Notas de Aula de Implementação de Banco de Dados

(POL4632)

Professor:
Rodrigo Augusto Barros Pereira Dias

Versão 3.0
2008-1

I N D E X

1 – Conceitos Básicos

1.1. Sistemas de Banco de Dados

Um sistema de banco de dados é basicamente um sistema computadorizado de armazenamento de registros, ou seja, um sistema cujo proósito geral é guardar informações e permitir ao usuário buscar e atualizar essas informações quando solicitado.

As informações podem ser qualquer coisa que tenha significado para o usuário, tudo que seja necessário para auxiliar no processo de tomada de decisões de negócios.

1.2. Arquitetura de S. G. B. D.s (Sistemas Gerenciadores de Banco de Dados)

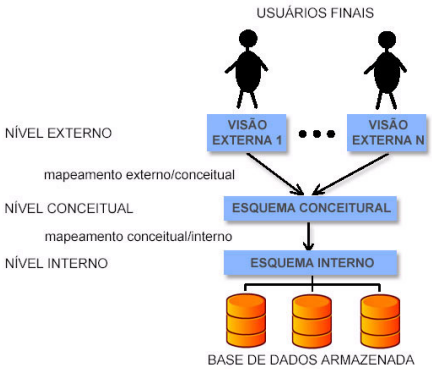
Arquitetura "Three-Schema" (A.K.A. arquitetura ANSI/SPARC - Tsichritzis e Klug, 1978): A meta desta arquitetura é separar as aplicações de usuários da base de dados física. Nesta arquitetura, esquemas podem ser definidos em três níveis:

O nível interno tem um esquema interno que descreve a estrutura de armazenamento físico da base de dados.

O nível conceitual tem um esquema conceitual que descreve a estrutura de toda a base de dados.

O nível externo ou visão possui esquemas externos ou visões de usuários. Cada esquema externo descreve a visão da base de dados de um grupo de usuários da base de dados.

Muitos SGBD's não separam os três níveis completamente. Pode acontecer que alguns SGBD's incluam detalhes do nível interno no esquema conceitual. Em muitos SGBD's que dão suporte à visões, os esquemas externos são especificados com o mesmo modelo de dados usado no nível conceitual. Note-se que os três esquemas são apenas descrições dos dados.



1.3. Arquitetura Cliente-Servidor

A tecnologia cliente/servidor é uma arquitetura na qual o processamento da informação é dividido em módulos ou processos distintos. Um processo é responsável pela manutenção da informação (servidores) e outros responsáveis pela obtenção dos dados (os clientes).

Os processos cliente enviam pedidos para o processo servidor, e este por sua vez processa e envia os resultados dos pedidos.

Nos sistemas cliente/servidor o processamento tanto do servidor como o do cliente são equilibrados, se for gerado um peso maior em um dos dois lados, provavelmente, esse não é um sistema cliente/servidor.

Geralmente, os serviços oferecidos pelos servidores dependem de processamento específico que só eles podem fazer. O processo cliente, por sua vez, fica livre para realizar outros trabalhos. A interação entre os processos cliente e servidor é uma troca cooperativa, em que o cliente é o ativo e o servidor reativo, ou seja o cliente requisita uma operação, e neste ponto o servidor processa e responde ao cliente.

1.3.1. Processos clientes:

O processo de cliente é ativo, ou seja são eles que solicitam serviços a outros programas, os servidores. Normalmente o cliente é dedicado à sessão do usuário, começando e terminando com a sessão. Um cliente pode interagir com um ou mais servidores, mas pelo menos um processo servidor é necessário.

A nível de aplicação, o primeiro ponto a residir no cliente é a interface com o usuário. Algumas tarefas a serem realizadas pelo Cliente: Manipulação de tela, Interpretação de menus ou comandos, Entrada e validação dos dados, Processamento de Ajuda, Recuperação de erro, Manipulação de janelas e Gerenciamento de som e vídeo.

Gerenciando a interação com o usuário, o cliente esconde do usuário o servidor e a rede, caso houver. Para o usuário a impressão é que a aplicação está sendo rodada completamente local.

Se, por acaso, o programa que interage com o usuário fizer simplesmente chamada de rotina, e ficar por conta do servidor todo o processamento este certamente não é um sistema cliente/servidor.

1.3.2. Processos servidores:

Servidores são programas que respondem as solicitações por serviços compartilhados. Ele é um processo reativo, disparado pela chegada de pedidos de seus clientes. Geralmente, o processo servidor roda o tempo todo, oferecendo serviços a muitos clientes.

O processamento do servidor geralmente inclui: acessar, armazenar, organizar os dados compartilhados, atualizar dados previamente armazenados e gerenciamento dos recursos compartilhados.

As aplicações em banco de dados cliente/servidor em sua maioria são montados em cima de banco de dados SQL prontos como Oracle, Informix, Ingress, Sybase, etc.

1.4. Modelo relacional e objeto-relacional

O modelo relacional é um modelo de dados, adequado a ser o modelo subjacente de um SGBD, que se baseia no princípio em que todos os dados estão guardados em tabelas (ou, matematicamente falando, relações). Toda sua definição é teórica e baseada na lógica de predicados e na teoria dos conjuntos.

O conceito foi criado por Edgar Frank Codd em 1970, sendo descrito no artigo "Relational Model of Data for Large Shared Data Banks". Na verdade o modelo relacional foi o primeiro modelo de dados descrito teoricamente, só então os bancos de dados já existentes passaram a ser conhecidos como bancos de dados (modelo hierárquico, modelo em redes e modelo de listas invertidas).

Na década de 90, o modelo baseado na orientação a objeto foi aplicado também aos bancos de dados, criando um novo modelo de programação conhecido como bancos de dados orientados a objeto. Os objetos são valores definidos segundo classes, ou tipos de dados complexos, com seus próprios operadores (métodos).

Com o passar do tempo, os sistemas gestores de bancos de dados orientados a objeto e os bancos de dados relacionais baseados na linguagem SQL se aproximaram. Muitos sistemas orientados a objeto são implementados sobre bancos de dados relacionais baseados em linguagem SQL.

O resultado comercial, porém, foi pequeno. Atualmente vários princípios de orientação a objeto foram adotados pelos bancos de dados relacionais, gerando o que pode ser chamado de banco de dados relacional estendido ou modelo objeto-relacional.

2 – Álgebra Relacional

A álgebra relacional é um conjunto de operações usadas para manipular relações e é oriunda da *teoria dos conjuntos* da matemática.

Estas operações são utilizadas para selecionar tuplas de relações individuais e para combinar tuplas relacionadas de relações diferentes para especificar uma consulta em um determinado banco de dados.

A *álgebra relacional* é fechada, o que significa que os resultados de uma ou mais operações relacionais são sempre uma relação, a qual também pode ser manipulada pela álgebra relacional.

Todos os exemplos envolvendo álgebra relacional implicam na utilização do banco de dados descrito no item **1.6** no fim deste capítulo.

2.1. Seleção

A operação de seleção é utilizada para selecionar um subconjunto de tuplas de uma relação, sendo que estas tuplas devem satisfazer uma condição de seleção.

A forma geral de uma operação de seleção é:

$\sigma_{\text{<condição de seleção>}}$ (<nome da relação>)

A letra grega σ (sigma) é utilizada para representar a operação de seleção, <condição de seleção> é uma expressão booleana aplicada sobre os atributos da relação e <nome da relação> é o nome da relação sobre a qual será aplicada a operação de seleção.

Exemplos:

A operação consulta1 = $\sigma_{\text{salario < 2.500,00}}$ (EMPREGADO)
geraria a seguinte tabela como resultado:

Tabela *consulta1*					
Nome	RG	CPF	Depto.	RGSupervisor	Salario
Ricardo	303030	33333333	2	101010	2.300,00
Renato	505050	55555555	3	202020	1.300,00

A operação consulta2 = $\sigma_{\text{(relação = “Filho”) and (sexo=”Feminino”)}}$ (DEPENDENTES)
geraria a seguinte tabela como resultado:

Tabela *consulta2*				
RGResp	Dependente	DtNasc	Relação	Sexo
303030	Andréia	01/05/1990	Filho	Feminino

As operações relacionais que podem ser aplicadas na operação de seleção são: <, >, ≤, ≥, =, ≠. Além dos operadores booleanos *and*, *or* e *not*.

O operador de seleção é unário; isto é, ele é aplicado sobre uma única relação. Logo, a seleção não pode ser usada para selecionar tuplas oriundas de mais de uma relação.

A operação de seleção é aplicada sobre cada tupla individualmente.

O grau da relação resultante da operação de seleção é igual ao grau da relação sobre a qual se operou, já que ambas têm os mesmos atributos.

O número de tuplas da relação resultante é sempre menor ou igual ao número de tuplas da relação original.

A fração das tuplas selecionadas por uma condição de seleção é dita ser a seletividade da condição.

2.2. Projeção

A operação de projeção seleciona um conjunto determinado de colunas de uma relação.

O resultado da projeção é uma nova relação com os atributos selecionados.

A forma geral de uma operação de projeção é:

$\pi_{\text{<lista de atributos>}}$ (<nome da relação>)

A letra grega π (pi) representa a operação de projeção, <lista de atributos> representa a lista de atributos que o usuário deseja selecionar e <nome da relação> representa a relação sobre a qual a operação de projeção será aplicada.

Exemplos:

A operação consulta3 = $\pi_{\text{Dependente, DtNasc}}$ (DEPENDENTES)
geraria a seguinte tabela como resultado:

Tabela *consulta3*	
Dependente	DtNasc
Jorge	27/12/1986
Luiz	18/11/1979
Fernanda	14/02/1969
Ângelo	10/02/1995
Andréia	01/05/1990

As operações de projeção e seleção podem ser utilizadas de forma combinada, permitindo que apenas determinadas colunas de determinadas tuplas possam ser selecionadas.

A forma geral de uma operação sequencializada é:

$\pi_{<\text{lista de atributos}>}$ ($\sigma_{<\text{condição de seleção}>}$ (<nome da relação>))

Veja o seguinte exemplo:

A operação consulta4 = $\pi_{\text{nome,depto,salario}}$ ($\sigma_{\text{salario} < 2.500,00}$ (EMPREGADO)) geraria a seguinte tabela como resultado:

Tabela *consulta4*		
Nome	Depto	Salario
Ricardo	2	2.300,00
Renato	3	1.300,00

A consulta4 pode ser reescrita da seguinte forma:

consulta5 = $\sigma_{\text{salario} < 2.500,00}$(EMPREGADO)

Tabela *consulta5*					
Nome	RG	CPF	Depto	RGSupervisor	Salario
Ricardo	303030	33333333	2	101010	2.300,00
Renato	505050	55555555	3	202020	1.300,00

consulta6 = $\pi_{\text{nome,depto,salario}}$ (CONSULTA5)

Tabela *consulta6*		
Nome	Depto	Salario
Ricardo	2	2.300,00
Renato	3	1.300,00

Porém é mais elegante utilizar a forma descrita na consulta4.

