

Comandos de Consulta ao Esquema

Devemos ressaltar que a linguagem SQL é utilizada tanto pelos profissionais responsáveis pelos dados, onde é ressaltada a figura do Administrador do Banco de Dados e dos Analistas de Dados, como também pelos desenvolvedores de Aplicações. Enquanto àqueles estão preocupados com o desempenho, integridade do Banco de Dados e utilizam toda gama de recursos disponíveis no SQL, estes estão preocupados apenas em "transformar dados em informações", portanto para os desenvolvedores costuma-se dizer que conhecer o "select" já basta. Em nosso curso enfatizaremos a importância de TODOS os comandos do SQL, mas sabemos de antemão que os professores responsáveis pelas linguagens IDEO, VB e Delphi, ressaltarão a preponderância da instrução "select", que será apresentada a seguir e não no final do curso de SQL como geralmente acontece, pelo fato de que diversas disciplinas necessitam especificamente deste comando, que passaremos a apresentar:

1) Seleção de todas os campos (ou colunas) da tabela de Departamentos.

Resp:

```
SELECT * FROM DEPT;
```

O exemplo utiliza o coringa "*" para selecionar as colunas na ordem em que foram criadas. A instrução *Select*, como pudemos observar seleciona um grupo de registros de uma (ou mais) tabela(s). No caso a instrução *From* nos indica a necessidade de pesquisarmos tais dados apenas na tabela Dept.

Where como base das Restrição de tuplas.

A cláusula "where" corresponde ao operador restrição da álgebra relacional. Contém a condição que as tuplas devem obedecer a fim de serem listadas. Ela pode comparar valores em colunas, literais, expressões aritméticas ou funções.

A seguir apresentamos operadores lógicos e complementares a serem utilizados nas expressões apresentadas em where.

Operadores lógicos

| Operador | Significado |
|----------|----------------------|
| = | Igual a |
| > | Maior que |
| >= | Maior que ou igual a |
| < | Menor que |
| <= | Menor que ou igual a |

Exemplos:

```
SELECT EMPNOME, EMPSERV
```

```
FROM EMP
```

```
WHERE DEPNUME > 10;
```

```
SELECT EMPNOME, EMPSERV
```

```
FROM EMP
```

```
WHERE EMPSERV = 'GERENTE';
```

O conjunto de caracteres ou datas devem estar entre apóstrofes (') na cláusula "where".

2) Selecione todos os departamentos cujo orçamento mensal seja maior que 100000. Apresente o Nome de tal departamento e seu orçamento anual, que será obtido multiplicando-se o orçamento mensal por 12.

Resp: Neste problema precisamos de uma expressão que é a combinação de um ou mais valores, operadores ou funções que resultarão em um valor. Esta expressão poderá conter nomes de colunas, valores numéricos, constantes e operadores aritméticos.

```
SELECT DEPNUME, DEPORCA * 12
```

```
FROM DEPT
```

```
WHERE DEPORCA > 100000;
```

3) Apresente a instrução anterior porém ao invés dos "feios" DepNome e DepOrca, os Títulos Departamento e Orçamento.

Resp: Neste exemplo deveremos denominar colunas por apelidos. Os nomes das colunas mostradas por uma consulta, são geralmente os nomes existentes no Dicionário de Dado, porém geralmente estão armazenados na forma do mais puro "informatiquês", onde "todo mundo" sabe que CliCodi significa Código do Cliente. É possível (e provável) que o usuário desconheça estes símbolos, portanto devemos os apresentar dando apelidos às colunas "contaminadas" pelo informatiquês, que apesar de fundamental para os analistas, somente são vistos como enigmas para os usuários.

```
SELECT DEPNUME "DEPARTAMENTO", DEPORCA * 12 "ORCAMENTO ANUAL"
```

```
FROM DEPT
```

```
WHERE DEPORCA > 100000;
```

4) Apresente todos os salários existentes na empresa, porém omita eventuais duplicidades.

Resp: A cláusula Distinct elimina duplicidades, significando que somente relações distintas serão apresentadas como resultado de uma pesquisa.

```
SELECT DISTINCT EMPSERV
```

```
FROM EMP;
```

5) Apresente todos os dados dos empregados, considerando sua existência física diferente de sua existência lógica (ou seja devidamente inicializado).

