Faculdades Integradas de Bauru - FIB

Banco de Dados

Aula 1

MBA em Engenharia de Software com ênfase em aplicações WEB e Mobile

Prof. Me. Marco Aurelio M. Antunes

Marco Aurelio Migliorini Antunes

Formação:
Graduação: Análise de Sistemas – USC
Graduação: Matemática - FAEL
Pós Graduação: Recursos Humanos com Sistemas de Informação – FIB
Mestrado: Tv Digital – FAAC – UNESP

Atuação Profissional:
- Banco de Dados
- Desenvolvimento de Sistemas
- Desenvolvimento de Sites

Contato:

- prof_antunes@outlook.com
- https://www.linkedin.com/in/marco-aurelio-m-antunes-85826027/
- http://lattes.cnpq.br/0148514838383451

Bibliografia:

MANZANO, J. A. N. G., **Microsoft SQL Server 2014 Express. Guia Prático e Interativo**, 1ª ed. Erica, 2014, 224p.

GONCALVES, R.R., **T-SQL com Microsoft SQL Server 2012 Express na prática**, 1ª ed. Erica, 2013, 120p.

BAPTISTA, L. F., **Linguagem SQL - Guia Prático de Aprendizagem**, 1ª ed., Erica, 2011, 160p.

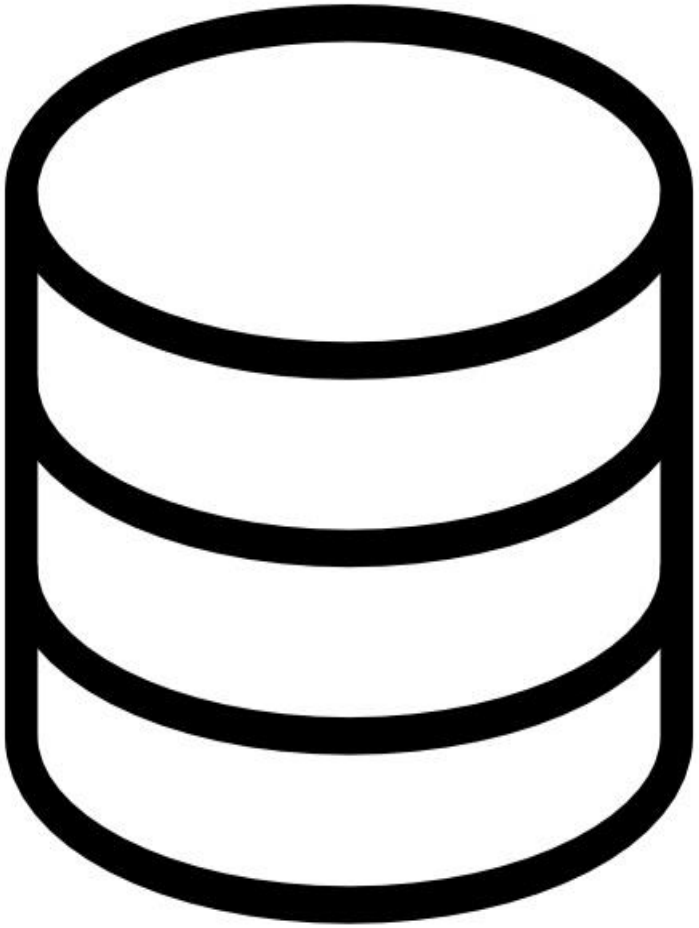
MANNINO, M. V., **Projeto, Desenvolvimento de Aplicações e Administração de Banco de Dados**, 3ª ed., McGraw-Hill, 2008, 702p.

SILBERSCHATZ, A., **Sistema de Banco de Dados**, 6ª ed., Campus, 2012, 808p.

SADALAGE, P.J., FOWLER, M., **NOSQL Essencial**, 1ª ed. Novatec, 2013, 216p.

MAYER-SCHONBERGER, V., CUKIER, K., **Big Data. Como Extrair Volume, Variedade, Velocidade e Valor da Avalanche de Informação Cotidiana**, 1ª ed., 2013, Campus, 163p.

BANCO DE DADOS
Conceitos Fundamentais



Engenharia de Software - Revisão de Conceitos

Banco de Dados

O que é Banco de Dados?

- **Coleção de dados inter-relacionados**, representando informações sobre um domínio específico, ou seja, sempre que for possível agrupar informações que se relacionam e tratam de um mesmo assunto.

- **Ferramenta para coletar e organizar informações**. Os bancos de dados podem armazenar informações sobre pessoas, empresas, produtos, pedidos ou qualquer outra coisa.

- Coleções de dados interligados entre si e organizados para **fornecer informações administrativas**, financeiras, estatísticas, comportamentais e uma infinidade de demandas.

