



Introdução a SQL para o Domínio Gerador de Relatórios

dominio
sistemas

Saiba que este documento não poderá ser reproduzido, seja por meio eletrônico ou mecânico, sem a permissão expressa por escrito da Domínio Sistemas Ltda. Nesse caso, somente a Domínio Sistemas poderá ter patentes ou pedidos de patentes, marcas comerciais, direitos autorais ou outros de propriedade intelectual, relacionados aos assuntos tratados nesse documento. Além disso, o fornecimento desse documento não lhe concede licença sobre tais patentes, marcas comerciais, direitos autorais ou outros de propriedade intelectual; exceto nos termos expressamente estipulados em contrato de licença da Domínio Sistemas.

É importante lembrar que as empresas, os nomes de pessoas e os dados aqui mencionados são fictícios; salvo indicação contrária.

© 1998 - 2009 Domínio Sistemas Ltda. Todos os direitos reservados.

Índice

Convenção de cursores, ícones e tipografia.....	9
Lista de Abreviaturas e Siglas.....	10
Visão Geral do Curso.....	11
Descrição.....	11
Público Alvo.....	11
Pré-requisitos.....	11
Objetivos.....	11
1. Conceitos Básicos.....	12
1.1. Introdução.....	12
1.2. Objetivo.....	12
1.3. Introdução ao Banco de Dados.....	12
1.4. Compreendendo uma Tabela.....	13
1.5. Colunas de uma Tabela.....	13
1.6. Linhas de uma Tabela.....	14
2. Introdução a SQL	15
2.1. O Ambiente SQL.....	15
2.2. O Modelo Proposto.....	15
2.3. Modelo Tabelas.....	16
2.4. Alguns Tipos de Dados Aceitos.....	16
2.5. Definindo Chave Primária (Primary Key).....	17
2.6. Chave Estrangeira (Foreign Key).....	17
2.7. Operadores Aritméticos.....	17
2.8. Operadores Relacionais ou de Comparação.....	18
2.9. Operadores Lógicos.....	18
2.10. Extraindo Informações em um Banco de Dados.....	19
3. Exercícios de Revisão.....	21
3.1. Exercícios de Banco de Dados.....	21
3.2. Exercícios do Modelo Proposto.....	22
4. Consultas Simples.....	23
4.1. Introdução.....	23
4.2. Objetivo.....	23
4.3. Sintaxe do Comando SELECT.....	23

4.4. Extraindo Dados.....	24
4.5. Utilizando Condições para Extração de Dados.....	26
4.6. Aplicando Operadores Lógicos na Cláusula WHERE.....	28
4.7. Distinct.....	30
4.8. Agrupamento por Group By.....	30
4.9. Null.....	31
4.10. Negando uma Condição (NOT).....	32
4.11. Sub-Consultas (Subqueries).....	33
4.12. Sub-Consultas na Cláusula Where do Select Principal.....	34
4.13. A Condição EXISTS.....	35
4.14. Uniões (Union).....	36
5. Funções.....	38
5.1. Funções de Agregação.....	39
5.2. Funções de Data e Hora.....	40
5.3. Funções String.....	44
6. Joins / Outer Joins.....	47
6.1. Produto Cartesiano (Relação entre Tabelas).....	47
6.2. Relacionamento (Outer Joins).....	49
7. Exercícios de Revisão 2.....	52
8. Caderno de Respostas.....	54
8.1. Exercícios de Revisão – Banco de Dados.....	54
8.2. Exercícios de Revisão – Modelo Proposto.....	54
8.3. Exercícios de Revisão 2.....	54
9. Tabelas Modelo.....	57

Convenção de cursores, ícones e tipografia


A fonte `Courier New` é utilizada para se referir a comandos (instruções), a serem utilizados na SQL, por exemplo:

```
SELECT      *
FROM funcionarios
```

A fonte *Arial em Itálico* é utilizada para definir os nomes dos sistemas da Domínio, bem como marcas registradas citadas nesse material, por exemplo:

Este treinamento é direcionado aos usuários dos sistemas da *Domínio Sistemas* que não têm conhecimento em SQL, sendo que as informações de SQL obtidas durante o curso são voltadas às necessidades do *Gerador de Relatório*.

O ícone abaixo, você encontrará nos resultados de comandos aplicados no ISQL, para a conferência dos mesmos, por exemplo:

	cod_func	nome	dt_admissao	substr(dt_admissao,6,2)
	1	João Anastácio	1995-01-01	01
	2	Pedro Brasona	2001-02-05	02
	3	Manoel da R. Pizzolo	1990-05-02	05
	4	João Malha	1999-04-10	04
	5	Antônio Nascimento	1985-01-10	01
	6	Maria de Jesus	1985-02-11	02
	7	Raimundo Pizzolo	1985-02-11	02

Lista de Abreviaturas e Siglas

- ANSI – American National Standard Institute;
- DBA – Database Administrator;
- DBP – Database Projector;
- DDL – Data Definition Language;
- DML – Data Manipulation Language;
- ISO – International Organization for Standardization;
- ISQL – Interactive Structured Query Language;
- SGBDR – Sistema Gerenciador de Banco de Dados Relacionais;
- SQL – Structured Query Language.

Visão Geral do Curso

Descrição

A finalidade do treinamento Introdução a SQL para o *Gerador de Relatórios* é apresentar aos nossos usuários e técnicos os seguintes tópicos:

- Noção de Banco de Dados Relacionais (SGBDR);
- Tipos de caracteres encontrados em um banco de dados;
- Sintaxe do comando select;
- Como buscar informações em uma banco de dados (select);
- Como aplicar funções que auxiliam a manipulação dos dados;
- Como fazer o agrupamento de dados (Group by);
- Como eliminar duplicidade em um banco de dados (distinct);
- Comandos que designam a não existência de valor no banco de dados (Null);
- Negação de condições;
- Como relacionar (vincular) várias tabelas (Outer Joins);
- Como fazer sub-consultas;
- Como unir consultas distintas (união), etc.

Público Alvo

Este treinamento é direcionado aos usuários dos sistemas da *Domínio Sistemas*, que não têm conhecimento em SQL, sendo que as informações de SQL obtidas durante o curso são voltadas às necessidades do *Gerador de Relatórios*.

Pré-requisitos

Para que o estudante tenha um bom desempenho durante o treinamento é necessário que apresente os seguintes requisitos:

- Conhecimento básico de *Windows 95/98/2000*;
- Montar tabelas com mais de 2 campos no *Word* ou *Excel*.

Objetivos

Ao término desse treinamento, o usuário estará apto a realizar as seguintes funções:

- Identificar o que é um Banco de Dados, uma tabela e seus atributos;
- Extrair informações via SQL de variadas formas;
- Utilizar funções auxiliares em seus comandos SQL.

1. Conceitos Básicos

1.1. Introdução

Neste módulo daremos uma visão geral (teórica) de banco de dados, tabelas, chaves primária e estrangeira, tipos de dados e operadores aritméticos, lógicos e de comparação.

1.2. Objetivo

Ao final desse módulo, você estará apto a:

- Definir o que é um Banco de Dados;
- Identificar uma tabela;
- Identificar Linguagem de Definição e de Manipulação;
- Identificar tipos de dados em uma tabela;
- Identificar chaves primária e estrangeira;
- Trabalhar com operadores lógicos, aritméticos e relacionais.

1.3. Introdução ao Banco de Dados

Todos nós temos conhecimento de que existem gigantescas bases de dados gerenciando nossas vidas. De fato, sabemos que nossa conta bancária faz parte de uma coleção imensa de contas de nosso banco. Nosso Título Eleitoral ou Cadastro de Pessoa Física certamente estão armazenados em Bancos de Dados colossais. Sabemos também que quando sacamos dinheiro no caixa eletrônico, nosso saldo e as movimentações existentes já estão à disposição.

Nessas situações, compreendemos que existe uma necessidade em realizar o armazenamento de uma série de informações, que não se encontram efetivamente isoladas uma das outras, ou seja, existe uma gama de dados que se referem a relacionamentos existentes entre as informações a serem manipuladas.

Esses Bancos de Dados, além de manterem todo esse volume de dados organizado, também devem permitir atualizações, inclusões e exclusões do volume de dados sem nunca perder a consistência. E não podemos esquecer que, na maioria das vezes, estaremos lidando com acessos concorrentes a várias tabelas de nosso banco de dados. Em algumas delas, com mais de um acesso ao mesmo registro de uma mesma tabela.

Um Banco de Dados contém os dados dispostos numa ordem pré-determinada, para um propósito bem definido, devido a um projeto de sistema.

Um Banco de Dados representa aspectos do mundo real. Assim sendo, uma Base de Dados (ou Banco de Dados, ou ainda BD) é uma fonte de onde podemos extrair uma gama de informações derivadas, que possui um nível de interação com eventos, como o mundo real que representa. A forma mais comum de interação Usuário e Banco de Dados se dá através de

sistemas específicos, que, por sua vez, acessam geralmente através da linguagem SQL o volume de informações.

Os Administradores de Banco de Dados (DBA) são responsáveis pelo controle de acesso aos dados e pela coordenação da utilização do BD. Já os projetistas de Banco de Dados (DBP) são analistas que identificam os dados a serem armazenados em um Banco de Dados e a forma como esses serão representados.

Os Analistas e Programadores de Desenvolvimento criam sistemas que acessam os dados da forma necessária ao Usuário Final, sendo aquele que interage diretamente com o Banco de Dados.

1.4. Compreendendo uma Tabela

Conceitualmente, tabela é um quadro onde se indica alguma “coisa”. Dependendo do tipo de informação que será demonstrada, o layout de uma tabela pode variar consideravelmente. Quem nunca viu uma tabela em estatística, mostrando, por exemplo, ano a ano a taxa de natalidade ou mortalidade infantil, ou então nos jornais, demonstrando a variação do dólar, etc.? Até mesmo softwares foram construídos, de modo que sua aparência trabalhasse em forma de tabela, como por exemplo as planilhas eletrônicas.

Em computação, uma tabela nada mais é do que uma unidade básica de armazenamento de dados em um banco de dados, ou ainda uma estrutura formada por campos ou colunas, onde as informações contidas nesses são de uma mesma espécie. Também considerada como um conjunto de linhas ou de listas de valores.

1.5. Colunas de uma Tabela

Podemos então dizer que as colunas são um conjunto de células dentro de uma tabela que armazenam sempre um mesmo tipo de informação em particular. Por exemplo, supondo que em uma tabela exista uma coluna onde será mostrado o percentual de natalidade de um determinado ano. Nessa coluna serão mostrados somente percentuais, pois é o tipo de informação que deve ser exibida, tendo em vista que não haveria sentido mostrar em algumas células percentuais e em outras células um outro nome ou uma data por exemplo.

Quando utilizamos tabelas em um banco de dados, também trabalhamos dessa forma, contudo existe ainda uma vantagem de que podemos definir quais tipos de valores uma coluna pode aceitar. Por exemplo: se queremos uma tabela de funcionários, onde uma coluna seria a data de admissão, determinamos então que essa coluna só poderá aceitar datas válidas, o que evita possíveis erros de digitação.

Mais adiante veremos alguns tipos de dados que uma coluna pode receber dentro de um banco de dados.

1.6. Linhas de uma Tabela

Também chamada de registro, é um conjunto de colunas que ficam dispostas horizontalmente. Todas as linhas possuem o mesmo número de colunas.

