Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática
Bases de Dados
2000

Introdução às Bases de Dados Oracle

Pedro Bizarro
bizarro@dei.uc.pt

Introdução às Bases de Dados Oracle

Versão 3.0.0
25 de Setembro de 2000

Engenheiro Pedro Bizarro
bizarro@dei.uc.pt

Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática
Bases de Dados

Conteúdo

Prefácio

Este documento irá servir de apoio às aulas práticas de Base de Dados 1 do Departamento de Engenharia Informática da Faculdade de Ciências da Universidade de Coimbra.

Será introduzida a linguagem SQL segundo o dialecto próprio do servidor de Bases de Dados Oracle 8. O manual divide-se em 7 aulas de SQL cada uma das quais engloba uma parte de matéria com exemplos e uma parte de exercícios a serem resolvidos nas aulas. Foram acrescentadas ainda mais 2 aulas sobre análise e a ferramenta Data Architect.

O manual foi concebido para poder ser lido de uma forma sequencial mas também para ser fácil de consultar para verificar a sintaxe de comandos ou exemplos de como usá-los.

O sucesso deste manual depende em grande parte dos alunos, das suas sugestões e suas correcções. Só assim se poderá manter um documento actualizado numa área de evolução tão rápida como é a nossa.

A todos os que já sugeriram correcções, o meu muito obrigado. A todos os que vão usar o manual pela primeira vez, lembrem-se, o professor não está cá apenas para ensinar; também está cá para aprender; para aprender como se deve ensinar e para aprender como se deve aprender.

Pedro Bizarro

bizarro@dei.uc.pt

Introdução

Em todas as aulas assume-se que os alunos têm acesso a uma BD (base de dados) Oracle 8 através de um cliente de BD, nomeadamente, SQL Worksheet ou SQL Plus. Nos computadores disponíveis presentes tanto nas aulas práticas como nas salas abertas ambos os clientes estão disponíveis. O SQL Plus é um software cliente menos fácil de usar e menos gráfico mas que permite tirar partido de algumas vantagens específicas desse cliente. Como nenhuma dessas vantagens vai ser usada nas aulas práticas, aconselha-se os alunos a usar o SQL Worksheet pela facilidade de utilização nomeadamente devido à característica de os comandos anteriores serem guardados em buffer e serem facilmente repetidos ou alterados.

O SQL Plus (versão 2, 3 ou 8) pode ser começado através de Start→ Program Files→ Oracle for Windows NT→ SQL Plus 8.0. O SQL Worksheet pode ser começado através de Start→ Program Files→ Oracle Enterprise Manager→ SQL Worksheet.

Para aceder ao servidor é necessário usar um dos utilizadores disponíveis para as aulas de Base de Dados. É necessário indicar também qual o serviço (*service*) que se quer aceder. Um serviço, algumas vezes também referido como *alias*, é apenas uma maneira abreviada de referir o computador, o porto e a instância de base de dados que se quer manipular. A criação de serviços, embora muito simples, não será abordada aqui. Tudo o que é necessário saber é que existe um serviço, de nome `bd` que deve ser usada em todas as aulas práticas e que permite a ligação ao servidor.

Existem 12 utilizadores à disposição dos alunos para serem usados nas aulas práticas. Os seus nomes e passwords são: `bd01`/`bd01`, `bd02`/`bd02`, ..., `bd11`/`bd11` e `bd12`/`bd12`.

Cada grupo de alunos deve usar apenas um e só um utilizador e nenhum utilizador deve ser usado por mais do que um grupo. Esta restrição permite impedir conflitos derivados da concorrência natural dos acessos simultâneos às BDs.

Figura 1 - Exemplo de ligação ao servidor usando o SQL Worksheet
É usado o utlizador `bd01`, com password `bd01` e o serviço `bd`.

Em cada conta dos utilizadores devem existir apenas 3 tabelas e mais nenhum objecto (tabela, vista, sequência, etc). As tabelas, o código necessário à sua criação, a inserção de dados e a listagem completas das tabelas são exibidos de imediato.

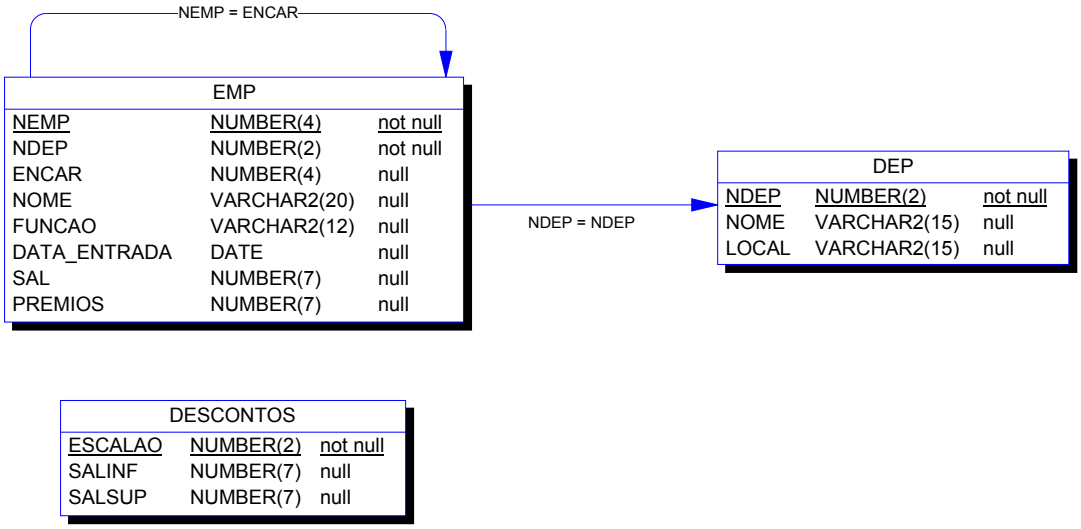


Figura 2 - Tabelas existentes no servidor
As tabelas no servidor pretendem modelar uma empresa com uma base de dados muito simples. Existe apenas informação sobre empregados, departamentos e escalões. Todos os empregados pertencem a um departamento. Esse relação é indicada através da coluna de chave forasteira, `ndep`, na tabela `emp`, que indica um valor existente na coluna de chave primária `ndep` da tabela `dep`. Cada empregado pertence também a um escalão de descontos de IRS. A relação entre empregados e descontos é calculada através do valor do salário do empregado. Assim, diz-se que um empregado, de número `nemp`, pertence a um escalão de número `escalao` se o seu valor de salário se situar entre os valores mínimo e máximo desse escalão; ou seja, apenas se e só se `descontos.salinf<=emp.sal<=descontos.salsup`.

As tabelas já estarão provavelmente criadas em todas as contas `bd01`, `bd02`, ..., `bd12`. Mesmo assim, os comandos para criar as tabelas são:

```
/* Cria a tabela dos descontos
 */
CREATE TABLE descontos
(
 escalao      NUMBER(2)      CONSTRAINT pk_esc_descontos PRIMARY KEY              ,
 salinf       NUMBER(7)      CONSTRAINT nn_inf_descontos CHECK (salinf IS NOT NULL),
 salsup       NUMBER(7)      CONSTRAINT nn_sup_descontos CHECK (salsup IS NOT NULL)
);
/


/* Cria a tabela dos departamentos
 */
CREATE TABLE dep
(
 ndep         NUMBER(2)      CONSTRAINT pk_ndep_dep      PRIMARY KEY              ,
 nome         VARCHAR2(15)   CONSTRAINT nn_nome_dep      CHECK (nome   IS NOT NULL),
 local        VARCHAR2(15)   CONSTRAINT nn_local_dep     CHECK (local  IS NOT NULL)
);
/


/* Cria a tabela dos empregados
 */
CREATE TABLE emp
(nemp         NUMBER(4)      CONSTRAINT pk_nemp_emp      PRIMARY KEY,
 nome         VARCHAR2(20)   CONSTRAINT nn_nome_emp      NOT NULL   ,
 funcao       VARCHAR2(12)   CONSTRAINT nn_funcao_emp    NOT NULL   ,
 encar        NUMBER         CONSTRAINT fk_encar_emp
                                REFERENCES emp(nemp)     NULL       ,
 data_entrada DATE           DEFAULT SYSDATE
                             CONSTRAINT nn_data_emp      NOT NULL   ,
 sal          NUMBER(7)      CONSTRAINT nn_sal_emp       NOT NULL   ,
 premios      NUMBER(7)      DEFAULT NULL                           ,
 ndep         NUMBER(2)      CONSTRAINT nn_ndep_emp      NOT NULL
                             CONSTRAINT fk_ndep_emp
                                REFERENCES dep(ndep)
);
/
```

Figura 3 - Código para criação das tabelas

```
/* Insere os departamentos.
 */
INSERT INTO dep VALUES (10, 'Contabilidade', 'Condeixa');
INSERT INTO dep VALUES (20, 'Investigação',  'Mealhada');
INSERT INTO dep VALUES (30, 'Vendas',        'Coimbra');
INSERT INTO dep VALUES (40, 'Planeamento',   'Montemor');


/* Insere os descontos.
 */
INSERT INTO descontos VALUES (1,  55000,   99999);
INSERT INTO descontos VALUES (2, 100000,  210000);
INSERT INTO descontos VALUES (3, 210001,  350000);
INSERT INTO descontos VALUES (4, 350001,  550000);
INSERT INTO descontos VALUES (5, 550001, 9999999);


/* Altera  o formato  de  inserção  de datas  para  poder
 * inserir as datas neste formato.
 */
ALTER SESSION SET NLS_DATE_FORMAT = 'yy.mm.dd';

/* Insere os empregrados.
 * Note-se  que  como  existe a  restricao  de  o  numero
 * do encarregado ser uma chave forasteira (que por acaso
 * aponta  para a  chave primaria  da  mesma  tabela)  os
 * empregados  teem  que  ser  inseridos na  ordem certa.
 * Primeiro o presidente (que nao tem superiores)  depois
 * os empregados cujo encarregado e' o presidente e assim
 * sucessivamente.
 */
INSERT INTO emp VALUES(1839, 10, 'Jorge Sampaio',  'Presidente'  ,null, '84.02.11', 890000,  null);

INSERT INTO emp VALUES(1566, 20, 'Augusto Reis',   'Encarregado' ,1839, '85.02.13', 450975,  null);
INSERT INTO emp VALUES(1698, 30, 'Duarte Guedes',  'Encarregado' ,1839, '91.11.25', 380850,  null);
INSERT INTO emp VALUES(1782, 10, 'Silvia Teles',   'Encarregado' ,1839, '86.11.03', 279450,  null);

INSERT INTO emp VALUES(1788, 20, 'Maria Dias',     'Analista'    ,1566, '82.11.07', 565000,  null);
INSERT INTO emp VALUES(1902, 20, 'Catarina Silva', 'Analista'    ,1566, '93.04.13', 435000,  null);

INSERT INTO emp VALUES(1499, 30, 'Joana Mendes',   'Vendedor'    ,1698, '84.10.04', 145600, 56300);
INSERT INTO emp VALUES(1521, 30, 'Nelson Neves',   'Vendedor'    ,1698, '83.02.27', 212250, 98500);
INSERT INTO emp VALUES(1654, 30, 'Ana Rodrigues',  'Vendedor'    ,1698, '90.12.17', 221250, 81400);
INSERT INTO emp VALUES(1844, 30, 'Manuel Madeira', 'Vendedor'    ,1698, '85.04.21', 157800,     0);
INSERT INTO emp VALUES(1900, 30, 'Tome Ribeiro',   'Continuo'    ,1698, '94.03.05',  56950,  null);

INSERT INTO emp VALUES(1876, 20, 'Rita Pereira',   'Continuo'    ,1788, '96.02.07',  65100,  null);
INSERT INTO emp VALUES(1934, 10, 'Olga Costa',     'Continuo'    ,1782, '86.06.22',  68300,  null);

INSERT INTO emp VALUES(1369, 20, 'Antonio Silva',  'Continuo'    ,1902, '96.12.22',  70800,  null);
```

Figura 4 - Código para inserção dos valores nas tabelas

Note-se que o comando INSERT permite duas alternativas de sintaxe: uma mais correcta mas mais extensa e outra menos correcta e menos extensa que é a usada no exemplo. Optou-se por esta hipótese para não sobrecarregar a figura.

```
SQLWKS> SELECT * FROM emp;

NEMP    NDEP    NOME           FUNCAO       ENCAR   DATA_ENTRADA SAL     PREMIOS
------- ------- -------------- ------------ ------- ------------ ------- -------
   1839      10 Jorge Sampaio  Presidente           84.02.11      890000
   1566      20 Augusto Reis   Encarregado     1839 85.02.13      450975
   1698      30 Duarte Guedes  Encarregado     1839 91.11.25      380850
   1782      10 Silvia Teles   Encarregado     1839 86.11.03      279450
   1788      20 Maria Dias     Analista        1566 82.11.07      565000
   1902      20 Catarina Silva Analista        1566 93.04.13      435000
   1499      30 Joana Mendes   Vendedor        1698 84.10.04      145600   56300
   1521      30 Nelson Neves   Vendedor        1698 83.02.27      212250   98500
   1654      30 Ana Rodrigues  Vendedor        1698 90.12.17      221250   81400
   1844      30 Manuel Madeira Vendedor        1698 85.04.21      157800       0
   1900      30 Tome Ribeiro   Continuo        1698 94.03.05       56950
   1876      20 Rita Pereira   Continuo        1788 96.02.07       65100
   1934      10 Olga Costa     Continuo        1782 86.06.22       68300
   1369      20 Antonio Silva  Continuo        1902 96.12.22       70800
14 rows selected.


SQLWKS> SELECT * FROM dep;

NDEP    NOME           LOCAL
------- -------------- --------------
     10 Contabilidade  Condeixa
     20 Investigação   Mealhada
     30 Vendas         Coimbra
     40 Planeamento    Montemor
4 rows selected.


SQLWKS> SELECT * FROM descontos;

ESCALAO SALINF  SALSUP
------- ------- -------
      1   55000   99999
      2  100000  210000
      3  210001  350000
      4  350001  550000
      5  550001 9999999
5 rows selected.
```

Figura 5 - Conteúdo das tabelas a usar nas aulas práticas

Caso verifique que a conta do utilizador que está a usar não contem estes dados, use o código fornecido, disponível também na página web da disciplina, e reponha os valores correctos.

Aula 1

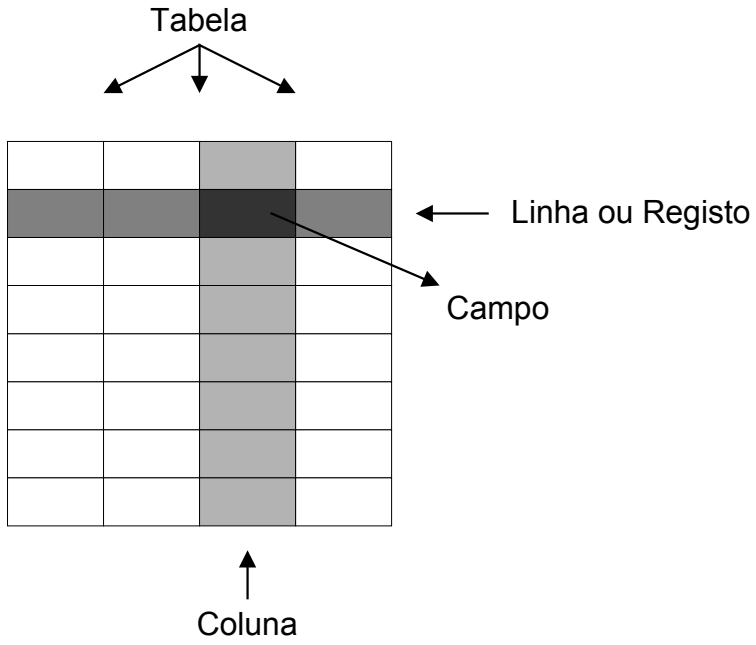
Visão geral da BD e comando SELECT

1.1. Introdução às Bases de Dados relacionais

Uma base de dados é essencialmente um repositório de informação.

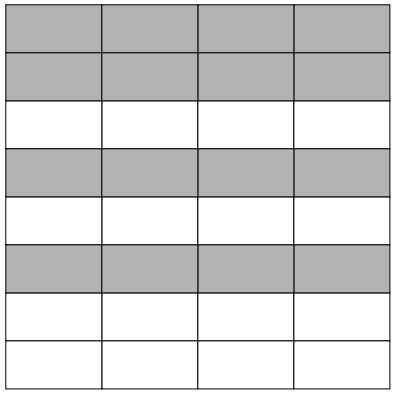
Em 1970, o Dr. E.F. Codd propôs o modelo relacional para bases de dados através de um documento intitulado "*A Relational Data Model of Data for Large Shared Data Banks*".

Uma base de dados relacional consiste num conjunto de tabelas a duas dimensões. Ao todo existem apenas 4 conceitos a compreender: tabela, linha, coluna e campo.



1.2. Relações

Podem definir-se relações entre as tabelas. As relações podem, por sua vez, ser vistas como novas tabelas. As relações possíveis são:



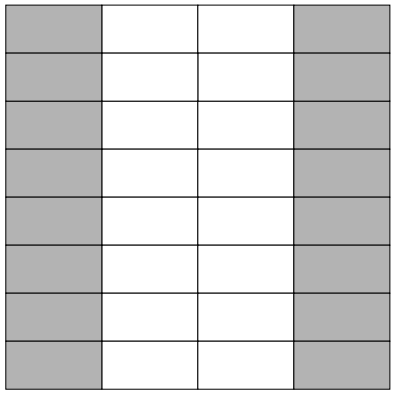
Restrição

1.2.1. Restrição

Subconjunto horizontal. Corresponde a seleccionar uma ou mais linhas da tabela ou conjunto de dados.

1.2.2. Projecção

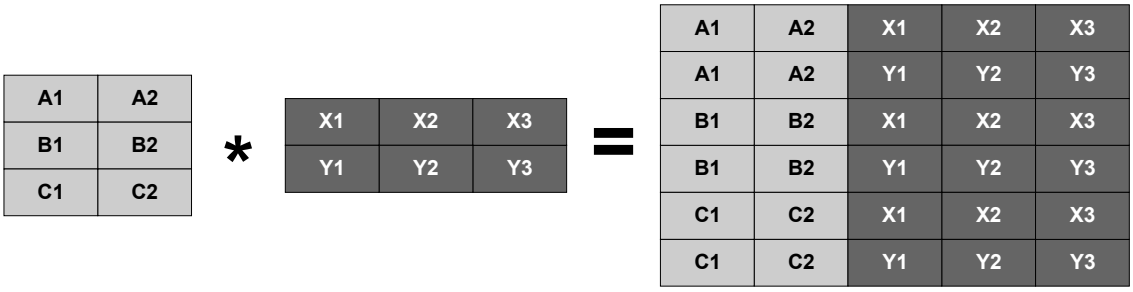
Subconjunto vertical. Corresponde a seleccionar uma ou mais colunas da tabela ou conjunto de dados.



Projecção

1.2.3. Produto de Tabelas

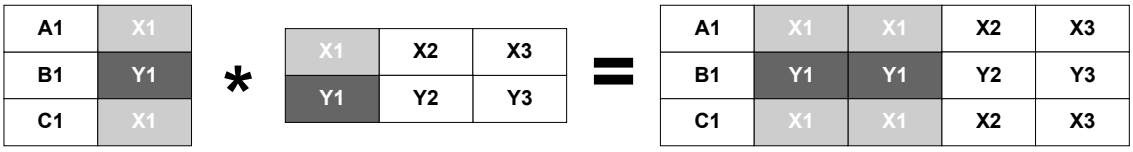
Concatenação de registos de dois conjuntos de dados. São concatenados todos os dados do primeiro conjunto com todos os dados do segundo conjunto. Gera normalmente um resultado muito grande e sem significado. Diz-se que o produto produz dados sem significado porque combina sem qualquer critério dois conjuntos de dados.



Produto

1.2.4. Junção Interna

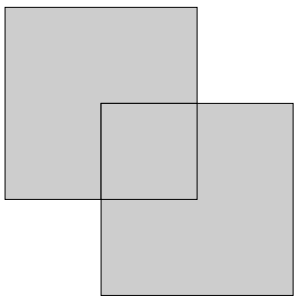
Representa uma concatenação (produto) que segue regras específicas. Isto é, são concatenados apenas alguns dados do primeiro conjunto com apenas alguns dados do segundo conjunto. Pode ver-se como um produto com restrições. Na figura seguinte, pode ver-se a junção como um produto onde se aplica de seguida uma restrição de a segunda coluna da primeira tabela ter um valor igual à primeira coluna da segunda tabela.



Junção

1.2.5. União

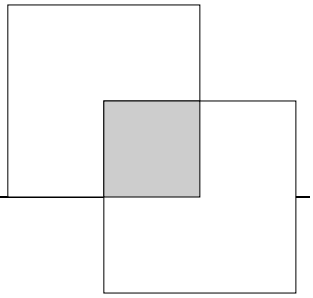
Apresenta resultados que apareçam no primeiro ou no segundo conjunto de dados ou em ambos. Pode imaginar-se como a união normal de conjuntos.



União

1.2.6. Intersecção

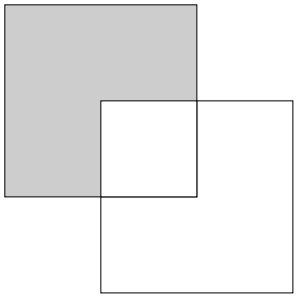
Exibe apenas os dados que pertençam simultaneamente aos dois conjuntos de dados.



Intersecção

1.2.7. Diferença

Apresenta os dados do primeiro conjunto que não aparecem também no segundo conjunto.



Diferença

1.3. Propriedades de uma base de dados relacional

- Para o utilizador/programador, a base de dados é vista apenas como um conjunto de tabelas.
- Existe um conjunto de operadores para separar e combinar as relações.
- Não existem ligações explícitas entre as tabelas. Elas são feitas unicamente através dos dados.
- Os comandos de extracção, inserção e manipulação de dados bem como os de alterações da base de dados estão todos incluídos numa linguagem, a SQL.
- A SQL é uma linguagem não procedimental adaptada à língua inglesa.
- O utilizador não sabe ou não precisa saber o formato ou a localização dos dados. Nem tão pouco precisa saber como (que algoritmo é usado para) obter o acesso aos dados.

1.4. Propriedades das tabelas

Para cada tabela:

- Não existem registos (linhas) duplicados.
- Não existem nomes de colunas duplicados.
- A ordem das colunas é irrelevante.
- A ordem das linhas é irrelevante.
- Os valores dos dados guardados nos campos são não decomponíveis.

1.5. Tipos de dados reconhecidos

Actualmente são reconhecidos pelo Oracle (versão 8) os seguintes tipos de dados:

- **VARCHAR2(*comprimento*)**
  Conjunto de caracteres (string) de tamanho variável. O *comprimento* varia entre um máximo de 2000 caracteres e um mínimo de 1.
- **NUMBER(*p*, *e*)**
  Representa um número com uma precisão de *p* e uma escala de *e*. A precisão *p* pode variar entre 1 e 38 e a escala *e* entre –84 e 127.
- **LONG**
  Conjunto de caracteres de tamanho variável com até 2 gigabytes (2^{31}-1 bytes).
- **DATE**
  Um valor de data entre 1 de Janeiro de 4712 AC e 31 de Dezembro de 4712 DC.
- **RAW(*comprimento*)**
  Dados binários em bruto de comprimento variável. O *comprimento* máximo é de 255 bytes.
- **LONG RAW**
  Dados binários em bruto com um comprimento variável e de tamanho máximo igual a 2 gigabytes.
- **ROWID**
  String hexadecimal que representa o endereço único de uma linha numa tabela.
- **CHAR(*comprimento*)**
  Conjunto de caracteres de tamanho fixo. O *comprimento* máximo é de 255 bytes e o comprimento por omissão é de 1 byte.
- **BLOB, CLOB, NCLOB** e **BFILE**
  Tipos de dados para conteúdos binários até 4 Gigabytes internos ou externos (BFILE) à base de dados.
- **NVARCHAR2**, **MLSLABEL** e **NCHAR**
  Outros tipos de dados possíveis mas menos usados. Consulte a documentação do Oracle.

1.6. Arquitectura do Sistema Geral de Bases de Dados Relacional da Oracle

O Sistema Geral de Bases de Dados Relacional (SGBDR) inclui o gestor de Bases de Dados e várias ferramentas destinadas a fazer o interface entre utilizadores e gestores (ou em inglês, *Data Base Administrators* ou *DBA*s) com a BD.

É da responsabilidade do *kernel* do SGBDR:

- A gestão e armazenamento dos dados
- Controlo e restrição de acessos a dados
- Gestão de concorrência
- Controlo de segurança e recuperação de dados
- Interpretação da linguagem SQL

A linguagem SQL representa o único acesso possível à BD.

A maior parte das vezes, as bases de dados são utilizadas através de computadores clientes com aplicações independentes da base de dados que consultam e alteram a informação e a estrutura desta através de uma ligação de rede.

1.7. Introdução à linguagem SQL

Os principais comandos de SQL são:

Para extrair informação

```
SELECT
```

Para modificar informação

```
INSERT
UPDATE
DELETE
```

Para modificar estruturas de dados

```
CREATE
ALTER
DROP
```

Para controlo de acessos

```
GRANT
REVOKE
```

O SQL não é *case-sensitive*, ou seja, é independente escrever os comandos com letras minúsculas ou maiúsculas ou mesmo misturando ambas. Os nomes das tabelas e das colunas também não são *case-sensitive*. A única coisa que é *case-sensitive* são os dados do tipo caracter.

Assim temos que:

```
SELECT nome, nemp, sal FROM emp;
```

É o mesmo que escrever

```
Select Nome, Nemp, SAL from emp;
```

Ou

```
seLECt nOMe, NEMP, sal fROM EmP;
```

Os comandos de SQL bem como os nomes e as colunas das tabelas podem ser escritos tanto em minúsculas como em maiúsculas.

Mas,

```
SELECT nome, nemp, sal
   FROM emp
   WHERE nome = 'JORGE';
```

já é diferente de ,

```
SELECT nome, nemp, sal
   FROM emp
   WHERE nome = 'jorge';
```

Os dados do tipo caracter têm que corresponder de forma exacta. Letra maiúscula é diferente de letra minúscula.

porque `'JORGE'` (e `'jorge'`) representa um dado do tipo caracter.

Os comandos em SQL terminam apenas com o sinal de ponto e vírgula e podem ocupar mais do que uma linha. Assim, qualquer uma das seguintes formas representa o mesmo comando e todas elas estão correctas sintacticamente. No entanto, para facilitar a leitura aconselhamos que se adopte o estilo usado no último exemplo.

```
SELECT nome, nemp, sal FROM emp WHERE nome = 'jorge';
```

ou,

```
SELECT nome, nemp,
sal FROM
emp
WHERE
nome = 'jorge'
;
```

ou,

```
SELECT nome, nemp, sal
   FROM emp
   WHERE nome = 'jorge';
```

1.8. Comando SELECT

O comando `SELECT` é o mais importante e mais complexo de todos os comandos de SQL. O seu objectivo é o de seleccionar dados, podendo para tal, aplicar vários tipos de relação (restrição, projecção, produto, junção, união, intersecção e diferença) às tabelas existentes na base de dados e executar operações sobre os valores retirados das tabelas antes de os mostrar.

O comando de *query* básico incluí apenas as cláusulas `SELECT` e `FROM`.

1.8.1. Cláusulas SELECT e FROM

A cláusula `SELECT` especifica uma lista de nomes de colunas das tabelas separadas por vírgulas. Permite ainda fazer operações aritméticas, de *strings* e de datas sobre os valores seleccionados. Permite ainda renomear as colunas de dados através de pseudónimos.

A cláusula `FROM` determina de que tabelas se vão buscar os dados.

Exemplos:

- Query simples:

  ```
  SELECT nome, nemp, sal
     FROM emp;
  ```

  Selecciona as colunas `nome`, `nemp` e `sal` da tabela `emp`.

- Uso de expressões aritméticas:

  ```
  SELECT nome, sal * 14
     FROM emp;
  ```

  Selecciona a coluna `nome` e selecciona o resultado da multiplicação da coluna `sal` por 14 (o salário anual já a contar com 13º mês e subsídio de férias)

- Uso de pseudónimos (ou alias em inglês)

```
SELECT nome, sal * 14 "Remuneracao Anual"
   FROM emp;
```

Selecciona as colunas `nome` e a multiplicação da coluna `sal` por 14 (o salário anual já a contar com 13º mês e subsídio de férias) e atribuí o pseudónimo de `"Remuneracao Anual"` à segunda coluna.

- Concatenação de colunas

```
SELECT nome || funcao "Nome mais funcao"
   FROM emp;
```

Junta os valores de nome e função numa string e mostra-os numa coluna cujo nome é `"Nome mais funcao"`.

- Uso de constantes

```
SELECT 'O Exmo Sr. ' ||
        nome ||
       ' trabalha como ' ||
       funcao "Descricao Formal"
   FROM emp;
```

Junta algumas constantes aos campos `nome` e `funcao` e concatena tudo numa única coluna de nome `"Descricao Formal"`.

NOTA: Repare que as *strings* são limitadas por plicas (‘ e ’) e os *alias* são limitados por aspas (“ e ”).

- Tratamento de valores nulos

```
SELECT nome, NVL (premios, 0)
   FROM emp;
```

Quando um determinado campo não tem qualquer valor atribuído, é usado um valor especial, o `NULL`. Um campo a `NULL` significa falta de valor ou falta de informação e não deve ser confundido com a string vazia, ‘’, nem com o valor zero (`0`). Não é possível realizar operações aritméticas ou de caracteres sobre valores nulos. Quando muito, pode substituir o valor nulo por outro valorm usando a função `NVL`, e usar esse valor na operação. A função `NVL` (abreviatura de Null VaLue) substituí o primeiro parâmetro pelo segundo no caso do primeiro ser nulo.

Exemplo:

```
SELECT NVL(nome, 'Sem nome'), sal
   FROM emp;
```

Se o empregado não tiver um nome associado, o valor dessa coluna aparece como `'Sem nome'`.

Exemplo:

```
SELECT nome, sal * 14 + NVL(premios, 0) "Ganho Anual"
   FROM emp;
```

NOTA: As versões actuais do Oracle (até à 8) interpretam a *string* vazia como sendo o valor **`NULL`**. No entanto, isso vai contra os standards de SQL e a Oracle já anunciou que futuramente esse comportamente será descontinuado.

