

(BD) SQL Teórica

From WikiNote

Contents

- 1 Alguns comandos SQL
 - 1.1 Chaves
 - 1.2 Instrução SELECT Básica
 - 1.3 Alias para coluna
 - 1.4 Operador de concatenação
 - 1.4.1 String de Carateres Literais
 - 1.5 DISTINCT
 - 1.6 Restringir e ordenar dados
 - 1.6.1 Usar o WHERE
 - 1.7 Strings de carateres e datas
 - 1.8 Outros operadores de comparação
 - 1.9 Operadores Lógicos
 - 1.10 Regras de precedência
 - 1.11 ORDER BY
 - 1.12 Funções de linha
 - 1.13 Funções de strings
 - 1.13.1 SUBSTR
 - 1.13.2 Separar nomes contendo mais do que 1 espaço
 - 1.14 Diferenças entre TO_CHAR e TO_DATE
 - 1.15 Subtrair Datas
 - 1.16 Função TO_CHAR com Data
 - 1.16.1 Função TO_CHAR com Data para mostrar os dias e semana
 - 1.16.2 FONTE do TO_CHAR
 - 1.17 Função TO_CHAR com números
 - 1.18 Função NVL
 - 1.18.1 Exemplo
 - 1.19 Função DECODE
 - 1.20 Funções aninhadas
- 2 Joins
 - 2.1 O que é um join ?
 - 2.2 Produto cartesiano
 - 2.3 Equijoin
 - 2.3.1 Usando o operador AND
 - 2.3.2 Simplificar pesquisas com alias de tabela
 - 2.3.3 Joining de mais de duas tabelas
 - 2.4 Non-Equijoin
 - 2.5 Outer join
 - 2.5.1 Syntax
 - 2.6 Self join
- 3 Agregação de Dados Usando Funções de Grupo
 - 3.1 Tipos de funções de grupo
 - 3.2 Syntax das funções de grupo
 - 3.3 Funções AVG e SUM
 - 3.4 Funções MIN e MAX
 - 3.5 Função COUNT
 - 3.6 Funções de grupo e Nulos
 - 3.7 Criar Grupos de Dados
 - 3.7.1 Clausula GROUP BY
 - 3.7.2 Group By por Mais de uma Coluna
 - 3.7.3 Queries Ilegais usando Funções de Grupo
 - 3.8 Excluir Resultados de Grupo
 - 3.8.1 Having
 - 3.8.2 Exemplo de exclusão
 - 3.8.3 2º Exemplo de exclusão
 - 3.9 Aninhar Funções de Grupo
- 4 Ordem de cálculo de clausulas
- 5 Operadores de Conjuntos
 - 5.1 União
 - 5.1.1 Exemplos com o UNION
 - 5.2 Interseção
 - 5.2.1 Exemplos com o INTERSECT
 - 5.3 Diferença
 - 5.3.1 Exemplo com MINUS
- 6 Subqueries
 - 6.1 Guias para Usar Subqueries
 - 6.2 Tipos de subqueries
 - 6.3 Subqueries de Linha Única
 - 6.3.1 Exemplo de Subquerie de Linha Única
 - 6.3.2 Usar Funções de Grupo numa Subquery
 - 6.3.3 Clausula HAVING com Subqueries
 - 6.4 Subqueries de Várias Linhas
 - 6.4.1 Usar o Operador ANY em Subqueries de Várias Linhas
 - 6.4.2 Usar o Operador ALL em Subqueries de Várias Linhas

- 6.5 Subqueries de Várias Colunas
 - 6.5.1 Subquery de Comparação Par
 - 6.5.2 Subquery de Comparação Não Par
- 6.6 Valores Nulos numa Subquery
- 6.7 Subconsultas Correlacionadas
 - 6.7.1 Características
 - 6.7.2 Ordem de execução de uma Subconsulta correlacionada
 - 6.7.3 Exemplo
 - 6.7.4 Operador EXISTS
 - 6.7.4.1 Exemplo com o EXISTS
 - 6.7.4.2 Exemplo com o NOT EXISTS
 - 6.7.5 Usar uma Subquery na clausula FROM
- 7 Manipulação de Dados
 - 7.1 INSERT
 - 7.1.1 Inserir uma nova linha
 - 7.1.2 Inserir valores nulos
 - 7.1.3 Inserir valores específicos
 - 7.1.4 Copiar Linhas de Outra Tabela
 - 7.2 UPDATE
 - 7.2.1 Actualizar Linhas numa Tabela
 - 7.2.2 Update com Subquery de Várias Colunas
 - 7.2.3 Actualizar Linhas Baseadas Noutra Tabela
 - 7.3 DELETE
 - 7.3.1 Apagar linhas de uma tabela
 - 7.3.2 Apagar Linhas Baseadas noutra Tabela
- 8 Criar e Gerir Tabelas
 - 8.1 O que são Dicionário de Dados ?
 - 8.2 CREATE
 - 8.2.1 Criar uma Tabela Usando uma Subquery
 - 8.3 DROP
 - 8.3.1 Apagar uma Tabela
 - 8.4 ALTER TABLE
 - 8.4.1 Adicionar uma Coluna
 - 8.4.2 Modificar uma Coluna
 - 8.4.3 Apagar uma coluna
- 9 Incluir Restrições
 - 9.1 O Que São Restrições?
 - 9.2 Definição de Restrições
 - 9.3 Restrição NOT NULL
 - 9.4 Restrição UNIQUE
 - 9.5 Restrição PRIMARY KEY
 - 9.6 Restrição FOREIGN KEY
 - 9.7 Restrição CHECK
 - 9.8 Adicionar uma Restrição
 - 9.9 Apagar uma Restrição
 - 9.10 Desligar Restrições
 - 9.11 Ligar Restrições
 - 9.12 Restrições em Cascata
 - 9.13 Ver as Restrições
 - 9.13.1 Ver as Colunas Associadas às Restrições
- 10 Fontes

Alguns comandos SQL

Chaves

PK (Primary Key)

- Os atributos únicos não nulos
- Só ha 1 PK em cada tabela

PK refere-se aos conjuntos de um ou mais campos cujos valores nunca se repetem na mesma tabela e podem ser usadas como indice de referência para criar relacionamentos entre tabelas.

FK (Foreign Key)

FK refere-se ao tipo de relacionamento entre tabelas distintas de dados. Uma FK é um campo que aponta pra uma PK de outra tabela ou até da mesma.

Instrução SELECT Básica

```
SELECT <lista de campos> FROM <nome da tabela> ;
```

NOTA SOBRE O NULL

- Um NULL é um valor que está indisponível, não atribuído, desconhecido ou não aplicável.
- O NULL não é o mesmo que 0
- Quando se faz operações com NULL o resultado vêm NULL

Alias para coluna

- Renomeia o cabeçalho de uma coluna
- Segue imediatamente o nome da coluna
- A palavra chave **AS** entre o nome da coluna e o alias é opcional
- Requer aspas se contém espaços ou caracteres especiais ou é case sensitive

```
SELECT nome AS ident, sal salario FROM emp;
```

IDENT	SALARIO
-----	-----

```
SELECT nome "Nome", sal "salario" FROM emp;
```

Nome	salario
-----	-----

Operador de concatenação

- Concatena colunas ou strings de caracteres a outras colunas
- É representado por duas barras verticais ||
- Cria uma coluna resultante

```
SELECT nome||funcao AS "Empregados" FROM emp;
```

Empregados

KINGPRESIDENTE
JOHNTROLHA
...
...