2. Introdução a SQL

Structured Query Language (Linguagem de Consulta Estruturada)

Quando os Bancos de Dados estavam sendo desenvolvidos, foram criadas linguagens destinadas a definições e manipulações desses bancos, e devido a essa necessidade, foi desenvolvida pelo Departamento de Pesquisas da IBM a SQL como forma de interface para os sistemas de Banco de dados relacionais. Mais tarde a American National Standard Institute (ANSI) definiu um padrão SQL mundial, atualmente a ANSI/ISO definem esses padrões.

A SQL é uma linguagem usada pela maioria dos bancos de dados relacionais, baseada no inglês. Os comandos de SQL utilizados para a definição de dados, são conhecidos como DDL (Data Definition Language) e compostos, entre outros, pelos comandos **Create**, que é destinado à criação do Banco de Dados e Tabelas que o compõem. Além das relações existentes entre as tabelas, o comando **Alter**, permite inserir e eliminar atributos nas tabelas existentes, e o comando **Drop**, utilizado para eliminar a definição da tabela, seus dados e referências.

Os comandos da série DML (Data Manipulation Language) são destinados às consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo, os comandos: **Select**, **Insert**, **Update** e **Delete**.

A linguagem SQL tem a capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro. Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens, independente das tabelas e organização lógica dos dados. Outra característica interessante na linguagem SQL é a capacidade que dispomos de cancelar uma série de atualizações, ou de gravá-las depois de iniciar uma seqüência de atualizações. Os comandos **Commit** e **Rollback** são responsáveis por essas facilidades.

Devemos notar que a linguagem SQL consegue implementar essas soluções por estar baseada em Banco de Dados, que garante a integridade das relações existentes entre as tabelas e seus índices.

2.1. O Ambiente SQL

Dispomos na *Domínio Sistemas* para a ministração do curso de SQL, voltado ao *Gerador de Relatórios*, da ferramenta *ISQL*.

2.2. O Modelo Proposto

Para futuros testes com comandos SQL, montamos um modelo contendo três tabelas:

Funcionários: será a tabela onde serão armazenadas algumas informações referentes aos funcionários.

Deptos: armazenaremos nessa tabela os departamentos onde um determinado funcionário estará ligado.

Bancos: armazenaremos os bancos, em caso do funcionário trabalhar com bancos.

Quando vocês visualizarem as tabelas construídas abaixo, imaginem que dentro de um banco de dados estarão exatamente dessa forma.

Observem a formação dessas tabelas no quesito colunas e linhas, e procurem identificar que tipo de informação cada coluna aceita.

2.3. Modelo Tabelas

TABELA DE FUNCIONÁRIOS (funcionários)						
cod_func	nome	cod_depto	salário	dt_admissão	dt_nascto	cod_banco
1	João Anastácio	1	500,00	1995-01-01	1975-01-02 15:12:13	1
2	Pedro Brasona	2	800,00	2001-02-05	1980-12-01 17:12:14	2
3	Manoel da R. Pizzolo	3	1500,00	1990-05-02	1930-10-08 12:00:00	3
4	João Malha	4	189,00	1999-04-10	1988-04-04 13:00:00	4
5	Antônio Nascimento	3	200,00	1985-01-10	1967-01-10 13:00:00	(NULL)
6	Maria de Jesus	4	180,00	1985-02-11	1967-02-11 13:00:00	(NULL)
7	Raimundo Pizzolo	4	180,00	1985-02-11	1967-02-11 14:00:00	(NULL)

TABELA DE DEPARTAMENTOS (deptos)	
cod_depto	Nome
1	Recursos Humanos
2	Desenvolvimento
3	Financeiro
4	Editoração

TABELA BANCOS (bancos)			
cod_banco	Nome	agência	número_banco
1	Banco do Brasil	0407-3	6620-6
2	Bradesco	500-4	1246-88
3	Bradesco	345-1	1232-2
4	Itaú	2323-24	34534

2.4. Alguns Tipos de Dados Aceitos

Char(n) : Onde n é o número de dígitos (caracteres alfanuméricos), o valor deve sempre ser colocado entre aspas. Ex.: 'a'.

Integer : Números inteiros (positivos ou negativos). Ex.: 1, 10, 5000.

Numeric (n,p): Onde n é o número de dígitos e p é a precisão. Ex.: 125,43.

Date : Data.

Time : Hora.

2.5. Definindo Chave Primária (Primary Key)

Chave primária é uma coluna ou conjunto de colunas (chave composta), que identifica, de forma única, os demais dados de uma linha. Por exemplo: na tabela **funcionários** o `cod_func` identifica cada linha. Assim, duas linhas não podem ter a mesma chave primária.

Na tabela de **deptos**, a coluna `cod_depto` identifica cada linha, da mesma forma, funcionários. Duas linhas não podem ter o mesmo valor na coluna `cod_depto`.

Na tabela **bancos**, a coluna `cod_banco` é quem identifica cada linha, funcionando como as tabelas anteriores.

2.6. Chave Estrangeira (Foreign Key)

As tabelas podem se relacionar entre si. Observe que na tabela **funcionários** precisamos saber a qual departamento ele pertence e também em qual banco (se ele trabalhar com banco) ele possui conta. Para isso, temos a coluna onde se armazena o código do departamento (`cod_depto`) e a coluna onde se armazena o código do banco (`cod_banco`).

Como vimos no item anterior, `cod_depto` e `cod_banco` são chaves primárias nas tabelas **deptos** e **bancos** respectivamente. Como essas colunas também existem na tabela de **funcionários**, e seus valores dependem das tabelas **deptos** e **bancos**, dizemos então que elas são chaves estrangeiras na tabela de **funcionários**.

Resumindo, as chaves estrangeiras correspondem ao valor da chave primária em outra tabela. No banco de dados do nosso modelo, se existir algum valor nas colunas `cod_depto` e `cod_banco`, da tabela **funcionários**, deve estar obrigatoriamente nas tabelas **depto** e **bancos** respectivamente.

2.7. Operadores Aritméticos

Operador	Descrição
+	Adição
-	Subtração
*	Multiplificação
/	Divisão

A precedência de operadores segue a mesma regra matemática, ou seja, primeiro serão executadas multiplicação e divisão, para depois adição e subtração.

2.8. Operadores Relacionais ou de Comparação

=	igual a	!=	não igual (diferente)
>	maior que	<>	não igual (diferente)
<	menor que	!>	não maior do que
>=	maior ou igual a	!<	Não menor do que
<=	menor ou igual a		

2.9. Operadores Lógicos

Para explicar operadores lógicos, vamos empregar algo que utilizamos muito em nosso cotidiano, a linguagem falada e escrita.

Observe que nas frases abaixo, os operadores lógicos estão em negrito e são chamados de **E** e **OU**.

“Se meu salário é pouco **E** estou cheio de dívidas, irei para o SPC.”
 1º condição **2º condição**

“Se está chovendo **E** eu estou sem guarda-chuva **E** não tenho nenhum abrigo, irei me molhar.”

“Se sou brasileiro **OU** sou Argentino, então sou sul-americano.”

“Se torço para o Flamengo **OU** torço para o Vasco **OU** torço para o Botafogo, então sou do Rio de Janeiro.”

“Se ganhei na sena **OU** ganhei na loto **OU** ganhei na mega-sena **E** faturei tudo isso sozinho, vou para o Caribe!”

No caso do operador **E**, para que uma determinada afirmação seja verdadeira, todos os membros da frase devem ser verdadeiros, ou seja, o que está a direita e o que está à esquerda do operador **E** devem ser verdadeiros.

Já no caso do operador **OU**, basta que um dos membros da frase seja verdadeiro, como o exemplo da terceira frase.

Para compreensão podemos também utilizar expressões matemáticas.

Ex.:(1) – Idade_func = 18 // uma variável que contém a idade do funcionário.

Se (Idade_func > 18) **E** (Idade_func < 20)
 Então “A idade do funcionário é de 19 anos.”
 Senão “A idade do funcionário não é de 19 anos.”

Perceba nesse exemplo, que as duas afirmações precisam ser verdadeiras para que a expressão seja verdadeira.

Ex.: (2) – Departamento = 10

```
Se (departamento = 1 ) OU (departamento = 2)
OU (departamento=10)
Então "Funcionário pertence ao departamento 1 ou 2 ou 10."
Senão "Funcionário não pertence aos departamentos 1, 2 e 10."
```

Perceba nesse exemplo, que as duas primeiras condições são falsas, porém, a última é verdadeira. Assim, a expressão torna-se verdadeira com o operador **OU**.

Os exemplos acima são apenas para compreensão dos operadores, pois, é assim que funciona na SQL. Contudo, todos os comandos em SQL são escritos em inglês, inclusive esses operadores. Quando você utilizar operadores lógicos em SQL, escreva **AND**, que significa **E** em inglês e **OR** que significa **OU**.

2.10. Extraindo Informações em um Banco de Dados

Como vimos, as informações de um banco de dados ficam armazenadas dentro de tabelas, que por sua vez possuem campos. Sendo assim, quando quisermos extrair informações, ou seja, saber por exemplo, em que departamento um determinado funcionário está, precisamos primeiramente saber qual o banco de dados, qual a tabela que representa os funcionários e em qual campo (ou coluna) está especificamente o código do departamento do funcionário.

Vamos então visualizar as tabelas anteriormente listadas, para tentar extrair a seguinte informação:

A data de admissão do funcionário cujo código é igual a 5.

Primeiramente, deve-se extrair do banco de dados a informação chamada “data de admissão”. Depois extraí-la também da tabela de **funcionários**. Porém, se tem uma condição para isso, só deverá ser extraído a data de admissão do funcionário que possui seu código igual a 5. Visualmente, direcionamos os olhos para a tabela de **funcionários**, para a coluna `cod_func` e intuitivamente selecionamos a linha onde o código é igual a 5. Direcionamos então os olhos para a direita até encontrarmos a coluna que possui a data de admissão. Pronto! Achamos o que queríamos!

Tente novamente. Agora quero saber o nome do departamento do funcionário que possui o código 3.

Da mesma forma, com os olhos, vamos à tabela de **funcionários** e na coluna `cod_func` selecionamos a linha que possui o código 3. Se direcionarmos os olhos para a direita, não encontraremos o nome do departamento, que é justamente o que estamos pedindo acima. Contudo, apesar de não termos o nome, pelo menos temos o código do mesmo, pois sabemos que em nosso modelo existe uma outra tabela, com o objetivo de armazenar os departamentos.

Após selecionar o funcionário de código 3, vamos até a coluna `cod_depto` e selecionamos o código que está contido. Com esse código, direcionamos os olhos para a

tabela **depto**, e na coluna `cod_depto`, selecionamos a linha que possui o valor igual ao que estava na coluna `cod_depto`, da tabela de **funcionários**. Uma vez selecionado, basta identificar qual o nome do departamento.

Realizados esses procedimentos, conseguimos extrair informações de nossas tabelas.

Devemos observar que sempre que quisermos extrair informações, precisamos previamente saber: qual a informação (qual(is) coluna(s)), de onde ela virá (qual(is) tabela(s)) e se existirá alguma condição para extraí-las.

Utilizando SQL para extrair informações de um banco de dados, funciona exatamente igual ao exercício que fizemos acima, porém podemos até fazer isso mentalmente para exemplos simples, mas, para que se concretize o que pensamos, devemos digitar alguns comandos e depois mandá-los executar.

3. Exercícios de Revisão

3.1. Exercícios de Banco de Dados

1.) Descreva o que você entende por Banco de Dados.

2.) Defina o que é uma tabela.

3.) Defina o que é a linha e a coluna de uma tabela.

4.) Defina a linguagem SQL.

5.) Quais os comandos de definição e manipulação de linguagem e para que servem?

6.) Para que utilizamos a chave primária?