1.8.2. Cláusula DISTINCT

A cláusula `DISTINCT` elimina linhas duplicadas do resultado.

```
SELECT DISTINCT funcao "Mostra que profissoes existem"
   FROM emp;
```

O comando anterior mostra apenas uma lista das profissões existentes. Se não usássemos o `DISTINCT` apareceriam as funções de **todos** os trabalhadores mesmo que existissem funções repetidas.

Exemplos:

```
SQLWKS> SELECT funcao "Com Repetidos"
     2>    FROM emp;

Com Repetido
------------
Presidente
Encarregado
Encarregado
Encarregado
Analista
Analista
Vendedor
Vendedor
Vendedor
Vendedor
Continuo
Continuo
Continuo
Continuo
14 rows selected.

SQLWKS> SELECT DISTINCT funcao "Sem Repetidos"
     2>    FROM emp;

Sem Repetido
------------
Analista
Continuo
Encarregado
Presidente
Vendedor
5 rows selected.


SQLWKS> SELECT DISTINCT funcao, ndep "Profissao e departamento"
     2>    FROM emp
     3>    ORDER BY ndep, funcao;

FUNCAO       Profissao e departamento
------------ ------------------------------
Continuo                              10
Encarregado                           10
Presidente                            10
Analista                              20
Continuo                              20
Encarregado                           20
Continuo                              30
Encarregado                           30
Vendedor                              30
9 rows selected.
```

O comando DISTINCT usa-se apenas uma vez por comando SELECT e a sua localização, é sempre depois da cláusula SELECT e antes da lista de colunas. Aplica-se à lista de colunas seleccionadas.

1.8.3. Cláusula ORDER BY

Uma vez que a ordem das linhas na tabela é irrelevante e uma vez que a ordem do resultado das *queries* depende fortemente do algoritmo de procura usado internamente pelo SGBDR, e uma vez que esse algoritmo é invisível ao utilizador, a única forma de obrigar a uma ordem específica dos resultados é através do uso da cláusula ORDER BY.

Exemplo:

```
SELECT nome, funcao, sal
   FROM emp
   ORDER BY ndep, sal DESC, nome;
```

ORDER BY, a ser usada terá que aparecer depois das cláusulas, SELECT, FROM e WHERE e aplica-se a uma lista de colunas. Os resultados são ordenados primeiro pela primeira coluna da lista referida, e em caso de empate pela segunda coluna referida e assim sucessivamente. A ordenação é ASCendente por omissão a não ser que se especifique que é DESCendente através da palavra DESC à frente da coluna respectiva.

Pode ainda especificar-se, na cláusula ORDER BY, o alias de uma coluna, ou o número de uma coluna. O número da coluna depende da ordem das colunas que aparecem na cláusula SELECT. A primeira coluna é a número 1 e não a número 0.

Exemplo: Ordena primeiro ascendentemente por funcao (2ª coluna), depois descendentemente por "Ganho Anual" e finalmente ascendentemente por nome.

```
SQLWKS> SELECT DISTINCT nome, funcao, sal*14 + nvl(premios, 0) "Ganho Anual"
     2>    FROM emp
     3>    ORDER BY 2 ASC, "Ganho Anual" DESC, nome ASC;
NOME                 FUNCAO       Ganho Anual
-------------------- ------------ ------------
Maria Dias           Analista          7910000
Catarina Silva       Analista          6090000
Antonio Silva        Continuo           991200
Olga Costa           Continuo           956200
Rita Pereira         Continuo           911400
Tome Ribeiro         Continuo           797300
Augusto Reis         Encarregado       6313650
Duarte Guedes        Encarregado       5331900
Silvia Teles         Encarregado       3912300
Jorge Sampaio        Presidente       12460000
Ana Rodrigues        Vendedor          3178900
Nelson Neves         Vendedor          3070000
Manuel Madeira       Vendedor          2209200
Joana Mendes         Vendedor          2094700
14 rows selected.
```

1.8.4. Cláusula WHERE

A cláusula WHERE permite restringir linhas através de uma condição. Apenas as linhas que satisfaçam a condição são devolvidas.

Podem usar-se no WHERE condições sobre uma ou mais colunas de uma ou mais tabelas ou vistas* desde que as tabelas ou vistas apareçam na cláusula FROM.

* As vistas serão introduzidas posteriormente. Para já interessa saber apenas que do ponto de vista de um comando SELECT (salvo algumas excepções), as vistas comportam-se como se fossem tabelas.

Pode comparar-se valores de colunas, expressões aritméticas e constantes. Podem aparecer na cláusula WHERE nomes de colunas que não apareçam na cláusula SELECT.

Não se podem usar pseudónimos de colunas.

Para além de fazer restrições simples sobre uma tabela, o uso mais comum do WHERE é o de permitir relacionar colunas de várias tabelas ou vistas.

A cláusula WHERE tem 3 elementos:

- nome de uma coluna
- um operador de comparação
- um nome de uma coluna, uma constante ou uma lista de valores

A cláusula WHERE, se usada, terá de aparecer depois da FROM.

Os operadores lógicos permitidos são os seguintes:

=	igual
<	menor que
<=	menor ou igual
>	maior
>=	maior ou igual
!=	diferente
<>	diferente

Existem ainda os seguintes operadores SQL:

BETWEEN...AND...	entre dois valores (inclusive)
IN (lista)	corresponde a qualquer elemento da lista
LIKE	cadeia de caracteres
IS [NOT] NULL	se (não) é um valor nulo
NOT (condição)	a negação de uma condição

Para testar mais do que uma condição pode fazer-se uso de ANDs e ORs. Note que a prioridade dos ANDs é maior. Pode-se usar parêntesis para alterar a ordem de execução das comparações.

Exemplos:

- Uso de =

```
SELECT *
   FROM emp
   WHERE ndep = 10;
```

  Devolve todos os empregados do departamento 10.

- Uso de AND e NOT (e diferente)

```
SELECT *
   FROM emp
   WHERE ndep  = 10 AND
         NOT funcao = 'Encarregado';
```

A última linha do SELECT poderia ter sido escrita de qualquer uma das seguintes maneiras:

```
         funcao != 'Encarregado';
```

ou

```
         funcao <> 'Encarregado';
```

- Uso de OR

```
SELECT *
   FROM emp
   WHERE  ndep  = 10
      AND (NOT funcao = 'Encarregado'
      OR  ndep  = 20);
```

- Uso de BETWEEN … AND …

```
SELECT nome, sal
   FROM emp
   WHERE sal BETWEEN 1000 AND 2000;
```

- Uso de IN (lista)

```
SELECT nome, nemp
   FROM emp
   WHERE nome IN ('Jorge Sampaio', 'Augusto Reis',
                  'Duarte Guedes');
```

- Uso de LIKE <cadeia_de_caracteres>

```
SELECT nome, nemp
   FROM emp
   WHERE nome LIKE 'A%'
      OR nome LIKE '_O%';
```

O caracter especial '%' representa qualquer cadeia de caracteres. O caracter especial '_' representa UM caracter com qualquer valor. Assim, o comando anterior devolve as linhas dos empregados com cujo nome começa por 'A' ou em que a segunda letra do nome é um 'O'. Só se usa o operador LIKE quando se pretende fazer comparações com caracteres e strings que usem os caracteres especiais de comparação '%' ou '_'.

- Uso de `IS NULL`

```
SELECT nome "Sem Premios"
   FROM emp
   WHERE premios IS NULL;

   Sem Premios
   --------------------
   Jorge Sampaio
   Augusto Reis
   Duarte Guedes
   Silvia Teles
   Maria Dias
   Catarina Silva
   Tome Ribeiro
   Rita Pereira
   Olga Costa
   Antonio Silva
   10 rows selected.
```

- Uso de `IS NOT NULL`

```
SELECT nome "Com Premios"
   FROM emp
   WHERE premios IS NOT NULL;

Com Premios
--------------------
Joana Mendes
Nelson Neves
Ana Rodrigues
Manuel Madeira
4 rows selected.
```

Poder-se-ia ter usado `NOT IS NULL` em vez de `IS NOT NULL`.

NOTA: Note que não se pode usar premios = NULL nem premios <> NULL. Como NULL não é um número, não pode ser comparado com o valor de premios. O resultado de qualquer das duas condições anteriores é sempre falso independentemente do valor de premios. Exemplos:

```
SQLWKS> SELECT nome "Sem Premios"
     2>    FROM emp
     3>    WHERE premios = NULL;

Sem Premios
--------------------
0 rows selected.


SQLWKS> SELECT nome "Com Premios"
     2>    FROM emp
     3>    WHERE premios <> NULL;

Com Premios
--------------------
0 rows selected.
```

1.9. Exercícios

Considere a base de dados de demonstração cujo esquema é fornecido em anexo.

1. Seleccione toda a informação da tabela `DEP`. O resultado deve ser similar ao que se segue:

```
NDEP       NOME           LOCAL
---------- -------------- --------------
        10 Contabilidade  Condeixa
        20 Investigação   Mealhada
        30 Vendas         Coimbra
        40 Planeamento    Montemor

4 rows selected.
```

Resposta:

2. Mostre a lista de todos os empregados contendo o nome de cada empregado, a sua função, o salário e o número do departamento a que pertence.

```
NOME                 FUNCAO       SAL        NDEP
-------------------- ------------ ---------- ----------
Jorge Sampaio        Presidente       890000         10
Augusto Reis         Encarregado      450975         20
Duarte Guedes        Encarregado      380850         30
Silvia Teles         Encarregado      279450         10
Maria Dias           Analista         565000         20
Catarina Silva       Analista         435000         20
Joana Mendes         Vendedor         145600         30
Nelson Neves         Vendedor         212250         30
Ana Rodrigues        Vendedor         221250         30
Manuel Madeira       Vendedor         157800         30
Tome Ribeiro         Continuo          56950         30
Rita Pereira         Continuo          65100         20
Olga Costa           Continuo          68300         10
Antonio Silva        Continuo          70800         20

14 rows selected.
```

Resposta:

3. Apresente a lista de todos os empregados (nome, número de departamento e salário) cujo salário está entre 150000 e 300000.

```
NOME                 NDEP       SAL
-------------------- ---------- ----------
Silvia Teles                 10     279450
Nelson Neves                 30     212250
Ana Rodrigues                30     221250
Manuel Madeira               30     157800
4 rows selected.
```

Resposta:

4. Apresente a lista de todos os departamentos ordenados decrescentemente por número de departamento.

```
NDEP       NOME            LOCAL
---------- --------------- ---------------
        40 Planeamento     Montemor
        30 Vendas          Coimbra
        20 Investigação    Mealhada
        10 Contabilidade   Condeixa

4 rows selected.
```

Resposta:

5. Mostre a lista de todas as funções existentes na empresa. Devem ser excluídas as repetições.

```
FUNCAO
------------
Analista
Continuo
Encarregado
Presidente
Vendedor

5 rows selected.
```

Resposta:

6. Apresente a lista de todos os empregados que recebem prémios, devendo a lista conter o nome do empregado, a sua função e o montante recebido em prémios.

```
NOME                 FUNCAO        PREMIOS
-------------------- ------------ ----------
Joana Mendes         Vendedor           56300
Nelson Neves         Vendedor           98500
Ana Rodrigues        Vendedor           81400
Manuel Madeira       Vendedor               0

4 rows selected.
```

Resposta:

7. Mostre a informação detalhada (toda a informação disponível) dos empregados dos que pertencem ao departamento 10 ou 30.

```
NEMP    NOME            FUNCAO      ENCAR DATA_ENTR SAL     PREMIOS    NDEP
------- --------------- ----------- ----- --------- ------- ------- -------
   1839 Jorge Sampaio   Presidente        11-FEB-84  890000              10
   1698 Duarte Guedes   Encarregado  1839 25-NOV-91  380850              30
   1782 Silvia Teles    Encarregado  1839 03-NOV-86  279450              10
   1499 Joana Mendes    Vendedor     1698 04-OCT-84  145600   56300      30
   1521 Nelson Neves    Vendedor     1698 27-FEB-83  212250   98500      30
   1654 Ana Rodrigues   Vendedor     1698 17-DEC-90  221250   81400      30
   1844 Manuel Madeira  Vendedor     1698 21-APR-85  157800       0      30
   1900 Tome Ribeiro    Continuo     1698 05-MAR-94   56950              30
   1934 Olga Costa      Continuo     1782 22-JUN-86   68300              10

9 rows selected.
```

Resposta:

8. Mostre os nomes de todos os analistas que trabalham no departamento 20.

```
NOME
--------------------
Maria Dias
Catarina Silva

2 rows selected.
```

Resposta:

9. Apresente a lista de funcionários (nome e função) cujos nomes aparecem as letras 'v' ou 'u'.

```
NOME                 FUNCAO
-------------------- ------------
Augusto Reis         Encarregado
Duarte Guedes        Encarregado
Silvia Teles         Encarregado
Catarina Silva       Analista
Nelson Neves         Vendedor
Ana Rodrigues        Vendedor
Manuel Madeira       Vendedor
Antonio Silva        Continuo

8 rows selected.
```

Resposta:

10. Mostre agora a lista de funcionários (nome e função) cujos nomes começam pela letra 'A' e contêm a sequência de letras 'us' e também os que começando por 'R' contêm a sequência de letras 'ei'. Tenha em atenção as diferenças de maísculas e minúsculas.

```
NOME                 FUNCAO
-------------------- ------------
Augusto Reis         Encarregado
Rita Pereira         Continuo

2 rows selected.
```

Resposta:

11. Mostre a lista da remuneração anual de todos os empregados contendo o nome do empregado, a função. A remuneração anual consiste em 14 vezes o salário mais o valor dos prémios, se existir.

```
Nome                 Função       Remuneração Anual
-------------------- ------------ -----------------
Jorge Sampaio        Presidente            12460000
Augusto Reis         Encarregado            6313650
Duarte Guedes        Encarregado            5331900
Silvia Teles         Encarregado            3912300
Maria Dias           Analista               7910000
Catarina Silva       Analista               6090000
Joana Mendes         Vendedor               2094700
Nelson Neves         Vendedor               3070000
Ana Rodrigues        Vendedor               3178900
Manuel Madeira       Vendedor               2209200
Tome Ribeiro         Continuo                797300
Rita Pereira         Continuo                911400
Olga Costa           Continuo                956200
Antonio Silva        Continuo                991200

14 rows selected.
```

Resposta:

12. Mostre a lista de vendedores cujos prémios foram menores do que 10% da remuneração anual (sal * 14 + premios). O resultado deve incluir o nome do vendedor, 10% da sua remuneração anual e ainda os prémios, e deve ser ordenado crescentemente pelos 10% de remuneração anual. No caso de haver vários vendedores com a mesma remuneração anual, estes devem surgir ordenados pelo nome do vendedor.

```
NOME                 10% Sal. Anual   PREMIOS
-------------------- -------------- ---------
Joana Mendes                209470     56300
Manuel Madeira              220920         0
Nelson Neves                307000     98500
Ana Rodrigues               317890     81400
4 linhas seleccionadas.
```

Resposta:

extra1: mostre os empregados cujo ultimo nome começe por M e tenham simultaneamente remuneracao anual superior a 2,100,000 ou premios acima de 1000, bem como tambem os empregados que tenham encarregado, sejam eles proprios encarregados e que tenham entrado na empresa depois de Janeiro de 1986.

extra2: mostre, para todos os empregados, o nome e o numero de anos e de meses e de dias passados na empresa, ordenados pelo numero de dias

Aula 2

Extrair dados de mais de uma tabela

2.1. Junções

Para além das restrições (subconjunto de linhas) e projecções (subconjuntos de colunas), a operação mais comum é a junção. Uma junção corresponde a seleccionar dados de mais de uma tabela ao mesmo tempo (ver também *Junção* na página *7*). Nas junções existe uma relação entre os dados de uma tabela e os dados da outra.

Se se fizer a selecção de dados de várias tabelas sem incluir na cláusula `WHERE` as condições de relação entre as várias tabelas obtém um produto de tabelas (ver também *Produto de Tabelas* na página *10*). De facto, este é um dos erros mais frequentes nos iniciados em junções. Note-se que ao obter-se o produto está a obter-se todas as combinações possíveis de dados de uma tabela com os dados de outra(s) tabela(s). Uma técnica simples é a de garantir que numa junção existam sempre *n-1* restrições com *n* a representar o número de tabelas incluídas na junção*.

Existem dois tipos de junções: as equi-junções e as não equi-junções.

2.1.1. Equi-Junção

Existe uma equi-junção quando a relação entre a(s) coluna(s) das duas tabelas é a de igualdade. Exemplo:

```
SELECT nemp, sal, local
   FROM emp, dep
   WHERE emp.ndep = dep.ndep;
```

Neste caso estão a juntar-se as linhas da tabela `emp` com todas as linhas da tabela `dep` que verifiquem a condição de o valor de `ndep` da tabela `emp` ser igual ao valor de `ndep` da tabela `dep`. O resultado seria:

```
NEMP       SAL        LOCAL
---------- ---------- ---------------
      1839     890000 Condeixa
      1566     450975 Mealhada
      1698     380850 Coimbra
      1782     279450 Condeixa
      1788     565000 Mealhada
      1902     435000 Mealhada
      1499     145600 Coimbra
      1521     212250 Coimbra
      1654     221250 Coimbra
      1844     157800 Coimbra
      1900      56950 Coimbra
      1876      65100 Mealhada
      1934      68300 Condeixa
      1369      70800 Mealhada

14 rows selected.
```

* Esta regra só é verdade para tabelas sem chaves concatenadas.

2.1.2. Não Equi-Junção

Existe uma não equi-junção quando é usado um comparador que não seja o de igualdade. Exemplo:

```
SELECT nome, sal, premios, escalao
   FROM emp, descontos
   WHERE sal BETWEEN salinf AND salsup;
```

Neste caso estão a juntar-se as linhas da tabela `emp` com todas as linhas da tabela `descontos` que verifiquem a condição de o valor de `sal` da tabela `emp` estar contido no intervalo de `salinf` a `salsup` da tabela `dep`. O resultado seria:

```
NOME                 SAL        PREMIOS    ESCALAO
-------------------- ---------- ---------- ----------
Tome Ribeiro              56950                     1
Rita Pereira              65100                     1
Olga Costa                68300                     1
Antonio Silva             70800                     1
Joana Mendes             145600      56300          2
Manuel Madeira           157800          0          2
Silvia Teles             279450                     3
Nelson Neves             212250      98500          3
Ana Rodrigues            221250      81400          3
Augusto Reis             450975                     4
Duarte Guedes            380850                     4
Catarina Silva           435000                     4
Jorge Sampaio            890000                     5
Maria Dias               565000                     5

14 rows selected.
```

2.1.3. Utilização de pseudónimos de colunas

Se existirem, nas diferentes tabelas, colunas com o mesmo nome é necessário distingui-las. Por exemplo, o seguinte comando não executa:

```
SELECT nome, sal, ndep, nome, local
   FROM emp, dep
   WHERE emp.ndep = dep.ndep;
```

É devolvido o código de erro: `ORA-00918: column ambiguously defined`.

Isto acontece porque existem colunas na cláusula `SELECT` que o Oracle não consegue determinar se pertencem à tabela `emp` ou à tabela `dep`. Ambas as tabelas têm colunas chamadas `nome` e `ndep`. Assim, poder-se-ia corrigir o comando anterior fazendo a correcção para:

```
SELECT emp.nome, sal, emp.ndep, dep.nome, local
   FROM emp, dep
   WHERE emp.ndep = dep.ndep;
```

Ou seja, basta acrescentar antes do nome da coluna, o nome da tabela a que ela pertence e um ponto. Para não tornar demasiado pesada a digitação de comandos e para evitar a repetição dos prefixos `emp.` e `dep.` vezes sem conta, o Oracle permite o uso de pseudónimos de tabelas. Um pseudónimo representa apenas um nome alternativo para a mesma tabela. Com pseudónimos, o comando anterior seria alterado para:

```
SELECT e.nome, sal, e.ndep, d.nome, local
   FROM emp e, dep d
   WHERE e.ndep = d.ndep;
```

Normalmente usa-se como pseudónimo a(s) primeira(s) letra(s) do nome da tabela embora se possa usar qualquer nome válido. Depois de se definir um pseudónimo num comando `SELECT` não se pode fazer referência à tabela sem ser através desse pseudónimo. Note-se ainda que nem todas as colunas da cláusula `SELECT` são precedidas do nome da tabela a que pertencem. De facto só é obrigatório usar o nome da tabela quando não é possível determinar a que tabela pertence determinada coluna. Por exemplo, sabe-se que a coluna `sal` pertence de certeza à tabela `emp` pois não existe mais nenhuma coluna com esse nome em nenhuma das tabelas escolhidas na cláusula `FROM`.

2.2. Outras Formas de Junção

Para além das equi-junções e não equi-junções, podemos ainda reunir dados de mais do que uma tabela recorrendo a junções externas e a operações sobre conjuntos de resultados (união, intersecção e diferença). Podemos ainda fazer uma junção entre uma tabela e ela própria.

2.2.1. Junção Externa

Numa junção, se um registo não satisfaz a condição de junção não aparece no resultado. Se se executar o comando:

```
SELECT e.nome "Nome", e.ndep "NDep", d.nome "Dep"
   FROM emp e, dep d
   WHERE e.ndep = d.ndep
   ORDER BY e.ndep;
```

Obtém-se o resultado:

```
Nome                 NDep       Dep
-------------------- ---------- ---------------
Jorge Sampaio                10 Contabilidade
Silvia Teles                 10 Contabilidade
Olga Costa                   10 Contabilidade
Augusto Reis                 20 Investigação
Rita Pereira                 20 Investigação
Catarina Silva               20 Investigação
Maria Dias                   20 Investigação
Antonio Silva                20 Investigação
Duarte Guedes                30 Vendas
Joana Mendes                 30 Vendas
Nelson Neves                 30 Vendas
Ana Rodrigues                30 Vendas
Manuel Madeira               30 Vendas
Tome Ribeiro                 30 Vendas

14 rows selected.
```

Ou seja, não aparece nenhuma referência ao departamento 40 porque nenhum empregado trabalha nesse departamento.

Isto acontece porque as junções normais (também chamadas de junções internas, ou *inner joins* apresentam apenas os valores de ambas as tabelas que estão relacionados entre si. Uma junção interna implica que os dados que aparecem estejam relacionados.

No entanto, algumas vezes, pretende-se mostrar, além dos dados da junção interna, todos os dados de uma tabela mesmo que não exista nenhuma registo na outra tabela que se relacione com esse. A solução é usar uma junção externa ou *outer join*.

Os registos em falta de uma tabela podem aparecer no resultado se se usar um operador de junção externa na condição de junção. Esse operador consiste num sinal mais entre parêntesis, `(+)`, colocado do lado da tabela que não tem registos para suficientes para se relacionar com a outra. Desta forma será criado um registo com todos os campos a `NULL` na tabela com o (+) para cada um dos registos da outra tabela para o qual não houvesse par. No caso concreto, será mostrado um registo a `NULL` na tabela `emp` que se vai ligar ao registo do departamento 40 da tabela `dep`.

Executando então o comando:

```
SELECT e.nome "Nome", e.ndep "NDep", d.nome "Dep"
   FROM emp e, dep d
   WHERE e.ndep (+) = d.ndep
   ORDER BY e.ndep;
```

Obtemos as 14 linhas correspondentes aos empregados e respectivos departamentos mais uma linha com um empregado a `NULL` a corresponder ao departamento 40:

```
Nome                 NDep       Dep
-------------------- ---------- ---------------
Jorge Sampaio                10 Contabilidade
Silvia Teles                 10 Contabilidade
Olga Costa                   10 Contabilidade
Augusto Reis                 20 Investigação
Rita Pereira                 20 Investigação
Catarina Silva               20 Investigação
Maria Dias                   20 Investigação
Antonio Silva                20 Investigação
Duarte Guedes                30 Vendas
Joana Mendes                 30 Vendas
Nelson Neves                 30 Vendas
Ana Rodrigues                30 Vendas
Manuel Madeira               30 Vendas
Tome Ribeiro                 30 Vendas
                                Planeamento
15 rows selected.
```

NOTA: Repare que o último campo da coluna `ndep` aparece a NULL. Isso acontece porque se usa o valor de `ndep` da tabela `emp` (que tem o símbolo de junção externa). Se se usasse o valor de `ndep` da tabela `dep`, iria aparecer o número do departamento do `'Planeamento'`. Exemplo:

```
SELECT e.nome "Nome", d.ndep "NDep", d.nome "Dep"
   FROM emp e, dep d
   WHERE e.ndep (+) = d.ndep
   ORDER BY e.ndep;
Nome                 NDep       Dep
-------------------- ---------- ---------------
Jorge Sampaio                10 Contabilidade
Silvia Teles                 10 Contabilidade
[...]
Tome Ribeiro                 30 Vendas
                             40 Planeamento
```

2.2.2. Juntar Uma Tabela Consigo Própria

Fazer uma junção de uma tabela com ela própria é um caso particular de junção. A única diferença é que na cláusula `FROM` aparece referida a mesma tabela mais do que uma vez. Quando tal acontece é obrigatório o uso de pseudónimos de tabelas para as distinguir. Por exemplo, para mostrar o encarregado de cada empregado pode executar-se o seguinte comando:

```
SELECT e1.nome "Empregado",   e1.nemp "N Emp",
       e2.nome "Encarregado", e2.nemp "N Encar"
   FROM emp e1, emp e2
   WHERE e1.encar = e2.nemp (+);
```

Que resulta em:

```
Empregado            N Emp      Encarregado          N Encar
-------------------- ---------- -------------------- ----------
Jorge Sampaio              1839
Augusto Reis               1566 Jorge Sampaio              1839
Duarte Guedes              1698 Jorge Sampaio              1839
Silvia Teles               1782 Jorge Sampaio              1839
Maria Dias                 1788 Augusto Reis               1566
Catarina Silva             1902 Augusto Reis               1566
Joana Mendes               1499 Duarte Guedes              1698
Nelson Neves               1521 Duarte Guedes              1698
Ana Rodrigues              1654 Duarte Guedes              1698
Manuel Madeira             1844 Duarte Guedes              1698
Tome Ribeiro               1900 Duarte Guedes              1698
Rita Pereira               1876 Maria Dias                 1788
Olga Costa                 1934 Silvia Teles               1782
Antonio Silva              1369 Catarina Silva             1902

14 rows selected.
```

Repare-se que aqui não só se faz a junção de `emp` consigo própria como se usa uma junção externa para poder mostrar-se a linha com o Presidente da empresa que não tem nenhum superior (encarregado).

2.2.3. Operação sobre Conjuntos – União, Intersecção e Diferença

Os operadores de conjuntos `UNION`, `UNION ALL` (ver *União* na página *10*), `INTERSECT` (ver *Intersecção* na página *10*) e `MINUS` (ver *Diferença* na página *11*) permitem que se construam comandos com resultados de diferentes `SELECT`s combinados. Os comandos de `SELECT` a combinar podem mesmo referir-se a tabelas diferentes.

- Exemplo com `UNION`

```
SELECT funcao
   FROM emp
   WHERE ndep = 10
UNION
SELECT funcao
   FROM emp
   WHERE ndep = 30
ORDER BY funcao;
```

Devolve:

```
FUNCAO
------------
Continuo
Encarregado
Presidente
Vendedor
4 rows selected.
```

Note que independentemente de quantos `SELECT`s se tiver só existe uma única cláusula de `ORDER BY` que terá que aparecer no fim. Note também que existe apenas um ponto e virgula no fim do comando todo.

Note que o primeiro `SELECT`, se executado sozinho, devolveria:

```
FUNCAO
------------
Presidente
Encarregado
Continuo
3 rows selected.
```

Enquanto que o segundo devolveria:

```
FUNCAO
------------
Encarregado
Vendedor
Vendedor
Vendedor
Vendedor
Continuo
6 rows selected.
```

Ou seja, o `UNION` elimina as linhas repetidas. Para não eliminar as linhas repetidas deve usar-se o `UNION ALL`. Veja o exemplo seguinte.

- Exemplo com `UNION ALL`

```
FUNCAO
------------
Continuo
Continuo
Encarregado
Encarregado
Presidente
Vendedor
Vendedor
Vendedor
Vendedor
9 rows selected.
```

- Outro exemplo com `UNION`

```
SELECT funcao "Misturada"
   FROM emp
   WHERE funcao LIKE 'C%'
UNION
SELECT nome "Nome Emp"
   FROM emp
   WHERE nome LIKE 'C%'
UNION
SELECT nome "Nome Dep"
   FROM dep
   WHERE nome LIKE 'C%'
ORDER BY 1;
```

Devolve:

```
Misturada
--------------------
Catarina Silva
Contabilidade
Continuo
3 rows selected.
```

Repare como se podem misturar colunas diferentes de tabelas diferentes num único resultado. Note ainda que a cláusula `ORDER BY` terá que ter uma referência numérica (no comando anterior leia-se "ordenar pela primeira coluna"). Repare também que no caso de existirem pseudónimos aparecem os usados no primeiro `SELECT` mesmo que este não apresente as suas linhas em primeiro no resultado.

- O `INTERSECT` devolve o resultado comum (a intersecção) dos dois comandos. Exemplo com `INTERSECT`:

```
SELECT funcao
   FROM emp
   WHERE ndep = 10
INTERSECT
SELECT funcao
   FROM emp
   WHERE ndep = 30
ORDER BY funcao;
```

Devolve

```
FUNCAO
------------
Continuo
Encarregado
2 rows selected.
```

- O operador `MINUS` retira ao resultado do primeiro `SELECT` o obtido pelo segundo. Exemplo com `MINUS` :

```
SELECT funcao
   FROM emp
   WHERE ndep = 10
MINUS
SELECT funcao
   FROM emp
   WHERE ndep = 30
ORDER BY funcao;
```

Devolve:

```
FUNCAO
------------
Presidente
1 row selected.
```

2.3. Exercícios

Considere a base de dados de demonstração cujo esquema é fornecido em anexo.