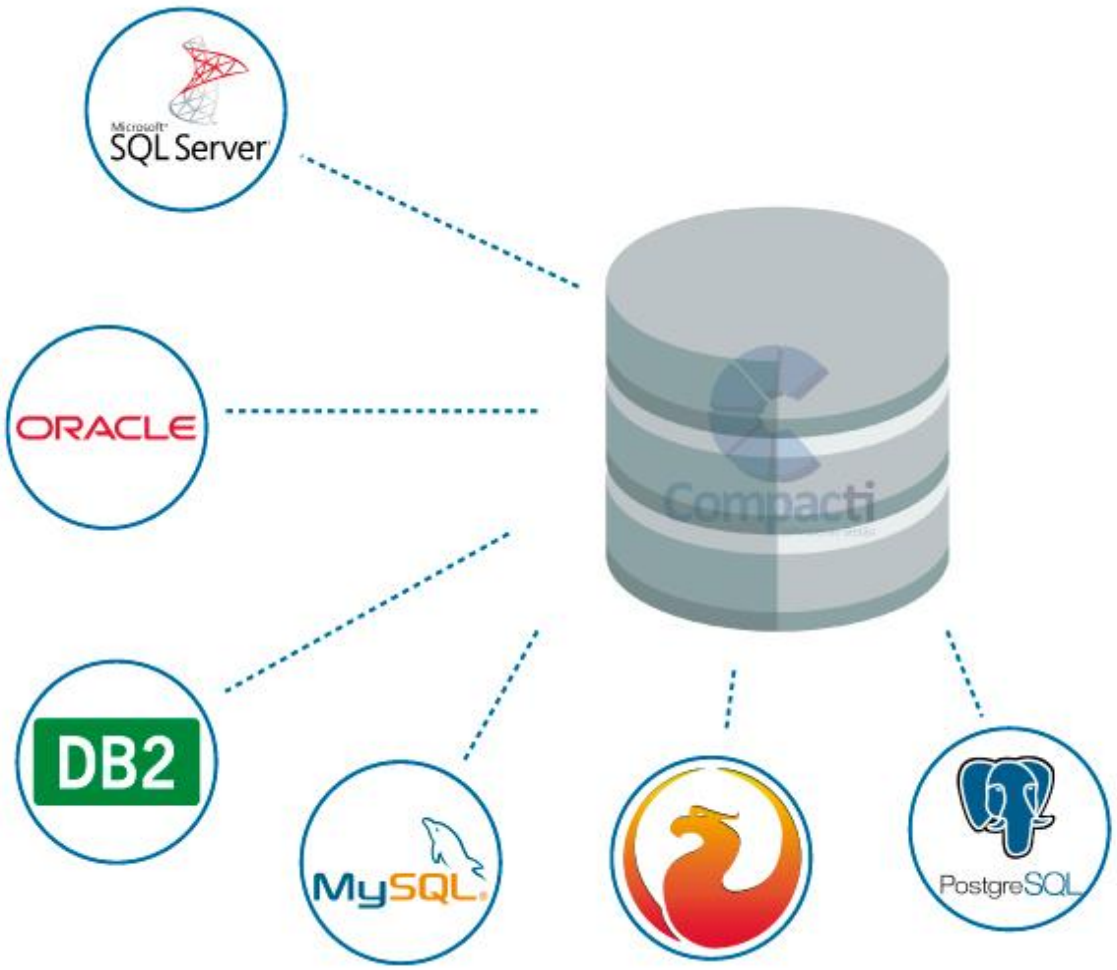
- **Armazenamento de dados referentes às operações de uma determinada empresa**, possibilitando a visualização de relatórios estratégicos, gráficos e/ou análises de grandes volumes de informações.

- **Possibilitam visões estratégicas** para várias iniciativas da empresa, como, por exemplo, uma política de promoções personalizada, ações de marketing mais bem segmentadas e até os projetos de expansão dos negócios em outras áreas, pois os bancos de dados, permitem identificar quem é cada cliente, quais são as suas preferências e o ímpeto de consumo.

Sistema Gerenciador de Banco de Dados - SGBD

Entende-se por **SGDB** como uma coleção de programas que permitem criar estruturas, manter dados, gerenciar transações efetuadas em tabelas, além de permitir a extração das informações de maneira rápida e segura

Recursos do SGBD:
– adição de novos arquivos;
– inserção de dados;
– recuperação de dados;
– atualização dos dados;
– eliminação dos dados;
– criação de visões;
– atribuição de privilégios;
– compartilhamento de dados;
– garantir consistência de dados;
– Segurança / Integridade;
– Controle de concorrência
– evitar redundância de dados.