7.) Relacione os operadores a seguir:

(1) Aritmético (2) Lógico (3) Relacional

- | | |
|---------|--------|
| () = | () - |
| () + | () > |
| () !< | () OR |
| () / | () !> |
| () >= | () * |
| () AND | () <= |
| () != | () < |
| () <> | |

3.2. Exercícios do Modelo Proposto

1.) Que tabela é usada para informar o salário dos funcionários?

2.) Que tipo de dado é utilizado no campo Nome da tabela **funcionário**?

3.) Que tipo de dado é utilizado no campo dt_admissão da tabela **funcionário** e qual o tipo de dado é utilizado no campo agência da tabela **bancos**?

4. Consultas Simples

4.1. Introdução

Neste módulo estudaremos comandos de Sintaxe e outros comandos de SQL que aplicaremos no *Gerador de Relatórios*, dando maior ênfase à sintaxe do Select.

4.2. Objetivo

Ao final desse módulo, você estará apto a:

- Selecionar informações no banco de dados;
- Utilização de algumas funções da SQL;
- Agrupar informações via SQL;
- Identificar e manipular valores nulos;
- Relacionamento entre tabelas.

4.3. Sintaxe do Comando SELECT

```
SELECT [distinct {*| tabela.*} [tabela.]campo1 [AS alias1]
[, [tabela.]campo2 [AS alias2] [, ...]]}
FROM expressão_tabela [, ...]
[WHERE condicoes... ]
[GROUP BY nome_coluna ]
[ORDER BY nome_colunas [ ASC | DESC ], ...]
```

<u>Parte</u>	<u>Descrição</u>
distinct	Especifica que todos os campos da tabela ou tabelas especificadas, que tenham valor duplicado, serão exibidas uma única vez, ou seja, na saída da consulta não constam valores duplicados.
*	Especifica que todos os campos da tabela ou tabelas especificadas serão selecionados.
tabela	O nome da tabela que contém os campos dos quais os registros serão selecionados.
campo1, campo2	Os nomes dos campos dos quais os dados serão recuperados. Se você incluir mais de um campo, eles serão recuperados na ordem listada.
alias1, alias2	Os nomes que serão usados como títulos de colunas em vez dos nomes originais das colunas na tabela.
Expressão_tabela	O nome da tabela ou tabelas contendo os dados que você quer recuperar.

4.4. Extraindo Dados

O comando abaixo seleciona todas as colunas de uma tabela.

```
SELECT funcionarios.cod_func,funcionarios.nome,funcionarios.
cod_depto,funcionarios.salario,funcionarios.dt_admissao,
funcionarios.dt_nascto,funcionarios.cod_banco
FROM funcionarios

SELECT fu.cod_func,fu.nome,fu.cod_depto,fu.salario,fu.dt_admissao,
fu.dt_nascto,fu.cod_banco
FROM funcionarios AS fu

SELECT *
FROM funcionarios
```


The screenshot displays the Interactive SQL (ISQL) interface with three main panels:

- Data Panel:** Shows the results of a query. A text box points to this panel with the text: "Aqui você visualiza os dados referentes as consultas realizadas por você através dos comandos digitados na última janela desta tela."

cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
1	João Anastácio	1	500.0000	1995-01-01	1975-01-02 15:12:13.000	1
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01 17:12:14.000	2
3	Manoel da Rosa Pinto	3	1500.0000	1990-05-02	1930-10-08 12:00:00.000	3
4	João Malha	4	189.0000	1999-04-10	1988-04-04 13:00:00.000	4
5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10 13:00:00.000	(NULL)
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11 13:00:00.000	(NULL)
7	Raimundo Pinto	4	180.0000	1985-02-11	1967-02-11 14:00:00.000	(NULL)
- Statistics Panel:** Shows "7 rows in query (I/O estimate 1)" and "PLAN> funcionarios (seq)". A text box points to this panel with the text: "Aqui você visualiza as estatísticas dos comandos realizados e também verifica se seu comando funcionou."
- Command Panel:** Shows the SQL command: "SELECT funcionarios.cod_func,funcionarios.nome,funcionarios. cod_ funcionarios.dt_nascto,funcionarios.cod_banco FROM funcionarios". A text box points to the "Execute" button with the text: "Aqui você vê o comando feito em ISQL, após digitar o comando clica-se no botão Execute."

O comando abaixo seleciona as colunas (cod_func,nome,cod_depto, dt_admissao) da tabela de **funcionários**.

```
SELECT cod_func,nome,cod_depto,dt_admissao
FROM funcionarios
```



 cod_func	nome	cod_depto	dt_admissao
1	João Anastácio	1	1995-01-01
2	Pedro Brasona	2	2001-02-05
3	Manoel da R. Pizzolo	3	1990-05-02
4	João Malha	4	1999-04-10
5	Antônio Nascimento	3	1985-01-10
6	Maria de Jesus	4	1985-02-11
7	Raimundo Pizzolo	4	1985-02-11

O comando abaixo seleciona as colunas (cod_func, nome) da tabela de **funcionários**, porém, mostra como cabeçalho os apelidos atribuídos a ela com a cláusula AS.

```
SELECT cod_func AS codigo, nome AS descricao
FROM funcionarios
```


ou

```
SELECT codigo=cod_func, descricao=nome
FROM funcionarios
```

 codigo	descricao
1	João Anastácio
2	Pedro Brasona
3	Manoel da R. Pizzolo
4	João Malha
5	Antônio Nascimento
6	Maria de Jesus
7	Raimundo Pizzolo


O comando abaixo seleciona todas as colunas da tabela **funcionários** e mostra os mesmos ordenados por ordem decendente de código.

```
SELECT *
FROM funcionarios
ORDER BY cod_func DESC
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
4	João Malha	4	189.0000	1999-04-10	1988-04-04	4
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1


O comando abaixo seleciona todas as colunas da tabela **funcionários** e mostra os mesmos ordenados por ordem ascendente de nome (ordem alfabética).

```
SELECT *
FROM funcionarios
ORDER BY nome ASC
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
4	João Malha	4	189.0000	1999-04-10	1988-04-04	4
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)

O comando a seguir seleciona todas as colunas da tabela **funcionários** e mostra os mesmos ordenados por ordem ascendente de nome e data de admissão.


```
SELECT *
FROM funcionarios
ORDER BY nome ASC, dt_admissao ASC
```

	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
	1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
	4	João Malha	4	189.0000	1999-04-10	1988-04-04	4
	3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
	6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
	2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
	7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)

4.5. Utilizando Condições para Extração de Dados

O comando abaixo seleciona todas as colunas da tabela **funcionários** que possui código igual a 5.

```
SELECT *
FROM funcionarios
WHERE cod_func = 5
```

	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)


O comando a abaixo seleciona algumas colunas do funcionário que possui código 5.

```
SELECT cod_func, nome, cod_depto, dt_admissao
FROM funcionarios
WHERE cod_func = 5
```

	cod_func	nome	cod_depto	dt_admissao
	5	Antônio Nascimento	3	1985-01-10


O comando abaixo seleciona todas as colunas da tabela **funcionários** com data de admissão maior que 02/05/1990. Quem foi admitido nessa data **não será selecionado**.

```
SELECT *
FROM funcionarios
WHERE dt_admissao > '1990/05/02'
```

	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
	2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
	4	João Malha	4	189.0000	1999-04-10	1988-04-04	4


O comando abaixo seleciona todas as colunas da tabela **funcionários** com data de admissão menor que 02/05/1990. Quem foi admitido nessa data **não será selecionado**.

```
SELECT *
FROM funcionarios
WHERE dt_admissao < '1990/05/02'
```

	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
	6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
	7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)


O comando a seguir seleciona todas as colunas da tabela **funcionários** com data de admissão maior ou igual a 02/05/1990. Quem foi admitido nessa data **SERÁ** selecionado.

```
SELECT *
FROM funcionarios
WHERE dt_admissao >= '1990/05/02'
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
4	João Malha	4	189.0000	1999-04-10	1988-04-04	4


O comando abaixo seleciona todas as colunas da tabela **funcionários** com data de admissão menor ou igual a 02/05/1990. Quem foi admitido nessa data **SERÁ** selecionado.

```
SELECT *
FROM funcionarios
WHERE dt_admissao <= '1990/05/02'
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)


O comando abaixo seleciona todas as colunas da tabela **funcionários** com código do departamento diferente de 4, ou seja, todos os que **não pertencem** ao departamento de código 4.

```
SELECT *
FROM funcionarios
WHERE cod_depto <> 4
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)


O comando abaixo seleciona todas as colunas da tabela **funcionários** que o nome inicie com “João”.

```
SELECT *
FROM funcionarios
WHERE nome LIKE 'Joao%'
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
4	João Malha	4	189.0000	1999-04-10	1988-04-04	4


O comando a seguir seleciona todas as colunas da tabela **funcionários** que o nome termina com “Pizzolo”.

```
SELECT *
FROM funcionarios
WHERE nome LIKE '%Pizzolo'
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)


O comando abaixo seleciona todas as colunas da tabela **funcionários** que o nome possui “de” em alguma posição do nome.

```
SELECT *
FROM funcionarios
WHERE nome LIKE '%de%'
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)

O comando abaixo seleciona todas as colunas da tabela **funcionários** que o nome inicie com a letra “M”, tenha qualquer caracter na segunda posição e as próximas duas letras do nome sejam “ri”. Ex.: “Maria”, “Mário”.


```
SELECT *
FROM funcionarios
WHERE nome LIKE 'M%ri%'
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)

4.6. Aplicando Operadores Lógicos na Cláusula WHERE


O comando abaixo seleciona todas as colunas da tabela **funcionários** cujo o cod_depto é igual a 4 e a dt_admissão é igual a 1985/02/11.

```
SELECT *
FROM funcionarios
WHERE cod_depto = 4 AND dt_admissao = '1985/02/11'
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)


O comando abaixo seleciona todas as colunas da tabela **funcionários** cujo o cod_banco é igual a 2 ou o cod_banco é igual a 4.

```
SELECT *
FROM funcionarios
WHERE cod_banco = 2 OR cod_banco = 4
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
4	João Malha	4	183.0000	1999-04-10	1988-04-04	4

O comando a seguir seleciona todas as colunas da tabela **funcionários** cujo o cod_depto é igual a 4. A data de admissão deve ser igual a 1985/02/11 ou 1985/01/10.


```
SELECT *
FROM funcionarios
WHERE cod_depto = 4 AND dt_admissao = '1985/02/11' OR
dt_admissao = '1985/01/10'
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)

Observe que o comando anterior não funcionou como deveria, pois fez retornar um funcionário que não pertence ao departamento de código 4. Isso ocorreu porque o operador **OR** dividiu a SQL em duas condições. Tudo o que está antes do **OR** é uma condição e tudo o que estiver depois é outra condição. Sendo assim, se uma das duas forem verdadeiras, a linha da tabela será retornada.

O próximo comando mostra como deveria ser feito para que a SQL retornasse corretamente o que foi descrito acima.

```
SELECT *
FROM funcionarios
WHERE cod_depto = 4 AND (dt_admissao = '1985/02/11' OR
dt_admissao = '1985/01/10')
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)


O comando abaixo seleciona todas as colunas da tabela **funcionários** cujo o **cod_func** é igual a 2 ou 3 ou 7.