String de Caracteres Literais

- Temos de usar '' para concatenar usando o ||

```
SELECT nome || ' e um ' || funcao AS "Detalhes do Empregado" FROM emp;
```

Detalhes do Empregado

KING e um PRESIDENTE
JOHN e um TROLHA
...
...

```
SELECT '0 livro ' || '' ||TITULO|| '' || ' custa ' || PRECO_TABELA AS "Titulo"
FROM LIVROS;
```

Titulo

0 livro "Antenas da Maya" custa 28
0 livro "Alice nas coisas" custa 21

DISTINCT

- Usado para eliminar linhas duplicadas

```
SELECT DISTINCT ndep FROM emp;
```

ndep

10
20
30
...
...

Restringir e ordenar dados

Usar o WHERE

```
SELECT nome, funcao, ndep FROM emp WHERE funcao='CONTABILISTA';
```

NOME	FUNCAO	NDEP
-----	-----	-----
JAMES	CONTABILISTA	30
SMITH	CONTABILISTA	10

Strings de caracteres e datas

- Valores de caracteres e datas vão para dentro de plicas
- Valores de caracteres são case sensitive
- Valores de data são format sensitive
- O formato default da data é DD-MON-YY

Outros operadores de comparação

BETWEEN AND

- Seleciona num intervalo

IN(list)

- Deixa especificar vários valores na cláusula **WHERE**

LIKE

- Permite pesquisar partes de strings

IS NULL

- Usado para comparar valores que sejam NULL

Operadores Lógicos

AND

- TRUE se ambas as condições forem TRUE

OR

- TRUE se pelo uma condição for TRUE

NOT

- TRUE se a condição for FALSE

Regras de precedência

1

- Todos os operadores de comparação

2

- NOT

3

- AND

4

- OR

Para alterar as regras de precedência podemos simplesmente usar parenteses

```
SELECT nome, funcao, sal FROM emp WHERE funcao="VENDEDOR"= OR funcao="PRESIDENTE" AND sal >1500;
```

O vendedor não tem de ganhar mais de 1500 de salário mas o presidente tem

```
SELECT nome, funcao, sal FROM emp WHERE (funcao="VENDEDOR" OR funcao="PRESIDENTE" AND sal >1500);
```

Agora têm ambos de ganhar mais de 1500 de salário, tanto o vendedor como o presidente

ORDER BY

- Ordena as linhas com a cláusula **ORDER BY**
- ASC -> ascendente e é o default
- DESC -> descendente
- O ORDER BY vem no fim do SELECT

```
SELECT nome, data FROM emp ORDER BY data;
```

nome	data
SMITH	17-DEC-80
JOHN	20-FEB-81

Funções de linha

- Aplicam-se a cada linha da tabela
- Manipulam os itens de dados
- Aceitam argumentos e devolvem um valor

Funções de strings

- Funções de conversão -> LOWER; UPPER; INITCAP
- Funções de manipulação de strings -> CONTACT; SUBSTR;LENGTH;INSTR;TRIM...

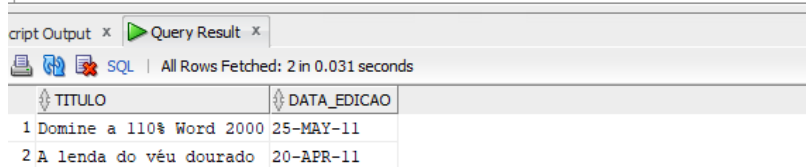
SUBSTR

Para dividir strings, podemos usar o SUBSTR.

SUBSTRING(string, start, length)

Exemplo que mostra os livros editados em 2011

```
SELECT TITULO ,DATA_EDICAO FROM LIVROS WHERE SUBSTR(DATA_EDICAO,8,9) = '11';
```



TITULO	DATA_EDICAO
1 Domine a 110% Word 2000	25-MAY-11
2 A lenda do véu dourado	20-APR-11

Separar nomes contendo mais do que 1 espaço

SELECT SUBSTR(NOME,1,INSTR(NOME,' ')-1) -> Pega no inicio da string e vai até ao primeiro ESPAÇO e retira o ESPAÇO com -1

SELECT SUBSTR(NOME,INSTR(NOME,' ',-1)+1) -> Pega no final da string e vai até ao ultimo ESPAÇO e retira o ESPAÇO com o +1

Diferenças entre TO_CHAR e TO_DATE

<p><code>to_char</code> function is used to convert the given data into character....</p> <pre>SQL> SELECT TO_CHAR(SYSDATE, 'dd/mm/yyyy') FROM dual;</pre> <p>TO_CHAR(SY ----- 04/04/2012</p>	<p><code>to_date</code> is used to convert the given data into date data format data type....</p> <p>eg: <code>to_date('070903', 'MMDDYY')</code> would return a date value of July 9, 2003.</p>
--	--

Subtrair Datas

<p>In Oracle, you can simply subtract two dates and get the difference in days. Also note that unlike SQL Server or MySQL, in Oracle you cannot perform a <code>select</code> statement without a <code>from</code> clause. One way around this is to use the builtin dummy table, <code>dual</code> :</p> <pre>SELECT TO_DATE('2000-01-02', 'YYYY-MM-DD') - TO_DATE('2000-01-01', 'YYYY-MM-DD') AS DateDiff FROM dual</pre>
--

Função TO_CHAR com Data

TO_CHAR(date, 'fmt')

fmt -> Remove os zeros e espaços atrás da data

- O modelo do formato:
 - Deve estar dentro de plicas e é case sensitive
 - Pode incluir qualquer elemento de formato de data válido
 - Está separado do valor data por vírgula

YYYY	Ano completo em números
YEAR	Ano soletrado
MM	Dois dígitos para o mês
MONTH	Nome completo do mês
DY	Abreviação com três letras do dia da semana
DAY	Nome completo do dia

Usar a Função TO_CHAR com datas

```
SQL> SELECT nome,
2  TO_CHAR(data_entrada, 'fmDD Month YYYY') DATA_ENTRADA
3  FROM emp;
```

```
NOME          DATA_ENTRADA
-----
KING          17 November 1981
BLAKE         1 May 1981
CLARK         9 June 1981
JONES         2 April 1981
MARTIN        28 September 1981
ALLEN         20 February 1981
...
14 rows selected.
```

Exemplo que mostra as datas no formato desejado

NOTA : DATA_EDICAO é uma string

```
SELECT TO_CHAR(DATA_EDICAO, 'DD-MM-YYYY') "Data" FROM LIVROS;
```

Data
1 02-08-2013
2 19-01-2014
3 14-01-2013
4 25-05-2011
5 17-02-2014
6 13-04-2005
7 26-01-2014
8 18-02-2014
9 03-01-2012
10 09-04-2002
11 09-04-2002
12 22-03-2013
13 01-04-2013
14 02-10-2013