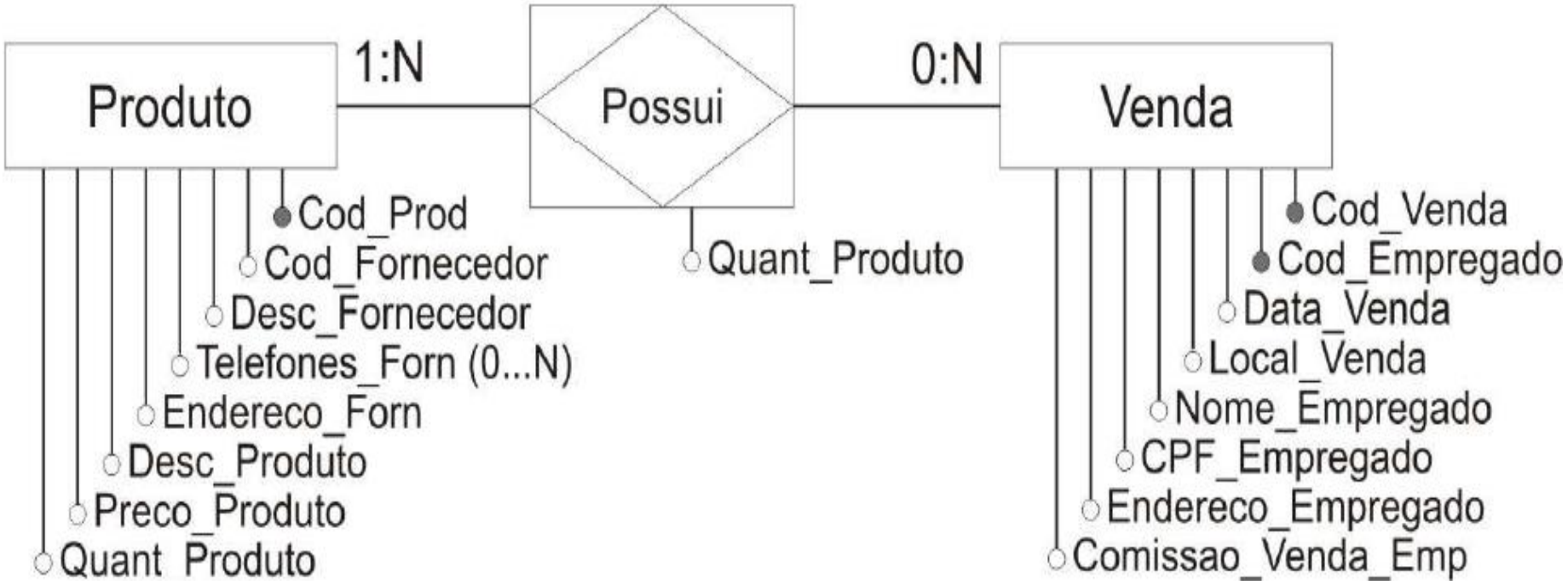

Microsoft
SQL Server


ORACLE
DATABASE


MySQL

Diagrama de Entidade e Relacionamento

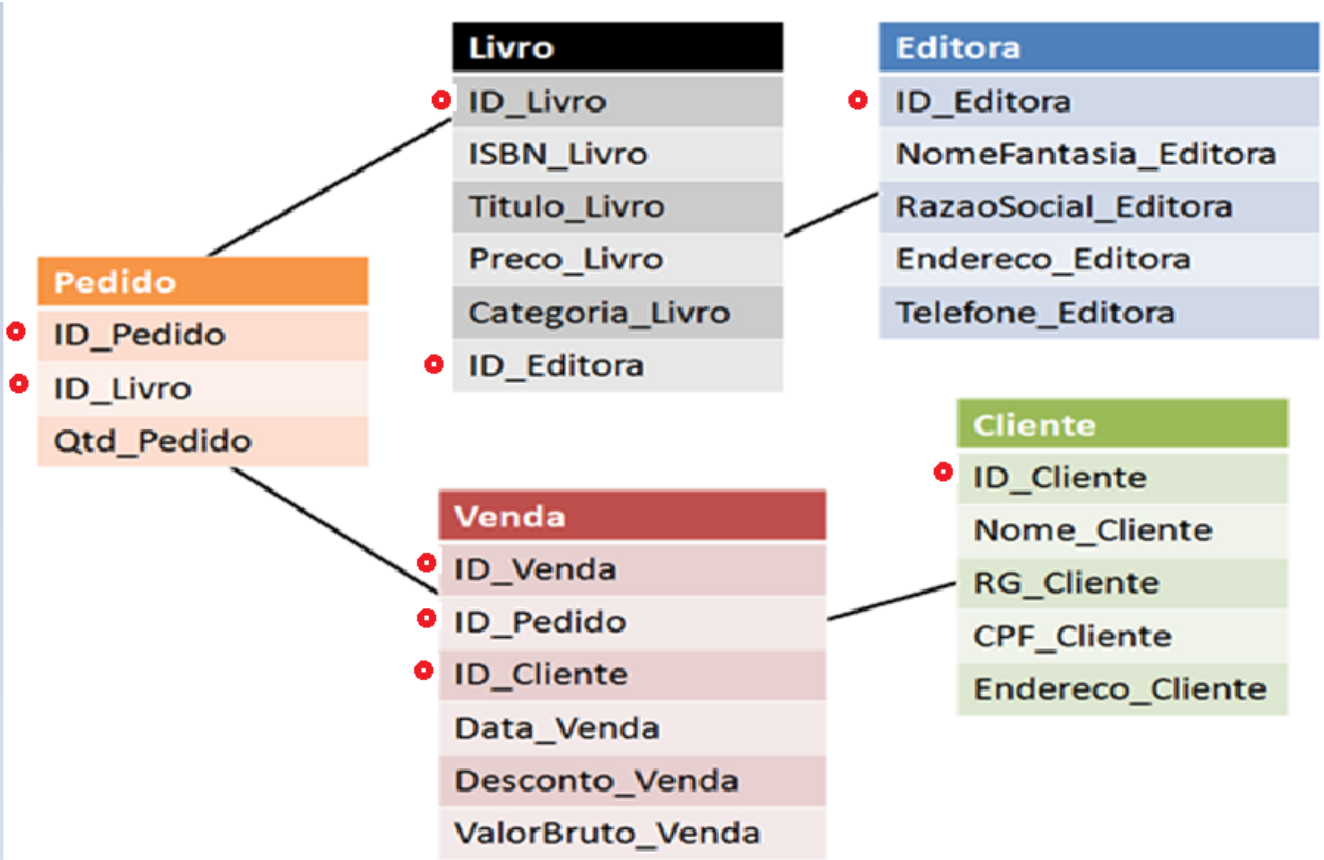
Através deste diagrama poderemos representar, de forma sucinta e bem estruturada, todos os elementos essenciais abstraídos no processo de análise de sistemas. Denominamos ***entidade*** (retângulo) estes elementos. Atribuímos a cada entidade definida ***atributos*** pertinentes ao sistema. Desta forma, podemos definir conceitualmente que representaremos como *entidades* aqueles elementos no qual gostaríamos de armazenar dados – que por sua vez, são representados pelos *atributos*.



