

# Aula 02

## *Banco de Dados II*



**Departamento de  
Computação**

Uni-FACEF



**ACADEMY**



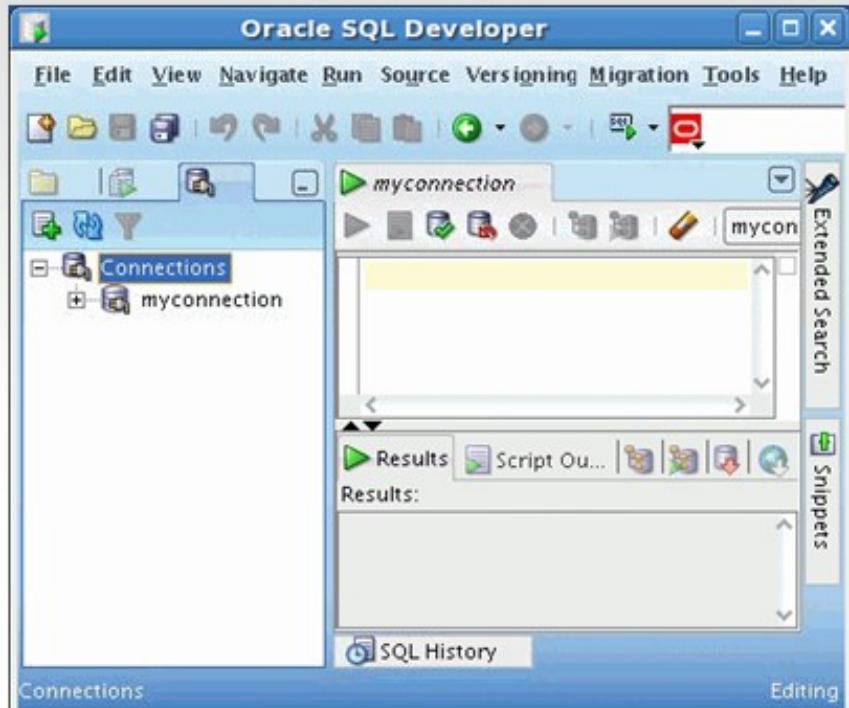
# Ferramentas

# Ferramentas

- *Diversas ferramentas* podem ser *utilizadas* para *estabelecer conexão* com um *BD Oracle*
- As *duas ferramentas* mais *habituais* são:
  - *SQL \*Plus* (*caractere*)
  - *SQL Developer* (*gráfico*)
- *Fornecidas* pela *Oracle Corporation*
- *SQL Developer* possui mais *recursos, comparado* com *SQL \*Plus*
- *iSQL \*Plus: apresentada* na *versão 9i* e *abandonada* na *versão 11g*

# Ferramentas

- **SQL \*Plus + SQL Developer**

A screenshot of a terminal window titled "Terminal". The window title bar also says "Terminal". The menu bar includes File, Edit, View, Terminal, Tabs, and Help. The terminal window displays the following text:

```
[oracle@EDRSR17P1 ~]$sqlplus
SQL*Plus: Release 11.2.0.0.2 Beta on Wed Jun 3 17:12:12 2009
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Enter user-name:
```

- *Ferramenta cliente/servidor*
- *Utilizada para:*
  - *Estabelecer conexão* com o BD
  - *Emissão de comandos SQL ad hoc*
  - *Criação de códigos PL/SQL*
  - *Possui recursos para formatação de resultados*
  - *Disponível em todas as plataformas* para as *quais* o BD foi *adaptado*

- O *arquivo executável* do SQL \*Plus no Linux é **sqlplus**
- **Variáveis:**
  - **\$ORACLE\_HOME**
    - *Aponta para o Oracle Home*
      - *Conjunto de regras e os diretórios que contêm os códigos e alguns arquivos de configuração*
  - **\$PATH**
    - *Inclui o diretório bin em Oracle Home*
  - **\$LD\_LIBRARY\_PATH**
    - *Inclui o diretório lib em Oracle Home*

- **Formato do *string* de *login*:**
  - *Nome* do *usuário* do BD (*barra* é *utilizada* como *delimitador*)
  - Em *seguida*, uma *senha* (@ é *utilizado* como *delimitador*)
  - Por fim, o *identificador* do *Oracle Net*
- **Exemplo:**
  - **Usuário** (*loja*); **senha** (*loja\_senha*); **BD** identificado por *orcl*  
`sqlplus loja/loja_senha@orcl`

# SQL Developer

- *O que é SQL Developer?*

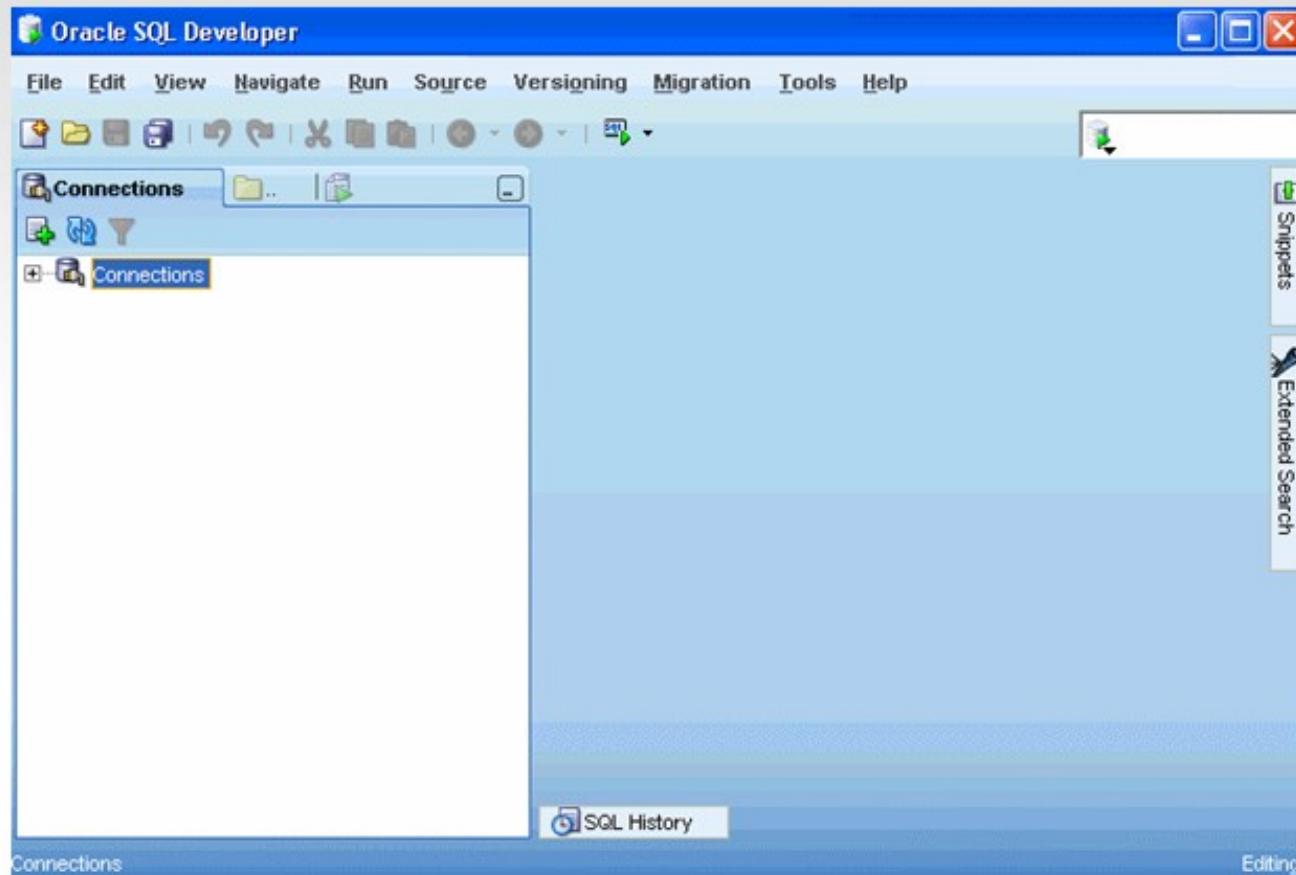
- É *uma ferramenta gráfica* que *aumenta* a *produtividade* e *simplifica* as *tarefas* de *desenvolvimento* de *banco* de *dados*
- Você pode se *conectar* a *qualquer esquema* de *banco* de *dados Oracle* usando a *autenticação* do *banco* de *dados padrão Oracle*

- *Especificações:*

- *Enviado juntamente* com o *Oracle Database 11g Release 2*
- *Desenvolvido* em *Java*
- *Compatível* com as *plataformas Windows, Linux e Mac OS*
- *Conectividade padrão*, usando o *driver (JDBC)*
- *Conecta-se* a **BD Oracle** versão **9.2.0.1** e ou *anteriores*
- Pode ser *baixado gratuitamente*

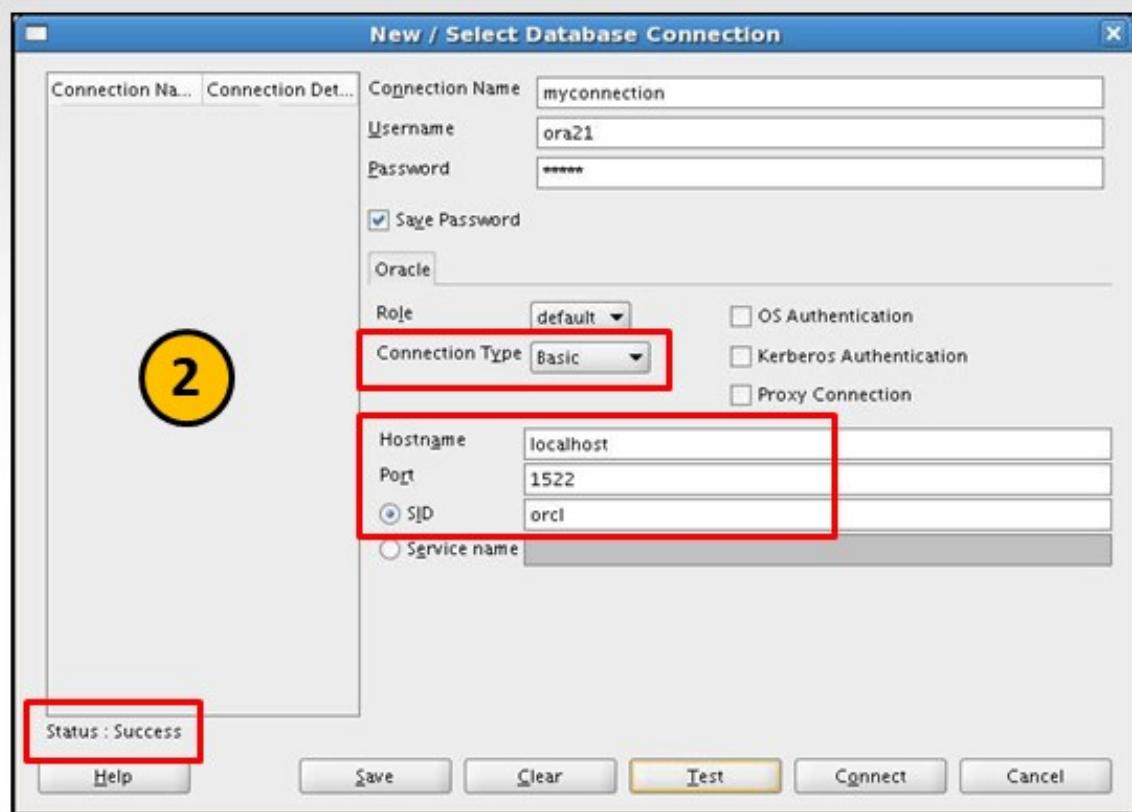
# SQL Developer

- *Interface de usuário* do SQL *Developer*:



# SQL Developer

- Criando uma *nova conexão*:



# SQL Developer

- **Criando uma nova conexão** (continuação):
  1. Em **Connections**, **botão direito** do **mouse em Connections** e selecione **New Connection**
  2. Na **janela “New / Select Database Connection”**, **digite o nome** da **conexão**. **Digite o nome** de **usuário** e a **senha** do **esquema** que você **deseja se conectar**

Os **botões** de **rádio Connection Type** permitem que **você escolha três opções**:

- **Basic:** solicita o **nome** do **servidor** do BD, a **porta** em que o **listener** do BD **aceitará** as **solicitações** de **conexão** e a **instância** (o SID) ou o **service** com o qual a **conexão** será **estabelecida**

# SQL Developer

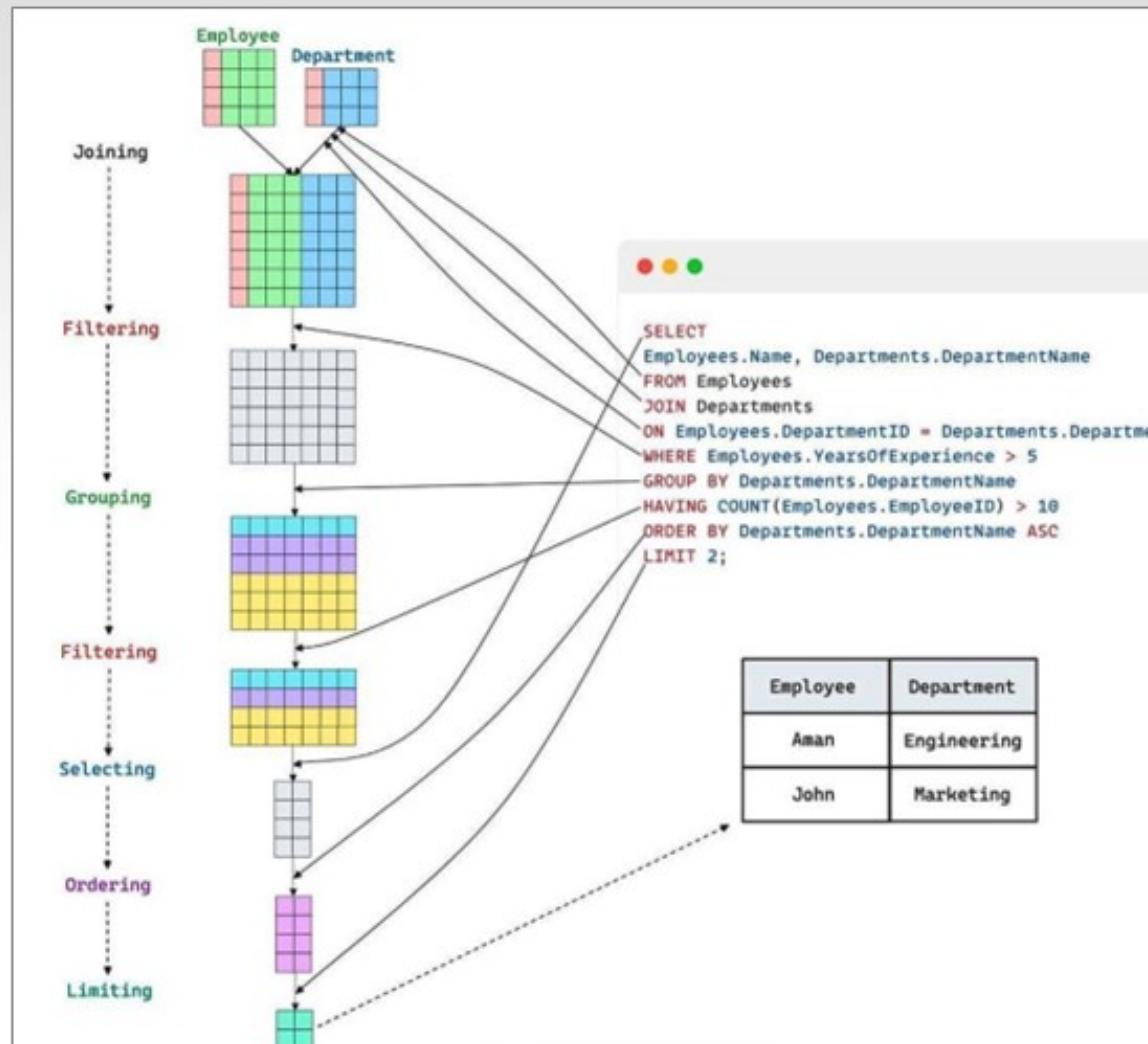
- **Criando uma nova conexão** (continuação):
  2. Na *janela “New / Select Database Connection”*, **digite o nome** da **conexão**. **Digite o nome** de **usuário** e a **senha** do **esquema** que você **deseja se conectar**  
Os **botões** de *rádio Connection Type* permitem que **você escolha três opções**:
    - **TNS**: caso um **método** de **resolução** de **nome esteja configurado**, um **alias** para o BD será **informado** (sem a **necessidade** de **detalhes** da **opção Basic**)
    - **Advanced**: **possibilita** a **entrada** de um **string** de **conexão JDBC** (**Java Database Connectivity**)
  3. A **conexão** é **adicionada** no **navegador** de **conexões** (**Connections Navigator**)



# Instrução SELECT

# Instrução SELECT

- *SQL Queries: Order of Execution*



# Instrução SELECT

- *Recupera dados* das *tabelas* do *BD*
- *Forma Simples:*
  - É *especificada* a *tabela* e as *colunas (atributos)* as quais *deseja-se recuperar dados*
- *Exemplo:*
  - *Recuperar* as *colunas id\_cliente, nome, sobrenome, dt\_nascimento* e *telefone* da tabela *tb\_clientes*

```
SELECT id_cliente, nome, sobrenome, dt_nascimento, telefone  
FROM tb_clientes;
```

# Instrução SELECT

- *Recuperando todas as colunas de uma tabela*
  - Utilizar o caractere asterisco ( \* )
- *Exemplo:*
  - *Recuperar todas as colunas da tabela tb\_clientes*

```
SELECT *
FROM tb_clientes;
```

# InSTRUÇÃO SELECT

- *Especificando* as *linhas* a serem *recuperadas* usando a *cláusula WHERE*
- *Mesmo* o *SGBD Oracle* possuindo uma *boa capacidade* de *armazenar* uma *grande quantidade* de *linhas* em uma *tabela*, eventualmente, você estará *interessado apenas* em um *subconjunto muito pequeno* de *linhas*

# Instrução SELECT

- **Exemplo:**

- Cláusula **WHERE** é utilizada para *recuperar* a *linha* da *tabela tb\_clientes* onde o *valor* da *coluna (atributo)* *id\_cliente* é *equivalente* ao *número 2*

```
SELECT *  
FROM tb_clientes  
WHERE id_cliente = 2;
```



# Identificadores de Linha

# Identificadores de Linha

- *Cada linha* de uma *tabela* no *Oracle* tem um *identificador exclusivo* denominado *rowid*
- **Rowid:**
  - Utilizado *internamente* para *armazenar a localização física* da *linha*
  - *Corresponde* ao *número constituído de 18 dígitos*

# Identificadores de Linha

- *Exemplo:*
  - *Recuperando* as **colunas ROWID** e ***id\_cliente*** em uma **consulta** utilizando a **tabela tb\_clientes**

```
SELECT ROWID, id_cliente  
FROM tb_clientes;
```

ROWID	ID_CLIENTE
AAAajLAAEAAAAD10AAA	1
AAAajLAAEAAAAD10AAB	2
AAAajLAAEAAAAD10AAC	3
AAAajLAAEAAAAD10AAD	4
AAAajLAAEAAAAD10AAE	5
AAAajLAAEAAAAD10AAF	6

# Identificadores de Linha

- *Utilizando o comando DESCRIBE, a coluna ROWID não aparece na saída, pois a mesma é utilizada internamente pelo BD*
- A coluna ROWID é conhecida como *pseudocoluna*

# Identificadores de Linha

- **Exemplo:**
  - Descrever a tabela `tb_clientes`
  - Observe que a coluna `ROWID` não aparece na saída

**DESCRIBE tb\_clientes;**

```
DESCRIBE tb_clientes
Name          Null    Type
-----
ID_CLIENTE    NOT NULL NUMBER(38)
NOME          NOT NULL VARCHAR2(10)
SOBRENOME     NOT NULL VARCHAR2(10)
DT_NASCIMENTO           DATE
TELEFONE        VARCHAR2(12)
FG_ATIVO        NUMBER(38)
```



# Números de Linha

# Números de Linha

- *Outra pseudocoluna é ROWNUM*
  - *Retorna o número da linha em um conjunto de resultados*
- *Exemplo (01):*
  - *Incluímos a coluna ROWNUM ao recuperar as linhas da tabela tb\_clientes*

```
SELECT ROWNUM, id_cliente, nome, sobrenome  
FROM tb_clientes;
```

ROWNUM	ID_CLIENTE	NOME	SOBRENOME
1	1	John	Brown
2	2	Cynthia	Green
3	3	Steve	White
4	4	Gail	Black
5	5	Doreen	Blue
6	6	Fred	Brown

# Números de Linha

- *Exemplo (02):*
  - Recuperar a *linha* da *tabela tb\_clientes* cujo *atributo (id\_cliente)* seja *equivalente* ao *número 3*

```
SELECT ROWNUM, id_cliente, nome, sobrenome  
FROM tb_clientes  
WHERE id_cliente = 3;
```

ROWNUM	ID_CLIENTE	NOME	SOBRENOME
1	3	Steve	White



# Cálculo Aritmético

# Cálculos Aritméticos

- *Possibilidade* de *efetuar cálculos* em *instruções SQL* usando *expressões aritméticas*
- As *expressões aritméticas* consistem em *dois operandos* (*números* ou *datas*) e um *operador aritmético*
- Os *quatro operadores aritméticos* são:
  - ( + ) *adição*
  - ( - ) *subtração*
  - ( \* ) *multiplicação*
  - ( / ) *divisão*

# Cálculos Aritméticos

- ***Exemplo (01):***

- A ***consulta*** a seguir ***utiliza*** o ***operador*** de ***multiplicação*** ( \* ) para ***calcular*** 2 ***multiplicado*** por 6
- Os ***números*** 2 e 6 são os ***operандos***

```
SELECT 2 * 6  
FROM dual;
```

# Cálculos Aritméticos

- As *regras normais* de *precedência* de *operador aritmético* se *aplicam* na *linguagem SQL*
- *Multiplicação* e *divisão* são *realizados primeiro*, seguidos pela *adição* e *subtração*
- *Exemplo (02):*
  - *Expressão*  $10 * 12 / 3 - 1$   
`SELECT 10 * 12 / 3 - 1  
FROM dual;`

# Cálculos Aritméticos

- Permite utilizar *parênteses* para *especificar* a *ordem de execução* dos *operadores*
  - No *exemplo abaixo*, os *parênteses* são *usados* para *efetuar primeiro* o *cálculo* de  $12 / 3 - 1$ , cujo *resultado* é então *multiplicado* por *10*
  - *Exemplo (03):*
    - *Expressão*  $10 * (12 / 3 - 1)$
- ```
SELECT 10 * (12 / 3 - 1)  
FROM dual;
```

# Aritmética de Data

- *Utilização* de *operadores* de *adição* e *subtração* com *datas*
- *Permite somar* um *número*, *representando* um determinado *número* de *dias* em uma *data*
- *Exemplo (01):*
  - *Somar dois dias* a **22 de junho de 2014**, *exibindo* a *data resultante*

```
SELECT TO_DATE('22/JUN/2014') + 2  
FROM dual;
```

# Aritmética de Data

- ***Exemplo (02):***

- *Subtrair uma data de outra, produzindo o número de dias entre as duas datas*
- *Subtrair 31 de dezembro de 2014, de 22 de junho de 2014*

```
SELECT TO_DATE('31/DEC/2014') - TO_DATE('22/JUN/2014')
FROM dual;
```

# Usando Colunas (Aritmética)

- Os *operandos* **não** precisam ser *números* ou *datas*
- *Podem* ser *colunas* de uma *tabela* qualquer
- *Exemplo:*
  - As *colunas* *nm\_produto* e *preco* são *recuperadas* da *tabela* *tb\_produtos*

```
SELECT nm_produto, preco, preco + 2.00  
FROM tb_produtos;
```

| NM_PRODUTO     | PRECO | PRECO+2.00 |
|----------------|-------|------------|
| Modern Science | 19.95 | 21.95      |
| Chemistry      | 30    | 32         |
| Supernova      | 25.99 | 27.99      |
| Tank War       | 13.95 | 15.95      |



## Tabela *dual*

# A tabela dual

- *Possui uma única linha*
- A saída do comando **DESCRIBE** mostra a *estrutura* da *tabela dual*
- *Observe que a tabela dual tem uma coluna do tipo VARCHAR2 nomeada de dummy e uma única linha com o valor X*