Se uma lista de atributos não incluir nenhum atributo-chave, tuplas duplicadas poderiam aparecer na relação resultante. Logo, para que isso seja evitado, a operação de projeção remove implicitamente quaisquer duplicatas de uma tupla.

O número de tuplas existentes na relação resultante de uma operação de projeção é menor ou igual ao número de tuplas existentes na relação original.

Se a lista de atributos incluir um atributo-chave, a relação resultante terá o mesmo número de tuplas que a relação original.

A operação de projeção não é comutativa.

2.3. Operações Matemáticas

Levando em consideração que as relações podem ser tratadas como conjuntos, podemos então aplicar um conjunto de operações matemáticas sobre as mesmas.

Estas operações são:

- união (∪)
- intersecção (∩)
- diferença (–)

Este conjunto de operações não é unário, ou seja, podem ser aplicadas sobre mais de uma tabela, porém, existe a necessidade das tabelas possuirem tuplas exatamente do mesmo tipo.

Estas operações podem ser definidas da seguinte forma:

União - o resultado desta operação representada por R ∪ S é uma relação T que inclui todas as tuplas que se encontram em R e todas as tuplas que se encontram em S;

Intersecção - o resultado desta operação representada por R ∩ S é uma relação T que inclui as tuplas que se encontram em R e em S ao mesmo tempo;

Diferença - o resultado desta operação representada por R – S é uma relação T que inclui todas as tuplas que estão em R mas não estão em S.

Leve em consideração a seguinte consulta: Selecione todos os empregados que trabalham no departamento número 2 ou que supervisionam empregados que trabalham no departamento número 2.

Vamos primeiro selecionar todos os funcionários que trabalham no departamento número 2.

consulta7 = $\sigma_{depto = 2}$ (EMPREGADO)

Tabela *consulta7*					
Nome	RG	CPF	Depto	RGSupervisor	Salario
Fernando	202020	22222222	2	101010	2.500,00
Ricardo	303030	33333333	2	101010	2.300,00
Jorge	404040	44444444	2	202020	4.200,00

Vamos agora selecionar os supervisores dos empregados que trabalham no departamento número 2.

consulta8 = $\pi_{rgsupervisor}$ (CONSULTA7)

Tabela *consulta8*
RGSupervisor
101010
202020

Vamos projetar apenas o rg dos empregados selecionados:

consulta9 = π_{rg}(CONSULTA7)

Tabela *consulta9*
RG
202020
303030
404040

E por fim, vamos unir as duas tabelas, obtendo o resultado final.

consulta10 = consulta8 $\cup$ consulta9

Tabela *consulta10*
RG
202020
303030
404040
101010

Leve em consideração a próxima consulta: Selecione todos os empregados que desenvolvem algum projeto e que trabalham no departamento número 2.

Vamos primeiro selecionar todos os empregados que trabalham em um projeto.

consulta11 = $\pi_{rgempregado}$(EMPREGADO_PROJETO)

Tabela *consulta11*
RGEmpregado
202020
303030
404040
505050

Vamos agora selecionar todos os empregados que trabalham no departamento 2.

consulta12 = π_{rg} ($\sigma_{depto = 2}$ (EMPREGADO))

Tabela *consulta12*
RGEmpregado
202020
303030
404040

Obtemos então todos os empregados que trabalham no departamento 2 e que desenvolvem algum projeto.

consulta13 = consulta11 ∩ consulta12

Tabela *consulta13*
RGEmpregado
202020
303030
404040

Leve em consideração a seguinte consulta: Selecione todos os usuários que não desenvolvem projetos.

consulta14 = $\pi_{rgempregado}$ (EMPREGADO_PROJETO)

Tabela *consulta14*
RGEmpregado
202020
303030
404040
505050

consulta15 = π_{rg} (EMPREGADO)

Tabela *consulta15*
RGEmpregado
101010
202020
303030
404040
505050

consulta16 = CONSULTA15 – CONSULTA14

Tabela *consulta16*
RGEmpregado
101010

2.4. Produto Cartesiano

O produto cartesiano é uma operação binária que combina todas as tuplas de duas tabelas.

Diferente da operação união, o produto cartesiano não exige que as tuplas das tabelas possuam exatamente o mesmo tipo.

O produto cartesiano permite então a consulta entre tabelas relacionadas utilizando uma condição de seleção apropriada.

O resultado de um produto cartesiano é uma nova tabela formada pela combinação das tuplas das tabelas sobre as quais aplicou-se a operação.

O formato geral do produto cartesiano entre duas tabelas R e S é: R X S

Leve em consideração a seguinte consulta: Encontre todos os funcionários que desenvolvem projetos em Campinas.

consulta17 = EMPREGADO_PROJETO X PROJETO

Tabela *consulta17*				
RGEmpregado	NumProjeto	Nome	Num	Localizacao
202020	5	Financeiro 1	5	São Paulo
202020	5	Motor 3	10	Rio Claro
202020	5	Prédio Central	20	Campinas
202020	10	Financeiro 1	5	São Paulo
202020	10	Motor 3	10	Rio Claro
202020	10	Prédio Central	20	Campinas
303030	5	Financeiro 1	5	São Paulo
303030	5	Motor 3	10	Rio Claro
303030	5	Prédio Central	20	Campinas
404040	20	Financeiro 1	5	São Paulo
404040	20	Motor 3	10	Rio Claro
404040	20	Prédio Central	20	Campinas
505050	20	Financeiro 1	5	São Paulo
505050	20	Motor 3	10	Rio Claro
505050	20	Prédio Central	20	Campinas

Vamos agora selecionar as tuplas resultantes que estão devidamente relacionadas que são as que possuem o mesmo valor em número do projeto e número e cuja localização seja “Campinas”.

consulta18 = $\pi_{rgempregado,num}(\sigma_{(num.projeto=número).and.(localização='Campinas')}$ (CONSULTA17))

Tabela *consulta18*	
RGEmpregado	Num
404040	20
505050	20

2.5. Operação de Junção

A operação junção atua de forma similar á operação produto cartesiano, porém, a tabela resultante conterá apenas as combinações das tuplas que se relacionam de acordo com uma determinada condição de junção. A forma geral da operação junção entre duas tabelas R e S é a seguinte:

R $_{<condição\ de\ junção>}$ S

Leve em consideração a consulta a seguir: Encontre todos os funcionários que desenvolvem projetos em Campinas.

consulta19 = EMPREGADOS_PROJETOS $_{numproj\ =\ num}$ PROJETOS

Tabela *consulta19*				
RGEmpregado	NumProj	Nome	Num	Localizacao
202020	5	Financeiro 1	5	São Paulo
202020	10	Motor 3	10	Rio Claro
303030	5	Financeiro 1	5	São Paulo
404040	20	Prédio Central	20	Campinas
505050	20	Prédio Central	20	Campinas

consulta20 = $\sigma_{\text{localização = 'Campinas'}}$ (CONSULTA19)

Tabela *consulta20*				
RGEmpregado	NumProj	Nome	Num	Localizacao
404040	20	Prédio Central	20	Campinas
505050	20	Prédio Central	20	Campinas

2.6. Tabelas para referência neste capítulo

Tabela EMPREGADO					
Nome	RG	CPF	Depto	RGSupervisor	Salario
João Luiz	*101010*	*11111111*	*1*	***NULO***	*3.000,00*
Fernando	*202020*	*22222222*	*2*	*101010*	*2.500,00*
Ricardo	*303030*	*33333333*	*2*	*101010*	*2.300,00*
Jorge	*404040*	*44444444*	*2*	*202020*	*4.200,00*
Renato	*505050*	*55555555*	*3*	*202020*	*1.300,00*

Tabela DEPARTAMENTO		
Nome	N	RGGerente
Contabilidade	*1*	*101010*
Engenharia Civil	*2*	*303030*
Engenharia Mecânica	*3*	*202020*

Tabela PROJETO		
Nome	N	Localizacao
Financeiro 1	*5*	*São Paulo*
Motor 3	*10*	*Rio Claro*
Prédio Central	*20*	*Campinas*

Tabela DEPENDENTES				
RGResp	Dependente	DtNasc	Relacao	Sexo
101010	Jorge	27/12/86	Filho	Masculino
101010	Luiz	18/11/79	Filho	Masculino
202020	Fernanda	14/02/69	Conjuge	Feminino
202020	Angelo	10/02/95	Filho	Masculino
303030	Adreia	01/05/90	Filho	Feminino

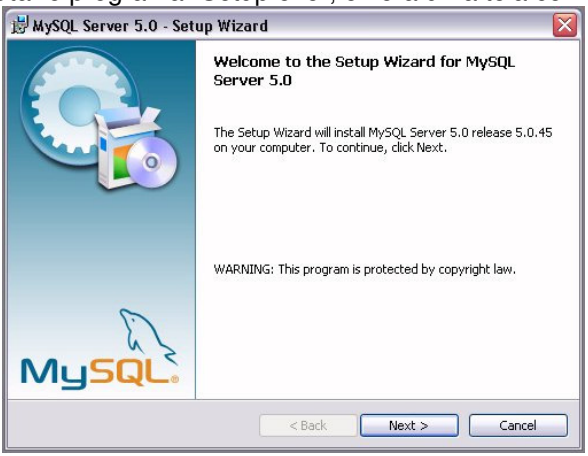
Tabela DEPARTAMENTO_PROJETO	
NumDepto	NumProj
2	*5*
3	*10*
2	*20*

Tabela EMPREGADO_PROJETO		
RGEmpregado	NumProj	Horas
202020	*5*	*10*
202020	*10*	*25*
303030	*5*	*35*
404040	*20*	*50*
505050	*20*	*35*

3 – Instalação do MySQL

Ao longo do curso usaremos o SGBD MySQL como ferramenta de estudo, no entanto como o objetivo do curso é o aprendizado da linguagem SQL, todos os comandos serão sempre prioritariamente exibidos com a forma padrão da linguagem.

3.1. Depois da obtenção do instalados do servidor do MySQL você poderá executar o programa “Setup.exe”, e verá uma tela como a seguinte:



Basta pressionar “Next >”.

3.2. Nesta tela você escolhe a forma de instalação. O parão “Typical” prevê todo o necessário para nossos objetivos. Mas pode-se optar pela opção “Complete”.

Depois, clique em “Next >”.