Resp: Desejamos um tratamento diferenciado para valores nulos. Qualquer coluna de uma tupla que não contenha informações é denominada de nula, portanto informação não existente. Isto não é o mesmo que "zero", pois zero é um número como outro qualquer, enquanto que um valor nulo utiliza um "byte" de armazenagem interna e são tratados de forma diferenciada pelo SQL.

```
SELECT EMPNOME, EMPSALA + EMPCOMI
```

```
FROM EMP;
```

```
SELECT EMPNOME, NVL(EMPSALA,0) + NVL(EMPCOMI,0)
```

```
FROM EMP;
```

Obs: a função "NVL" é utilizada para converter valores nulos em zeros.

6) Apresente os nomes e funções da cada funcionário contidos na tabela empresa, porém classificados alfabeticamente (A..Z) e depois alfabeticamente invertido (Z..A).

Resp: A cláusula Order By modificará a ordem de apresentação do resultado da pesquisa (ascendente ou descendente).

```
SELECT EMPNOME, EMPSERV
```

```
FROM EMP
```

```
ORDER BY EMPNOME;
```

```
SELECT EMPNOME, EMPSERV
```

```
FROM EMP
```

```
ORDER BY EMPNOME DESC;
```

Nota: Também é possível fazer com que o resultado da pesquisa venha classificado por várias colunas. Sem a cláusula "order by" as linhas serão exibidas na sequência que o SGBD determinar.

7) Selecione os Nomes dos Departamentos que estejam na fábrica.

Resp:

```
SELECT DEPNOOME
```

```
FROM DEPT
```

```
WHERE DEPLOCA = "SAO PAULO";
```

O exemplo exigiu uma restrição (São Paulo) que nos obrigou a utilizar da instrução Where. Alguns analistas costumam afirmar em tom jocoso que SQL não passa de

"Selecione algo De algum lugar Onde se verificam tais relações"

Acreditamos que esta brincadeira pode ser útil ao estudante, na medida em que facilita sua compreensão dos objetivos elementares do SQL.

Demais Operadores

| Operador | Significado |
|-------------------------|--------------------------------|
| between...and... | entre dois valores (inclusive) |
| in (...) | lista de valores |
| like | com um padrão de caracteres |
| is null | é um valor nulo |

Exemplos:

```
SELECT EMPNOME, EMPSALA
```

```
FROM EMP
```

```
WHERE EMPSALA BETWEEN 500 AND 1000;
```

```
SELECT EMPNOME, DEPNUME
```

```
FROM EMP
```

```
WHERE DEPNUME IN (10,30);
```

```
SELECT EMPNOME, EMPSERV
```

```

FROM EMP

WHERE EMPNOME LIKE 'F%';

SELECT EMPNOME, EMPSERV

FROM EMP

WHERE EMPCOMI IS NULL;

```

O símbolo "%" pode ser usado para construir a pesquisa ("% " = qualquer sequência de nenhum até vários caracteres).

Operadores Negativos

| Operador | Significado |
|-------------------|---|
| <> | diferente |
| not nome_coluna = | diferente da coluna |
| not nome_coluna > | não maior que |
| not between | não entre dois valores informados |
| not in | não existente numa dada lista de valores |
| not like | diferente do padrão de caracteres informado |
| is not null | não é um valor nulo |

8) Selecione os Empregados cujos salários sejam menores que 1000 ou maiores que 3500.

Resp: Necessitaremos aqui a utilização de expressão negativas. A seguir apresentamos operadores negativos.

```

SELECT EMPNOME, EMPSALA

FROM EMP

WHERE EMPSALA NOT BETWEEN 1000 AND 3500;

```

9) Apresente todos os funcionários com salários entre 200 e 700 e que sejam Vendedores.

Resp: Necessitaremos de consultas com condições múltiplas.

Operadores "AND" (E) e "OR" (OU).

```

SELECT EMPNOME, EMPSALA, EMPSERV

FROM EMP

```

WHERE EMPSALA BETWEEN 200 AND 700

AND EMPSERV = 'VENDEDOR';

10) Apresente todos os funcionários com salários entre 200 e 700 ou que sejam Vendedores.

Resp:

```
SELECT EMPNOME, EMPSALA, EMPSERV
```

```
FROM EMP
```

```
WHERE EMPSALA BETWEEN 200 AND 700
```

```
OR EMPSERV = 'VENDEDOR';
```

11) Apresente todos os funcionários com salários entre 200 e 700 e que sejam Vendedores ou Balconistas.