Modelagem de Dados

O conceito de Modelo é importante no processo de desenvolvimento de sistemas de informação, pois são formas de ***REPRESENTAÇÃO.***

A **Modelagem de Dados** é a criação de um modelo físico que explique a lógica por trás do sistema, com ela você é capaz de representar a forma como os dados e processos são organizados, explicando as características de funcionamento




SQL

Criando Banco de Dados e Tabelas

Tipos de Dados

Alfanuméricos	Contém cifras, números e letras. Apresentam uma longitude limitada (255 caracteres)
Numéricos	Existem de vários tipos, principalmente, inteiros (sem decimais) e reais (com decimais).
Booleanos	Possuem duas formas: Verdadeiro e falso (Sim ou Não)
Datas	Armazenam datas facilitando posteriormente sua exploração. Armazenar datas desta forma possibilita ordenar os registros por datas ou calcular os dias entre uma data e outra, com sua consistência automática.
Memos	São campos alfanuméricos de longitude ilimitada. Apresentam o inconveniente de não poder ser indexados.
Auto-incrementáveis	São campos numéricos inteiros que incrementam em uma unidade seu valor para cada registro incorporado.

```
-- criar banco de dados
create database posfib;

-- abrindo banco de dados
use posfib;

-- criando tabelas // entidades
create table alunos(
id              int,
aluno           varchar(40),
telefone        varchar(20),
idade           int
);


-- apagando uma tabela
drop table alunos;
```