Função TO_CHAR com Data para mostrar os dias e semana

```
SELECT DATA_EDICAO, TO_CHAR(DATA_EDICAO, 'D') "Dias da Semana" , TO_CHAR(DATA_EDICAO, 'W') AS "Nr da Semana" FROM LIVROS;
```

	DATA_EDICAO	Dias da Semana	Nr da Semana
1	02-AUG-13	6	1
2	19-JAN-14	1	3
3	14-JAN-13	2	2
4	25-MAY-11	4	4
5	17-FEB-14	2	3
6	13-APR-05	4	2
7	26-JAN-14	1	4
8	18-FEB-14	3	3
9	03-JAN-12	3	1
10	09-APR-02	3	2
11	09-APR-02	3	2
12	22-MAR-13	6	4
13	01-APR-13	2	1
14	23-DEC-12	1	4
15	05-FEB-14	4	1

FONTE do TO_CHAR

https://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements004.htm

Função TO_CHAR com números

TO_CHAR(number,'fmt')

- Usar estes formatos com a função TO_CHAR para mostrar o valor numérico como um carater

Função TO_CHAR com Números

TO_CHAR(number, 'fmt')

- Usar estes formatos com a função TO_CHAR para mostrar o valor numérico como um caracter:

9	Representa um número
0	Força a mostrar um zero
\$	Coloca um sinal de dólar flutuante
L	Usa o símbolo de moeda local flutuante
.	Imprime um ponto decimal
,	Imprime um indicador de milhar

Exemplo

```
SQL> SELECT TO_CHAR(sal, '$99,999') SALARIO
2 FROM emp
3 WHERE nome = 'SCOTT';
```

```
SALARIO
-----
$3,000
```

Função NVL

Converte NULL para um valor real

- Tipos de dados que não podem ser usados são date, charater e number
- Os tipos de dados devem concordar:
 - NVL(premios,0)
 - NVL(data_entrada,'01-JAN-97')

Exemplo

```
SQL> SELECT nome, sal, premios, (sal*12)+NVL(premios,0)
2 FROM emp;
```

NOME	SAL	PREMIOS	(SAL*12)+NVL(PREMIOS,0)
KING	5000		60000
BLAKE	2850		34200
CLARK	2450		29400
JONES	2975		35700
MARTIN	1250	1400	16400
ALLEN	1600	300	19500
...			

14 rows selected.

Exemplo

Mostre a lista de vendedores cujos prêmios foram menores do que o 10% do salário anual. O resultado deve incluir o nome do vendedor, 10% do seu salário anual e os prêmios, e deve ser ordenado pelos 10% de salário anual. No caso de haver vários vendedores com o mesmo salário anual, estes devem surgir ordenados pelo nome do vendedor.

```
SELECT NOME, (SAL * 12 * 0.1) "10% do salário", NVL(TO_CHAR(PREMIOS), 'Sem Premios')
FROM EMP
WHERE UPPER(FUNCAO) LIKE 'VENDEDOR%'
AND NVL(PREMIOS,0) < (SAL*12*0.1)
ORDER BY NOME;
```

Função DECODE

Facilita pesquisas condicionais fazendo o trabalho de uma instrução CASE ou IF-THEN-ELSE

Exemplo


```
SQL> SELECT funcao, sal,
2          DECODE(funcao, 'ANALISTA', SAL*1.1,
3                    'CONTADOR', SAL*1.15,
4                    'GESTOR', SAL*1.20,
5                    SAL)
6          SALARIO_REVISTO
7 FROM emp;
```

FUNCAO	SAL	SALARIO_REVISTO
PRESIDENTE	5000	5000
GESTOR	2850	3420
GESTOR	2450	2940
...		

14 rows selected.

Se for analista recebe sal x 1.1,
se for contador recebe sal x 1.15,
se for gestor recebe sal x 1.20,
se não o salário não altera

36

Funções aninhadas

São funções de funções.

As funções aninhadas são calculadas do nível mais interior para o mais exterior.

Exemplo

```
SQL> SELECT nome,
2          NVL(TO_CHAR(encar), 'Sem Encarregado')
3 FROM emp
4 WHERE encar IS NULL;
```

Como é NULL temos de fazer
WHERE encar IS NULL

NOME	NVL(TO_CHAR(ENCAR), 'Sem encarregado')
KING	Sem Encarregado

Pega nos empregados que não tem encarregado e como o encar estava a NULL altera de NULL para Sem Encarregado

39

Joins

O que é um join ?

Usar um join para pesquisar dados de mais do que uma tabela.

```
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column = table2.column;
```

- Escrever a condição de join na cláusula WHERE.
- Colocar o nome da tabela antes do nome da coluna quando o mesmo nome de coluna aparece em mais do que uma tabela.

Produto cartesiano

- ▶ Um produto cartesiano é formado quando:
 - ▶ É omitida uma condição join.
 - ▶ Uma condição join é inválida.
 - ▶ Todas as linhas da primeira tabela são joined com todas as linhas da segunda.
- ▶ Para evitar um produto cartesiano, incluir sempre uma condição join válida na cláusula WHERE.

Equijoin

O equijoin é quando usamos o "="

EMP			DEPT		
NEMP	NOME	NDEP	NDEP	NOME	LOC
7839	KING	10	10	CONTABILIDADE	NEW YORK
7698	BLAKE	30	30	VENDAS	CHICAGO
7782	CLARK	10	10	CONTABILIDADE	NEW YORK
7566	JONES	20	20	INVESTIGACAO	DALLAS
7654	MARTIN	30	30	VENDAS	CHICAGO
7499	ALLEN	30	30	VENDAS	CHICAGO
7844	TURNER	30	30	VENDAS	CHICAGO
7900	JAMES	30	30	VENDAS	CHICAGO
7521	WARD	30	30	VENDAS	CHICAGO
7902	FORD	20	20	INVESTIGACAO	DALLAS
7369	SMITH	20	20	INVESTIGACAO	DALLAS
...			...		
14 rows selected.			14 rows selected.		

Foreign key Primary key

```
SQL> SELECT emp.nemp, emp.nome, emp.ndep,
2      dep.ndep, dep.loc
3 FROM emp, dep
4 WHERE emp.ndep=dep.ndep;
```

NEMP	NOME	NDEP	NDEP	LOC
7839	KING	10	10	NEW YORK
7698	BLAKE	30	30	CHICAGO
7782	CLARK	10	10	NEW YORK
7566	JONES	20	20	DALLAS
...				
14 rows selected.				

