Banco de Dados I

Desenvolvido por:
Prof[a]: Tanisi Pereira de Carvalho
Prof[a]: Simone Vicari

Revisado por:
Profa. Patrícia Hübler

Sumário

1. Introdução a Sistemas de Bancos de Dados....4
1.1 Objetivos dos sistemas de bancos de dados....4
1.2 Funções de um SGBD....6
1.3 Níveis de Visões....7
1.4 Usuários do SGBD....8
2. Modelos de Dados....9
2.1 Modelo Hierárquico....9
2.2 Modelo de Rede....12
3. Diagrama Entidade-Relacionamento....14
3.1 Elementos Básicos....14
3.2 Outros Conceitos....17
3.3 Que Cuidados Tomar ao Construir um Modelo E-R....18
4. O Modelo Relacional....21
4.1 Chaves....21
4.2 Mapeamento do Modelo ER para Modelo Relacional....23
4.2.1 Implementação de relacionamento 1:1....23
4.2.2 Implementação de Relacionamento 1:N....24
4.2.3 Implementação de Relacionamento N:N....24
4.2.4 Implementação de Generalização/Especialização....25
4.2.5 Implementação de Agregação....26
4.3 Normalização....27
5. Linguagens de consulta....33
5.1 Linguagem SQL (Structured Query Language)....33
5.2 Álgebra Relacional....39
6.Visões e Segurança de Acesso....45
6.1 Visões....45
6.2 Segurança de Acesso....47
7. Anexo - Tipo de Dados DATE do Oracle....48

Caro Aluno,

Esta apostila tem como objetivo permitir que você possa acompanhar a aula, fazendo algumas anotações e servindo como orientação no momento em que você for estudar os temas abordados. Apesar disso, não dispensa a leitura dos livros indicados na bibliografia, principalmente do livro texto da disciplina, uma vez que eles tratam os assuntos com mais detalhes, permitindo, assim, o enriquecimento no estudo de Bancos de Dados.

Bom trabalho!

Alguns DER , exemplos e conceitos foram retirados do livro Projeto de Banco de Dados do prof. Carlos Alberto Heuser.

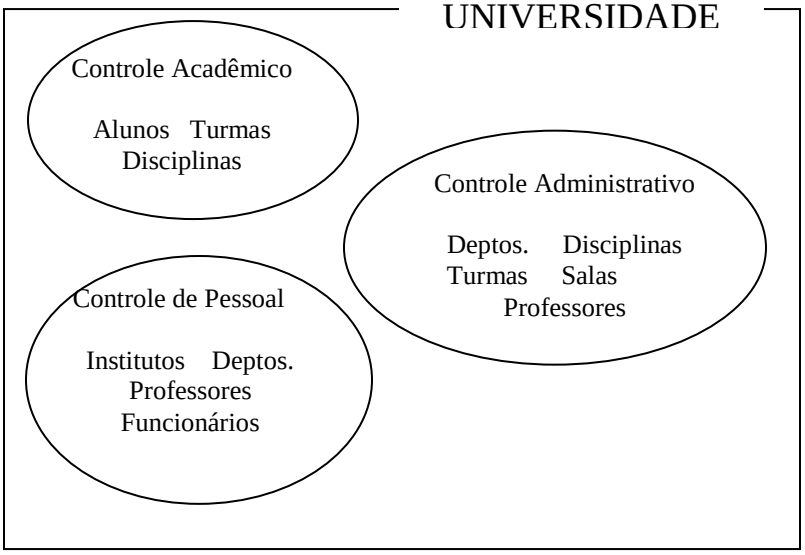
1. Introdução a Sistemas de Bancos de Dados

Para entender o estudo de banco de dados é necessário a definição de três termos principais:

- *banco de dados*: corresponde a um conjunto de informações relacionadas que possuem algum significado. São informações referentes a um empreendimento particular;
- *sistema gerenciador de banco de dados (SGBD)*: consiste em uma coleção de dados interrelacionados e em um conjunto de programas para acessá-los. O principal objetivo de um SGBD é prover um ambiente que seja conveniente e eficiente para recuperar informações de banco de dados;
- *sistema de banco de dados (SBD)*: é constituído pelo banco de dados propriamente dito e o software necessário para gerenciá-lo (SGBD) e manipulá-lo (consultas e aplicações do usuário).

1.1 Objetivos dos sistemas de bancos de dados

Para entender a importância dos SBD observe o exemplo abaixo baseado num sistema de arquivos convencional, ou seja, que não utiliza um banco de dados.



Situação:

- Cada aplicação da organização com o seu conjunto de dados.
- Descrição dos dados fica dentro da aplicação.

- Não existe compartilhamento de dados entre as aplicações.

Problemas:

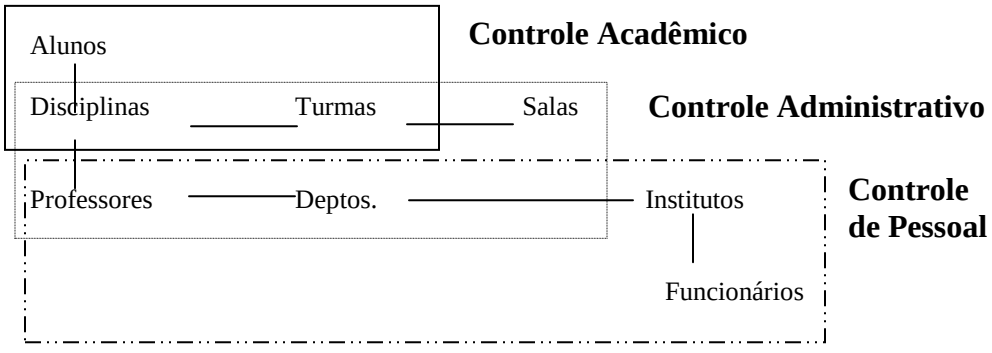
- Redundância de dados: um sistema de arquivos pode gerar um alto nível de redundância e duplicação de dados, porque o mesmo dado é armazenado em diferentes arquivos;
- Difícil manutenção dos dados (inconsistência dos dados): a manutenção fica difícil porque é necessário fazer a manutenção em mais de um lugar. Quando a mudança de um mesmo dado é feita em um arquivo mas não nos outros, o resultado são dados inconsistentes;
- Falta de uma padronização na definição dos dados: posso ter o mesmo dado com o formato e tipo diferentes;
- Não há preocupação com a segurança dos dados (segurança de acesso e segurança contra falhas);
- Limitações no compartilhamento dos dados: cada usuário tem os seus arquivos e programas de aplicação. Fica difícil o compartilhamento de dados contribuindo para a redundância e ineficiência no geral. Devido a este ambiente descentralizado fica difícil estabelecer um padrão e um controle sobre quais dados são processados.

Necessidade de um banco de dados:

- melhor organização e gerência dos dados;
- controle centralizado dos dados.

* Os dados, em um banco de dados, são armazenados de forma independente dos programas que os utilizam, servindo assim a múltiplas aplicações de uma organização.

Exemplo com BD

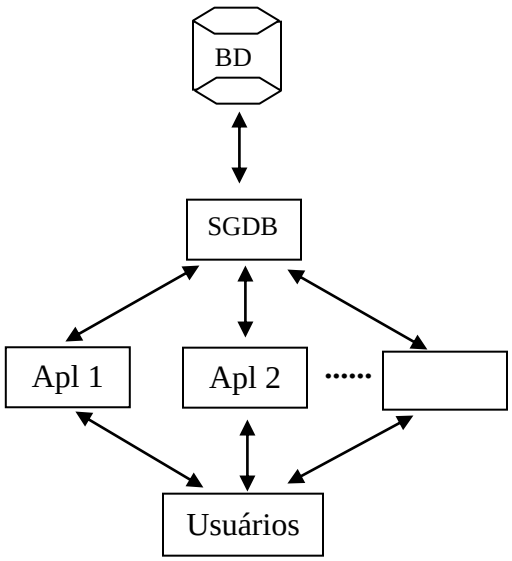


Vantagens:

- Dados são armazenados em um único local físico
- Dados são compartilhados pelas aplicações

- Independência dos dados
- Aplicações não se preocupam com a gerência dos dados

Um Banco de Dados dentro do contexto de um ambiente computacional



1.2 Funções de um SGBD

a) Definição de dados e métodos de acesso

- DDL (Data Definition Language): especificação do esquema do BD
- DD (Dicionário de Dados): armazenamento dos metadados (catálogo do sistema)
- DML (Data Manipulation Language): permite a manipulação de dados
- Processamento eficaz de consulta

b) Restrições de Integridade: controle sobre a integridade dos dados armazenados.
   - Estados possíveis de serem assumidos pelos dados. Ex: o código de um aluno deve ser um valor entre 000 e 999.
   - Manutenção de relacionamentos válidos entre os dados. Ex: todo professor deve pertencer a pelo menos um departamento.
   - Triggers

c) Segurança dos Dados
   - Controle de acesso (autorização)
   - Segurança contra falhas: Transação, Sistema e Meio de armazenamento

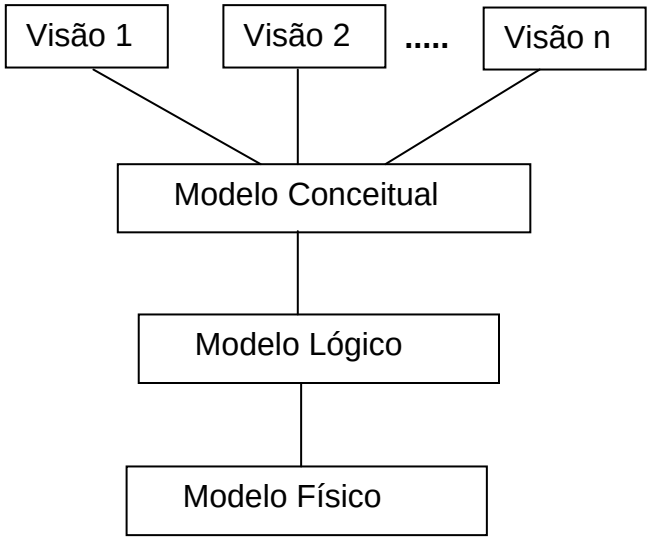
d) Controle de Concorrência: sanar conflitos de acesso a dados

e) Independência dos dados

- Independência Física: modificações no esquema de gerenciamento dos dados (esquema físico) sem afetar a implementação das aplicações e a definição conceitual dos dados.
- Independência Lógica: independência da estrutura conceitual dos dados.
  - ❖ Visões diferenciadas da estrutura conceitual (Ok!)
  - ❖ Modificações na estrutura conceitual (problema!)