# A tabela dual

**DESCRIBE dual;**

```
DESCRIBE dual
Name Null Type
-----
DUMMY      VARCHAR2(1)
```

**SELECT \***  
**FROM dual;**

|       |
|-------|
| DUMMY |
| X     |



## Usando *Alias*

# Usando apelidos de Coluna

- **Não** estamos *limitados* a *usar* o *cabeçalho* gerado pelo *Oracle*
- É *possível fornecer* o seu *próprio cabeçalho*, usando um *apelido (alias)*
- *Exemplo (01):*
  - A *expressão preco \* 2 recebe* o *apelido DOBRO\_PREÇO*

```
SELECT preco, preco * 2 DOBRO_PREÇO  
FROM tb_produtos;
```

| PRECO | DOBRO_PREÇO |
|-------|-------------|
| 19.95 | 39.9        |
| 30    | 60          |
| 25.99 | 51.98       |
| 13.95 | 27.9        |

# Usando apelidos de Coluna

- ***Exemplo (02):***

- *Preservando a caixa de texto do seu nome alternativo*

```
SELECT preco, preco * 2 "Dobro do Preço"  
FROM tb_produtos;
```

| PRECO | Dobro do Preço |
|-------|----------------|
| 19.95 | 39.9           |
| 30    | 60             |
| 25.99 | 51.98          |
| 13.95 | 27.9           |

# Usando apelidos de Coluna

- *Exemplo (03):*
  - Usando a palavra-chave opcional **AS** antes do nome alternativo

```
SELECT preco, preco * 2 AS "Dobro do Preço"  
FROM tb_produtos;
```

| PRECO | Dobro do Preço |
|-------|----------------|
| 19.95 | 39.9           |
| 30    | 60             |
| 25.99 | 51.98          |
| 13.95 | 27.9           |



# Concatenação

# Concatenação entre colunas

- *Permite combinar os valores de colunas usando concatenação*
- *Criação de uma saída mais amigável*
- *Operador de concatenação ( || )*

# Concatenação entre colunas

- **Exemplo:**

- Na *tabela tb\_clientes*, as *colunas nome* e *sobrenome* contêm o *nome* do *cliente*
- *Não* seria **ótimo combinar** as *duas colunas*?

```
SELECT nome || ' ' || sobrenome AS "Nome do Cliente"
```

```
FROM tb_clientes;
```

| Nome do Cliente |
|-----------------|
| John Brown      |
| Cynthia Green   |
| Steve White     |
| Gail Black      |
| Doreen Blue     |
| Fred Brown      |



# Valores Nulos

# Valores Nulos

- *Representação de um valor desconhecido no BD (valor nulo)*
- Um **valor nulo** **não** é um **string em branco**, e sim um **valor único** cujo **significado** do **valor** da **coluna** é **desconhecido**
- O **cliente** de **número 4** tem um **valor nulo** na **coluna DT\_NASCIMENTO**
- O **cliente** de **número 5** tem um **valor nulo** na **coluna TELEFONE**

# Valores Nulos

- *Tabela tb\_clientes:*

| ID_CLIENTE | NOME    | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|---------|-----------|---------------|--------------|----------|
| 1          | John    | Brown     | 01-JAN-65     | 800-555-1211 | 1        |
| 2          | Cynthia | Green     | 05-FEB-68     | 800-555-1212 | 1        |
| 3          | Steve   | White     | 16-MAR-71     | 800-555-1213 | 1        |
| 4          | Gail    | Black     | (null)        | 800-555-1214 | 1        |
| 5          | Doreen  | Blue      | 20-MAY-70     | (null)       | 1        |
| 6          | Fred    | Brown     | 01-JAN-70     | 800-555-1215 | 1        |

# Valores Nulos

- *Verificando a existência de valores nulos ( IS NULL )*
- *Exemplo (01):*
  - O *cliente n º 4* é *recuperado*, pois seu *valor correspondente* a **DT\_NASCIMENTO** é **nulo**

```
SELECT id_cliente, nome, sobrenome, dt_nascimento  
FROM tb_clientes  
WHERE dt_nascimento IS NULL;
```

# Valores Nulos

- **Verificando a existência de valores nulos ( IS NULL )**
- **Exemplo (02):**
  - O *cliente n º 5* é *recuperado*, pois seu *valor correspondente* ao **TELEFONE** é **nulo**

```
SELECT id_cliente, nome, sobrenome, telefone  
FROM tb_clientes  
WHERE telefone IS NULL;
```

# Valores Nulos

- *Valores nulos nada exibem*
- Como você *identifica a diferença entre um valor nulo e um string em branco?*
- *Resposta:*
  - Utilizando a função interna NVL() do Oracle

# Valores Nulos

- **NVL**(abreviatura de **Null VaLue**):
  - *Retorna outro valor no lugar de um valor nulo*
  - *Aceita dois parâmetros*
  - Uma *coluna (qualquer expressão) que resulte em um valor*)
  - *Valor a ser retornado, caso o primeiro parâmetro seja nulo*

# Valores Nulos

- **Exemplo (01):**

- *NVL retorna o string “Número de telefone desconhecido” quando a coluna telefone possuir um valor nulo*

```
SELECT id_cliente, nome, sobrenome,  
NVL(telefone, 'Número do telefone desconhecido') AS Número_Telefone  
FROM tb_clientes;
```

| ID_CLIENTE | NOME    | SOBRENOME | NÚMERO_TELEFONE                 |
|------------|---------|-----------|---------------------------------|
| 1          | John    | Brown     | 800-555-1211                    |
| 2          | Cynthia | Green     | 800-555-1212                    |
| 3          | Steve   | White     | 800-555-1213                    |
| 4          | Gail    | Black     | 800-555-1214                    |
| 5          | Doreen  | Blue      | Número do telefone desconhecido |
| 6          | Fred    | Brown     | 800-555-1215                    |

# Valores Nulos

- **Exemplo (02):**

- *NVL retorna a data “22 JUN 2013” quando a coluna DT\_NASCIMENTO possuir um valor nulo*

```
SELECT id_cliente, nome, sobrenome,  
       NVL(dt_nascimento, '22/JUN/2013') AS "Data de Nascimento"  
FROM tb_clientes;
```

| ID_CLIENTE | NOME    | SOBRENOME | Data de Nascimento |
|------------|---------|-----------|--------------------|
| 1          | John    | Brown     | 01-JAN-65          |
| 2          | Cynthia | Green     | 05-FEB-68          |
| 3          | Steve   | White     | 16-MAR-71          |
| 4          | Gail    | Black     | 22-JUN-13          |
| 5          | Doreen  | Blue      | 20-MAY-70          |
| 6          | Fred    | Brown     | 01-JAN-70          |

# Valores Nulos

- **Função NULLIF**

- Realiza a *comparação* entre *duas expressões*

- **Sintaxe:**

**NULLIF (expressão1, expressão2)**

- **NULLIF** *compara* *expressão1* e *expressão2*, se elas forem *equivalentes*, a *função retorna* **NULL**. Caso elas *sejam distintas*, a função retorna *expressão1*

# Valores Nulos

- **Exemplo:**

- O **LENGTH** do **NOME** da **TB\_FUNCIONARIOS** é *comparado* com o **LENGTH** do **SOBRENOME** pertinente a mesma tabela

```
SELECT nome, LENGTH(nome) "expressão1",
       sobrenome, LENGTH(sobrenome) "expressão2",
       NULLIF(LENGTH(nome), LENGTH(sobrenome)) "resultado"
```

```
FROM tb_funcionarios;
```

| AZ | NOME      | AZ | expressão1 | AZ     | SOBRENOME | AZ | expressão2 | AZ     | RESULTADO |
|----|-----------|----|------------|--------|-----------|----|------------|--------|-----------|
|    | Doreen    |    | 6          | Penn   |           |    | 4          |        | 6         |
|    | Frank     |    | 5          | Howard |           |    | 6          |        | 5         |
|    | Fred      |    | 4          | Hobbs  |           |    | 5          |        | 4         |
|    | Henry     |    | 5          | Heyson |           |    | 6          |        | 5         |
|    | James     |    | 5          | Smith  |           |    | 5          | (null) |           |
|    | Jane      |    | 4          | Brown  |           |    | 5          |        | 4         |
|    | Jean      |    | 4          | Blue   |           |    | 4          | (null) |           |
|    | Jefferson |    | 9          | Mendes |           |    | 6          |        | 9         |

# Valores Nulos

- **Função COALESCE**

- A *vantagem* da *função COALESCE* sobre a *função NVL* é que a *função COALESCE* pode assumir diversos valores;
- Se a *primeira expressão* **não** for *nula*, a *função COALESCE* retorna essa *expressão*, caso *contrário*, ela realiza um *COALESCE* das *expressões restantes*

- **Sintaxe:**

**COALESCE (*expressão1, expressão2, ... expressão<sub>n</sub>*)**

- *expressão1* é *retornada* se **não** for *nula*
- *expressão2* é *retornada* se a *expressão1* for *nula* e *ela não*
- *expressão3* é *retornada* se as *expressões anteriores forem nulas*

# Valores Nulos

- **Exemplo:**

- Caso o **ID\_GERENTE** **não** for **nulo**, o mesmo será **apresentado**.
- Se **ID\_GERENTE** for **nulo**, **PERCENTUAL\_COMISSAO** é **apresentado**
- Se **ambos forem nulos**, o **string** “*Nenhuma comissão e nenhum gerente*” será **apresentado**

```
SELECT sobrenome, id_empregado, percentual_comissao, id_gerente,  
       COALESCE(TO_CHAR(percentual_comissao),  
                  TO_CHAR(id_gerente),  
                  'Nenhuma comissão e nenhum gerente')  
FROM tb_empregado;
```

# Valores Nulos

- **Exemplo:**

- Caso o **ID\_GERENTE** **não** for **nulo**, o mesmo será **apresentado**.
- Se **ID\_GERENTE** for **nulo**, **PERCENTUAL\_COMISSAO** é **apresentado**
- Se **ambos forem nulos**, o **string** “*Nenhuma comissão e nenhum gerente*” será **apresentado**

| SOBRENOME | ID_EMPREGADO | PERCENTUAL_COMISSAO | ID_GERENTE | COALESCE(TO_CHAR(PERCENTUAL_COMISSAO |
|-----------|--------------|---------------------|------------|--------------------------------------|
| King      | 100          | (null)              | (null)     | Nenhuma comissão e nenhum gerente    |
| Kochhar   | 101          | (null)              | 100        | 100                                  |
| De Haan   | 102          | (null)              | 100        | 100                                  |
| Hunold    | 103          | (null)              | 102        | 102                                  |
| Ernst     | 104          | (null)              | 103        | 103                                  |
| Austin    | 105          | (null)              | 103        | 103                                  |
| Pataballa | 106          | (null)              | 103        | 103                                  |



# **Distinct**

# Exibindo Linhas Distintas

- ***Exemplo (01):***

- *Listar os clientes que compraram produtos de nossa loja virtual*
- *Recuperar a coluna ID\_CLIENTE da tabela tb\_compras*

```
SELECT id_cliente  
FROM tb_compras;
```

| ID_CLIENTE |
|------------|
| 1          |
| 2          |
| 2          |
| 1          |
| 1          |

# Exibindo Linhas Distintas

- Alguns *clientes* realizaram *mais de uma compra* e, portanto, *aparecem duplicado*
- *Como eliminar as linhas duplicadas que contêm a mesma identificação de cliente?*
- *Resposta:*
  - **DISTINCT** (*suprimir as linhas duplicadas*)

# Exibindo Linhas Distintas

- **Exemplo (02):**

- **Eliminando** as *linhas duplicadas* da *consulta anterior* através do uso de **DISTINCT**

```
SELECT DISTINCT id_cliente  
FROM tb_compras;
```

| ID_CLIENTE |
|------------|
| 1          |
| 2          |



# Comparando Valores

# Comparando Valores

- *Operadores utilizados para comparar valores:*

| Operador | Descrição                                                    |
|----------|--------------------------------------------------------------|
| =        | igual                                                        |
| < ou !=  | diferente                                                    |
| <        | menor que                                                    |
| >        | maior que                                                    |
| <=       | menor ou igual a                                             |
| >=       | maior ou igual a                                             |
| ANY      | compara um valor com qualquer valor em uma lista             |
| SOME     | idêntico ao operador ANY (você deve usar ANY "mais legível") |
| ALL      | compara um valor com todos os valores em uma lista           |

