  - Evento
  - Tipo
  - Corpo

Triggers - momento

- Momento define quando uma *trigger* irá ser acionada. Pode ser:
  - BEFORE (tabela)
  - AFTER (tabela)
  - INSTEAD OF (view)

Triggers - momento

- BEFORE indica que os comandos PL/SQL do corpo da *trigger* serão executados ANTES dos dados da tabela serem alterados. Normalmente usamos BEFORE nos casos em que precisamos incializar variáveis globais, validar regras de negócios, alterar o valor de *flags* ou para salvar o valor de uma coluna antes de alterarmos o valor delas. Exemplo:

```
CREATE OR REPLACE TRIGGER novo_func
   BEFORE...
.
.
END;
/
```

Triggers - momento

- AFTER indica que os comando PL/SQL do corpo da *trigger* será executado APÓS os dados da tabela serem alterados. Normalmente usamos AFTER para completar os dados de outras tabelas e para completar a atividade de outra *trigger* de momento BEFORE. Exemplo:

```
CREATE OR REPLACE TRIGGER novo_func
   AFTER...
.
.
END;
/
```

Triggers - momento

- INSTEAD OF indica que a *trigger* irá ser executada no lugar da instrução que disparou a *trigger*. Literalmente, a instrução é substituída pela *trigger*

- Essa técnica permite que façamos, por exemplo, alterações em uma tabela através de uma *view*. É usado nos casos em que a *view* não pode alterar uma tabela por não referenciar à uma coluna com a restrição *not null*. Nesse caso a *trigger* pode atualizar a coluna que a *view* não tem acesso

Triggers - momento

- Dois detalhes muito importantes sobre INSTEAD OF:
  - Só funcionam com *views*
  - É sempre de linha. Será considerado assim, mesmo que "FOR EACH ROW" for omitido

- Exemplo:

```
CREATE OR REPLACE TRIGGER novo_func
   INSTEAD OF INSERT ON vemp
FOR EACH ROW
WHEN ...
.
.
END;
/
```

Triggers - evento

- O evento define qual é a instrução DML que aciona/disparar a *trigger*. Pode ser:
  - INSERT
  - UPDATE
  - DELETE

- Quando o evento for um UPDATE podemos informar quais colunas que, ao serem alteradas, irão disparar a *trigger*. O mesmo NÃO ocorre com INSERT e DELETE porque essas instruções sempre afetam a linha por inteiro

Triggers - evento

- Exemplo:

```
CREATE OR REPLACE TRIGGER novo_func
   AFTER INSERT ON emp
.
.
END;
/
```

- O evento pode conter uma, duas ou todas as três operações DML em uma única linha de comando. Exemplo:

```
CREATE OR REPLACE TRIGGER novo_func
   BEFORE INSERT OR UPDATE OR DELETE ON emp
.
.
END;
/
```

Triggers - tipo

- O tipo define quantas vezes uma *trigger* será executa. A *trigger* pode ser executada uma vez para a instrução que a disparou ou ser disparada para cada linha afetada pela instrução que a disparou. Pode ser:
  - Instrução (STATEMENT)
  - Linha (ROW)

Triggers - tipo

- Quando a *trigger* for do tipo instrução ela será disparada uma vez para cada evento de *trigger*, mesmo que nenhuma linha tenha sido afetada. São úteis para aquelas *trigger* que eventualmente não alteram dados ou para situações onde o que queremos é uma resposta da *trigger*, por exemplo, em uma restrição complexa de negócio. Por DEFAULT toda *trigger* é deste tipo. Exemplo:

```
CREATE OR REPLACE TRIGGER novo_func
   BEFORE INSERT OR UPDATE OR DELETE ON emp
   FOR EACH STATEMENT
.
.
END;
/
```

Triggers - tipo

- Quando a trigger for do tipo linha, a *trigger* será executada toda vez que a tabela for afetada pelo evento da *trigger*. Se nenhuma linha for afetada a *trigger* não será executada. São muito úteis quando a ação da *trigger* depende dos dados afetados pelo evento da *trigger*. Exemplo:

```
CREATE OR REPLACE TRIGGER novo_func
   BEFORE INSERT OR UPDATE OR DELETE ON emp
   FOR EACH ROW
.
.
END;
/
```

Triggers - corpo

- O corpo define a ação que uma *trigger* irá executar quando acionada. O corpo de uma *trigger* é composto por um bloco PL/SQL, a chamada de uma PROCEDURE ou por um procedimento JAVA. Por definição, o tamanho de uma *trigger* não pode ultrapassar 32K

- Como, normalmente, precisamos trabalhar com os valores antes e depois da alteração dos dados, a *trigger* permite que façamos referencia aos valores antes da alteração (OLD) e após a alteração (NEW)