1. Mostre os nomes dos empregados, a sua função e o nome do departamento em que cada empregado trabalha. O resultado deve estar ordenado pelo nome de departamento e dentro de cada departamento pelo nome do empregado. O resultado deve ser semelhante ao que se segue:

```
NOME                 FUNCAO       NOME
-------------------- ------------ ---------------
Jorge Sampaio        Presidente   Contabilidade
Olga Costa           Continuo     Contabilidade
Silvia Teles         Encarregado  Contabilidade
Antonio Silva        Continuo     Investigação
Augusto Reis         Encarregado  Investigação
Catarina Silva       Analista     Investigação
Maria Dias           Analista     Investigação
Rita Pereira         Continuo     Investigação
Ana Rodrigues        Vendedor     Vendas
Duarte Guedes        Encarregado  Vendas
Joana Mendes         Vendedor     Vendas
Manuel Madeira       Vendedor     Vendas
Nelson Neves         Vendedor     Vendas
Tome Ribeiro         Continuo     Vendas
14 rows selected.
```

Resposta:

1(b) faça o mesmo usando JOIN

2. Apresente o nome de empregado, o salário, assim como o número e o nome do departamento de todos os empregados cujo nome começa por `'A'` e o apelido por `'R'`. Assuma que os nome dos empregados são todos constituídos por apenas um nome próprio e um apelido, i.e., não têm nomes do meio.

```
NOME                 SAL        NDEP       NOME
-------------------- ---------- ---------- ---------------
Augusto Reis             450975         20 Investigação
Ana Rodrigues            221250         30 Vendas
2 rows selected.
```

Resposta:

Desafio: Tente resolver o problema para uma situação em que os empregados possam ter mais do que 2 nomes.

3. Apresente o nome, salário, nome do departamento e respectiva localização para todos os empregados cujo salário é inferior a 150000.

```
NOME                 SAL        NOME            LOCAL
-------------------- ---------- --------------- ---------------
Joana Mendes             145600 Vendas          Coimbra
Tome Ribeiro              56950 Vendas          Coimbra
Rita Pereira              65100 Investigação    Mealhada
Olga Costa                68300 Contabilidade   Condeixa
Antonio Silva             70800 Investigação    Mealhada
5 rows selected.
```

Resposta:

4. Mostre o escalão de descontos (ou escalões) de cada função. O resultado deve ficar ordenado por escalão e dentro de cada escalão por função. Como pode existir mais do que uma pessoa por função dentro do mesmo escalão poderiam aparecer linhas repetidas. Garanta que NÃO aparecem linhas repetidas.

```
FUNCAO       ESCALAO
------------ ----------
Continuo              1
Vendedor              2
Encarregado           3
Vendedor              3
Analista              4
Encarregado           4
Analista              5
Presidente            5
8 rows selected.
```

Resposta:

5. Mostre o nome, função e salário de todos os empregados de escalão salarial igual a 4, sendo o resultado ordenado por nome de empregado.

```
NOME                 FUNCAO       SAL
-------------------- ------------ ----------
Augusto Reis         Encarregado      450975
Catarina Silva       Analista         435000
Duarte Guedes        Encarregado      380850
3 rows selected.
```

Resposta:

6. A mesma informação que na questão anterior mas agora mostre também o nome do departamento de cada empregado.

```
NOME                 FUNCAO       DEPARTAMENTO    SAL
-------------------- ------------ --------------- ----------
Augusto Reis         Encarregado  Investigação        450975
Catarina Silva       Analista     Investigação        435000
Duarte Guedes        Encarregado  Vendas              380850
3 rows selected.
```

Resposta:

7. Mostre o nome, função, salário e local de trabalho de todos os empregados de `'Coimbra'` e cujo salário é superior a 150000.

```
NOME                 FUNCAO       SAL        LOCAL
-------------------- ------------ ---------- ---------------
Duarte Guedes        Encarregado      380850 Coimbra
Nelson Neves         Vendedor         212250 Coimbra
Ana Rodrigues        Vendedor         221250 Coimbra
Manuel Madeira       Vendedor         157800 Coimbra
4 rows selected.
```

Resposta:

8. Apresente o nome, função, escalão salarial e nome de departamento para todos os empregados com excepção dos empregados cuja função é `'Continuo'`. O resultado deve ficar ordenado por ordem decrescente de escalão salarial.

```
NOME                 FUNCAO       ESCALAO    DEPARTAMENTO
-------------------- ------------ ---------- ---------------
Jorge Sampaio        Presidente            5 Contabilidade
Maria Dias           Analista              5 Investigação
Augusto Reis         Encarregado           4 Investigação
Duarte Guedes        Encarregado           4 Vendas
Catarina Silva       Analista              4 Investigação
Silvia Teles         Encarregado           3 Contabilidade
Nelson Neves         Vendedor              3 Vendas
Ana Rodrigues        Vendedor              3 Vendas
Joana Mendes         Vendedor              2 Vendas
Manuel Madeira       Vendedor              2 Vendas
10 rows selected.
```

Resposta:

9. Faça novamente o exercício 1 mas apresente também o nome dos departamentos onde não exista nenhum empregado a trabalhar.

```
EMPREGADO            FUNCAO       DEPARTAMENTO
-------------------- ------------ ---------------
Jorge Sampaio        Presidente   Contabilidade
Silvia Teles         Encarregado  Contabilidade
Olga Costa           Continuo     Contabilidade
Augusto Reis         Encarregado  Investigação
Rita Pereira         Continuo     Investigação
Catarina Silva       Analista     Investigação
Maria Dias           Analista     Investigação
Antonio Silva        Continuo     Investigação
                                  Planeamento
Duarte Guedes        Encarregado  Vendas
Joana Mendes         Vendedor     Vendas
Nelson Neves         Vendedor     Vendas
Ana Rodrigues        Vendedor     Vendas
Manuel Madeira       Vendedor     Vendas
Tome Ribeiro         Continuo     Vendas
15 rows selected.
```

Resposta:

10. Mostre uma lista dos encarregados e seus subordinados. Ordene os resultados por nome de encarregado e depois por nome de empregado.

```
ENCARREGADO          EMPREGADO
-------------------- --------------------
Augusto Reis         Catarina Silva
Augusto Reis         Maria Dias
Catarina Silva       Antonio Silva
Duarte Guedes        Ana Rodrigues
Duarte Guedes        Joana Mendes
Duarte Guedes        Manuel Madeira
Duarte Guedes        Nelson Neves
Duarte Guedes        Tome Ribeiro
Jorge Sampaio        Augusto Reis
Jorge Sampaio        Duarte Guedes
Jorge Sampaio        Silvia Teles
Maria Dias           Rita Pereira
Silvia Teles         Olga Costa
13 rows selected.
```

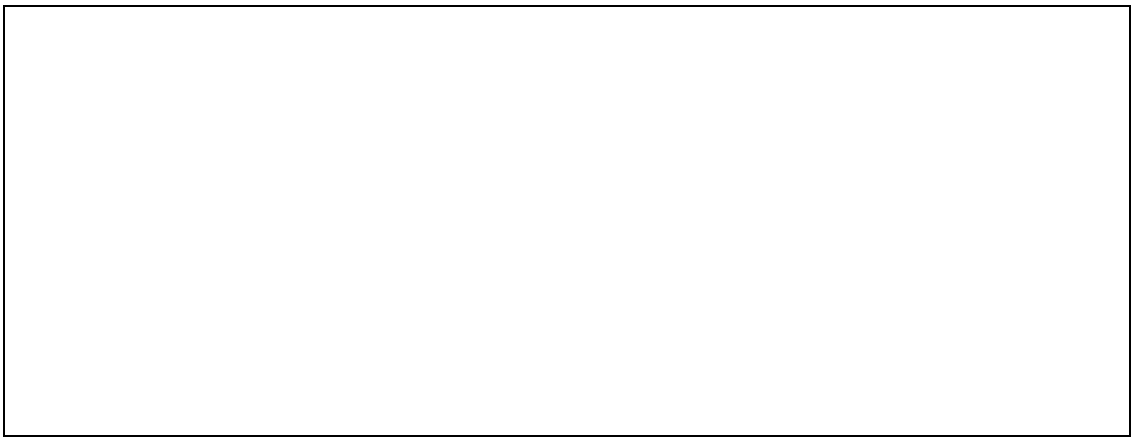
Resposta:

11. Produza o seguinte resultado que consiste em todos os nomes e números dos departamentos e em todos os nomes e números dos empregados. Note que foram acrescentadas linhas antes da sequência de departamentos e antes da sequência de empregados para aumentar a visibilidade. Note ainda que nas linhas acrescentadas foi inserido um número na segunda coluna para numa ordenação sobre essa coluna o resultado aparecer como desejado.

Sugestão: Faça uso de constantes para produzir as quatro linhas acrescentadas.

```
NOMES                NUMEROS
-------------------- ----------
                              0
DEPARTAMENTOS:                1
Contabilidade                10
Investigação                 20
Vendas                       30
Planeamento                  40
                            999
EMPREGADOS:                1000
Antonio Silva              1369
Joana Mendes               1499
Nelson Neves               1521
Augusto Reis               1566
Ana Rodrigues              1654
Duarte Guedes              1698
Silvia Teles               1782
Maria Dias                 1788
Jorge Sampaio              1839
Manuel Madeira             1844
Rita Pereira               1876
Tome Ribeiro               1900
Catarina Silva             1902
Olga Costa                 1934
22 rows selected.
```

Resposta:



12. Mostre o departamento que não tem empregados usando o operador `MINUS`.

```
NDEP       NOME            LOCAL
---------- --------------- ---------------
        40 Planeamento     Montemor
1 row selected.
```

Resposta:

13. Mostre o nome do empregado, departamento em que trabalha e escalão salarial do encarregado do empregado Maria Dias

14. Mostre os pares de escaloes que teem intersecção não nula de intervalo [salinf,salsup]
14(b) mude o valor de salinf do escalao 2 para comecar em 99995, de forma a passar a ter intersecção nao nula com o escalão 1
14(c) repita agora o comando e veja se funcionou.

15. Faça a seguinte sequencia:
15(a) comandos que encontrem (a) os empregados do departamento 'Vendas' (b) os empregados com ordenado acima de 200000
15(b) comando que mostre a união dos dois comandos
15(c) comando que encontre a intersecção dos empregados do departamento 'Vendas' com os empregados com ordenado acima de 200000
15(d) comando que encontre a diferença dos empregados do departamento 'Vendas' com os empregados com ordenado acima de 200000

Aula 3

Funções de linha e funções de grupo

As funções são usadas para efectuar cálculos sobre dados, modificar itens individuais de informação, manipular resultados de visualização de datas e converter tipos de dados.

Quanto à quantidade de informação que processam de cada vez, podemos classificar as funções em dois tipos: de registo e de grupo.

No que diz respeito ao tipo de dados que manipulam, existem funções de caracteres, numéricas, de datas (ficam para a próxima aula), de conversão (ficam para a próxima aula) e que aceitam qualquer tipo de argumentos (ficam para a próxima aula).

```
escalao | salinf | salsup | escalao | salinf | salsup
---------+--------+--------+---------+--------+--------
      1 |  55000 |  99999 |       2 |  99995 | 210000
```

3.1. Funções de registo ou de linha

- Actuam sobre cada registo
- Produzem apenas um valor por registo
- Podem receber um ou mais argumentos de entrada
- Podem ser encadeadas
- Podem ser utilizadas onde se utilizam colunas, expressões, cláusulas `SELECT`, `WHERE` e `ORDER BY`.

3.1.1. Funções de manipulação de caracteres

- `ASCII(char)`

  Devolve o número que corresponde a esse caracter no código ASCII. Ver `CHR`. Exemplo:

```
SQL>SELECT ASCII('S')
  2    FROM dual;

ASCII('S')
----------
        83
```

NOTA: A tabela **`DUAL`** é uma tabela especial do sistema que existe apenas para se poderem fazer chamadas de funções. Em SQL não existe a possibilidade de fazer chamadas de funções directamente na linha de comando. A única alternativa é incluir as chamadas de funções num comando SQL. Assim, existe uma tabela especial do sistema, com uma coluna e uma linha, onde ninguém pode inserir mas todos podem seleccionar. Essa tabela, é a **`DUAL`**.

- `CHR(n)`

  Devolve o caracter com esse número. Ver `ASCII`. Exemplo:

```
SQL>SELECT RPAD(CHR(65), 2) "65",
  2        RPAD(CHR(67), 2) "67",
  3        RPAD(CHR(69), 2) "69",
  4        RPAD(CHR(71), 2) "71",
  5        RPAD(CHR(73), 2) "73",
  6        RPAD(CHR(75), 2) "75",
  7        RPAD(CHR(77), 2) "77",
  8        RPAD(CHR(79), 2) "79",
  9        RPAD(CHR(81), 2) "81",
 10        RPAD(CHR(83), 2) "83",
 11        RPAD(CHR(85), 2) "85",
 12        RPAD(CHR(87), 2) "87",
 13        RPAD(CHR(89), 2) "89",
 14        CHR(83) || CHR(73) || CHR(70) || CHR(73) ||
 15        CHR(67) || CHR(65) || CHR(80) "Juntar"
 16    FROM dual;

65 67 69 71 73 75 77 79 81 83 85 87 89 Juntar
-- -- -- -- -- -- -- -- -- -- -- -- -- -------
A  C  E  G  I  K  M  O  Q  S  U  W  Y  SIFICAP
```

- `CONCAT(frase1, frase2)`
  Junta duas strings ou frases. Exemplo:

```
SQL>SELECT CONCAT('Sifi', 'cap') "Juntar"
  2     FROM dual;

Juntar
-------
Sificap
```

  Também se podem concatenar strings com o operador ||. Exemplo:

```
SQL>SELECT 'Sifi' || 'cap' "Juntar"
  2     FROM dual;

Juntar
-------
Sificap
```

- `INITCAP(col|string)`
  Converte a primeira letra para maiúscula e as restantes para minúsculas. Ver `LOWER` e `UPPER`.
- `INSTR(col|valor, 'cadeia')`
  Localiza a posição da primeira ocorrência de *`'cadeia'`* em col ou valor. Ver `SUBSTR`.

  `INSTR(col|valor, 'cadeia', pos, n)`
  Localiza a posição da *`n`*-esima ocorrência de *`'cadeia'`* a partir da posição *`pos`* em *`col`* ou *`valor`*.

  Exemplo:

```
SQL> SELECT nome,
  2     INSTR(nome, 'a')       "'a'",
  3     INSTR(nome, 'nt')      "'nt'",
  4     INSTR(nome, 'a', 6, 1) "'a', 6, 1"
  5     FROM DEP;

NOME                    'a'      'nt' 'a', 6, 1
--------------- --------- --------- ---------
Contabilidade           5         3        11
Investigação            9         0         9
Vendas                  5         0         0
Planeamento             3         9         6
```

- `LENGTH(col|string)`
  Devolve o comprimento da *`string`*.
- `LOWER(col|string)`
  Converte as letras para minúsculas. Ver `UPPER` e `INITCAP`.
- `LPAD(col|string, n, ['cadeia'])`
  Coloca espaços ou repetições de *`'cadeia'`* à esquerda de *`string`* até o comprimento total da concatenação atingir um comprimento de *`n`*. Se *`n`* for menor que o tamanho da *`string`* inicial, esta é cortada. Ver `RPAD`.
- `LTRIM(col|string, ['car'])`
  Remove todos os espaços (ou ocorrências do caracter *`'car'`*, à esquerda da *`string`* de entrada. Ver `RTRIM`, `LPAD` e `RPAD`.
- `REPLACE(col|string, cadeia_inicial, cadeia_final)`

Procura em string sub-strings iguais a `cadeia_inicial` e substitui-as por *`cadeia_final`*. Se *`cadeia_final`* não for especificada, as sub-strings são apenas retiradas.

- `RPAD(col|string, n, ['cadeia'])`
  Devolve uma nova string que representa a string de entrada mais uma string constituída por *`n`* espaços se *`'cadeia'`* for omitida ou então com repetições da string *`'cadeia'`*. Ver `LPAD`, `RTRIM` e `LTRIM`.

- `RTRIM(col|string, ['car'])`
  Retira todas as ocorrências do caracter *`'car'`* que estejam à direita da *`string`*, ou caso *`'car'`* não seja especificado, retirada todos os espaços em branco. Ver `LTRIM`, `RPAD` e `LPAD`.

- `SUBSTR(col|valor, pos, [n])`
  Este funcao devolve sub-strings da funcao de entrada a começar no caracter *`pos`* e de comprimento (facultativo) *`n`*. Se *`n`* não for especificado, a sub-string começa no caracter *`pos`* e vai até ao fim. Ver `INSTR`.

  Exemplo:

```
SQL> SELECT nome,
  2         SUBSTR('ORACLE', 2,4) COL1,
  3         SUBSTR(nome, 2)       COL2,
  4         SUBSTR(nome,3,5)      COL3
  5     FROM DEP;

NOME            COL1 COL2           COL3
--------------- ---- -------------- -----
Contabilidade   RACL ontabilidade   ntabi
Investigação    RACL nvestigação    vesti
Vendas          RACL endas          ndas
Planeamento     RACL laneamento     aneam
```

- `TRANSLATE(col|string, de, para)`
  Transforma as ocorrências de caracteres do conjunto *`de`* nos respectivos caracteres do conjunto *`para`*. Se *`para`* não for especificado, retira apenas as ocorrências de *`de`*.

- `UPPER(col|string)`
  Converte as letras para maiúsculas. Ver `LOWER` e `INITCAP`.

- Existem mais funções de caracteres (`INSTRB`, `LENGTHB`, `NLS_INITCAP`, `NLS_LOWER`, `NLS_UPPER`, `NLSSORT`, `SUBSTRB`) embora sejam menos usadas. Para mais informações ver os manuais do Oracle.

3.1.2. Funções numéricas

- `ABS(col|num)`
  Devolve o valor absoluto da coluna ou expressão.

- `CEIL(col|num)`
  Devolve o menor inteiro que seja maior ou igual ao parâmetro de entrada. Ver `FLOOR`.

- `FLOOR(col|num)`
  Devolve o maior inteiro que seja menor ou igual ao parâmetro de entrada. Ver `CEIL`.

- `MOD(col1|num1, col2|num2)`

Determina o resto da divisão do primeiro parâmetro pelo segundo.

- `POWER(col|num, n)`
  Eleva a coluna ou expressão à potência de *n*. *n* tem que ser inteiro.
- `ROUND(col|num, n)`
  Arredonda o valor para o inteiro mais próximo se *n* não for especificado. Se *n* for positivo arrendonda na casa decimal com esse número. Se *n* for negativo arredonda em casas à esquerda da virgula. Ver `TRUNC`. Exemplo ::

```
SQLWKS> SELECT round(523.456)     "R 0",
     2>        round(523.456, 0)  "R 0",
     3>        round(523.456, 1)  "R 1",
     4>        round(523.456, 2)  "R 2",
     5>        round(523.456, -1) "R-1",
     6>        round(523.456, -2) "R-2",
     7>        round(523.456, -3) "R-3",
     8>        round(523.456, -4) "R-4"
     9>   FROM dual;
R 0     R 0     R 1     R 2     R-1     R-2     R-3     R-4
------- ------- ------- ------- ------- ------- ------- -------
    523     523   523.5  523.46     520     500    1000       0
```

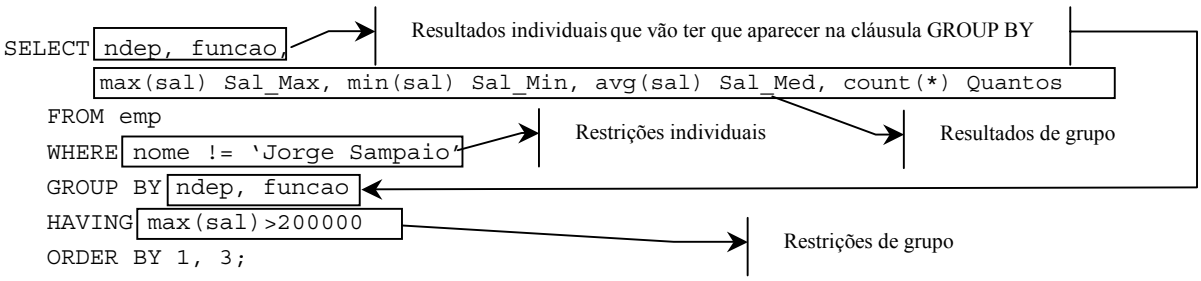
- `SIGN(col|num)`
  Devolve –1 se a coluna ou expressão tiverem um valor negativo, 1 se tiverem um valor positivo ou 0 se tiverem um valor de 0.
- `SQRT(col|num)`
  Devolve a raiz quadrada da coluna ou expressão.
- `TRUNC(col|num, n)`
  Faz o mesmo que o `ROUND` mas em vez de arredondar trunca. Ver `ROUND`.
- Mais funções disponíveis nos manuais do Oracle (`ACOS`, `ASIN`, `ATAN`, `ATAN2`, `COS`, `COSH`, `EXP`, `LN`, `LOG`, `SIN`, `SINH`, `TAN`, `TANH`).

3.2. Funções de grupo

- Actuam sobre conjuntos de registos
- Produzem um valor por cada conjunto de registos
- Por omissão, todos os registos de uma tabela são considerados um único grupo
- A cláusula `GROUP BY` da instrução `SELECT` permite subdividir uma tabela em grupos mais pequenos
- A cláusula `HAVING` permite efectuar condições de restrição sobre os resultados de conjuntos de registos da mesma maneira que a cláusula `WHERE` efectua condições de restrição sobre registos individuais.

NOTA: Se se incluir funções de grupo num comando `SELECT` não é possível incluir também resultados individuais a não ser que os resultados individuais apareçam na cláusula `GROUP BY`.

- Exemplo:



Para se compreender melhor as claúsulas de `GROUP BY` e `HAVING` convém perceber a sequência de passos que o servidor usa até obter o resultado final:

- As linhas e as colunas são pré-seleccionadas com informação das cláusulas `SELECT` e `WHERE`.
- Esse conjunto de linhas é dividido em grupos em função da informação existente na cláusula `GROUP BY`. Para cada um dos grupos são calculadas as respectivas funções de grupo que aparecem na claúsula `SELECT` como por exemplo os valores de `max`, `min`, `avg`, etc.
- Só são mostrados os grupos que passarem nas condições de `HAVING`.

A confusão mais comum com as funções de grupo é saber em que cláusula, `WHERE` ou `HAVING`, devem aparecer as restrições. E a resposta é fácil: se se estiver a restringir registos individuais deve usar-se `WHERE`, se se estiver a restringir grupos de registos deve usar-se `HAVING`. No entanto, há casos especiais que se podem resolver tanto com restrições em `WHERE` como em `HAVING`. Por exemplo, se se quiser saber qual é o salário máximo por função para todas as funções da empresa à excepção da função `'Presidente'` há duas soluções.

- Exemplo de 2 comandos que produzem o mesmo resultado mas apresentam as restrições na cláusula `WHERE` ou na `HAVING` Repare que neste exemplo, a coluna `funcao` é simultaneamente uma característica do registo individual e do grupo, e é isso que permite as duas soluções.

```
SQLWKS> SELECT funcao, max(sal)
     2>   FROM emp
     3>   WHERE funcao != 'Presidente'
     4>   GROUP BY funcao
     5>   ORDER BY max(sal) ASC;
FUNCAO       MAX(SAL)
------------ ----------
Continuo          70800
Vendedor         221250
Encarregado      450975
Analista         565000


SQLWKS> SELECT funcao, max(sal)
     2>   FROM emp
     3>   GROUP BY funcao
     4>   HAVING funcao != 'Presidente'
     5>   ORDER BY max(sal) ASC;
FUNCAO       MAX(SAL)
------------ ----------
Continuo          70800
Vendedor         221250
Encarregado      450975
Analista         565000
```

- Exemplo de um erro comum usando funções de grupo. Se se pedissem as profissões e os seus salários máximos mas apenas para aquelas cujo máximo fosse superior a 300000, um erro comum seria:

```
SQLWKS> SELECT funcao, max(sal)
     2>   FROM emp
     3>   WHERE max(sal) > 300000
     4>   GROUP BY funcao
     5>   ORDER BY max(sal) ASC;

  WHERE max(sal) > 300000
        *
ORA-00934: group function is not allowed here
```

A solução certa é colocar a restrição no `HAVING`:

```
SQLWKS> SELECT funcao, max(sal)
     2>   FROM emp
     3>   GROUP BY funcao
     4>   HAVING max(sal) > 300000
     5>   ORDER BY max(sal) ASC;

FUNCAO       MAX(SAL)
------------ ----------
Encarregado      450975
Analista         565000
Presidente       890000
```

3.2.1. Funções de grupo:

Todas as funções de grupo, com a excepção da `COUNT(*)`, ignoram os valores nulos. Para cada uma das funções de grupo existe a possibilidade de se contarem apenas os valores distintos usando a palavra chave `DISTINCT` ou contarem todos usando a palavra chave `ALL`. A opção `ALL` é a usada por omissão.

- `AVG([DISTINCT|ALL] col|num)`
  Devolve a media de valores que essa expressão numérica representa. Note que apesar de os valores da coluna serem apenas inteiros, o valor da média pode ser um número fraccionário.

- `COUNT([DISTINCT|ALL] valor | *)`
  Devolve o numero de registos na tabela que correspondem a uma expressão não nula. O caso especial `COUNT(*)` devolve o número de registos desse grupo mesmo que existam valores nulos.

  Exemplo:

```
SQL> SELECT COUNT(*)               "Todos",
  2         COUNT(sal)             "#Sal não null",
  3         COUNT(premios)         "#Premios não null",
  4         COUNT(sal*14+premios) "#Expressão não null"
  5     FROM emp;

    Todos #Sal não null #Premios não null #Expressão não null
--------- ------------- ----------------- -------------------
       14            14                 4                   4
```

  Exemplo :

```
SQL> SELECT count(ALL funcao)      "Todos",
  2>        count(DISTINCT funcao) "Profissões",
  3>        count(funcao)          "Igual a ALL"
  4>   FROM emp;

Todos       Profissões  Igual a ALL
----------- ----------- -----------
         14           5          14
1 row selected.
```

- `MAX([DISTINCT|ALL] valor)`
  Devolve o máximo dos valores que essa expressão representa.

- `MIN([DISTINCT|ALL] valor)`
  Devolve o mínimo dos valores que essa expressão representa.

- `SUM([DISTINCT|ALL] valor)`
  Devolve a soma dos valores que essa expressão representa.

- Existem mais funções de grupo mas são menos usadas (`GLB`, `LUB`, `STDDEV`, `VARIANCE`). Para mais informações ver os manuais do Oracle.

3.3. Exercícios

Considere a base de dados de demonstração fornecida.

1. Mostre o nome, função e departamento dos vendedores e apenas deles. O nome deverá aparecer todo em maiúsculas, a função em minúsculas e o departamento com a primeira letra em maiúscula e o resto em minúsculas. Ordene o resultado por nome do funcionário.

```
NOME                 FUNCAO       DEPARTAMENTO
-------------------- ------------ ---------------
ANA RODRIGUES        vendedor     Vendas
JOANA MENDES         vendedor     Vendas
MANUEL MADEIRA       vendedor     Vendas
NELSON NEVES         vendedor     Vendas
```

Resposta:

2. Escreva um comando que devolva o número de empregados. O resultado deve ser semelhante ao que se segue.

```
Total de empregados
-------------------
                 14
```

Resposta:

3. Escreva um comando que determine quantos empregados **não** ganham prémios. O resultado deve ser semelhante ao que se segue.

```
Empregados sem premio
---------------------
                   10
```

Resposta:

4. Escreva um comando que conte o número de empregados existentes, calcule o salário médio mensal e o total de remuneração anual auferido pelo conjunto de todos os empregados.

```
Total de empregados Salário médio mensal Remuneração total anual
------------------- -------------------- ----------------------
                 14            285666.07                56226750
```

Resposta

5. Mostre a lista dos Contínuos e o respectivo salário com um aumento de 13,55% para estes empregados. O salário depois do aumento deverá ser arredondado na primeira casa decimal.

```
NOME                 FUNCAO             SAL SALARIO_COM_AUMENTO
-------------------- ----------- --------- --------------------
Tome Ribeiro         Continuo         56950              64666.7
Rita Pereira         Continuo         65100              73921.1
Olga Costa           Continuo         68300              77554.7
Antonio Silva        Continuo         70800              80393.4
```

Resposta:

6. O mesmo que na pergunta anterior, mas agora com o salário depois do aumento arredondado para um numero inteiro.

```
NOME                 FUNCAO             SAL SALARIO_COM_AUMENTO
-------------------- ----------- --------- --------------------
Tome Ribeiro         Continuo         56950                64667
Rita Pereira         Continuo         65100                73921
Olga Costa           Continuo         68300                77555
Antonio Silva        Continuo         70800                80393
```

Resposta:

7. Escreva um comando que produza o seguinte resultado. Note que antes de cada nome de empregado existem 4 sinais de '>' (maior que) e um espaço, e que depois de cada nome existe um espaço e caracteres '<' (menor que) suficientes até se atingir um comprimento total de 25 caracteres.

```
Mariquices com strings
-------------------------
>>>> Ana Rodrigues <<<<<<
>>>> Antonio Silva <<<<<<
>>>> Augusto Reis <<<<<<<
>>>> Catarina Silva <<<<<
>>>> Duarte Guedes <<<<<<
>>>> Joana Mendes <<<<<<<
>>>> Jorge Sampaio <<<<<<
>>>> Manuel Madeira <<<<<
>>>> Maria Dias <<<<<<<<<
>>>> Nelson Neves <<<<<<<
>>>> Olga Costa <<<<<<<<<
>>>> Rita Pereira <<<<<<<
>>>> Silvia Teles <<<<<<<
>>>> Tome Ribeiro <<<<<<<
```

Resposta

8. Encontre o salário mais baixo, mais alto e o salário médio de todos os funcionários:

```
SALÁRIO_MAIS_BAIXO SALÁRIO_MAIS_ALTO SALÁRIO_MÉDIO
------------------ ----------------- -------------
             56950            890000     285666.07
```

Resposta

9. Encontre a diferença entre o salário mais alto e o mais baixo para cada departamento.

```
Departamento Diferença
------------ ---------
          10    821700
          20    499900
          30    323900
```

Resposta

10. Mostre quantos empregados existem para cada função. Ordene o resultado pela função.

```
Funcao       Quantidade
------------ ----------
Analista              2
Continuo              4
Encarregado           3
Presidente            1
Vendedor              4
```

Resposta

11. Repita o comando anterior, mas apenas para as funções que terminam com a letra 'o' ou para os 'Analistas'.

```
Funcao       Quantidade
------------ ----------
Analista              2
Continuo              4
Encarregado           3
```

Resposta

12. Mostre o salário mais baixo dos empregados que trabalham para cada Encarregado. Exclua os grupos em que o salário mínimo seja inferior a 200000 e ordene o resultado por salário.

```
Encarregado Salário Minimo
----------- --------------
       1839         279450
       1566         435000
                    890000
```

Resposta

13. Mostre o salário médio para cada tipo de função, ordenando o resultados por ordem crescente dos salários médios. O salário médio deverá ser arredondado para o menor inteiro possível que seja superior ou igual ao seu valor. Apenas deverão ser mostradas funções onde exista mais de 1 trabalhador.

```
Função       Salario Medio
------------ -------------
Continuo              65288
Vendedor             184225
Encarregado          370425
Analista             500000
```

Resposta

14. Indique o salário máximo, mínimo e médio e a quantidade de empregados para cada função e de cada departamento (grupos e subgrupos). Exclua registos individuais de nome `'Jorge Sampaio'` e resultados colectivos que apresentem um salário máximo inferior ou igual a 200 contos. Ordene por departamento e depois por salário máximo.

```
NDEP       FUNCAO       SAL_MAX    SAL_MIN    SAL_MED    QUANTOS
---------- ------------ ---------- ---------- ---------- ----------
        10 Encarregado      279450     279450     279450          1
        20 Encarregado      450975     450975     450975          1
        20 Analista         565000     435000     500000          2
        30 Vendedor         221250     145600     184225          4
        30 Encarregado      380850     380850     380850          1
```

Resposta

Aula 4

Funções de linha (parte II) e subconsultas

As funções são usadas para efectuar cálculos sobre dados, modificar itens individuais de informação, manipular resultados de visualização de datas e converter tipos de dados.