```
SELECT *
FROM funcionarios
WHERE cod_func = 2 AND cod_func = 3 AND cod_func = 7
```

Observe que o comando anterior não retornou nada, pois quando se utiliza o operador **AND** todas as condições têm de ser verdadeiras para a linha ser retornada.


O próximo comando retorna o que foi descrito anteriormente:

```
SELECT *
FROM funcionarios
WHERE cod_func = 2 OR cod_func = 3 OR cod_func = 7
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)


O comando abaixo seleciona todas as colunas da tabela **funcionários** cujo o **cod_func** é igual a 2 ou 3 ou 7. Sintaxe melhor que a anterior.

```
SELECT *
FROM funcionarios
WHERE cod_func IN (2,3,7)
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)


O comando abaixo seleciona todas as colunas da tabela **funcionários** cujo o **cod_func** é maior e igual a 2 e o **cod_func** é menor e igual a 6.

```
SELECT *
FROM funcionarios
WHERE cod_func >= 2 AND cod_func <= 6
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
4	João Malha	4	189.0000	1999-04-10	1988-04-04	4
5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)

O comando abaixo seleciona todas as colunas da tabela **funcionários** cujo o **cod_func** deve ser estar entre 2 e 6. Sintaxe melhor que a anterior.

```
SELECT *
FROM funcionarios
WHERE cod_func BETWEEN 2 AND 6
```


 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
4	João Malha	4	189.0000	1999-04-10	1988-04-04	4
5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)

4.7. Distinct

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

O comando abaixo mostra na tela todos os salários diferentes que são pagos aos funcionários cadastrados, pois a cláusula **DISTINCT** só mostra valores distintos.

```
SELECT distinct salario
FROM funcionarios
```

 salario
500.0000
800.0000
1500.0000
189.0000
200.0000
180.0000

4.8. Agrupamento por Group By

As funções de grupo operam sobre grupos de linhas. Retornam resultados baseados em grupos de linhas em vez de resultados de funções por linha individual.

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

O comando abaixo mostra na tela o total de salários pagos por departamento. Utiliza a função de agregação **sum()** em conjunto com **group by**.

```
SELECT cod_depto, sum(salario)
FROM funcionarios group by cod_depto
```

cod_depto	sum(salario)
1	500.0000
2	800.0000
3	1700.0000
4	549.0000

O comando abaixo mostra na tela o total de funcionários por departamento. Utiliza a função de agregação count() em conjunto com group by.

```
SELECT cod_depto, count(*)
FROM funcionarios group by cod_depto
```

cod_depto	"count"(*)
1	1
2	1
3	2
4	3

4.9. Null

O valor nulo (NULL) é um valor especial dentro de um banco de dados. Diferente de qualquer outro tipo de dado, esse valor indica que o conteúdo de determinada coluna (ou campo) em uma determinada linha (registro) não é conhecido, ou seja, para um valor NULL não existe valor designado.

Qualquer cálculo envolvendo alguma operação matemática, quando um dos membros da expressão contenha o valor nulo, o resultado será nulo (IS NULL).

Ex.: Vamos supor que tenhamos três colunas: vlr_irrf = 10, vlr_horas_extras = NULL, vlr_inss = 15.

Fazendo-se a soma (VLR_IRRF + VLR_HORAS_EXTRAS + VLR_INSS) o resultado será nulo (NULL), pois em um dos membros da expressão existe um valor NULL.

Para solucionar esse problema poderíamos antes de somar o valor das horas extras, perguntar se esse valor é nulo ou não. Caso seja, considerar que o valor é zero.

Ex.: (VLR_IRRF + [é nulo VLR_HORAS_EXTRAS? então é igual a zero, senão soma o valor de VLR_HORAS_EXTRAS] + VLR_INSS)


Intuitivamente resolveríamos o problema assim, porém em ISQL uma das maneiras de resolver isto seria utilizando-se uma função. A função chama-se ISNULL e está exemplificada a seguir:

```
(VLR_IRRF + ISNULL(VLR_HORAS_EXTRAS,0) + VLR_INSS)
```

A função ISNULL verifica se o valor de VLR_HORAS_EXTRAS é nulo. Se for, pega o valor zero que está no segundo parâmetro, caso contrário pega o próprio valor do campo VLR_HORAS_EXTRAS.


O comando a seguir mostra na tela todos os funcionários em que o valor da coluna código do banco seja NULL.

```
SELECT *
FROM funcionarios
WHERE cod_banco is null
```

	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
	6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)
	7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)

O comando abaixo mostra o código, nome e código do banco do funcionário. Se o código do banco for NULL, será mostrado zero.

```
SELECT cod_func, nome, isnull(cod_banco, 0)
FROM funcionarios
```


	cod_func	nome	isnull(cod_banco,0)
	1	João Anastácio	1
	2	Pedro Brasona	2
	3	Manoel da R. Pizzolo	3
	4	João Malha	4
	5	Antônio Nascimento	0
	6	Maria de Jesus	0
	7	Raimundo Pizzolo	0

4.10. Negando uma Condição (NOT)

Até agora vimos vários exemplos onde utilizamos condições para selecionar linhas dentro de uma tabela. Para tanto, utilizamos operadores lógicos, operadores de condição, etc. Porém há casos em que é mais fácil eliminar linhas que não se deseja mostrar do que criar uma seleção para mostrar somente as linhas que desejamos. Vejamos o exemplo:


Vamos supor que se deseje mostrar os funcionários que possuem os códigos 2 e 7. O melhor comando a ser utilizado seria:

```
SELECT *
FROM funcionarios
WHERE cod_func in (2,7)
```

	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
	7	Raimundo Pizzolo	4	180.0000	1985-02-11	1967-02-11	(NULL)


Mas vamos supor então que se deseje mostrar todos os funcionários, exceto os que possuem códigos 2 e 7.

```
SELECT *
FROM funcionarios
WHERE cod_func <> 2 AND cod_func <> 7
```

	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
	3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
	4	João Malha	4	189.0000	1999-04-10	1988-04-04	4
	5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
	6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)

Contudo, para operações onde é mais fácil excluir as linhas do que criar condições para selecioná-las, podemos utilizar o operador **NOT** para negar a condição. Ex.:

```
SELECT *
FROM funcionarios
WHERE cod_func NOT in (2,7)
```



	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
	3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
	4	João Malha	4	189.0000	1999-04-10	1988-04-04	4
	5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)
	6	Maria de Jesus	4	180.0000	1985-02-11	1967-02-11	(NULL)

Tanto a forma anterior quanto essa funcionam, mas verifique que a segunda é menor e mais clara. Irá selecionar todos os funcionários onde o código não seja 2 ou 7.

Outros exemplos:


O comando abaixo mostra na tela todos os funcionários em que o valor da coluna código do banco não seja NULL.

```
SELECT *
FROM funcionarios
WHERE cod_banco is not null
```

	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
	2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
	3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
	4	João Malha	4	189.0000	1999-04-10	1988-04-04	4

O comando abaixo mostra todos os funcionários que não pertencem ao departamento de código 4.

```
SELECT *
FROM funcionarios
WHERE not cod_depto = 4 → nesse caso seria melhor o operador <> (diferente)
```


	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
	1	João Anastácio	1	500.0000	1995-01-01	1975-01-02	1
	2	Pedro Brasona	2	800.0000	2001-02-05	1980-12-01	2
	3	Manoel da R. Pizzolo	3	1500.0000	1990-05-02	1930-10-08	3
	5	Antônio Nascimento	3	200.0000	1985-01-10	1967-01-10	(NULL)

4.11. Sub-Consultas (Subqueries)

Sub-consulta é um comando “SELECT”, que é aninhado dentro de outro “SELECT” e que devolve resultados intermediários. Uma sub-consulta acontece quando temos uma SQL principal que pode retornar “n” linhas, e para cada linha retornada nessa SQL principal é executado um ou mais SQL que retornarão sempre um único valor, pois em sub-consulta não é permitido retornar mais de uma linha.

Exemplo(1): Vamos supor que eu deseje imprimir o código do departamento, o nome do departamento e o total de funcionários de cada departamento, porém esse total quero obter através de uma sub-consulta. Podemos fazer assim:


```
SELECT cod_depto,nome, total_funcs =
      (SELECT count()
       FROM funcionarios
       WHERE funcionarios.cod_depto=deptos.cod_depto)
FROM deptos
```

 cod_depto	nome	total_func
1	Recursos Humanos	1
2	Desenvolvimento	1
3	Financeiro	2
4	Editoração	3

Observe que criamos uma coluna temporária para armazenar o total de funcionários por departamento.

Exemplo(2): Vamos supor que se deseje mostrar o código e nome do funcionário juntamente com a agência e conta bancária. Essas duas últimas informações obter através de sub-consultas.

```
SELECT  cod_func, nome,
agencia= (SELECT agencia
          FROM  bancos
          WHERE  bancos.cod_banco = funcionarios. cod_banco),
conta_bancaria = (SELECT numero_banco
                  FROM  bancos
                  WHERE  bancos.cod_banco = funcionarios.cod_banco)
          FROM  funcionarios
```

 cod_func	nome	agencia	conta_bancaria
1	João Anastácio	0407-3	6620-6
2	Pedro Brasona	500-4	1246-88
3	Manoel da R. Pizzolo	345-1	1232-2
4	João Malha	2323-24	34534
5	Antônio Nascimento	(NULL)	(NULL)
6	Maria de Jesus	(NULL)	(NULL)
7	Raimundo Pizzolo	(NULL)	(NULL)

Não existe segredo em utilizar sub-consultas. O único cuidado que tem que se tomar é fazer corretamente os relacionamentos, pois se esses não forem executados corretamente a sub-consulta poderá retornar mais de uma linha ocasionando assim um erro.

Os dois exemplos acima mostram o uso de sub-consultas, mas você pode observar que poderia ser feito sem utilizá-las. Pelo modelo apresentado ser extremamente simples, não existem muitos casos em que possa utilizar sub-consultas.

As sub-consultas são de grande utilidade e, em muitos casos, é extremamente vantajoso utilizá-las a relacionar várias tabelas no select principal, pois geralmente elas são executadas mais rapidamente.

4.12. Sub-Consultas na Cláusula Where do Select Principal

A utilização de sub-consulta na cláusula WHERE do SELECT principal é muito comum e segue as mesmas regras de sub-consultas vistas até então, diferenciando apenas o local onde a mesma é colocada.

Até agora estamos acostumados a utilizar as sub-consultas antes da cláusula FROM do comando SELECT. Vamos ver então a utilização nesse mesmo comando, porém depois da cláusula WHERE. Um modo muito fácil dessa utilização é em conjunto com a condição EXISTS, porém veremos outros modos que são equivalentes e inclusive foram vistos anteriormente.

4.13. A Condição EXISTS

A sintaxe da condição EXISTS é a seguinte:

.... EXISTS (sub-consulta)

A condição EXISTS é verdadeira se o resultado da sub-consulta retornar no mínimo uma linha, caso contrário a mesma será falsa.

Quando utilizamos uma sub-consulta em conjunto com a condição EXISTS não teremos aquele conhecido problema da mesma retornar mais de uma linha. Isso acontece porque a condição EXISTS é uma condição lógica e só pode retornar um valor, no caso, verdadeiro ou falso.

Sendo assim, enquadra-se no conceito visto anteriormente, onde vimos que uma sub-consulta só pode retornar um valor singular, ou seja, uma única linha.