Usando o operador AND

EMP			DEP		
NEMP	NOME	NDEP	NDEP	NOMEDEP	LOC
7839	KING	10	10	CONTABILIDADE	NEW YORK
7698	BLAKE	30	30	VENDAS	CHICAGO
7782	CLARK	10	10	CONTABILIDADE	NEW YORK
7566	JONES	20	20	INVESTIGAÇÃO	DALLAS
7654	MARTIN	30	30	VENDAS	CHICAGO
7499	ALLEN	30	30	VENDAS	CHICAGO
7844	TURNER	30	30	VENDAS	CHICAGO
7900	JAMES	30	30	VENDAS	CHICAGO
7521	WARD	30	30	VENDAS	CHICAGO
7902	FORD	20	20	INVESTIGAÇÃO	DALLAS
7369	SMITH	20	20	INVESTIGAÇÃO	DALLAS
...			...		
14 rows selected.			14 rows selected.		

```
SELECT emp.nome, dep.loc
FROM emp, dep
WHERE emp.ndep = dep.ndep
AND emp.nome = 'King'
```

Simplificar pesquisas com alias de tabela

```
SQL> SELECT emp.nemp, emp.nome, emp.ndep,
2      dep.ndep, dep.loc
3 FROM emp, dep
4 WHERE emp.ndep=dep.ndep;
```

```
SQL> SELECT e.nemp, e.nome, e.ndep,
2      d.ndep, d.loc
3 FROM emp e, dep d
4 WHERE e.ndep= d.ndep;
```

Joining de mais de duas tabelas

CLIENTE

NOME	CLIID
-----	-----
JOCKSPORTS	100
TKB SPORT SHOP	101
VOLLYRITE	102
JUST TENNIS	103
K+T SPORTS	105
SHAPE UP	106
WOMENS SPORTS	107
...	...
9 rows selected.	

PEDIDO

CLIID	PEDID
-----	-----
101	610
102	611
104	612
106	601
102	602
106	
106	
...	...
21 rows selected.	

ITEM

PEDID	ITEMID
-----	-----
610	3
611	1
612	1
601	1
602	1
...	...
64 rows selected.	

Non-Equijoin

O equijoin é quando não usamos o "="

EMP

NEMP	NOME	SAL
-----	-----	-----
7839	KING	5000
7698	BLAKE	2850
7782	CLARK	2450
7566	JONES	2975
7654	MARTIN	1250
7499	ALLEN	1600
7844	TURNER	1500
7900	JAMES	950
...		
14 rows selected.		

DESCONTOS

ESCALAO	SALINF	SALSUP
-----	-----	-----
1	700	1200
2	1201	1400
3	140	2000
4	2001	3000
5	3001	9999

“salario na tabela EMP
está entre o salário inferior e o
salário superior da tabela
DESCONTOS”

```
SQL> SELECT e.nome, e.sal, s.escalao
2 FROM emp e, descontos s
3 WHERE e.sal
4 BETWEEN s.salinf AND s.salsup;
```

NOME	SAL	ESCALAO
-----	-----	-----
JAMES	950	1
SMITH	800	1
ADAMS	1100	1
...		
14 rows selected.		

Outer join

EMP		DEP	
NOME	NDEP	NDEP	NOME
-----			-----
KING	10	10	CONTABILIDADE
BLAKE	30	30	VENDAS
CLARK	10	10	CONBILIDADE
JONES	20	20	INVESTIGAÇÃO
...		...	
		40	OPERACOES

Não há empregados no departamento de OPERAÇÕES

- Usa-se um outer join para ver também linhas que não verificam realmente a condição join.
- O operador outer join é o sinal (+).
- Coloca-se o sinal (+) no sitio onde não tem correspondência

Syntax

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column (+) = table2.column;
```

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column = table2.column (+);
```

Eu quero ver o nome do empregado com o nome do departamento mas tambem gostava de ver qual é o departamento que nao tem empregados.

```
SQL> SELECT  e.nome, d.ndep, d.nome
2 FROM      emp e,   dept d
3 WHERE     e.ndep (+) = d.ndep
4 ORDER BY e.ndep;
```

```
NOME          NDEP  NOME
-----
KING           10  CONTABILIDADE
CLARK          10  CONTABILIDADE
...
40 OPERAÇÕES
15 rows selected.
```

Self join

Fazer um join de uma tabela dela própria

EMP (TRABALHADOR)			EMP (ENCARREGADO)	
NEMP	NOME	ENCAR	NEMP	NOME
-----			-----	-----
7839	KING			
7698	BLAKE	7839	7839	KING
7782	CLARK	7839	7839	KING
7566	JONES	7839	7839	KING
7654	MARTIN	7698	7698	BLAKE
7499	ALLEN	7698	7698	BLAKE

“ENCAR na tabela TRABALHADOR é igual a NEMP na tabela ENCARREGADO”

Por exemplo se eu quiser saber, como se chama o chefe de um empregado?

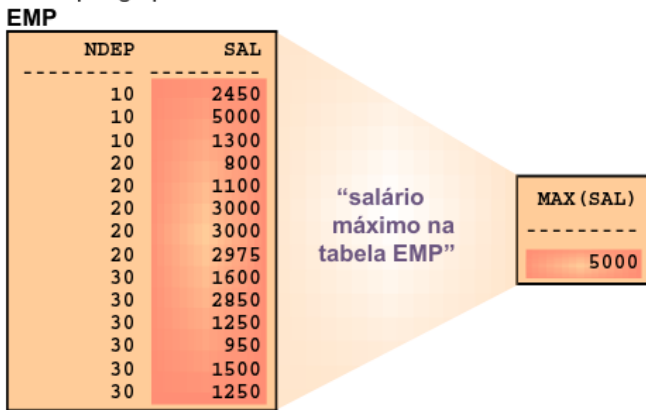
```
SQL> SELECT trab.nome || ' trab para ' || enc.nome
2 FROM emp trab, emp enc
3 WHERE trab.encar = enc.nemp;
```

```
TRAB.NOME || 'TRABPARA' || ENC.NOME
-----
BLAKE trab para KING
CLARK trab para KING
JONES trab para KING
MARTIN trab para BLAKE
...
13 rows selected.
```

Agregação de Dados Usando Funções de Grupo

As funções de grupo operam em conjuntos de linhas para dar um resultado por grupo.

Se quisermos fazer vários grupos temos de usar o **GROUP BY**



Tipos de funções de grupo

- AVG
- COUNT
 - Conta linhas
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

Syntax das funções de grupo

```
SELECT      [column,] group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

Funções AVG e SUM

```
SQL> SELECT AVG(sal), MAX(sal),
2 MIN(sal), SUM(sal)
3 FROM emp
4 WHERE funcao LIKE 'VEND%';
```

AVG (SAL)	MAX (SAL)	MIN (SAL)	SUM (SAL)
1400	1600	1250	5600

Funções MIN e MAX

```
SQL> SELECT MIN(data_entrada), MAX(data_entrada)
2 FROM emp;
```

MIN(DATA_ENTRADA)	MAX(DATA_ENTRADA)
17-DEC-80	12-JAN-83

Função COUNT

```
SQL> SELECT COUNT(*)
2 FROM emp
3 WHERE ndep = 30;
```

COUNT(*)
6

Aqui estamos a ver quantos empregados há no departamento 30

COUNT(expr) devolve o número de linhas não nulas.

```
SQL> SELECT COUNT(premios)
2 FROM emp
3 WHERE ndep = 30;
```

COUNT(PREMIOS)
4

Podemos transformar o nulo em 0 e assim já conseguimos contar todas as linhas mesmo usando o COUNT com valores nulos na tabela.

```
SELECT COUNT(NVL(premios)) FROM emp WHERE ndep = 30;
```

Funções de grupo e Nulos

- ▶ As funções de grupo ignoram os valores nulos na coluna.