Resp:

```
SELECT EMPNOME, EMPSALA, EMPSERV
```

```
FROM EMP
```

```
WHERE EMPSALA BETWEEN 200 AND 700
```

```
AND ( EMPSERV = 'BALCONISTA' OR EMPSERV = 'VENDEDOR' );
```

Funções de Caracteres

| Função | Significado |
|--------------------|--|
| Lower | força caracteres maiúsculos aparecerem em minúsculos.v |
| Upper | força caracteres minúsculos aparecerem em maiúsculos. |
| Concat(x,y) | concatena a string "x" com a string "y". |
| Substring(x,y,str) | extrai um substring da string "str", começando em "x", e termina em "y". |
| To_Char(num) | converte um valor numérico para uma string de caracteres. |
| To_Date(char,fmt) | converte uma string caracter em uma data. |
| ^Q | converte data para o formato apresentado. |

12) Apresente o nome de todos os empregados em letras minúsculas.

Resp:

```
SELECT LOWER( EMPNOME )
```

```
FROM EMP;
```

13) Apresente o nome de todos os empregados (somente as 10 primeiras letras).

Resp:

```
SELECT SUBSTRING (1,10,EMPNOME)
```

```
FROM EMP;
```

14) Apresente o nome de todos os empregados admitidos em 01/01/80.

Resp:

```
SELECT *
```

```
FROM EMP
```

```
WHERE EMPADMI = ^Q"DD-AAA-YYYY"("01-JAN-1980");
```

ou

```
SELECT *
```

```
FROM EMP
```

```
WHERE EMPADMI = ^Q("01-JAN-1980");
```

Funções Agregadas (ou de Agrupamento)

| Função | Retorno |
|--------------------|---|
| avg(n) | média do valor n, ignorando nulos |
| count(expr) | vezes que o número da expr avalia para algo não nulo |
| max(expr) | maior valor da expr |
| min(expr) | menor valor da expr |
| sum(n) | soma dos valores de n, ignorando nulos |

15) Apresente a Média, o Maior, o Menor e também a Somatória dos Salários pagos aos empregados.

Resp:

```
SELECT AVG(EMPSALA) FROM EMP;
```

```
SELECT MIN(EMPSALA) FROM EMP;
```

```
SELECT MAX(EMPSALA) FROM EMP;
```

```
SELECT SUM(EMPSALA) FROM EMP;
```

Agrupamentos

As funções de grupo operam sobre grupos de tuplas(linhas). Retornam resultados baseados em grupos de tuplas em vez de resultados de funções por tupla individual. A cláusula "group by" do comando "select" é utilizada para dividir tuplas em grupos menores.

A cláusula "GROUP BY" pode ser usada para dividir as tuplas de uma tabela em grupos menores. As funções de grupo devolvem uma informação sumarizada para cada grupo.

16) Apresente a média de salário pagos por departamento.

Resp:

```
SELECT DEPNUME, AVG(EMPSALA)
```

```
FROM EMP
```

```
GROUP BY DEPNUME;
```

Obs.: Qualquer coluna ou expressão na lista de seleção, que não for uma função agregada, deverá constar da cláusula "group by". Portanto é errado tentar impor uma "restrição" do tipo agregada na cláusula Where.

Having

A cláusula "HAVING" pode ser utilizada para especificar quais grupos deverão ser exibidos, portanto restringindo-os.

17) Retome o problema anterior, porém apresente resposta apenas para departamentos com mais de 10 empregados.

Resp:

```
SELECT DEPNUME, AVG(EMPSALA)
```

```
FROM EMP
```

```
GROUP BY DEPNUME
```


HAVING COUNT(*) > 3;

Obs.: A cláusula "group by" deve ser colocada antes da "having", pois os grupos são formados e as funções de grupos são calculadas antes de se resolver a cláusula "having".

A cláusula "where" não pode ser utilizada para restringir grupos que deverão ser exibidos.