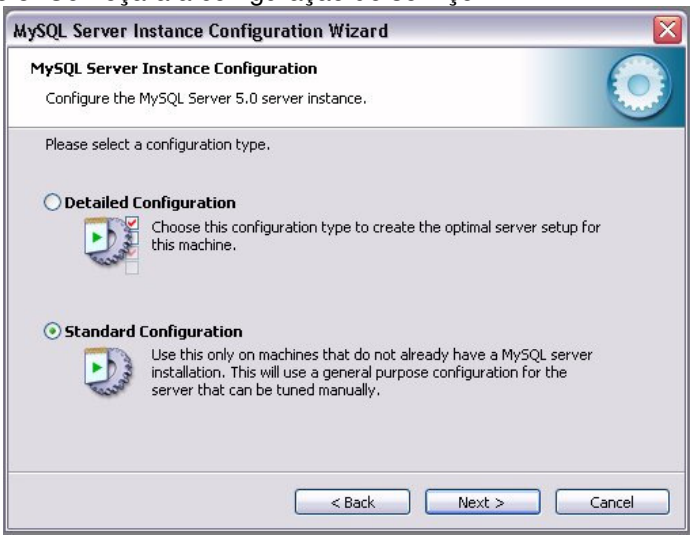
3.3. Deverá aparecer a seguinte tela de confirmação:

Agora clique em “Install”.

3.4. No fim da instalação pode-se optar por configurar o serviço do banco de dados, ele não estará funcionando enquanto não for configurado pela primeira vez.

Mantenha a opção marcada e clique em “Finish”.

3.5. Começará a confguração do serviço:



A opção “Standard Configuration” é mais simples e oferece tudo o necessário para nossos objetivos.

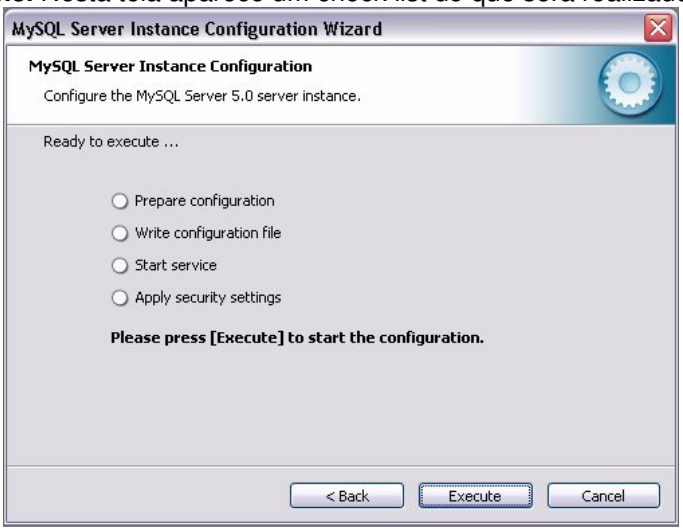
3.6. Nesta tela, podemos decidir se o servidor executará como serviço (primeira check box) e se o diretório do MySQL será incluído no path do Windows (segunda check box).

Marque as duas e clique em “Next >”.

3.7. Em seguida devemos definir a senha do usuário administrador (*root*) do SGBD. Cuidado: Mesmo senhas fracas como o próprio username *root* serão aceitas neste ponto. Certifique-se de marcar as demais check boxes que aparecem nesta janela (premitir acesso remoto do root e criar uma conta anônima).

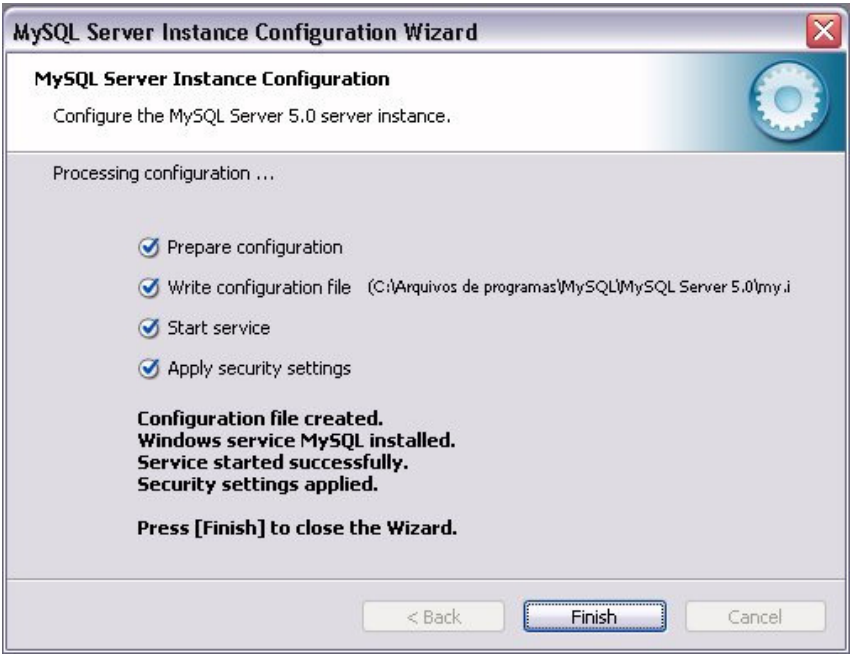
Clique novamente em “Next >”.

3.8. Nesta tela aparece um check list do que será realizado:



Agora clique em “Execute”.

3.9. Se não houver nenhum problema, todos os items serão checados e a instalação estará concluída.



Clique em “Finish” para terminar o processo de configuração.

Em acessório ao MySQL o aluno pode optar por utilizar o pacote de aplicativos conhecido como MySQL GUI Tools, distribuído gratuitamente no site oficial do SGBD.

A sua instalação é extremamente trivial e não requer nenhuma configuração. Inclusive, pode-se instalar o pacote sem ter-se instalado o servidor, basta configurar o endereço de acesso da máquina que funcionará como servidora do Banco de Dados, informando o endereço da rede, o número da porta de conexão, o nome do usuário a se conectar e a senha de conexão do mesmo.

4 – S Q L (Parte I)

Parte I – DDL

A DDL (Data Definition Language - Linguagem de Definição de Dados) que permite ao usuário definir esquemas e tabelas novos e elementos associados. A maioria dos bancos de dados de SQL comerciais tem extensões proprietárias no DDL, mas os comandos básicos sequem o padrão ANSI:

- CREATE - Cria um objeto dentro da base de dados.
- DROP - Apaga um objeto do banco de dados.

4.1. Conectando ao servidor

O MySQL já vem com um modo de interação com o usuário via prompt de comando que pode ser acessado através do comando a seguir caso o serviço já esteja habilitado:

- *mysql -h ENDEREÇO_HOST -u NOME_USUÁRIO –p*

Onde ENDEREÇO_HOST (precedido da opção -h) pode ser o nome da máquina host do serviço na rede, o número do IP da máquina ou a constante localhost se desejar conectar num servidor local. E NOME_USUÁRIO (precedido da opção -u) é o nome do usuário com o qual o usuário irá se conectar ao servidor. Por fim a opção –p especifica que será solicitada uma senha para o usuário ao logar.

Para desconectar do servidor pasta digitar QUIT.

Muitos usuários preferem utilizar clientes de interface gráfica (ou GUI – do inglês Graphic Users Interface) como MySQL Query Browser.

4.2. Gerenciamento de Esquemas

Esquemas nada mais são do que os banco de dados que serão gerenciados pelo SGBD. Aqui aprenderemos a: Exibir (show); criar (create); utilizar (use) e apagar (drop).

4.2.1. Exibir esquemas disponíveis:

SHOW DATABASES;

O comando mostra para o usuário todos os banco de dados ao qual o mesmo tem direito de acesso. Aqui vale a observação que todo comando SQL é terminado pelo caractere ‘;’ (ponto e vírgula).

4.2.2. Criando novos esquemas:

Se um usuário tiver a permissão de criar novos bancos de dados, ele deve utilizar o comando:

CREATE DATABASE nome_do_banco;

4.2.3. Utilizando um banco de dados:

Para passar a ter acesso aos dados de um banco de dados específico um usuário deve solicitar para utilizar o mesmo:

USE nome_do_banco;

Note que depois de criar um banco, o usuário não passa a utiliza-lo automaticamente, sendo necessário o comando use.

4.2.4. Apagando um banco de dados:

Um usuário deve ter cautela ao eliminar um banco, pois se o mesmo tiver privilágios para tal, eliminará toda a informação do mesmo com um único comando, tornando impossível a recuperação da informação sem um back-up dos dados:

DROP DATABASE nome_do_banco;

4.3. Gerenciamento de Tabelas

Assim como os bancos, as tabelas podem ser gerenciadas plenamente pelos usuários aos quais tem permissão para tal. Mas antes de começar a manipular a estrutura das mesmas vamos fazer um apêndice para saber os domínios e tipos de dados aceitos no SQL.

4.3.1. Domínios e tipos de dados:

Dados numéricos:

Tipo	Bytes	De	Até
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

Data e Hora:

Tipo de Coluna	Valor ``Zero''
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	00000000000000 (depende do tamanho do display)
TIME	'00:00:00'
YEAR	0000

Strings:

Tipo	Tam.maxímo	Bytes
TINYTEXT ou TINYBLOB	2^8-1	255
TEXT ou BLOB	2^16-1 (64K-1)	65535
MEDIUMTEXT ou MEDIUMBLOB	2^24-1 (16M-1)	16777215
LONGBLOB	2^32-1 (4G-1)	4294967295

Char e Varchar:

Valor	char(4)	**Armazen.**	varchar(4)	**Armazen.**
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Exigências de Armazenamento:

Tipo da coluna	**Tamanho exigido**
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 se X <= 24 ou 8 se 25 <= X <= 53
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D)	M + 2 bytes se D > 0, M + 1 bytes se D = 0 (D + 2, se M < D)
NUMERIC(M,D)	M + 2 bytes se D > 0, M + 1 bytes se D = 0 (D + 2, se M < D)
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte
CHAR(M)	M bytes, 1 <= M <= 255
VARCHAR(M)	L+1 bytes, onde L <= M e 1 <= M <= 255
TINYBLOB, TINYTEXT	L+1 bytes, onde L < 2^8
BLOB, TEXT	L+2 bytes, onde L < 2^16
MEDIUMBLOB	L+3 bytes, onde L < 2^24
MEDIUMTEXT	L+3 bytes, onde L < 2^24
LONGBLOB	L+4 bytes, onde L < 2^32
LONGTEXT	L+4 bytes, onde L < 2^32
ENUM('valor1','valor2' ,...)	1 ou 2 bytes, dependendo do número de valores enumerados (65535 valores no máximo)
SET('valor1','valor2', ...)	1, 2, 3, 4 or 8 bytes, dependendo do número de membros do conjunto (64 membros no máximo)

4.3.2. Criação de tabelas:

```
CREATE TABLE [IF NOT EXISTS] nome_da_tabela (
        nome_campo1 tipo_campo1 [*],
        nome_campo2 tipo_campo2 [*],
        nome_campo3 tipo_campo3 [*],...
        PRIMARY KEY (nome_campo1),
        FOREIGN KEY nome (campo) REFERENCES tabela (campo));
```

* Observação importante 1: Ao se criar uma chave estrangeira referenciando uma outra tabela, pode-se omitir o nome da constraint que representará a chave estrangeira, mas o MySQL adota o pradrão de nomear o índice da chave como mostra o padrão a seguir: nometabela_ibfk_*nºsequencial*.