```
SQL> SELECT AVG(premios)
2 FROM emp;
```

AVG(PREMIOS)
550

- ▶ A função NVL força as funções de grupo a incluir os valores nulos.

```
SQL> SELECT AVG(NVL(premios,0))
2 FROM emp;
```

AVG(NVL(PREMIOS,0))
157.14286

Criar Grupos de Dados

EMP

NDEP	SAL		NDEP	AVG(SAL)
10	2450		10	2916.6667
10	5000		20	2175
10	1300		30	1566.6667
20	800			
20	1100			
20	3000			
20	3000			
20	2975			
30	1600			
30	2850			
30	1250			
30	950			
30	1500			
30	1250			

“salário médio de cada departamento na tabela EMP”

Clausula GROUP BY

- Divide as linhas de uma tabela em pequenos grupos usando a clausula GROUP BY

Restrições de grupo é com o HAVING

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

- Todas as colunas na lista do SELECT que não estão nas funções de grupo têm de estar na clausula GROUP BY.

```
SQL> SELECT ndep, AVG(sal)
2 FROM emp
3 GROUP BY ndep;
```

NDEP	AVG(SAL)
10	2916.6667
20	2175
30	1566.6667

A coluna GROUP BY não tem de estar na lista do SELECT.

```
SQL> SELECT AVG(sal)
2 FROM emp
3 GROUP BY ndep;
```

AVG(SAL)
2916.6667
2175
1566.6667

Group By por Mais de uma Coluna**EMP**

NDEP	FUNCAO	SAL		NDEP	FUNCAO	SUM(SAL)
10	GESTOR	2450		10	CONTADOR	1300
10	PRESIDENTE	5000		10	GESTOR	2450
10	CONTADOR	1300		10	PRESIDENTE	5000
20	CONTADOR	800		20	ANALISTA	6000
20	CONTADOR	1100		20	CONTADOR	1900
20	ANALISTA	3000		20	GESTOR	2975
20	ANALISTA	3000		30	CONTADOR	950
20	GESTOR	2975		30	GESTOR	2850
30	VENDEDOR	1600		30	VENDEDOR	5600
30	GESTOR	2850				
30	VENDEDOR	1250				
30	CONTADOR	950				
30	VENDEDOR	1500				
30	VENDEDOR	1250				

“O somatório dos salários na tabela EMP para cada função, agrupado por departamento”

```
SQL> SELECT ndep, funcao, sum(sal)
2 FROM emp
3 GROUP BY ndep, funcao;
```

NDEP	FUNCAO	SUM(SAL)
10	CONTADOR	1300
10	GESTOR	2450
10	PRESIDENTE	5000
20	ANALISTA	6000
20	CONTADOR	1900
...		

9 rows selected.

Queries Ilegais usando Funções de Grupo

- Qualquer coluna ou expressão na lista do SELECT que não seja uma função de agregado tem de estar na cláusula GROUP BY.

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

Falta da coluna na cláusula GROUP BY

```
SELECT ndep, COUNT(nome)
*
ERROR at line 1:
ORA-00937: not a single-group group function
```

- Não pode usar-se a cláusula WHERE para restringir grupos
- Usa-se a cláusula HAVING para restringir grupos.

```
SQL> SELECT ndep, AVG(sal)
2 FROM emp
3 WHERE AVG(sal) > 2000
4 GROUP BY ndep;
```

Não pode usar-se a cláusula WHERE para restringir grupos

```
WHERE AVG(sal) > 2000
*
ERROR at line 3:
ORA-00934: group function is not allowed here
```

Excluir Resultados de Grupo

Having

- Usar a cláusula HAVING para restringir grupos
 - As linhas são agrupadas.
 - A função de grupo é aplicada.
 - Os grupos que satisfazem a cláusula HAVING são mostrados.

```
SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

Exemplo de exclusão

EMP

NDEP	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

5000

3000

2850

"salário máximo por departamento maior do que \$2900"

NDEP	MAX(SAL)
10	5000
20	3000

```
SQL> SELECT ndep, max(sal)
2 FROM emp
3 GROUP BY ndep
4 HAVING max(sal) > 2900;
```

NDEP	MAX(SAL)
10	5000
20	3000

2º Exemplo de exclusão

```
SQL> SELECT funcao, SUM(sal) VENC
2 FROM emp
3 WHERE funcao NOT LIKE 'VEND%'
4 GROUP BY funcao
5 HAVING SUM(sal) > 5000
6 ORDER BY SUM(sal);
```

Soma dos salários, que são superior a 5000, por função excluindo todas as funções que começam por VEND em ordem ascendente

FUNCAO	VENC
ANALISTA	6000
GESTOR	8275

Aninhar Funções de Grupo

Calcula a média dos salários por departamento e depois escolhe a maior média

```
SQL> SELECT max(avg(sal))
2 FROM emp
3 GROUP BY ndep;
```

MAX(AVG(SAL))
2916.6667

Ordem de cálculo de clausulas

```
SELECT column, group_function(column)
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

► Ordem de cálculo das cláusulas:

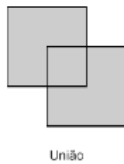
- Clausula WHERE
- Clausula GROUP BY
- Clausula HAVING

Operadores de Conjuntos

Os operadores de conjuntos UNION, UNION ALL, INTERSECT e MINUS permitem que se construam comandos com resultados de diferentes SELECTs combinados. Os comandos de SELECT a combinar podem mesmo referir-se a tabelas diferentes.

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.



Exemplos com o 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.

O UNION elimina as linhas repetidas. Para não eliminar as linhas repetidas deve usar-se o UNION ALL.

```
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:

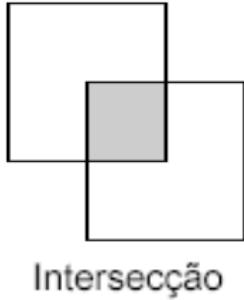
Misturada

 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.

Interseção

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



Exemplos com o INTERSECT

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

Devolve:

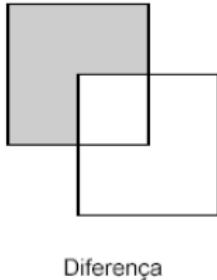
```
FUNCAO
-----
Continuo
Encarregado
```

2 rows selected.

O INTERSECT devolve o resultado comum (a intersecção) dos dois comandos.

Diferença

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



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.
```

O operador MINUS retira ao resultado do primeiro SELECT o obtido pelo segundo.

Mostra todas as funções do departamento 10 mas vai tirar todas as funções que aparecem no departamento 10 que são do departamento 30.

Subqueries

```
SELECT  select_list
FROM    table
WHERE   expr operator
        (SELECT  select_list
         FROM    table);
```

- ▶ A subquery (query interior) executa uma vez antes da query principal.
- ▶ O resultado da subquery é usado pela query principal (query exterior).