1.3 Níveis de Visões

Forma de visão dos dados dentro do contexto aplicação-SGBD

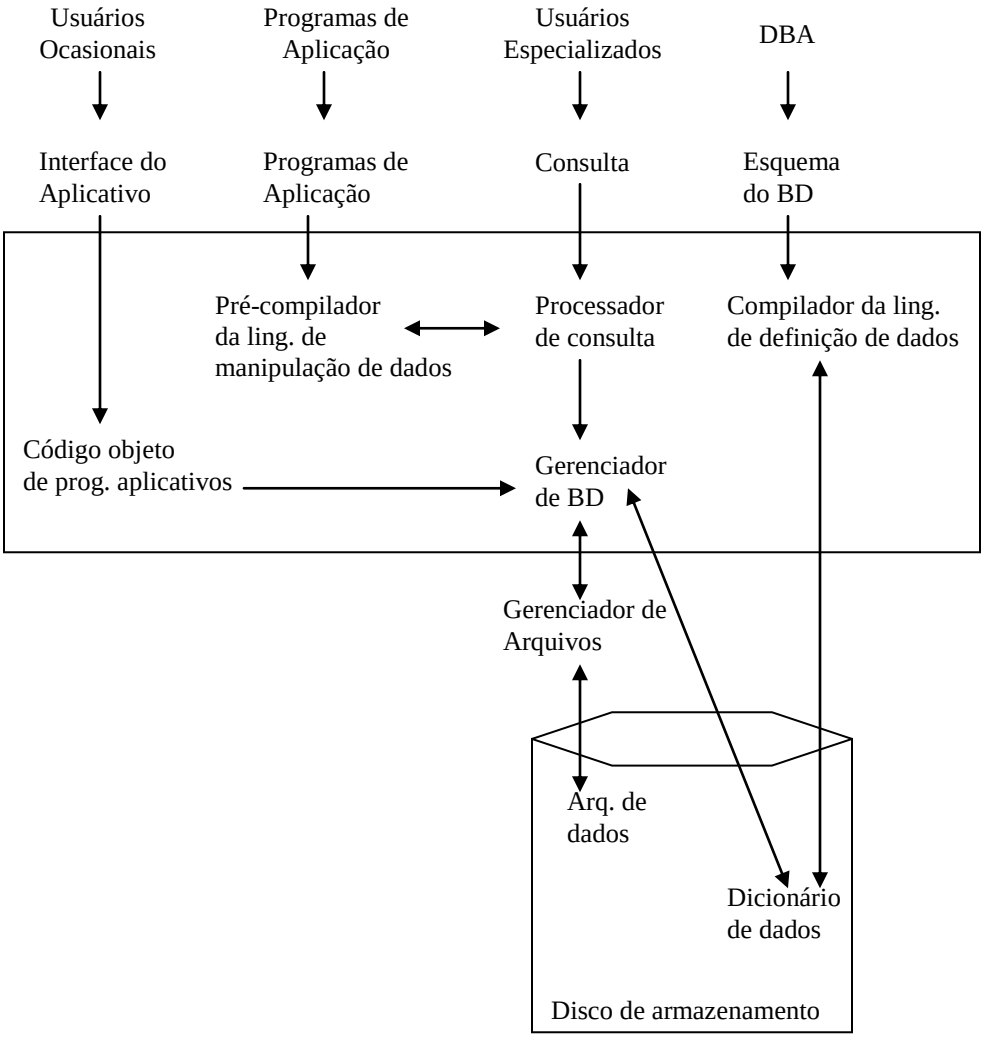


1.4 Usuários do SGBD

a) Administrador de banco de dados (DBA): controle de diversas funcionalidades do SGBD
   - definição do esquema conceitual
   - definição da estrutura de armazenamento e métodos de acesso
   - modificação do esquema conceitual, estrutura de armazenamento e métodos de acesso
   - concessões de autorização de acesso
   - especificação de restrições de integridade
   - controle das operações de recuperação após falhas

b) Usuários especializados: interagem diretamente com o SGBD
c) Programadores de aplicação: definem aplicações para acesso aos dados
d) Usuários ocasionais: utilizam aplicações que acessam o BD.

Estrutura Geral do SGBD



2. Modelos de Dados

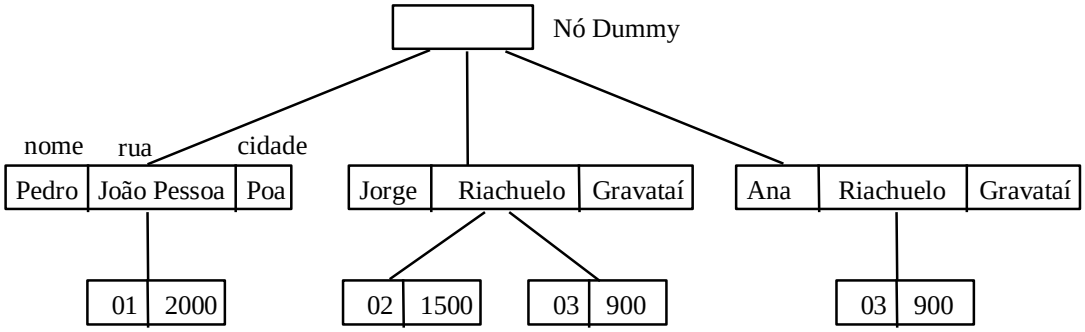
⇒ **Hierárquico**
⇒ **Rede**
⇒ **Relacional**
⇒ **Abordagem pós-relacional**

2.1 Modelo Hierárquico

Importância: IMS (information Management System) da IBM, largamente utilizado durante a década de 70 e início da década de 80;

Características
- Dados organizados em uma coleção de registros interconectados através de ligações.
  Registros: Conjuntos de campos (informações ≠ s)
  Ligação : Associação entre dois registros .

- Esquema definido em forma de árvore
EX .: Sistema Bancário : 2 registros
⇒ cliente
⇒ conta



Representação Diagramática

- Diagrama de Estrutura em Àrvore (DEA)

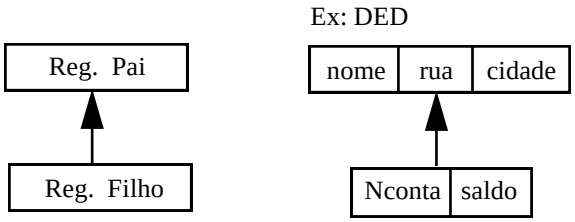
a) Características

⇒ Caixas: Tipos de Registros
⇒ Linhas: Ligações

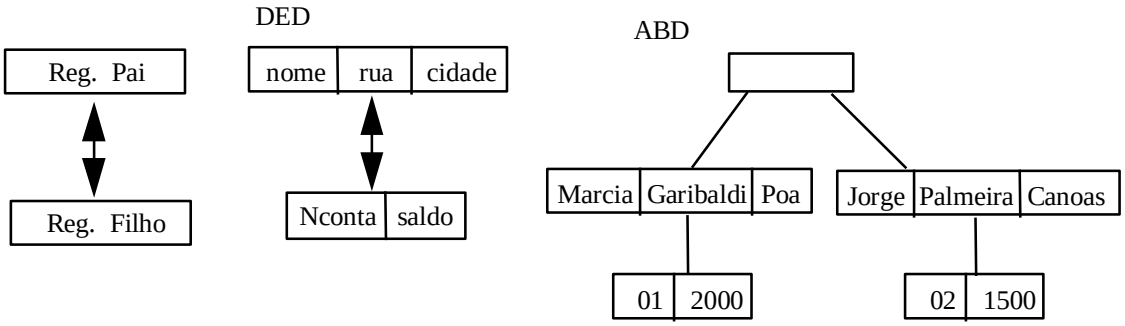
DED - O diagrama de estrutura de dados é um esquema para o BD. Consiste de dois componentes: caixas (que correspondem a tipos de registros) e linhas (que correspondem a ligações entre este registros).

b) Representação dos Relacionamentos

1-Relacionamentos um-para-muitos:



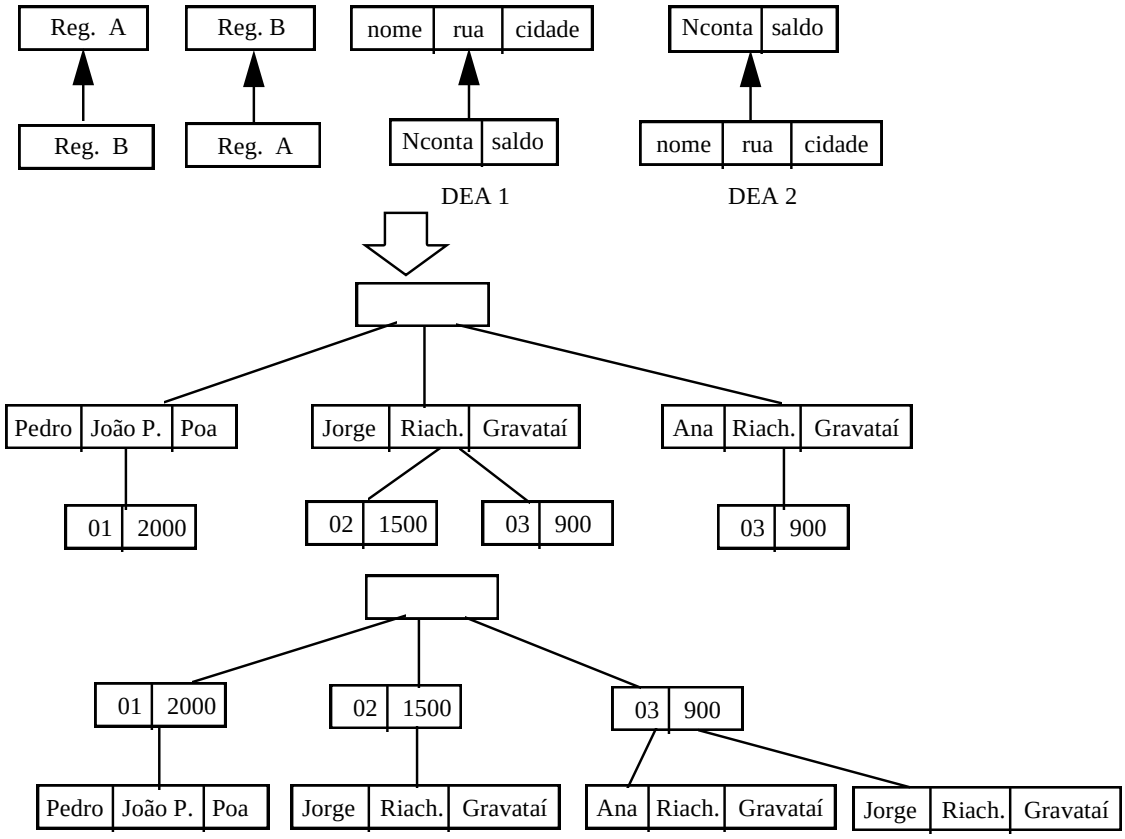
2-Relacionamento um-para-um:



3-Relacionamentos muitos-para-muitos:

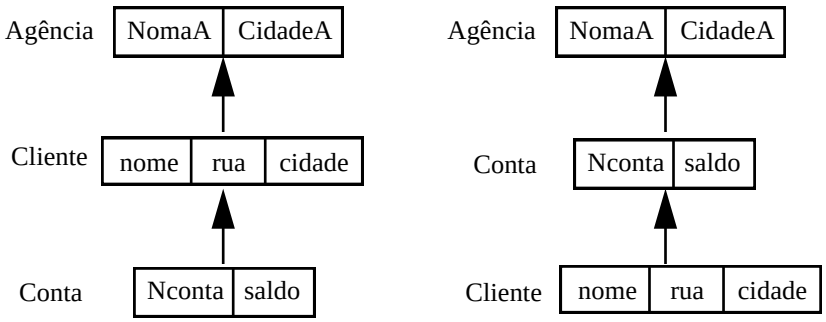
O modelo não suporta este relacionamento diretamente.

Solução: Aplicar o princípio um-para-muitos p/cada um dos dois tipos de registros.



- 2 DEAs: Redundância Maior
- 1 DEAs: Depende do tipo de Consulta

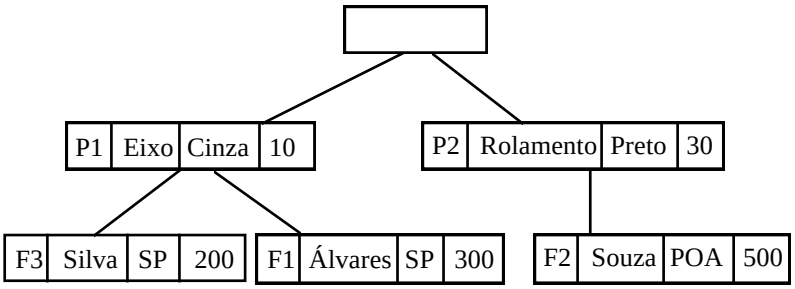
Exemplo2: Um cliente pode ter diversas contas em uma agência de banco. Uma conta pode pertencer a diversos clientes.



Problemas

⇒ Redundância de informação
ex: relacionamento muitos-para-muitos (cliente-conta)

⇒ Dificil atualização de registros filhos
ex: percorrer toda a árvore para atualizar todos os saldos de uma determinada conta
⇒ Problema na inserção
ex: para inserir os dados de um forncedor que ainda não forneceu nenhuma peça, seria necessário criar um registro fantasma



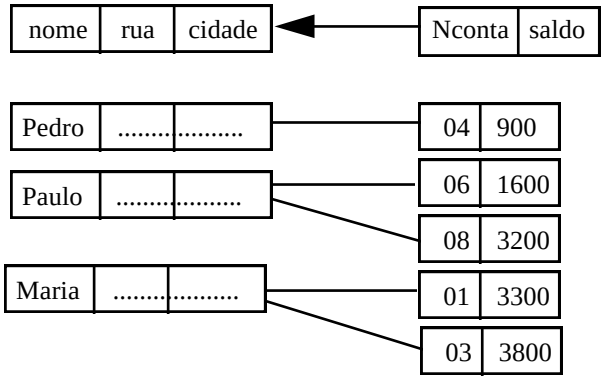
⇒ Problemas na exclusão
ex: a exclusão da única peça fornecida por um fornecedor inplicaria na exclusão de seus dados
⇒ Problemas de Assimetria nas Consultas
Tempo de acesso a registros dos níveis mais baixos na hierarquia

2.2 Modelo de Rede

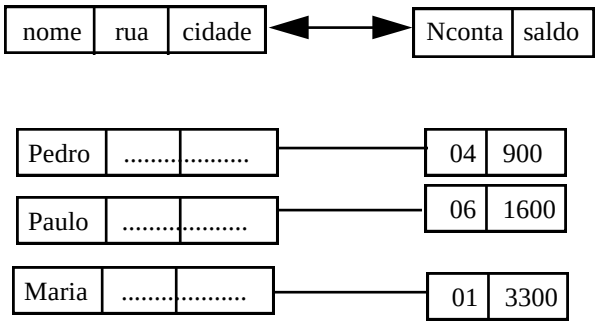
Características
⇒ Dados organizados em registros conectados através de ligações
⇒ Ligações : Associações entre exatamente 2 tipos de registros.
⇒ Permite todos os tipos de relacionamentos: 1:1 , 1:N , M:N
⇒ 1971 : Especificação padrão de banco de dados: Modelo DBTG (Data Base Task Group) CODASYL (Conference on Data Systems and Languages)
⇒ surgem diversos BD comerciais baseados no modelo de Rede.

Relacionamentos

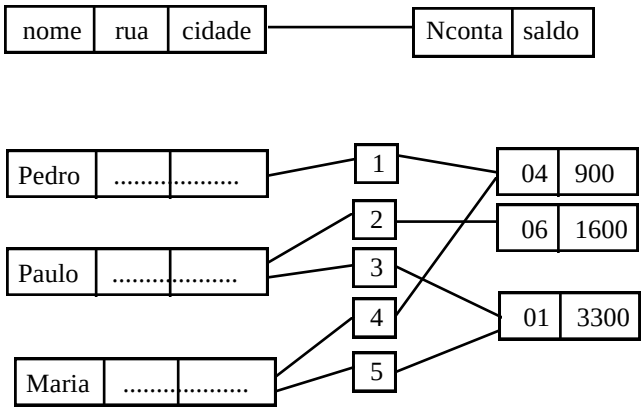
⇒ relacionamento 1:N



⇒ relacioanamento 1:1



⇒ relacionamento M:N



Vantagens em relação ao modelo Hierárquico

⇒ Representação de todos os tipos de relacionamentos
⇒ Sem redundância de informações
⇒ Maior simetria nas consultas

Desvantagens do Modelo de Rede

⇒ Bancos de dados com muitos tipos de entidades podem resultar em esquemas muito confusos;
⇒ Utilização de ligação física entre os registros;

3. Diagrama Entidade-Relacionamento

Alguns DER , exemplos e conceitos foram retirados do livro Projeto de Banco de Dados do prof. Carlos Alberto Heuser.

3.1 Elementos Básicos

A) ENTIDADE

⇒ Conjunto de objetos da realidade modelada sobre os quais deseja-se manter informação.

B) RELACIONAMENTO

⇒ Conjunto de associações entre entidades

<u>Cardinalidade de Relacionamentos</u>

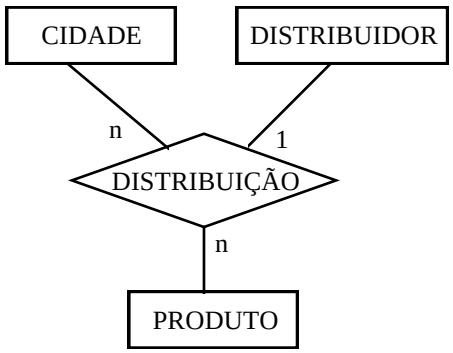
⇒ Cardinalidade (mínima, máxima) de entidade em relacionamento = número (mínimo, máximo) de ocorrências de entidade associadas a uma ocorrência da entidade em questão através do relacionamento.

⇒ Cardinalidade máxima: 1 ou muitos (n)
⇒ Ex. com cardinalidade máxima:

1 - expressa que a uma ocorrência de empregado pode estar associada uma ocorrência de departamento;
n - expressa que a uma ocorrência de departamento podem estar associadas muitas ocorrências de empregados;

Relacionamentos Ternários

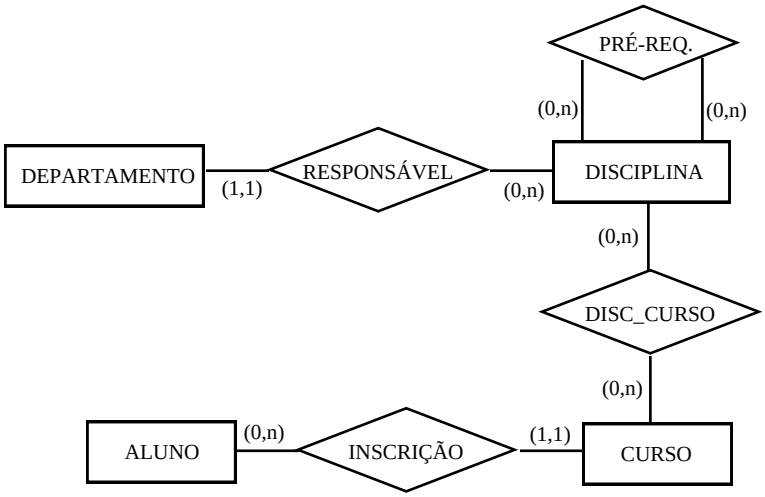
⇒ Em um relacionamento ternário a cardinalidade se refere a pares de entidades.