* Observação importante 2: Eventualmente uma tabela pode ser referenciada por uma chave estrangeira em outra tabela, neste caso é necessário excluir primeiro a referência na tabela de origem.

4.3.2.1. Cláusulas extras:

- UNIQUE – Não aceita repetição
- NOT NULL – Não aceita valor vazio
- DEFAULT – Pode especificar um valor padrão
- UNSIGNED – Usado para números, ignora os negativos. Deve vir junto a definição do tipo numérico.
- AUTO_INCREMENT – Incremento automático

Por exemplo: vamos criar a tabela PROJETO do capítulo 2, que aparece descrita no item 2.6:

```
CREATE TABLE projeto (
        num INT UNIQUE NOT NULL,
        nome VARCHAR(50) NOT NULL,
        localizacao VARCHAR(50) NOT NULL,
        PRIMARY KEY num);
```

4.3.3. Visualização de tabelas:

Para visulizar todas as tabelas existentes em um banco de dados utilize o comando:

SHOW TABLES;

Mas se você desejar visualizar as informações a cerca da constituição de uma tabela utilize o comando:

SHOW COLUMNS FROM nome_da_tabela;

No MySQL (e em vários outros SGBDs) este comando possui "apelidos" mais curtos como por exemplo: "*DESCRIBE nome_da_tabela;"* Note que este último comando ("describe" ou *desc*) não é padrão de SQL, mas implementado, algumas vezes com sintaxes diferentes pelos SGBDs.

4.3.4. Alteração de tabelas:

```
ALTER TABLE nome_tabela
        ADD [COLUMN] nome_atributo_1 tipo_1 [{RIs}]
                [{, nome_atributo_n tipo_n [{RIs}]}]
        MODIFY [COLUMN] nome_atributo_1 tipo_1 [{RIs}]
                [{, nome_atributo_n tipo_n [{RIs}]}]
        CHANGE [COLUMN] nome_antigo_1 nome_novo_1 tipo_1 [{RIs}]
                [{, nome_antigo_n nome_novo_n tipo_n [{RIs}]}]
        DROP COLUMN nome_atributo_1
                [{, nome_atributo_n }]
        [ADD|DROP] [PRIMARY KEY ...|FOREIGN KEY ...]
```

Exemplos para alter table:

ALTER TABLE ambulatorios ADD nome VARCHAR(30)
ALTER TABLE medicos DROP PRIMARY KEY
ALTER TABLE pacientes DROP COLUMN doenca, DROP COLUMN cidade
ALTER TABLE funcionários ADD FOREIGN KEY(nroa) REFERENCES Ambulatórios
ALTER TABLE funcionarios CHANGE COLUMN cic cpf INT(11) NOT NULL;

Se a alteração desejada na verdade é a alteração do nome da tabela, deve-se utilizar o comando "*rename*" com a seguinte sintaxe:

RENAME TABLE nome_atual_da_tabela TO novo_nome;

4.3.5. Apagando uma table:

DROP TABLE nome_da_tabela;

Assim como drop tatabase esse comado apagará a tabela com todo seu conteúdo.

4.4. Exercícios:

1. Crie um banco de dados com nome Clinica.
2. Passe o foco de utilização para o banco de dados clinica, que acabou de ser usado.
3. Crie as seguintes tabelas no banco, considerando que os atributos sublinhados são chaves primárias e os em itálico são chaves estrangeiras:

- Ambulatorios: nroa (int), andar (numeric(3)) (não nulo), capacidade (smallint)
- Medicos: codm (int), nome (varchar(40)) (não nulo), idade (smallint) (não nulo), especialidade (char(20)), CPF (numeric(11)) (único), cidade (varchar(30)), *nroa* (int)
- Pacientes: codp (int), nome (varchar(40)) (não nulo), idade (smallint) (não nulo), cidade (char(30)), CPF (numeric(11)) (único), doenca (varchar(40)) (não nulo)
- Funcionarios: codf (int), nome (varchar(40)) (não nulo), idade (smallint), CPF (numeric(11)) (único), cidade (varchar(30)), salário (numeric(10)), cargo (varchar(20))
- Consultas: *codm* (int), *codp* (int), data e hora (datetime)

4. Crie a coluna nroa (int) na tabela Funcionários.
5. Remover as colunas cargo e nroa da tabela de Funcionários.
6. Apague a tabela Pacientes.
7. Apague todo o banco de dados Clinica.

4.5. Resposta dos Exercícios:

1. CREATE DATABASE clinica;

2. USE clinica;

3. CREATE TABLE ambulatorio (nroa INT NOT NULL UNIQUE, andar INT(3) NOT NULL, capacidade INT, PRIMARY KEY (nroa));

   CREATE TABLE medicos(codm INT NOT NULL UNIQUE, nome VARCHAR(40) NOT NULL, idade INT NOT NULL, especialidade VARCHAR(20), cpf INTEGER(11) UNIQUE, cidade VARCHAR(30), nroa INT, PRIMARY KEY (codm), FOREIGN KEY fk_nroa (nroa) REFERENCES ambulatorio (nroa));

   CREATE TABLE pacientes(codp INT NOT NULL UNIQUE, nome VARCHAR(40) NOT NULL, idade INT NOT NULL, cidade VARCHAR(30), cpf INTEGER(11) UNIQUE, doenca VARCHAR(40) NOT NULL, PRIMARY KEY (codp));

   CREATE TABLE funcionarios(codf INT NOT NULL UNIQUE, nome VARCHAR(40) NOT NULL, idade INT, cpf INTEGER(11) UNIQUE, cidade VARCHAR(30), salario FLOAT(10), cargo VARCHAR(20), PRIMARY KEY (codf));

   CREATE TABLE consultas(codm INT NOT NULL, codp INT NOT NULL, datahora DATETIME NOT NULL, PRIMARY KEY (codm, datahora), FOREIGN KEY fk_codm (codm) REFERENCES medicos (codm), FOREIGN KEY fk_codp (codp) REFERENCES pacientes (codp));

4. ALTER TABLE funcionarios ADD nroa INT NOT NULL;
   ALTER TABLE funcionarios ADD FOREIGN KEY fk_nroa (nroa) REFERENCES ambulatorio (nroa);

5. ALTER TABLE funcionarios DROP cargo;
   ALTER TABLE funcionarios DROP FOREIGN KEY fk_nroa;
   ALTER TABLE funcionarios DROP nroa;

6. ALTER TABLE consultas DROP FOREIGN KEY fk_codp;
   DROP TABLE pacientes;

7. DROP DATABASE clinica;

5 – S Q L (Parte II)

Parte II – DML

A DML (Data Manipulation Language - Linguagem de Manipulação de Dados) que permite ao usuário inserir, alterar, excluir e recuperar dados em tabelas já existentes. A maioria dos bancos de dados de SQL comerciais tem extensões proprietárias no DML, mas os comandos básicos sequem o padrão ANSI:

- INSERT - Insere registros em uma tabela.
- UPDATE - Altera registros de uma tablea.
- DELETE - Apaga registros de uma tabela.
- SELECT - Recupera registros em uma tabela.

5.1. Inserindo dados em tabelas

Para inserir dados em uma tabela é necessário se referenciar o nome da tabela na qual a inserção será realizada, seguida do nome das colunas a serem afetadas pela inserção, seguido dos seus respectivos valores, em ordem. Note que nem todas as colunas precisam ser preenchidas para que se inclua um registro em uma tabela, apenas aquelas em que a restrição de preenchimento é marcada como NOT NULL (ou seja, não pode ser nula). A sintaxe do comando *INSERT* aparece a seguir:

INSERT INTO nome_tabela [(nome_coluna1,...)]
VALUES (valor_coluna1,...);

Existem alguns SGBDs nos quais não é necessário se informar a ordem das colunas se o usuário desejar fazer uma inserção em todas as colunas da tabela com os valores na seqüência em que as colunas foram definidas. Assim:

INSERT INTO nome_tabela VALUES (valor_coluna1,...);

Vejamos como exemplo a tabela a **empregado** do item 2.6 :

Nome	**RG**	**CPF**	**Depto**	**RGSupervisor**	**Salario**
João Luiz	*101010*	*11111111*	*1*	***NULO***	*3.000,00*
Fernando	*202020*	*22222222*	*2*	*101010*	*2.500,00*
...	*...*	*...*	*...*	*...*	*...*

O comando para criar esta tabela seria:

```
CREATE TABLE empregado (
        rg INTEGER(6) NOT NULL UNIQUE,
        nome VARCHAR(50) NOT NULL,
        cpf INTEGER(8) NOT NULL UNIQUE,
        depto INTEGER(2) NOT NULL,
        rgsupervisor INTEGER(6),
        salario FLOAT NOT NULL,
        PRIMARY KEY (rg),
        FOREIGN KEY (depto) REFERENCES departamento (n),
        FOREIGN KEY (rgsupervisor) REFERENCES empregado (rg)  );
```

Agora que a tabela foi criada (note que imagina-se que a tabela departamento já exista, pois faz-se refeência de chave estrangeira para ela), pode-se inserir dados nesta tabela.

Vamos inserir os registros dos funcionários João Luiz e Fernando:

```
INSERT INTO empregado (rg, nome, cpf, depto, rgsupervisor, salario)
    VALUES (101010, 'João Luiz', 11111111, 1, NULL, 3000);

INSERT INTO empregado VALUES (202020, 'Fernando', 22222222, 2, 101010, 2500);
```

Observe que as duas inserções foram feitas de maneira diferente e ambas estão corretas.

Quando se informa valores numéricos não é necessário nenhuma marcação especial, como acontece com as seqüência de caracteres, que por definição são informadas entre aspas simples (‘’ – ou *piclas*). No caso de se desejar deixar uma coluna vazia, pode-se informar a constante NULL, ou não passar o valor para a coluna especificada. Porém se a mesma tiver definido um valor default, este será atribuído a mesma.