# Comparando Valores

- **Exemplo (01):**

- *Recuperar as linhas da tabela tb\_clientes cujo valor na coluna ID\_CLIENTE é diferente de 2:*

```
SELECT *  
FROM tb_clientes  
WHERE id_cliente <> 2;
```

| ID_CLIENTE | NOME   | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|--------|-----------|---------------|--------------|----------|
| 1          | John   | Brown     | 01-JAN-65     | 800-555-1211 | 1        |
| 3          | Steve  | White     | 16-MAR-71     | 800-555-1213 | 1        |
| 4          | Gail   | Black     | (null)        | 800-555-1214 | 1        |
| 5          | Doreen | Blue      | 20-MAY-70     | (null)       | 1        |
| 6          | Fred   | Brown     | 01-JAN-70     | 800-555-1215 | 1        |

# Comparando Valores

- ***Exemplo (02):***

- *Recuperar* as ***colunas ID\_PRODUTO e NM\_PRODUTO***, onde a coluna ***ID\_PRODUTO*** seja *maior* que ***2***:

```
SELECT id_produto, nm_produto  
FROM tb_produtos  
WHERE id_produto > 2;
```

| ID_PRODUTO | NM_PRODUTO |
|------------|------------|
| 3          | Supernova  |
| 4          | Tank War   |

# Comparando Valores

- **Exemplo (03):**

- Usa a *pseudocoluna ROWNUM* e o operador `<=` para *recuperar* as *três primeiras linhas* da *tabela tb\_produtos*:

```
SELECT ROWNUM, id_produto, nm_produto  
FROM tb_produtos  
WHERE ROWNUM <= 3;
```

| ROWNUM | ID_PRODUTO | NM_PRODUTO       |
|--------|------------|------------------|
| 1      |            | 1 Modern Science |
| 2      |            | 2 Chemistry      |
| 3      |            | 3 Supernova      |

# Comparando Valores

- ***Exemplo (04):***

- *Uso do operador ANY em uma cláusula WHERE para comparar um valor com qualquer um dos valores de uma lista*
- ***Pré-requisito:***
  - Inserir um operador =, <>, <, >, <= ou >= **antes** de ANY

# Comparando Valores

- **Exemplo (04):**

- Recuperar as *linhas* da *tabela tb\_clientes* onde o *valor* na *coluna ID\_CLIENTE* é *maior* do que *qualquer* um dos *valores* 2, 3 ou 4

```
SELECT *  
FROM tb_clientes  
WHERE id_cliente > ANY (2, 3, 4);
```

| ID_CLIENTE | NOME   | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|--------|-----------|---------------|--------------|----------|
| 3          | Steve  | White     | 16-MAR-71     | 800-555-1213 | 1        |
| 4          | Gail   | Black     | (null)        | 800-555-1214 | 1        |
| 5          | Doreen | Blue      | 20-MAY-70     | (null)       | 1        |
| 6          | Fred   | Brown     | 01-JAN-70     | 800-555-1215 | 1        |

# Comparando Valores

- ***Exemplo (05):***

- *Uso do operador ALL em uma cláusula WHERE para comparar um valor com todos os valores de uma lista*
  - *Pré-requisito:*
    - Inserir um operador =, <>, <, >, <= ou >= **antes** de ALL

# Comparando Valores

- **Exemplo (05):**

- Recuperar as *linhas* da *tabela tb\_clientes* onde o *valor* na *coluna ID\_CLIENTE* é *maior* do que os *valores* 2, 3 ou 4

```
SELECT *  
FROM tb_clientes  
WHERE id_cliente > ALL (2, 3, 4);
```

| ID_CLIENTE | NOME   | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|--------|-----------|---------------|--------------|----------|
| 5          | Doreen | Blue      | 20-MAY-70     | (null)       | 1        |
| 6          | Fred   | Brown     | 01-JAN-70     | 800-555-1215 | 1        |



# Operadores SQL

# Usando os Operadores SQL

- *Permitem limitar as linhas com base na correspondência de padrão de strings, listas de valores, intervalos de valores e valores nulos*

| Operador    | Descrição                                                    |
|-------------|--------------------------------------------------------------|
| LIKE        | corresponde a padrão em strings                              |
| IN          | corresponde a lista de valores                               |
| BETWEEN     | corresponde a um intervalo de valores                        |
| IS NULL     | corresponde a valores nulos                                  |
| IS NAN      | corresponde a um valor especial NAN (not a number)           |
| IS INFINITE | corresponde a valores BINARY_FLOAT e BINARY_DOUBLE infinitos |

# Usando os Operadores SQL

- *Possibilidade de utilizar NOT para inverter o significado de um operador:*
  - **NOT LIKE**
  - **NOT IN**
  - **NOT BETWEEN**
  - **IS NOT NULL**
  - **IS NOT NAN**
  - **IS NOT INFINITE**

# Operador LIKE

- *Uso do operador LIKE em uma cláusula WHERE para procurar um padrão em um string*
- Os *padrões* são *especificados* usando uma *combinação* de *caracteres normais* e os *dois caracteres curinga*:
  - *Sublinhado ( \_ )*: *corresponde a um caractere em uma posição específica*
  - *Porcentagem ( % )*: *corresponde a qualquer número de caracteres a partir de uma determinada posição*

# Operador LIKE

- *Exemplo de Padrão:*

- '\_**o**%'

- *Sublinhado ( \_ ) corresponde a qualquer caractere na primeira posição*
    - “**o**” *corresponde a um caractere “o” na segunda posição*
    - *Porcentagem ( % ) corresponde a todos os caracteres após o caractere “o”*

# Operador LIKE

- **Exemplo (01):**

- *Uso do operador LIKE para procurar o padrão '\_o%' na coluna nome da tabela tb\_clientes:*

```
SELECT *  
FROM tb_clientes  
WHERE nome LIKE '_o%';
```

| ID_CLIENTE | NOME   | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|--------|-----------|---------------|--------------|----------|
| 1          | John   | Brown     | 01-JAN-65     | 800-555-1211 | 1        |
| 5          | Doreen | Blue      | 20-MAY-70     | (null)       | 1        |

# Operador LIKE

- ***Exemplo (02):***

- Se for *necessário procurar os caracteres de sublinhado ou porcentagem reais* em um *string*, use a *opção ESCAPE* para identificá-los

# Operador LIKE

- **Exemplo:**

- '%\%%' ESCAPE '\'
- ESCAPE diz ao BD como *diferenciar* os *caracteres* a serem *pesquisados* dos *curingas (barra invertida)*
- O *primeiro %* é *tratado* como *curinga, correspondendo a qualquer número de caracteres*
- O *segundo %* é *tratado* como um *caractere real* a ser *procurado*
- O *terceiro %* é *tratado* como *curinga, correspondendo a qualquer número de caracteres*

# Operador LIKE

- ***Exemplo (02):***

- *Uso do operador LIKE para procurar o padrão '%\%%' ESCAPE '\'* na coluna NOME da tabela tb\_promocao:

```
SELECT nome  
FROM tb_promocao  
WHERE nome LIKE '%\%%' ESCAPE '\';
```

| NOME                   |
|------------------------|
| 10% off Z Files        |
| 20% off Pop 3          |
| 30% off Modern Science |
| 20% off Tank War       |
| 10% off Chemistry      |
| 20% off Creative Yell  |
| 15% off My Front Line  |

# Operador IN

- O uso do **operador IN** em uma **cláusula WHERE** para **recuperar** as **linhas** cujo **valor** da **coluna** está em **uma lista**
- **Exemplo (01):**
  - O uso do **IN** para **recuperar** as **linhas** da **tabela tb\_clientes** onde **ID\_CLIENTE** corresponde a **2, 3 ou 5**:

```
SELECT *  
FROM tb_clientes  
WHERE id_cliente IN (2, 3, 5);
```

| ID_CLIENTE | NOME    | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|---------|-----------|---------------|--------------|----------|
| 2          | Cynthia | Green     | 05-FEB-68     | 800-555-1212 | 1        |
| 3          | Steve   | White     | 16-MAR-71     | 800-555-1213 | 1        |
| 5          | Doreen  | Blue      | 20-MAY-70     | (null)       | 1        |

# Operador IN

- **Exemplo (02):**

- O **uso** do **NOT IN** para *recuperar* as *linhas não recuperadas* por **IN**:

```
SELECT *  
FROM tb_clientes  
WHERE id_cliente NOT IN (2, 3, 5);
```

| ID_CLIENTE | NOME | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|------|-----------|---------------|--------------|----------|
| 1          | John | Brown     | 01-JAN-65     | 800-555-1211 | 1        |
| 4          | Gail | Black     | (null)        | 800-555-1214 | 1        |
| 6          | Fred | Brown     | 01-JAN-70     | 800-555-1215 | 1        |

# Operador IN

- **Exemplo (03):**

- *Observação: NOT IN retorna falso se o valor da lista for nulo*
- A **consulta** a seguir **não retorna nenhuma linha**, pois um **valor nulo** está **incluído** na **lista**

```
SELECT *  
FROM tb_clientes  
WHERE id_cliente NOT IN (2, 3, 5, NULL);
```

# Operador BETWEEN

- O *uso* do *operador BETWEEN* em uma *cláusula WHERE* para *recuperar* as *linhas* cujo *valor* da *coluna* está em um *intervalo específico*
- *Intervalo:*
  - *Inclui os valores* das *duas extremidades* do *intervalo*

# Operador BETWEEN

- **Exemplo (01):**

- O **uso** do **BETWEEN** para *recuperar* as *linhas* da *tabela tb\_clientes* onde o **ID\_CLIENTE** esteja **entre 1 e 3**:

```
SELECT *  
FROM tb_clientes  
WHERE id_cliente BETWEEN 1 AND 3;
```

| ID_CLIENTE | NOME    | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|---------|-----------|---------------|--------------|----------|
| 1          | John    | Brown     | 01-JAN-65     | 800-555-1211 | 1        |
| 2          | Cynthia | Green     | 05-FEB-68     | 800-555-1212 | 1        |
| 3          | Steve   | White     | 16-MAR-71     | 800-555-1213 | 1        |

# Operador BETWEEN

- *Exemplo (02):*

- **NOT BETWEEN** *recupera* as *linhas não recuperadas* por **BETWEEN**:

```
SELECT *
FROM tb_clientes
WHERE id_cliente NOT BETWEEN 1 AND 3;
```

| ID_CLIENTE | NOME   | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|--------|-----------|---------------|--------------|----------|
| 4          | Gail   | Black     | (null)        | 800-555-1214 | 1        |
| 5          | Doreen | Blue      | 20-MAY-70     | (null)       | 1        |
| 6          | Fred   | Brown     | 01-JAN-70     | 800-555-1215 | 1        |

# Operadores Lógicos

- *Permitem limitar as linhas com base em condições lógicas*

| Operador           | Descrição                                                                 |
|--------------------|---------------------------------------------------------------------------|
| $x \text{ AND } y$ | retorna verdadeiro quando $x$ e $y$ são verdadeiros                       |
| $x \text{ OR } y$  | retorna verdadeiro quando $x$ ou $y$ são verdadeiros                      |
| $\text{NOT } x$    | retorna verdadeiro se $x$ for falso e retorna falso se $x$ for verdadeiro |

# Operadores Lógicos

- **Exemplo (01):**

- O *uso* do *operador AND* para *recuperar* as *linhas* da *tabela tb\_clientes* onde as *duas condições* são *verdadeiras*:

- A *coluna DT\_NASCIMENTO* é *maior* que *1º de janeiro de 1970*
  - A *coluna ID\_CLIENTE* é *maior* que *3*

```
SELECT *
FROM tb_clientes
WHERE dt_nascimento > '01/JAN/1970' AND
      id_cliente > 3;
```

| ID_CLIENTE | NOME   | SOBRENOME | DT_NASCIMENTO | TELEFONE | FG_ATIVO |
|------------|--------|-----------|---------------|----------|----------|
| 5          | Doreen | Blue      | 20-MAY-70     | (null)   | 1        |

# Operadores Lógicos

- **Exemplo (02):**

- O *uso* do *operador OR* para *recuperar* as *linhas* da *tabela tb\_clientes* onde uma das *duas condições* é *verdadeira*:

- A *coluna DT\_NASCIMENTO* é *maior* que *1º de janeiro de 1970*
- A *coluna ID\_CLIENTE* é *maior* que *3*

```
SELECT *
FROM tb_clientes
WHERE dt_nascimento > '01/JAN/1970' OR
      id_cliente > 3;
```

| ID_CLIENTE | NOME   | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|--------|-----------|---------------|--------------|----------|
| 3          | Steve  | White     | 16-MAR-71     | 800-555-1213 | 1        |
| 4          | Gail   | Black     | (null)        | 800-555-1214 | 1        |
| 5          | Doreen | Blue      | 20-MAY-70     | (null)       | 1        |
| 6          | Fred   | Brown     | 01-JAN-70     | 800-555-1215 | 1        |

# Precedência de Operadores

- Mesclar AND e OR na *mesma expressão*
- AND terá *precedência* sobre OR (“*ter precedência*” – *executado primeiro*)
- Operadores de *comparação* tem *precedência* sobre AND
- Anulação da *precedência* (*usar parênteses*) para *indicar a ordem* que *deseja executar* as *expressões*

# Precedência de Operadores

- **Exemplo (01):**

- Recupera as *linhas* da *tabela tb\_clientes* onde *uma* das *três condições* é *verdadeira*:

- A *coluna DT\_NASCIMENTO* é *maior* que *1º de janeiro de 1970*
    - A *coluna ID\_CLIENTE* é *menor* que *2*
    - A *coluna TELEFONE* tem *1211* no *final*

**SELECT \***

**FROM tb\_clientes**

**WHERE dt\_nascimento > '01/JAN/1970' OR**

**id\_cliente < 2 AND**

**telefone LIKE '%1211';**

# Precedência de Operadores

- **Exemplo (01):**

- Recupera as *linhas* da *tabela tb\_clientes* onde *uma* das *três condições* é *verdadeira*:

- A *coluna DT\_NASCIMENTO* é *maior* que **1º de janeiro de 1970**
    - A *coluna ID\_CLIENTE* é *menor* que **2**
    - A *coluna TELEFONE* tem **1211** no *final*

| ID_CLIENTE | NOME   | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|--------|-----------|---------------|--------------|----------|
| 1          | John   | Brown     | 01-JAN-65     | 800-555-1211 | 1        |
| 3          | Steve  | White     | 16-MAR-71     | 800-555-1213 | 1        |
| 5          | Doreen | Blue      | 20-MAY-70     | (null)       | 1        |



# **ORDER BY**

# Cláusula ORDER BY

- *Usada para classificar as linhas recuperadas por uma consulta*
- Pode *especificar uma ou mais colunas nas quais os dados serão classificados*



# Cláusula ORDER BY

- **Exemplo (01):**

- **Uso** do ORDER BY para *classificar* por SOBRENOME as *linhas recuperadas* da *tabela tb\_clientes*:

```
SELECT *  
FROM tb_clientes  
ORDER BY sobrenome;
```

| ID_CLIENTE | NOME  | SOBRENOME | DT_NASCIMENTO | TELEFONE | FG_ATIVO |
|------------|-------|-----------|---------------|----------|----------|
| 4 Gail     | Black | (null)    | 800-555-1214  |          | 1        |
| 5 Doreen   | Blue  | 20-MAY-70 | (null)        |          | 1        |
| 1 John     | Brown | 01-JAN-65 | 800-555-1211  |          | 1        |
| 6 Fred     | Brown | 01-JAN-70 | 800-555-1215  |          | 1        |
| 2 Cynthia  | Green | 05-FEB-68 | 800-555-1212  |          | 1        |
| 3 Steve    | White | 16-MAR-71 | 800-555-1213  |          | 1        |

# Cláusula ORDER BY

- *Palavra-chave DESC* pode ser *utilizada* para *classificar* as *colunas* em *ordem decrescente*
- *Palavra-chave ASC* pode ser *utilizada* para *especificar explicitamente* uma *classificação crescente (default)*



# Cláusula ORDER BY

- **Exemplo (02):**

- *Uso do ORDER BY para classificar as linhas recuperadas da tabela tb\_clientes por NOME em ordem crescente e SOBRENOME em ordem decrescente*

```
SELECT *
FROM tb_clientes
ORDER BY nome ASC, sobrenome DESC;
```

| ID_CLIENTE | NOME    | SOBRENOME | DT_NASCIMENTO | TELEFONE     | FG_ATIVO |
|------------|---------|-----------|---------------|--------------|----------|
| 2          | Cynthia | Green     | 05-FEB-68     | 800-555-1212 | 1        |
| 5          | Doreen  | Blue      | 20-MAY-70     | (null)       | 1        |
| 6          | Fred    | Brown     | 01-JAN-70     | 800-555-1215 | 1        |
| 4          | Gail    | Black     | (null)        | 800-555-1214 | 1        |
| 1          | John    | Brown     | 01-JAN-65     | 800-555-1211 | 1        |
| 3          | Steve   | White     | 16-MAR-71     | 800-555-1213 | 1        |

# Cláusula ORDER BY

- Um *número de posição* de *coluna* na *cláusula ORDER BY* pode ser *usado* para *indicar* qual *coluna* deve ser *classificada*
- Use **1** para *classificar* pela *primeira coluna*, **2** para *classificar* pela *segunda* e *assim sucessivamente*



# Cláusula ORDER BY

- **Exemplo (03):**

- A **coluna 1 (ID\_CLIENTE)** é **usada** para **classificar** as **linhas**

```
SELECT id_cliente, nome, sobrenome  
FROM tb_clientes  
ORDER BY 1;
```

| ID_CLIENTE | NOME    | SOBRENOME |
|------------|---------|-----------|
| 1          | John    | Brown     |
| 2          | Cynthia | Green     |
| 3          | Steve   | White     |
| 4          | Gail    | Black     |
| 5          | Doreen  | Blue      |
| 6          | Fred    | Brown     |



# JOIN

# SELECT (duas tabelas)

- *Permite obter informações de várias tabelas simultaneamente*
- *Exemplo:*
  - *Obter o nome do produto e o nome do tipo de produto*
  - As *tabelas tb\_produtos e tb\_tipos\_produtos* são *relacionadas* entre si *por meio* da *coluna* de *chave estrangeira ID\_TIPO\_PRODUTO*
  - A *coluna ID\_TIPO\_PRODUTO (chave estrangeira)* da *tabela tb\_produtos aponta para a coluna ID\_TIPO\_PRODUTO (chave primária)* da *tabela TIPOS\_PRODUTOS*

# SELECT (duas tabelas)

- *Exemplo (01):*

- *Recuperar as colunas NM\_PRODUTO e ID\_TIPO\_PRODUTO da tabela tb\_produtos para o produto de nº 3:*

```
SELECT nm_produto, id_tipo_produto  
FROM tb_produtos  
WHERE id_produto = 3;
```

| NM_PRODUTO | ID_TIPO_PRODUTO |
|------------|-----------------|
| Supernova  | 2               |

# SELECT (duas tabelas)

- *Exemplo (02):*

- Recuperar a coluna NM\_TIPO\_PRODUTO da tabela tb\_tipos\_produtos para o id\_tipo\_produto de nº 2:

```
SELECT nm_tipo_produto  
FROM tb_tipos_produtos  
WHERE id_tipo_produto = 2;
```

| NM_TIPO_PRODUTO |
|-----------------|
| Video           |

# SELECT (duas tabelas)

- *Recuperar o nome do produto e o nome de seu tipo de produto usando apenas uma única consulta*
- *Recurso:*
  - Utilize uma JOIN de tabela na consulta
- *Inclua as duas tabelas na cláusula FROM da consulta e inclua as colunas relacionadas de cada tabela na cláusula WHERE*

# SELECT (duas tabelas)

- *Exemplo:*

**FROM** tb\_produtos, tb\_tipos\_produtos

- E na *cláusula WHERE:*

**WHERE** tb\_produtos.id\_tipo\_produto = tb\_tipos\_produtos.id\_tipo\_produto

**AND** tb\_produtos.id\_produto = 3;

# SELECT (duas tabelas)

- JOIN: é a *primeira condição* da *cláusula WHERE*

tb\_produtos.id\_tipo\_produto = tb\_tipos\_produtos.id\_tipo\_produto

- Normalmente as *duas colunas* na JOIN são PK de *uma tabela* e uma FK da *outra tabela*
- Como o *operador* de *igualdade* ( = ) é *utilizado* na *condição JOIN*, esta JOIN é *nomeada* de EQUIJOIN
- Existe uma *coluna* ID\_TIPO\_PRODUTO na *tabela tb\_produtos* e outra na *tabela tb\_tipos\_produtos*, *necessitando* assim *informar* ao BD em *qual tabela* está a *coluna* que se *deseja utilizar*

# SELECT (duas tabelas)

- *Eventualmente*, se as *colunas tivessem nomes distintos*, **não** seria necessário informar os *nomes* das *tabelas* em *questão*
- Dica: *incluir sempre* os *nomes* das *tabelas* para *tornar claro* de onde *vêm* as *colunas*
- *Cláusula SELECT da consulta:*

```
SELECT tb_produtos.nm_produto,  
       tb_tipos_produtos.nm_tipo_produto
```

# SELECT (duas tabelas)

- *Exemplo (01):*

```
SELECT tb_produtos.nm_produto,  
        tb_tipos_produtos.nm_tipo_produto  
FROM tb_produtos, tb_tipos_produtos  
WHERE tb_produtos.id_tipo_produto = tb_tipos_produtos.id_tipo_produto  
AND tb_produtos.id_produto = 3;
```

| NM_PRODUTO | NM_TIPO_PRODUTO |
|------------|-----------------|
| Supernova  | Video           |

# SELECT (duas tabelas)

- *Exemplo (02):*

- *Obter todos os produtos ordenados pela coluna NM\_PRODUTO*

```
SELECT tb_produtos.nm_produto,  
       tb_tipos_produtos.nm_tipo_produto  
FROM tb_produtos, tb_tipos_produtos  
WHERE tb_produtos.id_tipo_produto = tb_tipos_produtos.id_tipo_produto  
ORDER BY tb_produtos.nm_produto;
```

| NM_PRODUTO          | NM_TIPO_PRODUTO |
|---------------------|-----------------|
| 2412: The Return    | Video           |
| Chemistry           | Book            |
| Classical Music     | CD              |
| Creative Yell       | CD              |
| From Another Planet | DVD             |
| Modern Science      | Book            |
| Pop 3               | CD              |
| Space Force 9       | DVD             |
| Supernova           | Video           |
| Tank War            | Video           |
| Z Files             | Video           |

# SELECT (duas tabelas)

- *Observação:*

- O *produto* cujo *nome corresponde* a “*My Front Line*” se *encontra ausente* do *resultado*
- O *valor correspondente* ao **ID\_TIPO\_PRODUTO** para essa *linha* de *produto* é *nulo* e a *condição* da JOIN *não retorna* a *linha*
- *Solução:*
  - *Uso* de JOINS EXTERNAS

# SELECT (duas tabelas)

- ***tb\_produtos:***

| ID_PRODUTO | ID_TIPO_PRODUTO | NM_PRODUTO          | DS_PRODUTO                               | PRECO | FG_ATIVO |
|------------|-----------------|---------------------|------------------------------------------|-------|----------|
| 6          | 2               | Z Files             | Series on mysterious activities          | 49.99 | 1        |
| 7          | 2               | 2412: The Return    | Aliens return                            | 14.95 | 1        |
| 8          | 3               | Space Force 9       | Adventures of heroes                     | 13.49 | 1        |
| 9          | 3               | From Another Planet | Alien from another planet lands on Earth | 12.99 | 1        |
| 10         | 4               | Classical Music     | The best classical music                 | 10.99 | 1        |
| 11         | 4               | Pop 3               | The best popular music                   | 15.99 | 1        |
| 12         | 4               | Creative Yell       | Debut album                              | 14.99 | 1        |
| 13         | (null)          | My Front Line       | Their greatest hits                      | 13.49 | 1        |
| 1          | 1               | Modern Science      | A description of modern science          | 19.95 | 1        |
| 2          | 1               | Chemistry           | Introduction to Chemistry                | 30    | 1        |
| 3          | 2               | Supernova           | A star explodes                          | 25.99 | 1        |
| 4          | 2               | Tank War            | Action movie about a future war          | 13.95 | 1        |