No exemplo abaixo, o select deve retornar somente os bancos que tenham algum funcionário vinculado a ele, ou seja, somente os bancos que o código esteja em alguma linha da tabela **funcionários**.

```
SELECT *
FROM bancos
WHERE EXISTS
      (SELECT funcionarios.cod_banco FROM funcionarios
      WHERE funcionarios.cod_banco = bancos.cod_banco)
```



cod_banco	nome	agencia	numero_banco
1	Banco do Brasil	0407-3	6620-6
2	Bradesco	500-4	1246-88
3	Bradesco	345-1	1232-2
4	Itaú	2323-24	34534

Da mesma forma podemos fazer para tabela de **deptos**, conforme exemplo a seguir:


```
SELECT *
FROM deptos
WHERE EXISTS (SELECT funcionarios.cod_depto
              FROM funcionarios
              WHERE funcionarios.cod_depto = deptos.cod_depto)
```




cod_depto	nome
1	Recursos Humanos
2	Desenvolvimento
3	Financeiro
4	Editoração

Comandos equivalentes:

```
SELECT *
FROM bancos
WHERE (SELECT count()
      FROM funcionarios
      WHERE funcionarios.cod_banco = bancos.cod_banco) > 0
```


 cod_banco	nome	agencia	numero_banco
1	Banco do Brasil	0407-3	6620-6
2	Bradesco	500-4	1246-88
3	Bradesco	345-1	1232-2
4	Itaú	2323-24	34534

```
SELECT *
FROM deptos
WHERE (SELECT count()
FROM funcionarios
WHERE funcionarios.cod_depto = deptos.cod_depto) > 0
```


 cod_depto	nome
1	Recursos Humanos
2	Desenvolvimento
3	Financeiro
4	Editoração

ou

```
SELECT *,
TOTFUNC = (SELECT count()
FROM funcionarios
WHERE funcionarios.cod_banco = bancos.cod_banco)
from bancos where totfunc > 0
```

 cod_banco	nome	agencia	numero_banco	TOTFUNC
1	Banco do Brasil	0407-3	6620-6	1
2	Bradesco	500-4	1246-88	1
3	Bradesco	345-1	1232-2	1
4	Itaú	2323-24	34534	1

```
SELECT *,
TOTFUNC = (SELECT count()
FROM funcionarios
WHERE funcionarios.cod_depto = deptos.cod_depto)
from deptos where totfunc > 0
```

 cod_depto	nome	TOTFUNC
1	Recursos Humanos	1
2	Desenvolvimento	1
3	Financeiro	2
4	Editoração	3


Deve-se tomar cuidado para sempre que utilizar uma sub-consulta na cláusula WHERE do SELECT principal e essa sub-consulta não estiver em conjunto com a condição EXISTS, utilizar funções de agregação, para impedir que a sub-consulta possa retornar mais de uma linha e assim ocorrer erro na execução. Caso não deseje utilizar uma função de agregação, deve-se então ter certeza que o resultado da sub-consulta vai retornar sempre uma linha apenas.

4.14. Uniões (Union)

As uniões também são de extrema importância para relatórios mais complexos, em que não podemos mostrar tudo que queremos em uma mesma SQL. As Uniões podem ser formadas por várias SQL's que resulta na soma de todas as linhas executadas por cada SQL.

Vamos supor que precisemos gerar uma SQL que mostre primeiro todos os funcionários seguidos de todos os departamentos e depois todos os bancos, ordenados por código.

Cada SQL deverá ser identificado, por um número, iniciando do número 1.

SELECT	identificador = 1, cod_func, nome		identificador	cod_func	nome
FROM	funcionarios		1	1	João Anastácio
			1	2	Pedro Brasona
			1	3	Manoel da R. Pizzolo
UNION All			1	4	João Malha
			1	5	Antônio Nascimento
			1	6	Maria de Jesus
SELECT	identificador = 2, cod_depto, nome	1	7	Raimundo Pizzolo	
FROM	deptos	2	1	Recursos Humanos	
		2	2	Desenvolvimento	
UNION All		2	3	Financeiro	
		2	4	Editoração	
		3	1	Banco do Brasil	
SELECT	identificador = 3, cod_banco, nome	3	2	Bradesco	
FROM	bancos	3	3	Bradesco	
ORDER BY	1	3	4	Itaú	

Perceba que o número de colunas de cada SQL tem que ser idêntico para todos. Nesse exemplo, todas as linhas geradas pela primeira SQL terão identificador = 1, pela segunda SQL terão identificador = 2 e pela terceira SQL identificador = 3. Como a cláusula ORDER BY está utilizando a ordem pela primeira coluna (identificador), primeiro virão listados todos os funcionários seguidos de todos os departamentos e depois todos os bancos.

Basicamente, quem sabe trabalhar com SQL não terá dificuldade em utilizar uniões.

5. Funções

Pode-se dizer que em computação as funções são nossas melhores amigas. Quando se tem algum problema, quase sempre tem uma função para salvar a “pátria”. Além de evitar que se digite vários comandos para chegar a um determinado resultado, as funções tornam a SQL mais clara reduzindo o seu tamanho, e são geralmente mais rápidas na execução.

Em SQL existem várias funções para manipulação de datas, tempo, números, caracteres alfanuméricos, etc.

As funções sempre retornam um valor singular (apenas um valor) e podem receber ou não parâmetros, dependendo qual o objetivo da função.

Quem já trabalhou com planilhas eletrônicas (como Excel) deve ter visto e utilizado funções. Apesar de não terem nenhuma relação com as funções em SQL, o conceito de passagem de parâmetros e retorno de valores é o mesmo.

Se alguém ainda não entendeu o que são parâmetros, podemos exemplificar mais. Vamos supor que você queira saber o valor do SENO de um número. Você pega a calculadora e utiliza a função Sin (SENO), porém precisa passar qual o valor que você quer saber o SENO. Esse valor é o parâmetro. Já se você quiser saber qual a data de hoje, basta ir ao calendário e olhar, sem que precise enviar nenhum parâmetro. Assim funciona em SQL.

A sintaxe de uma função geralmente segue a regra a seguir:

```
NOME_DA_FUNCAO([<parametro1,parametro2..parametro>])  
--> os parâmetros são opcionais, por isto estão entre colchetes.
```

onde:

```
NOME_DA_FUNCAO: é o nome da função que será executada.  
Parâmetro(1,2,N): são possíveis valores que deverão ser enviados.
```

Ex.: TODAY() → essa é uma função que retorna a data atual. Note que ela não possui parâmetros.

COS(<número>) → essa é uma função que retorna o coseno de um número. Note que o número não está entre colchetes, portanto é obrigatório.

Nota: Se você executar em SQL uma função que deveria obrigatoriamente enviar parâmetros será gerado o seguinte erro:


“*wrong number of parameter to funcion ‘cos’*” (número de parâmetros incorreto na função ‘cos’)

5.1. Funções de Agregação

Propósito: Agrupar as informações de forma resumida de um número de linhas determinado pela SQL.


Os comandos abaixo retornam o número de linhas de suas respectivas tabelas.

```
SELECT count()
FROM funcionarios
```




Function	Result
count(*)	7

```
SELECT count()
FROM deptos
```



Function	Result
count(*)	4


```
SELECT count()
FROM bancos
```



Function	Result
count(*)	4

O comando abaixo totaliza o salário de todos os funcionários.


```
SELECT sum(salario)
FROM funcionarios
```



Function	Result
sum(salario)	3549.00

O comando abaixo totaliza o salário de todos os funcionários que pertencem ao departamento de código 4.


```
SELECT sum(salario)
FROM funcionarios
WHERE cod_depto = 4
```



Function	Result
sum(salario)	549.00

O comando abaixo calcula a média de salários.


```
SELECT avg(salario)
FROM funcionarios
```



Function	Result
avg(salario)	507.0000000

O comando abaixo retorna o menor salário dentre os funcionários cadastrados.


```
SELECT min(salario)
FROM funcionarios
```



Function	Result
min(salario)	180.00

O comando abaixo retorna o maior salário dentre os funcionários cadastrados.


```
SELECT max(salario)
FROM funcionarios
```



Function	Result
max(salario)	1500.00

O comando abaixo retorna uma lista de códigos de funcionários (separados por vírgula) que pertencem ao departamento 4.

```
SELECT list(cod_func)
FROM funcionarios
WHERE cod_depto = 4
```




Function	Result
list(cod_func)	4,6,7

5.2. Funções de Data e Hora

Propósito: Permite a conversão, extração e manipulação de informações do tipo data e hora.

O comando abaixo mostra na tela a data da máquina:

```
SELECT today()
```

 **today(*)**
2004-02-03


O comando abaixo mostra na tela a data e hora da máquina:

```
SELECT getdate()
```

 **getdate(*)**
2004-02-03 09:11:05.433


O comando abaixo converte a data e hora para apenas data:

```
SELECT date(getdate())
```

 **"date"(getdate(*)**
2004-02-03


O comando abaixo mostra na tela o nome do dia da semana referente à data da máquina, porém em inglês:

```
SELECT dayname(today())
```

 **dayname(today(*)**
Tuesday


O comando abaixo mostra na tela o nome do mês referente à data da máquina, porém em inglês:

```
SELECT monthname(today())
```

 **monthname(today(*)**
February

O comando abaixo mostra na tela o código e nome do funcionário e o ano da sua data de admissão:


```
SELECT cod_func, nome, dt_admissao, year(dt_admissao)
FROM funcionarios
```



cod_func	nome	dt_admissao	year(dt_admissao)
1	João Anastácio	1995-01-01	1995
2	Pedro Brasona	2001-02-05	2001
3	Manoel da R. Pizzolo	1990-05-02	1990
4	João Malha	1999-04-10	1999
5	Antônio Nascimento	1985-01-10	1985
6	Maria de Jesus	1985-02-11	1985
7	Raimundo Pizzolo	1985-02-11	1985

O comando abaixo mostra na tela o código e nome do funcionário e o ano da sua data de admissão acrescido de 1:


```
SELECT cod_func, nome, dt_admissao, years(dt_admissao, 1)
FROM funcionarios
```



cod_func	nome	dt_admissao	years(dt_admissao, 1)
1	João Anastácio	1995-01-01	1996-01-01 00:00:00.000
2	Pedro Brasona	2001-02-05	2002-02-05 00:00:00.000
3	Manoel da R. Pizzolo	1990-05-02	1991-05-02 00:00:00.000
4	João Malha	1999-04-10	2000-04-10 00:00:00.000
5	Antônio Nascimento	1985-01-10	1986-01-10 00:00:00.000
6	Maria de Jesus	1985-02-11	1986-02-11 00:00:00.000
7	Raimundo Pizzolo	1985-02-11	1986-02-11 00:00:00.000


O comando abaixo mostra na tela o código e nome do funcionário e o ano da sua data de admissão deduzindo 1.

```
SELECT  cod_func, nome, dt_admissao, years(dt_admissao, -1)
FROM    funcionarios
```

 cod_func	nome	dt_admissao	years(dt_admissao,-1)
1	João Anastácio	1995-01-01	1994-01-01 00:00:00.000
2	Pedro Brasona	2001-02-05	2000-02-05 00:00:00.000
3	Manoel da R. Pizzolo	1990-05-02	1989-05-02 00:00:00.000
4	João Malha	1999-04-10	1998-04-10 00:00:00.000
5	Antônio Nascimento	1985-01-10	1984-01-10 00:00:00.000
6	Maria de Jesus	1985-02-11	1984-02-11 00:00:00.000
7	Raimundo Pizzolo	1985-02-11	1984-02-11 00:00:00.000

O comando abaixo mostra na tela o código e nome do funcionário e há quantos anos o funcionário está admitido:


```
SELECT  cod_func, nome, dt_admissao, years(dt_admissao, current date)
FROM    funcionarios
```

 cod_func	nome	dt_admissao	years(dt_admissao,current dat
1	João Anastácio	1995-01-01	9
2	Pedro Brasona	2001-02-05	2
3	Manoel da R. Pizzolo	1990-05-02	13
4	João Malha	1999-04-10	4
5	Antônio Nascimento	1985-01-10	19
6	Maria de Jesus	1985-02-11	18
7	Raimundo Pizzolo	1985-02-11	18

Quando passamos como parâmetro para a função YEARS duas datas, ela interpreta que queremos saber a diferença entre as datas, sendo assim ela calcula a (segunda data - primeira data) e retorna o resultado em anos. Se as duas datas estiverem compreendidas dentro do mesmo ano, o retorno será zero.