- O nome de uma trigger deve ser único dentro de um mesmo esquema

Triggers - corpo

- Sintaxe básica:

```
CREATE [OR REPLACE] TRIGGER [schema.] nome_da_trigger
      [BEFORE|AFTER]
      [DELETE|OR INSERT|OR UPDATE[OF coluna]]
      ON [schema.] nome_da_tabela_ou_da_view
      [REFERENCING [OLD [AS] OLD] [NEW [AS] NEW]
      [FOR EACH ROW|STATEMENT]
      [WHEN [condição]]
BLOCO PL/SQL
```

  - Bloco PL/SQL inicia-se normalmente com BEGIN e encerra com END

Triggers – exemplos práticos

- Antes de trabalharmos com *triggers* em tabelas, vamos entender o seu funcionamento com alguns eventos de sistema, tais como:
  - AFTER SERVERERROR
  - AFTER LOGON
  - BEFORE LOGOFF
  - AFTER STARTUP
  - BEFORE SHUTDOWN

Triggers – exemplos práticos

- O próximo exemplo irá auditar o nome do usuário e a data e hora que ele realizou *login* no sistema

- Portanto, vamos criar uma tabela para armazenar isto:

```
create table sys.AUDITORIA_LOGIN (
    USUARIO       varchar2(30)  not null,
    DATA_ACESSO   date          not null
);
```

Triggers – exemplos práticos

- Observe como ficaria a nossa *trigger* de auditoria:

```
CREATE OR REPLACE TRIGGER audita_login
    AFTER LOGON ON DATABASE
BEGIN
    INSERT INTO sys.AUDITORIA_LOGIN
        (USUARIO, DATA_ACESSO) values (USER, SYSDATE);
    COMMIT;
END;
/
```

Triggers – exemplos práticos

- A *trigger* pode apresentar problemas no momento de sua criação e exibir mensagem como a abaixo:

```
Warning: Trigger created with compilation errors.
```

- Logo em seguida faça o comando "SHOW ERRORS" que ele exibirá o que ocorreu:

```
SQL> show errors;
Errors for TRIGGER AUDITA_LOGIN:

LINE/COL ERROR
-------- -----------------------------------------
2/3      PL/SQL: SQL Statement ignored
2/19     PL/SQL: ORA-00942: table or view does not exist
```

Triggers – exemplos práticos

- Caso não seja logo em seguida, você pode a qualquer momento realizar o comando "SHOW ERRORS TRIGGER nome_da_trigger":

```
SQL> SHOW ERRORS TRIGGER audita_login;
No errors.
```

- Caso deseje saber informações sobre as *triggers* basta consultar a table USER_TRIGGERS ou mesmo ALL_TRIGGERS. Exemplo:

```
SQL> select TRIGGER_NAME from USER_TRIGGERS
```

Triggers – exemplos práticos

- Até mesmo o corpo da *trigger* pode ser acessado pela USER_TRIGGERS:

```
SELECT trigger_name, trigger_type, triggering_event,
           table_name, referencing_names,
           status, trigger_body
FROM user_triggers
WHERE trigger_name = 'NOME_DA_TRIGGER';
```

Triggers – exemplos práticos

- Caso descubra que não precisa mais da *trigger* existe duas formas de tratar a situação: eliminá-la ou desabilitá-la.

- Eliminando a trigger:
  - drop trigger NOME_DA_TRIGGER

- Desabilitando:
  - alter trigger NOME_DA_TRIGGER **disable**

- Habilitando:
  - alter trigger NOME_DA_TRIGGER **enable**

Triggers – exemplos práticos

- Vamos continuar com nossos exemplos práticos. Existe uma tabela chamada RELATORIO que possui uma única linha e possui um campo QTDE_FUNCIONARIO e TOTAL_SALARIO. Observe:

```
create table RELATORIO (
    ID_RELATORIO        number(1)     not null,
    QTDE_FUNCIONARIO    number(4)     not null,
    TOTAL_SALARIO       number(11,2)  not null,
    constraint pk_relatorio primary key (ID_RELATORIO),
    constraint ck_unico check (ID_RELATORIO=1)
);
```

Triggers – exemplos práticos

- Suponha que exista a seguinte tabela:

```
create table FUNCIONARIO (
    ID_FUNCIONARIO      number(4)     not null,
    NOME                varchar2(60)  not null,
    SALARIO             number(8,2)   not null,
    constraint pk_empregado primary key (ID_FUNCIONARIO)
);
```

Triggers – exemplos práticos

- Portanto, para iniciar a tabela RELATORIO basta:

```
insert into RELATORIO
   (ID_RELATORIO, QTDE_FUNCIONARIO, TOTAL_SALARIO)
values
   (1,
     (select count(*) from FUNCIONARIO),
     (select nvl(sum(SALARIO),0) from FUNCIONARIO)
   );
```

- Como manter este relatório sempre atualizado sem ter que ficar sempre contando todos os funcionários e somando todos os salários?