5.2. Alterando dados de tabelas

Algumas vezes faz-se necessário alterar alguns dados já existente em uma tabela. Fazer uma alteração, uma atualização. O comando com esta responsabilidade é o *UPDATE*. A seguir a sintaxe do comando:

UPDATE nome_tabela SET nome_coluna1=valor_coluna1 [, nome_coluna2=valor_coluna2 ...] WHERE (condição_de_restrição);

Vamos alterar o salário do funcionário Fernando de R$ 2.500,00 para R$ 2.800,00:

```
UPDATE empregado SET salario=2800 WHERE (rg=202020);
```

A condição de restrição deve ser rígida suficiente para que a alteração que desejamos realizar se aplique apenas nos registros esérados. Dessa maneira no exemplo acima optou-se por marcar o registro cujo campo rg tem valor igual a 202020, pois como o rg é chave primária da tabela, não tem repetição, e garante que apenas o registro onde ocorre o casamento será alterado.

5.3. Excluindo dados de tabelas

A operação de exclusão também é importante, e o comando que realiza esta tarefa em SQL é o *DELETE*. A seguir a sintaxe do comando:

DELETE FROM nome_tabela WHERE (condição_de_restrição);

Vamos excluir o Fernando do quadro de funcionários da empresa:

```
DELETE FROM empregado WHERE (rg=202020) ;
```

A *condição de restrição* tem o mesmo objetivo e importância da usada no comando UPDATE.

5.4. Recuperando dados de tabelas

A operação de seleção (*SELECT*) é a mais largamente utilizada quando se trata de banco de dados relacionais. É através do coamando *select* que se pode obter os dados que estão armazenados em uma ou mais tabelas. Uma consulta (*query*) em SQL consiste de 6 cláusulas, onde as duas primeiras são obrigatórias. As cláusulas entre {...} são opcionais:

```
SELECT   [ALL / DISTINCT]   < lista-atributos / * >
        FROM  tabela1 { , tabela2 ... }
        { WHERE < condição >   }
        { GROUP BY atributo1 { , atributo2 ... }  }
        { HAVING  < condição >   }
        { ORDER BY atributo1 { , atributo2 ... }  } ;
```

A seleção de campos pode ser efetuada pelos nomes dos campos ou por um asterisco (*), que seleciona todos os campos. Por exemplo uma listagem de todos os dados de todos os empregados:

```
SELECT * FROM EMPREGADO;
```

Ou, listar apenas o nome e matricula de todos os funcionários:

```
SELECT nome, matricula FROM EMPREGADO;
```

A condição do *WHERE* indica as linhas da tabela a serem selecionadas mediante o casamento de uma condição. Como por exemplo, listar o nome e matrícula dos funcionários do departamento 20:

```
SELECT nome, matricula FROM empregado WHERE depto = 20;
```

Ou, listar o nome e matricula dos funcionários que são mulheres:

```
SELECT nome, matricula FROM empregado WHERE sexo = 'F' ;
```

E se fosse necessário listar o nome e matrícula das mulheres do departamento 20?

```
SELECT nome, matricula FROM EMPREGADO
                WHERE depto = 20 AND sexo = 'F' ;
```

Pode-se utilizar as conições lógicas *AND* (equivalente ao *E* lógico) e *OR* (equivalente ao *OU* lógico).

Quando o casamento a ser feito for com um campo textual a recuperação pode ser feita por aproximação usando LIKE. Onde:

- _ uma ocorrência de qualquer carater
- % nenhuma ou várias ocorrências de quaisquer caracteres

Veja a listagem dos nomes de funcionários que começam por 'J':

```
SELECT nome FROM EMPREGADO WHERE nome LIKE 'J%' ;
```

E a listagem dos nomes de funcionários que tenham 'Silva' como parte do nome:

```
SELECT nome FROM EMPREGADO
                WHERE nome LIKE ' %Silva%' ;
```

As palavras-chave *distinct* e *all* especificam o que fazer com linhas duplicadas no resultado da seleção. As linhas duplicada são eliminadas pela palavra *DISTINCT*; e *ALL* é o default e retorna todas as linhas selecionas (inclusive a maioria absoluta das queries dispensa o uso do *all*).

Listar os salários dos empregados:

```
SELECT salario FROM EMPREGADO ;
                ou
SELECT ALL salario FROM EMPREGADO ;
```

E se fosse pra listar os diferentes salários pagos aos empregados:

```
SELECT DISTINCT salario FROM EMPREGADO ;
```

A clausula *ORDER BY* faz a ordenação de linhas a partir dos valores de colunas em ordem ascendente (*ASC*), que é seu default (e pode ser omitido sem diferenciação no resultado), ou em ordem descendente (*DESC*). No exemplo a seguir listamos o nome e salário dos funcionários ordenados pelo nome do funcionário.

```
SELECT nome, salario FROM EMPREGADO ORDER BY nome ;
```

Liste o nome e salário dos funcionários ordenados do maior ao menor salário e pelo nome do funcionário (para desempate):

```
SELECT nome, salario FROM EMPREGADO
                ORDER BY salario DESC, nome ;
```

Os operadores aritméticos + , - , * e / podem ser usados em colunas de tipos numéricos. Note que os valores da tabela original NÃO são alterados. Vejamos, a listagem dos diferentes salários pagos e o novo salário com aumento de 10%:

```
SELECT salario, 1.1 * salario FROM EMPREGADO ;
```

Liste o nome dos funcionários e o aumento a receber:

```
SELECT nome, (1.1 * salario - salario) FROM EMPREGADO ;
```

Para facilitar o entendimento do resultado de uma pesquisa, podemos definir um novo nome para uma coluna ou incluir um texto no resultado. Vamos refazer a listagem dos diferentes salários pagos e o novo salário com aumento de 10%:

```
SELECT salario AS 'Salário atual', (1.1 * salario) AS 'Novo sálario'
                FROM EMPREGADO ;
```

5.4.1. Funções de agregação: Servem para sumarizar valores em um conjunto de linhas. São elas *SUM* (soma), *MAX* (máximo), *MIN* (mínimo), *AVG* (média), COUNT (quantidade).

Liste o total de empregados, a soma de salários, o maior salário, o menor salário e a média salarial da empresa:

```
SELECT COUNT(*), SUM(salario), MAX(salario),
        MIN(salario), AVG (salario) FROM EMPREGADO ;
```

Liste a média salarial dos funcionários do departamento 20:

```
SELECT  AVG(salario) AS 'Média salarial do departamento 20'
                FROM EMPREGADO WHERE depto = 20 ;
```

Liste o número de departamentos da empresa:

```
SELECT COUNT(*)AS 'Quantidade de departamentos`
                FROM DEPARTAMENTO ;
```

A cláusula *distinct* pode ser usada nas funções count, sum e avg. Veja o exemplo para a listagem do total de diferentes valores de salários:

```
SELECT COUNT (DISTINCT salario)
        AS 'Quantidade de diferentes salários pagos'
                FROM EMPREGADO ;
```

Note bem que estas funções retornam um valor numérico, e não um conjunto (set) de valores. Assim podem ser usadas para comparações lógicas com queries aninhadas. Como por exemplo a listagem do nome dos funcionários que tem dois ou mais dependentes:

```
SELECT nome FROM empregado E
        WHERE (SELECT COUNT(*) FROM dependente D
                WHERE E.matricula = D.matricula) >= 2 ;
```

A cláusula *Group By* pode aplicar funções em grupos de linhas com os mesmos valores de atributos. Como por exemplo listar o número de empregados por departamento:

```
SELECT depto, COUNT(*) FROM empregado GROUP BY depto;
```

Ou a listadem da quantidade de homens e mulheres da empresa:

```
SELECT sexo, COUNT(*) FROM empregado GROUP BY sexo ;
```

A cláusula *Having* seleciona as linhas a partir de condições aplicadas sobre os valores de funções. Como na listagem dos departamentos que possuem mais de 2 funcionários:

```
SELECT depot FROM empregado
        GROUP BY depto HAVING COUNT(*) > 2 ;
```

Note que a cláusula *having* é bem diferente da *where*. O *where* seleciona linhas antes da agregação e o *having* só seleciona as linhas depois da agregação, ou seja o *having* está sempre condivionado a um *group by*. Vejamos a diferença nestas listagens:

Liste, por departamento, o nº de func. que ganham + de 300,00:

```
SELECT depto, COUNT(*) FROM empregado
                WHERE salario > 300  GROUP BY depto ;
```

E liste, para os departamentos que tenham média salarial > 300,00 o total de funcionários e a média salarial:

```
SELECT depto, COUNT(*), AVG(salario) FROM empregado
                GROUP BY depto HAVING AVG(salario) > 300 ;
```

E quando for necessário casar com um campo em que não exista um valor naquele registro? Para isso usamos a palavra *NULL*. Um campo com *NULL* não participa nas funções. Por exemplo:

- média (2, 4, 0) = 2 pois: (6 / 3)
- média (2, 4, NULL) = 3 pois: (6 / 2)

Veja um exemplo de listagem; a do nome dos funcionários que não tem um supervisor:

```
SELECT nome FROM empregado WHERE supervisor IS NULL ;
```

5.4.2. Produto cartesiano de tabelas: No produto cartesiano, temos mais de uma tabela declarada na cláusula 'FROM', e com isso a combinação de cada linha de uma tabela com cada uma das linhas da outra tabela. Veja como fica quando juntam-se as linhas da tabela EMPREGADO com as linhas da tabela DEPARTAMENTO:

```
SELECT * FROM empregado, departamento ;
```

Ou o resultado de juntar as linhas da tabela EMPREGADO com as linhas da tabela DEPENDENTE:

```
SELECT * FROM empregado, dependente ;
```

A junção (ou *JOINT*) de tabelas é executada ao se declarar uma condição na cláusula 'WHERE' em um produto cartesiano. Ela em geral indica o relacionamento entre as tabelas, e ajuda a manter apenas as linhas que tem algum significado, reduzindo substancialmente o número de linhas da junção. Veja a listagem do nome dos empregados e do departamento em que trabalham:

```
SELECT E.nome, D.nome FROM empregado E, departamento D
    WHERE E.depto = D.deptonum ;
```

Podemos criar apelidos para as tabelas utilizadas, isso ajuda no entendimento dos casamentos e acaba com as possíveis ambigüidades de nomes de colunas de diferentes tabelas. Veja na listagem a seguir o nome dos empregados e de seus dependentes:

```
SELECT E.nome, D.nome FROM empregado E, dependente D
    WHERE E.matricula = D.matricula ;
```

O *outer join* é uma cláusula de junção de tabelas especial. Normalmente uma junção mostra apenas as linhas que satisfazem a condição especificada. Ocasionalmente, podemos querer ver as linhas de uma tabela que não satisfaçam a condição. O *OUTER JOIN* mostra, além das linhas que satisfazem a condição, as linhas de uma das tabelas que não a satisfazem. A implementação, dependendo do fabricante, pode ser feita na cláusula *FROM* (como no MySQL) ou na cláusula *WHERE*.

- LEFT JOIN: inclui as linhas da 1ª tabela que não satisfazem a condição;
- RIGHT JOIN: inclui as linhas da 2ª tabela que não satisfazem a condição.