O comando abaixo mostra na tela o código e nome do funcionário e o mês da sua data de admissão.

```
SELECT  cod_func, nome, dt_admissao, month(dt_admissao)
FROM    funcionarios
```

 cod_func	nome	dt_admissao	month(dt_admissao)
1	João Anastácio	1995-01-01	1
2	Pedro Brasona	2001-02-05	2
3	Manoel da R. Pizzolo	1990-05-02	5
4	João Malha	1999-04-10	4
5	Antônio Nascimento	1985-01-10	1
6	Maria de Jesus	1985-02-11	2
7	Raimundo Pizzolo	1985-02-11	2


O comando abaixo mostra na tela o código e nome do funcionário e o mês da sua data de admissão, acrescido de 1.

```
SELECT  cod_func, nome, dt_admissao, months(dt_admissao, 1)
FROM    funcionarios
```

 cod_func	nome	dt_admissao	months(dt_admissao,1)
1	João Anastácio	1995-01-01	1995-02-01 00:00:00.000
2	Pedro Brasona	2001-02-05	2001-03-05 00:00:00.000
3	Manoel da R. Pizzolo	1990-05-02	1990-06-02 00:00:00.000
4	João Malha	1999-04-10	1999-05-10 00:00:00.000
5	Antônio Nascimento	1985-01-10	1985-02-10 00:00:00.000
6	Maria de Jesus	1985-02-11	1985-03-11 00:00:00.000
7	Raimundo Pizzolo	1985-02-11	1985-03-11 00:00:00.000


O comando abaixo mostra na tela o código e nome do funcionário e o mês da sua data de admissão, deduzindo 1.

```
SELECT  cod_func, nome, dt_admissao, months(dt_admissao, -1)
FROM    funcionarios
```

 cod_func	nome	dt_admissao	months(dt_admissao,-1)
1	João Anastácio	1995-01-01	1994-12-01 00:00:00.000
2	Pedro Brasona	2001-02-05	2001-01-05 00:00:00.000
3	Manoel da R. Pizzolo	1990-05-02	1990-04-02 00:00:00.000
4	João Malha	1999-04-10	1999-03-10 00:00:00.000
5	Antônio Nascimento	1985-01-10	1984-12-10 00:00:00.000
6	Maria de Jesus	1985-02-11	1985-01-11 00:00:00.000
7	Raimundo Pizzolo	1985-02-11	1985-01-11 00:00:00.000

O comando abaixo mostra na tela o código e nome do funcionário e há quantos meses o funcionário está admitido.


```
SELECT  cod_func, nome, dt_admissao, months(dt_admissao, today())
FROM    funcionarios
```

 cod_func	nome	dt_admissao	months(dt_admissao,today[*])
1	João Anastácio	1995-01-01	109
2	Pedro Brasona	2001-02-05	35
3	Manoel da R. Pizzolo	1990-05-02	165
4	João Malha	1999-04-10	57
5	Antônio Nascimento	1985-01-10	228
6	Maria de Jesus	1985-02-11	227
7	Raimundo Pizzolo	1985-02-11	227

Quando passamos como parâmetro para a função MONTHS duas datas, ela interpreta que queremos saber a diferença entre as datas, sendo assim ela calcula a (segunda data - primeira data) e retorna o resultado em meses.


O comando abaixo mostra na tela o código e nome do funcionário e o dia da sua data de admissão.

```
SELECT  cod_func, nome, dt_admissao, day(dt_admissao)
FROM    funcionarios
```

 cod_func	nome	dt_admissao	day(dt_admissao)
1	João Anastácio	1995-01-01	1
2	Pedro Brasona	2001-02-05	5
3	Manoel da R. Pizzolo	1990-05-02	2
4	João Malha	1999-04-10	10
5	Antônio Nascimento	1985-01-10	10
6	Maria de Jesus	1985-02-11	11
7	Raimundo Pizzolo	1985-02-11	11


O comando abaixo mostra na tela o código e nome do funcionário e o dia da sua data de admissão, acrescido de 1.

```
SELECT  cod_func, nome, dt_admissao, days(dt_admissao, 1)
FROM    funcionarios
```

 cod_func	nome	dt_admissao	days(dt_admissao,1)
1	João Anastácio	1995-01-01	1995-01-02 00:00:00.000
2	Pedro Brasona	2001-02-05	2001-02-06 00:00:00.000
3	Manoel da R. Pizzolo	1990-05-02	1990-05-03 00:00:00.000
4	João Malha	1999-04-10	1999-04-11 00:00:00.000
5	Antônio Nascimento	1985-01-10	1985-01-11 00:00:00.000
6	Maria de Jesus	1985-02-11	1985-02-12 00:00:00.000
7	Raimundo Pizzolo	1985-02-11	1985-02-12 00:00:00.000


O comando abaixo mostra na tela o código e nome do funcionário e o dia da sua data de admissão, reduzindo 1.

```
SELECT  cod_func, nome, dt_admissao, days(dt_admissao, -1)
FROM    funcionarios
```

 cod_func	nome	dt_admissao	days(dt_admissao,-1)
1	João Anastácio	1995-01-01	1994-12-31 00:00:00.000
2	Pedro Brasona	2001-02-05	2001-02-04 00:00:00.000
3	Manoel da R. Pizzolo	1990-05-02	1990-05-01 00:00:00.000
4	João Malha	1999-04-10	1999-04-09 00:00:00.000
5	Antônio Nascimento	1985-01-10	1985-01-09 00:00:00.000
6	Maria de Jesus	1985-02-11	1985-02-10 00:00:00.000
7	Raimundo Pizzolo	1985-02-11	1985-02-10 00:00:00.000

O comando abaixo mostra na tela o código e nome do funcionário e há quantos dias o funcionário está admitido.

```
SELECT  cod_func, nome, dt_admissao, days(dt_admissao, today())
FROM    funcionarios
```

 cod_func	nome	dt_admissao	days(dt_admissao,today(*))
1	João Anastácio	1995-01-01	3320
2	Pedro Brasona	2001-02-05	1093
3	Manoel da R. Pizzolo	1990-05-02	5025
4	João Malha	1999-04-10	1760
5	Antônio Nascimento	1985-01-10	6963
6	Maria de Jesus	1985-02-11	6931
7	Raimundo Pizzolo	1985-02-11	6931

Quando passamos como parâmetro para a função DAYS duas datas, ela interpreta que queremos saber a diferença entre as datas, sendo assim ela calcula a (segunda data - primeira data) e retorna o resultado em dias.

O comando abaixo mostra na tela o código e nome do funcionário e a hora do seu nascimento.

```
SELECT  cod_func, nome, dt_nascto, hour(dt_nascto)
FROM    funcionarios
```

 cod_func	nome	dt_nascto	hour(dt_nascto)
1	João Anastácio	1975-01-02	15
2	Pedro Brasona	1980-12-01	17
3	Manoel da R. Pizzolo	1930-10-08	12
4	João Malha	1988-04-04	13
5	Antônio Nascimento	1967-01-10	13
6	Maria de Jesus	1967-02-11	13
7	Raimundo Pizzolo	1967-02-11	14


O comando abaixo mostra na tela o código e nome do funcionário e o minuto do seu nascimento:

```
SELECT  cod_func, nome, dt_nascto, minute(dt_nascto)
FROM    funcionarios
```

 cod_func	nome	dt_nascto	minute(dt_nascto)
1	João Anastácio	1975-01-02	12
2	Pedro Brasona	1980-12-01	12
3	Manoel da R. Pizzolo	1930-10-08	0
4	João Malha	1988-04-04	0
5	Antônio Nascimento	1967-01-10	0
6	Maria de Jesus	1967-02-11	0
7	Raimundo Pizzolo	1967-02-11	0


O comando abaixo mostra na tela o código e nome do funcionário e o(s) segundo(s) do seu nascimento:

```
SELECT  cod_func, nome, dt_nascto, second(dt_nascto)
FROM    funcionarios
```

 cod_func	nome	dt_nascto	second(dt_nascto)
1	João Anastácio	1975-01-02	0
2	Pedro Brasona	1980-12-01	0
3	Manoel da R. Pizzolo	1930-10-08	0
4	João Malha	1988-04-04	0
5	Antônio Nascimento	1967-01-10	0
6	Maria de Jesus	1967-02-11	0
7	Raimundo Pizzolo	1967-02-11	0

O comando abaixo mostra na tela todos os funcionários que foram admitidos no ano de 2001:

```
SELECT  *
FROM    funcionarios
WHERE   year(dt_admissao) = 2001
```

 cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banco
2	Pedro Brasona	2	800.00	2001-02-05	1980-12-01	2

5.3. Funções String


Propósito: Efetuar conversões, extrações ou manipulações com strings ou então retornar informações sobre uma string.

Antes de utilizarmos funções para tratamento de strings, devemos saber o que é uma string. String é uma seqüência de caracteres alfanuméricos, ou seja, nessa seqüência podem haver letras, números, dígitos, sinais, etc. Qualquer palavra e letra também é considerada uma string. Todas as strings, quando não estão em seu formato variável (não são variáveis do tipo string), devem estar entre aspas. Ex.: '017.061.079-94', 'Joao', '123456'.

Soma de strings: 'JDK' + '1.2.2' → irá gerar uma string única, unindo as duas anteriores (concatenação): 'JDK1.2.2'.


O comando abaixo mostra na tela todos os departamentos, porém em letras minúsculas:

```
SELECT  lcase(nome)
FROM    deptos
```

 lcase(nome)
recursos humanos
desenvolvimento
financeiro
editoração


O comando abaixo mostra na tela todos os departamentos, porém em letras maiúsculas:

```
SELECT  ucase(nome)
FROM    deptos
```

 ucase(nome)
RECURSOS HUMANOS
DESENVOLVIMENTO
FINANCEIRO
EDITORACÃO

O comando abaixo mostra na tela o código, nome e data de admissão de todos os funcionários:

```
SELECT cod_func, nome, dt_admissao, left(dt_admissao, 4)
FROM funcionarios
```


 cod_func	nome	dt_admissao	"left"(dt_admissao,4)
1	João Anastácio	1995-01-01	1995
2	Pedro Brasona	2001-02-05	2001
3	Manoel da R. Pizzolo	1990-05-02	1990
4	João Malha	1999-04-10	1999
5	Antônio Nascimento	1985-01-10	1985
6	Maria de Jesus	1985-02-11	1985
7	Raimundo Pizzolo	1985-02-11	1985

O último comando onde foi utilizada a função LEFT serve para mostrar o ano em que o funcionário foi admitido.

A função LEFT pega um determinado número de caracteres partindo da esquerda. Esse número é definido no segundo argumento.

O comando abaixo mostra na tela o código, nome e data de admissão de todos os funcionários:

```
SELECT cod_func, nome, dt_admissao, right(dt_admissao, 2)
FROM funcionarios
```


 cod_func	nome	dt_admissao	"right"(dt_admissao,2)
1	João Anastácio	1995-01-01	01
2	Pedro Brasona	2001-02-05	05
3	Manoel da R. Pizzolo	1990-05-02	02
4	João Malha	1999-04-10	10
5	Antônio Nascimento	1985-01-10	10
6	Maria de Jesus	1985-02-11	11
7	Raimundo Pizzolo	1985-02-11	11

O último comando que foi utilizada a função RIGHT serve para mostrar o dia em que o funcionário foi admitido.