Quanto à quantidade de informação que processam de cada vez, podemos classificar as funções em dois tipos: de registo e de grupo (ver última aula).

No que diz respeito ao tipo de dados que manipulam, existem funções de caracteres (ver última aula), numéricas (ver última aula), de datas, de conversão e funções que aceitam qualquer tipo de argumentos.

4.1. Funções de registo ou de linha

4.1.1. Recapitulação

Características das funções de registo:

- Actuam sobre cada registo.
- Produzem apenas um valor por registo.
- Podem receber um ou mais argumentos de entrada.
- Podem ser encadeados.
- Podem ser utilizadas onde se utilizam colunas, expressões, cláusulas `SELECT`, `WHERE` e `ORDER BY`.

4.1.2. Funções de datas:

Em Oracle as datas guardam valores sobre o século, ano, mês, dia, hora, minuto e segundo. Podem representar valores entre 1 de Janeiro de 4712 AC e 31 de Dezembro de 4712 DC. A data é guardada internamente através de um formato de dados desconhecido para o utilizador/programador. No entanto, é possível alterar o formato usado pelo Oracle para mostrar datas. Pode, por exemplo, ver-se datas com 2 dígitos no ano ou com 4 dígitos. Mas seja qual for a escolha, internamente é guardada toda a informação do ano.

- Operações aritméticas

  As seguintes operações aritméticas são válidas para datas:

```
data  + dias     = nova_data
data  - dias     = nova_data
data1 - data2    = dias_diferenca (numero real positivo)
data  + horas/24 = data_mais_horas
data  + dias + horas/24 + minutos/1440 + segundos/86400 = nova_data;
```

  Exemplo a somar dias, horas, minutos e segundos a uma data:

```
SQL>SELECT TO_CHAR(TO_DATE('98-07-01 00:00:00', 'YY-MM-DD hh24:mi:ss')
  2                          + 8         -- mais  8 dias
  3                          + 12/24     -- mais 12 horas
  4                          + 34/1440   -- mais 34 minutos
  5                          + 56/86400, -- mais 56 segundos
  6                'yyyy-mm-dd hh24:mi:ss') "Somar"
  7    FROM DUAL;

Somar
-------------------------------------------------------------------
1998-07-09 12:34:56
```

  Exemplo para subtrair datas:

```
SQL>SELECT TO_DATE('98-07-01 00:00:00', 'YY-MM-DD hh24:mi:ss')
  2      - TO_DATE('98-07-01 00:00:00', 'YY-MM-DD hh24:mi:ss')
  3                          + 8         -- soma 8 dias
  4                          + 12/24     -- mais 12 horas
  5                          + 34/1440   -- mais 34 minutos
  6                          + 56/86400  -- mais 56 segundos
  7           "Subtrai datas"
  8    FROM DUAL;

Subtrai datas
-------------
    8.5242593
```

- `ADD_MONTHS(data, n)`
  Devolve o valor da soma de *`n`* meses com *`data`*. O valor de *`n`* tem que ser inteiro mas pode ser negativo.

- `LAST_DAY(data)`
  Devolve a data do último dia do mês que contém o dia indicado por *`data`*.

- `MONTHS_BETWEEN(data1, data2)`
  Devolve um número fraccionário, maior, menor ou igual a zero que representa a diferença em meses entre a data1 e data2. Para efeitos da parte fraccionária é considerado que um mês tem 31 dias. Assim, `months_between('27-02-1998', '31-01-1998')` devolve um valor de 0,870967742 que é igual a 27/31 (e não a 27/28).

- `NEW_TIME(data, fuso1, fuso2)`
  Permite calcular a diferença horária entre dois fusos horários. Consultar os manuais do Oracle para mais informações.

- `NEXT_DAY(data, dia_semana)`
  Devolve o valor de data do próximo dia da semana especificado por `dia_semana` a seguir a data. Valores válidos para dia_semana são: `'sun'`, `'mon'`, `'tue'`, `'wed'`, `'thu'`, `'fri'`, `'sat'`, que representam as abreviaturas em inglês para os dias da semana. Também se podem usar os nomes completos em inglês e não interessa se se escreve em minúsculas ou maiúsculas. Poder-se-á usar nomes de dias da semana numa língua diferente do inglês se se alterar a língua da sessão em uso. Também se podem usar os números de 1 (Domingo) a 7 (Sábado).

  Exemplo:

```
SQL> SELECT NEXT_DAY(sysdate, 'Sun') "Domingo",
  2         NEXT_DAY(sysdate, 'Mon') "Segunda",
  3         NEXT_DAY(sysdate, 'Tue') "Terça",
  4         NEXT_DAY(sysdate, 'Wed') "Quarta",
  5         NEXT_DAY(sysdate, 'Thu') "Quinta",
  6         NEXT_DAY(sysdate, 'Fri') "Sexta",
  7         NEXT_DAY(sysdate, 'Sat') "Sábado"
  8     FROM dual;

Domingo  Segunda  Terça    Quarta   Quinta   Sexta    Sábado
-------- -------- -------- -------- -------- -------- --------
98-11-01 98-11-02 98-11-03 98-10-28 98-10-29 98-10-30 98-10-31
```

  Apesar de não aparecer no exemplo, os valores de hora, minuto e segundo são iguais aos valores do primeiro argumento (no exemplo – `sysdate`, que devolve a data do sistema). Ver função `sysdate` mais abaixo.

- NLS_DATE_FORMAT

  O Oracle representa as datas segundo o formato definido em NLS_DATE_FORMAT. Pode alterar-se o seu valor através do comando ALTER SESSION.

  Alguns dos campos válidos do NLS_DATE_FORMAT são:

DDD	Dia do ano (número)
DD	Dia do mês (número)
DAY	Dia da semana (extenso)
MM	Número do mês
MON	Nome abreviado do mês
MONTH	Nome por extenso do mês
YYYY YYY YY Y	Representar o ano com 4, 3, 2 ou 1 dígitos respectivamente
HH HH12	Hora do dia (0-12)
HH24	Hora do dia (0-24)
MI	Minutos
SS	Segundos
SSSSS	Segundos depois da meia-noite
- / , . ; : "texto"	pontuação ou texto entre aspas

  Exemplo para alterar o formato da data:

```
SQL>ALTER SESSION
  2    SET NLS_DATE_FORMAT = '"Data:" YYYY-MON-DD HH24:MI:SS';

Session altered.

SQL>SELECT SYSDATE FROM dual;

SYSDATE
--------------------------
Data: 1998-OCT-27 20:23:35
```

NOTA: O comando ALTER SESSION permite modificar muitas outras opções NLS (*National Language Support* - ou numa tradução livre *Suporte à Língua Nacional* - tais como a língua, o território, a moeda, etc.) e ainda outro tipo de opções relacionadas com a ligação à base de dados. Consulte os manuais do Oracle para saber mais sobre o comando ALTER SESSION. Para poder ser usado, o utilizador precisa ter atribuído o privilégio de sistema com o mesmo nome.

- `ROUND(data, ['DAY'|'MONTH'|'YEAR'|'outro'])`
  Arredonda a data ao dia ou ao dia da semana ou ao mês ou ao ano. Para arredondar ao dia não se usa o segundo argumento. Todos os valores de tempo antes do meio-dia são arredondados para as zero horas desse próprio dia. Os restantes valores são arredondados para o dia seguinte. Para arredondar ao dia da semana (para a segunda-feira mais perto) usa-se o segundo argumento com o valor de `'DAY'`. O valor de `'MONTH'` serve para arredondar ao mês e o de `'YEAR'` para arredondar ao ano. Existem mais opções de arredondamento. Consultar os manuais da Oracle.

  NOTA: Considera-se que o meio da semana é quarta-feira ao meio-dia. Tudo o que estiver compreendido entre uma segunda-feira e o meio-dia da quarta-feira da mesma semana é arredondado para as zero horas dessa segunda-feira. Tudo o que estiver depois do meio-dia da quarta-feira da mesma semana é arredondado para as zero horas da segunda-feira da semana seguinte. Em termos de meses, considera-se que a metade é a meia-noite entre os dias 15 e 16 independentemente do tamanho dos meses. Em termos de anos, considera-se metade a meia-noite entre o dia 30 de Junho e o dia 01 de Julho muito embora a segunda "metade" seja constituída por 184 dias e a primeira por 181 dias (ou 182 em anos bissextos). Em qualquer dos casos, o valor de horas, minutos e segundos fica igual a zero.

- `SYSDATE`
  Função sem argumentos que devolve a data e hora do servidor.

- `TRUNC(data, ['DAY'|'MONTH'|'YEAR'|'outro'])`
  Arredonda a data para as zeros horas do dia actual (se não se usar o segundo argumento) ou para o princípio da semana actual (se o segundo argumento for `'DAY'`), ou para o princípio do mês actual (se o segundo argumento for `'MONTH'`) ou para o princípio do ano actual (se o segundo argumento for `'YEAR'`). Em qualquer dos casos, o valor de horas, minutos e segundos fica igual a zero. Consultar os manuais do Oracle para as restantes opções.

4.1.3. Funções de conversão:

- `TO_CHAR(d [,fmt[,'nlsparams']])` – Para converter datas
  Permite transformar um valor do tipo data num valor do tipo string. A string *`fmt`* define o tipo de transformação e é construída através dos campos vistos em `NLS_DATE_FORMAT` (ver página *57*).

  Exemplo:

```
SQL>SELECT RPAD(TO_CHAR(sysdate, 'yyyy-mm-dd hh24:mi:ss'), 19)    "Col1",
  2        RPAD(TO_CHAR(sysdate, 'fmddth, "of" Month, yyyy'), 22) "Col2"
  3    FROM dual;

Col1                Col2
------------------- ----------------------
1998-10-28 00:52:27 28th, of October, 1998
```

  Ver significado dos *`'nlsparams'`* na documentação do Oracle.

- `TO_CHAR(n [,fmt[,'nlsparams']])` – Para converter números

  A string *`fmt`* define como é feita a transformação de números para caracteres. O significado dos *`'nlsparams'`* deve ser consultado na documentação do Oracle.

  A string de formatação *`fmt`* pode ter os seguintes elementos e respectivos significados:

Símbolo	Significado
`9`	Um símbolo 9 para cada algarismo significativo a representar antes ou depois da vírgula (note-se que nas strings de formatação a vírgula é representada por um ponto por herança da numeração anglo-saxónica). Se existirem mais 9s que algarismos a representar só são representados os dígitos que existem.
`0`	Faz o mesmo que o 9 mas se o número não tiver tantos algarismos como definido em *`fmt`* é incluído o algarismo 0 por cada um que não existe.
`B`	Se o número for 0 apresenta apenas um espaço.
`S`	Coloca-se antes ou depois da sequência de 0s e/ou 9s e indique onde deve aparecer o sinal do número. Se S não for indicado aparece o sinal apenas para os números negativos e nos positivos aparece um espaço.
`L`	Quando usado, o valor aparece como uma representação monetária. É incluído o símbolo dólar ($) ou outro dependendo dos parâmetros NLS. Pode ser colocado antes ou depois da sequência de 0s e/ou 9s.
`fm`	Se usado, os espaços a mais são removidos.
`E`	É usado para indicar quantos dígitos se devem apresentar no expoente usando a notação científica.

Exemplos:

Número	'fmt'	Resultado
-1234567890	9999999999S	'1234567890-'
0	99.99	' 0.00'
+0.1	99.99	' .10'
-0.2	99.99	' -.20'
0	90.99	' 0.00'
+0.1	90.99	' .10'
-0.2	90.99	' -0.20'
0	9999	' 0'
1	9999	' 1'
0	B9999	' '
1	B9999	' 1'
0	B90.99	' '
+123.456	999.999	' 123.456'
-123.456	999.999	'-123.456'
+123.456	FM999.009	'123.456'
+123.456	9.9EEEE	' 1.2E+02'
+1E+123	9.9EEEE	' 1.0E+123'
+123.456	FM9.9EEEE	'1.23E+02'
+123.45	FM999.009	'123.45'
+123.0	FM999.009	'123.00'
+123.45	L999.99	' $123.45'
+123.45	FML99.99	'$123.45'
+1234567890	9999999999S	'1234567890+'

- `TO_DATE(char [,fmt[,'nlsparams']])`
  Transforma uma string numa data exactamente da mesma maneira que `TO_CHAR` (para converter datas) transforma uma data numa string. Ver também `NLS_DATE_FORMAT` e `ALTER SESSION` (pag 57) e a documentação do Oracle.

  Usa-se frequentemente esta função para inserir valores de datas.

  Exemplo:

```
SQLWKS> SELECT to_date('2000/09/24', 'yyyy/mm/dd') "Data"
     2>   FROM dual;
Data
------------------------------
September 24, 2000 - Sunday
```

- `TO_NUMBER(char [,fmt[,'nlsparams']])`
  Transforma uma string numa data exactamente da mesma maneira que `TO_CHAR` (para converter números) transforma um número numa string. Ver também a documentação do Oracle.

- Existem mais funções de conversão (`CHARTOROWID`, `CONVERT`, `HEXTORAW`, `RAWTOHEX`, `ROWIDTOCHAR`, `TO_CHAR` (conversão de *labels*), `TO_LABEL`, `TO_MULTI_BYTE`, `TO_SINGLE_BYTE`). Consulte os manuais do Oracle para mais informações.

4.1.4. Outras funções:

- `DECODE(col|expr, compara1, resultado1 [, compara2, resultado2, ...], valor_omissao)`

  É provavelmente uma das funções mais poderosas em SQL porque permite fazer testes semelhantes aos IFs de outras linguagens. O primeiro parâmetro é comparado com `compara1`. Se for igual é devolvido o valor de `resultado1`. Caso contrário é comparado com `compara2` e assim sucessivamente. Se não for igual a nenhum dos `compara`s então é devolvido o valor de `valor_omissao`.

  Exemplo:

```
SQL>SELECT funcao, DECODE(funcao, 'Presidente', 'Chefão',
  2                               'Continuo',   'Escravo',
  3                               'INDEFINIDO') DECODE
  4     FROM emp;

FUNCAO       DECODE
------------ ----------
Presidente   Chefão
Encarregado  INDEFINIDO
Encarregado  INDEFINIDO
Encarregado  INDEFINIDO
Analista     INDEFINIDO
Analista     INDEFINIDO
Vendedor     INDEFINIDO
Vendedor     INDEFINIDO
Vendedor     INDEFINIDO
Vendedor     INDEFINIDO
Continuo     Escravo
Continuo     Escravo
Continuo     Escravo
Continuo     Escravo
14 rows selected.
```

- `GREATEST(expr [,expr] ...)`

  Devolve o maior dos parâmetros de entrada. Se, para a comparação for necessário, converte todos os parâmetros para o tipo de dados usado no primeiro parâmetro. Exemplo:

```
SQLWKS> SELECT greatest(12, 5, 80, 25) "GREATEST"
     2>   FROM dual;
GREATEST
----------
        80
```

- `LEAST(expr [,expr] ...)`

  Devolve o menor dos parâmetros de entrada. Se, para a comparação for necessário, converte todos os parâmetros para o tipo de dados usado no primeiro parâmetro. Exemplo:

```
SQLWKS> SELECT least(12, 5, 80, 25) "LEAST"
     2>   FROM dual;
LEAST
----------
         5
```

- `NVL(expr1, expr2)`

  Devolve *`expr2`* se *`expr1`* tiver valor nulo. Caso contrário devolve *`expr1`*. Exemplo:

```
SQLWKS> SELECT nome,
     2>        NVL(premios, 0)  "PREMIOS"
     3>   FROM emp
     4>   ORDER BY NVL(premios, 0);
NOME                 PREMIOS
-------------------- ----------
Jorge Sampaio                 0
Augusto Reis                  0
Duarte Guedes                 0
Silvia Teles                  0
Maria Dias                    0
Manuel Madeira                0
Rita Pereira                  0
Antonio Silva                 0
Olga Costa                    0
Tome Ribeiro                  0
Catarina Silva                0
Joana Mendes              56300
Ana Rodrigues             81400
Nelson Neves              98500
```

- `UID`
  Devolve o número do utilizador.
- `USER`
  Devolve o nome do utilizador.
- `USERENV(option)`
  Permite verificar os valores das variáveis da sessão como `'LANG'`, `'LANGUAGE'`, `'TERMINAL'`, `'SESSIONID'`, `'CLIENT_INFO'` e outras. Consulte os manuais do Oracle para mais informações.

  Exemplo:

```
SQL>SELECT USERENV('LANGUAGE')
  2    FROM dual;

USERENV('LANGUAGE')
----------------------------------------------------
AMERICAN_AMERICA.WE8ISO8859P1
```

- `VSIZE(expr)`

  Devolve o tamanho em bytes de uma expressão.

  Exemplo:

```
SQL>SELECT nome, VSIZE(nome)
  2     FROM emp;

NOME                 VSIZE(NOME)
-------------------- -----------
Jorge Sampaio                 13
Augusto Reis                  12
Duarte Guedes                 13
Silvia Teles                  12
Maria Dias                    10
Catarina Silva                14
Joana Mendes                  12
Nelson Neves                  12
Ana Rodrigues                 13
Manuel Madeira                14
Tome Ribeiro                  12
Rita Pereira                  12
Olga Costa                    10
Antonio Silva                 13
```

- Existem mais funções que aceitam qualquer tipo de parâmetros (`DUMP`, `GREATEST_LB`, `LEAST_UB`). Consulte os manuais do Oracle para mais informações.

4.2. Subconsultas

Uma das pesquisas mais comuns é encontrar, por exemplo, os dados do empregado que possuí o maior salário. Para obter essa informação é preciso fazer duas coisas: primeiro, determinar qual é o maior salário da empresa e depois seleccionar o empregado com o salário igual a esse valor. Este problema e toda a sua classe de problemas (encontrar a informação em dois ou mais passos) resolve-se com uma técnica chamada subconsulta.

Uma subconsulta simples é uma consulta dentro de outra consulta. E os resultados da consulta interna vão influenciar os da externa. Mais à frente, vamos ver subconsultas correlaccionadas, onde é a consulta externa que influencia o consulta interna.

Exemplo:

```
SELECT  col1, col2, ..
      FROM lista_tabelas
      WHERE  colx = (SELECT  col_a1
                        FROM lista_tabelas2
                        WHERE ...);
```

Numa subconsulta simples o `SELECT` interno é executado primeiro e o(s) seu(s) valor(es) é/são passado(s) para o `SELECT` externo para efeitos de teste. O `SELECT` interno tem que ser colocado entre parêntesis. Não existem limites teóricos para o nível de encadeamento de `SELECT`s.

Se o `SELECT` interno devolver apenas um valor o comparador da cláusula `WHERE` do `SELECT` externo pode ser qualquer um. Se pelo contrário, devolver um conjunto de valores, então o teste de comparação na cláusula `WHERE` externa deve usar o operador `IN`.

Exemplo:

```
SELECT  col1, col2, ..
      FROM lista_tabelas
      WHERE  colx IN (SELECT  col_1x
                         FROM lista_tabelas2
                         WHERE ...);
```

Até agora foram vistos `SELECT`s internos que apenas devolvem uma coluna de dados (embora possam devolver mais de uma linha). No entanto pode desejar-se que o `SELECT` interno devolva várias linhas e várias colunas. Para tal deve colocar-se os nomes das colunas do `WHERE` externo entre parêntesis.

Exemplo:

```
SQL>SELECT  nome, sal, ndep
  2    FROM emp
  3    WHERE  (ndep, sal) IN (SELECT ndep, max(sal)
  4                               FROM emp
  5                               GROUP BY ndep)
  6    ORDER BY ndep, sal;

NOME                        SAL      NDEP
-------------------- --------- ---------
Jorge Sampaio           890000        10
Maria Dias              565000        20
Duarte Guedes           380850        30
```

Note que o `SELECT` interno pode representar por si só uma consulta bastante complicada, fazendo uso das cláusulas `WHERE`, `GROUP BY` e `HAVING`.

4.2.1. Regras para subconsultas simples

- Pode usar-se subconsultas nas cláusulas `WHERE` e `HAVING`.
- As subconsultas devem aparecer sempre entre parêntesis.
- As subconsultas são executadas primeiro e os seus valores passados para a consulta imediatamente a exterior (da mais interior para a mais exterior).
- Não existem limites para o número de `SELECT`s encadeados numa instrução.
- Os `SELECT`s encadeados podem conter todas as cláusulas à excepção da `ORDER BY`.
- A aparecer um `ORDER BY` este terá que ser referido apenas como cláusula do `SELECT` externo.
- Se uma subconsulta devolver apenas uma linha e uma coluna podem ser usados todos os operadores lógicos (`=`, `>`, `<`, `>=`, `<=` e `!=`) e operadores SQL (`BETWEEN...AND...`, `NOT`, etc.).
- Se uma subconsulta devolver mais do que uma linha ou coluna ter-se-á que usar o comparador `IN`.
- Se uma subconsulta devolver mais do que uma coluna (*n* colunas) então a comparação terá que ser feita entre *n* expressões e a subconsulta. As *n*-expressões devem aparecer entre parêntesis, separadas por vírgulas e devem corresponder em tipo de dados às colunas devolvidas pela subconsulta.

4.2.2. Operador ANY

O operador `ANY` compara um valor com todos os valores de uma lista. Devolve verdade se a comparação resultar em **verdade para pelo menos um** desses valores da lista. Usa-se antes de uma lista e depois de um termo de comparação.

Exemplo:

```
SQL>SELECT nome, sal, ndep
  2    FROM emp
  3    WHERE sal > ANY (SELECT sal
  4                        FROM emp
  5                        WHERE ndep = 10
  6                           AND funcao != 'Presidente')
  7       AND funcao != 'Presidente'
  8    ORDER BY sal;

NOME                        SAL      NDEP
-------------------- --------- ---------
Antonio Silva            70800        20
Joana Mendes            145600        30
Manuel Madeira          157800        30
Nelson Neves            212250        30
Ana Rodrigues           221250        30
Silvia Teles            279450        10
Duarte Guedes           380850        30
Catarina Silva          435000        20
Augusto Reis            450975        20
Maria Dias              565000        20
```

O comando anterior devolve a lista de empregados (sem contar com o `'Presidente'`) com salário superior ao **menor** dos salários dos empregados do departamento 10 (sem contar com o `'Presidente'`).

4.2.3. Operador ALL

O operador `ALL` compara um valor com todos os valores de uma lista. Devolve verdade se a comparação resultar em **verdade para todos os valores** da lista. Usa-se antes de uma lista e depois de um termo de comparação.

Exemplo:

```
SQL>SELECT nome, sal, ndep
  2    FROM emp
  3    WHERE sal > ALL (SELECT sal
  4                        FROM emp
  5                        WHERE ndep = 10
  6                           AND funcao != 'Presidente')
  7       AND funcao != 'Presidente'
  8    ORDER BY sal;

NOME                        SAL      NDEP
-------------------- --------- ---------
Duarte Guedes           380850        30
Catarina Silva          435000        20
Augusto Reis            450975        20
Maria Dias              565000        20
```

O comando anterior devolve a lista de empregados (sem contar com o `'Presidente'`) com salário superior ao **maior** dos salários dos empregados do departamento 10 (sem contar com o `'Presidente'`).

4.3. Exercícios

Considere a base de dados de demonstração fornecida.

1. Encontre o salário mais baixo de todos.

```
SALARIO_MINIMO
--------------
         56950
```

Resposta:

2. Encontre o empregado que ganha o salário mais baixo de todos (utilize subconsulta).

```
NOME                 FUNCAO              SAL
-------------------- ------------ ---------
Tome Ribeiro         Continuo         56950
```

Resposta:

3. Encontre todos os empregados que têm a mesma função da `'Olga Costa'` e mostre também os seus ordenados.

```
NOME                 FUNCAO              SAL
-------------------- ------------ ---------
Tome Ribeiro         Continuo         56950
Rita Pereira         Continuo         65100
Olga Costa           Continuo         68300
Antonio Silva        Continuo         70800
```

Resposta:

4. Encontre os empregados que ganham o maior salário em cada departamento (`IN`)

```
NOME                        SAL      NDEP
-------------------- --------- ---------
Jorge Sampaio           890000        10
Maria Dias              565000        20
Duarte Guedes           380850        30
```

Resposta:

5. Encontre os empregados que ganham mais do que o salário mais baixo do departamento 30. Não use a função `min`. Ordene os salários mostrados por ordem decrescente. (`ANY`)

```
NOME                        SAL FUNCAO            NDEP
-------------------- --------- ------------ ---------
Jorge Sampaio           890000 Presidente           10
Maria Dias              565000 Analista             20
Augusto Reis            450975 Encarregado          20
Catarina Silva          435000 Analista             20
Duarte Guedes           380850 Encarregado          30
Silvia Teles            279450 Encarregado          10
Ana Rodrigues           221250 Vendedor             30
Nelson Neves            212250 Vendedor             30
Manuel Madeira          157800 Vendedor             30
Joana Mendes            145600 Vendedor             30
Antonio Silva            70800 Continuo             20
Olga Costa               68300 Continuo             10
Rita Pereira             65100 Continuo             20
13 rows selected.
```

Resposta:

6. Encontre os empregados que ganham mais do que qualquer empregado do departamento cujo nome é `'Vendas'`. Não use a função `max`. Ordene os salários mostrados por ordem decrescente. (`ALL`)

```
NOME                 SAL        FUNCAO       NDEP
-------------------- ---------- ------------ ----------
Jorge Sampaio            890000 Presidente           10
Maria Dias               565000 Analista             20
Augusto Reis             450975 Encarregado          20
Catarina Silva           435000 Analista             20
4 rows selected.
```

Resposta:

7. Mostre os departamentos que têm um salário médio superior ao do departamento 30. (`HAVING`)

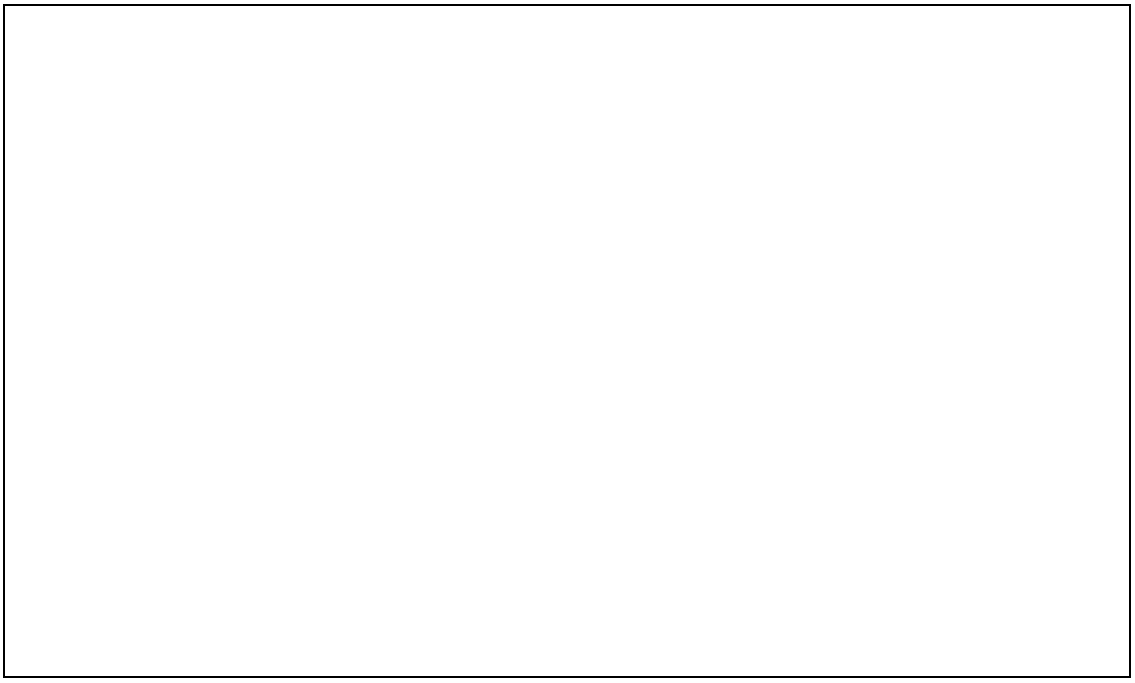
```
NOME           SAL_MED
-------------- ----------
Contabilidade  412583.333
Investigação       317375
2 rows selected.
```

Resposta:

8. Sem executar o comando seguinte determine e escreva em Português, o que o comando faz e mencione qual o seu resultado. Teste a sua resposta.

```
SQL>SELECT  avg(sal) "Media", ndep "NDep"
  2     FROM emp
  3     WHERE  (ndep, sal) IN (SELECT ndep, max(sal)
  4                            FROM emp
  5                            GROUP BY ndep)
  6         OR (ndep, sal) IN (SELECT ndep, min(sal)
  7                            FROM emp
  8                            GROUP BY ndep)
  9     GROUP BY ndep
 10     HAVING ndep = (SELECT ndep
 11                    FROM emp
 12                    GROUP BY ndep
 13                    HAVING count(*) = (SELECT max(count(*))
 14                                       FROM emp
 15                                       GROUP BY ndep))
 16     ORDER BY ndep, avg(sal);
```

Resposta:



Aula 5

Introdução ao DataArchitect

Introdução à ferramenta de construção de ERs da Sybase, o DataArchitect

Aula 6

Continuação da Aula Anterior

Aula 7

Subconsultas Avançadas e Alteração de Dados

7.1. Subconsultas Correlacionadas

As subconsultas correlacionadas possuem as seguintes características:

- `SELECT` da subconsulta é executado uma vez por cada registo candidato* gerado pelo `SELECT` externo (ao contrário do que acontece em subconsultas não correlacionadas onde o `SELECT` interno é executado apenas uma vez).
- `SELECT` interno usa uma coluna ou pseudónimo do `SELECT` externo.
- Apesar do `SELECT` interno ser executado uma vez para cada registo candidato, não existe nada que indique que a totalidade do comando demore mais tempo de execução†.