Exemplificando ERRO típico - Restringindo Média Maior que 1000:

```
SELECT DEPNUME, AVG(EMPSALA)
```

```
FROM EMP
```

```
WHERE AVG(SALARIO) > 1000
```

```
GROUP BY DEPNUME;
```

(Esta seleção está ERRADA!)

```
SELECT DEPNUME, AVG(EMPSALA)
```

```
FROM EMP
```

```
GROUP BY DEPNUME
```

```
HAVING AVG(EMPSALA) > 1000;
```

(Seleção Adequada)

Seqüência no comando "Select":

```
SELECT ;coluna(s)
```

```
FROM ;tabela(s)
```

```
WHERE ;condição(ões) da(s) tupla(s)
```

```
GROUP BY ;condição(ões) do(s) grupo(s) de tupla(s)
```

```
HAVING ;condição(ões) do(s) grupo(s) de tupla(s)
```

```
ORDER BY ;coluna(s);
```

A "sql" fará a seguinte avaliação:

- a) WHERE, para estabelecer tuplas individuais candidatas (não pode conter funções de grupo)
- b) GROUP BY, para fixar grupos.
- c) HAVING, para selecionar grupos para exibição.

Equi-Junção (Junção por igualdade)

O relacionamento existente entre tabelas é chamado de equi-junção, pois os valores de colunas das duas tabelas são iguais. A Equi-junção é possível apenas quando tivermos definido de forma adequada a chave estrangeira de uma tabela e sua referência a chave primária da tabela precedente. Apesar de admitir-se em alguns casos, a equi-junção de tabelas, sem a correspondência Chave Primária-Chave Estrangeira, recomendamos fortemente ao estudante não utilizar este tipo de construção, pois certamente em nenhum momento nos exemplos propostos em nossa disciplina ou nas disciplinas de Análise e Projeto de Sistemas, serão necessárias tais junções.

18) Listar Nomes de Empregados, Cargos e Nome do Departamento onde o empregado trabalha.

Resp: Observemos que dois dos três dados solicitados estão na Tabela Emp, enquanto o outro dado está na Tabela Dept. Deveremos então acessar os dados restringindo convenientemente as relações existentes entre as tabelas. De fato sabemos que DEPNUME é chave primária da tabela de Departamentos e também é chave estrangeira da Tabela de Empregados. Portanto, este campo será o responsável pela equi-junção.

```
SELECT A.EMPNUME, A.EMPSEV, B.DEPNUME  
  
;FROM EMP A, DEPT B  
  
WHERE A.DEPNUME = B.DEPNUME;
```

Obs.: Note que as tabelas quando contém colunas com o mesmo nome, usa-se um apelido "alias" para substituir o nome da tabela associado a coluna. Imagine que alguém tivesse definido NOME para ser o Nome do Empregado na Tabela de Empregados e também NOME para ser o Nome do Departamento na Tabela de Departamentos. Tudo funcionaria de forma adequada, pois o aliás se encarregaria de evitar que uma ambigüidade fosse verificada. Embora SQL resolva de forma muito elegante o problema da nomenclatura idêntica para campos de tabelas, recomendamos que o estudante fortemente evite tal forma de nomear os campos. O SQL nunca confundirá um A.NOME com um B.NOME, porém podemos afirmar o mesmo de nós mesmos?

19) Liste os Códigos do Cada Funcionário, seus Nomes, seus Cargos e o nome do Gerente ao qual este se relaciona.

Resp: Precisamos criar um auto-relacionamento, ou seja, juntar uma tabela a ela própria. É possível juntarmos uma tabela a ela mesma com a utilização de apelidos, permitindo juntar tuplas da tabela a outra tuplas da mesma tabela.

```
SELECT A.EMPNUME, A.EMPNOME, A.EMPSERV, B.EMPNOME  
FROM EMP A, EMP B  
WHERE A.EMPGERE = B.EMPNUME;
```

As Sub-Consultas

Uma sub-consulta é um comando "select" que é aninhado dentro de outro "select" e que devolve resultados intermediários.

20) Relacione todos os nomes de funcionários e seus respectivos cargos, desde que o orçamento do departamento seja igual a 300000.

Resp:

```
SELECT EMPNOME, EMPSERV  
FROM EMP A  
WHERE 300000 IN ( SELECT DEPORCA  
FROM DEPT  
WHERE DEPT.DEPNUME = A.DEPNUME );
```

Nota: Observe que a cláusula IN torna-se verdadeira quando o atributo indicado está presente no conjunto obtido através da subconsulta.

21) Relacione todos os departamentos que possuem empregados com remuneração maior que 3500.

Resp:

```
SELECT DEPNOME  
FROM DEPT A  
WHERE EXISTS (SELECT *
```

FROM EMP

WHERE EMPSALA > 3500 AND EMP.DEPNUME = A.DEPNUME');

Nota: Observe que a cláusula EXISTS indica se o resultado de uma pesquisa contém ou não tuplas. Observe também que poderemos verificar a não existência (NOT EXISTS) caso esta alternativa seja mais conveniente.