A função RIGHT pega um determinado número de caracteres partindo da direita. Esse número é definido no segundo argumento.


O comando abaixo mostra na tela o nome do departamento e quantos caracteres ele possui:

```
SELECT nome, length(nome)
FROM deptos
```

 nome	length(nome)
Recursos Humanos	16
Desenvolvimento	15
Financeiro	10
Editoração	10


O comando abaixo mostra na tela os caracteres (string) que digitamos entre aspas.

```
SELECT 'Teste de Função'
```

 'Teste de Função'
Teste de Função


O comando abaixo mostra na tela os caracteres (string) que digitamos entre aspas, retirando possíveis espaços em branco à esquerda.

```
SELECT ltrim(' Teste de Função')
```

 ltrim(' Teste de Função')
Teste de Função


O comando abaixo mostra na tela os caracteres (string) que digitamos entre aspas, retirando possíveis espaços em branco à direita:

```
SELECT rtrim('Teste de Função ')
```

 **rtrim('Teste de Função ')**
Teste de Função


O comando abaixo mostra na tela os caracteres (string) que digitamos entre aspas, retirando possíveis espaços em branco a direita e a esquerda.

```
SELECT trim(' Teste de Função ')
```

 **trim(' Teste de Função ')**
Teste de Função

O comando abaixo mostra na tela o código, nome e data de admissão do funcionário.

```
SELECT cod_func, nome, dt_admissao, substr(dt_admissao, 6, 2)
FROM funcionarios
```



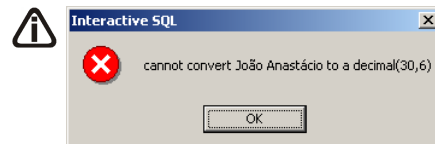
cod_func	nome	dt_admissao	substr(dt_admissao,6,2)
1	João Anastácio	1995-01-01	01
2	Pedro Brasona	2001-02-05	02
3	Manoel da R. Pizzolo	1990-05-02	05
4	João Malha	1999-04-10	04
5	Antônio Nascimento	1985-01-10	01
6	Maria de Jesus	1985-02-11	02
7	Raimundo Pizzolo	1985-02-11	02

O último dado a ser mostrado será o mês da data de admissão do funcionário, sendo que para isto não estamos utilizando a função mais apropriada, que seria a MONTH().


O comando abaixo mostra na tela o código e nome do funcionário separado por um hífen.

Note que esse comando utiliza um operador matemático para a soma de números. Porém, nesse caso não queremos somar números e sim somar letras de forma a unir em uma string única (concatenação). Contudo, pela coluna cod_func ser um número inteiro, a SQL irá “pensar” que estou querendo fazer uma soma de números e não uma soma (concatenação) de caracteres (string) e irá retornar um erro, pois não é possível somar um número mais uma letra. Para solucionar isso, devemos antes transformar (converter) o que é número para caracter (string) para que a SQL saiba que estamos efetuando uma concatenação, conforme segundo exemplo:

```
SELECT cod_func + ' - ' + nome
FROM funcionarios → retornará um erro
```



```
SELECT string(cod_func) + ' - ' + nome
FROM funcionarios
```

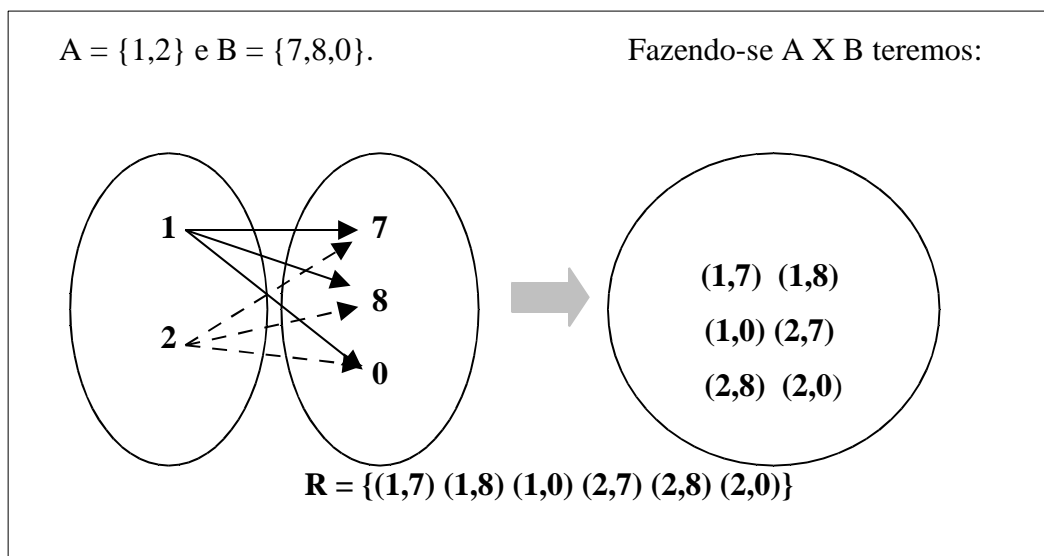
 **string(cod_func)+' - '+nome**
 1 - João Anastácio
 2 - Pedro Brasona
 3 - Manoel da R. Pizzolo
 4 - João Malha
 5 - Antônio Nascimento
 6 - Maria de Jesus
 7 - Raimundo Pizzolo

6. Joins / Outer Joins

6.1. Produto Cartesiano (Relação entre Tabelas)


O objetivo é gerar um subconjunto de um produto cartesiano, baseado em um conjunto de domínios.

Quando relacionamos duas tabelas, ocorre o que matematicamente chamamos de produto cartesiano, ou seja, cada elemento do conjunto domínio formará um par ordenado com um elemento do conjunto imagem. Matematicamente falando, teremos o seguinte exemplo:




Perceba que o comando abaixo está mostrando o código do funcionário, o seu nome e o nome do seu departamento. Nesse caso é obrigatório que se coloque o nome da tabela antes da coluna nome, pois existem colunas com esse nome na tabela de **funcionários** e **deptos**. Sendo assim, a SQL não saberia a que nome você se refere. Outro detalhe e o mais importante, é que esse SQL abaixo retornou para cada funcionário tantas linhas quando existiam na tabela de **deptos**, ou seja, fez exatamente o que foi descrito no diagrama anterior. Considerou a tabela **funcionários** como o conjunto A e a tabela **deptos** como o conjunto B e efetuou um produto cartesiano das duas.

```
SELECT  cod_func,funcionarios.nome,deptos.nome
FROM    funcionarios,deptos
```

 cod_func	nome	nome
1	João Anastácio	Recursos Humanos
1	João Anastácio	Desenvolvimento
1	João Anastácio	Financeiro
1	João Anastácio	Editoração
2	Pedro Brasona	Recursos Humanos
2	Pedro Brasona	Desenvolvimento
2	Pedro Brasona	Financeiro
2	Pedro Brasona	Editoração
3	Manoel da R. Pizzolo	Recursos Humanos
3	Manoel da R. Pizzolo	Desenvolvimento
3	Manoel da R. Pizzolo	Financeiro
3	Manoel da R. Pizzolo	Editoração
4	João Malha	Recursos Humanos
4	João Malha	Desenvolvimento
4	João Malha	Financeiro
4	João Malha	Editoração
5	Antônio Nascimento	Recursos Humanos
5	Antônio Nascimento	Desenvolvimento
5	Antônio Nascimento	Financeiro
5	Antônio Nascimento	Editoração
6	Maria de Jesus	Recursos Humanos
6	Maria de Jesus	Desenvolvimento
6	Maria de Jesus	Financeiro
6	Maria de Jesus	Editoração
7	Raimundo Pizzolo	Recursos Humanos
7	Raimundo Pizzolo	Desenvolvimento
7	Raimundo Pizzolo	Financeiro
7	Raimundo Pizzolo	Editoração


Contudo, não queremos que esse tipo de informação seja mostrada na tela, pois imagine um relatório impresso mostrado nesse formato. Para solucionarmos esse problema, devemos efetuar o relacionamento entre as duas tabelas para que retorne apenas uma linha para cada funcionário e com o nome correto do seu departamento. O comando a seguir implementa isto:

```
SELECT cod_func,funcionarios.nome,deptos.nome
FROM funcionarios,deptos
WHERE funcionarios.cod_depto = deptos.cod_depto
```

 cod_func	nome	nome
1	João Anastácio	Recursos Humanos
2	Pedro Brasona	Desenvolvimento
3	Manoel da R. Pizzolo	Financeiro
4	João Malha	Editoração
5	Antônio Nascimento	Financeiro
6	Maria de Jesus	Editoração
7	Raimundo Pizzolo	Editoração

Verifique que o comando anterior funcionou perfeitamente, pois só listou o nome dos departamentos cujo código é igual ao código presente na coluna cod_depto da tabela de **deptos**. O fato de ter feito o relacionamento também eliminou a multiplicidade de linhas (produto cartesiano). O comando abaixo funciona da mesma maneira, porém tem uma vantagem em sua sintaxe, observe:


```
SELECT cod_func,funcionarios.nome,deptos.nome
FROM funcionarios KEY JOIN deptos
```

 cod_func	nome	nome
1	João Anastácio	Recursos Humanos
2	Pedro Brasona	Desenvolvimento
3	Manoel da R. Pizzolo	Financeiro
4	João Malha	Editoração
5	Antônio Nascimento	Financeiro
6	Maria de Jesus	Editoração
7	Raimundo Pizzolo	Editoração

Perceba que o comando anterior não possui o relacionamento na cláusula WHERE, pois esse está sendo efetuado automaticamente pela cláusula KEY JOIN. Cuidado, essa cláusula só funciona perfeitamente quando sabe-se que o relacionamento entre as duas tabelas é direto, ou seja, como é o relacionamento funcionários → departamentos. Nesse relacionamento não pode haver valor na coluna cod_depto de **funcionários** que não exista na tabela **deptos**, inclusive não podem haver valores NULL.

O comando abaixo seleciona o código e nome do departamento, e seleciona automaticamente todos os campos da tabela **funcionários**.

```
SELECT deptos.cod_depto,deptos.nome,funcionarios.*
FROM funcionarios KEY JOIN deptos
```

 cod_depto	nome	cod_func	nome	cod_depto	salario	dt_admissao	dt_nascto	cod_banc
1	Recursos Humanos	1	João Anastácio	1	500.00	1995-01-01	1975-01-02	1
2	Desenvolvimento	2	Pedro Brasona	2	800.00	2001-02-05	1980-12-01	2
3	Financeiro	3	Manoel da R. Pizzolo	3	1500.00	1990-05-02	1930-10-08	3
4	Editoração	4	João Malha	4	189.00	1999-04-10	1988-04-04	4
3	Financeiro	5	Antônio Nascimento	3	200.00	1985-01-10	1967-01-10	(NULL)
4	Editoração	6	Maria de Jesus	4	180.00	1985-02-11	1967-02-11	(NULL)
4	Editoração	7	Raimundo Pizzolo	4	180.00	1985-02-11	1967-02-11	(NULL)

6.2. Relacionamento (Outer Joins)


Existem ocasiões onde relacionamos mais de uma tabela e que o conteúdo da coluna que foi relacionada não existe na outra tabela.

Em nosso exemplo, na relação funcionário/departamentos, por ser obrigatório todo funcionário ter um departamento, não existe valor que esteja na coluna cod_depto de **funcionários** que não esteja na tabela de **deptos**. Mas se observarmos a coluna cod_banco verificamos que alguns funcionários possuem nessa coluna o valor NULL. Isto se faz necessário porque em nosso modelo pode acontecer de algum funcionário não trabalhar com nenhum banco e nesse caso não há necessidade de informá-lo.