Durante a execução do comando é seguida a ordem:

1. Obtenção de registo candidato pelo `SELECT` externo.
2. Execução do `SELECT` interno baseado em valores do registo candidato.
3. Uso dos valores do `SELECT` interno para qualificar ou não o registo candidato.
4. Repetir até não existirem mais registos candidatos.

Exemplo:

```
SELECT nemp, nome, sal, ndep
   FROM emp e
   WHERE sal > (SELECT avg(sal)
                  FROM emp
                  WHERE ndep = e.ndep)
   ORDER BY ndep, sal DESC;

NEMP       NOME                 SAL        NDEP
---------- -------------------- ---------- ----------
      1839 Jorge Sampaio            890000         10
      1788 Maria Dias               565000         20
      1566 Augusto Reis             450975         20
      1902 Catarina Silva           435000         20
      1698 Duarte Guedes            380850         30
      1654 Ana Rodrigues            221250         30
      1521 Nelson Neves             212250         30
7 rows selected.
```

O exemplo anterior selecciona todos os empregados com salário superior à média dos salários do seu departamento. A subconsulta é correlacionada porque o `SELECT` interno usa a informação `e.ndep` proveniente do `SELECT` externo para saber sobre que pessoas calcular a média (as do mesmo departamento). Tente fazer o mesmo sem subconsulta correlacionada!

* Registos candidatos são aqueles que ainda não se sabem se vão ser seleccionados ou não.

† Pelo menos é o que é afirmado nos manuais da Oracle.

7.1.1. Operador EXISTS

Para além dos operadores lógicos e operadores SQL, é frequente aparecer nas subconsultas correlacionadas o operador EXISTS.

O operador EXISTS devolve verdade se a subconsulta produzir uma ou mais linhas e devolve falso caso contrário. Obviamente, se a consulta não fosse correlacionada, o SELECT interno seria executado apenas uma vez, e assim o seu valor seria o mesmo para todos os registos do SELECT externo.

Exemplo:

```
SELECT *
   FROM dep d
   WHERE EXISTS (SELECT *
                    FROM emp
                    WHERE ndep = d.ndep);
```

O exemplo mostra os departamentos nos quais existem empregados:

```
NDEP       NOME            LOCAL
---------- --------------- ---------------
        10 Contabilidade   Condeixa
        20 Investigação    Mealhada
        30 Vendas          Coimbra
3 rows selected.
```

O operador NOT continua a poder ser aplicado. Exemplo:

```
SELECT *
   FROM dep d
   WHERE NOT EXISTS (SELECT *
                        FROM emp
                        WHERE ndep = d.ndep);
```

Mostra os departamentos nos quais não existem quaisquer empregados:

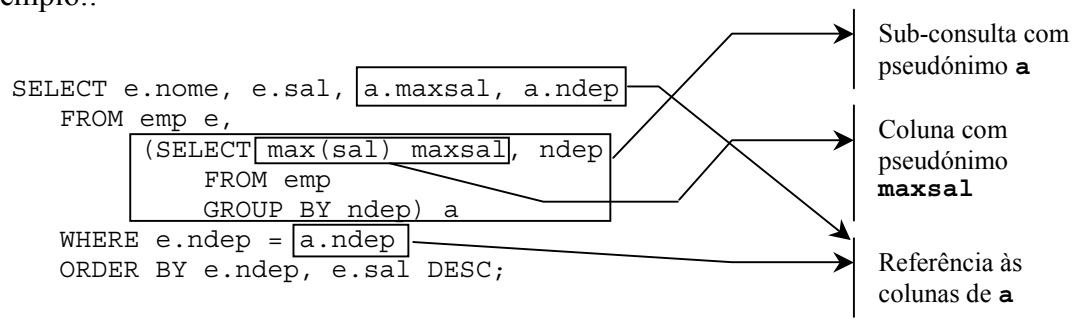
```
NDEP       NOME            LOCAL
---------- --------------- ---------------
        40 Planeamento     Montemor
1 row selected.
```

7.2. Subconsultas na cláusula FROM :

Até ao momento foram vistas subconsultas correlacionadas ou não nas cláusulas WHERE e HAVING do comando SELECT. No comando SELECT podem ainda existir subconsultas na cláusula FROM. As subconsultas podem aparecer também noutros comandos (UPDATE, DELETE, INSERT e CREATE TABLE).

Usar uma subconsulta na cláusula FROM corresponde ao mesmo que usar uma vista na cláusula FROM. Ou seja, em vez de se seleccionar dados de uma tabela, seleccionam-se dados provenientes de um comando SELECT.

Exemplo::



No exemplo a cláusula `FROM` indica a origem dos dados: a tabela **`emp`** com pseudónimo **`e`** a uma subconsulta com pseudónimo **`a`**. A subconsulta não necessita obrigatoriamente de ter um pseudónimo mas é a única maneira de referir as suas colunas. Se não usássemos pseudónimos não poderíamos fazer a junção na cláusula `WHERE` nem poderíamos seleccionar as colunas `MAXSAL` e `NDEP` da subconsulta `a` na cláusula `SELECT`. Note ainda o uso de pseudónimos nas colunas da subconsulta. Repare como se atribuiu à primeira coluna da subconsulta o nome de `maxsal`. E veja como a referência a esse coluna no `SELECT` externo já usa o referência.

O comando mostra os nomes e os salários dos empregados bem como o salário máximo e o número do seu departamento:

```
NOME                 SAL        MAXSAL     NDEP
-------------------- ---------- ---------- ----------
Jorge Sampaio            890000     890000         10
Silvia Teles             279450     890000         10
Olga Costa                68300     890000         10
Maria Dias               565000     565000         20
Augusto Reis             450975     565000         20
Catarina Silva           435000     565000         20
Antonio Silva             70800     565000         20
Rita Pereira              65100     565000         20
Duarte Guedes            380850     380850         30
Ana Rodrigues            221250     380850         30
Nelson Neves             212250     380850         30
Manuel Madeira           157800     380850         30
Joana Mendes             145600     380850         30
Tome Ribeiro              56950     380850         30
14 rows selected.
```

NOTA: O uso de subconsultas na cláusula `FROM` é particularmente útil quando se quer incluir em registos individuais informação de funções de grupo (ver exemplo anterior).

Exemplo:

```
SELECT a.ndep                                                    "Ndep",
       a."N Emp Dep"                                             "#Emp",
       TO_CHAR(a."N Emp Dep"/b."N Emp Total"*100,'999.0')        "%Emp",
       a."Sum Sal Dep"                                           "Sum Sal",
       TO_CHAR(a."Sum Sal Dep"/b."Sum Sal Total"*100,'999.0')    "%Sal"
   FROM (SELECT ndep, COUNT(*) "N Emp Dep", SUM(sal) "Sum Sal Dep"
            FROM emp
            GROUP BY ndep) a,
         (SELECT COUNT(*) "N Emp Total", SUM(sal) "Sum Sal Total"
            FROM emp) b
```

O comando devolve o número do departamento, quantos empregados tem cada departamento, a percentagem de empregados desse departamento em relação ao total, a soma dos salários desse departamento e a percentagem da soma em relação ao total.

Repare como os pseudónimos das subconsultas são usados no SELECT exterior. Se se usar aspas nos pseudónimos também se tem que usar aspas quando se refere a eles.

O comando resulta em:

```
Ndep       #Emp       %Emp   Sum Sal    %Sal
---------- ---------- ------ ---------- ------
        10          3   21.4    1237750   30.9
        20          5   35.7    1586875   39.7
        30          6   42.9    1174700   29.4
3 rows selected.
```

7.3. Manipulação de dados

Entende-se como manipulação de dados a inserção (INSERT), modificação (UPDATE) e apagamento (DELETE) de dados de tabelas.

7.3.1. Inserção de dados (INSERT)

O comando INSERT tem as seguintes opções:

```
INSERT INTO (tabela|vista) [(nome_col1, nome_col2, …)]
   VALUES (valor1, valor2, …)
```

Ou

```
INSERT INTO (tabela|vista) [(nome_col1, nome_col2, …)]
   subconsulta
```

A primeira opção insere apenas um registo de cada vez enquanto que a segunda insere um ou mais registos. Na segunda opção, os valores a inserir são retirados de uma subconsulta.

Exemplo 1 - Inserção de um registo e uso dos nomes das colunas da tabela destino:

```
INSERT INTO emp (nemp, nome, funcao, encar,
                 data_entrada, sal, premios, ndep)
   VALUES (2222, 'Lobo Mau', 'Comilao', null,
           sysdate, 555, null, null);

1 row processed.
```

O primeiro conjunto de parêntesis serve apenas para indicar a que correspondem os valores que estão a ser inseridos. É facultativo desde que os valores mencionados a seguir à cláusula VALUE estejam na mesma ordem das colunas da tabela e desde que sejam todos mencionadas. Assim, o comando do exemplo 1 produz o mesmo resultado que o comando do exemplo 2.

Exemplo 2 – Inserção de um registo sem especificar os nomes das colunas (produz o mesmo resultado que o exemplo 1):

```
INSERT INTO emp
   VALUES (2222, 'Lobo Mau', 'Comilao', null,
           sysdate, 555, null, null);

1 row processed.
```

Exemplo 3 – Inserção de um registo sem incluir todos os dados e alterando a ordem de algumas colunas (produz o mesmo resultado que os dois exemplos anteriores):

```
INSERT INTO emp (nome, funcao, nemp, data_entrada, sal)
   VALUES ('Lobo Mau', 'Comilao', 2222, sysdate, 555)

1 row processed.
```

Exemplo 4 – Inserção de registos através de uma subconsulta. Comando que insere todos os empregados da tabela `emp` outra vez. Note que se somou 1000 ao número do empregado para garantir que após a inserção cada empregado tivesse um valor de `nemp` diferente de todos os outros. Os restantes valores de cada empregado foram mantidos com a excepção do `nome` a que foi acrescentado o prefixo de `'CLONE'` e a `data_entrada` que ficou com o valor da data actual.

```
INSERT INTO emp
   SELECT nemp+1000, 'CLONE ' || nome, funcao, encar,
          sysdate, sal, premios, ndep
      FROM emp;

14 rows processed.
```

Depois de executado, se se seleccionar alguma informação da tabela obtém-se:

```
SELECT nemp, nome, funcao, encar, data_entrada
   FROM emp
   ORDER BY nemp;

NEMP       NOME                 FUNCAO        ENCAR      DATA_ENTR
---------- -------------------- ------------ ---------- ---------
      1369 Antonio Silva        Continuo           1902 22-DEC-96
      1499 Joana Mendes         Vendedor           1698 04-OCT-84
      1521 Nelson Neves         Vendedor           1698 27-FEB-83
      1566 Augusto Reis         Encarregado        1839 13-FEB-85
      1654 Ana Rodrigues        Vendedor           1698 17-DEC-90
      1698 Duarte Guedes        Encarregado        1839 25-NOV-91
      1782 Silvia Teles         Encarregado        1839 03-NOV-86
      1788 Maria Dias           Analista           1566 07-NOV-82
      1839 Jorge Sampaio        Presidente             11-FEB-84
      1844 Manuel Madeira       Vendedor           1698 21-APR-85
      1876 Rita Pereira         Continuo           1788 07-FEB-96
      1900 Tome Ribeiro         Continuo           1698 05-MAR-94
      1902 Catarina Silva       Analista           1566 13-APR-93
      1934 Olga Costa           Continuo           1782 22-JUN-86
      2369 CLONE Antonio Silva  Continuo           1902 04-NOV-98
      2499 CLONE Joana Mendes   Vendedor           1698 04-NOV-98
      2521 CLONE Nelson Neves   Vendedor           1698 04-NOV-98
      2566 CLONE Augusto Reis   Encarregado        1839 04-NOV-98
      2654 CLONE Ana Rodrigues  Vendedor           1698 04-NOV-98
      2698 CLONE Duarte Guedes  Encarregado        1839 04-NOV-98
      2782 CLONE Silvia Teles   Encarregado        1839 04-NOV-98
      2788 CLONE Maria Dias     Analista           1566 04-NOV-98
```

```
      2839 CLONE Jorge Sampaio  Presidente              04-NOV-98
      2844 CLONE Manuel Madeira Vendedor           1698 04-NOV-98
      2876 CLONE Rita Pereira   Continuo           1788 04-NOV-98
      2900 CLONE Tome Ribeiro   Continuo           1698 04-NOV-98
      2902 CLONE Catarina Silva Analista           1566 04-NOV-98
      2934 CLONE Olga Costa     Continuo           1782 04-NOV-98
28 rows selected.
```

NOTA: Apenas no fim da execução do comando `INSERT` é que o SGBD da ORACLE verifica se existiu ou não violação de integridade. No exemplo anterior, se o primeiro elemento a ser inserido tivesse um valor de encar de por exemplo 5555 (que é um número de empregado que não existe) a inserção continuava até ao último dos elementos. Só então, no fim do comando, seria detectada a violação e desfeito o comando. Mas se em vez de 5555, o primeiro elemento (`'CLONE Antonio Silva'`) tivesse o encarregado número 2934 (o número de empregado da última inserção) o resultado já seria diferente. Durante a inserção do `'CLONE Antonio Silva'` existe realmente violação de integridade uma vez que o 2934 ainda não foi inserido. A violação não é detectada porque o comando `INSERT` ainda não terminou. Por outro lado, quando o comando `INSERT` termina, a referência a 2934 já não representa uma violação de integridade porque esse elemento foi entretanto inserido. Ou seja, o `INSERT` permite violar as restrições de dados temporariamente.

7.3.2. Modificação de Dados (UPDATE)

A instrução que permite modificar dados é a `UPDATE`. As suas opções são as seguintes:

```
UPDATE (tabela|vista)
   SET col1 = val1, col2 = val2, …,
       (col101, col102, …) = (subconsulta)
   [WHERE (condição)];
```

A cláusula `UPDATE` determina sobre que tabela ou vista se vão executar as alterações.

A cláusula `SET` indica pares de colunas e valores a atribuir a colunas. Os pares estão separados por vírgulas. Além de ou invés dos pares de colunas pode ainda incluir-se atribuições através de subconsultas. Nesse caso, as colunas a terem valores atribuídos deverão aparecer entre parêntesis e separadas por vírgulas.

A cláusula `WHERE` indica sobre que registos da tabela ou vista serão executadas as alterações. A condição pode ser também uma subconsulta.

As subconsultas podem ser correlacionadas.

Exemplo 1 – Aumenta o salário de todos os empregados em 10%:

```
UPDATE emp
   SET sal = sal * 1.1;
14 rows processed.
```

Exemplo 2 – Aumenta o salário de todos os empregados do departamento 20 em 10%:

```
UPDATE emp
   SET sal = sal * 1.1
   WHERE ndep = 20;
5 rows processed.
```

Exemplo 3 – Muda o salário e a função do `'Jorge Sampaio'`:

```
UPDATE emp
   SET sal = sal * 2, funcao = 'CHEFAO'
   WHERE nome = 'Jorge Sampaio';
1 rows processed.
```

Exemplo 4 – Mistura atribuições normais e subconsultas na cláusula `SET`. Usa subconsulta correlacionada na cláusula `SET` e subconsulta na cláusula `WHERE`. O exemplo altera o número do departamento dos registos afectados para o número do departamento de nome `'Coimbra'` (1ª subconsulta do `SET`). Altera os pares salário, prémios para 110% da média dos salários e 150% da média dos prémios dos empregados do seu departamento (2ª subconsulta do `SET` – subconsulta correlacionada). Efectua as alterações apenas para os registos que correspondam a empregados dos departamentos de `'Coimbra'` e da `'Mealhada'` (subconsulta do `WHERE`).

```
UPDATE emp a
   SET ndep = (SELECT ndep
                  FROM dep
                  WHERE local = 'Coimbra'),
       (sal, premios) = (SELECT 1.1*AVG(sal), 1.5*AVG(premios)
                            FROM emp b
                            WHERE a.ndep = b.ndep)
   WHERE ndep IN (SELECT ndep
                     FROM dep
                     WHERE local = 'Mealhada'
                        OR local = 'Coimbra');
11 rows processed.
```

Após o `UPDATE` o resultado seria:

```
SELECT nome, sal, premios, ndep
   FROM emp
   ORDER BY nome

NOME                 SAL         PREMIOS    NDEP
-------------------- ---------- ---------- ----------
Ana Rodrigues            215362      88575         30
Antonio Silva            349113                    30
Augusto Reis             349113                    30
Catarina Silva           349113                    30
Duarte Guedes            215362      88575         30
Joana Mendes             215362      88575         30
Jorge Sampaio            890000                    10
Manuel Madeira           215362      88575         30
Maria Dias               349113                    30
Nelson Neves             215362      88575         30
Olga Costa                68300                    10
Rita Pereira             349113                    30
Silvia Teles             279450                    10
Tome Ribeiro             215362      88575         30
14 rows selected.
```

Repare como a subconsulta correlacionada funcionou. Os empregados que já eram do departamento 30 (`'Coimbra'`) ficaram com um salário de 215362. Os que eram do 20 (`'Mealhada'`) e passaram para o 30 ficaram com um salário de 349113. Tente perceber porquê (caso não entenda existe sempre a lista da cadeira: `bd@eden.dei.uc.pt` ☺).

Pergunta: Ao todo, quantos `SELECT`s foram executados no último exemplo?

7.3.3. Remoção de Dados (DELETE)

Usa-se o comando DELETE para remover registos das tabelas ou vistas. O comando permite apagar apenas registos inteiros. Não é possível apagar um campo com o DELETE. No máximo poder-se-ia colocar o valor NULL nesse campo através de um comando UPDATE.

As opções do comando DELETE são as seguintes:

```
DELETE FROM tabela|vista
   WHERE condições
```

Como de costume, as condições na cláusula WHERE podem incluir subconsultas e as vistas na cláusula FROM podem ser definidas através de subconsultas.

Exemplo 1 – Apagamento de todos os registos da tabela emp.

```
DELETE FROM emp;
14 rows processed.
```

NOTA: Um erro comum é escrever 'DELETE * FROM emp'. O asterisco usa-se no SELECT para indicar as colunas todas. Mas no comando DELETE não faz sentido indicar a lista de colunas a apagar porque ou se apaga o registo todo ou não se apaga o registo. Não é possível apagar apenas algumas colunas.

Exemplo 2 – Apagamento dos registos da tabela emp que correspondem aos empregados com salários menores que 100000.

```
DELETE
   FROM emp
   WHERE sal < 100000;
4 rows processed.
```

Exemplo 3 – Apagamento dos registos da tabela emp que correspondem aos empregados com salários menores que 100000 e que pertençam ao departamento 20.

```
DELETE
   FROM emp
   WHERE sal < 100000
      AND ndep = 20;
2 rows processed.
```

Exemplo 4 – Apagamento dos registos da tabela `emp` em que a selecção é feita através de subconsultas. O comando apaga todos os empregados do departamento com mais pessoas.

```
DELETE
   FROM emp
   WHERE ndep = (SELECT ndep
                 FROM emp
                 GROUP BY ndep
                 HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                                    FROM emp
                                    GROUP BY ndep));
6 rows processed.
```

7.4. Exercícios

Considere a base de dados de demonstração fornecida.

1. Seleccione o número e o nome dos empregados que são encarregados de mais de dois outros empregados.

```
NEMP       NOME
---------- --------------------
      1839 Jorge Sampaio
      1698 Duarte Guedes
2 rows selected.
```

Resposta:

2. Fazer o mesmo mas a mostrar quantos empregados são geridos por cada encarregado. (subconsulta no `FROM`).

```
NEMP       NOME                 QUANTOS
---------- -------------------- ----------
      1839 Jorge Sampaio                 3
      1698 Duarte Guedes                 5
2 rows selected.
```

Resposta:

3. Abra um browser (Netscape ou Internet Explorer) no link http://www.dei.uc.pt/~bizarro/aulas/bd1/2000/aula07/criaEmp2.sql e execute o conteúdo desse ficheiro. Alternativamente digite manualmente o comando:

```
CREATE TABLE emp2
   (nemp          NUMBER(4)      CONSTRAINT pk_emp PRIMARY KEY,
    nome          VARCHAR2(20)   CONSTRAINT nn_nome NOT NULL
                                 CONSTRAINT upper_nome
                                    CHECK (nome = UPPER(nome)),
    funcao        VARCHAR2(12),
    encar         NUMBER         CONSTRAINT fk_encar
                                    REFERENCES emp2(nemp),
    data_entrada  DATE           DEFAULT SYSDATE,
    sal           NUMBER(7)      CONSTRAINT ck_sal
                                    CHECK (sal > 70000),
    premios       NUMBER(7)      DEFAULT NULL,
    ndep          NUMBER(2)      CONSTRAINT nn_ndep NOT NULL
                                 CONSTRAINT fk_ndep
                                    REFERENCES dep(ndep)
   );
```

number ->NUMERIC
varchar2 -> varchar

O comando cria uma nova tabela, `emp2` muito semelhante a `emp`.

Se a tabela já existir apague-a com `'drop table emp2'` e execute novamente o comando de criação da tabela. Repare que a tabela `emp2` é quase uma cópia da `emp`. As principais diferenças são algumas restrições nas colunas `nome` e `sal`.

Insira em `emp2` o empregado com as seguintes características:

```
nemp = 5555,
nome = 'CHICO FININHO',
funcao = 'Cantor',
encar = null,
data_entrada = hoje,
sal = 100000,
premios = null
ndep = 40.
```

Resposta:

Naprawiaony kod:

```
CREATE TABLE emp2
 (nemp NUMERIC(4),
 nome VARCHAR(20) CONSTRAINT nn_nome NOT NULL,
 funcao VARCHAR(12),
 encar NUMERIC CONSTRAINT fk_encar REFERENCES emp2
(nemp),
 data_entrada DATE DEFAULT now(),
 sal NUMERIC(7),
 premios NUMERIC(7) DEFAULT NULL,
 ndep NUMERIC(2) CONSTRAINT nn_ndep NOT NULL
     CONSTRAINT fk_ndep REFERENCES dep(ndep),

CONSTRAINT pk_emp PRIMARY KEY(nemp),
CONSTRAINT upper_nome CHECK (nome = UPPER(nome)),
CONSTRAINT ck_sal CHECK (sal > 70000)
 );
```

4. Insira em `emp2` todos os empregados de `emp` excepto os de números 1902 e 1369. Note que ao inserir os dados terá que ter algum cuidado a formatar as colunas de `nome` e `sal`. Deverá converter o `nome` para maiúsculas e o `sal` para pelo menos 70001. Depois de inseridos os dados, a instrução `'SELECT nemp, nome, encar, sal, premios, ndep FROM emp2 ORDER BY ndep;'` devolve o seguinte resultado:

```
NEMP       NOME                 ENCAR      SAL        PREMIOS    NDEP
---------- -------------------- ---------- ---------- ---------- ----------
      1839 JORGE SAMPAIO                    890000                    10
      1782 SILVIA TELES              1839     279450                    10
      1934 OLGA COSTA                1782      70001                    10
      1566 AUGUSTO REIS              1839     450975                    20
      1876 RITA PEREIRA              1788      70001                    20
      1788 MARIA DIAS                1566     565000                    20
      1698 DUARTE GUEDES             1839     380850                    30
      1844 MANUEL MADEIRA            1698     157800          0         30
      1900 TOME RIBEIRO              1698      70001                    30
      1654 ANA RODRIGUES             1698     221250      81400         30
      1499 JOANA MENDES              1698     145600      56300         30
      1521 NELSON NEVES              1698     212250      98500         30
      5555 CHICO FININHO                      100000                    40
13 rows selected.
```

Qual deverá ser o comando `INSERT`? (Note que o empregado `'CHICO FININHO'` resulta do comando anterior e não deste.)

Resposta:

5a. Apague todos os empregados do departamento 10. Tente perceber porque é que obtém um código de erro.

5b. Apague então todos os empregados do departamento 20. Após o comando, a tabela `emp2` deverá conter os valores (nem todas as colunas são mostradas):

```
SQLWKS> SELECT nemp, nome, encar, sal, premios, ndep
     2>    FROM emp2
     3>    ORDER BY ndep;

NEMP       NOME                 ENCAR      SAL        PREMIOS    NDEP
---------- -------------------- ---------- ---------- ---------- ----------
      1839 JORGE SAMPAIO                      890000                    10
      1782 SILVIA TELES                 1839 279450                    10
      1934 OLGA COSTA                   1782  70001                    10
      1698 DUARTE GUEDES                1839 380850                    30
      1521 NELSON NEVES                 1698 212250      98500         30
      1844 MANUEL MADEIRA               1698 157800          0         30
      1900 TOME RIBEIRO                 1698  70001                    30
      1654 ANA RODRIGUES                1698 221250      81400         30
      1499 JOANA MENDES                 1698 145600      56300         30
      5555 CHICO FININHO                      100000                    40
10 rows selected.
```

Resposta:

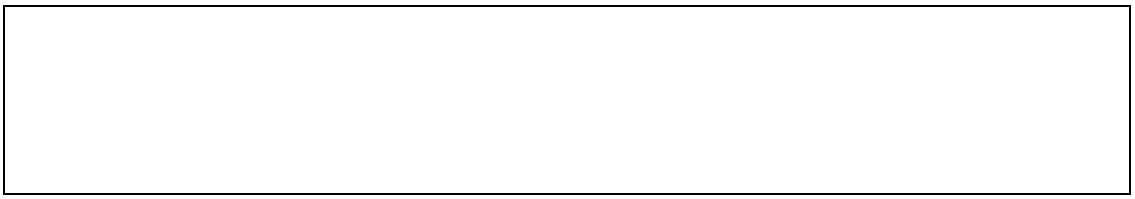
6. Apague de `emp2` o empregado de número 1654.

Resposta:

7. Apague de `emp2` os empregados que recebam menos que a média de salários dos empregados do seu departamento (use subconsulta correlacionada).

```
SQLWKS> SELECT nemp, nome, encar, sal, premios, ndep
     2>    FROM emp2
     3>    ORDER BY ndep;
NEMP       NOME                 ENCAR      SAL        PREMIOS    NDEP
---------- -------------------- ---------- ---------- ---------- ----------
      1839 JORGE SAMPAIO                    890000                    10
      1698 DUARTE GUEDES              1839     380850                    30
      1521 NELSON NEVES               1698     212250      98500         30
      5555 CHICO FININHO                       100000                    40
4 rows selected.
```

Resposta:

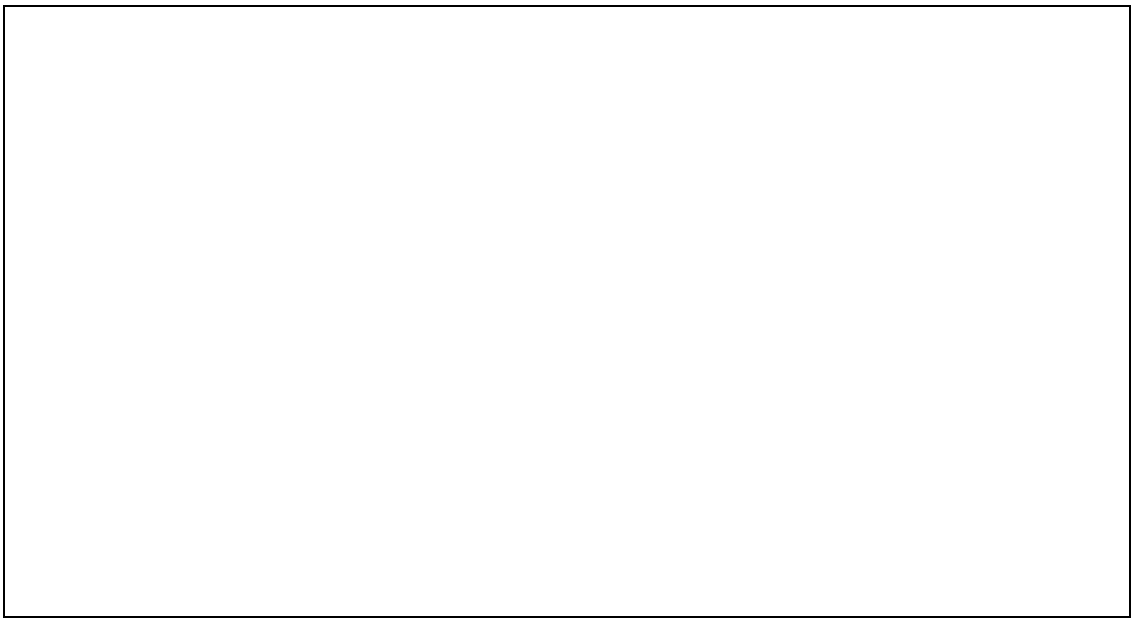


8. Apague todos os elementos de `emp2`. Insira cópias dos empregados de `emp` em `emp2` com o cuidado de alterar o nome para maiúsculas e garantir que o salário cumpre as restrições. Altere os elementos dos empregados da seguinte maneira: mude os empregados do departamento 10 para o 20, os do 20 para o 30 e os do 30 para o 40. Altere ainda os seus salários para 110% da média de empregados do seu departamento e altere os premios para 150% da média dos prémios do seu departamento (subconsulta correlacionada). Altere apenas os empregados que não sejam os que ganhem mais de cada departamento. Execute todas as alterações num único comando `UPDATE`.