Por exemplo, liste o nome de todos os empregados, e de seus dependentes, se houver (ou seja, mesmo que um empregado não tenha dependente deve aparecer na listagem):

```
SELECT E.nome, D.nome
        FROM empregado E LEFT JOIN dependente D
                ON E.matricula = d.Matricula ;
```

Esta é a sintaxe aceita pelo MySQL, mas existe outra aceita por outros SGBDs, que seria :

```
SELECT E.nome, D.nome FROM empregado E, dependente D
        WHERE E.Matricula *= D.Matricula ;
```

5.4.3. Queries aninhadas: Acontece quando temos a cláusula *WHERE* usando outra query. A query interna retorna como resposta uma tabela ou conjunto (*set*), que pode, inclusive, ser vazio. Utilizam diferentes operadores, como *IN*, *EXISTS*, *ANY*, *ALL* e *UNION*.

O operador *IN* compara um valor *v* com um conjunto de valores *C* e retorna *TRUE* se *v* é um elemento de *C*. Pode ser invertido com a palavra reservada *NOT*.

E o operador *EXISTS* retorna *TRUE* quando o valor de uma query aninhada é vazio. Também pode ser invertido com o uso do *NOT*.

Vejamos exemplos de uso do *IN*. Como na listagem do nome e endereço dos empregados do departamento 'Compras':

```
SELECT nome, endereco FROM empregado
        WHERE depto IN (SELECT deptonum
                FROM departamento WHERE nome = 'Compras');
```

Ou na listagem do nome dos funcionários que não tem dependentes:

```
SELECT nome FROM empregado E
        WHERE E.matricula NOT IN (SELECT D.matricula FROM
                dependente D WHERE E.matricula = D.matricula);
```

Podemos usar um conjunto explícito de valores para formar um set de comparação na cláusula *WHERE* para o *IN*. Como na listagem do nome e a matricula dos funcionários que trabalham nos projetos 1, 2 ou 3:

```
SELECT nome, matricula FROM empregado E, trabalhando T
        WHERE E.matricula = T.matricula AND T.codigo IN (1,2,3);
```

Agora o uso do *EXISTS*. Vamos listar o nome e endereço dos empregados do departamento 'Compras'. O resultado será o mesmo que o obtido com o *IN*, mas com sintaxe bem diferente:

```
SELECT nome, endereco FROM empregado E
        WHERE EXISTS (SELECT * FROM departamento D
                WHERE D.nome = 'Compras'
                AND E.depto = D.deptonum) ;
```

E, a listagem do nome dos funcionários que não tem dependentes:

```
SELECT nome FROM empregado E WHERE NOT EXISTS
        (SELECT * FROM dependente D
                WHERE E.matricula = D.matricula) ;
```

O operador *v > ALL* compara um valor *v* com um conjunto de valores *C* e retorna *TRUE* se *v* for maior que *TODOS* os elementos de *C*. E o operador *v > ANY* compara um valor *v* com um conjunto de valores *C* e retorna *TRUE* se *v* for maior que *ALGUM* dos elementos de *C*.

Estes operadores podem ser usades em conjunto com os comparadores lógicos:

> (maior)	*< (menor)*
>= (maior ou igual)	*<= (menor ou igual)*
= (igual)	<> ou != *(diferente)*

O uso do *ALL* pode ser visto na listagem do nome dos empregados que ganham os maiores salários da empresa.

```
SELECT nome FROM empregado WHERE
        salario >= ALL (SELECT salario FROM empregado);
```

E o uso do ANY na listagem do nome dos empregados que ganham mais que o menor salário da empresa.

```
SELECT nome FROM empregado WHERE
        salario > ANY (SELECT salario FROM empregado);
```

A união de tabelas é um recurso no qual podemos unir o resultado de várias consultas, desde que o TIPO dos resultados sejam os mesmos.

Como para listar a localização dos departamentos juntamente com a localização dos projetos:

```
(SELECT DISTINCT localizacao FROM departamento)
        UNION
(SELECT DISTINCT localizacao FROM projeto);
```

6 – Exercícios para o Capítulo 5

6.1. Definição do banco de dados para o exercício

```
CREATE DATABASE empresa;

USE empresa;

CREATE TABLE empregado (
  MATRICULA int(5) NOT NULL auto_increment,
  NOME varchar(200) NOT NULL,
  DATANASC date default NULL,
  ENDERECO varchar(250) default NULL,
  SEXO char(1) NOT NULL,
  SALARIO decimal(10,2) NOT NULL,
  SUPERVISOR int(5) default NULL,
  DEPTO int(5) default NULL,
  PRIMARY KEY  (MATRICULA),
  FOREIGN KEY (SUPERVISOR) REFERENCES empregado (MATRICULA)
);

INSERT INTO empregado (MATRICULA, NOME, DATANASC, ENDERECO,
SEXO, SALARIO, SUPERVISOR, DEPTO) VALUES
 (1,'Maria','1975-12-22','Rua A, 9','F','2000.00',NULL,1),
 (2,'Marcus','1970-11-12','Rua B, 2','M','1500.00',1,1),
 (3,'Abel','1960-11-13','Rua Z, 6','M','4000.00',1,1),
 (4,'Carlos','1966-03-05','Rua J, 8','M','3000.00',2,3),
 (5,'Édson','1970-04-16','Rua M, 9','M','1000.00',3,2),
 (6,'Flávio','1973-09-19','Rua R, 3','M','4000.00',3,2),
 (7,'Gilda','1974-10-20','Rua S, 4','F','1000.00',3,2),
 (8,'Hilton','1965-11-01','Rua Z, 7','M','2000.00',2,3),
 (9,'Irene','1950-02-05','Rua B, 3','M','3000.00',2,3),
 (10,'José','1957-02-07','Rua M, 10','M','4000.00',3,3),
 (11,'Kátia','1970-06-12','Rua W, 19','F','1000.00',2,2),
 (12,'Noel','1968-07-13','Rua Y, 78','M','2000.00',3,3),
 (13,'Otto','1975-03-15','Rua B, 10','M','2000.00',3,3),
 (14,'Ana','1973-12-07','Rua D, 12','F','2000.00',3,1);

CREATE TABLE dependente (
  matricula int(5) NOT NULL,
  nome varchar(200) NOT NULL,
  sexo char(1) default NULL,
  datanasc date default NULL,
  parentesco varchar(50) default NULL,
  PRIMARY KEY  (`matricula`,`nome`),
  FOREIGN KEY (matricula) REFERENCES empregado(MATRICULA)
);

INSERT INTO dependente (matricula, nome, sexo, datanasc, parentesco) VALUES
 (1,'Lucas','M','1998-10-22','Filho'),
 (2,'Clara','F','1995-11-02','Filha'),
```

```
 (2,'Patrick','M','1980-08-29','Filho'),
 (3,'Alan','M','1987-07-15','Filho');

CREATE TABLE departamento (
  deptonum int(5) NOT NULL auto_increment,
  nome varchar(200) default NULL,
  matgerente int(5) default NULL,
  PRIMARY KEY  (deptonum),
  FOREIGN KEY (matgerente) REFERENCES empregado (MATRICULA)
);

INSERT INTO departamento (deptonum, nome, matgerente) VALUES
 (1,'Direção Geral',1),
 (2,'Produção',2),
 (3,'Desenvolvimento',3);

CREATE TABLE deptolocal (
  depto int(5) NOT NULL,
  localizacao varchar(50) NOT NULL,
  PRIMARY KEY  (depto),
  FOREIGN KEY (depto) REFERENCES departamento(deptonum)
);

INSERT INTO deptolocal (depto,localizacao) VALUES
 (1,'Meier'),
 (2,'Del Castilho'),
 (3,'Centro');

CREATE TABLE projeto (
  CODIGO int(5) NOT NULL auto_increment,
  NOME varchar(200) default NULL,
  LOCALIZACAO varchar(200) default NULL,
  DEPTO int(5) default NULL,
  PRIMARY KEY  (CODIGO),
  FOREIGN KEY (DEPTO) REFERENCES departamento(deptonum)
);

INSERT INTO projeto (CODIGO, NOME, LOCALIZACAO, DEPTO) VALUES
 (1,'Tamar','Itaúna,  ES',2),    (2,'Jubarte','Abrolhos,  BA',2),    (3,'Pantanal','Bonito,
MS',3);

CREATE TABLE trabalhando (
  MATRICULA int(5) NOT NULL,
  CODIGO int(5) NOT NULL,
  HORAS int(5) default NULL,
  PRIMARY KEY  (MATRICULA,CODIGO),
  FOREIGN KEY (MATRICULA) REFERENCES empregado(MATRICULA),
  FOREIGN KEY (CODIGO) REFERENCES projeto(CODIGO)
);

INSERT INTO trabalhando (MATRICULA, CODIGO, HORAS) VALUES
 (4,3,40),  (5,1,40),  (6,1,40),  (7,2,40),  (8,3,40),  (9,1,20),  (9,3,20),  (10,1,20),
(10,3,20),  (11,2,40),  (12,2,20),  (12,3,20),  (13,2,20),  (13,3,20),  (14,3,40);
```

6.2. Exercícios

6.2.1. Parte 1

1 - Liste o nome e data de nascimento dos funcionários ordenados por data de nascimento.
2 - Liste o nome e salário dos funcionários ordenados por nome.
3 - Liste o nome e parentesco dos dependentes.
4 - Liste o nome dos projetos e a sua localização, ordenados por localização.
5 - Liste a matrícula, nome e salário dos empregados que ganham entre R$ 200,00 e R$500,00.
6 - Liste o nome dos dependentes do sexo masculino.
7 - Liste o nome dos projetos localizados no Méier, em ordem decrescente.
8 - Liste o nome do departamento de numero 10.
9 - Liste o nome das mulheres que trabalham no departamento 10.
10 - Liste o número do departamento de Produção.
11 - Liste o numero e nome dos departamentos começados pela letra ´C´.
12 - Liste a matrícula e nome dos funcionários que tenham 'Silva' como parte do nome.
13 - Liste a matrícula e nome dos funcionários que tenham 'Silva' como último sobrenome.
14 - Liste o numero dos departamentos que tenham algum projeto.
15 - Liste o numero dos departamentos que tenham algum projeto, sem duplicação de linhas.
16 - Liste os parentescos que existem na tabela DEPENDENTE.
17 - Liste os diferentes parentescos da tabela DEPENDENTE.
18 - Liste as localizações dos departamentos.
19 - Liste as diferentes localizações dos departamentos.
20 - Liste as localizações dos projetos.
21 - Liste as diferentes localizações dos projetos.