# SELECT (duas tabelas)

- A *sintaxe* de **JOIN** vista até o *presente momento* *utiliza* a *sintaxe* do *Oracle* para **JOINS** baseada no *padrão SQL/86* do ANSI
- *Oracle Database 9i e superiores (SQL/92 ANSI)*
- *Oracle Database 8i e inferiores (SQL/86 ANSI)*

# SELECT (duas tabelas)

- *Usando apelidos de tabela*
  - Inserir os *nomes* das *tabelas* (*tb\_produtos* e *tb\_tipos\_produtos*) nas *cláusulas* **SELECT** e **WHERE**
  - *Digitação redundante* (*usar apelidos* para *referenciar* as *tabelas*)
- *Exemplo:*
  - *Consulta usando apelido* “*p*” para a *tabela tb\_produtos* e “*tp*” para a *tabela tb\_tipos\_produtos*

```
SELECT p.nm_produto, tp.nm_tipo_produto  
FROM tb_produtos p, tb_tipos_produtos tp  
WHERE p.id_tipo_produto = tp.id_tipo_produto  
ORDER BY p.nm_produto;
```

# Produto Cartesiano

- A *ausência* da *condição JOIN* promove a *união* de *todas* as *linhas* de *uma tabela* com *todas* as *linhas* da *outra tabela*
- Esse *conjunto resultante* é *nomeado* de **PRODUTO CARTESIANO**
- *Exemplo:*
  - *Imagine* que *existisse* uma *tabela* *contendo 50 linhas* e uma *segunda tabela* *contendo 100 linhas*
  - Se for *selecionado* as *colunas* das *duas tabelas* **sem** uma JOIN, como *resultado, obteríamos 5.000 linhas* (*cada linha* da *tabela A* seria *juntada* a *cada linha* da *tabela B*)

# Produto Cartesiano

- **Exemplo:**

- Apresenta um **subconjunto** das *linhas* de um **produto cartesiano entre** as **tabelas tb\_produtos e tb\_tipos\_produtos**

```
SELECT p.id_tipo_produto, tp.id_tipo_produto  
FROM tb_produtos p, tb_tipos_produtos tp;
```

# SELECT (várias tabelas)

- JOINs podem ser *utilizadas* para *conectar “n” tabelas*
- *Exemplo:*
  - *Consulta complexa, fazendo uso de 4 tabelas, exigindo 3 JOINs*
  - *Recuperar as seguintes informações:*
    - As *compras* feitas por *cada cliente* (*tb\_compras*)
    - O *nome* e *sobrenome* do *cliente* (*tb\_clientes*)
    - O *nome* do *produto* que eles *compraram* (*tb\_produtos*)
    - O *nome* do *tipo* de *produto* (*tb\_tipos\_produtos*)

# SELECT (várias tabelas)

- **Exemplo:**

- Para **obter** essas **informações**, será **necessário consultar** as **tabelas tb\_compras, tb\_clientes, tb\_produtos e tb\_tipos\_produtos**
- As **JOINS usarão** os **relacionamentos** de **FK entre** essas **tabelas**

```
SELECT c.nome, c.sobrenome, p.nm_produto AS produto,  
       tp.nm_tipo_produto AS tipo  
FROM tb_clientes c, tb_compras co, tb_produtos p, tb_tipos_produtos tp  
WHERE c.id_cliente = co.id_cliente AND  
      p.id_produto = co.id_produto AND  
      p.id_tipo_produto = tp.id_tipo_produto  
ORDER BY p.nm_produto;
```

# Tipos de JOIN

- *Existem dois tipos de condição de JOIN, as quais são baseadas no operador utilizado na JOIN:*
  - EQUIJOINS:
    - Utilizam o operador de igualdade ( = )
  - NÃO-EQUIJOINS:
    - Utilizam operadores que não correspondem ao de igualdade (<, >, BETWEEN, etc)

# Tipos de JOIN

- *Especificação* de junções (JOINS):
  - JOINS INTERNAS:
    - *Retornam* uma *linha somente* quando as *colunas* da JOIN *contêm valores* que *satisfazem* essa *condição*
    - Se *uma linha possui um valor nulo* em uma das *colunas* na *condição* de JOIN, ela **não** é *retornada*
  - JOINS EXTERNAS:
    - *Retornam* uma *linha mesmo* quando *uma das colunas* na *condição* de JOIN *contém* um *valor nulo*
  - AUTOJOINS:
    - *Retornam linhas unidas* na *mesma tabela*

# Não-Equijoin

- Utiliza um *operador* que **NÃO** é o de *igualdade* na JOIN
- *Operadores:*
  - *diferente* ( $<>$ )
  - *menor* que ( $<$ )
  - *maior* que ( $>$ )
  - *menor* ou *igual* a ( $\leq$ )
  - *maior* ou *igual* a ( $\geq$ )
  - *LIKE*
  - *IN*
  - *BETWEEN*

# Não-Equijoin

- *Exemplo:*
  - Obter os *níveis salariais* dos *funcionários*

```
SELECT *  
FROM tb_grades_salarios;
```

| ID_SALARIO | BASE_SALARIO | TETO_SALARIO | FG_ATIVO |
|------------|--------------|--------------|----------|
| 1          | 1            | 250000       | 1        |
| 2          | 250001       | 500000       | 1        |
| 3          | 500001       | 750000       | 1        |
| 4          | 750001       | 999999       | 1        |

# Não-Equijoin

- **Exemplo:**

- Consulta utiliza uma **não-equijoin** para **recuperar** o **salário** e os **níveis** dos **funcionários**; o **nível salarial** é **determinado** pelo **operador BETWEEN**:

```
SELECT f.nome, f.sobrenome, f.cargo, f.salario, gs.id_salario  
FROM tb_funcionarios f, tb_grades_salarios gs  
WHERE f.salario BETWEEN gs.base_salario AND gs.teto_salario  
ORDER BY gs.id_salario;
```

# Joins-Externas

- *Join externa recupera uma linha mesmo quando uma de suas colunas contém um valor nulo*
- *Operador de join externa (Oracle):*
  - *Sinal de adição entre parênteses (+)*

# Joins-Externas

- **Exemplo:**

- O *produto* “*My Front Line*” cujo *ID\_TIPO\_PRODUTO* é *nulo*
- O *operador* (+) esta no *lado oposto* da *coluna ID\_TIPO\_PRODUTO* na *tabela tb\_produtos* (*coluna que contém o valor nulo*)

```
SELECT p.nm_produto AS produto, tp.nm_tipo_produto AS tipo  
FROM tb_produtos p, tb_tipos_produtos tp  
WHERE p.id_tipo_produto = tp.id_tipo_produto (+)  
ORDER BY 1;
```

| PRODUTO             | TIPO   |
|---------------------|--------|
| 2412: The Return    | Video  |
| Chemistry           | Book   |
| Classical Music     | CD     |
| Creative Yell       | CD     |
| From Another Planet | DVD    |
| Modern Science      | Book   |
| My Front Line       | (null) |
| Pop 3               | CD     |
| Space Force 9       | DVD    |
| Supernova           | Video  |
| Tank War            | Video  |
| Z Files             | Video  |

# Joins-Externas (direita e esquerda)

- As **JOINs externas** podem ser *divididas* em *dois tipos*:
  - *Join externa esquerda*
  - *Join externa direita*
- **Sintaxe:**

**SELECT ...**

**FROM tabela1, tabela2**

...

- **Suponha** que as **tabelas** serão **unidas** em **tabela1.coluna1** e **tabela2.coluna2**
- **Suponha** que a **tabela1** contenha uma **linha** com um **valor nulo** na **coluna1**

# Joins-Externas (direita e esquerda)

- Para *realizar* uma **JOIN externa ESQUERDA**, a **cláusula WHERE** fica:

WHERE tabela1.coluna1 = tabela2.coluna2 **(+)**

- *Observação:*
  - Em uma *join externa ESQUERDA*, o *operador* de *join externa* fica, na *verdade*, à **DIREITA** do *operador*

# Joins-Externas (direita e esquerda)

- Suponha que a *tabela2* contenha uma *linha* com *um valor NULO* na *coluna2*
- *Join externa DIREITA*, basta *trocar* a *posição* do *operador* de *join externa* para a *ESQUERDA* do *operador* de *igualdade*:

WHERE tabela1.coluna1 **(+)** = tabela2.coluna2

# Joins-Externas (direita e esquerda)

- **Exemplo:**

- **Join externa ESQUERDA**
- As *linhas* da *tabela tb\_produtos* são *recuperadas, incluindo* a *linha "My Front Line"*, a qual *possui* um *valor NULO* na *coluna ID\_TIPO\_PRODUTO*

```
SELECT p.nm_produto AS produto, tp.nm_tipo_produto AS tipo  
FROM tb_produtos p, tb_tipos_produtos tp  
WHERE p.id_tipo_produto = tp.id_tipo_produto (+)  
ORDER BY 1;
```

| PRODUTO             | TIPO   |
|---------------------|--------|
| 2412: The Return    | Video  |
| Chemistry           | Book   |
| Classical Music     | CD     |
| Creative Yell       | CD     |
| From Another Planet | DVD    |
| Modern Science      | Book   |
| My Front Line       | (null) |
| Pop 3               | CD     |
| Space Force 9       | DVD    |
| Supernova           | Video  |
| Tank War            | Video  |
| Z Files             | Video  |

# Joins-Externas (direita e esquerda)

- **Exemplo:**

- *Join externa DIREITA*
- A *tabela tb\_tipos\_produtos contém um tipo de produto não referenciado na tabela tb\_produtos*
- *Não existe produtos do tipo “Magazine” na tabela tb\_produtos*

```
SELECT *  
FROM tb_tipos_produtos;
```

| ID_TIPO_PRODUTO | NM_TIPO_PRODUTO | FG_ATIVO |
|-----------------|-----------------|----------|
| 1               | Book            | 1        |
| 2               | Video           | 1        |
| 3               | DVD             | 1        |
| 4               | CD              | 1        |
| 5               | Magazine        | 1        |

# Joins-Externas (direita e esquerda)

- ***Exemplo:***

- ***Join externa DIREITA***
- ***Recuperar o tipo de produto “Magazine” através de uma junção externa DIREITA realizada entre as tabelas tb\_produtos e tb\_tipos\_produtos***

```
SELECT p.nm_produto AS produto, tp.nm_tipo_produto AS tipo
FROM tb_produtos p, tb_tipos_produtos tp
WHERE p.id_tipo_produto (+) = tp.id_tipo_produto
ORDER BY 1;
```

| PRODUTO             | TIPO     |
|---------------------|----------|
| 2412: The Return    | Video    |
| Chemistry           | Book     |
| Classical Music     | CD       |
| Creative Yell       | CD       |
| From Another Planet | DVD      |
| Modern Science      | Book     |
| Pop 3               | CD       |
| Space Force 9       | DVD      |
| Supernova           | Video    |
| Tank War            | Video    |
| Z Files             | Video    |
| (null)              | Magazine |

# Limitações das Joins-Externas

- *Exemplo (01):*

- **Não** é *possível* colocar o *operador de join externa* em *ambos os lados*

```
SELECT p.nm_produto AS produto, tp.nm_tipo_produto AS tipo  
FROM tb_produtos p, tb_tipos_produtos tp  
WHERE p.id_tipo_produto (+) = tp.id_tipo_produto (+)  
ORDER BY 1;
```

# Limitações das Joins-Externas

- *Exemplo (02):*

- **Não** é possível usar uma *condição de join externa* na JOIN que esteja *usando o operador OR*

```
SELECT p.nm_produto AS produto, tp.nm_tipo_produto AS tipo  
FROM tb_produtos p, tb_tipos_produtos tp  
WHERE p.id_tipo_produto (+) = tp.id_tipo_produto  
OR p.id_tipo_produto = 1;
```

# Autojoins

- É *realizada na mesma tabela (recursiva)*
- *Necessidade de utilizar apelidos (alias) distintos para identificar cada referência à tabela*
- *Exemplo:*
  - A *tabela tb\_funcionarios* possui uma *coluna ID\_GERENTE* para cada *funcionário*
  - Se o *funcionário não possuir gerente*, o *ID\_GERENTE* é *nulo*