Após as alterações, a tabela `emp2` deverá conter os dados:

```
SQLWKS> SELECT nemp, nome, encar, sal, premios, ndep
     2>    FROM emp2
     3>    ORDER BY ndep;
NEMP        NOME                 ENCAR      SAL        PREMIOS    NDEP
---------- -------------------- ---------- ---------- ---------- ----------
      1839 JORGE SAMPAIO                      890000                    10
      1782 SILVIA TELES               1839     454465                    20
      1934 OLGA COSTA                 1782     454465                    20
      1788 MARIA DIAS                 1566     565000                    20
      1566 AUGUSTO REIS               1839     350191                    30
      1698 DUARTE GUEDES              1839     380850                    30
      1902 CATARINA SILVA             1566     350191                    30
      1369 ANTONIO SILVA              1902     350191                    30
      1876 RITA PEREIRA               1788     350191                    30
      1499 JOANA MENDES               1698     217754      88575         40
      1521 NELSON NEVES               1698     217754      88575         40
      1654 ANA RODRIGUES              1698     217754      88575         40
      1844 MANUEL MADEIRA             1698     217754      88575         40
      1900 TOME RIBEIRO               1698     217754      88575         40
14 rows selected.
```

Resposta:



9. Por fim apague a tabele emp2 com comando '`DROP TABLE emp2`'.

Aula 8

Criar, Alterar e Apagar Tabelas e Sequências

Em aulas anteriores já foi visto como seleccionar dados de tabelas, como inserir dados, como modificar dados ou como apagar dados. Os comandos de manipulação de dados vistos até ao momento (`SELECT`, `INSERT`, `UPDATE` e `DELETE`) fazem parte de um subconjunto de comandos do SQL intitulado *DML – Data Manipulation Language*, ou seja *Linguagem de Manipulação de Dados*. Por outro lado, os comandos relativos à criação, alteração e apagamento de objectos (`CREATE`, `ALTER` e `DROP`) fazem parte de um subconjunto de comandos do SQL intitulado *DDL – Data Definition Language*, ou seja *Linguagem de Definição de Dados*. Através dos comandos DDL pode-se criar objectos como tabelas, vistas, sequências, procedimentos, funções, etc.

Os comandos do tipo DML são usados frequentemente durante a vida da BD. Os comandos do tipo DDL, por provocarem mudanças significativas na estrutura da BD são de uso muito menos frequente. Normalmente, usam-se os comandos DDL na criação e destruição da BD ou para mudanças pontuais da sua estrutura. Pode estabelecer-se um paralelo com a instalação, desinstalação e *upgrades* de *software* para os comandos DDL e uso das funções do *software* para os comandos DML.

8.1. Criar Tabelas (CREATE TABLE)

Uma tabela define-se pelo seu nome, pelo nome das suas colunas, pelo tipo de dados que comporta cada coluna, por restrições de colunas e por restrições de tabela. Na realidade existem ainda outras opções avançadas que se podem definir nas tabelas. Essas opções avançadas têm haver com a forma como o servidor guarda a tabela em disco, com parâmetros de optimização de acesso aos dados e outros. Por agora vamos ficar apenas pelas características básicas.

O comando SQL para criar uma tabela é o seguinte:

```
CREATE TABLE nome_tabela
( col1  tipo_col1  [DEFAULT expre] [restrição1 [restrição2 […]]],
  col2  tipo_col2  [DEFAULT expre] [restrição1 [restrição2 […]]],
  col3  tipo_col3  [DEFAULT expre] [restrição1 [restrição2 […]]],
  …,
  coln  tipo_coln  [DEFAULT expre] [restrição1 [restrição2 […]]],
  [restrição1_tabela,]
  [restrição2_tabela,]
  [restrição3_tabela,]
  …,
  [restriçãon_tabela]
);
```

Restrições de coluna

Restrições de tabela

A seguir às palavras `CREATE TABLE` declara-se o nome da tabela a criar. Entre parênteses define-se uma lista de colunas ou restrições à tabela separadas por vírgulas. Ao definir-se as colunas determina-se qual o nome da coluna, que tipo de dados comporta, qual o valor por omissão dos dados nessa coluna e que restrições se devem verificar sobre os dados dessa coluna. As restrições da tabela indicam normalmente restrições sobre dados de mais do que uma coluna e usam uma sintaxe ligeiramente diferente porque indicam explicitamente a que coluna(s) se referem. As restrições de coluna, por outro lado, indicam implicitamente a que coluna se referem através da linha em que se localizam.

Exemplo 8.1 – Criar uma tabela, `alunos`, com duas colunas, `numero` e `nome`. A primeira coluna é do tipo numérico com até 10 algarismos e a segunda é do tipo `varchar2` (cadeia de caracteres) com espaço até 40 caracteres por nome. Não é indicada nenhuma restrição sobre nenhuma coluna da tabela nem é definida nenhuma chave primária.

```
CREATE TABLE alunos
(numero       NUMBER(10),
 nome         VARCHAR2(40)
);
```

8.1.1. Restrição de Chave Primária (PRIMARY KEY)

A chave primária de uma tabela é a coluna ou conjunto de colunas que servem para definir univocamente um registo na tabela. Uma coluna de chave primária não pode conter valores nulos nem pode conter valores repetidos.

Apesar de o Oracle permitir que se crie uma tabela sem definir uma chave primária normalmente nenhuma tabela é criada sem uma.

Pode declarar-se a chave primária na zona das restrições de coluna da chave primária ou no conjunto de restrições da tabela. No caso de chaves primárias concatenadas (quando a chave é definida pelos valores de mais de uma coluna) a única maneira de definir a chave é através das restrições da tabela.

Colunas com dados do tipo `LONG` e `LONG RAW` não podem ser chaves primárias de uma tabela.

O **prefixo** normalmente usado para os nomes das restrições de chave primária é **`pk_`** (as iniciais de *primary key*, "*chave primária"* em inglês). Ver *Nomes a Usar nas Restrições* na página *100*.

Para as chaves primárias, o **caracter** usado internamente pelo Oracle para distinguir restrições é o **P** (de *primary*). Investigue as vistas sobre restrições do dicionário de dados.

A sintaxe para se definir uma coluna como chave primária é uma das seguintes:

```
[CONSTRAINT nome_restrição] PRIMARY KEY
```

no caso de se definir a restrição nas restrições de coluna (ver Exemplo 8.2 e Exemplo 8.3), ou

```
[CONSTRAINT nome_restrição] PRIMARY KEY (coluna)
```

no caso de se definir a restrição nas restrições de tabela (ver Exemplo 8.4) ou

```
[CONSTRAINT nome_restrição] PRIMARY KEY (coluna1, coluna2, ...)
```

para chave primárias concatenadas (ver Exemplo 8.5).

Exemplo 8.2 – Definir uma chave primária nas restrições de coluna.

```
CREATE TABLE alunos
(numero       NUMBER(10)    PRIMARY KEY,
 nome         VARCHAR2(40)
);
```

Exemplo 8.3 – Definir uma chave primária nas restrições de coluna e dar um nome à restrição.

```
CREATE TABLE alunos
(numero       NUMBER(10)    CONSTRAINT pk_numero_alunos PRIMARY KEY,
 nome         VARCHAR2(40)
);
```

Note que o nome da restrição é opcional. Caso seja omitido, o Oracle define automaticamente um nome para a restrição. Neste exemplo chama-se `pk_numero_alunos`.

Exemplo 8.4 – Definir uma chave primária nas restrições de tabela e dar um nome à restrição.

```
CREATE TABLE alunos
(numero       NUMBER(10),
 nome         VARCHAR2(40),
 CONSTRAINT   pk_numero_alunos PRIMARY KEY(numero)
);
```

Note que quando se define a restrição de chave primária nas restrições da tabela é necessário indicar qual das colunas é a chave primária e indicar o seu nome entre parêntesis a seguir às palavras `PRIMARY KEY`.

Novamente note que o nome da restrição é opcional. Caso seja omitido, o Oracle define automaticamente um nome para a restrição.

Exemplo 8.5 – Definir uma chave primária concatenada nas restrições de tabela e dar um nome à restrição. No exemplo assume-se que o aluno é definido univocamente pelo seu ano de entrada e pelo seu número de entrada nesse ano.

```
CREATE TABLE alunos
(ano_entr      NUMBER(6),
 num_entr      NUMBER(4),
 nome          VARCHAR2(40),
 CONSTRAINT    pk_ano_num_alunos PRIMARY KEY(ano_entr, num_entr));
```

Para chaves primárias concatenadas, o único local possível para a definição da restrição é na secção de restrições da tabela. A chave primária da tabela pode ser definida pela concatenação de todas as colunas da tabela. O nome da restrição é opcional.

8.1.2. Restrição de Unicidade (UNIQUE)

Uma coluna que obedeça à restrição de Unicidade não pode ter valores repetidos. Note no entanto que uma coluna nestas condições **PODE conter valores nulos**. Assim, uma coluna deste género não é nunca uma chave candidata* por causa da possibilidade de existirem valores nulos.

Pode declarar-se a coluna com restrições de Unicidade nas restrições dessa coluna ou no conjunto de restrições da tabela.

* Uma chave candidata é uma coluna que à semelhança da chave primária identifica univocamente um registo numa tabela. Ou seja, uma chave candidata também poderia ser chave primária.

O **prefixo** normalmente usado para os nomes das restrições de unicidade é **uni_** ou **unq_** (as iniciais de *unique*, "*único"* em inglês). Ver Nomes a Usar nas Restrições na página 100.

Para as colunas do tipo UNIQUE, o **caracter** usado internamente pelo Oracle para distinguir restrições é o **U** (de *unique*).

A sintaxe para se definir uma coluna com restrição de Unicidade é uma das seguintes:

```
[CONSTRAINT nome_restrição] UNIQUE
```

no caso de se definir a restrição nas restrições de coluna (ver Exemplo 8.6), ou

```
[CONSTRAINT nome_restrição] UNIQUE (coluna1, coluna2, ...)
```

no caso de se definir a restrição nas restrições de tabela (ver Exemplo 8.7). Note que se poderia ter mais do que uma coluna na restrição UNIQUE à semelhança do que se pode fazer com as chaves primárias. Nesse caso a restrição significa que o conjunto de valores compostos pelas colunas em causa é único.

Exemplo 8.6 – Definir uma coluna, `bi`, com restrição de Unicidade e dar um nome à restrição.

```
CREATE TABLE alunos
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   ,
 bi           NUMBER(10)     CONSTRAINT uni_bi_alunos UNIQUE
);
```

Exemplo 8.7 – Definir uma coluna, `bi`, com restrição de Unicidade e dar um nome à restrição. A diferença em relação ao Exemplo 8.6 é que a restrição é declarada na zona de restrições da tabela em vez de estar na zona de restrições da coluna.

```
CREATE TABLE alunos
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   ,
 bi           NUMBER(10)     ,
 CONSTRAINT   uni_bi_alunos  UNIQUE(bi));
```

8.1.3. Restrição de Integridade Geral (CHECK)

É possível obrigar a que na inserção de dados sejam verificadas condições. As condições a verificar apenas permitem a inserção de dados se o valor devolvido for Verdade ou Desconhecido (ver Exemplo 8.8) no caso de existir comparações com valores nulos.

As condições a verificar podem incluir expressões aritméticas e chamadas de funções SQL.

As condições a verificar não podem incluir:

- chamadas às funções `SYSDATE`, `UID`, `USER` e `USERENV`
- referências às pseudo-colunas `CURRVAL`, `NEXTVAL`, `LEVEL` e `ROWNUM`
- subconsultas
- referências a colunas de outras tabelas ou a outras linhas da mesma tabela

Numa restrição de integridade se aparecerem referências a colunas da mesma tabela assume-se que indicam campos desse mesmo registo. Este tipo de restrições onde se

refere mais do que uma coluna na mesma restrição terá que ser definida na zona de restrições da tabela.

Pode declarar-se a restrição de integridade na zona de restrições da coluna ou na zona de restrições da tabela.

O **prefixo** normalmente usado para os nomes das restrições de integridade gerais é **ck_** (de *check*, "*verifica que"* em inglês). Ver *Nomes a Usar nas Restrições* na página *100*.

Para as colunas deste tipo, o **caracter** usado internamente pelo Oracle para distinguir restrições é o **C** (de *check*). Ver vistas sobre restrições no Apêndice A – Dicionário de Dados.

A sintaxe para se definir uma coluna com restrição de integridade geral é:

```
[CONSTRAINT nome_restrição] CHECK (condicao)
```

Exemplo 8.8 – Definir uma coluna, `ano`, que se contiver um valor terá que ser obrigatoriamente acima de 1980 e dar um nome à restrição.

```
CREATE TABLE alunos
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   ,
 ano          NUMBER(4)    CONSTRAINT ck_ano_alunos CHECK(ano>1980)
 );
```

Note que pode inserir-se um registo com o valor do campo `ano` a `NULL`. Nesse caso a condição devolve o valor de Desconhecido e permite a inserção. Veja o resultado de algumas tentativas de inserção na tabela criada:

```
SQLWKS> INSERT INTO alunos VALUES (1111, 'Ana',   1970);
ORA-02290: check constraint (UTILIZADOR1.CK_ANO_ALUNOS) violated
SQLWKS> INSERT INTO alunos VALUES (2222, 'Julia', NULL);
1 row processed.
SQLWKS> INSERT INTO alunos VALUES (3333, 'Romeu', 1981);
1 row processed.
```

Exemplo 8.9 – Faz o mesmo que o Exemplo 8.8 mas define a restrição na zona de restrições da tabela.

```
CREATE TABLE alunos
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   ,
 ano          NUMBER(4)      ,
 CONSTRAINT   ck_ano_alunos  CHECK(ano>1980)
);
```

Exemplo 8.10 – Criar a tabela `empregados` com duas restrições (para além da de chave primária). A primeira restrição, `ck_nome_empregados`, obriga a que os nomes tenham um espaço no meio, ou seja, que exista pelo menos um nome e uma apelido para cada empregado. A outra restrição, `ck_funcao_empregado`, obriga a que a função do empregado seja uma de entre `'Presidente'`, `'Analista'` e `'Vendedor'`.

```
CREATE TABLE empregados
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   ,
 funcao       VARCHAR2(20)   ,
 CONSTRAINT   ck_nome_empregados   CHECK (nome LIKE '% %'),
 CONSTRAINT   ck_funcao_empregados CHECK (funcao IN ('Presidente',
                                                     'Analista',
                                                     'Vendedor'))
);
```

Exemplo 8.11 – É possível o uso de funções avançadas como o `DECODE` para garantir que por exemplo, o par da `'Julieta'` é sempre o `'Romeu'` e o da `'Cleopatra'` é sempre o `'Cesar'`.

```
CREATE TABLE couple
(num          NUMBER(4)      PRIMARY KEY,
 boy          VARCHAR2(10)   ,
 girl         VARCHAR2(10)   ,
 CONSTRAINT   ck_boy_girl_couple
                             CHECK (boy = DECODE (girl,
                                                  'Julieta',   'Romeu',
                                                  'Cleopatra', 'Cesar',
                                                  boy))
);
```

Veja exemplos de comandos `INSERT` nessa tabela:

```
SQLWKS> INSERT INTO couple VALUES (1111, 'Cesar', 'Julieta');
ORA-02290: check constraint (UTILIZADOR1.CK_BOY_GIRL_COUPLE) violated

SQLWKS> INSERT INTO couple VALUES (2222, 'Romeu', 'Julieta');
1 row processed.

SQLWKS> INSERT INTO couple VALUES (3333, 'Zeca',  'Xica');
1 row processed.
```

Exemplo 8.12 – Criar a tabela `empregados` com várias restrições. O salário tem que ser maior que 55 e menor que 800. O salário mensal também tem que ser superior ao valor dos prémios e o que o empregado recebe anualmente nunca pode ser superior a 5600. Note que o Oracle não verifica se as condições entram em conflito ou se são mutuamente exclusivas ou não. Repare, por exemplo, que um salário de 600 contos, apesar de passar na segunda restrição nunca passa na penúltima independentemente do valor dos prémios.

```
CREATE TABLE empregados
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   ,
 sal          NUMBER(4)      ,
 premios      NUMBER(4)      ,
 CONSTRAINT   upper_nome_empregados   CHECK (nome = UPPER(nome),
 CONSTRAINT   ck_sal_empregados       CHECK (sal BETWEEN 55 AND 800),
 CONSTRAINT   ck_sal_prem_empregados  CHECK (sal > premios),
 CONSTRAINT   ck_sal_anual_empregados
                    CHECK (sal * 14 + NVL(premios, 0) < 5600),
 CONSTRAINT   nn_sal_empregados       CHECK (sal IS NOT NULL));
```

A primeira restrição **NÃO converte** o nome para maiúsculas como pode levar a crer. O que a condição faz é verificar se o nome é igual à sua versão em maiúsculas. Se for a inserção pode ser efectuada. Ou seja, a restrição obriga a que apenas sejam inseridos nomes em maiúsculas. Apesar de ser uma restrição do tipo `CHECK`, usa-se o prefixo `upper_` por ser mais indicativo do significado da restrição. Ver *Nomes a Usar nas Restrições* na página *100*.

A restrição de impedir valores nulos numa coluna é um caso particular de uma restrição do tipo `CHECK`. Usa-se normalmente outro prefixo, `nn_`, por ser mais indicativo do significado da restrição. Ver *Nomes a Usar nas Restrições* na página *100*. Ver subcapítulo seguinte sobre restrição a valores nulos.

8.1.4. Restrição de Proibição de Valores Nulos (NOT NULL)

Uma coluna sem valores nulos é uma em que é sempre obrigatória a entrada de um valor. Os valores podem ser repetidos.

Pode declarar-se a coluna sem valores nulos nas restrições dessa coluna ou no conjunto de restrições da tabela.

O **prefixo** normalmente usado para os nomes das restrições de colunas sem valores nulos é **`nn_`** (as iniciais de *not null*, "*não nulos"* em inglês). Ver *Nomes a Usar nas Restrições* na página *100*.

Para as colunas do tipo NOT NULL, o **caracter** usado internamente pelo Oracle para distinguir restrições é o **C** (de *check*). Ver vistas sobre restrições no Apêndice A – Dicionário de Dados. Na realidade, a restrição NOT NULL é um caso particular da restrição geral CHECK, e daí o uso da letra C. Ver subcapítulo *Restrição de Integridade Geral (CHECK)*.

A sintaxe para se definir uma coluna com restrição de valores não nulos é uma das seguintes:

```
[CONSTRAINT nome_restrição] NOT NULL
```

no caso de se definir a restrição nas restrições de coluna (ver Exemplo 8.13), ou

```
[CONSTRAINT nome_restrição] CHECK (coluna IS NOT NULL)
```

no caso de se definir a restrição nas restrições de tabela (ver Exemplo 8.14).

Exemplo 8.13 – Definir uma coluna, `nome`, sem valores nulos e dar um nome à restrição.

```
CREATE TABLE alunos
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   CONSTRAINT nn_nome_alunos NOT NULL
);
```

Exemplo 8.14 – Definir uma coluna, `nome`, sem valores nulos e dar um nome à restrição. A diferença em relação ao Exemplo 8.13 é que a restrição é declarada na zona de restrições da tabela em vez de estar na zona de restrições da coluna.

```
CREATE TABLE alunos
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   ,
 CONSTRAINT   nn_nome_alunos CHECK (nome IS NOT NULL)
);
```

8.1.5. Restrições de Chave Forasteira (FOREIGN KEY… REFERENCES)

Uma chave forasteira ou estrangeira (`FOREIGN KEY`) é uma referência a um valor que existe noutra coluna da mesma tabela ou de outra tabela. A coluna referenciada tem que ser uma chave primária ou uma coluna `UNIQUE`. A chave forasteira (simples ou composta) e a(s) coluna(s) referenciada(s) precisam concordar em tipo de dados e em número de colunas. Uma chave forasteira não pode ser do tipo `LONG` nem do tipo `LONG RAW`. Numa coluna de chave forasteira podem existir valores repetidos e valores nulos.

O **prefixo** normalmente usado para os nomes das restrições de colunas de chave forasteira é **`fk_`** (as iniciais de *foreign key*, "*chave forasteira"* em inglês). Ver *Nomes a Usar nas Restrições* na página *100*.

Para as colunas de chave forasteira, o **caracter** usado internamente pelo Oracle para distinguir restrições é o **R** (de *references*). Ver vistas sobre restrições no Apêndice A – Dicionário de Dados.

A sintaxe para se definir uma coluna com restrição de chave forasteira é uma das seguintes:

```
[CONSTRAINT nome_restrição] REFERENCES tabela[(coluna)]
                            [ON DELETE CASCADE]
```

no caso de se definir a restrição nas restrições de coluna (ver Exemplo 8.15). A seguir à palavra `REFERENCES` indica-se o nome da tabela e coluna referenciadas. O nome da coluna é opcional e se não for mencionado é usada a coluna de chave primária da tabela referida.

A opção `ON DELETE CASCADE` indica que se o registo da chave referenciada for apagado então geram-se apagamentos em cascata para todos os registos dependentes desse.

Alternativamente pode definir-se uma chave forasteira da seguinte forma:

```
[CONSTRAINT nome_restrição] FOREIGN KEY(col1, col2, ...)
                            REFERENCES tabela(colA, colB, ...)
                            [ON DELETE CASCADE]
```

no caso de se definir a restrição nas restrições de tabela (ver Exemplo 8.16). Note que a única maneira de definir chaves forasteiras compostas é usando a sintaxe anterior.

Exemplo 8.15 – Uso de chave forasteira simples. Neste exemplo considera-se que cada aluno teve um padrinho de praxe académica. O padrinho não é mais que uma referência a outro aluno. Cada aluno pertence também a um departamento cuja descrição está guardada numa tabela de departamentos. Para simplificar, as restantes restrições são declaradas sem nome.

```
CREATE TABLE alunos
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   NOT NULL,
 bi           NUMBER(10)     UNIQUE,
 morada       VARCHAR2(80)   ,
 padrinho     NUMBER(10)     CONSTRAINT fk_padrinho_alunos
                                REFERENCES alunos(numero),
 ndep         NUMBER(3)      CONSTRAINT fk_ndep_alunos
                                REFERENCES departamentos(ndep)
                                ON DELETE CASCADE
);
```

A restrição `fk_padrinho_alunos` indica apenas que a coluna `padrinho` guarda um número que terá que aparecer na coluna `numero` da tabela `alunos`. Da mesma forma, a restrição `fk_ndep_alunos` indica que o número de departamento do aluno, `ndep`, terá que indicar um valor que já exista na coluna com o mesmo nome na tabela `departamentos`.

A opção `ON DELETE CASCADE` nesta chave forasteira indica que quando se apaga o registo pai, todos os dependentes são apagados também. Ou seja, se se apagar um departamento da tabela de `departamentos` todos os seus alunos são também apagados. No caso da chave forasteira `padrinho`, isso não acontece. Assim, só é possível apagar um aluno se ele não for o padrinho de ninguém.

Exemplo 8.16 – Uso de chave forasteira compostas ou concatenadas. Neste exemplo considera-se que a BD tem 4 tabelas: `alunos`, `disciplinas`, `exame_freq` e `notas`. A tabela `exame_freq` tem como chave forasteira o código da disciplina. A tabela `notas` tem várias chaves forasteiras. Uma é o número de aluno que se refere à tabela de `alunos`, outra é o código da disciplina que se refere à tabela das `disciplinas` (sem referir a coluna) e outra que indica a época em que saiu a nota que é uma chave forasteira composta que se refere à chave primária (composta também) da tabela `exame_freq`. Como na tabela de `notas` se indica `ON DELETE CASCADE` para a chave forasteira do número de aluno isso significa que se o aluno for apagado todas as suas notas são apagadas também. Pelo contrário, como não se indica `ON DELETE CASCADE` para a chave forasteira `disc` em `notas`, isso significa que uma disciplina só pode ser apagada se não existirem notas atribuídas a essa disciplina.

```
CREATE TABLE alunos
(num          NUMBER(10)    PRIMARY KEY,
 nome         VARCHAR2(30) NOT NULL
);

CREATE TABLE disciplinas
(disc         NUMBER(4)     PRIMARY KEY,
 nome         VARCHAR2(30) NOT NULL);

CREATE TABLE exame_freq
(ano_lec      NUMBER(4)     ,
 epoca        VARCHAR2(20) ,
 disc         NUMBER(4)     REFERENCES disciplinas,
 data         DATE          ,
 PRIMARY KEY (ano_lec, epoca, disc)
);

CREATE TABLE notas
(ano_lec      NUMBER(4)     ,
 epoca        VARCHAR2(20) ,
 disc         NUMBER(4)     REFERENCES disciplinas,
 num_alu      NUMBER(10)    REFERENCES alunos(num)
                               ON DELETE CASCADE,
 CONSTRAINT   fk_notas      FOREIGN KEY(ano_lec, epoca, disc)
                            REFERENCES exame_freq(ano_lec, epoca, disc)
);
```

8.1.6. Nomes a Usar nas Restrições

A importância de atribuir nomes significativos às restrições ainda não foi realçada. Os nomes das restrições são usados pelo Oracle para dar uma pista ao utilizador do que é que correu mal durante uma instrução SQL.

Por exemplo imagine-se que se cria a tabela:

```
CREATE TABLE alunos
(numero       NUMBER(10)     PRIMARY KEY,
 nome         VARCHAR2(40)   NOT NULL,
 bi           NUMBER(10)     UNIQUE,
 morada       VARCHAR2(80)   ,
 entrada      DATE           DEFAULT SYSDATE
);
```

Se se tentar inserir dois registos com a mesma chave primária o sistema devolve o seguinte código de erro:

```
ORA-00001: unique constraint (UTILIZADOR1.SYS_C002403) violated
```

O que isto significa é que o comando que se tentava executar violava a restrição `SYS_C002403` do `UTILIZADOR1`. E isto transmite muito pouca informação útil. `SYS_C002403` é o nome que o Oracle atribuiu à restrição de chave primária na tabela anterior.

Se se tivesse criado a tabela da seguinte forma:

```
CREATE TABLE alunos
(numero       NUMBER(10)     CONSTRAINT pk_numero_alunos PRIMARY KEY,
 nome         VARCHAR2(40)   CONSTRAINT nn_nome_alunos NOT NULL,
 bi           NUMBER(10)     CONSTRAINT uni_bi_alunos UNIQUE,
 morada       VARCHAR2(80)   ,
 entrada      DATE          DEFAULT SYSDATE
);
```

para a mesma situação, o código de erro já seria o seguinte:

```
ORA-00001: unique constraint (UTILIZADOR1.PK_NUMERO_ALUNOS) violated
```

Desta vez o código de erro inclui o nome da restrição definido na criação da tabela. É portanto importante definir nomes de restrições que permitam saber exactamente que tipo de restrição (em que tabela, em que coluna, chave primária, chave forasteira, `NOT NULL`, etc.) impediu a execução normal do comando.

Cada programador ou equipa de programadores deve definir o seu próprio esquema de nomeação de restrições. Não existe nenhuma regra fixa mas uma sugestão é usar:

`descrição_coluna_tabela`

Onde **`descrição`** indica que tipo de restrição se está a usar, **`coluna`** é o nome sobre o qual a restrição actua e **`tabela`** é o nome da tabela dessa coluna.

Normalmente, `descricao` tem um dos seguintes valores:

Descrição	Tipo de Restrição
`pk`	`PRIMARY KEY`
`fk`	`FOREIGN KEY`
`ck`	`CHECK`
`nn`	`NOT NULL`
`uni` ou `unq`	`UNIQUE`
Outro	Quando o prefixo indica melhor que tipo de condição se pretende assegurar.

A parte `coluna` às vezes não é incluída no nome da restrição por ser óbvio a que coluna se refere. Normalmente acontece nas chaves primárias. Às vezes inclui-se ainda mais do que um nome de coluna (caso de chaves concatenadas) ou um nome mais significativo que a coluna (quando se faz comparações complicadas, por exemplo).

Exemplos de nomes de restrições aconselhados:

```
pk_num_aluno
pk_aluno
fk_padrinho_aluno
nn_nome_aluno
ck_sal_emp
ck_sal_premios_emp
ck_ganho_annual_emp
upper_nome_aluno
```

Exemplos de nomes de restrições desaconselhados:

`restricao1_emp` – não indica qual a restrição em causa
`ck_emp` – não indica qual a coluna e não é obvio qual é
`pk_num` – não indica qual a tabela

8.1.7. Valores por Omissão (DEFAULT)

Quando se inserem registos numa tabela é possível não especificar todos os valores de todos os campos. Os campos em falta serão inseridos com um valor por omissão, se este estiver definido, ou com `NULL`.

Os valores por omissão a usar em cada campo durante a inserção são definidos no comando de criação da tabela em causa.

A sintaxe para definir um valor por omissão para uma coluna é a seguinte:

```
CREATE TABLE tabela
(coluna1      tipo_coluna1    DEFAULT valor1    [restrições_coluna1],
 coluna2      tipo_coluna2    DEFAULT valor2    [restrições_coluna2],
 ...
);
```

Se o valor por omissão for especificado terá que aparecer imediatamente a seguir ao tipo de dados.

O valor não pode conter referências a colunas nem a pseudo-colunas e tem que ser do mesmo tipo que a coluna em causa. Veja o exemplo seguinte.

Exemplo 8.17 – Para além dos restrições de chave primária e de valores não nulos, o comando de criação da tabela `empregados` também define quais os valores a usar para as colunas `data_entrada`, `funcao` e `data_pagamento`.

```
CREATE TABLE empregados
(numero          NUMBER(10)      PRIMARY KEY,
 nome            VARCHAR2(40)    NOT NULL,
 data_entrada    DATE            DEFAULT SYSDATE,
 funcao          VARCHAR2(20)    DEFAULT 'Programador',
 sal             NUMBER(6)       DEFAULT NULL,
 premios         NUMBER(6)       ,
 data_pagamento DATE             DEFAULT
                              NEXT_DAY(ROUND(SYSDATE+14, 'MONTH'), 'TUE')
);
```

Se se inserir um registo especificando apenas valores para os campos `numero` e `nome`, em todos os outros serão usados os valores por omissão:

```
INSERT INTO empregados (numero, nome)
   VALUES (1111, 'Joca');

ALTER SESSION
   SET NLS_DATE_FORMAT = 'yy-mm-dd';

SET DATEWIDTH 15;
SET CHARWIDTH 15;

SELECT * FROM empregados;

NUMERO  NOME  DATA_ENTRADA  FUNCAO       SAL  PREMIOS  DATA_PAGAMENTO
------- ----- ------------- ------------ ---- -------- ---------------
   1111 Joca   98-11-17      Programador               98-12-08
1 row selected.
```

NOTA: Repare que `DEFAULT NULL` é redundante uma vez que normalmente, se não existir nenhum valor por omissão definido será sempre usado o `NULL`. Veja os casos dos campos `sal` e `premios`.

NOTA: Os comandos `SET` não fazem parte do programa. Para ter uma noção do que é que pode alterar experimente o comando `SHOW ALL`. Se tiver tempo e paciência brinque um pouco com os parâmetros alterando alguns dos valores.

8.1.8. Criar Tabelas Com Subconsultas (CREATE TABLE ... AS SELECT...)

O comando de `CREATE TABLE` permite uso de subconsultas. A subconsulta a usar faz duas coisas: copia os dados de outras tabelas para a tabela a criar e determina os nomes e os tipos de dados das colunas da nova tabela.