Triggers – exemplos práticos

- Manteremos a quantidade de funcionários sempre atualizada se:
  - ao inserir mais um funcionário, incrementarmos a quantidade
  - ao excluir um funcionário, decrementarmos a quantidade

- Manteremos o total de salário sempre atualizado se:
  - ao inserir mais um funcionário, somarmos o seu salário
  - ao excluir um funcionário, diminuirmos o seu salário
  - ao modificar o salário de um funcionário, diminuirmos seu antigo salário e somarmos o seu novo salário

Triggers – exemplos práticos

- Manteremos a quantidade de funcionários sempre atualizada se:
  - ao inserir mais um funcionário, incrementarmos a quantidade
  - ao excluir um funcionário, decrementarmos a quantidade

```
create or replace trigger TRIGGER_QTDE_FUNC
     AFTER INSERT OR DELETE ON FUNCIONARIO
     FOR EACH ROW
begin
   IF INSERTING THEN
      update RELATORIO
         set QTDE_FUNCIONARIO = QTDE_FUNCIONARIO+1;
   ELSIF DELETING THEN
      update RELATORIO
         set QTDE_FUNCIONARIO = QTDE_FUNCIONARIO-1;
   END IF;
end;
/
```

Triggers – exemplos práticos

- Manteremos o total de salário sempre atualizado se:

```
create or replace trigger TRIGGER_TOTAL_SALARIO
     AFTER INSERT OR DELETE OR UPDATE OF SALARIO ON FUNCIONARIO
     FOR EACH ROW
begin
   IF INSERTING THEN
      update RELATORIO
         set TOTAL_SALARIO = TOTAL_SALARIO + :NEW.SALARIO;
   ELSIF DELETING THEN
      update RELATORIO
         set TOTAL_SALARIO = TOTAL_SALARIO - :OLD.SALARIO;
   ELSIF UPDATING THEN
      update RELATORIO
         set TOTAL_SALARIO = TOTAL_SALARIO -
                             :OLD.SALARIO + :NEW.SALARIO;
   END IF;
end;
/
```