Uniões

Podemos eventualmente unir duas linhas de consultas simplesmente utilizando a palavra reservada UNION.

22) Liste todos os empregados que tenham códigos > 10 ou Funcionários que trabalhem em departamentos com código maior que 10.

Resp: Poderíamos resolver esta pesquisa com um único Select, porém devido ao fato de estarmos trabalhando em nosso exemplo com apenas duas tabelas não conseguimos criar um exemplo muito adequado para utilização deste recurso.

(Select *

From Emp

Where EmpNume > 10)

Union

(Select *

From Emp

Where DepNume > 10);

Inserções, Alterações e Exclusões

Uma linguagem direcionada a extração de informações de um conjunto de dados, em tese não deveria incorporar comandos de manipulação dos dados. Devemos observar contudo que a mera existência de uma linguagem padronizada para acesso aos dados "convidava" os desenvolvedores a aderirem a uma linguagem "padrão" de manipulação de tabelas. Naturalmente cada desenvolvedor coloca "um algo mais" em seu SQL (SQL PLUS, SQL *, ISQL, e toda sorte de nomenclaturas), por um lado desvirtuando os objetivos da linguagem (padronização absoluta), mas em contrapartida otimiza os acessos ao seu banco de dados e por maior que sejam estas mudanças, jamais são tão importantes que impeçam que um programador versado em SQL tenha grandes dificuldades em se adaptar ao padrão de

determinada implementação. De fato as diferenças entre o SQL da Sybase, Oracle, Microsoft, são muito menores dos que as existentes entre o C, o BASIC e o Pascal, que são chamadas de linguagens "irmãs", pois todas originam-se conceitualmente no FORTRAN. Podemos observar que todas as três linguagens mencionadas possuem estruturas de controle tipo "para" (for), "enquanto" (while) e repita (do..while, repeat..until). Todas trabalham com blocos de instrução, todas tem regras semelhantes para declaração de variáveis e todas usam comandos de tomada decisão baseadas em instruções do tipo "se" ou "caso", porém apesar de tantas semelhanças (sic), é praticamente impossível que um programador excelente em uma linguagem consiga rapidamente ser excelente em outra linguagem do grupo. Poderíamos arriscar a dizer que um excelente programador C que utilize a implementação da Symantech terá que passar por um breve período de adaptação para adaptar-se ao C da Microsoft.

O que ocorreria então se este programador tiver que adaptar-se ao Delphi (Pascal) da Borland?

De forma alguma o mesmo ocorrerá com o especialista em SQL ao ter que migrar do Banco de Dados X para o Banco de Dados Y. Naturalmente existirá a necessidade de aprendizado, mas este programador poderá ir adaptando-se aos poucos sem precisar ser retreinado, o que é um aspecto extremamente vantajoso para as empresas.

Inserir (Insert)

```
INSERT INTO <tabela> [<campos>] [VALUES <valores>]
```

Ex:

```
INSERT INTO DEPT;
```

Possibilita a inserção de registros de forma interativa.

```
INSERT INTO DEPT (DEPNUME,DEPNOME,DEPLOCA) VALUES  
(70,"PRODUCAO","RIO DE JANEIRO");
```

Possibilita a inserção de registros em tabelas sem digitação dos dados.

Atualizar (Update)

```
UPDATE <tabela> SET <campo> = <expressão> [WHERE <condição>];
```

Ex:

```
UPDATE EMP SET EMPSALA = EMPSALA* 1.2 WHERE EMPSALA< 1000;
```

Excluir (Delete)

```
DELETE FROM <tabela> [WHERE <condição>];
```

Ex:

```
DELETE FROM emp WHERE EMPSALA > 5000;
```

Transações

Muitas vezes gostaríamos que determinado processo, caso fosse abortado por qualquer motivo, pudesse ser inteiramente cancelado. Imaginemos por exemplo um usuário digitando um pedido. Imaginemos ainda que o sistema possa reservar cada item solicitado de maneira "on line", ou seja ao mesmo tempo em que estou digitando a quantidade o sistema já "empenhe" uma quantidade equivalente no estoque. Imaginemos ainda que o sistema deve cancelar todas as operações se apenas um dos itens não puder ser atendido. Grande problema, caso não pudéssemos anular todos os processos a partir de determinada condição.