Cardinalidade Mínima

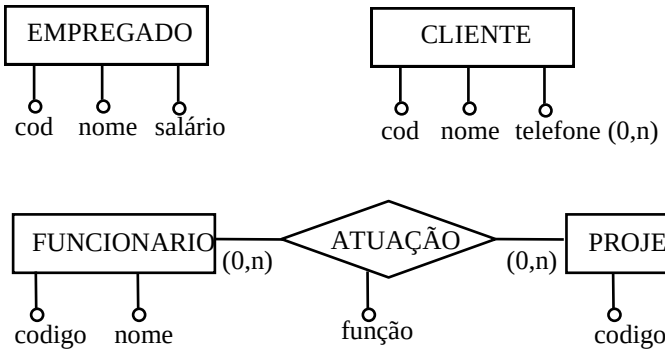
⇒ Número mínimo de ocorrências de entidades que são associadas a uma ocorrência de entidade através de um relacionamento.
⇒ Cardinalidades: 0 (associação opcional) ou 1 (associação obrigatória).

⇒ DER para o controle acadêmico de uma universidade

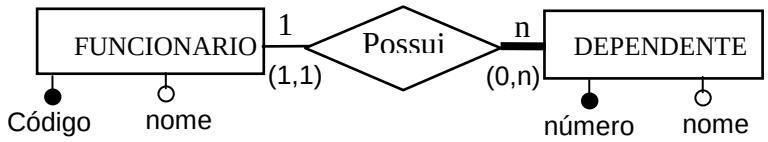


C) ATRIBUTO

⇒ Dado que é associado a cada ocorrência de uma entidade ou de um relacionamento.



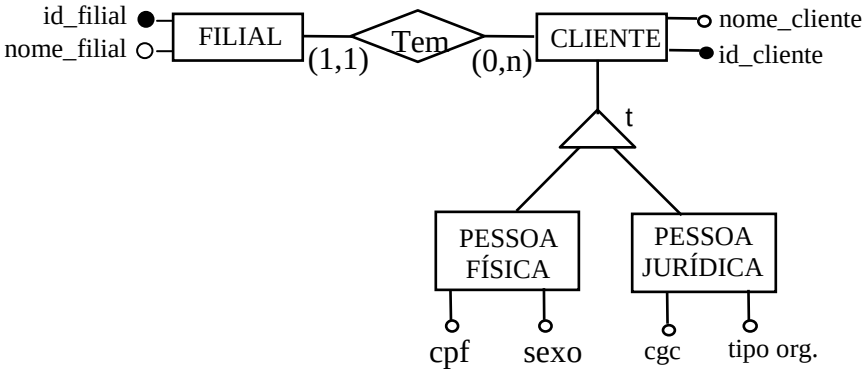
Identificador de Entidade



Identificador de Relacionamentos

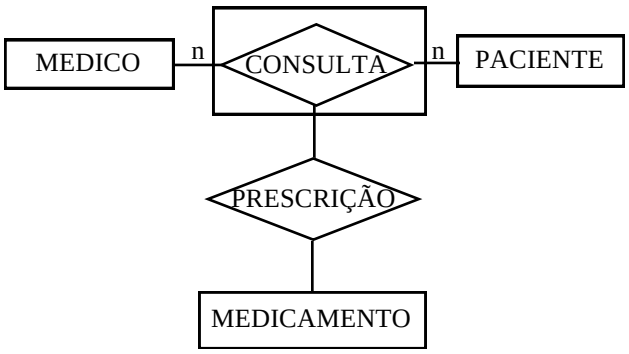
3.2 Outros Conceitos

GENERALIZAÇÃO/ESPECIALIZAÇÃO



ENTIDADE ASSOCIATIVA

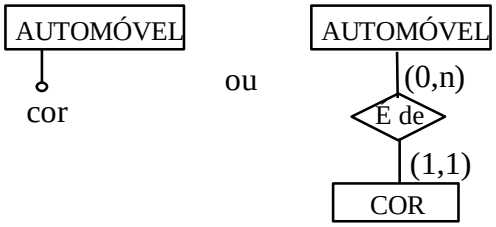
⇒ incluir os medicamentos que são prescritos por uma médico a um paciente em uma consulta;



3.3 Que Cuidados Tomar ao Construir um Modelo E-R

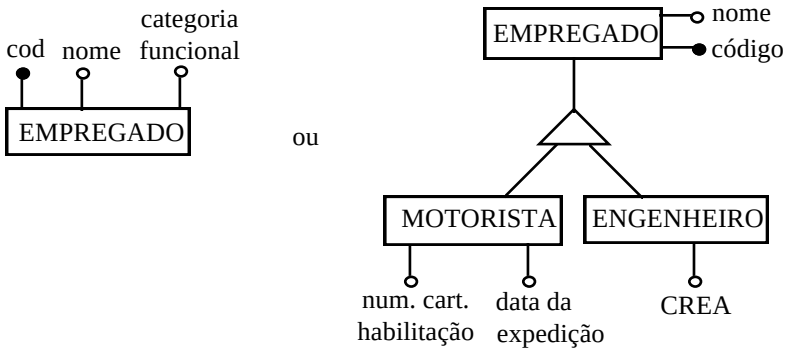
a) Atributo versus entidade relacionada

⇒ Caso o objeto cuja modelagem está em discussão esteja vinculado a outros objetos, o objeto deve ser modelado como entidade. Caso contrário, o obj. deve ser modelado como atributo.



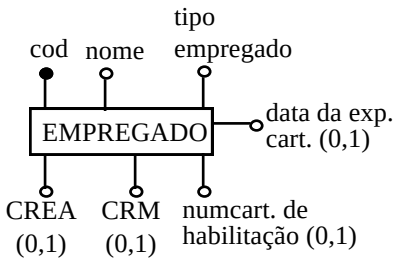
b) Atributo versus generalização/especialização

⇒ Uma especialização deve ser utilizada quando sabe-se que as classes especializadas de entidades possuem propriedades particulares.

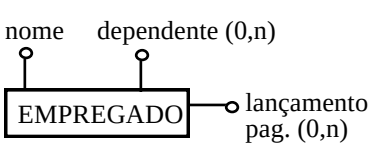


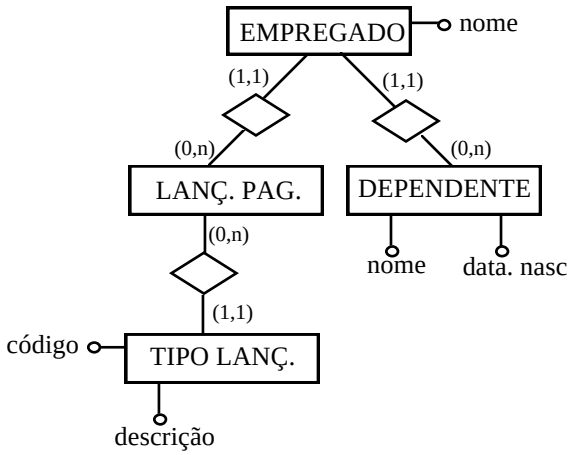
c) Atributos opcionais e multi-valorados

⇒ Atributos opcionais indicam subconjuntos de entidades que são modeladas mais corretamente através de especializações.



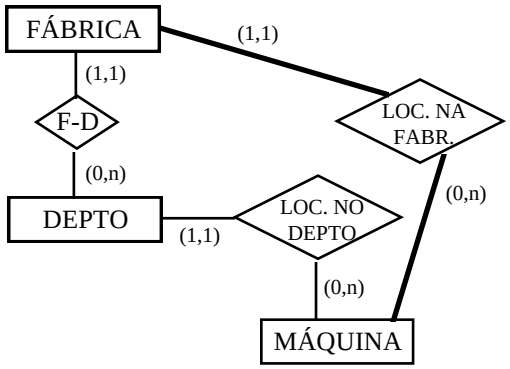
⇒ Atributos multi-valorados são indesejáveis.



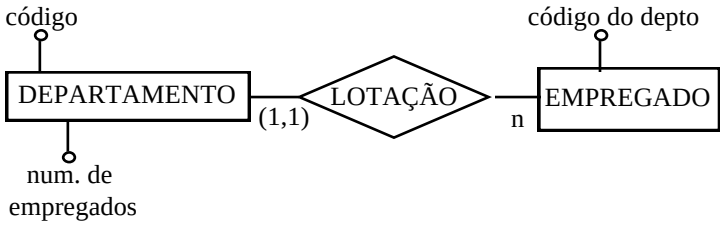


d) O modelo deve ser livre de redundância

⇒ Relacionamentos redundantes - são relacionamentos que são resultado da combinação de outros relacionamentos entre as mesmas entidades.



⇒ Atributos redundantes - são atributos deriváveis a partir da execução de procedimentos de busca de dados e/ou cálculos sobre a base de dados.



4. O Modelo Relacional

Um banco de dados relacional é composto por um único tipo de construção: a tabela. Uma tabela é composta linhas (tuplas) e colunas (atributos).

As ligações entre linhas de diferentes tabelas são feitas através do uso de valores de atributos.

Por exemplo:

Tabela de Departamentos

id_departamento	nome_departamento
100	Vendas
101	RH
102	Informática

Tabela de Funcionários

id_funcionario	nome_funcionario	Id_departamento
1	Ana	100
2	Heitor	101
3	Ricardo	100

∗ Estas tabelas podem estar relacionadas? Sim, a partir do campo id_departamento

4.1 Chaves

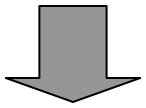
Um conceito importante para o modelo relacional é o conceito de chave. Nesse contexto tem-se três tipos de chaves:

- *chave primária*: é qualquer coluna ou combinação de colunas que identifica uma única tupla em uma tabela;
- *chave estrangeira*: é uma coluna ou combinação de colunas, cujos valores aparecem necessariamente na chave primária de uma tabela;
- *chave alternativa*: em alguns casos, mais de uma coluna ou combinações de colunas podem servir para distinguir uma linha das demais. Uma das colunas é escolhida como chave primária e as demais são chamadas chaves alternativas.