Vamos supor então que eu queira listar o código, nome do funcionário e nome do banco de todos os funcionários. Para os funcionários que não trabalharem com banco, o nome do mesmo não será mostrado, mas o código e nome do funcionário sim.

Conforme visto até agora, utilizaríamos:

```
SELECT cod_func,funcionarios.nome,bancos.nome
FROM funcionarios,bancos
WHERE funcionarios.cod_banco = bancos.cod_banco
```


 cod_func	nome	nome
1	João Anastácio	Banco do Brasil
2	Pedro Brasona	Bradesco
3	Manoel da R. Pizzolo	Bradesco
4	João Malha	Itaú

Esse comando possui um problema. Ele só trará os funcionários que possuírem algum valor no campo cod_banco da tabela de **funcionários**, em função da relação funcionário → banco, que diz que o código do banco existente na tabela de **funcionários** deve ser igual a um código existente na tabela de **bancos**. Porém, como vimos, o valor NULL, comparado a

qualquer coisa é NULL, e no caso desses relacionamentos serão desprezados os funcionários que possuem banco igual a NULL.

Para solucionar esse problema, utilizaremos o mesmo comando acima com uma pequena diferença:

```
SELECT cod_func,funcionarios.nome,bancos.nome
FROM funcionarios,bancos
WHERE funcionarios.cod_banco *= bancos.cod_banco
```


 cod_func	nome	nome
1	João Anastácio	Banco do Brasil
2	Pedro Brasona	Bradesco
3	Manoel da R. Pizzolo	Bradesco
4	João Malha	Itaú
5	Antônio Nascimento	(NULL)
6	Maria de Jesus	(NULL)
7	Raimundo Pizzolo	(NULL)

Perceba que no relacionamento existe um asterisco (*). Esse asterisco, internamente comunica à SQL que:

“Traga-me todas as linhas da tabela à esquerda (**funcionários**) mesmo que não exista um valor relacionado na tabela da direita.”


Mas e se a SQL estivesse dessa forma, primeiro a tabela **bancos** e depois a tabela **funcionários**?

```
SELECT cod_func,funcionarios.nome,bancos.nome
FROM funcionarios,bancos
WHERE bancos.cod_banco = funcionarios.cod_banco
```

 cod_func	nome	nome
1	João Anastácio	Banco do Brasil
2	Pedro Brasona	Bradesco
3	Manoel da R. Pizzolo	Bradesco
4	João Malha	Itaú


Então faríamos isto:

```
SELECT cod_func,funcionarios.nome,bancos.nome
FROM funcionarios,bancos
WHERE bancos.cod_banco =* funcionarios.cod_banco
```

 cod_func	nome	nome
1	João Anastácio	Banco do Brasil
2	Pedro Brasona	Bradesco
3	Manoel da R. Pizzolo	Bradesco
4	João Malha	Itaú
5	Antônio Nascimento	(NULL)
6	Maria de Jesus	(NULL)
7	Raimundo Pizzolo	(NULL)


Os comandos a seguir são similares aos com asterisco e fazem o relacionamento automaticamente:

```
SELECT cod_func,funcionarios.nome,bancos.nome
FROM funcionarios LEFT OUTER JOIN bancos ( *= )
```



cod_func	nome	nome
1	João Anastácio	Banco do Brasil
2	Pedro Brasona	Bradesco
3	Manoel da R. Pizzolo	Bradesco
4	João Malha	Itaú
5	Antônio Nascimento	(NULL)
6	Maria de Jesus	(NULL)
7	Raimundo Pizzolo	(NULL)

```
SELECT  cod_func,funcionarios.nome,bancos.nome
FROM    bancos RIGHT OUTER JOIN funcionarios (=*)
```



cod_func	nome	nome
1	João Anastácio	Banco do Brasil
2	Pedro Brasona	Bradesco
3	Manoel da R. Pizzolo	Bradesco
4	João Malha	Itaú
5	Antônio Nascimento	(NULL)
6	Maria de Jesus	(NULL)
7	Raimundo Pizzolo	(NULL)

7. Exercícios de Revisão 2

- 1 – Selecionar todas as informações da tabela de **funcionários** em ordem decrescente de salários somente para os funcionários que foram admitidos em 1985.
- 2 – Selecionar todas as informações da tabela de **funcionários** e mostrar também na tela a idade de cada um, sendo que no cabeçalho da SQL deve constar “Idade”.
- 3 – Selecionar o código e nome do departamento, e quantos funcionários existem em cada departamento com admissão em 1985. Deve estar ordenado alfabeticamente por nome do departamento.
- 4 – Selecione todos os bancos que tenham em alguma posição a palavra “Brasil”. No cabeçalho deve aparecer “Codigo_do_Banco” e “Nome_do_Banco”.
- 5 – Selecione todos os funcionários que tenham sido admitidos depois do dia primeiro de 1990, que tenham salário maior que R\$ 500 reais e nascido ao meio dia.
- 6 – Selecione todos os funcionários que não possuem banco. O funcionário que não possui banco tem valor *NULL* na coluna *cod_banco*.
- 7 – Selecione todos os funcionários do departamento “Financeiro”. Não pode utilizar o código do departamento para a seleção dos funcionários.
- 8 – Selecione todos os funcionários em que o salário seja maior ou igual a R\$ 200,00 e menor ou igual a R\$ 500,00.
- 9 – Mostre na tela o nome do dia da semana que cairá o seu aniversário.
- 10 – Selecione todos os funcionários que trabalham com o seguintes bancos: (1,3,4)
- 11 – Selecione o nome e salário acrescido de 2% para os funcionários que trabalham com banco e não pertençam ao departamento de código 4. O nome do funcionário deve estar escrito em minúsculo.
- 12 – Mostre o código do funcionário, nome do funcionário, código do banco, nome do banco, código do departamento e nome do departamento. O nome do departamento deve estar sempre em maiúsculo e o nome do banco sempre em minúsculo. A SQL deve estar ordenado por código do funcionário. Se for preciso utilizar o nome da tabela antes do campo, deve-se utilizar um apelido.
- 13 – Mostrar qual o menor e maior código de departamento cadastrado.
- 14 – Mostrar o código, nome e data de nascimento do funcionário, sendo que essa data deve desprezar a hora/minuto/segundo. Somente devem ser mostrados os funcionários que tiverem mais de 21 anos.

15 – Mostrar o código e nome do funcionário. Também deve ser mostrado o dia/mês/ano do nascimento do funcionário no seguinte formato: Dia: dd Mês: mm Ano: yyyy.

16 – O comando a seguir selecionará o que em sua tabela?

```
SELECT nome, salário _____  
FROM funcionários _____  
WHERE cod_depto _____  
IN ('1','2') _____  
_____
```

8. Caderno de Respostas

8.1. Exercícios de Revisão – Banco de Dados

R: (1) – Conjunto de dados dispostos numa ordem pré-determinada em função de um projeto.

R: (2) – Tabela é um quadro onde se indica alguma “coisa”.

R: (3) – É um conjunto de colunas que ficam dispostas horizontalmente. Todas as linhas que possuem o mesmo número de colunas.

R: (4) – SQL é uma Linguagem destinada à definição e manipulação desses bancos.

R: (5) – DDL – create, alter, drop
 DML – insert, delete, up_date, select

R: (6) – Chave primária é uma coluna ou conjunto de colunas (chave composta) que identifica de forma única os demais dados de uma linha dada. Por exemplo: na tabela **funcionários** o *cod_func* identifica cada linha. Assim, duas linhas não podem ter a mesma chave primária.

R: (7)

(1) Aritmético (2) Lógico (3) Relacional

(3) =	(1) -
(1) +	(3) >
(3) !<	(2) OR
(1) /	(3) !>
(3) >=	(1) *
(2) AND	(3) <=
(3) !=	(3) <
(3) <>	

8.2. Exercícios de Revisão – Modelo Proposto

R: (1) – **funcionários**

R: (2) – *Char*

R: (3) – *Date, Char*

8.3. Exercícios de Revisão 2

R: (1)

```
Select  *
from    funcionarios
where   years(dt_admissao) = 1985
order  by salario DESC
```

R: (2)

```
select *,
'Idade' = years(dt_nascto, today())
from funcionarios
```

R: (3)

```
select deptos.cod_depto, deptos.nome,
Tot_func= (select count()
           from funcionarios
           where funcionarios.cod_depto = deptos.cod_depto and
           year(dt_admissao) = 1985)
           from deptos
order by deptos.nome asc
```

R: (4)

```
select 'Codigo do banco' = bancos.cod_banco,
'Nome do banco' = bancos.nome
from bancos
where nome like '%Brasil%'
```

R: (5)

```
select *
from funcionarios
where dt_admissao > '1990/01/01' and
      salario > 500 and
      hour(dt_nascto) = 12
```

R: (6)

```
select *
from funcionarios
where cod_banco is null
```

R: (7)

```
select *
from funcionarios key join deptos
where deptos.nome = 'Financeiro'
```

ou

```
select *
from funcionarios, deptos
where funcionarios.cod_depto = deptos.cod_depto and
      deptos.nome = 'Financeiro'
```

R: (8)

```
select *
from funcionarios
where salario >= 200 and salario <= 500
```

ou

```
select *
from funcionarios
where salario between 200 and 500
```

R: (9)

```
select dayname('2002/06/24')
```

R: (10)

```
select *
from funcionarios
where cod_banco in (1,3,4)
```

R: (11)

```
select  lcase(nome), (salario * 1.02)
from    funcionarios
where   cod_banco is not null and cod_depto <> 4
```

R: (12)

```
select  cod_func, func.nome, func.cod_banco, ba.nome, func.cod_depto,
        de.nome
from    funcionarios func, bancos ba, deptos de
where   func.cod_banco = ba.cod_banco and func.cod_depto = de.cod_depto
order  by cod_func
```

R: (13)

```
select  max(cod_depto), min(cod_depto)
from    deptos
```

R: (14)

```
select  cod_func, nome, date(dt_nascto)
from    funcionarios
where   years(dt_nascto, today()) > 21
```

R: (15)

```
select  cod_func, nome, 'Dia: ' + right(date(dt_nascto), 2) ,
        'Mês: ' + substr(dt_nascto, 6, 2), 'Ano: ' + left(dt_nascto, 4)
from    funcionarios
```


9. Tabelas Modelo

TABELA DE FUNCIONÁRIOS (funcionários)						
cod_func	nome	cod_depto	salário	dt_admissão	Dt_nascto	cod_banco
1	João Anastácio	1	500,00	1995-01-01	1975-01-02 15:12:13	1
2	Pedro Brasona	2	800,00	2001-02-05	1980-12-01 17:12:14	2
3	Manoel da R. Pizzolo	3	1500,00	1990-05-02	1930-10-08 12:00:00	3
4	João Malha	4	189,00	1999-04-10	1988-04-04 13:00:00	4
5	Antônio Nascimento	3	200,00	1985-01-10	1967-01-10 13:00:00	(NULL)
6	Maria de Jesus	4	180,00	1985-02-11	1967-02-11 13:00:00	(NULL)
7	Raimundo Pizzolo	4	180,00	1985-02-11	1967-02-11 14:00:00	(NULL)

TABELA DE DEPARTAMENTOS (deptos)	
cod_depto	Nome
1	Recursos Humanos
2	Desenvolvimento
3	Financeiro
4	Editoração

TABELA BANCOS (bancos)			
cod_banco	Nome	agencia	numero_banco
1	Banco do Brasil	0407-3	6620-6
2	Bradesco	500-4	1246-88
3	Bradesco	345-1	1232-2
4	Itaú	2323-24	34534