Vamos simular tal ocorrência com nosso banco de dados EMP. Imaginemos que ao invés de digitarmos DELETE FROM emp WHERE salario > 5000; tivéssemos digitado DELETE FROM emp WHERE salario > 500; Ao invés de eliminarmos 2 registros, praticamente teríamos eliminado o banco de dados todo. Para evitarmos que um erro de digitação, ou um processo iniciado porém sem condição de ser completado integralmente comprometa todos nossos dados podemos criar uma transação que nos assegurará que nossos testes sejam bem sucedidos ou cancelados sem comprometer nossos dados.

```
begin transaction;
```

```
delete from emp where salario > 500;
```

```
if SQL_RECORDCOUNT > 20 THEN;
```

```
ROLLBACK TRASACTION;
```

```
else
```

```
COMMIT;
```

```
endif;
```

```
end transaction;
```

Visões

Uma visão consiste basicamente de uma tabela derivada de outras tabelas. Considerando o exemplo TRABALHO, poderíamos criar uma visão baseada na Tabela de Empregados (EMP) e na Tabela de Departamentos (DEPT) onde tivéssemos somente os Nomes dos Funcionários e os Departamentos nos quais estes trabalhassem. Teríamos algo assemelhado ao abaixo representado

```
CREATE VIEW EMP_DEP  
  
AS SELECT E.EMPNOME, D.DEPNOME  
  
FROM EMP E, DEPT D  
  
WHERE E.DEPNUME = D.DEPNUME;
```

Devemos observar que:

- 1- Uma visão definida sobre uma única tabela somente será atualizável se os atributos da tal visão contiverem a chave primária de tal tabela.
- 2- Visões sobre várias tabelas não são passíveis de atualizações.
- 3- Visões que se utilizam de funções de agrupamentos, também não poderão ser atualizadas.

PARTE III - Relatórios

Comando:

```
REPORT DISTINCT / UNIQUE
```

```
[ atributo(s) ]
```

```
REPORTTOP
```

```
PAGETOP
```

```
TOP
```

```
DETAIL
```

```
NONE
```

```
BOTTOM
```

PAGEBOTTOM

REPORTBOTTOM

FROM [tabela(s)]

[WHERE clausula-where]

[GROUP BY clausula-grupo]

[ORDER BY clausula-order by];

Como exemplo converteremos um simples Select em um Report, temos:

SELECT EMPNOME

FROM EMP

WHERE DEPNUME = 1000;

REPORT

DETAIL EMPNOME

WHERE DEPNUME = 1000;

Podemos direcionar a saída de um relatório tanto para um arquivo como para uma impressora.

Para um arquivo:

```
;REPORT ON "RELAT.DAT" ...
```

Para uma impressora:

```
REPORT ON LP:" ...
```

Agora incrementando um report temos:

REPORT

```
REPORTTOP COL 10, "*** RELATORIO DE FUNCIONARIOS *** ",
```



```
TODAY %Q"DD/MM/YY", SKIP,  
  
COL 10, "=====", SKIP 2  
  
DETAIL COL 10, NOME %C22, SALARIO %FS, ADMISSAO  
%Q"DD/MM/YY"  
  
REPORTBOTTOM COL 10,  
  
"=====", SKIP,  
  
COL 20, "TOTAL:", TOTAL(SALARIO)  
  
FROM EMP  
  
ORDER BY NOME;
```

Onde:

REPORTTOP - O que sera impresso no topo do relatório.

PAGETOP - Impresso em cada topo de pagina.

TOP - Impresso em cada Topo do Sort-Grupo do relatório.

DETAIL - O que sera impresso em cada linha.

NONE - Se não tiver resultado o select, não sera impresso o relatório.

BOTTOM - Impresso em cada Bottom do Sort-Grupo do relatório

PAGEBOTTOM - O que sera impresso no rodapé de cada pagina.

REPORTBOTTOM - O que sera impresso no rodape do relatório.

Formatos:

%C - character

%D - data

y - ano,

n - mes numérico,

a - mes alfanumérico,

d - dia,

j - dia e ano juliano

Exemplo: %D"dd/mm/yy"

%I - inteiro

%F - ponto flutuante

%FSZ onde: S - separador de 3 dígitos e decimal point

Z - zeros serão suprimidos

%Q - data

%J - Hora

h - hora, m - minutos, s - segundos

%T - hora

E temos as funções: TOTAL, AVERAGE, MAXIMUM, MINIMUM.