Exemplos:

Dependente

Id_funcionario	NoDependente	Nome	Tipo	Data_nascimento
1	01	João	filho	12/12/1991
1	02	Maria	esposa	01/01/1950
2	01	Ana	esposa	05/11/1955
5	01	Paula	esposa	04/07/1962
5	02	José	filho	03/02/1985

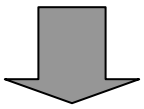


chave primária

∗ Em algumas situações (como na tabela Dependente), apenas um dos valores dos campos que compõem a chave não é suficiente para distinguir uma linha das demais. É necessário a utilização de outro campo, neste caso a tabela possui uma chave primária composta. No exemplo, (CodEmpregado,NoDependente) é a chave primária da tabela Dependente.

Departamentos

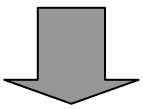
id_departamento	Nome_departamento
100	Vendas
101	RH
102	Informática



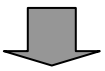
chave primária

Funcionários

id_funcionario	nome_funcionario	id_departamento
1	Ana	100
2	Heitor	101
3	Ricardo	100



chave primária



chave estrangeira em relação à tabela Departamentos

Tabela de Funcionários

id_funcionario	nome_funcionario	id_departamento	RG
1	Ana	100	122.200.321
2	Heitor	101	220.110.450
3	Ricardo	100	895.325.562

chave alternativa

4.2 Mapeamento do Modelo ER para Modelo Relacional

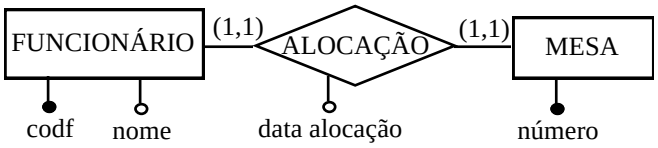
O Modelo ER (Entidade-Relacionamento) é um modelo de dados semântico que descreve a realidade independente dos aspectos de implementação.

O Modelo Relacional descreve a realidade a nível de SGBD relacional (tabela, atributos e relacionamentos implementados através de chaves estrangeiras).

Sendo assim, é necessário regras de transformação do modelo ER para o modelo relacional.

4.2.1 Implementação de relacionamento 1:1

a) As duas entidades são obrigatórias no relacionamento

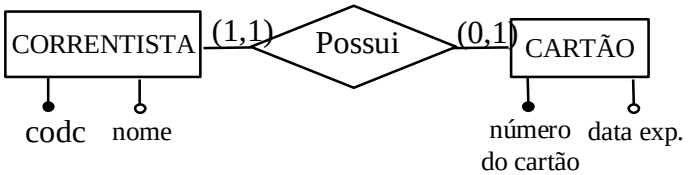


Funcionario(<u>codf</u>,nome,data,num_mesa)

A implementação preferida é a fusão de tabelas mas pode ser utilizada, tabela própria ou adição de coluna.

b) Uma das entidades é opcional no relacionamento

A implementação preferida é a fusão das tabelas, mas poderia ser utilizada a adição de colunas.



Correntista(<u>codc</u>, nome)

Cartao(<u>num_car</u>, data, codc)

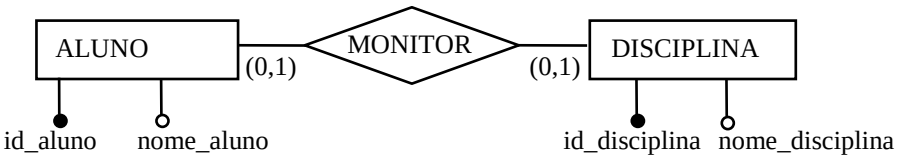
↑

chave estrangeira

obs: a implementação utilizando-se uma única tabela geraria muitos campos vazios, caso houvesse muitos correntistas sem cartão de crédito;

c) As duas entidades são opcionais no relacionamento

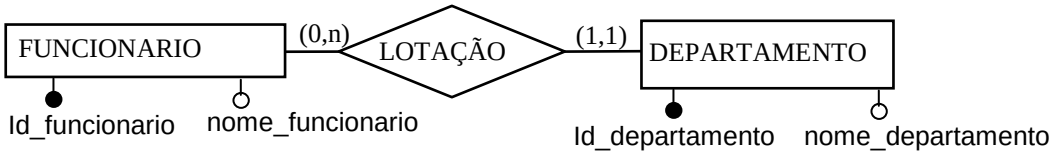
A alternativa preferida é a adição de coluna.



Disciplina (id_disciplina, nome_disciplina);
Aluno (id_aluno, nome_aluno, id_disciplina);

4.2.2 Implementação de Relacionamento 1:N

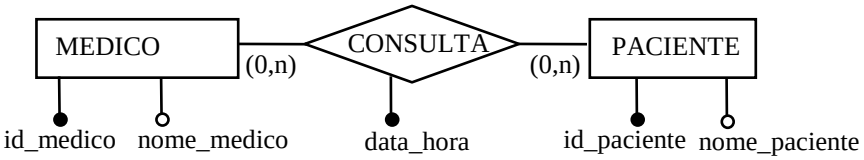
A alternativa preferida para implementação de relacionamento 1:n é a adição de coluna.



Departamento (<u>id_departamento</u>, nome_departamento)

Funcionário(<u>id_funcionario</u>, nome_funcionario, id_departamento)
↑
chave estrangeira

4.2.3 Implementação de Relacionamento N:N



O relacionamento deve ser implementado através de uma tabela;

Medico(id_medico, nome_medico);

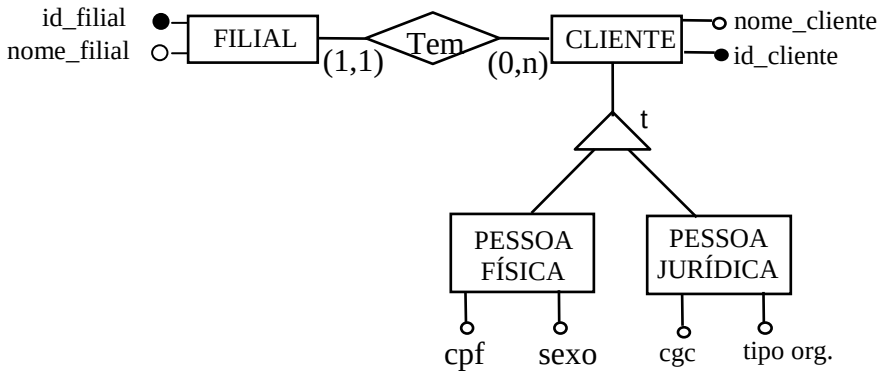
Paciente(id_paciente, nome_paciente);

Consulta(id_medico, id_paciente, data_hora);

↑ ↑

chaves estrangeiras

4.2.4 Implementação de Generalização/Especialização



⇒ Alternativa 1

Filial (id_filial, nome, tipo);

Cliente (id_cliente, nome_cliente, id_filial); dados de todos os clientes

Pessoa_Fisica (id_cliente, cpf,sexo); só pessoas físicas

Pessoa_Juridica (id_cliente, cgc,tipo_org); só pessoa jurídica

obs: id_cliente é chave estrangeira nas tabelas Pessoa_Fisica e Pessoa_Juridica.

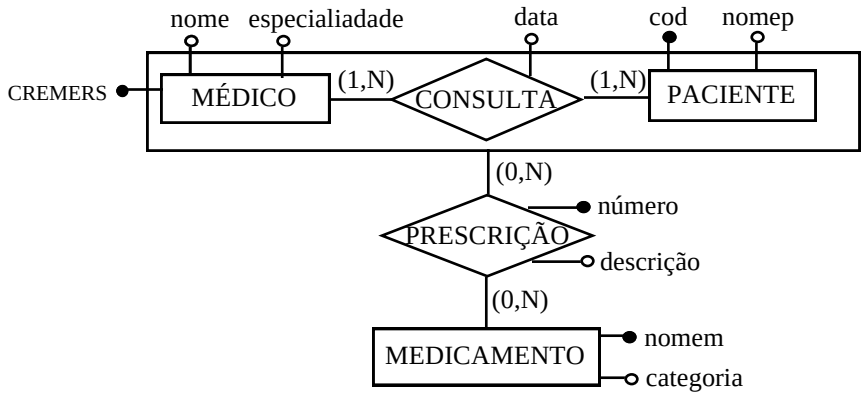
⇒ Alternativa 2:

Cliente (id_cliente, nome_cliente, id_filial, cpf,sexo, cgc,tipo_org);

⇒ Alternativa 3

Filial (id_filial, nome, tipo);
Pessoa_Fisica (id_cliente, nome_cliente, id_filial , cpf,sexo); só pessoas físicas
Pessoa_Juridica (id_cliente, nome_cliente, id_filial , cgc,tipo_org); só pessoa jurídica
Para generalizações parciais deve-se criar uma tabela para a entidade genérica.

4.2.5 Implementação de Agregação



Medico(cremers, nome, especialidade);
Paciente(cod, nomep);
Consulta (cremers, cod, data);
Medicamento(nomem, categoria);
Prescricao(numero, cremers, cod, nomem, descricao);

4.3 Normalização

Normalização é o processo através do qual uma tabela relacional não normalizada é transformada em um conjunto de tabelas normalizadas, que representam da forma mais adequada a realidade modelada.