# Autojoins

- *Tabela tb\_funcionarios:*

| ID_FUNCIONARIO | ID_GERENTE | NOME   | SOBRENOME | CARGO           | SALARIO | FG_ATIVO |
|----------------|------------|--------|-----------|-----------------|---------|----------|
| 1              | (null)     | James  | Smith     | CEO             | 800000  | 1        |
| 2              | 1          | Ron    | Johnson   | Sales Manager   | 600000  | 1        |
| 3              | 2          | Fred   | Hobbs     | Salesperson     | 150000  | 1        |
| 4              | 2          | Susan  | Jones     | Salesperson     | 500000  | 1        |
| 5              | 2          | Rob    | Green     | Sales Person    | 400000  | 1        |
| 6              | 4          | Jane   | Brown     | Support Person  | 450000  | 1        |
| 7              | 4          | John   | Grey      | Support Manager | 300000  | 1        |
| 8              | 7          | Jean   | Blue      | Support Person  | 290000  | 1        |
| 9              | 6          | Henry  | Heyson    | Support Person  | 300000  | 1        |
| 10             | 1          | Kevin  | Black     | Ops Manager     | 100000  | 1        |
| 11             | 10         | Keith  | Long      | Ops Person      | 500000  | 1        |
| 12             | 10         | Frank  | Howard    | Ops Person      | 450000  | 1        |
| 13             | 10         | Doreen | Penn      | Ops Person      | 470000  | 1        |

# Autojoins

- **Exemplo (01):**

- *James Smith (diretor executivo CEO)* tem um **valor nulo** para o **ID\_GERENTE**
- *Utilizar a autojoin para exibir os nomes de cada funcionário e seu respectivo gerente*

```
SELECT f.nome || ' ' || f.sobrenome || ' trabalha para ' || g.nome  
FROM tb_funcionarios f, tb_funcionarios g  
WHERE f.id_gerente = g.id_funcionario  
ORDER BY f.nome;
```

# Autojoins

- ***Exemplo (01):***

- *James Smith (diretor executivo CEO)* tem um **valor nulo** para o **ID\_GERENTE**
- Utilizar a **autojoin** para **exibir os nomes de cada funcionário e seu respectivo gerente**

|  | F.NOME  "  F.SOBRONOME  'TRABALHAPARA'  G.NOME |
|--|------------------------------------------------|
|  | Doreen Penn trabalha para Kevin                |
|  | Frank Howard trabalha para Kevin               |
|  | Fred Hobbs trabalha para Ron                   |
|  | Henry Heyson trabalha para Jane                |
|  | Jane Brown trabalha para Susan                 |
|  | Jean Blue trabalha para John                   |
|  | John Grey trabalha para Susan                  |
|  | Keith Long trabalha para Kevin                 |
|  | Kevin Black trabalha para James                |
|  | Rob Green trabalha para Ron                    |
|  | Ron Johnson trabalha para James                |
|  | Susan Jones trabalha para Ron                  |

- **Exemplo (02):**

- *Como* o ID\_GERENTE de *James Smith* é *nulo*, a *condição não retorna a tupla*
- *Função NVL* é *utilizada* para *incluir* um *string sinalizando* que *Smith trabalha* para os *acionistas*

```
SELECT f.nome || ' trabalha para ' || NVL(g.sobrenome, 'os acionistas')  
FROM tb_funcionarios f, tb_funcionarios g  
WHERE f.id_gerente = g.id_funcionario (+)  
ORDER BY f.sobrenome DESC;
```

# Autojoins

- **Exemplo (02):**

- **Como** o ID\_GERENTE de *James Smith* é **nulo**, a **condição não retorna a tupla**
- **Função NVL** é **utilizada** para **incluir** um **string sinalizando** que *Smith trabalha* para os *acionistas*

```
F.NOME||'TRABALHAPARA'||NVL(G.SOBRENOME,'OSACIONISTAS')
James trabalha para os acionistas
Doreen trabalha para Black
Keith trabalha para Black
Susan trabalha para Johnson
Ron trabalha para Smith
Frank trabalha para Black
Fred trabalha para Johnson
Henry trabalha para Brown
John trabalha para Jones
Rob trabalha para Johnson
Jane trabalha para Jones
Jean trabalha para Grey
Kevin trabalha para Smith
```

# Join (SQL/92)

- Os *exemplos* de JOIN vistos *até agora exploram o padrão SQL/86*
- *Oracle 9i:*
  - Utiliza o *padrão SQL/92 ANSI* para JOINs
- *Observação:*
  - Utilizar o *padrão SQL/92* em *versões superiores* a *9i* para *evitar produtos cartesianos indesejáveis*

# Join (SQL/92)

- O *padrão SQL/92* *introduz* as *cláusulas INNER JOIN e ON* para *realizar* uma **JOIN INTERNA**
- **Exemplo (SQL/92):**

```
SELECT p.nm_produto AS PRODUTO, tp.nm_tipo_produto AS TIPO
FROM tb_produtos p
INNER JOIN tb_tipos_produtos tp ON (p.id_tipo_produto = tp.id_tipo_produto)
ORDER BY p.nm_produto;
```

# Join (SQL/92)

- *Utilizar operadores de NÃO-EQUIJOIN com a cláusula ON*
- **Exemplo (SQL/92):**

```
SELECT f.nome, f.sobrenome, f.cargo, f.salario, gs.id_salario  
FROM tb_funcionarios f  
INNER JOIN tb_grades_salarios gs ON (f.salario BETWEEN gs.base_salario  
   AND gs.teto_salario)  
ORDER BY gs.id_salario;
```

# Joins com USING

- SQL/92 permite *simplificar* ainda mais a *condição* de JOIN com a *cláusula USING*
- *Limitações:*
  - A *consulta* deve *usar* uma EQUIJOIN
  - As *colunas* na EQUIJOIN devem ter o *mesmo NOME*
- *Exemplo:*
  - *Uso* de USING substituindo ON

```
SELECT p.nm_produto AS PRODUTO, tp.nm_tipo_produto AS TIPO  
FROM tb_produtos p  
INNER JOIN tb_tipos_produtos tp  
USING (id_tipo_produto);
```

# Joins com USING

- Caso seja necessário exibir o ID\_TIPO\_PRODUTO, será necessário somente fornecer o nome da coluna sozinho, sem um nome ou apelido da tabela na cláusula SELECT:

```
SELECT p.nm_produto AS PRODUTO,  
       tp.nm_tipo_produto AS TIPO, id_tipo_produto  
FROM tb_produtos p  
INNER JOIN tb_tipos_produtos tp  
USING (id_tipo_produto);
```

# Joins com USING

- O *nome* da *coluna dentro* da *cláusula USING* deverá estar *sozinho*
- *Exemplo:*
  - **USING(p.id\_tipo\_produto)**

```
SELECT p.nm_produto AS PRODUTO,  
       tp.nm_tipo_produto AS TIPO, id_tipo_produto  
FROM tb_produtos p  
INNER JOIN tb_tipos_produtos tp  
USING (p.id_tipo_produto);
```

# Join (SQL/92)

- JOINs *internas* em *mais* de *duas tabelas* usando SQL/92
- *Exemplo (SQL/92):*

```
SELECT c.nome, c.sobrenome, p.nm_produto AS produto,  
       tp.nm_tipo_produto AS tipo  
FROM tb_clientes c  
INNER JOIN tb_compras co USING (id_cliente)  
INNER JOIN tb_produtos p USING (id_produto)  
INNER JOIN tb_tipos_produtos tp USING (id_tipo_produto)  
ORDER BY p.nm_produto;
```

# Join (SQL/92)

- *Realizando JOINs internas em várias colunas*
  - Se uma JOIN *utiliza mais de uma coluna* nas *duas tabelas, forneça* essas *colunas* em sua *cláusula ON, utilizando, paralelamente, o operador AND*
  - *Exemplo:*
    - *Temos duas tabelas nomeadas de tabela1 e tabela2, necessitando de juntá-las usando as colunas nomeadas de coluna1 e coluna2 em ambas as tabelas*
- SELECT ...**
- FROM tabela1**
- INNER JOIN tabela2 ON (tabela1.coluna1 = tabela2.coluna1**
- AND tabela1.coluna2 = tabela2.coluna2);**

# Join (SQL/92)

- *Realizando JOINs internas em várias colunas*
- Para *simplificar, poderíamos acrescentar na consulta a cláusula USING, mas, somente se estiver realizando EQUIJOIN e os nomes das colunas forem idênticos*
- *Exemplo:*

```
SELECT ...
FROM tabela1
INNER JOIN tabela2
USING (coluna1, coluna2);
```

# Joins Externas (SQL/92)

- Até o *presente momento*, utilizamos o *operador de join externa (+)* (*Oracle*)

- *Sintaxe (Joins Externas):*

**FROM** tabela1 {**LEFT** | **RIGHT** | **FULL**}

**OUTER JOIN** tabela2

- *Onde:*

- *tabela1* e *tabela2* são as *tabelas pertinentes* a junção
- **LEFT**: realiza uma *JOIN externa esquerda*
- **RIGHT**: realiza uma *JOIN externa direita*
- **FULL**: realiza uma *JOIN externa integral*

# Joins Externas (SQL/92)

- *Exemplo (LEFT OUTER JOIN):*

```
SELECT p.nm_produto AS produto, tp.nm_tipo_produto AS tipo
FROM tb_produtos p
LEFT OUTER JOIN tb_tipos_produtos tp USING (id_tipo_produto)
ORDER BY p.nm_produto;
```

# Joins Externas (SQL/92)

- *Exemplo (RIGHT OUTER JOIN):*

```
SELECT p.nm_produto AS produto, tp.nm_tipo_produto AS tipo
  FROM tb_produtos p
RIGHT OUTER JOIN tb_tipos_produtos tp USING (id_tipo_produto)
 ORDER BY p.nm_produto;
```

# Joins Externas (SQL/92)

- *Exemplo (**FULL OUTER JOIN**):*

```
SELECT p.nm_produto AS produto, tp.nm_tipo_produto AS tipo
  FROM tb_produtos p
 FULL OUTER JOIN tb_tipos_produtos tp USING (id_tipo_produto)
 ORDER BY p.nm_produto;
```

# Auto Join (SQL/92)

- *Exemplo (INNER JOIN):*

```
SELECT f.nome || ' ' || f.sobrenome || ' trabalha para' || g.nome  
FROM tb_funcionarios f  
INNER JOIN tb_funcionarios g ON(f.id_gerente = g.id_funcionario)  
ORDER BY f.nome;
```

# Join Cruzada (SQL/92)

- JOIN entre duas tabelas resulta em um *produto cartesiano*
- Usando a sintaxe de JOIN, evitamos a produção acidental de um *produto cartesiano*
  - Uso da cláusula ON ou USING
- Forçando um *produto cartesiano* (uso explícito das palavras-chave CROSS JOIN)
- Exemplo:

```
SELECT *
FROM tb_tipos_produtos
CROSS JOIN tb_produtos;
```



# Usando Variáveis

# Usando Variáveis

- *Criar variáveis para serem utilizadas no lugar de valores reais (SQL)*
- *Variáveis nomeadas de variáveis de substituição*
- *Existem dois tipos de variáveis de substituição:*
  - Variáveis Temporárias:
    - *Possui validade apenas para a instrução SQL em que é usada*
  - Variáveis Definitivas:
    - *Possui validade até sua remoção explícita*

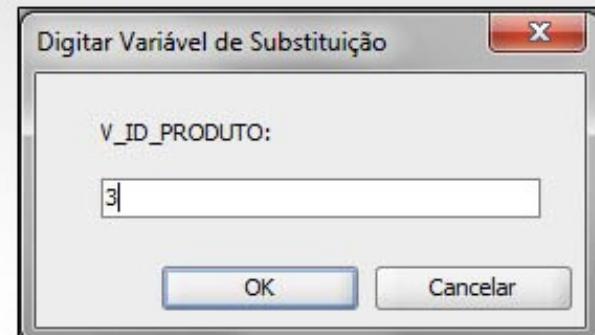
# Variáveis Temporárias

- A *definição* de uma *variável temporária* é *imposta* pelo *uso* do *caractere “E” comercial (&)*
- *Sintaxe:*  
**&identificador\_variável**
- *Exemplo:*  
**&v\_id\_produto**
  - (*define* uma *variável nomeada* de *v\_id\_produto*)