A sintaxe para usar subconsultas no comando `CREATE TABLE` é:

```
CREATE TABLE tabela AS
   subconsulta
```

A subconsulta pode por sua vez conter outras subconsultas.

NOTA: A subconsulta copia os dados e a estrutura de colunas de outras tabelas mas NÃO copia as restrições sobre essas colunas nem os valores por omissão que essas colunas tinham. Ou seja, uma tabela criada com uma subconsulta não tem qualquer tipo de restrição de integridade. Nem de chave primária, forasteira ou `NOT NULL`s.

Exemplo 8.18 – Copiar os dados e a estrutura de colunas da tabela `emp`.

```
CREATE TABLE emp2 AS
   SELECT * FROM emp;
```

Exemplo 8.19 – Cria uma tabela, `emp2`, com alguns dos dados dos empregados que ganham mais em cada departamento. Note que no caso de se aplicarem funções sobre as colunas originais é forçoso o uso de pseudónimos.

```
CREATE TABLE emp2 AS
   SELECT nemp,
          UPPER(nome) nome,
          funcao,
          sal*14+NVL(premios, 0) sal_anual
   FROM emp
   WHERE (sal, ndep) IN (SELECT max(sal), ndep
                         FROM emp
                         GROUP BY ndep);
```

Veja como ficou a estrutura de `emp2`:

```
SQLWKS> DESC emp2;

Column Name                    Null?    Type
------------------------------ -------- ----
NEMP                                    NUMBER(4)
NOME                                    VARCHAR2(20)
FUNCAO                                  VARCHAR2(12)
SAL_ANUAL                               NUMBER
```

E veja o seu conteúdo:

```
SQLWKS> SELECT * FROM emp2;

NEMP       NOME                 FUNCAO       SAL_ANUAL
---------- -------------------- ------------ ----------
      1698 DUARTE GUEDES        Encarregado     5331900
      1788 MARIA DIAS           Analista        7910000
      1839 JORGE SAMPAIO        Presidente     12460000
3 rows selected.
```

Pode testar e verificar que `emp2` não tem qualquer tipo de restrição de integridade sobre as suas colunas. Insira por exemplo:

```
-- Insere com nemp repetido
INSERT INTO emp2 (1698, 'Joca', 'Programador', 1234567);

-- Insere com nemp a NULL
INSERT INTO emp2 (NULL, 'Joca', 'Programador', 1234567);

-- Insere com tudo a NULL
INSERT INTO emp2 (NULL, NULL, NULL, NULL);

-- Insere com tudo a NULL outra vez
INSERT INTO emp2 (NULL, NULL, NULL, NULL);
```

8.2. Alterar Tabelas (ALTER TABLE)

Depois de criar uma tabela é possível alterá-la. Embora não seja um comando usado frequentemente pode ser particularmente importante conseguir alterar a tabela sem ter que a destruir e voltar a criar novamente com outro comando. É particularmente importante se a tabela já contiver dados que não se podem perder.

As alterações possíveis são:

- Acrescentar colunas
- Redefinir colunas no que diz respeito ao tipo de dados, comprimento e valor por omissão
- Acrescentar e retirar restrições
- Activar e desactivar restrições

Repare que não é possível remover uma coluna. Note ainda que só é possível diminuir o tamanho dos dados de uma coluna se a tabela estiver vazia.

8.2.1. Acrescentar Colunas

Para acrescentar uma coluna a uma tabela já existente usa-se a seguinte sintaxe:

```
ALTER TABLE tabela
   ADD (coluna tipo_coluna [restricoes] [valor_omissao]);
```

Exemplo 8.20 – Acrescentar uma coluna a empregados. Definir restrições sobre a coluna bem como valores por omissão.

```
ALTER TABLE empregados
   ADD (sexo CHAR(1)
        DEFAULT 'F'
        CONSTRAINT ck_sexo_empregados CHECK (sexo IN ('M', 'F'))
        );
```

8.2.2. Alterar Colunas

Para alterar colunas (tipo, comprimento dos dados, valores por omissão) usa-se o comando:

```
ALTER TABLE tabela
   MODIFY (coluna [tipo_coluna] [valor_omissao]);
```

Note que não é possível com a cláusula `MODIFY coluna` alterar ou remover restrições sobre essas colunas. Para isso é preciso a cláusula `DROP restrição`. Os parêntesis são opcionais porque a cláusula `MODIFY` refere-se sempre a uma e uma só coluna.

Exemplo 8.21 – Alterar o tipo de dados

```
ALTER TABLE empregados
   MODIFY (sexo CHAR(3));
```

Exemplo 8.22 – Alterar o valor por omissão

```
ALTER TABLE empregados
   MODIFY (sexo DEFAULT 'Fem');
```

Note que este valor por omissão entra em conflito com a restrição de integridade, `ck_sexo_empregados`, definida no Exemplo 8.20.

8.2.3. Remover Restrições

Para remover uma restrição usa-se:

```
ALTER TABLE tabela
   DROP CONSTRAINT restrição;
```

Exemplo 8.23 – Remover a restrição da coluna `sexo`

```
ALTER TABLE empregados
   DROP CONSTRAINT ck_sexo_empregados;
```

8.2.4. Acrescentar Restrições

Para acrescentar uma restrição usa-se:

```
ALTER TABLE tabela
   ADD CONSTRAINT nome_restricao condicao [DISABLE];
```

Se se usar a palavra `DISABLE`, a restrição é acrescentada mas não é activada. Ou seja, existe mas ainda não está a funcionar.

Exemplo 8.24 – Acrescentar uma restrição à coluna `sexo`

```
ALTER TABLE empregados
   ADD CONSTRAINT ck_sexo_empregados CHECK (sexo IN ('Fem', 'Mas'));
```

Repare que a menos que a tabela esteja vazia este comando resulta no seguinte código de erro:

```
ORA-02293: cannot enable (UTILIZADOR1.CK_SEXO_EMPREGADOS) - check
constraint violated
```

Isso acontece porque os registos já inseridos na tabela têm o campo `sexo` com o valor `'F'` ou `'M'`.

Para poder acrescentar a restrição pretendida ou ter-se-ia que apagar todos os registos da tabela ou ter-se-ia que alterar os `'F'`s para `'Fem'`s e os `'M'`s para `'Mas'`s. Um comando possível seria:

```
UPDATE empregados
   SET sexo = DECODE(sexo, 'F', 'Fem', 'M', 'Mas', null);
```

Agora, garantidamente, já seria possível acrescentar a restrição de acordo com o Exemplo 8.23.

Notar que a sintaxe a usar para acrescentar restrições segue a sintaxe

8.2.5. Desactivar Restrições (DISABLE CONSTRAINT)

Pode desactivar-se restrições. Apesar de existir, temporariamente a restrição não será verificada. Um exemplo onde isso poderia ser usado é na Base de Dados de exemplo usada nas aulas práticas. Todos os empregados têm um encarregado. Isso obriga a que só depois de se inserir os encarregados é possível inserir os subordinados. Para evitar que se tenha que inserir os dados de exemplo da tabela numa ordem específica pode temporariamente desactivar a restrição. No fim da inserção basta activar de novo a restrição

A sintaxe para desactivar restrições é:

```
ALTER TABLE tabela
   DISABLE CONSTRAINT nome_condição;
```

Exemplo 8.25 – Desactiva a restrição `ck_sexo_empregados`

```
ALTER TABLE empregados
   DISABLE CONSTRAINT ck_sexo_empregados;
```

8.2.6. Activar Restrições

Da mesma forma que é possível desactivar restrições também é possível activá-las.

A sintaxe para activar restrições é:

```
ALTER TABLE tabela
   ENABLE CONSTRAINT nome_condição;
```

Exemplo 8.26 – Activa a restrição `ck_sexo_empregados`

```
ALTER TABLE empregados
   ENABLE CONSTRAINT ck_sexo_empregados;
```

8.3. Apagar Tabelas (DROP TABLE)

Para apagar tabelas usa-se o comando `DROP TABLE` seguido do nome da tabela.

Exemplo para apagar a tabela `emp`:

```
DROP TABLE emp;
```

Se existirem chave forasteiras, noutras tabelas, a apontar para algum dos registos da tabela a apgar, o comando devolve um erro:

```
SQLWKS> DROP TABLE dep
     2>
DROP TABLE dep
           *
ORA-02449: unique/primary keys in table referenced by foreign keys
```

Para forçar o apagamento da tabela, anulando as restrições de integridade necessárias, o comando a executar é:

```
DROP TABLE dep CASCADE CONSTRAINTS;
```

8.4. Criar Sequências (CREATE SEQUENCE)

Uma sequência é um objecto que permite gerar números inteiros únicos. Mesmo que vários utilizadores (ou o mesmo utilizador através de várias sessões) acedam à sequência, é garantido que nenhum obtém um número que outro já tenha obtido.

O principal uso de sequências é para a geração automática de chaves primárias.

O comando para criar sequências é:

```
CREATE SEQUENCE nome_sequência
   START WITH num_principio
   INCREMENT BY num_intervalo
   [MAXVALUE num_max | NOMAXVALUE]
   [MINVALUE num_min | NOMINVALUE]
   [CYCLE | NOCYCLE]
   [ORDER | NOORDER]
   [CACHE num_cache | NOCACHE]
```

Onde:

- **`nome_sequência`** é o nome da sequência a criar
- **`START WITH num_principio`** é o primeiro número da sequência e pode ser qualquer valor inteiro, positivo, negativo ou zero.
- **`INCREMENT BY num_intervalo`** específica quantos inteiros separam dois números obtidos consecutivamente da sequência. Pode ser qualquer valor positivo ou negativo mas não pode ser zero.
- **`MAXVALUE num_max`** limita os números da sequência a este valor máximo. Pode ser qualquer número desde que seja maior ou igual a `num_princípio` e maior que `num_min`.
- **`NOMAXVALUE`** especifica um valor máximo de 10^{27} no caso de uma sequência ascendente e –1 no caso de uma sequência descendente. Por omissão as sequências são `NOMAXVALUE`.
- **`MINVALUE num_min`** limita os números da sequência a este valor mínimo. Pode ser qualquer número desde que seja menor ou igual a `num_princípio` e menor que `num_max`.
- **`NOMINVALUE`** especifica um valor mínimo de -10^{27} no caso de uma sequência ascendente e 1 no caso de uma sequência descendente. Por omissão as sequências são `NOMINVALUE`.
- **`CYCLE`** determina que uma sequência ascendente depois de atingir o valor máximo recomeça a partir do valor mínimo. No caso de uma sequência descendente, depois de atingido o valor mínimo a contagem recomeça no valor máximo.
- **`NOCYCLE`** é o valor por omissão e determina que depois de atingido qualquer um dos limites a sequência não gera mais valores.

- **ORDER** determina que os números da sequência são gerados pela ordem de pedidos. Os números só poderiam ser gerados fora de ordem se se estivesse a usar um servidor de BD paralelo. Se esta opção for indica, os números da sequência nunca são gerados antecipadamente e as opções CACHE e NOCACHE são irrelevantes.
- **NOORDER** é o valor por omissão e indica que não é forçoso que os números da sequência sejam gerados pela ordem dos pedidos. No que diz respeito à geração de chaves primárias, não costuma ser importante a ordem dos números.
- **CACHE num_cache** indica quantos números o Oracle gera em avanço para aumentar a rapidez. O valor por omissão é CACHE 20. É preciso garantir que o valor de num_cache é menor ou igual ao número total de valores que a sequência pode gerar.
- **NOCACHE** determina que nunca são gerados números em avanço

Para usar as sequências basta fazer:

nome_seq.nextval para obter o próximo valor (também funciona para o primeiro)

nome_seq.currval para devolver o valor actual (não funciona a primeira vez)

O uso destas pseudo-colunas funciona como se tratasse de funções. Ou seja, pode-se incluir qualquer destas formas num comando SELECT, INSERT, UPDATE, etc. mas não se pode executar directamente na linha de comando. Pode-se no entanto fazer uso da tabela especial DUAL.

Exemplo 8.27 – Usar uma sequência várias vezes consecutivas. Note que a primeira vez que se usa a sequência não se pode usar a forma currval (porque ainda não existe um valor corrente).

```
SQLWKS> SELECT minha_sequencia.currval FROM dual;
ORA-08002: sequence MINHA_SEQUENCIA.CURRVAL is not yet defined in this session

SQLWKS> SELECT minha_sequencia.nextval FROM dual;
NEXTVAL
----------
         1
1 row selected.

SQLWKS> SELECT minha_sequencia.nextval FROM dual;
NEXTVAL
----------
         2
1 row selected.

SQLWKS> SELECT minha_sequencia.nextval FROM dual;
NEXTVAL
----------
         3
1 row selected.

SQLWKS> SELECT minha_sequencia.currval FROM dual;
CURRVAL
----------
         3
1 row selected.
```

8.5. Apagar Sequências (DROP SEQUENCE)

Para apagar sequências usa-se o comando DROP SEQUENCE seguido do nome da sequência. Exemplo:

```
DROP SEQUENCE minha_sequencia;
```

Exemplo 8.28 – Usar uma sequência num INSERT.

```
SQLWKS> INSERT INTO emp (nemp, nome, sal, ndep)
     2>    VALUES (minha_sequencia.nextval, 'Joca', 55000, 10);
1 row processed.

SQLWKS> INSERT INTO emp (nemp, nome, sal, ndep)
     2>    VALUES (minha_sequencia.nextval, 'Becas', 55000, 10);
1 row processed.

SQLWKS> INSERT INTO emp (nemp, nome, sal, ndep)
     2>    VALUES (minha_sequencia.nextval, 'Egas', 55000, 10);
1 row processed.

SQLWKS> SELECT nemp, nome FROM emp
     2>    WHERE nemp < 1000;
NEMP       NOME
---------- --------------------
         4 Joca
         5 Becas
         6 Egas
3 rows selected.
```

Exemplo 8.29 – Criar uma sequência de 0 a 100, com intervalos de 5 em 5. Quando chega ao fim volta ao princípio.

```
CREATE SEQUENCE seq
   START WITH 0
   INCREMENT BY 5
   MAXVALUE 100
   MINVALUE 0
   CYCLE;
```

Note que por omissão `MINVALUE` é 1 (ou seja, `NOMINVALUE` para sequências ascendentes). Se no comando anterior não se especificasse o valor de `MINVALUE` seria gerado um erro porque nesse caso `START WITH` seria menor que `MINVALUE`.

Exemplo 8.30 – Criar uma sequência de 1 a 10E27, com intervalos de 1 em 1. Quando chega ao fim volta ao princípio.

```
CREATE SEQUENCE seq;
```

Exemplo 8.31 – Faz o mesmo que o anterior mas a sequência não volta ao princípio

```
CREATE SEQUENCE seq
   NOCYCLE;
```

Exemplo 8.32 – Criar uma sequência descendente de 30 a 10. Começa em 20 e quando chega ao mínimo volta ao valor máximo.

```
CREATE SEQUENCE seq
   START WITH 20
   INCREMENT BY -2
   MAXVALUE 30
   MINVALUE 10
   CYCLE
   NOCACHE
```

NOTA: Note que o valor por omissão é `CACHE 20`. Note também que a sequência ao todo tem apenas 10 números possíveis. Assim, se não se especificar a opção `NOCACHE` (ou alternativamente `CACHE` com um valor inferior ou igual a 10) o comando gera um erro.

8.6. Exercícios

Considere a base de dados de demonstração fornecida.

1. Cria uma tabela, **`projectos`**, com as colunas, **`nproj`** (`number(7)`), **`nome`** (`varchar2(20)`), e `descricao` (`varchar2(60)`). A chave primária deve ser `nproj`. A coluna `nome` deve ser `NOT NULL` e `UNIQUE`. A coluna `descricao` deve ser `NOT NULL`.

Resposta:

2. Cria uma sequência, `seq_nproj` para usar nos valores de `nproj`. A sequência deve começar em 500 e avançar de 10 em 10. Não deverá voltar ao princípio quando terminar e não deve estar limitada a nenhum valor máximo.

Resposta:

3. Insira vários registos na tabela `projectos` (usando a sequência `seq_nproj`) . Insira por exemplo os projectos:

Nome	**Descrição**
'Sificap'	'BD para controlo das actividades das pescas'
'IPQ'	'Instituto Português da Qualidade'
'Apolo11'	'This is a small step for man, but a giant leap for Mankind'

Resposta:

4. Faça uma cópia da tabela `emp` para uma nova tabela `emp2`. Uso o comando `CREATE TABLE` com subconsulta. Na cópia de valores transforme os nomes dos empregados para maiúsculas e garanta que todos os empregados ganhem mais do que 70000$. Lembre-se da resolução do exercício 4 da aula anterior. Relembre o uso das funções `UPPER` e `GREATEST` (se não se lembrar veja no manual).

Note que como está a produzir alterações sobre algumas colunas irá ter que usar pseudónimos para a tabela `emp2` possuir colunas com nomes válidos.

Resposta:

5. Como ao criar a nova tabela `emp2` as restrições não são copiadas altere a tabela de modo a incluir as restrições que existiam na tabela `emp2` na aula 7 (veja em http://www.dei.uc.pt/~bizarro/aulas/bd1/2000/aula07/criaEmp2.sql). Use o comando `ALTER TABLE`.

Resposta:

6. Cria uma sequência `emp2_number` que comece em 1 e aumente de 10 em 10 e que depois de chegar ao fim não recomece do princípio.

Resposta:

7. Insira em `emp2` os empregados com as seguintes características:

```
nome = 'CHICO FININHO',
funcao = 'Cantor',
encar = null,
data_entrada = hoje, (use sysdate)
sal = 100000,
premios = null
ndep = 40.

nome = 'CHICO FINÃO',
funcao = 'Cantor Pop',
encar = null,
data_entrada = hoje, (use sysdate)
sal = 200000,
premios = 100
ndep = 30.

nome = 'CHICO FINISSIMO',
funcao = 'Cantor Rock',
encar = null,
data_entrada = hoje, (use sysdate)
sal = 400000,
premios = 200
ndep = 10.
```

Repare com os números de empregado foram introduzidos.

Resposta:

8. Acrescente uma restrição a `emp2` que garanta que o valor do prémio é sempre inferior ao valor do `sal` para cada empregado. Teste se a restrição está a funcionar.

Resposta:

9. Acrescente uma coluna, `nproj`, do tipo `number(7)` a `emp2`.

9a. Garanta que `nproj` é `NOT NULL`. Acha que é possível fazer isso? Porquê? Ou como?

Resposta:

10. Garanta então que `nproj` (em `emp2`) é `NOT NULL` mas mantenha a restrição no modo `DISABLE`. Atribua o nome `nn_nproj_emp2` à restrição.

Resposta:

11. Modifique a coluna `nproj` da tabela `emp2` de modo a que passa a ser uma chave forasteira que se refira à coluna do mesmo nome na tabela `projectos`. Repare que pode declarar a coluna `nproj` como sendo uma chave forasteira mesmo com esta a conter valores nulos.

Resposta:

12. Altere os valores de `nproj` na tabela `emp2` de modo a colocar os empregados a trabalhar em projectos.

Resposta:

13. Active (`ENABLE`) a restrição `nn_nproj_emp2` que tinha sido criada no exercício 10 mas que estava desactivada (`DISABLE`).

Resposta:

Aula 9

Exercícios de Revisão

Vão ser resolvidos exercícios de frequências e exames anteriores, ou das aulas práticas, ou ainda outros exercícios sugeridos pelos alunos.

Anexo A

Desafios – Só Para Arrojados!

*"***desafio**, s. m.
.acto ou efeito de desafiar ou chamar alguem para combate;
.provocacao;
.incitacao;
.despique entre dois cantadores;
.duelo;
.luta;
.competicao desportiva;
in Dicionario da Lingua Portuguesa v1.0, Porto Editora, 1996

Pondo de lado o "despique entre dois cantadores", todas as outras definicoes se ajustam. O objectivo deste e dos proximos desafios a Bases de Dados 1 e 2 e' o de incentivar-vos a resolver problemas relativamente complicados (and we mean it!).

Pelo caminho, vao ser obrigados a puxar da vossa capacidade de "problem solving" e irao, se tudo correr bem, ganhar muita fluencia em SQL. E isto traduz-se em confianca acrescentada a resolver problemas do dia-a-dia que vao desde as frequencias e exames aos trabalhos semestrais. E pode ser que se note quando voces chegarem ao mercado de trabalho.

Fica aqui lancado o primeiro dos desafios a BD para o ano lectivo de 1998/99. O aluno vencedor sera' aquele que fizer chegar às nossas maos um comando SQL que produza, sem batota, o resultado correcto. Se existir mais do que uma solucao, serao considerados vencedores os alunos que enviarem a primeira solucao de cada tipo.

Mandem resultados apenas por mail ou por disquete.

Lembrem-se, se nao ganharem este desafio tentem nos proximos...

Vale tudo menos prejudicar o adversario!

Boa sorte,

Pedro Bizarro & Henrique Madeira"

Mail enviado para a lista bd@dei.uc.pt para motivar os alunos a participar, disponível em http://www.dei.uc.pt/majordomo/bd/

A.1. Centrar Strings

```
       -------------------------------------------------
       --  ENUNCIADO DO 1o DESAFIO DE BASES DE DADOS 1  --
       --                Centrar Strings                --
       -------------------------------------------------
Codifique  um comando SELECT que seleccione  apenas o nome proprio
de cada  empregado  (retirando,  portanto, o  apelido). Esse  nome
devera' aparecer  centrado  numa  coluna de 20 caracteres  de nome
"Centrar Strings".

'A esquerda e 'a direita do nome proprio devera' ser  acrescentado
um caracter espaco (' ').

Os restantes caracteres deverao ser preenchidos da seguinte forma:
 - 'a esquerda do nome (e do espaco esquerdo) devera' aparecer uma
   sequencia de  numeros a comecar em  1 e a acabar no valor maior
   possivel.
 - 'a direit a do nome (e do espaco direito) devera'  aparecer uma
   sequencia de numeros a comecar no valor maior  valor possivel e
   a acabar em 1.

Caso  nao seja possivel  centrar exactamente  o nome, este devera'
aparecer deslocado um caracter para a esquerda.

O resultado do  comando correcto sobre a tabela emp, assumindo que
existe uma clausula ORDER BY nome devera' ser o seguinte:

Centrar Strings
--------------------
1234567 Ana 87654321
12345 Antonio 654321
12345 Augusto 654321
12345 Catarina 54321
123456 Duarte 654321
123456 Joana 7654321
123456 Jorge 7654321
123456 Manuel 654321
123456 Maria 7654321
123456 Nelson 654321
1234567 Olga 7654321
1234567 Rita 7654321
123456 Silvia 654321
1234567 Tome 7654321

14 rows selected.

Veja mais pormenores nas seccoes seguintes.
```

```
       -------------------------------------------------
       -- PROCEDIMENTOS PARA CRIAR UM AMBIENTE DE TESTE --
       -------------------------------------------------
-----------------------------------------------------------------
-- 1) Se ja' existir uma tabela de teste, apaga-a.             --
-----------------------------------------------------------------
drop table e;        -- Nao se engane e nao apague a tabela emp!!!

-----------------------------------------------------------------
-- 2) Cria uma tabela igual 'a emp.                            --
-----------------------------------------------------------------
create table e as
select * from emp;

-----------------------------------------------------------------
-- 3) Apaga o conteudo dessa tabela.                           --
-----------------------------------------------------------------
delete from e;     -- Nao se engane e nao apague da tabela emp!!!

-----------------------------------------------------------------
-- 4) Insere nomes (com apelido) nessa tabela.                 --
-- Os nomes teem varios tamanhos para mais facilmente se poder --
-- testar se o algoritmo funciona ou nao.                      --
-- Se funcionar para esta tabela tambem funciona para a emp.   --
-----------------------------------------------------------------
insert into e values(1,  'a a', null, null, null, null, null, null);
insert into e values(2,  'ab a', null, null, null, null, null, null);
insert into e values(3,  'abc a', null, null, null, null, null, null);
insert into e values(4,  'abcd a', null, null, null, null, null, null);
insert into e values(5,  'abcde a', null, null, null, null, null, null);
insert into e values(6,  'abcdef a', null, null, null, null, null, null);
insert into e values(7,  'abcdefg a', null, null, null, null, null, null);
insert into e values(8,  'abcdefgh a', null, null, null, null, null, null);
insert into e values(9,  'abcdefghi a', null, null, null, null, null, null);
insert into e values(10, 'abcdefghij a', null, null, null, null, null, null);
insert into e values(11, 'abcdefghijk a', null, null, null, null, null, null);
insert into e values(12, 'abcdefghijkl a', null, null, null, null, null, null);
insert into e values(13, 'abcdefghijklm a', null, null, null, null, null, null);
insert into e values(14, 'abcdefghijklmn a', null, null, null, null, null, null);

-----------------------------------------------------------------
-- 5) Confirma as alteracoes.                                  --
-----------------------------------------------------------------
commit;
```

```
       ------------------------------------------------
       -- TESTAR O COMANDO SELECT NA TABELA TEMPORARIA  --
       ------------------------------------------------
-----------------------------------------------------------------
-- 1) Se dermos 'a coluna o pseudonimo ou alias de             --
-- "12345678900987654321" (para contar melhor os caracteres)   --
-- e se usarmos um ORDER BY nome, entao o resultado do comando --
-- SELECT correcto sobre a tabela e devera' dar o seguinte:    --
-----------------------------------------------------------------

12345678900987654321
--------------------
12345678 a 987654321
12345678 ab 87654321
1234567 abc 87654321
1234567 abcd 7654321
123456 abcde 7654321
123456 abcdef 654321
12345 abcdefg 654321
12345 abcdefgh 54321
1234 abcdefghi 54321
1234 abcdefghij 4321
123 abcdefghijk 4321
123 abcdefghijkl 321
12 abcdefghijklm 321
12 abcdefghijklmn 21

14 rows selected.


-----------------------------------------------------------------
-- 2) Se dermos 'a coluna o pseudonimo ou alias de             --
-- "Centrar Strings" e se usarmos um ORDER BY nome, entao o    --
-- resultado do comando SELECT correcto sobre a tabela emp     --
-- devera' dar o seguinte:                                     --
-----------------------------------------------------------------

Centrar Strings
--------------------
1234567 Ana 87654321
12345 Antonio 654321
12345 Augusto 654321
12345 Catarina 54321
123456 Duarte 654321
123456 Joana 7654321
123456 Jorge 7654321
123456 Manuel 654321
123456 Maria 7654321
123456 Nelson 654321
1234567 Olga 7654321
1234567 Rita 7654321
123456 Silvia 654321
1234567 Tome 7654321

14 rows selected.
```

A.2. Os 3 Sálarios Maximos

```
-------------------------------------------------
--  ENUNCIADO DO 2o DESAFIO DE BASES DE DADOS 1  --
--            Os 3 salarios maximos              --
-------------------------------------------------
```

Codifique um comando SQL que, dada a tabela EMP usada nas aulas praticas, devolva os empregados que correspondem aos 3 salarios de valor mais elevado. Ignore os valores dos premios.

Ignore ainda a possibilidade de existir mais do que uma pessoa com o mesmo salario.

Se conseguir fazer, tente arranjar um algoritmo generico o suficiente de modo a permitir que exista mais do que um empregado por valor de salario.

NOTA: Uma vez que este tipo de queries parece bastante comum seria logico pensar que o SQL resolvesse facilmente este tipo de situacoes. Na realidade nao resolve facil ou elegantemente. No segundo semestre iremos ver o que um ciclo FOR e uma coisa chamada CURSOR podem fazer por nos...