```
SQL> SELECT nome
2 FROM emp
3 WHERE sal >
4       (SELECT sal
5        FROM emp
6        WHERE nome='Jones');
```

2975 Como só retorna 1 valor, podemos usar esta subquery

```
NOME
-----
KING
FORD
SCOTT
```

Seleciona os empregados que recebem mais do que o salário do Jones

Guias para Usar Subqueries

- ▶ Incluir as subqueries em parenteses.
- ▶ Colocar as subqueries do lado direito do operador de comparação.
- ▶ Não acrescentar uma clausula ORDER BY a uma subquery. (antes do Oracle 8i)
- ▶ Usar operadores de linha única com subqueries de linha única.
- ▶ Usar operadores de várias linhas com subqueries de várias linhas.

Tipos de subqueries

Subquery de linha única



Subquery de várias linhas



Subquery de várias colunas



Subqueries de Linha Única

- ▶ Devolvem somente uma linha
- ▶ Usam operadores de comparação de linha única

Operador	Significado
=	Igual a
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
<>	Diferente de

Exemplo de Subquerie de Linha Única

```

SQL> SELECT  nome, funcao
2  FROM      emp
3  WHERE     funcao =
4              (SELECT  funcao
5                  FROM    emp
6                  WHERE   nemp = 7369)
7  AND       sal >
8              (SELECT  sal
9                  FROM    emp
10                 WHERE  nemp = 7876);

```

Diagrama de anotações: Uma seta vermelha aponta do texto "CONTADOR" para a subquery de função. Outra seta vermelha aponta do valor "1100" para a subquery de salário.

ENAME	FUNCAO
MILLER	CONTADOR

Quais são os empregados que têm a função do empregado 7369 e que têm o salário superior ao salário do emp 7876

Usar Funções de Grupo numa Subquery

```

SQL> SELECT  nome, funcao, sal
2  FROM      emp
3  WHERE     sal =
4              (SELECT  MIN(sal)
5                  FROM    emp);

```

Diagrama de anotações: Uma seta vermelha aponta do valor "800" para a subquery de função MIN(sal).

NOME	FUNCAO	SAL
SMITH	CONTADOR	800

Mostra o menor salário de todos os empregados

Clausula HAVING com Subqueries

- ▶ O Oracle Server executa primeiro as subqueries.
- ▶ O Oracle Server devolve os resultados para a clausula HAVING da query principal.

```
SQL> SELECT      ndep, MIN(sal)
2 FROM          emp
3 GROUP BY      ndep
4 HAVING        MIN(sal) >
5               (SELECT MIN(sal)
6 FROM          emp
7 WHERE         ndep = 20);
```

Diagrama: Uma seta vermelha aponta do valor 800 para a expressão MIN(sal) na cláusula HAVING.

Subqueries de Várias Linhas

- ▶ Devolve mais do que uma linha
- ▶ Usa operadores de comparação de várias linhas

Operador	Significado
IN	Igual a qualquer membro da lista
ANY	Compara o valor com cada valor devolvido pela subquery
ALL	Compara o valor com todos os valores devolvidos pela subquery

Usar o Operador ANY em Subqueries de Várias Linhas

```
SQL> SELECT nemp, nome, funcao
2 FROM emp
3 WHERE sal < ANY
4         (SELECT sal
5 FROM emp
6 WHERE funcao = 'CONTADOR')
7 AND funcao <> 'CONTADOR';
```

Diagrama: Setas vermelhas apontam dos valores 1300, 1100, 800 e 950 na coluna funcao para a expressão ANY na cláusula WHERE.

NEMP	NOME	FUNCAO
7654	MARTIN	VENDEDOR
7521	WARD	VENDEDOR

Quais são os empregados cujo o sal é menor do que qualquer empregado que tenha a funcao CONTADOR e que nao tenha a funcao CONTADOR

<ANY significa menor do que o máximo >ANY significa mais do que o mínimo
=ANY é equivalente a IN

Usar o Operador ALL em Subqueries de Várias Linhas

```
SQL> SELECT nemp, nome, funcao
2 FROM emp
3 WHERE sal > ALL
4 (SELECT avg(sal)
5 FROM emp
6 GROUP BY ndep);
```

NEMP	NOME	FUNCAO
7839	KING	PRESIDENTE
7566	JONES	GESTOR
7902	FORD	ANALISTA
7788	SCOTT	ANALISTA

>ALL significa maior do que o máximo

<ALL significa menor do que o mínimo

Subqueries de Várias Colunas

Subquery de Comparação Par

- Mostrar o id do pedido, o id do produto, e a quantidade de itens da tabela principal que satisfazem **ambos** o id do produto e a quantidade de um item com id do pedido 605.

```
SQL> SELECT pedid, prodid, qty
2 FROM item
3 WHERE (prodid, qty) IN
4 (SELECT prodid, qty
5 FROM item
6 WHERE pedid = 605)
7 AND pedid <> 605;
```

Dos pedidos diferentes de 605 quais são aqueles cujo prodid e qty é o mesmo do prodid e qty do pedido com pedid = 605

Subquery de Comparação Não Par

- Mostrar o id do pedido, o id do produto, e a quantidade de qualquer item para o qual o id do produto e quantidade satisfazem qualquer id de produto e qualquer quantidade de um item com id do pedido 605.

Vou ver os itens cujo o id do produto é o mesmo do 605 (mas podem ter quantidades diferentes) e a quantidade pode ser a mesma do 605 mas com o id do produto diferente e que não seja o produto 605

```
SQL> SELECT pedid, prodid, qty
2 FROM item
3 WHERE prodid IN (SELECT prodid
4 FROM item
5 WHERE pedid = 605)
6 AND qty IN (SELECT qty
7 FROM item
8 WHERE pedid = 605)
9 AND pedid <> 605;
```

Valores Nulos numa Subquery

- Mostrar os empregados que não têm subordinados

```
SQL> SELECT empregados.nome
2 FROM emp empregados
3 WHERE empregados.nemp NOT IN
4         (SELECT gestor.encar
5           FROM emp gestor);
no rows selected.
```

- Como um dos valores devolvidos pela query interior é NULL a query não devolve linhas. Porque todas as condições comparadas com um valor NULL dão NULL.
- Sempre que é possível aparecer um NULL não usar o NOT IN

Subconsultas Correlacionadas

- SELECT da subconsulta é executado uma vez por cada registo candidato (aqueles que não se sabe se vão ser seleccionados ou não) gerado pelo SELECT externo (ao contrário do que acontece em subconsultas não correlacionadas onde o SELECT interno é executado apenas uma vez).

Características

- 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 (segundo a ORACLE).

Ordem de execução de uma Subconsulta correlacionada

- Obtenção de registo candidato pelo SELECT externo.
- Execução do SELECT interno baseado em valores do registo candidato.
- Uso dos valores do SELECT interno para qualificar ou não o registo candidato.
- 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.

Todos os empregados com salário superior à média dos salários do seu departamento.

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 com o EXISTS

```

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

```

NDEP	NOME	LOCAL
10	Contabilidade	Condeixa
20	Investigação	Mealhada
30	Vendas	Coimbra

3 rows selected.

Os departamentos nos quais existem empregados.

Exemplo com o NOT EXISTS

```

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

```

NDEP	NOME	LOCAL
40	Planeamento	Montemor

1 row selected.

Os departamentos nos quais não existem quaisquer empregados

Usar uma Subquery na clausula FROM

- Mostrar os nomes dos empregados, salários, números de departamento e salários médios de todos os empregados que ganham mais que o salário médio do seu departamento.