Integridade e Criação de Tabelas

Integridade e Criação de Tabelas

Integridade Referencial – Constraints

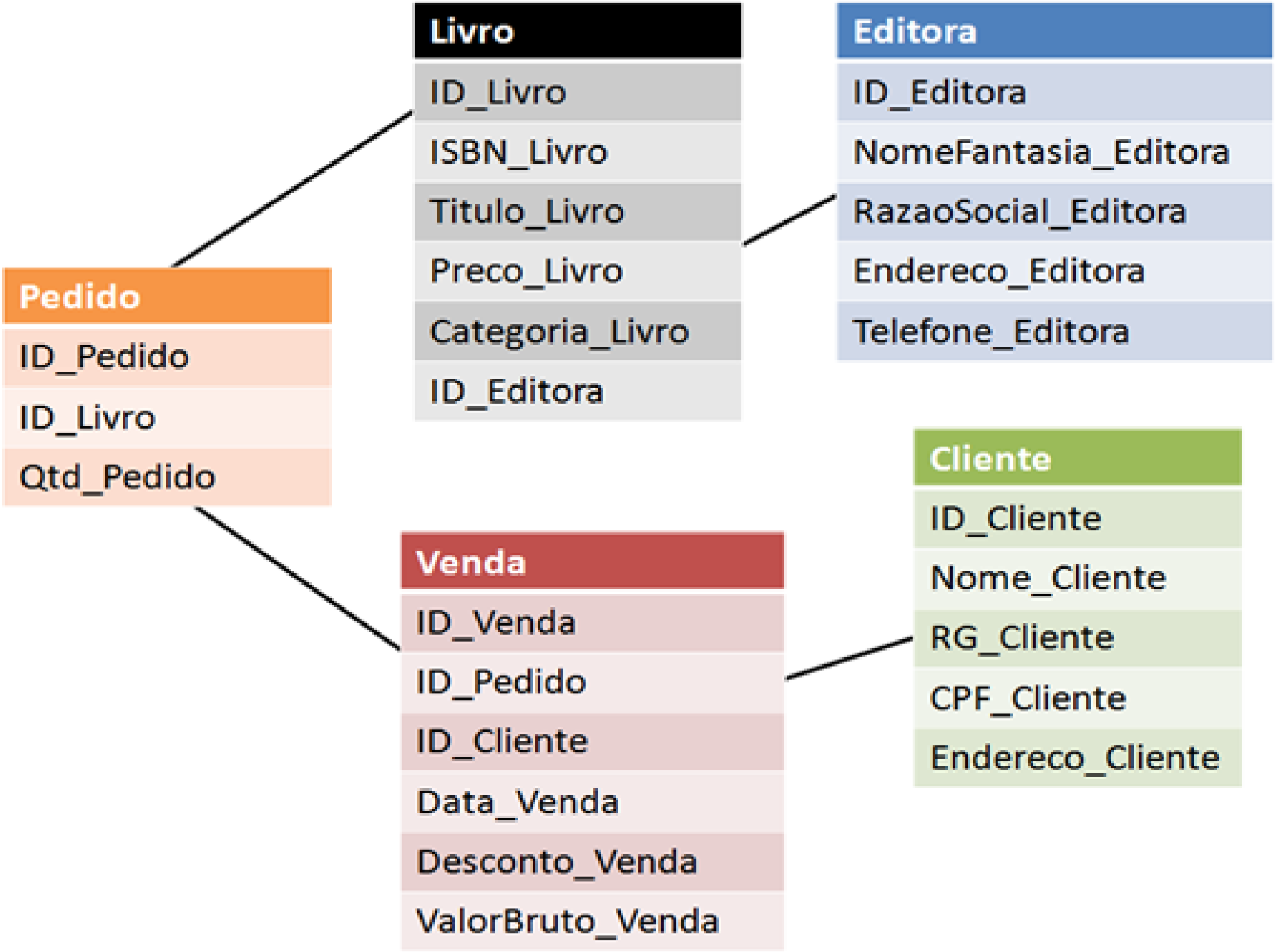
Constraints são regras agregadas a colunas ou tabelas. Assim, pode-se definir como obrigatório o preenchimento de uma coluna que tenha um valor-padrão quando uma linha for incluída na tabela ou quando aceitar apenas alguns valores preenchidos. No caso de regras aplicadas a tabelas, tem-se a definição de chaves primárias, estrangeiras e relacionamentos.

NOT NULL - Indica que o conteúdo de uma coluna não poderá ser nulo.

NULL - Indica que o conteúdo de uma coluna poderá ser nulo.

PRIMARY KEY - Chave primária é a coluna ou grupo de colunas, que permite identificar um único registro na tabela.

FOREIGN KEY - Chave estrangeira é o campo que estabelece relacionamento entre duas tabelas. Assim, uma coluna, ou grupo de colunas, de uma tabela corresponde à mesma coluna, ou grupo de colunas, que é a chave primária de outra tabela.


Livro
ID_Livro
ISBN_Livro
Titulo_Livro
Preco_Livro
Categoria_Livro
ID_Editora
Editora
ID_Editora
NomeFantasia_Editora
RazaoSocial_Editora
Endereco_Editora
Telefone_Editora
Pedido
ID_Pedido
ID_Livro
Qtd_Pedido
Venda
ID_Venda
ID_Pedido
ID_Cliente
Data_Venda
Desconto_Venda
ValorBruto_Venda
Cliente
ID_Cliente
Nome_Cliente
RG_Cliente
CPF_Cliente
Endereco_Cliente

```sql
-- criando tabelas utilizando constraints
create table alunos(
id          int          not null,
aluno       varchar(40) not null,
telefone    varchar(20) null,
idade       int          null
primary key(aluno)
);

create table professores(
codprof int          not null,
nome    varchar(40) not null,
curso   varchar(20) not null
primary key(codprof)
);

create table dependentes(
coddep  int          not null,
nome    varchar(40) not null,
codprof int          not null
primary key(coddep),
foreign key(codprof) references professores(codprof)
);
```

Estudo de Caso 1

clientes
codcli
nome
fone

vendedor
codven
nome
fone

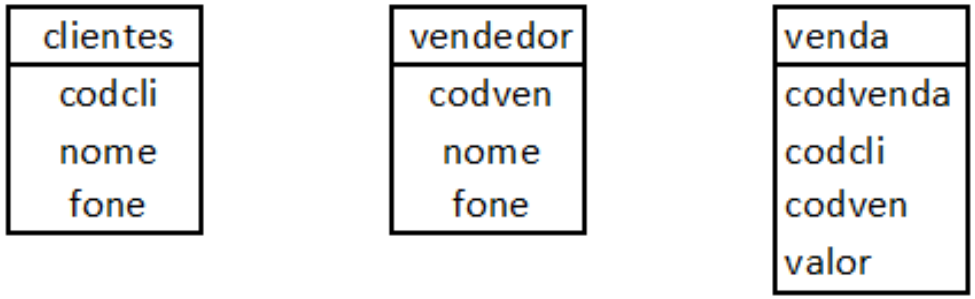
produtos
codprod
produto
preco
codfornec

fornecedor
codfor
empresa
fone
contato

pedido
codped
codprod

venda
codcli
codven
codped
valorvenda

Atividade 1



Criar o banco de dados LOJA

Criar as tabelas conforme diagrama acima observando a utilização de chave primária

Estabelecer relacionamento entre tabelas utilizando chave estrangeira

Nomes, tipos e tamanhos de campos ficam a critério do desenvolvedor

Incluindo Dados nas Tabelas

Incluindo dados em tabelas

Depois de criar e relacionar as tabelas para agilizar as pesquisas, é necessário inserir linhas(registros nas tabelas).