Anexo B

Comandos SQL mais comuns

B.1. SELECT

```
SELECT * FROM tabela;

SELECT sysdate FROM dual;

SELECT user FROM dual;

SELECT * FROM user_objects;

SELECT view_name FROM all_views;

SELECT *
  FROM emp
 WHERE ndep = 10
   AND funcao = 'Vendedor'
 ORDER BY sal ASC, nome ASC;

SELECT nome, sal*14 + NVL(premios, 0) "Remuneracao Annual"
  FROM emp;

SELECT funcao, max(sal) "Max Sal", count(*) "Quantidade"
  FROM emp
 GROUP BY funcao
HAVING count(*) >= 2;

SELECT e.nome, d.nome, escalao
  FROM emp e, dep d, descontos
 WHERE e.ndep = d.ndep
   AND sal BETWEEN salinf AND salsup;

SELECT *
  FROM emp
 WHERE nome LIKE 'A% R___';

SELECT *
  FROM emp
 WHERE nome LIKE 'A% R#_%' ESCAPE '#';

SELECT *
  FROM emp
 WHERE sal = (SELECT max(sal) FROM emp);

SELECT *
  FROM emp
 WHERE (funcao, sal) IN (SELECT funcao, max(sal)
                           FROM emp
                          GROUP BY funcao)
 ORDER BY funcao ASC;

SELECT DISTINCT funcao
  FROM emp;

SELECT DISTINCT funcao, d.nome
  FROM emp e, dep d
 WHERE e.ndep = d.ndep;




SELECT count(*), max(sal), min(sal), avg(sal), d.nome
```

```
  FROM emp e, dep.d
 WHERE e.ndep = d.ndep
 GROUP BY d.ndep
 ORDER BY d.nome;

SELECT nome || ' trabalha no dep ' || ndep "Descrição"
  FROM emp;

SELECT nemp, nome, sal, ndep
  FROM emp e
 WHERE sal > (SELECT avg(sal)
                FROM emp
               WHERE ndep = e.ndep)
 ORDER BY ndep ASC, sal DESC;

SELECT *
  FROM dep d
 WHERE NOT EXISTS (SELECT *
                     FROM emp
                    WHERE ndep = d.nep);

SELECT e.nome, e.sal, depmax.maxsal, depmax.ndep
  FROM emp e,
       (SELECT max(sal) maxsal, ndep
          FROM emp
         GROUP BY ndep) depmax
 WHERE e.ndep = depmax.ndep
 ORDER BY e.ndep, e.sal DESC;
```

B.2. INSERT, UPDATE, DELETE

```
INSERT INTO emp (nemp, nome, funcao, encar,
                 data_entrada, sal, premios, ndep)
       VALUES (1234, 'Joca', 'Aluno', 1566,
                sysdate, 100000, null, 10);

INSERT INTO emp
       VALUES (1234, 'Joca', 'Aluno', 1566,
                sysdate, 100000, null, 10);

INSERT INTO emp (nemp, nome, funcao, encar,
                 data_entrada, sal, ndep)
       VALUES (seq_emp.nextval, 'Joca', 'Aluno', 1566,
                sysdate, 100000, 10);

INSERT INTO emp2
       SELECT * FROM emp;



UPDATE emp
   SET sal = 200000
 WHERE nemp = 1369;

UPDATE emp
   SET sal = sal * 1.1;

UPDATE emp
   SET sal = 200000,
       funcao = 'Vendedor'
 WHERE ndep = 20
   AND sal < 150000
   AND ndep IN (SELECT ndep
                  FROM dep
                 WHERE local = 'Mealhada'
                    OR local = 'Coimbra');



DELETE FROM emp;

DELETE FROM emp
 WHERE ndep = 10;
```

B.3. CREATE, DROP, ALTER

```
CREATE TABLE emp
(nemp           NUMBER(4)       CONSTRAINT pk_nemp_emp       PRIMARY KEY,
 nome           VARCHAR2(20)    CONSTRAINT nn_nome_emp       NOT NULL
                                CONSTRAINT upper_nome_emp
                                  CHECK (nome = UPPER(nome))           ,
 funcao         VARCHAR2(12)    CONSTRAINT nn_funcao_emp     NOT NULL   ,
 encar          NUMBER          CONSTRAINT fk_encar_emp
                                   REFERENCES emp(nemp)      NULL       ,
 data_entrada   DATE            DEFAULT SYSDATE
                                CONSTRAINT nn_data_emp       NOT NULL   ,
 sal            NUMBER(7)       CONSTRAINT ck_sal_emp
                                   CHECK (sal > 70000)
                                CONSTRAINT nn_sal_emp        NOT NULL   ,
 premios        NUMBER(7)       DEFAULT NULL                            ,
 ndep           NUMBER(2)       CONSTRAINT nn_ndep_emp       NOT NULL
                                CONSTRAINT fk_ndep_emp
                                   REFERENCES dep(ndep)
);


DROP TABLE emp;

DROP TABLE dep CASCADE CONSTRAINTS;


ALTER TABLE emp
   ADD (sexo CHAR(1)
             DEFAULT 'F'
             CONSTRAINT ck_sexo_emp CHECK (sexo IN ('F', 'M'));

ALTER TABLE emp
   MODIFY (sexo CHAR(3));

ALTER TABLE emp
   MODIFY (sexo DEFAULT 'Fem'));

ALTER TABLE emp
   DROP CONSTRAINT ck_sexo_emp;

ALTER TABLE emp
   ADD CONSTRAINT ck_sexo_emp CHECK (sexo IN ('Fem', 'Mas'));

ALTER TABLE emp
   DISABLE CONSTRAINT ck_sexo_emp;

ALTER TABLE emp
   ENABLE CONSTRAINT ck_sexo_emp;


CREATE SEQUENCE seq_emp;

CREATE SEQUENCE seq_emp
   START WITH 1000
   INCREMENT BY 10
   MAXVALUE 2000
   NOCYCLE;

DROP SEQUENCE seq_emp;
```

Anexo C

Dicionário de Dados

C.1. Dicionário de Dados

O dicionário de dados é um conjunto de informação sobre os objectos existentes na base de dados. São dados sobre os dados, ou seja, são meta-dados. Toda esta informação está guardada em tabelas dos sistema. Nenhum utilizador normal tem acesso a estas tabelas. No entanto, existem uma série de vistas construídas sobre as tabelas do dicionário de dados que os utilizadores podem usar em modo de leitura.

Os motivos porque se permite que os utilizadores apenas acedam às vistas e não directamente às tabelas são:

- Protecção extra,
- Portabilidade, pois assim podem alterar-se as tabelas do dicionário de dados entre versões do servidor mas manter o interface com o utilizador constante.

Para ver todas as vistas acessíveis pelo utilizador:

```
SELECT view_name FROM ALL_VIEWS
 ORDER BY 1;
```

Existem 3 grandes grupos de vistas que dependem do seu prefixo: `DBA_`, `ALL_` e `USER_`. As vistas do tipo `DBA_` contêm informação relativa ao Administrador de Bases de Dados (o DBA). As vistas do tipo `ALL_` contêm informação relativa a todos os objectos que o utilizador actual consegue ver. As vistas do tipo `USER_` contêm informação relativa a todos os objectos que pertencem ao utilizador actual. As vistas do tipo `ALL_` têm uma coluna a mais do que as vistas `USER_`. Essa coluna é normalmente a `owner` e indica a quem pertence o objecto descrito.

Repare que diferentes utilizadores podem receber diferentes resultados ao `SELECT` anterior em função das suas permissões.

Como o utilizador normal nunca tem acesso às vistas do DBA, as que realmente interessam são as seguintes:

Vistas ALL_	Vistas USER_
ALL_CATALOG	USER_CATALOG
ALL_CONSTRAINTS	USER_CONSTRAINTS
ALL_CONS_COLUMNS	USER_CONS_COLUMNS
ALL_ERRORS	USER_ERRORS
ALL_EXTENTS	USER_EXTENTS
ALL_FREE_SPACE	USER_FREE_SPACE
ALL_INDEXES	USER_INDEXES
ALL_IND_COLUMNS	USER_IND_COLUMNS
ALL_OBJECTS	USER_OBJECTS
ALL_OBJECT_SIZE	USER_OBJECT_SIZE
ALL_SEGMENTS	USER_SEGMENTS
ALL_SEQUENCES	USER_SEQUENCES
ALL_SNAPSHOTS	USER_SNAPSHOTS
ALL_SOURCE	USER_SOURCE
ALL_SYNONYMS	USER_SYNONYMS
ALL_TABLES	USER_TABLES
ALL_TABLESPACES	USER_TABLESPACES
ALL_TAB_COLUMNS	USER_TAB_COLUMNS
ALL_TAB_COMMENTS	USER_TAB_COMMENTS
ALL_TRIGGERS	USER_TRIGGERS
ALL_VIEWS	USER_VIEWS

Anexo D

Provas Resolvidas

D.1. Frequência de 1999.01.11

Frequência de Bases de Dados I
11 de Janeiro de 1999
Correcção e Comentários da Parte Prática

por Pedro Bizarro

1. Considere a Bases de Dados usada nas aulas práticas

a) Quantas colunas e quantas linhas de dados produz o seguinte comando:

```
SELECT *
   FROM emp, dep, descontos;
```

Sabendo que as tabelas têm as seguintes características:

	Linhas	Colunas
emp	14	8
dep	4	3
descontos	5	3

Resposta: O resultado é um produto de tabelas: obtém-se todas as combinações possíveis de linhas de uma tabela com todas as outras linhas da(s) outra(s) tabela(s) sem qualquer critério de junção. Assim, tem-se um total de 14*4*5=280 linhas e 8+3+3=14 colunas.

Mais informações: Aula 1 e Aula 2 da sebenta das práticas em geral e ver Produto na página 3 e Junções na página 18 e seguintes em particular.

Erros mais frequentes:
- Somar linhas em vez de multiplicar.
- Em vez de somar colunas, fazer max(8, 3, 3) = 8.
- Somar colunas e subtrair colunas repetidas (8+3+3-1=13).
- Erros nos cálculos. Sem desconto.

Comentários: Numa base de dados relacional, saber como relacionar tabelas (e saber como não relacionar tabelas - uma junção sem claúsula `WHERE`) faz parte daquilo que é mais básico na manipulação de dados. Não saber responder à pergunta implica a não compreensão do, provavelmente mais importante e omnipresente, conceito.

b) Mostre toda a informação possível dos empregados que satisfaçam as seguintes condições:

- seu departamento é o da 'Contabilidade' ou o de 'Coimbra'.
- seu escalão seja o 1,3 ou 5.
- seu nome tenha pelo menos um 'a' (minúsculo).
- A sua função não seja 'Presidente'.

Tenha especial atenção com o uso de parêntesis, dos ANDs e dos ORs para definir as restrições.

Resposta:

```
SELECT *
   FROM emp e, dep d, descontos de
   WHERE  e.ndep = d.ndep
      AND e.sal BETWEEN de.salinf AND de.salsup
      AND (d.nome = 'Contabilidade' OR d.local = 'Coimbra')
      AND (de.escalao = 1 OR de.escalao = 3 OR de.escalao = 5)
      AND e.nome LIKE '%a%'
      AND e.funcao != 'Presidente';
```

Mais informações: Aula 2 da sebenta das práticas em geral e ver os exercícios dessa aula com especial destaque para o exercício 8 na página 30. Ver Cláusula WHERE na página 9 e seguintes.

Erros mais frequentes:

- Uso de sub-consultas. Sem desconto se o comando ainda assim produzisse o resultado correcto.
- Omissão das condições de junção entre as tabelas - primeiras duas linhas da cláusula WHERE na resposta proposta.
- Responder com 4 comandos separados em vez de um único comando para todas as restrições pedidas. Sem desconto se todos comandos estivessem correctos – o português da frequência não estava claro o suficiente mas não era o que era pedido.
- Uso de colunas erradas para fazer as restrições. Por exemplo: …AND (d.nome = 'Contabilidade' OR d.**nome** = 'Coimbra')…
- Uso de colunas certas mas das tabelas erradas. Por exemplo: …AND (d.nome = 'Contabilidade' OR **emp.local** = 'Coimbra')…
- Falta de uma tabela na cláusula FROM e uso de colunas dessa tabela na cláusula WHERE.
- Uso de condições irrelevantes. Por exemplo:
  …AND (e.nome LIKE '%a%' **or e.nome LIKE 'a%' or e.nome LIKE '%a'**)…
  Sem desconto se continuasse ainda a produzir o resultado correcto.
- Considerar que era para restringir os dados aos escalões "1,3" e "5".
  …AND (de.escalao **= 1,3** OR de.escalao = 5)…

  Sem desconto – o português da frequência não estava claro o suficiente mas dever-se-ia compreender facilmente que uma coluna de valores inteiros não poderia nunca ter departamentos de número 1,3… ☺!

Comentários: Esta pergunta determina até que ponto os alunos estão preparados para construir comandos em SQL. Embora não seja uma pergunta complexa, bem pelo contrário, é exaustiva. Envolve equi-junções (pag 19 da sebenta), não equi-junções (pag 20 da sebenta), uso de pseudónimos de colunas, restrições de valores numéricos, restrições de cadeias de caracteres e uso de LIKE e *wildcards*. No entanto para a resposta são usados apenas os comandos mais simples e as primeiras duas aulas práticas deveriam chegar: não se usa valores nulos, funções de grupo, funções de data, funções de strings, junções complexas, operadores de conjuntos, sub-consultas, sub-consultas correlacionadas, etc. Quem

respondesse correctamente demonstrava segurança na matéria, saberia distinguir o trigo do joio e conseguiria não ligar o *complicometro*[*].

c) Considere a base de dados usada nas aulas práticas. Codifique o comando que encontra o empregado com o salário mais baixo do departamento `'Contabilidade'`. Para esse empregado deverá mostrar os campos `nemp`, `nome`, `funcao` e `sal`. Repare que não pode partir do princípio que sabe o número desse departamento.

Respostas:

- de Pedro Bizarro (adaptada da página 54 da sebenta)

```
SELECT nemp, nome, funcao, sal
   FROM emp
   WHERE  (sal, ndep) IN (SELECT min(sal), ndep
                          FROM emp
                          GROUP by ndep
                          HAVING ndep = (SELECT ndep
                                         FROM dep
                                         WHERE nome = 'Contabilidade'));
```

- de Domitilia Nora

```
SELECT nemp, e.nome, funcao, sal
   FROM emp e,
        (SELECT e1.ndep, min(sal) salmin
            FROM emp e1, dep d
            WHERE e1.ndep = d.ndep
               AND d.nome = 'Contabilidade'
            GROUP BY e1.ndep) m
   WHERE e.ndep = m.ndep
      AND e.sal = m.salmin;
```

- de Nuno José Duarte

```
SELECT e.nemp, e.nome, e.funcao, e.sal
   FROM emp e, dep d
   WHERE (d.nome = 'Contabilidade')
      AND (e.ndep = d.ndep)
      AND sal <= ALL (SELECT sal
                      FROM emp e, dep d
                      WHERE d.nome = 'Contabilidade'
                         AND e.ndep = d.ndep);
```

- de Rui Filipe Silva

```
SELECT e.nemp, e.nome, e.funcao, e.sal
   FROM emp e, dep d
   WHERE (d.nome = 'Contabilidade')
      AND (e.ndep = d.ndep)
      AND sal =  (SELECT min(sal)
                  FROM emp e, dep d
                  WHERE d.nome = 'Contabilidade'
                     AND e.ndep = d.ndep);
```

Mais informações: Ver Funções de Grupo na Aula 3 da sebenta (pag. 38 e segs.) e Sub-Consultas na Aula 4 da sebenta (pag. 54 e segs.). Ver em particular o último exemplo da página 54; note como esse exemplo representa quase a resposta certa; falta apenas acrescentar a sub-consulta para limitar os departamentos ao de `'Contabilidade'`.

Erros mais frequentes:

- Usar colunas que não são funções de grupo num comando com funções de grupo. Por exemplo:
  `SELECT` **`nome`**`, max(sal)`

[*] *Complicómetro* - instrumento de perturbação de desempenho observado frequentemente em situações de pressão. Afecta sobretudo os estudantes de Engenharia Informática menos seguros na matéria e os resultados mais visíveis são a escrita de 10 vezes mais linhas de código que o necessário. Muitas vezes, a correcção de um elevado número de casos destes numa só prova leva o docente a questionar o sentido da vida, a metafísica, a existência de Deus e a qualidade do barman.

```
   FROM emp;
```
ou
```
SELECT funcao, max(sal)
   FROM emp
   GROUP BY ndep;
```
- Na sub-consulta usar apenas uma coluna quando se deveria usar duas. Exemplo:
  `…WHERE (sal) IN (SELECT min(sal)…`
  em vez de
  `…WHERE (sal, e.ndep) IN (SELECT min(sal), d.ndep…`
- Trocar restrições de `HAVING` com restrições de `WHERE`.
- Não usar uma junção quando era necessário.
- Não usar um `GROUP BY` quando era necessário.
- Falta de uma tabela na cláusula `FROM`.

Comentários: Esta era a pergunta mais difícil da parte prática. Mas também era a aquela onde era mais provável aparecerem respostas novas, originais ou inteligentes. De facto isso veio mesmo a acontecer. Especial destaque para os alunos seguintes que apresentaram todos respostas correctas (ou quase) ou raciocínios correctos usando um algorítmo diferente daquele sugerido na página 54 da sebenta:

Originais sem estarem inteiramente correctas:
- Nuno Miguel Silva
- Onésimo Duarte Pinto

Originais e correctas:
- António Ribeiro
- João Duque
- João Ramos
- Luís Pedro Simões

Muito originais e correctas
- **Domitília Nora**
- **Nuno José Duarte**
- **Rui Filipe Silva**

2. Crie uma tabela, pessoas, de colunas ID, nome, BI, data_nascimento e cod_morada.

O ID é um número de 10 algarismos, o nome uma cadeia de caracteres de até 100 caracteres, o BI deve ser representado por um número de 10 algarismos e o cod_morada tem 8 dígitos.

O ID é a chave primária da tabela. O campo nome tem que ser sempre obrigatoriamente preenchido assim como o campo BI e a data de nascimento. O BI é único. Por omissão, o valor da da de nascimento deverá ser a data actual com as horas, minutos e segundos a zero. Cod_morada é uma chave forasteira que aponta para uma coluna do mesmo nome na tabela moradas.

d) Assuma que a tabela moradas já existe e codifique o comando que cria a tabela pessoas.

Resposta:

```
CREATE TABLE pessoas
(id               NUMBER(10)     CONSTRAINT pk_pessoas_id PRIMARY KEY,
 nome             VARCHAR2(100)  CONSTRAINT nn_pessoas_nome NOT NULL,
 bi               NUMBER(10)     CONSTRAINT uni_pessoas_bi UNIQUE
                                 CONSTRAINT nn_pessoas_bi NOT NULL,
 data_nascimento  DATE           DEFAULT TRUNC(SYSDATE, 'DAY')
                                 CONSTRAINT nn_pessoas_data NOT NULL,
 cod_morada       NUMBER(8)      CONSTRAINT fk_pessoas_morada
                                   REFERENCES morada(cod_morada));
```

Mais informações: Ver Aula 6 na sebenta (páginas 77 a 93). No exercício 3 da Aula 5 na página 72 da sebenta aparece um exemplo de criação de tabela relativamente complexo onde podiam ver, por exemplo, como declarar mais do que uma restrição para uma só coluna. O Exemplo 6.17 na página 91 utiliza um valor de `DEFAULT` para a data usando uma função `ROUND`. Também podiam ver funções de data a partir da página 46.

Erros mais frequentes:

- Não conseguir definir o valor de omissão da data como sendo a data actual com as horas, minutos e segundos a zero.
- Não definir uma restrição
- Não definir uma restrição em colunas com duas restrições. O caso mais comum foi a restrição `NOT NULL` na `data_nascimento`.
- Falhar o tipo de dados de uma coluna. O caso mais comum foi o da chave forasteira `cod_morada` que é numérica (diz-se que tem 8 dígitos) e foi representada por um `VARCHAR2(8)` ou mesmo `VARCHAR2(10)` por muitos alunos.
- Seperar as restrições da mesma coluna por vírgulas. Sem desconto.

Comentários: Este pergunta, e aliás este grupo, eram de resposta simples. As únicas situações especiais na criação da tabela eram colocar os valores de hora, minuto e segundo a zero para o `DEFAULT` da data e saber como fazer para declarar mais do que uma restrição numa só coluna. O resto era interpretação da pergunta.

e) Crie uma sequência, de nome id_pessoa, que comece em 100.000.000 e termine em 999.999.999. A sequência deve avançar de 9 em 9 e quando atingir o máximo não deverá voltar ao princípio.

Resposta:

```
CREATE SEQUENCE id_pessoa
   START WITH 100000000
   INCREMENT BY 9
   MAXVALUE 999999999
   NOCYCLE;           /* Redundante por ser valor por DEFAULT. */
```

Mais informações: Aula 6 a partir da página 99.

Erros mais frequentes:

- Erros de sintaxe: "CREAT" em vez de "CREATE", "INCREMENT = 9" em vez de "INCREMENT BY 9", etc. Sem desconto porque assumo que em condições reais de trabalho qualquer pessoa poderia descobrir facilmente a sintaxe correcta consultando o Help do Oracle, um livro ou a sebenta das práticas.

Comentários: Esta era dada.

f) Insera duas linhas na tabela pessoas, usando a sequência criada no ponto anterior, com os seguintes dados:

Nome:'Pai Natal'

BI:'1234567890'

Data de nascimento:'1888-01-01'

Morada:null

Nome:'Recém-nascido'

BI:'9876543210'

Data de nascimento:hoje

Morada:1234

Resposta:

```
INSERT INTO pessoas (id, nome, bi, data_nascimento, cod_morada)
                    VALUES(id_pessoa.nextval,
                           'Pai Natal',
                           1234567890,
                           to_date('1888-01-01', 'yyyy-mm-dd'),
                           null);

INSERT INTO pessoas (id, nome, bi, cod_morada)
                    VALUES(id_pessoa.nextval,
                           'Recém-nascido',
                           9876543210,
                           1234); /* Notar que como data não é especificada */
                                  /* será usado o valor DEFAULT, data de    */
                                  /* hoje. Ver pergunta 2a).                */
```

Mais informações: A partir da página 65, na Aula 5.

Erros mais frequentes:

- Não usar ou usar mal a sequência.
- `cod_morada` não era um `VARCHAR2` mas sim um `NUMBER`. E assim não deve levar plicas, `'1234'`, como foi feito por muitos.
- `cod_morada` podia ter um valor nulo porque é uma chave forasteira: SE for específicado um valor ele terá que existir como chave primária noutra tabela, mas não é obrigatório que uma chave primária tenha um valor diferente de `NULL`.
- Não era para inserir `'hoje'` (☺) nem `'1999-01-11'` porque isso são cadeias de caracter e não valores de data. Dever-se-ia ter usado `SYSDATE` ou então deixar o valor `DEFAULT` funcionar.
- Quando não se quer especificar um valor para deixar os DEFAULTs funcionar não se deve fazer

```
INSERT INTO tabela (col1, col2, col3)
                    VALUES(valor1, , valor3); /* Errado!! …, ,… */
```

mas sim

```
INSERT INTO tabela (col1, col3)
                    VALUES(valor1, valor3);  /* É usado o       */
                                             /* DEFAULT de col2 */
```

Comentários: Falta de estudo? Falta de prática? Falta de tempo?

"Always Look on the Bright Side of Life"

Cheer up, Brian. You know what they say.
Some things in life are bad,
They can really make you mad.
Other things just make you swear and curse.
When you're chewing on life's gristle,
Don't grumble, give a whistle!
And this'll help things turn out for the best...
And...

(the music fades into the song)

..always look on the bright side of life!
(whistle)

Always look on the bright side of life...
If life seems jolly rotten,
There's something you've forgotten!
And that's to laugh and smile and dance and sing,

When you're feeling in the dumps,
Don't be silly chumps,
Just purse your lips and whistle -- that's the thing!
And... always look on the bright side of life...

(whistle)
Come on!

(other start to join in)
Always look on the bright side of life...
(whistle)

For life is quite absurd,
And death's the final word.
You must always face the curtain with a bow!
Forget about your sin -- give the audience a grin,
Enjoy it -- it's the last chance anyhow!

So always look on the bright side of death!
Just before you draw your terminal breath.
Life's a piece of shit,
When you look at it.

Life's a laugh and death's a joke, it's true,
You'll see it's all a show,
Keep 'em laughing as you go.
Just remember that the last laugh is on you!

And always look on the bright side of life...
(whistle)
Always look on the bright side of life
(whistle)

-in "Life of Brian", dos Monty Phyton

D.2. Exame de 1999.02.11

1. Considere as tabelas das aulas práticas, tendo a tabela `emp` o seguinte conteúdo:

```
SQLWKS> SELECT nemp,nome, funcao, sal, premios, ndep
     2>    FROM emp
     3>    ORDER BY ndep, sal;
NEMP       NOME                 FUNCAO       SAL        PREMIOS    NDEP
---------- -------------------- ----------- ---------- ---------- ----------
      1934 Olga Costa           Continuo         68300                    10
      1782 Silvia Teles         Encarregado     279450                    10
      1839 Jorge Sampaio        Presidente      890000                    10
      1876 Rita Pereira         Continuo         65100                    20
      1369 Antonio Silva        Continuo         70800                    20
      1902 Catarina Silva       Analista        435000                    20
      1566 Augusto Reis         Encarregado     450975                    20
      1788 Maria Dias           Analista        565000                    20
      1900 Tome Ribeiro         Continuo         56950                    30
      1499 Joana Mendes         Vendedor        145600      56300         30
      1844 Manuel Madeira       Vendedor        157800          0         30
      1521 Nelson Neves         Vendedor        212250      98500         30
      1654 Ana Rodrigues        Vendedor        221250      81400         30
      1698 Duarte Guedes        Encarregado     380850                    30
14 rows selected.
```

a) Mostre qual o resultado produzido pelo seguinte comando:

```
SELECT COUNT(*)                                  a,
       COUNT(premios)                            b,
       COUNT(NVL(premios, 100)) - COUNT(premios) c,
       COUNT(ndep)                               d,
       MAX(sal)                                  e,
       'blabla'                                  lixo
  FROM emp
  GROUP BY ndep;
```

RESPOSTA

```
A          B          C          D          E          LIXO
---------- ---------- ---------- ---------- ---------- ------
         3          0          3          3     890000 blabla
         5          0          5          5     565000 blabla
         6          4          2          6     380850 blabla
3 rows selected.
```

b) Se se retirasse a última linha do comando da alínea anterior, a do GROUP BY, o comando seria executado à mesma? Se não, porquê? Se sim, qual o resultado?

NOTA: Se e apenas se não conseguiu responder à alínea 1.a), codifique o comando que seleciona o nome do empregado com o salário máximo para cada departamento.

RESPOSTA
Sim e o resultado seria:

```
A          B          C          D          E          LIXO
---------- ---------- ---------- ---------- ---------- ------
        14          4         10         14     890000 blabla
1 row selected.
```

Resposta para a pergunta do NOTA

```
SELECT nome
  FROM emp
  WHERE (ndep, sal) IN (SELECT ndep, max(sal)
                        FROM emp
                        GROUP BY ndep);
```

c) Codifique o comando que seleciona o número do departamento que apresenta a média de salários mais alta. Use apenas a tabela emp.

RESPOSTA

```
SELECT ndep
  FROM emp
  GROUP BY ndep
  HAVING avg(sal) IN (SELECT max(avg(sal)) FROM emp GROUP BY ndep);
```

d) Crie uma tabela, emp2, que consiste nas colunas ndep, nome, funcao, sal, encar e dep e cujo conteúdo seja uma cópia do conteúdo das colunas com o mesmo nome da tabela emp. A cópia deve excluir os empregados que são 'Vendedores'.

RESPOSTA

```
CREATE TABLE emp2 AS
   SELECT nemp, nome, funcao, sal, encar, ndep
      FROM emp
      WHERE funcao != 'Vendedor';
```

e) Acrescente à tabela emp2 as seguintes restrições:

- nemp deve ser chave primária
- encar deve ser uma chave forasteira que aponta para nemp da mesma tabela
- ndep deve conter apenas os valores 10 ou 20 ou 30.
- sal não pode assumir valores nulos

RESPOSTA

```
ALTER TABLE emp2
   ADD CONSTRAINT pk_nemp_emp2 PRIMARY KEY (nemp);
ALTER TABLE emp2
   ADD CONSTRAINT fk_encar_emp2 FOREIGN KEY (encar) REFERENCES emp2(nemp);
ALTER TABLE emp2
   ADD CONSTRAINT ck_ndep_emp2 CHECK(ndep IN (10, 20, 30));
ALTER TABLE emp2
   ADD CONSTRAINT nn_sal_emp2 CHECK(sal IS NOT NULL);
```

f) Assuma que existe uma sequência de nome seq_emp2 cujo próximo valor é garantidamente maior que qualquer número de empregado que exista em emp2. Insira dois empregados na tabela emp2. Os dados de nome, sal e ndep dos empregados a inserir são os seguintes : 'nome 1', 100, 10 para o primeiro e 'nome 2', 200, 20 para o segundo. A funcao de ambos é 'SelfMadeMan'. Os números de empregado deverão ser obtidos através da sequência. Os números de encar deverão ser o número do próprio empregado.

RESPOSTA

```
CREATE SEQUENCE seq_emp2
  START WITH 5000

INSERT INTO emp2(nemp, nome, funcao, sal, encar, ndep)
   VALUES  (seq_emp2.nextval,  'nome  1',  'SelfMadeMan',  100,  seq_emp2.currval,
10);

INSERT INTO emp2(nemp, nome, funcao, sal, encar, ndep)
   VALUES  (seq_emp2.nextval,  'nome  2',  'SelfMadeMan',  200,  seq_emp2.currval,
20);
```

D.3. Exame Especial de 1999.03.30

Exame Especial de Bases de Dados I
30 de Março de 1999
Correcção da Parte Prática

por Pedro Bizarro

1) Dadas as tabelas das aulas práticas resolva as seguintes alíneas.

1.a) Na tabela dos empregados existe uma hierarquia. Os empregados possuem superiores. O superior ou encarregado de um empregado A é o empregado B onde o número de empregado de B, `nemp`, é igual ao número de encarregado de A, `encar`.

Digite o comando que gera uma listagem com três colunas onde a coluna da esquerda tem o nome de "`Grande Chefão`", a do meio tem o nome de "`Pequeno Chefão`" e a outra de "`Trabalhador`". Os empregados em "`Grande Chefão`" têm que ser encarregados dos empregados da mesma linha em "`Pequeno Chefão`" que por sua vez são encarregados dos empregados da mesma linha em "`Trabalhador`". Ordene os resultados por ordem alfabética crescente com prioridade na coluna da esquerda, depois na do meio e finalmente na da direita.

Resposta:

```
SELECT e1.nome "Grande Chefao",
       e2.nome "Pequeno Chefao",
       e3.nome "Trabalhador"
   FROM emp e1, emp e2, emp e3
   WHERE  e1.nemp = e2.encar
      AND e2.nemp = e3.encar
   ORDER BY 1,2,3;
```

1.b) Apresente o comado que calcula para cada empregado, o total da soma dos salários dos seus dependentes sem contar com o seu próprio salário.

Resposta:

```
SELECT e1.nome, sum(nvl(e2.sal, 0)) TotalSal
   FROM emp e1, emp e2
   WHERE e1.nemp = e2.encar
   GROUP BY e1.nome
   ORDER BY 2 desc
```

Ou (como sugerido pelo aluno António Ribeiro)

```
SELECT e1.nome, sum(e2.sal) + e1.sal TotalSal
   FROM emp e1, emp e2
   WHERE e1.nemp = e2.encar
   GROUP BY e1.nome
```

1.c) Apresente novo comando que calcule o mesmo total da alínea anterior mas incluíndo também o próprio salário.

Resposta:

```
SELECT e1.nome, sum(nvl(e2.sal, 0)) TotalSal
   FROM emp e1, emp e2
   WHERE e1.nemp = e2.encar
      OR e1.nemp = e2.nemp
   GROUP BY e1.nome
   ORDER BY 2 desc
```

1.d) Diga qual o código a usar para insirir na tabela dois empregados (com os números de empregado de 1111 e 2222) onde um seja o encarregado do outro e vice-versa. Os nomes deverão ser 'Vice' e 'Versa', a funcao será 'Confusão', o salário e o número de departamento serão, para ambos, 1000 e 10 respectivamente. Note que a inserção simples não é suficiente devido à existência de chaves forasteiras.

Resposta:

```
ALTER TABLE emp
  DISABLE CONSTRAINT encar_fk;

INSERT INTO emp VALUES (1111, 'Vice',  'Confusao',
                        2222, SYSDATE, 1000, null, 10);
INSERT INTO emp VALUES (2222, 'Versa', 'Confusao',
                        1111, SYSDATE, 1000, null, 10);

ALTER TABLE emp
  ENABLE CONSTRAINT encar_fk;
```

1.e) Produza os comandos que acrescentam à tabela `emp` uma coluna de nome 'Chefe' e tipo cadeia de caracteres variável com comprimento máximo de 20. Digite ainda num único comando, a instrução que actualiza os valores dessa coluna de modo a, para cada empregado, o seu valor de 'Chefe' corresponder ao nome do seu encarregado.

Resposta:

```
ALTER TABLE emp
  ADD chefe VARCHAR2(20);

UPDATE emp e SET chefe = (SELECT nome FROM emp WHERE e.encar = nemp)
```

Anexo E

Índice

E.1. Índice

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W