```
SQL> SELECT  a.nome, a.sal, a.ndep, b.salavg
2 FROM      emp a, (SELECT  ndep, avg(sal) salavg
3                FROM      emp
4                GROUP BY ndep) b
5 WHERE     a.ndep = b.ndep
6 AND       a.sal > b.salavg;
```

Tabela Virtual

NOME	SAL	NDEP	SALAVG
KING	5000	10	2916.6667
JONES	2975	20	2175
SCOTT	3000	20	2175
...			

6 rows selected.

Manipulação de Dados

- Uma instrução **DML** é executada quando se:
 - Acrescentam novas colunas a uma tabela
 - Modificam linhas existentes numa tabela
 - Removem linhas existentes de uma tabela

INSERT

Inserir uma nova linha

```
SQL> INSERT INTO    dep (ndep, nome, loc)
2 VALUES           (50, 'DESENVOLV', 'DETROIT');
1 row created.
```

Inserir valores nulos

- Método implícito: Omitir a coluna da lista das colunas.

```
SQL> INSERT INTO    dep (ndep, nome )
2 VALUES           (60, 'MIS');
1 row created.
```

- Método explícito: Especificar a palavra chave NULL.

```
SQL> INSERT INTO    dep
2 VALUES           (70, 'FINANCA', NULL);
1 row created.
```

Inserir valores específicos

- Adicionar um novo empregado.

```
SQL> INSERT INTO emp
2 VALUES (2296, 'AROMANO', 'VENDEDOR', 7782,
3         TO_DATE('FEB 3, 1997', 'MON DD, YYYY'),
4         1300, NULL, 10);
1 row created.
```

- Verificar a adição.

NEMP	NOME	FUNCAO	ENCAR	DATA_ENTRADA	SAL	PREMIOS	NDEP
2296	AROMANO	SALESMAN	7782	03-FEB-97	1300		10

Copiar Linhas de Outra Tabela

- ▶ Escrever a instrução INSERT com uma subquery.

```
SQL> INSERT INTO gestores(id, nome, salario, data_entrada)
2      SELECT nemp, nome, sal, data_entrada
3      FROM emp
4      WHERE funcao = 'GESTOR';
3 rows created.
```

- ▶ Não usar a cláusula VALUES.
- ▶ Satisfazer o número de colunas da clausula INSERT com as da subquery.

 Isto é OBRIGATÓRIO

UPDATE

Actualizar Linhas numa Tabela

- ▶ Linhas ou linhas específicas são modificadas quando as especificamos na clausula WHERE.

```
SQL> UPDATE emp
2 SET ndep = 20
3 WHERE nemp = 7782;
1 row updated.
```

- ▶ Todas as linhas da tabela são modificadas se omitirmos a clausula WHERE.

```
SQL> UPDATE emp
2 SET ndep = 20;
14 rows updated.
```

Update com Subquery de Várias Colunas

- ▶ Actualizar a função e o departamento do empregado 7698 para os do empregado 7499.

```
SQL> UPDATE emp
2 SET (funcao, ndep) =
3      (SELECT funcao, ndep
4      FROM emp
5      WHERE nemp = 7499)
6 WHERE nemp = 7698;
1 row updated.
```

Actualizar Linhas Baseadas Noutra Tabela

- ▶ Usar subqueries nas instruções UPDATE para actualizar linhas da tabela, baseadas nos valores de outras tabelas.

```
SQL> UPDATE emp
2 SET ndep = (SELECT ndep
3      FROM emp
4      WHERE nemp = 7788)
5 WHERE funcao = (SELECT funcao
6      FROM emp
7      WHERE nemp = 7788);
2 rows updated.
```

DELETE

Apagar linhas de uma tabela

- ▶ Linhas específicas são apagadas quando se especifica a cláusula WHERE.

```
SQL> DELETE FROM    dep
      2 WHERE        nome = 'DESENV';
      1 row deleted.
```

- ▶ Todas as linhas na tabela são apagadas se for omitida a cláusula WHERE.

```
SQL> DELETE FROM    dep;
      4 rows deleted.
```

Apagar Linhas Baseadas noutra Tabela

- ▶ Usar subqueries nas instruções DELETE para remover linhas de uma tabela, baseadas em valores de outra tabela.

```
SQL> DELETE FROM    emp
      2 WHERE        ndep =
      3                (SELECT    ndep
      4                        FROM      dep d
      5                        WHERE     d.nome = 'VENDAS');
      6 rows deleted.
```

Criar e Gerir Tabelas

O que são Dicionário de Dados ?

- Colecção de tabelas criadas e mantidas pelo Oracle server
- Contém informação da base de dados

Descreve as tabelas pertencentes ao utilizador.

```
SQL> SELECT    *
      2 FROM      user_tables;
```

Mostra tipos de objectos distintos pertencentes ao utilizador.

```
SQL> SELECT    DISTINCT object_type
      2 FROM      user_objects;
```

Mostra tabelas, vistas e sequências pertencentes ao utilizador.

```
SQL> SELECT    *
      2 FROM      user_catalog;
```

CREATE

► Cria a tabela.

```
SQL> CREATE TABLE dep
2      (ndep    NUMBER(2),
3       nome    VARCHAR2(14),
4       loc     VARCHAR2(13));
Table created.
```

► Confirmar a criação da tabela.

```
SQL> DESCRIBE dep
```

Name	Null?	Type
-----	-----	-----
NDEP		NUMBER(2)
NOME		VARCHAR2(14)
LOC		VARCHAR2(13)

Criar uma Tabela Usando uma Subquery

```
SQL> CREATE TABLE   dep30
2  AS
3  SELECT   nemp, nome, sal*12 SAL_A, DATA_ENTRADA
4  FROM     emp
5  WHERE    ndep = 30;
Table created.
```

```
SQL> DESCRIBE dep30
```

Name	Null?	Type
-----	-----	-----
NEMP	NOT NULL	NUMBER(4)
NOME		VARCHAR2(10)
SAL_A		NUMBER
DATA_ENTRADA		DATE

DROP

Apagar uma Tabela

- Todos os dados e estrutura da tabela são apagados.
 - (Se for o dono da tabela ou um utilizador com o privilégio DROP ANYTABLE).
- Todas as transações pendentes são confirmadas (*commit*).
- Todos os indexes são apagados.
- Não pode ser feito o *rollback* desta instrução.

```
SQL> DROP TABLE dep30;
Table dropped.
```

ALTER TABLE

Adicionar uma Coluna

- ▶ Usa-se a cláusula **ADD** para adicionar colunas.

```
SQL> ALTER TABLE dep30
2 ADD (funcao VARCHAR2(9));
Table altered.
```

- ▶ A nova coluna passa a ser a última coluna.

NEMP	NOME	SAL_A	DATA_ENTRADA	FUNCAO
7698	BLAKE	34200	01-MAY-81	
7654	MARTIN	15000	28-SEP-81	
7499	ALLEN	19200	20-FEB-81	
7844	TURNER	18000	08-SEP-81	
...				

6 rows selected.

Modificar uma Coluna

- ▶ Pode-se mudar o tipo de dados, tamanho e valor por defeito de uma coluna.