Sintaxe:

INSERT INTO nome_da_tabela (colunas)

values (conteúdos);

Exemplo:

```
-- criando tabela autor
create table autor
(
cod_autor          int              not null,
nome_autor         varchar(40)      not null,
fone_autor         varchar(20)      null,
primary key(cod_autor)
);
```

```
-- inserindo registros
insert into autor values (3,'Maria','3262-0000’);

-- inserindo registros
insert into autor (cod_autor,nome_autor,fone_autor) values (1,'José','3262-000’);

-- inserindo registros em alguns campos
insert into autor (cod_autor,nome_autor) values (2,'João');

-- inserindo registros em alguns campos
insert into autor (cod_autor,nome_autor,fone_autor) values (2,'João‘,NULL);
```

Obs. A lista de colunas é opcional no comando INSERT, pois caso você não especifique em quais colunas está incluindo valores, assume-se a inclusão de valores em **TODAS** as colunas e a ordem de inclusão (lista de conteúdos) corresponderá àquela da criação da tabela.

As colunas não atribuídas terão valor nulo e caso você queira incluir um valor nulo, você deve informar NULL no lugar correspondente a esta coluna(campo), mas isso caso não haja restrição para valores nulos (NOT NULL).

Importar Dados

Para incluir diversas linhas em uma tabela do banco de dados, utiliza-se o comando INSERT em conjunto com o comando SELECT. Mas nesse caso, apenas se copiam linhas de uma tabela para outra. Isso pode ser muito útil para criar tabelas temporárias ou importar dados de uma tabela.

Sintaxe:

```
INSERT INTO destino
SELECT * FROM origem;
```

Exemplo 1: Quando a lista de colunas do comando SELECT deve corresponder à totalidade das colunas na mesma sequëncia das colunas da tabela.

```
-- criando tabela autor temporaria
create table autor_tmp
(
cod_autor       int             not null,
nome_autor      varchar(40)     not null,
fone_autor      varchar(20)     null,
primary key(cod_autor)
);
 -- importando dados, com campos correspondentes
insert into autor_tmp
select * from autor;
```

Exemplo 2: Quando os campos não são correspondentes

```
-- criando tabela autor2
create table autor_temp2
(
odigo           int             not null,
nome            varchar(40)             not null,
telefone        varchar(20)     null,
email           varchar(18)     null,
primary key(codigo)
);

-- importando dados de campos não correspondentes
insert into autor_temp2(codigo,nome,telefone)
select cod_autor,nome_autor,fone_autor from autor;

-- verificando dados
select * from autor
select * from temp_autor2
```

Exemplo 3: Quando existes condições/restrições para importar dados

```
-- criando tabela autor3
create table autor_temp3
(
codigo     int              not null,
nome       varchar(40)      not null,
telefone   varchar(20)      null,
email      varchar(18)      null,
primary key(codigo)
);
 -- importando dados de campos não correspondentes
insert into autor_temp3(codigo,nome,telefone)
select cod_autor,nome_autor,fone_autor from autor
where cod_autor <=2;

-- verificando dados
select * from autor;
select * from autor_temp3;
```

Atualizando Dados das Tabelas

Para alterar o conteúdo de uma ou mais colunas, ou até o conteúdo de uma coluna em diversas linhas. Para isso utilizamos o comando UPDATE.

Sintaxe:

UPDATE nome_da_tabela

SET coluna=conteúdo

WHERE condição;

```
-- alterando dados
update autor_temp3
set telefone = '99898-7788'
where codigo = 2;

select * from autor_temp2;
update autor_temp2
set telefone = '3234-9898', email='pedro@fib.com.br'
where codigo = 5;
```

Excluindo Dados de Tabelas

Exclusão de dados em tabelas

O comando DELETE pode afetar uma ou mais linhas da tabela, excluindo os registros armazenados, de acordo com a **condição**.

sintaxe:

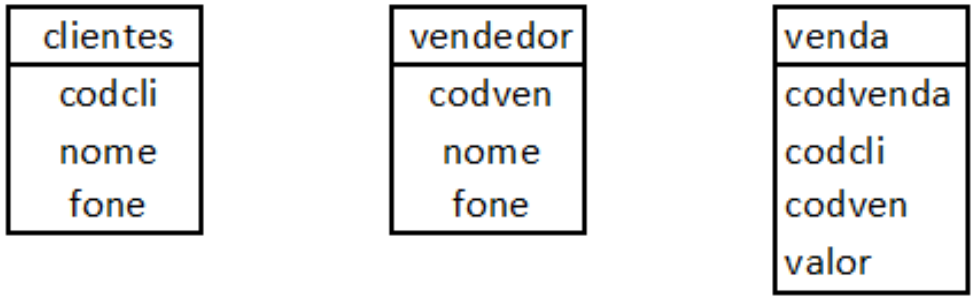
DELETE from tabela

WHERE condição;