6.2.2. Parte 2

22 - Liste o nome dos departamentos que não tem gerente.
23 - Liste o nome e matrícula dos funcionários que não tem supervisor.
24 - Liste a soma dos salários, o maior salário, o menor salário e a média salarial dos empregados do departamento 10.
25 - Liste o total de funcionários.
26 - Liste o total de funcionários do departamento 10.
27 - Liste o total de funcionários por número de departamento.
28 - Liste o número identificador dos departamentos que tenham dois ou mais funcionários.
29 - Liste o total de projetos por localização.
30 - As localizações que possuem mais de um projeto.
31 - Liste o total de departamentos por localização.
32 - As localizações que possuem mais de um departamento.
33 - Liste o total de diferentes parentescos da tabela DEPENDENTES.

34 - Liste o total de dependentes por tipo de parentesco.
35 - Liste o total de dependentes por sexo.
36 - Liste o total de dependentes do funcionário de matrícula 1001.
37 - A matrícula dos funcionários que possuam mais de dois dependentes.
38 - Liste a matricula dos supervisores e o número de funcionários que supervisionam.
39 - Liste a matricula dos supervisores que supervisionam mais de 2 funcionários.
40 - Liste a matrícula dos gerentes e o número de departamentos que gerenciam.
41 - Liste a matricula dos gerentes que gerenciam mais de um departamento.

6.2.3. Parte 3

42 - Liste o nome das mulheres que trabalham no departamento 'Desenvolvimento'.
43 - Liste o nome do departamento e o nome das mulheres que trabalham no departamento 10, ordenado por departamento.
44 - Liste os nomes dos departamentos e de seus gerentes em ordem crescente.
45 - Liste o nome do gerente do departamento 'Produção'.
46 - Liste o nome e matricula dos funcionários homônimos.
47 - Liste o nome e matricula dos empregados que tenham um dependente com o mesmo nome que seu.
48 - Liste o nome, matricula e data de nascimento dos funcionários que nasceram no mesmo dia.
49 - Liste o nome do departamento e o nome dos funcionários subordinados ao gerente 'Mario'.
50 - Liste o nome dos funcionários que trabalharam mais de 20 horas no projeto 'Tamar'.
51 - Liste o nome, endereço e data de nascimento dos gerentes dos departamentos localizados no 'Méier'.
52 - Liste todos os dados de projetos do departamento 'Produção'.
53 - Liste o nome, matricula e departamento dos funcionários que trabalham no projeto 'Jubarte' ou no projeto 'Tamar'.
54 - Liste o nome, matricula e departamento dos funcionários que trabalham no projeto 'Jubarte' ou no projeto 'Tamar', sem duplicação de linhas.
55 - Liste a soma do salário, o maior salário, o menor salário e a média salarial dos empregados do departamento 'Produção'.
56 - Liste o total de funcionários por nome de departamento.
57 - Liste o total de funcionários do departamento 'Produção'.
58 - Liste o nome dos departamentos com mais de um funcionário
59 - Liste o nome e matricula dos funcionários com mais de dois dependentes.

60 - Liste o total de dependentes do funcionário ‘Marcus’.
61 - Liste o nome dos funcionários que trabalham em dois ou mais projetos.
62 - Liste o nome dos funcionários que gerenciam mais de um departamento.
63 - Liste a matricula e nome dos supervisores.
64 - Liste a matricula e o nome dos supervisores que supervisionam mais de 2 funcionários.
65 - Liste, para cada projeto, seu código, nome, número de funcionários que nele trabalham e a média de horas.

6.2.4. Parte 4

66 - Liste o nome das mulheres que trabalham no departamento ’Produção’.
67 - Liste o nome do gerente do departamento ‘Produção’.
68 - Liste a soma do salário, o maior salário, o menor salário e a média salarial dos empregados do departamento ‘Produção’.
69 - Liste o total de funcionários do departamento ‘Produção’.
70 - Liste o nome dos departamentos com mais de um funcionário.
71 - Liste a matricula e nome dos supervisores e o número de funcionários que supervisionam.
72 - Liste a matricula e o nome dos supervisores que supervisionam mais de 2 funcionários.
73 - Liste o nome do departamento e o nome dos funcionários subordinados ao gerente ‘Mario Souza’.
74 - Liste o nome dos funcionários que gerenciam mais de um departamento.
75 - Liste o nome, endereço e data de nascimento dos gerentes dos departamentos localizados no ‘Meier’.
76 - Liste o nome e matricula dos funcionários com mais de dois dependentes.
77 - Liste o total de dependentes do funcionário ‘Marcus’.
78 - Liste o nome dos funcionários que tem algum dependente.
79 - Liste o nome dos gerentes que tem algum dependente.
80 - Liste o nome e matricula dos empregados que tenham um dependente com o mesmo nome que o seu.
81 - Liste o nome dos funcionários que trabalham em dois ou mais projetos.
82 - Liste o nome dos funcionários que trabalharam mais de 20 horas no projeto ‘FornecNorte’.
83 - Liste o nome, matricula e departamento dos funcionários que trabalham no projeto ‘FornecNorte’ ou no projeto ‘FornecSul’.
84 - Liste o nome, matricula e departamento dos funcionários que trabalham no projeto ‘FornecNorte’ e no projeto ‘FornecSul’.

6.3. Respostas dos exercícios

6.3.1. Parte 1

1 - select nome,datanasc from empregado order by datanasc ;
2 - select nome,salario from empregado order by nome;
3 - select nome, parentesco from dependente;
4 - select nome, localizacao from projeto where order by localizacao;
5 - select matricula,nome,salario from empregado where salario >= 200 and salario <= 500;
6 - select nome from dependente where sexo='m';
7 - select nome from projeto where localizacao = "Méier" order by nome desc;
8 - select nome from departamento where deptonum=10;
9 - select nome from empregado where sexo='F' and depto=10;
Ou: select nome from empregado where depto=10 and sexo='F';
10 - select deptonum from departamento where nome='Produção';
Ou: SELECT deptonum AS NUMERO_DEPARTAMENTO_PRODUCAO FROM departamento where nome = 'produção';
11 - select deptonum, nome from departamento where nome like 'c%';
12 - select matricula, nome from empregado where nome like '%Silva%';
13 - select matricula, nome from empregado where sobrenome like '%silva';
14 - select distinct d.deptonum from departamento d, projeto p where d.deptonum = p.depto;
15 - SELECT distinct depto FROM projeto;
Ou: select distinct deptonum from departamento, projeto where deptonum=depto;
16 - select parentesco from dependente;
Ou: SELECT PARENTESCO FROM DEPENDENTE ORDER BY PARENTESCO;
17 - select distinct parentesco from dependente;
18 - select distinct localizacao from deptolocal;
19 - SELECT DISTINCT localizacao FROM deptolocal;
20 - SELECT localizacao FROM projeto;
21 - select distinct localizacao from projeto;

6.3.2. Parte 2

22 - select nome from departamento where matgerente is null;
Ou: select nome from departamento where matgerente = 'NULL' ;
23 - select nome, matricula from empregado where supervisor is NULL;
24 - select sum(salario), max(salario),min(salario),media(salario) from empregado where depto=10;
25 - select count(*) from empregado;
Ou: SELECT COUNT(*) AS TOTAL_EMPREGADOS FROM EMPREGADO;

26 - select count(*) from empregado where depto='10';
Ou: SELECT count(*) as 'Total Funcionários' FROM departamento where depto='10';
27 - select depto, count(*) from empregado group by depto;
28 - select depto from empregado group by depto having count(depto) >=2;
29-SELECT localizacao, COUNT(*) AS 'Quantidade de Projetos' FROM projeto GROUP BY localizacao;
Ou: select localizacao, count(*) from projeto group by localizacao;
30 - select localizacao from projeto group by localizacao having count(*)>1;
Ou: select loc.localizacao, prj.* from deptolocal loc, (select * from (select depto, count(*) as num_projetos from projeto group by depto) gui where num_projetos >1) prj where loc.depto = prj.depto;
31 - SELECT localizacao, count(depto) FROM deptolocal group by localizacao ;
Ou: Select localização, count (*) From deptolocal Group by localizacao;
32 - select localizacao from deptolocal group by localizacao having count(*) >1;
33 - select count(distinct parentesco) from dependente;
34 - select parentesco, count(parentesco) as total from dependente d group by parentesco;
35 - select sexo, count(*) from dependente group by sexo;
36 - SELECT COUNT(*) FROM dependente D WHERE matricula=1001;
Ou: SELECT COUNT(*) FROM DEPENDENTE WHERE MATRICULA=1001;
37 - SELECT matricula FROM dependente group by matricula having count(matricula) >2;
Ou: select matricula from empregado E where (select count (*) from dependente D where (E.matricula=D.matricula)) > 1;
38 - select supervisor, count(*) from empregado group by supervisor;
39 - select supervisor from empregado group by supervisor having count(*) > 2;
40 - SELECT matgerente, count(*) FROM departamento group by matgerente;
41 - select matgerente from departamento group by matgerente having count(matgerente) > 1;

6.3.3. Parte 3

42 - select nome from empregado where depto in (select deptonum from departamento where nome ='desenvolvimento') and sexo = 'f';
43 - select d.nome, e.nome from empregado e, departamento d where depto = 10 and sexo = 'f' and d.deptonum = e.depto order by d.nome;
44 - SELECT d.nome, e.nome FROM departamento d, empregado e where d.matgerente = e.matricula order by d.nome;

45 - select emp.nome from empregado emp, departamento dep where emp.matricula = dep.matgerente and dep.nome = 'Producao';
Ou: SELECT B.NOME FROM DEPARTAMENTO A, EMPREGADO B WHERE A.MATGERENTE = B.MATRICULA AND A.NOME = 'Produção'
46 - select nome, matricula from empregado where nome in (select nome from empregado group by nome having count(*)> 1);
Ou: SELECT distinct(e.nome), t.matricula FROM empregado e, trabalhando t where e.matricula = t.matricula;
47 - SELECT e.nome, e.matricula FROM dependente d, empregado e where d.matricula = e.matricula and e.nome = d.nome;
48 - select matricula, nome, datanasc from empregado where extract(day from datanasc) in (select extract(day from datanasc) from empregado group by extract(day from datanasc) having count(extract(day from datanasc)) > 1) order by extract(day from datanasc);
49 – select g.matgerente, d.nome, e.nome from departamento g, departamento d, empregado e where g.matgerente=(SELECT matricula FROM empregado where nome='Mario');
Ou: SELECT E.nome, D.nome FROM empregado E JOIN departamento D ON E.supervisor = (SELECT matricula FROM empregado where nome='Mario') where d.deptonum=e.depto;
50 - select nome from empregado E where (select matricula from trabalhando D where E.matricula=D.matricula and horas>20 and codigo=(select codigo from projeto where nome ='Tamar'));
Ou: SELECT prj.nome, emp.nome as nome_funcionario, t.* FROM projeto prj, trabalhando t, empregado emp where prj.nome='tamar' and prj.depto = t.codigo and t.matricula = emp.matricula and t.horas >20
51 - SELECT e.nome, endereco, datanasc FROM empregado e, departamento d,deptolocal dl where matricula=matgerente and d.deptonum=dl.depto and dl.localizacao='Meier';
Ou: Select e.nome, e.endereço, e.datanasc From empregado e, departamento d , deptolocal dl
Where e.matricula = d.matgerente And dl.depto = d.deptonum And dl.localizacao = ‘Meier’
52 - SELECT * from projeto where depto = (select deptonum from departamento where nome=’Producao’);
53 - select nome, matricula, depto from empregado where matricula in (select matricula from trabalhando where codigo in (select codigo from projeto where nome = "Jubarte" or nome = "Tamar"));
54 - select distinct e.nome, e.matricula, e.depto from empregado e, projeto p, departamento d where e.depto = p.depto and p.depto = d.deptonum and p.nome in ('Jubarte','Tamar');
55 - SELECT sum(salario),max(salario),min(salario), avg(salario) FROM empregado where depto IN(select deptonum from departamento where nome="produção");