Comercial de Peças		CGC: 00.900.111/0001-07 Insc.Est.: 096/00089076
Código do Cliente: x25 – O Rei das Oficinas Endereço do Cliente: xxxxxxxxxxxxxxxxxxxxxxxxxxxx Endereço para Entrega do Pedido: xxxxxxxxxxxxxxxx	CGC: 99.888.888/0001-07 Insc. Est.: 078/9090099	Pedido Nº **1026**

Descriminação do Pedido

Código Item	Qtde Pedida	Descrição do Item	Preço Unitário	Preço Total	%IPI	Valor IPI
1740	10	parafuso modelo 1B	1,00	10,00	10	1,00
2770	20	parafuso modelo 11C	2,00	40,00	10	4,00
2718	10	roela para parafuso	1,00	10,00	0	0,00

Data do Pedido: 24/03/98	Data de Entrega: 30/03/98		Total do Pedido R$ 60,00		Total do IPI R$ 5,00
Vendedor: 055 – João Carlos	Observações: 01 volume	***Obrigado pela Preferência***	Forma de Pagamento		
Transporte: Rodoviário			1 2	24/03/98 24/04/98	R$ 30,00 R$ 30,00
Transportadora: 001-Transportes Silva					

A normalização pode ser utilizada de duas formas:

- **sentido de cima para baixo (top-down)**: após a definição de um modelo de dados, aplica-se a normalização para se obter uma síntese dos dados, bem como uma decomposição das entidades e relacionamentos em elementos mais estáveis, tendo em vista sua implementação física em um banco de dados;
- **sentido de baixo para cima (bottom-up)**: aplicar a normalização como ferramenta de projeto do modelo de dados, usando os relatórios, formulários e documentos utilizados pela realidade em estudo, constituindo-se em uma ferramenta de levantamento.

Observando-se o formulário de Pedido da empresa Comercial de Peças Sai da Frente Ltda, podemos considerar uma tabela com a seguinte apresentação♣:

- <u>número do pedido</u>
- código do cliente
- nome do cliente
- endereço faturamento
- endereço entrega
- cgc cliente
- insc cliente
- código do item
- quantidade do pedido
- descrição do item
- preço unitário do item
- %IPI do item
- data do pedido
- data de entrega
- código do vendedor
- nome do vendedor
- tipo de transporte
- código da transportadora
- nome da transportadora
- observações do pedido
- número da parcela
- data vencimento da parcela
- valor da parcela

A tabela acima constitui uma tabela não-normalizada, cuja chave primária é número do pedido (em sublinhado). A utilização de uma tabela única para representar este pedido, em um banco de dados, traria alguns problemas: um cliente só pode ser incluído se estiver vinculado a um pedido, se um produto tiver seu preço unitário alterado, será preciso percorrer toda a entidade para realizar múltiplas alterações. Em vista disso, a utilização da normalização permite organizar as informações das tabelas de uma forma simples, relacional e estável evitando perda e repetição da informação e oferecendo, ainda, uma representação adequada para o que se deseja armazenar.

Para normalizar uma tabela, são utilizadas três regras básicas, chamadas de primeira, segunda e terceira formas normais, respectivamente, 1FN, 2FN e 3FN.

***1FN**: "cada ocorrência da chave primária deve corresponder a uma e somente uma informação de cada atributo, ou seja, a entidade não deve conter grupos repetidos (multivalorados)".*

Então, observando o formulário, temos para a tabela acima:

♣ Os campos preço total, valor ipi, total do pedido, total do IPI não são apresentados, pois são campos calculados a partir dos valores outros campos, não precisando, portanto, ser armazenados no banco de dados

- número do pedido
- código do cliente
- nome do cliente
- endereço faturamento
- endereço entrega
- cgc cliente
- insc cliente
* *código do item*
* *quantidade do pedido*
* *descrição do item*
* *preço unitário do item*
* *%IPI do item*
- data do pedido
- data de entrega
- código do vendedor
- nome do vendedor
- tipo de transporte
- código da transportadora
- nome da transportadora
- observações do pedido
* *número da parcela*
* *data vencimento da parcela*
* *valor da parcela*

Assim, são criadas tantas tabelas quantos forem os grupos de elementos repetidos:

1FN:

Pedido	Item-Pedido	FormaPagto
número do pedido	número do pedido	número do pedido
código do cliente	código do item	número da parcela
nome do cliente	quantidade do pedido	data vencimento da parcela
endereço faturamento	descrição do item	valor da parcela
endereço entrega	preço unitário do item	
cgc cliente	%IPI do item	
insc cliente		
data do pedido		
data de entrega		
código do vendedor		
nome do vendedor		
tipo de transporte		
código da transportadora		
nome da transportadora		
observações do pedido		

⇒ Identificação das chaves:

a) tomar a chave primária da tabela embutida original
b) para a chave primária da tabela embutida, fazer a seguinte pergunta:
/ Um valor da chave primária aparece associado a exatamente um ou a muitos valores da chave primária da tabela externa?

- Um - a chave primária da tabela externa não faz parte da chave primária da tabela na PFN.
- Muitos - a chave primária da tabela externa faz parte da chave primária da tabela na PFN.

A 2FN e a 3FN dependem de um outro conceito: dependência funcional.

A dependência funcional é definida da seguinte forma: “dada uma entidade qualquer, um atributo ou conjunto de atributos A é dependente funcional de um outro atributo B contido na mesma entidade, se a cada valor de B existir nas linhas da entidade em que aparece, um único valor de A.

A dependência funcional pode ocorrer de duas formas:

- ***total***: na ocorrência de uma chave primária concatenada, um atributo ou conjunto de atributos depende de forma completa ou total desta chave primária concatenada, se e somente se, a cada valor da chave (e não a parte dela), está associado um valor para cada atributo;
- ***transitiva***: quando um atributo ou conjunto de atributos A depende de outro atributo B que não pertence à chave primária, mas é dependente funcional desta.

Assim, para a 2FN, deve-se analisar a dependência total da chave:

Passos para passagem à 2FN:

a) Copiar para a SFN cada tabela que tenha chave primária simples ou que não tenha atributos não chave;
b) Para tabelas com chave composta e atributos não chave:
b.1) Criar na SFN uma tabela com as chaves primárias da tabela na PFN
b.2) Para cada atributo não chave fazer a seguinte pergunta: O atributo depende de toda chave?
Sim - Copiar o atributo para a SFN.
Não - Criar, caso não exista, uma tabela na SFN que contenha como chave primária a parte da chave à qual o atributo pertence. Copiar o atributo dependente para a tabela criada.

2FN:

A tabela Pedido permanece como na 1FN (não tem chave concatenada);
A tabela Item-Pedido possui três campos que dependem apenas de parte da chave, neste caso, só do código do item, então gera-se uma outra tabela chamada Item.
A tabela FormaPagto permanece como na 1FN, pois os atributos dependem totalmente da chave, ou seja, do número do pedido e do número da parcela.

Item-Pedido	FormaPagto
número do pedido	número do pedido
código do item	número da parcela
quantidade do pedido	data vencimento da parcela

* descrição do item * preço unitário do item * %IPI do item	valor da parcela

Resultado da aplicação da 2FN:

Pedido	Item-Pedido	Item	FormaPagto
Idem 1FN	número do pedido código do item quantidade do pedido	código do item descrição do item preço unitário do item %IPI do item	idem 1FN

Para a 3FN, deve-se analisar a dependência transitiva da chave:

Pedido
número do pedido código do cliente * nome do cliente * endereço faturamento endereço entrega * cgc cliente * insc cliente data do pedido data de entrega código do vendedor * nome do vendedor tipo de transporte código da transportadora * nome da transportadora observações do pedido

Passos para passagem à 3FN:

a) Copiar para a TFN cada tabela que tenha zero ou um atributo não chave.

b) Para tabelas com mais de um atributo não chave:

b.1) Criar uma tabela na TFN com chave primária em questão;

b2.) Para cada atributo não chave fazer a seguinte pergunta: O atributo depende de algum outro atributo não chave (dependência transitiva ou indireta)?

Não - Copiar o atributo para a tabela na TFN.

Sim - Executar três passos:

1. Criar, caso ainda não exista, uma tabela na TFN que contenha como chave primária o atributo do qual há dependência indireta.
2. Copiar o atributo dependente para a tabela criada.

3. O atributo do qual há dependência deve permanecer também na tabela criada no passo b1.

Então, temos, na 3FN:

Pedido	Cliente	Vendedor
<u>número do pedido</u> código do cliente endereço entrega data do pedido data de entrega código do vendedor tipo de transporte observações do pedido código da transportadora	<u>código do cliente</u> nome do cliente endereço faturamento cgc cliente insc cliente	<u>código do vendedor</u> nome do vendedor
		Transportadora
		<u>código da transportadora</u> nome da transportadora

Item-Pedido	Item	FormaPagto
idem 2FN	idem 2FN	idem 1FN

5. Linguagens de consulta

Uma linguagem de consulta é uma linguagem na qual um usuário requisita informações do banco de dados. Estas linguagens são tipicamente de mais alto nível do que as linguagens de programação padrão. As linguagens de consulta podem ser classificadas como:

- procedurais: o usuário instrui o sistema para executar uma seqüência de operações no banco de dados para chegar ao resultado desejado;
- não-procedurais: nesse tipo de linguagem, o usuário descreve a informação desejada sem fornecer um procedimento específico para obter tal informação.

A maioria dos SBD relacionais comerciais oferece uma linguagem que inclui elementos procedurais e não-procedurais.