Sintaxe: Apagando somente um registro

```
-- excluindo o primeiro registro
select * from autor_temp2;
delete from autor_temp2
where codigo = 3;
```

Atividade 2



Testar Inserir / Alterar / Excluir dados das tabelas criadas

Inserir 4 clientes

Inserir 4 vendedores

Inserir 4 vendas

Alterar o telefone de 1 cliente

Alterar o valor da venda para 100 de todos os clientes com código 1

Excluir 1 vendedor

Importar a tabela de vendas para uma tabela temporária

Importar a tabela de vendas para outra tabela temporária mas somente dos vendedores com código = 2 e valor da venda superior a 100

Pesquisa em Tabelas

```
SELECT *nome do(s) campo(s)
FROM nome da(s) tabela(s);
```

Exemplos:

```
-- consulta simples
select * from autor;

-- filtrar colunas
select cod_autor, nome_autor, fone_autor
from autor;

-- filtrando eliminando valores repetidos
select distinct fone_autor
from autor;
```

DISTINCT – não mostra valores repetidos de colunas

Ordenando Tabelas

Para determinar a ordem em que são mostrados os dados de uma tabela usamos a cláusula ORDER BY.

SELECT nome_do(s)_campo(s)

FROM nome_da_tabela

ORDER BY nome_da(s)_coluna(s) [asc,desc]

asc – a ordem da indexação é ascendente(opção padrão quando omitido)

desc – determina que a ordem de indexação é descendente.

Exemplos:

```
select * from produtos
order by produto;
```

```
select * from produtos
order by codfor,produto;
```

Filtrando Linhas

Para filtrar linhas em uma pesquisa, utilizamos a cláusula WHERE. Definimos uma expressão lógica(condição) que será avaliada e mostrará apenas as linhas que atenderem ao critério estabelecido.

Sintaxe:

SELECT [DISTINCT] [*] coluna

FROM nome

WHERE condição

ORDER BY nome_do(s)_campo(s)

Operadores Relacionais

Operadores relacionais definem um tipo de condição básica. Testando igualdade, diferença, maior, menor ou igual.

=	igual	codprod = 2
<	menor que	preco < 35
<=	menor ou igual a	preco <= 35
>	maior	preco > 35
>=	maior ou igual a	preco >= 35
<>	diferente	codprod <> 2

Exemplo:

```
-- testes com os operadores relacionais acima
select * from produtos
where condição
```

Operadores Lógicos

Operadores lógicos são usados quando somente uma condição não é suficiente para determinar o critério de busca.

AND - indica que TODAS as condições devem ser verdadeiras para que a linha seja mostrada.

```
-- AND - as 2 condições são verdadeiras
select *
from produtos
where preco > 10 and preco < 20;
```

OR - indica que uma das condições deve ser verdadeira (ou ambas), para que a linha seja mostrada.

```
-- OR - uma das condições é verdadeira
select *
from produtos
where preco > 20 or codfor= 1;
```

CUIDADO!! Não há limitação no uso e na combinação de condições utilizando operadores AND e OR. Mas é necessário atenção na combinação de ambos, pois a avaliação é feita da esquerda para a direita e resultados não exatamente esperados podem aparecer. É conveniente utilizar parênteses para determinar o que você quer comparar.

Exemplo: Combinação de AND e OR

```
-- sem parênteses
select *
from produtos
where codfor = 1 or codfor = 2 and preco > 20;

-- com parênteses
select *
from produtos
where (codfor = 1 or codfor = 2) and preco > 20;
```

Operadores Especiais

BETWEEN - Operador utilizado para determinar um intervalo de busca de forma simplificada.

```
select * from produtos
where preco between 6  and 8
order by preco;
```

LIKE - Operador compara cadeias de caracteres, utilizando padrões de comparação para um ou mais caracteres.

% substitui um ou mais caracteres – coringa

_ substitui um caracter

```
select * from produtos
where produto LIKE '%p%';
```

```
select * from produtos
where produto like '_a%';
```

Operadores Especiais

IN - Operador que compara o valor de uma coluna com um conjunto de valores.

```
select * from produtos
where codprod in(1,4);
```

Cálculos

```
select codprod, produto, preco, preco * 1.10 as 'preço com reajuste'
from produtos;
```

```
select codprod, produto, preco, preco * 1.10 'preço com reajuste'
from produtos;
```

Atividade 3

clientes
codcli
nome
fone

vendedor
codven
nome
fone

produtos
codprod
produto
preco
codfornec

fornecedor
codfor
empresa
fone
contato

pedido
codped
codprod

venda
codcli
codven
codped
valorvenda

Testar pesquisas / ordenação de tabelas / operadores

- Calcular reajuste de 15% para os produtos do fornecedor = 1

- **Elaborar um script que utilize o operador IN para a tabela de clientes**
- **Elaborar um script que utilize o operador LIKE para a tabela fornecedores**
- **Elaborar um script que utilize o operador BETWEEN para a tabela vendedor**
- **Mostrar ordenado por valor da venda as vendas com código do pedido entre 1 e 5 ou do vendedor = 3**

Funções de grupo e agrupamentos

Funções de grupo operam conjunto de linhas visando a fornecer um resultado para o grupo, que podem ser toda a tabela ou subgrupos da tabela.