# Variáveis Temporárias

- A *consulta SQL solicita* a *inserção de um valor* para **v\_id\_produto**, utilizando esse valor **posteriormente** na **cláusula WHERE**
- **Exemplo:**

```
SELECT id_produto, nm_produto, preco  
FROM tb_produtos  
WHERE id_produto = &v_id_produto;
```



| ID_PRODUTO | NM_PRODUTO | PRECO |
|------------|------------|-------|
| 3          | Video      | 25,99 |

# Variáveis Temporárias

- *Substituindo nome de tabela e colunas usando variáveis*
- *Exemplo:*
  - Definição de **variáveis** para um **nome** de **coluna** (**v\_coluna**), um **nome** de **tabela** (**v\_tabela**) e um **valor** de **coluna** (**v\_id\_produto**)

```
SELECT nm_produto, &v_coluna  
FROM &v_tabela  
WHERE &v_coluna = &v_id_produto;
```

# Variáveis Temporárias

- *Utilizando* o **&&** podemos *evitar* a *digitação repetitiva* de uma **determinada variável**
- *Exemplo:*

```
SELECT nm_produto, &&v_coluna  
FROM &v_tabela  
WHERE &&v_coluna = &v_id_produto;
```

# Variáveis Definidas

- *Permite a definição da variável antes de sua utilização*
- *Persiste* até que seja *removida explicitamente*
- *DEFINE*: comando *utilizado* para *definir* uma *variável*
- *UNDEFINE*: comando *utilizado* para *remover* uma *variável*
- *Utilizadas* para *geração* de *relatórios simples*

# Variáveis Definidas

- *Exemplo:*

- *Definir uma variável nomeada v\_id\_produto, atribuindo o valor 7*

```
DEFINE v_id_produto = 7;
```

```
SELECT nm_produto, id_produto  
FROM tb_produtos  
WHERE id_produto = &v_id_produto;
```

# Variáveis Definidas

- *Definindo e configurando variáveis com o comando ACCEPT*
- *ACCEPT: permite que o usuário digite um valor para ser atribuído a uma determinada variável*
- *Permite especificar o tipo e o valor inicial*
- *Exemplo:*

ACCEPT nome\_variável [tipo][FORMAT formato]  
[PROMPT *prompt*] [HIDE]

# Variáveis Definidas

- **nome\_variável**: é o *identificador* da **variável**
- **tipo**: *tipo de dado utilizado* (CHAR, NUMBER e DATE). Por *padrão*, as **variáveis** são *definidas* com o **tipo CHAR**
- **formato**: *formato usado* para a **variável**
  - *Exemplos*: A15 (15 *caracteres*), 9999 (um *número* de 4 *dígitos*) e DD-MM-YYYY (uma *data*)
- **prompt**: *texto exibido* pelo **console** como **prompt** para o *usuário* *digitar* o **valor** da **variável**
- **hide**: permite *ocultar* o **valor** quando ele é *digitado*
  - *Exemplos*: *senhas* e ou *informações confidenciais*

# Variáveis Definidas

- *Exemplo:*

- Define uma *variável nomeada* de **v\_id** como um *número* de no *máximo dois dígitos*

```
ACCEPT v_id NUMBER FORMAT 99 PROMPT 'Entre com o ID';
```

```
SELECT id_produto, nm_produto, preco  
FROM tb_produtos  
WHERE id_produto = &v_id;
```

# Removendo Variáveis

- **UNDEFINE:** comando utilizado para *remover variáveis*
- **Exemplo:**
  - Uso do comando **UNDEFINE** para *remover* uma **variável**

```
DEFINE v_id_produto = 7;
```

```
SELECT nm_produto, id_produto  
FROM tb_produtos  
WHERE id_produto = &v_id_produto;
```

```
UNDEFINE v_id_produto;
```

# Relatórios Simples

- *Possibilidade de gerar relatórios usando variáveis em um script SQL*
- *Não permite gerar relatórios complexos (Oracle Reports)*
- *Usando variáveis temporárias em um script*
  - *script: teste\_1.sql*
  - uso de uma *variável temporária nomeada* de “*v\_id\_produto*” na *cláusula WHERE*

# Relatórios Simples

- *Exemplo (01):*
  - Criando o script identificado de **teste\_1.sql**
  - Armazenar o script em \home\oracle

```
SET ECHO OFF
```

```
SET VERIFY OFF
```

```
SELECT id_produto, nm_produto, preco  
FROM tb_produtos  
WHERE id_produto = &v_id_produto;
```

# Relatórios Simples

- ***Invocando o script:***

**@ \home\oracle\teste\_1.sql**

Executar Script  
(F5)

- ***Observação:***

– *Espaços nos **nomes** dos **diretórios** (*inserir entre aspas tudo após o @*)*



# Relatórios Simples

- ***Exemplo (02):***

- O **script abaixo** ( teste\_2.sql ) utiliza o **comando ACCEPT** para **definir** uma **variável** chamada **v\_id\_produto**

-- suprime a exibição das instruções e mensagens de verificação

**SET ECHO OFF**

**SET VERIFY OFF**

**ACCEPT v\_id\_produto NUMBER FORMAT 99 PROMPT 'Entre com o ID do produto: ';**

**SELECT id\_produto, nm\_produto, preco**

**FROM tb\_produtos**

**WHERE id\_produto = &v\_id\_produto;**

# Relatórios Simples

- *Exemplo (02):*
  - *Invocando o script:*  
**@ \home\oracle\teste\_2.sql**

# Relatórios Simples

- ***Exemplo (03):***

- *Passando um valor para uma variável em um script*
  - *Referenciar a variável no script utilizando (&1)*
  - O *script abaixo* será *identificado* como **teste\_3.sql**
    - suprime a exibição das instruções e mensagens de verificação

**SET ECHO OFF**

**SET VERIFY OFF**

```
SELECT id_produto, nm_produto, preco
FROM tb_produtos
WHERE id_produto = &1;
```

# Relatórios Simples

- *Exemplo (03):*
  - *Invocando o script:*

`@ \home\oracle\teste_3.sql 2`



*1ª variável*

# Relatórios Simples

- ***Exemplo (04):***

- O *script abaixo* será ***identificado*** como teste\_4.sql

- suprime a exibição das instruções e mensagens de verificação

```
SET ECHO OFF
```

```
SET VERIFY OFF
```

```
SELECT id_produto, nm_produto, preco
```

```
FROM tb_produtos
```

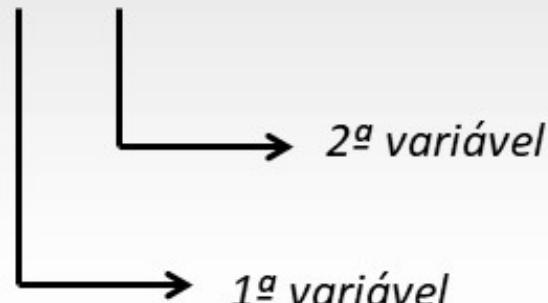
```
WHERE id_produto = &1
```

```
AND preco > &2;
```

# Relatórios Simples

- *Exemplo (04):*
  - *Invocando o script:*

**@ \home\oracle\teste\_4.sql 6 19.99**



# Gerando Instruções SQL

- *Permite gerar instruções SQL automaticamente*
- *Exemplo:*
  - *Produzir instruções DROP TABLE (remoção de tabelas)*
  - *Constitui uma série de instruções DROP TABLE que removem as tabelas do esquema LOJA*

```
SELECT 'DROP TABLE ' || table_name || ';'  
FROM user_tables;
```



# Exercícios



- ***Exercício (01):***
  - ***Criar um bloco anônimo*** usando **PL/SQL**
  - **Objetivo:** *realizar uma carga de dados*
  - ***Observações (Bloco Anônimo):***
    - **Não** possui *nome*
    - **Não** encontra-se *armazenado* no **SGBD**

- **Exercício (01):**

```
CREATE TABLE tb_teste(  
    ID      INTEGER,  
    valor   VARCHAR2(100));
```

```
BEGIN
```

```
    FOR v_loop IN 1..100000 LOOP  
        INSERT INTO tb_teste(ID, valor)  
        VALUES (v_loop, 'DBA_' || v_loop);  
    END LOOP;
```

```
END;
```

**Continua....**

- *Exercício (01) - continuação:*

```
SELECT *  
FROM tb_teste  
ORDER BY 1;
```

```
TRUNCATE TABLE tb_teste;
```

- ***Exercício (02):***
  - Criar um *stored procedure nomeado de manipula\_dados*
  - Objetivo: realizar *instruções* de **INSERT**, **UPDATE** e **DELETE** em uma *determinada tabela* por meio da *passagem de parâmetros*
  - *Promove facilidade na manutenção e gerenciamento das instruções DML aplicadas às tabelas constituintes do esquema do BD*

- ***Exercício (02) - Parte 1:***

```
CREATE TABLE tb_cliente_teste(  
    id_cliente      INTEGER,  
    ds_cliente      VARCHAR2(40),  
    nm_cliente      VARCHAR2(40),  
    valor           NUMERIC,  
    fg_ativo        INTEGER,  
    PRIMARY KEY(id_cliente)  
);
```

*continua....*

- ***Exercício (02) - Parte 2:***

```
CREATE OR REPLACE PROCEDURE manipula_dados(  
    p_id_cliente IN tb_cliente_teste.id_cliente%TYPE,  
    p_descricao  IN tb_cliente_teste.ds_cliente%TYPE,  
    p_nome       IN tb_cliente_teste.nm_cliente%TYPE,  
    p_valor      IN tb_cliente_teste.valor%TYPE,  
    p_fg_ativo   IN tb_cliente_teste.fg_ativo%TYPE,  
    p_opcao      IN CHAR)  
  
AS  
    v_controle INTEGER;
```

*continua....*

- ***Exercício (02) - Parte 3:***

**BEGIN**

```
-- verifica a existência de tuplas na tb_cliente_teste
SELECT COUNT(*) INTO v_controle
FROM tb_cliente_teste
WHERE id_cliente = p_id_cliente
AND fg_ativo = 1;
```

*continua....*

- **Exercício (02) - Parte 4:**

```
-- opcao = I (INSERT)
IF (p_opcao = 'I') THEN
    IF (v_controle != 1) THEN
        INSERT INTO tb_cliente_teste(id_cliente, ds_cliente, nm_cliente, valor,
                                       fg_ativo)
        VALUES (p_id_cliente, p_descricao, p_nome, p_valor, p_fg_ativo);
        COMMIT;
        dbms_output.put_line('Cliente inserido com sucesso');

    ELSE
        dbms_output.put_line('ID do cliente já existe');
    END IF;
END IF;
```

*continua....*

- **Exercício (02) - Parte 5:**

```
-- opcao = U (UPDATE)
IF (p_opcao = 'U') THEN
  IF (v_controle = 1) THEN
    UPDATE tb_cliente_teste SET ds_cliente      = p_descricao,
                                nm_cliente       = p_nome,
                                valor            = p_valor,
                                fg_ativo         = p_fg_ativo
    WHERE id_cliente = p_id_cliente
    AND fg_ativo = 1;

  COMMIT;
  dbms_output.put_line('Cliente alterado com sucesso');
```

*continua....*

- **Exercício (02) - Parte 6:**

**ELSE**

```
    dbms_output.put_line('ID do cliente não existe');
```

**END IF;**

**END IF;**

-- opcao = D (DELETE)

**IF (p\_opcao = 'D') THEN**

**IF (v\_controle = 1) THEN**

**DELETE**

**FROM tb\_cliente\_teste**

**WHERE id\_cliente = p\_id\_cliente**

**AND fg\_ativo = 1;**

**COMMIT;**

*continua....*

- ***Exercício (02) - Parte 7:***

```
dbms_output.put_line('Cliente excluído com sucesso');
```

```
ELSE
```

```
    dbms_output.put_line('ID do cliente não existe');
```

```
END IF;
```

```
END IF;
```

```
EXCEPTION
```

```
    WHEN OTHERS THEN
```

```
        ROLLBACK;
```

```
END manipula_dados;
```

- **Exercício (02) - Parte 8:**

-- opcao inserir (parâmetro opcao = I)

```
CALL manipula_dados(1, 'Cliente 1', 'Nome Cliente 1', 22.33, 1, 'I');
```

-- opcao inserir (parâmetro opcao = I)

```
CALL manipula_dados(2, 'Cliente 2', 'Nome Cliente 2', 99.99, 1, 'I');
```

-- opcao alterar (parâmetro opcao = U)

```
CALL manipula_dados(2, 'Cliente - Alterado hoje', 'Alterado', 99.99, 1, 'U');
```

-- opcao excluir (parâmetro opcao = D)

```
CALL manipula_dados(2, NULL, NULL, NULL, NULL, 'D');
```