5.1 Linguagem SQL (Structured Query Language)

Relações:

Consulta(cod_med, cod_pac, data_hora, setor, sala)
Medico(cod_med, cremers, nome, idade)
Especialidade(cod_espec, espec)
Med_Espec (cod_med, cod_espec)
Paciente (cod_pac, nome, idade, endereco, prontuario)
Funcionario(cod_func, nome, idade, salario, setor)

DDL

1. Definição de uma relação ou tabela

- Comando **Create Table**

ex:

```
CREATE TABLE Medico
    (cod_med NUMBER   CONSTRAINT pkcod_med PRIMARY KEY,
     cremers VARCHAR2(10),
     nome VARCHAR2(20),
     idade NUMBER)
/

CREATE TABLE Funcionario
    ( cod_func NUMBER
     CONSTRAINT pkcod_func PRIMARY KEY
     CONSTRAINT checkcod_func CHECK (cod_func BETWEEN 1 AND 999),
     nome VARCHAR2(20),
     salario NUMBER(7,2),
     idade NUMBER,
     setor VARCHAR2(1) CONSTRAINT checksetor CHECK (setor IN ('A', 'B', 'C', 'D')) )
/
```

```
CREATE TABLE Especialidade
        (cod_espec NUMBER
        CONSTRAINT pkEspec PRIMARY KEY,
         espec VARCHAR2(30) )
/

CREATE TABLE Med_Espec
        (cod_med NUMBER,
        cod_espec NUMBER,
        CONSTRAINT pkMesEspec PRIMARY KEY(cod_med, cod_espec));
/

ALTER TABLE Med_Espec add constraint fkcod_med Foreign Key(cod_med) references Medico
(cod_med)
/

ALTER TABLE Med_Espec add constraint fkcod_espec Foreign Key(cod_espec) references
Especialidade (cod_espec)
/
```

2. Remoção de uma relação

- Comando DROP TABLE

ex: drop table Medico

DML

*** Comandos de Atualização do BD**

1. Inserção de tuplas

- comando: insert into <nome_tabela><lista de atributos> values <campos>

ex: **insert into** Especialidade(cod_espec,espec) **values** (100, 'Pediatria');

2. Alteração de tuplas

- comando: **update** <nome_tabela> **set** <alteração>
  [**where** <condição>]

ex: Alterar o setor do funcionário Gláucio para A

```
update Funcionario
set setor = 'A'
where nome = 'Gláucio'
```

3. Remoção de tuplas

- comando: **delete from** <nome_tabela>
  [**where** <condição>]

ex: Remover o médico Sérgio

```
delete from Médico
where nome = 'Sérgio'
```

*** Comandos de Consulta**

Estrutura Básica

Select <lista_atributos>
From <lista_relações>
[Where <condição>]

select a1, a2, ..., an
from r1, r2, ..., rn ⇒ πa1, a2,....an(σc (r1 X r2 X....X rn)
where c

ex: Buscar todos os dados dos médicos cadastrados

```
select cod_med, cremers, nome, idade
from medico
OU
select *
from Medico
```

1. Cláusula SELECT

a) DISTINCT - elimina duplicatas no resultado da consulta

ex: buscar todas os códigos dos médicos que tem consulta marcada com o paciente de código 100

```
select distinct cod_med
from consulta
where cod_pac = 100
```

b) Retorno de valores calculados

ex: busca o código e o salário de todos os empregados com um aumento de 20%

```
select cod, salario*1.2
from funcionarios
```

c) Funções de Agregação - operam sobre um conjunto de tuplas

Úteis na determinação de alguns cálculos matemáticos na consulta.

Existem cinco funções de agregação:

- Count: contador de ocorrências. Não conta valores nulos.

ex: total de tuplas da tabela de pacientes.

```
select count(*)
from paciente
```

- Sum: Somador de valores de atributos numéricos

ex: soma dos salários dos funcionários do setor B.

```
select sum(salario)
from funcionario
where setor='B'
```

- Avg: Média de valores de atributos numéricos
ex:média de idade dos médicos
select avg(idade)
from medico

- Max e Min: Maior/Menor valores de um atributo
ex: maior e menor salários pagos aos funcionários
select max(salario), min(salario)
from funcionario

2. Cláusula WHERE

a) Cláusula [NOT] LIKE
* Permite a definição de padrões de busca
* Padões possíveis
- LIKE ‘c%’ - o valor do atributo inicia com o caracter ou string c.
- LIKE ‘%c’ - o valor do atributo termina com o caracter ou string c.
- LIKE ‘%c%’ - o valor do atributo possui o caracter o string c no meio da cadeia.

ex: Buscar o nome de todos pacientes que começam com a letra C

```
select nome
from paciente
where nome like ‘C%’
```

b) Cláusula IS [NOT] NULL
* Permite o teste sobre valores nulos de atributos.

ex: Buscar os os dados dos funcionários que não trabalham em nenhum dos setores do hospital.

```
select *
from funcionario
where setor is null
```

c) Cláusula UNION
* Permite a união de duas tabelas compatíveis.

ex: Buscar o nome de todas os médicos e pacientes

```
select nome
from medico
union
select nome
from paciente
```

3. Consultas envolvendo mais de uma tabela
* FROM <lista_tabelas> - produto catersiano implícito

ex: Buscar o nome dos pacientes com consulta marcada na sala 124.

```
select nome
from paciente, consulta
```

```
where paciente.codp = consulta.codp and consulta.sala = 124
OU
select nome
from paciente P, consulta C
where P.cod_pac = C.cod_pac and C.sala = 124
```

3.1 Cláusulas para tratamento de subconsultas: IN, ANY, ALL, EXISTS

a) Cláusula [NOT] IN

* elemento ∈ conjunto
* sintaxe: where <atributo ou expressão> in (<subconsulta>)

ex: Nome de todos os paciente com consulta marcada com o médico de código 1001

```
select nomep
from paciente
where codp in (select codp
                from consulta
                where cod_med = 1001)
```

b) Cláusula ANY

* Permite outras formas de comparação elemento conjunto.

= any (<subconsulta>) - in

> any (<subconsulta>) - verdadeiro se o atributo comparado for maior do que algum valor de atributo das tuplas resultantes da subconsulta.

< any (<subconsulta>)

< > any (<subconsulta>)

ex: Buscar o nome de todos os funcionários exceto o mais idoso.

```
select nome
from funcionario
where idade < any (select idade
                   from funcionario)
```

c) Cláusula ALL

* Uma condição deve ser satisfeita para todos os elementos do conjunto.

= all (<subconsulta>) - igual a todos

> all (<subconsulta>) - maior que todos

< all (<subconsulta>) - menor que todos

< > all (<subconsulta>) - diferente de todos (not in)

ex: Buscar o nome dos funcionários com salário maior que o maior salário paga aos funcionários do setor B.

```
select nome
from funcionario
where salario > all (select salario
                     from funcinario
                     where setor = 'B')
```

d) Cláusula [NOT] EXISTS

* exists (<subconsulta>) - é verdadeiro se a tabela resultante da subconsulta não for vazia.

ex:Buscar o nome de todos médicos com consulta marcada.

```
select nome
from medico
where exists (select *
                from consulta
                where cod_med = consulta.cod_med)
```

⇒ Cláusula ORDER BY

- Permite a ordenação do resultado da consulta
- Utilizada após a cláusula Where.
- sintaxe: order by <lista_atributos>

ex: Buscar os dados de todos os funcionarios, ordenados pelo nome

```
select *
from funcionario
order by nome
```

ex: Buscar os dados de todas as consultas da paciente Silvana, ordenadas de forma decrescente pela hora da consulta.

```
select *
from consulta
where cod_pac in (select cod_pac
                    from paciente
                    where nome = 'Silvana')
order by hora desc
```

⇒ Cláusula GROUP BY

- group by - agrupa partes do resultado de uma consulta, a partir do qual é possível utilizar funções de agregação

- having - especifica condições para a formação de um grupo. Só existe associado à cláusula group by. As condições só podem envolver os atributos a serem buscados ou alguma função de agregação.

ex: Buscar o código dos médicos e o total de consultas para cada médico

```
select cod_med, count(*)
from consulta
group by cod_med
```

ex: Buscar os códigos dos médicos com mais de 5 consultas marcadas

```
select cod_med, count(data)
from consulta
group by cod_med
having count(*) > 5
```

⇒ Comando de atualização + Query Language

* Expressões de consulta podem estar associados a comandos de atualização de dados, para melhor restringir o unirso de tuplas a serem atualizadas.

ex: Remover todas as consultas do médico Ricardo da Silva

```
delete from consulta
where cod_med in (select cod_med
                  from medico
                  where nome = 'Ricardo da Silva')
```

5.2 Álgebra Relacional

A álgebra relacional é uma linguagem de consulta procedural. Possui as seguintes características:

- Descreve qualquer operação de consulta sobre relações;
- Linguagem orientada à manipulação de relações e não de registros;
- O resultado de uma consulta sobre uma ou mais relações gera uma relação;
- É base para o desenvolvimento de DMLs de mais alto nível

A álgebra relacional possui um conjunto de operações básicas. Essas serão explicadas utilizando as seguintes relações exemplos:

Agencia(codag, nomeag, cidadeag)

codag	nomeag	cidadeag
100	X	POA
450	Y	Gravataí
102	Z	POA

Cliente(codc, nomec, idadec)

codc	nomec	idadec
200	Pedro	30
201	Ana	28
202	Paulo	46

Conta(codag, numconta, codc, saldo)

codag	numconta	codc	saldo
102	389	202	2500
450	678	202	3700
100	987	201	6000

Emprestimo(codag, numemp, codc, quantia)

codag	numemp	codc	saldo
450	560	202	2500
450	561	200	3700

- Operações Fundamentais

A) SELEÇÃO

- Seleciona tuplas que satisfazem uma dada condição (predicado);
- Produz um subconjunto horizontal de uma relação;
- Notação:

$\sigma_{<predicado>}$ (<relação>)

- <predicado> permite o uso dos seguintes operadores de comparação: =, ≠, <, <=, >, >=

ex: Obter informações sobre todas as agencias de POA.

$\sigma_{cidadeag = 'POA'}$ (Agencia)

resultado:

100 X POA
102 Z POA

B) PROJEÇÃO

- Seleciona atributos de interesse
- Produz um subconjunto vertical de uma relação
- Notação:

$\pi_{<Lista\_atributos>}$ (<relação>)

ex: Obter os códigos das agências de POA.