Ou: select sum(salario), max(salario), min(salario), avg(salario) from empregado where depto=(select deptonum from departamento where nome="produção");

56 - SELECT count(*),d.nome FROM departamento d,empregado e where e.depto=d.deptonum group by d.nome;

Ou: SELECT COUNT(DISTINCT A.MATRICULA),B.NOME FROM EMPREGADO A,DEPARTAMENTO B WHERE B.DEPTONUM=A.DEPTO GROUP BY B.NOME;

57 - SELECT count(*) FROM empregado where depto=(select deptonum from departamento where nome = 'Produção');

58 - SELECT d.nome FROM departamento d, empregado e where d.deptonum = e.depto group by e.depto having count(*) > 1;

59 - SELECT e.nome, e.matricula FROM dependente d, empregado e where d.matricula = e.matricula group by d.matricula having count(*) > 1;

60 - SELECT count(*) FROM dependente d, empregado e where d.matricula = e.matricula and e.nome='Marcus' group by d.matricula ;

61 - select nome from empregado where matricula in (SELECT matricula FROM trabalhando group by matricula having count(matricula) > 2);

62 - select a.NOME from empregado a where (select count(*) from departamento b where a.MATRICULA = b.matgerente)>=2 ;

63 - SELECT matricula, nome from empregado where matricula in (select distinct supervisor from empregado);

64 - SELECT matricula, nome from empregado where matricula in (select distinct supervisor from empregado group by supervisor having count(*) >2);

65 - select proj.codigo, proj.nome, count(*) as 'Quantidade de Funcionarios', avg(trab.horas) as 'Média de Horas' FROM projeto proj, trabalhando trab where proj.codigo=trab.codigo group by trab.codigo ;

Ou: SELECT A.CODIGO,A.NOME,COUNT(*) AS TOTAL_FUNCIONARIOS,AVG(HORAS) AS MEDIA_HORAS FROM PROJETO A, TRABALHANDO B WHERE A.CODIGO = B.CODIGO GROUP BY A.CODIGO,A.NOME;

6.3.4. Parte 4

66 - select e.nome from empregado e, departamento d where e.sexo='f' and e.depto=d.deptonum and d.nome='Produção';

Ou: SELECT e.nome FROM departamento d, empregado e where d.deptonum = e.depto and d.nome = 'Produção' and e.sexo='f';

67 - SELECT e.nome from empregado e, departamento d where d.matgerente=e.matricula and d.nome='Produção';

68 - select sum(salario) as Total_Sal, max(salario) as Maior_Sal, min(salario) as Menor_Sal, avg(salario) as Media_Sal from empregado where depto=(select deptonum from departamento where nome = 'Produção');

69 - SELECT COUNT(*) AS 'Total de Funcionarios' FROM empregado where depto=(SELECT deptonum FROM departamento where nome='producao');

Ou: select count(*) from empregado where depto in(select deptonum from departamento where nome='producao');

70 - select nome from departamento D where (select count(*) from empregado E where E.depto = D.deptonum)>1;

Ou: Select d.localizacao, func.* from (select * from (SELECT depto, count(matricula) as num_funcionarios FROM empregado e group by depto) tb where num_funcionarios > 1) func, deptolocal d where d.depto = func.depto;

71 - Select e.matricula , e.nome, count(*) From empregado e Where e.supervisor = e.matricula group by e.matricula,e.nome;

72 - select matricula, nome from empregado where matricula in (select supervisor from empregado group by supervisor having count(*)>2);

73 - select d.nome, e.nome from departamento d, empregado e where d.deptonum = e.depto and supervisor in (select matricula from empregado where nome = 'Mario Souza');

74 - select distinct e.nome from empregado e, departamento d where e.matricula in (select matgerente from departamento group by matgerente having count(*)>1);

75 - SELECT nome,endereco,datanasc FROM empregado where depto IN(select depto from deptolocal where localizacao="meier");

76 - SELECT nome,matricula FROM empregado E WHERE (SELECT COUNT(*) FROM dependente D WHERE E.matricula = D.matricula) > 2;

77 - select count (*) from dependentes where matricula=(select matricula from empregado where nome = ‘Marcus’);

78 - select nome from empregado where matricula IN (select matricula from dependente);

79 - select nome from empregado where matricula in (select distinct matgerente from departamento where matgerente in (select distinct matricula from dependente));

80 - SELECT e.nome, e.matricula FROM dependente d, empregado e where d.matricula = e.matricula and e.nome = d.nome;

81 - SELECT e.nome from empregado e, trabalhando t where e.matricula=t.matricula group by t.matricula having count(t.codigo)>=2;

82 - select nome from empregado where depto in (select depto from projeto where codigo in (select codigo from trabalhando where horas > 20) and nome = 'FornecNorte');

83 - select e.nome, e.depto, t.matricula, p.nome from empregado e, trabalhando t, projeto p where p.nome='fornecnorte' or p.nome='fornesul';

84 - select e.nome, e.depto, t.matricula, p.nome from empregado e, trabalhando t, projeto p where p.nome='fornecnorte' and p.nome='fornesul';

6.4. Agradecimento aos alunos colaboradores:

Sem os seguintes alunos, cursando banco de dados 2 no turno da manhã e implementação de banco de dados no turno da noite, todos no período de 2007-2 no Campus Nova América da Universidade Estácio de Sá, não seria possível a correção de toda esta lista:

AILTON PEREIRA DA CRUZ JUNIOR; ALCIDINEI FERNANDES DE ANDRADE; BRUNO ANTONIO DE SOUZA; BRUNO SABBADO DE FRANCO; BRUNO SCHMIDT E SILVA; FERNANDO CRUZ TABOADA DA SILVA; FLAVIA PEREIRA BASTOS; JULIANA CLAUDIO DE SANTANA; LAÍS BARBATTI BORGES; LEONARDO MIRANDA DO NASCIMENTO; LUIZ CARLOS DE SALLES CUNHA JUNIOR; RAFAEL DE SOUZA AZEREDO; RODRIGO FREIRE DE MELLO; TATIANE DE SOUZA ALBUQUERQUE NOGUEIRA; DIENNE EVELIN LIMA OLIVEIRA; GUILHERME OTAVIO FERREIRA LAUBERT; JESSE SEMINIO COUTINHO; JORGE LUIZ RODRIGUES TEIXEIRA; JORGE MIGUEL DA COSTA HENRIQUES CARDOSO; JOSÉ ALBERTO BRITO E SILVA; JOSE CLAUDIO ALMEIDA MARTINS; LEONARDO ROSSETTO; MARCELO MENDES BATTISTELLA; MARCOS VINICIUS DO AMARAL TEIXEIRA; MARISA LESSA FONTES DE MELO e WALTER ALEXANDER AUGUSTE JUNIOR

Obrigado a todos!

7 – S Q L (Parte III)

7.1. Visões (ou *VIEWs*)

Uma *VIEW* é uma tabela derivada de outras tabelas. Estas outras tabelas podem ser tabelas base ou outras views pré-definidas. É considerada uma tabela virtual.

Visões são criadas para restringir o acesso ao banco de dados, tornar consultas complexas mais simples, para permitir independência dos dados e/ou para apresentar diferentes visões do mesmo dado.

Em sua criação é dado um nome para a tabela virtual, uma lista de atributos (opcional) e uma query que determina o conteúdo da view:

```
CREATE VIEW nome-view  { nome_coluna1, nome_coluna2…}
        AS   <comando select>
```

Por exemplo:

```
CREATE VIEW infodepto AS
        SELECT          D.nome AS NomeDepto,
                        count(*) AS NumFunc,
                        sum(E.salario) AS TotalSal
        from departamento D, empregado E
        where (D.deptonum = E.depto) group by D.deptonum;
```

Com esta view fica bem fácil acessar diretamente o número de funcionários e a soma salarial de cada departamento da empresa. É só fazer um query de select na VIEW como se fosse em uma tabela:

```
SELECT * FROM infodepto;
```

Da mesma forma para se apagar uma visão, procede-se como se fosse para uma tabela comum, apenas trocando o *TABLE* por *VIEW*:

```
DROP VIEW infodepto;
```

8 – Bibliografia

Bibliografia Básica para o curso de Banco de Dados 2 :

- ELMASRI, Ramez; NAVATHE, Shamkant B. *Sistemas de banco de dados*. São Paulo: Pearson, 2005.

- OLIVEIRA, Celso Henrique Poderoso B. *SQL: curso prático*. São Paulo: Novatec, 2002.

Bibliografia Complementar para o curso de Banco de Dados 2 :

- DATE, C. J. *Introdução a sistemas de bancos de dados*. Rio de Janeiro: Campus, 2000.

- SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. *Sistema de bancos de dados*. 3ª ed. São Paulo: Makron, 1999.

Bibliografia Básica para o curso de Implementação de BD :

- DATE, C. J. *Introdução a sistemas de bancos de dados*. Rio de Janeiro: Campus, 2000.

- SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. *Sistema de bancos de dados*. 3ª ed. São Paulo: Makron, 1999.

- MELO, Rubens N; SILVA, Sidney Dias da; TANAKA, Astério K. *Banco de dados em aplicações cliente-servidor*. Rio de Janeiro: Infobook, 1997.

- GUIA de referência SQL versão 7.2. São Paulo: dbExperts, 2002.

- LONEY, Kevin; THERIAULT, Marlene. *Oracle 8i: o manual do DBA*. Tradução Kátia A. Roque. Rio de Janeiro: Campus, 2000.

Bibliografia de Referência para estas Notas de Aula :

- GUIA de referência do MySQL versão 4.1. 2006.