```
SQL> ALTER TABLE dept30
2 MODIFY (nome VARCHAR2(15));
Table altered.
```

- ▶ Uma mudança no valor por defeito afecta somente as subseqüentes inserções à tabela.

Apagar uma coluna

- ▶ Usa-se a clausula **DROP COLUMN** para apagar colunas da tabela que já não sejam necessárias.

```
SQL> ALTER TABLE dep30
2 DROP COLUMN funcao ;
Table altered.
```

Incluir Restrições

O Que São Restrições?

- As restrições reforçam regras ao nível da tabela.
- As restrições previnem que uma tabela seja apagada se existirem dependências.
- Os seguintes tipos de restrições são válidos no Oracle:
 - **NOT NULL**
 - **UNIQUE**
 - **PRIMARY KEY**
 - **FOREIGN KEY**
 - **CHECK**

Definição de Restrições

```
CREATE TABLE emp(
  nemp NUMBER(4),
  nome VARCHAR2(10),
  ...
  ndep NUMBER(2) NOT NULL,
  CONSTRAINT emp_nemp_pk
  PRIMARY KEY (NEMP));
```

► Restrição ao nível da coluna

```
column [CONSTRAINT constraint_name] constraint_type,
```

► Restrição ao nível da tabela

```
column,...
[CONSTRAINT constraint_name] constraint_type
(column, ...),
```

Restrição NOT NULL

► Definida ao nível da coluna

```
SQL> CREATE TABLE emp(
2     nemp      NUMBER(4),
3     nome      VARCHAR2(10) NOT NULL,
4     função    VARCHAR2(9),
5     encar     NUMBER(4),
6     data_entrada DATE,
7     sal       NUMBER(7,2),
8     premios   NUMBER(7,2),
9     ndep      NUMBER(7,2) NOT NULL);
```

Restrição UNIQUE

► Pode ser definida ao nível da coluna ou ao nível da tabela

```
SQL> CREATE TABLE dep(
2     ndep      NUMBER(2),
3     nome      VARCHAR2(14),
4     loc       VARCHAR2(13),
5     CONSTRAINT dep_nome_uk UNIQUE(nome));
```

Restrição PRIMARY KEY

► Pode ser definida ao nível da tabela ou ao nível da coluna

```
SQL> CREATE TABLE dep(
2     ndep      NUMBER(2),
3     nome      VARCHAR2(14),
4     loc       VARCHAR2(13),
5     CONSTRAINT dep_nome_uk UNIQUE (nome),
6     CONSTRAINT dep_ndep_pk PRIMARY KEY(ndep));
```

Restrição FOREIGN KEY

► Pode ser definido ao nível da tabela ou ao nível da coluna

```
SQL> CREATE TABLE emp(
2     nemp      NUMBER(4),
3     nome      VARCHAR2(10) NOT NULL,
4     funcao     VARCHAR2(9),
5     encar      NUMBER(4),
6     data_entrada DATE,
7     sal       NUMBER(7,2),
8     premios   NUMBER(7,2),
9     ndep      NUMBER(7,2) NOT NULL,
10    CONSTRAINT emp_ndep_fk FOREIGN KEY (ndep)
11    REFERENCES dep (ndep));
```

- **FOREIGN KEY:**
 - Define a coluna na tabela filha ao nível da restrição da tabela
- **REFERENCES:**
 - Identifica a tabela e a coluna na tabela mãe
- **ON DELETE CASCADE:**
 - Permite apagar na tabela mãe e apagar as linhas dependentes na tabela filha

Restrição CHECK

Define uma condição que cada linha deve satisfazer

```
..., ndep    NUMBER(2),
CONSTRAINT emp_ndep_ck
CHECK (NDEP BETWEEN 10 AND 99),...
```

Adicionar uma Restrição

- Adicionar ou apagar, mas não modificar, uma restrição
- Ligar ou desligar restrições
- Adicionar uma restrição NOT NULL usando a cláusula MODIFY (se a tabela não contiver dados)

- ▶ Adicionar uma restrição FOREIGN KEY à tabela EMP indicando que um gestor já deve existir como um empregado válido na tabela EMP.

```
SQL> ALTER TABLE    emp
2  ADD CONSTRAINT    emp_encar_fk
3      FOREIGN KEY(encar) REFERENCES emp(nemp);
Table altered.
```

Apagar uma Restrição

- ▶ Remove a restrição de gestor na tabela EMP.

```
SQL> ALTER TABLE    emp
2  DROP CONSTRAINT    emp_encar_fk;
Table altered.
```

- ▶ Remove a restrição PRIMARY KEY na tabela DEP e apaga a restrição de FOREIGN KEY associada na coluna EMP.NDEP.

```
SQL> ALTER TABLE    dep
2  DROP PRIMARY KEY CASCADE;
Table altered.
```

Desligar Restrições

- Executa a cláusula DISABLE da instrução ALTER TABLE para desactivar uma restrição de integridade.
- Aplica a opção CASCADE para desligar as restrições de integridade dependentes.

```
SQL> ALTER TABLE    emp
2  DISABLE CONSTRAINT    emp_nemp_pk CASCADE;
Table altered.
```

Ligar Restrições

```
SQL> ALTER TABLE    emp
2  ENABLE CONSTRAINT    emp_nemp_pk;
Table altered.
```

Restrições em Cascata

- A cláusula CASCADE CONSTRAINTS é usada juntamente com a cláusula DROP COLUMN.
- A cláusula CASCADE CONSTRAINTS apaga todas as regras de integridade referencial que se referem às chaves primárias ou únicas definidas nas colunas apagadas.

- A cláusula CASCADE CONSTRAINTS também apaga todas as restrições de colunas compostas (várias colunas) definidas nas colunas apagadas.

Ver as Restrições

```
SQL> SELECT constraint_name, constraint_type,
2         search_condition
3 FROM   user_constraints
4 WHERE  table_name = 'EMP';
```

CONSTRAINT_NAME	C SEARCH_CONDITION
SYS_C00674	C NEMP IS NOT NULL
SYS_C00675	C NDEP IS NOT NULL
EMP_NEMP_PK	P
...	

Ver as Colunas Associadas às Restrições

```
SQL> SELECT constraint_name, column_name
2 FROM   user_cons_columns
3 WHERE  table_name = 'EMP';
```

CONSTRAINT_NAME	COLUMN_NAME
EMP_NDEP_FK	NDEP
EMP_NEMP_PK	NEMP
EMP_ENCAR_FK	ENCAR
SYS_C00674	NEMP
SYS_C00675	NDEP

Fontes

- <https://eufacoprogramas.com/sql-funcoes-com-numeros/>
- <https://aserlorenzo.com:8443/manSQL/Oracle/dml/subconsultas/SubConsultaMultiLine.htm> (SUBQUERIES)
- eden.dei.uc.pt/~bizarro/files/manual_praticas_bd1_v3.pdf IMPORTANTE
- <https://www.w3schools.in/mysql/ddl-dml-dcl/> (DDL,DML,DCL)

Retrieved from "[http://zebisnaga.pt/wiki/index.php?title=\(BD\)_SQL_Teórica&oldid=1155](http://zebisnaga.pt/wiki/index.php?title=(BD)_SQL_Teórica&oldid=1155)"