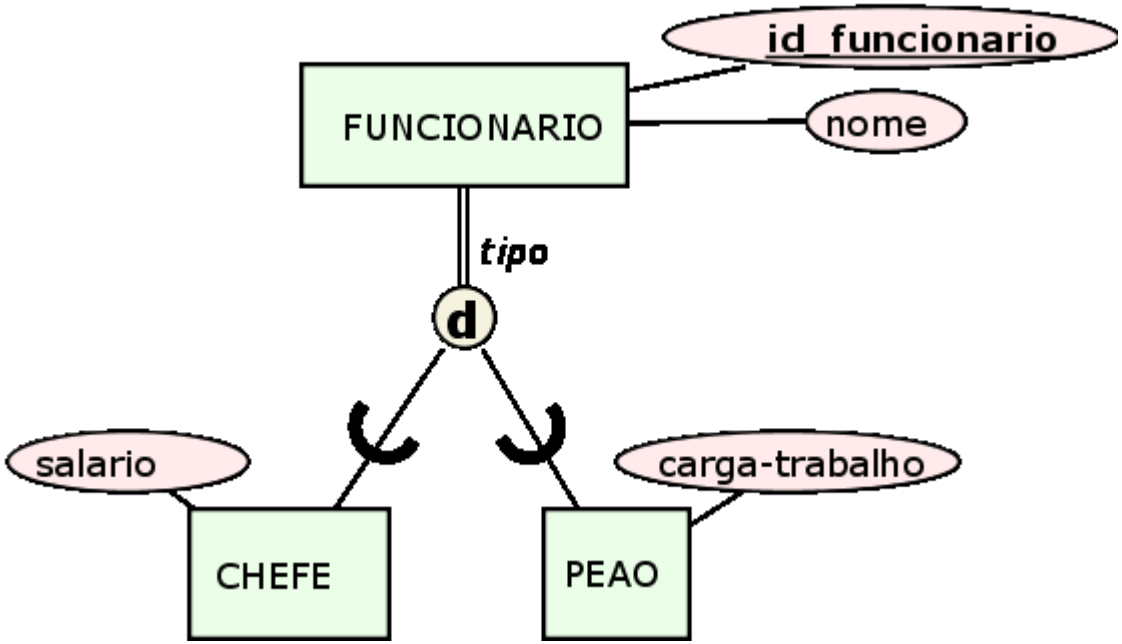
Triggers – exemplos práticos

- Teste:

```
    [1     0          0  ]      (select * from RELATORIO)
insert into FUNCIONARIO values (1, 'A', 43.9);
    [1     1       43.9  ]
insert into FUNCIONARIO values (2, 'B', 106.1);
    [1     2      150.0  ]
insert into FUNCIONARIO values (3, 'C', 40.0);
    [1     3      190.0  ]
update FUNCIONARIO set SALARIO = 116.1
  where ID_FUNCIONARIO = 2;
    [1     3      200.0  ]
delete from FUNCIONARIO where ID_FUNCIONARIO = 3;
    [1     2      160.0  ]
```

Triggers – exemplos práticos

- Solucionando herança por disjunção:



- Um funcionário é peão ou chefe. Como impedir que seja criado um mesmo funcionário como peão e como chefe?

Triggers – exemplos práticos

- Restrição por chave estrangeira não é suficiente:

```
create table FUNCIONARIO (
   ID_FUNCIONARIO   number(3)       not null,
   NOME             varchar2(60)    not null,
   constraint PK_FUNCIONARIO primary key (ID_FUNCIONARIO)
);

create table CHEFE (
   ID_CHEFE         number(3)       not null,
   SALARIO          number(8,2)     not null,
   constraint PK_CHEFE primary key (ID_CHEFE),
   constraint fk_chefe FOREIGN KEY (ID_CHEFE)
            references FUNCIONARIO(ID_FUNCIONARIO)
);

create table PEAO (
   ID_PEAO          number(3)       not null,
   CARGA_TRABALHO   number(3)       not null,
   constraint PK_PEAO primary key (ID_PEAO),
   constraint fk_peao FOREIGN KEY (ID_PEAO)
            references FUNCIONARIO(ID_FUNCIONARIO)
);
```

Triggers – exemplos práticos

- Manteremos a integridade de disjunção se:
  - Ao vincular um funcionário como CHEFE, garantir que ele não seja PEÃO

```
create or replace trigger TG_CHEFE
     BEFORE INSERT OR UPDATE OF ID_CHEFE ON CHEFE
     FOR EACH ROW
declare
   contador NUMBER;
begin
   select count(ID_PEAO) into contador from PEAO
             where ID_PEAO = :NEW.ID_CHEFE;
   IF contador != 0 THEN
     RAISE_APPLICATION_ERROR(-20515,'JA É PEÃO!');
   END IF;
end;
/
```

Triggers – exemplos práticos

- Manteremos a integridade de disjunção se:
  - Ao vincular um funcionário como PEÃO, garantir que ele não seja CHEFE

```
create or replace trigger TG_PEAO
     BEFORE INSERT OR UPDATE OF ID_PEAO ON PEAO
     FOR EACH ROW
declare
   contador NUMBER;
begin
   select count(ID_CHEFE) into contador from CHEFE
             where ID_CHEFE = :NEW.ID_PEAO;
   IF contador != 0 THEN
     RAISE_APPLICATION_ERROR(-20515,'JA É CHEFE!');
   END IF;
end;
/
```

Referência Bibliográfica

- DAWES, Chip; THOMAS, Biju. **OCA/OCP: Introduction to Oracle9i™ SQL**. São Francisco: Sibex, 2002.

- GOYA, Milton. **Oracle – Trigger**. Disponível em: <http://www.linhadecodigo.com.br/Artigo.aspx?id=322>. Acesso em: 09 set. 2008.

- GURSAHANI, Ajay. **Materialized Views in Oracle**. Disponível em: <http://www.databasejournal.com/features/oracle/article.php/2192071>. Acesso em: 29 maio 2008.

- ORACLE. **Oracle Database Sample Schemas**. Disponível em: <http://download.oracle.com/docs/cd/B12037_01/server.101/b10771.pdf>. Acesso em: 22 fev. 2008.

Referência Bibliográfica

- **Oracle PL/SQL Tutorial: Index**. Disponível em: <http://www.java2s.com/Tutorial/Oracle/0180__Index/0020__Create-Index.htm>. Acesso em: 06 jun. 2008.

- **Oracle Roles.** Disponível em: <http://www.psoug.org/reference/roles.html>. Acesso em: 13 jun. 2008.

- SILBERSCHATZ. A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de bancos de dados**. Tradução de Daniel Vieira. Rio de Janeiro: Editora Campus, 2006. Título original: Database system concepts. 5 ed.

- TEOREY, T; LIGHTSTONE, S; NADEAU, T. **Projeto e Modelagem de banco de dados**. Rio de Janeiro: Elsevier, 2007.