π_{codag} ($\sigma_{cidadeag = 'POA'}$ (Agencia))

resultado:

100 102

C) PRODUTO CATESIANO

- Combinação de todas as tuplas de duas relações
- Utilizado quando necessita-se obter dados presentes em duas ou mais relações.
- Notação:

<relação1> X <relação2>

ex1: Obter o nome de todos os clientes que tem conta na agencia de código 450

* Expressão não otimizada

$\pi_{nomec}(\sigma_{codag = 450 \wedge Cliente.codc = Conta.codc}(Cliente \times Conta))$

etapas da executação:

❖ **relação resultante do produto cartesiano**

Cliente Conta

codc	nomec	idadec	codag	numconta	codc	saldo
200	Pedro	30	102	389	202	2500
200	Pedro	30	450	678	202	3700
200	Pedro	30	100	987	201	6000
201	Ana	28	102	389	202	2500
201	Ana	28	450	678	202	3700
201	Ana	28	100	987	201	6000
202	Paulo	46	102	389	202	2500
202	Paulo	46	450	678	202	3700
202	Paulo	46	100	987	201	6000

❖ **resultado da seleção aplicada a relação anterior**

Cliente Conta

codc	Nomec	idadec	codag	numconta	codc	saldo
202	Paulo	46	450	678	202	3700

❖ **resultado da consulta**

Paulo

<u>Expressão Otimizada da consulta anterior</u>

$\pi_{nomec}(\sigma_{Cliente.codc = Conta.codc}(\pi_{codc,nomec}(Cliente) \times \pi_{codc}(\sigma_{codag = 450}(Conta))))$

etapas da execução:

❖ **relação resultante da seleção e projeção sobre a relação Conta.**

codc
202

❖ **relação resultante da projeção sobre a relação Cliente**

codc	nomec
200	Pedro
201	Ana

202	Paulo

❖ **relação resultante do produto cartesiano**

Cliente Conta

codc	nomec	codc
200	Pedro	202
201	Ana	202
202	Paulo	202

❖ **resultado da consulta**

Paulo

D) UNIÃO

- Une as tuplas de duas relações que sejam compatíveis
- Notação:

<relação 1 > ∪ <relação2>

obs: operadores matemáticos (união, diferença e interseção) aplicam-se a duas relações ditas compatíveis, ou seja:

- relações com o mesmo grau (número de atributos)
- relações cujos domínios dos atributos são iguais, na mesma ordem de definição de colunas.

ex: Obter o código de todos os clientes da agencia 450

$\pi_{codc}(\sigma_{codag = 450}(\text{Conta})) \cup \pi_{codc}(\sigma_{codag = 450}(\text{Emprestimo}))$

resultado: 200 202

E) DIFERENÇA

- Retorna as tuplas de uma relação1 cujos valores não estão presentes em uma relação2.
- Notação:

<relação1> - <relação2>

ex: obter o código dos clientes que não fizeram empréstimos

$\pi_{codc}(\text{Cliente}) - \pi_{codc}(\text{Emprestimos})$

resultado: 201

F) INTERSECÇÃO

- Retorna as tuplas cujos valores de seus atributos sejam comuns às relações 1 e 2.

- Notação: <relação1> ∩ <relação2>
ex: obter o código de todos os clientes que possuem uma conta e um empréstimo

π_{codc} (Conta) ∩ π_{codc} (Emprestimos)
resultado: 202

G) JUNÇÃO NATURAL (JOIN)

- Combinação dos operadores produto cartesiano e seleção (retorna as tuplas de um produto cartesiano que satisfazem uma dada condição).
- Assume uma junção por um ou mais atributos comuns sem repetir este atributo na relação resultante.
- Notação: <relação1> [x] <relação2>
ex: Obter o nome de todos os clientes que tem conta na agencia de código 450

π_{nomec} ($\pi_{codc,nomec}$ (Cliente) [x] π_{codc} ($\sigma_{codag = 450}$ (Conta)))

❖ **relação resultante do Join**

codc	nomec
202	Paulo

❖ **resultado da consulta:**

Paulo

H) DIVISÃO

- Operadores de divisão:
Dividendo (relação R1 com grau m + n)
Divisor (relação R2 com grau n)
Quociente (relação resultante com grau m)
- Utilizada quando se deseja extrair de uma relação R1 uma determinada parte que possui as características (valores de atributos) da relação R2.
- Os atributos de grau n devem possuir o mesmo domínio.
- Notação: <relação1> ÷ <relação2>

ex: Obter o nome de todos os cliente que tem conta em todas as agencia de POA

$\pi_{nomec, codag}$ (Cliente [x] Conta) ÷ π_{codag} ($\sigma_{nomeag = 'POA'}$ (Agencia))

❖ **relação resultante do dividendo**

nomec	codag
Paulo	102
Paulo	450
Ana	100

- **relação resultante do divisor**

codag
100
102

- **resultado da consulta : vazio (nenhuma tupla no resultado)**

6.Visões e Segurança de Acesso

6.1 Visões

* Relações → tabelas físicas armazenadas no BD
* Visões→ relações virtuais derivadas das relações do BD

*Uso de visões:

a) Segurança de acesso (Autorização)
b) Criação de visões com as consultas mais comuns que envolvem muitas tabelas

* Exemplos:

a) Um funcionario do hospital não deve ter acesso a todos os dados pessoais de um paciente, somente ao seu código e nome:
b) Pode ser interessante vincular os dados de um médico aos dados de suas consultas.

* Visões em SQL

a) Criação:

Create view<nome_visão >as
<expressão_consulta>

b) Remoção:

drop view,nome_visão>

A visão especifica é eliminada (isto é, a definição é removida) e todas as visões definidas em termos desta visão também são automaticamente anuladas.

-Exemplos:

a) **create view DadosPac as**
selec cod_pac, nome
from paciente

Operações realizadas sobre uma visão se refletem diretamente sobre as tabelas físicas das quais ela deriva.

- **Consultas**

a) o funcionário do hospital deseja buscar o nome de todos os pacientes cadastrados que começam com a letra R.

```
select nome
from DadosPac
where nome like 'R%'
```

b) buscar a data das consultas do médico Pedro

```
Select data_hora
From MedCons
Where nome = 'Pedro'
```

- Atualizações

* Operações de inserção, atualização e remoção de tuplas em uma visão muitas vezes não são permitidas

Situação 1: visão MedCons
Inserção de tuplas nesta visão:
Alguns problemas:
- violação da regra de integridade de entidade (não tenho o código do médico e do paciente)
- perde-se os relacionamentos entre médicos e consultas (algumas consultas não teriam resultados satisfatórios)

Situação 2: Visão que relaciona o total de consultas de um determinado paciente
- não podem suportar operações de insert, update e delete

```
Create view ConsPac (codp,totcons) as
        Select cod_pac, count(*)
        From consulta
        Group by cod_pac
```

6.2 Segurança de Acesso

Proteção quanto a acesso não autorizado.

Autorização de acesso no SQL:
a) Comando GRANT:

GRANT <lista_privilégios> ON <nome_relação/visão> TO <lista_usuarios>
Ex:
Autorização de leitura e atualização dos atributos da relação funcionario para o usuário Pedro.

Grant select, update on Funcionario to Pedro

a) Comando REVOKE:

REVOKE <lista_privilégios> ON <nome_relação/visão> FROM <lista_usuarios>
Ex:

Revoke update on Funcionario from Pedro

7. Anexo - Tipo de Dados DATE do Oracle

Para fazer comparação com data e hora no Oracle utilize as funções TO_DATE e TO_CHAR.

* Inserção de valores na tabela consulta

```
insert into consulta (cod_med, cod_pac, data_hora, sala, setor)
values(100,200,to_date('12/05/1999 14:30','dd/mm/yyyy hh24:mi'), 122, 8);
insert into consulta (cod_med, cod_pac, data_hora, sala, setor)
values(100,201,to_date('20/05/1999 16:00','dd/mm/yyyy hh24:mi'), 132, 8);
insert into consulta (cod_med, cod_pac, data_hora, sala, setor)
values(101,202,to_date('12/06/1999 15:30','dd/mm/yyyy hh24:mi'), 140, 6);
```

1. Recuperar as datas das consultas do médico de código 100

```
select to_char(data_hora,'dd/mm/yyyy') Data
from consulta
where cod_med = 100;
```

```
DATA
----------
12/05/1999
20/05/1999
```

2. Recuperar o codigo dos médicos e a respectiva data da consultas para as consultas realizadas até o dia 20 de maio de 1999 inclusive.

```
select cod_med, to_char(data_hora,’dd/mm/yyyy’) Data
from consulta
where data_hora < to_date ('21/05/1999’,’dd/mm/yyyy’);
```

```
COD_MED     DATA
--------------      ----------
     100          12/05/1999
     100          20/05/1999
```

3. Recuperar as datas das consultas a partir do dia 19 de maio de 1999.

```
select to_char(data_hora,’dd/mm/yyyy’) Data
from consulta
where to_char(data_hora,'yyyymmdd') >= '19990519';
```

```
DATA
```

```
----------
20/05/1999
12/06/1999
```

4. Recuperar o código dos médicos que tiveram alguma consulta entre os dias 10 e 24 de maio até às 15:00 hs.

```
select cod_med
from consulta
where data_hora between to_date (‘10/05/1999’,’dd/mm/yyyy’)
and to_date (‘24/05/1999’,’dd/mm/yyyy’)
and to_char(data_hora,’hh24:mi’) <= ’15:00’;

COD_MED
--------------
100
```

BIBLIOGRAFIA

- DATE, C. J. Introdução a Sistemas de Bancos de Dados. Rio de Janeiro: Campus, 2000. 7ª edição.
- ULLMAN, J. A First Course in Database Systems. Upper Saddle River: Prentice Hall, 2002. 2ª edição.
- ELMASRI, R. & NAVATHE, S.B. Fundamentals of database systems. Redwood City: The Benjamin/Cummings, 2003.4ª edição.
- HEUSER, Carlos Alberto. Projeto de Bancos de Dados. Ed. Sagra-Luzzatto, 2001.5ª edição.
- KORTH, Henry F. e SILBERSCHATZ, Abraham. Sistema de Bancos de Dados. São Paulo: Makron Books, 1999. 3ª edição revisada.CHEN, Peter. Gerenciando banco de dados: a abordagem entidade-relacionamento para projeto lógico. McGraw-Hill. 1990.
- KORTH, H.F.; SILBERSCHATZ, A.; SUDARSHAN, S. Database Systems Concepts. (3rd Edition) McGraw Hill, Inc., 1997.
- NAVATHE, S.B. et al. Conceptual Database Design. Redwood City: The Benjamin/Cummings, 1992.
- BOWMAN, J.; EMERSON, S.; DARNOVSKY, M. The Practical SQL Handbook (3rd Edition) Addison-Wesley, 1996.
- KORTH, Henry F. e SILBERSCHATZ, Abraham. Sistema de Bancos de Dados. São Paulo: Makron Books, 1995. 2ª edição revisada.