FUNÇÃO	AÇÃO
COUNT	Retorna o número de linhas afetadas pelo comando.
SUM	Retorna o somatório do valor das colunas especificadas.
AVG	Retorna a média aritmética dos valores das colunas.
MIN	Retorna o menor valor da coluna de um grupo de linhas.
MAX	Retorna o maior valor da coluna de um grupo de linhas.
STDEV	**Retorna o desvio-padrão da coluna.**
VARP	**Retorna a variância da coluna.**

Funções de Grupo

COUNT - Retorna o número de linhas que atende determinada condição, podendo utilizá-la com (*) para indicar a quantidade total de linhas, independentemente de haver linhas nulas ou não.

```
select count(*)  from produtos;
```

```
select count(*)  from produtos
where preco < 7;
```

SUM - Retorna o valor total de uma determinada coluna em um determinado grupo de linhas.

```
select sum(preco)
from produtos;
```

```
select sum(preco)
from produtos
where codprod in(1,4);
```

Funções de Grupo

MIN - Retorna o menor valor de um grupo de linhas. Pode ser usado em colunas com valores numéricos, data ou alfanuméricos.

MAX - Retorna o menor valor de um grupo de linhas. Pode ser usado em colunas com valores numéricos, data ou alfanuméricos.

```
select min(preco) from produtos;
```

```
select min(preco)
from produtos
where codfor between 2 and 4;
```

```
select max(preco) from produtos;
```

```
select max(preco)
from produtos
where codfor between 2 and 3;
```

Atividade 4

clientes
codcli
nome
fone

vendedor
codven
nome
fone

produtos
codprod
produto
preco
codfornec

fornecedor
codfor
empresa
fone
contato

pedido
codped
codprod

venda
codcli
codven
codped
valorvenda

- Verifique o preço médio dos produtos
- Verifique a maior venda
- Verifique a menor venda do vendedor 1

Agrupamento e Relacionamento

GROUP BY - Agrupa um conjunto de linhas selecionadas em um conjunto de linhas de resumo pelos valores de uma ou mais colunas ou expressões no SQL Server.

Uma linha é retornada para cada grupo. As funções de agregação na lista de <seleção> da cláusula SELECT fornecem informações sobre cada grupo em vez de linhas individuais.

Ver modelos 1, 2 e 3

```sql
create database AULA;
use AULA;

create table grupo
(
cod_grupo  int           not null,
grupo      varchar(20)   not null,
primary key(cod_grupo)
);

insert into grupo values(1,'verduras');
insert into grupo values(2,'legumes');
insert into grupo values(3,'frutas');
insert into grupo values(4,'granja');

-- grupos cadastrados
select * from grupo;
```

```
create table produtos
(
cod_produto int          not null,
produto     varchar(30) not null,
preco       decimal(6,2)not null,
cod_grupo   int          not null,
primary key(cod_produto),
foreign key(cod_grupo) references grupo(cod_grupo)
);

insert into produtos values(1,'alface',2,1);
insert into produtos values(2,'abobrinha',1.90,2);
insert into produtos values(3,'morango',5.45,3);
insert into produtos values(4,'rucula',1.85,1);
insert into produtos values(5,'pepino',3.90,2);
insert into produtos values(6,'maça',2.25,3);
insert into produtos values(7,'uva',9.85,3);
insert into produtos values(8,'batata',1.25,2);
insert into produtos values(9,'repolho',0.95,1);
insert into produtos values(10,'cebola',1.25,2);
-- produtos cadastrados
select * from produtos;
```

```sql
-- grupos cadastrados utilizados
select distinct cod_grupo
from produtos;

-- quantidade de produtos em cada grupo
select cod_grupo,count(*)
from produtos
group by cod_grupo;

-- O preço médio de cada produto por grupo
select cod_grupo,avg(preco)
from produtos
group by cod_grupo;

-- grupos utilizados na tabela produtos
select distinct produtos.cod_grupo, grupo.grupo
from produtos,grupo
where produtos.cod_grupo = grupo.cod_grupo;
```

```sql
-- uniao de tabelas e ordenando o resultado por grupo
select produtos.codgrupo, grupo.grupo, produtos.produto
from produtos, grupo
where produtos.codgrupo = grupo.codgrupo
order by produtos.codgrupo;

-- uniao de tabelas e ordenando o resultado por grupo
-- classificando por grupo e ordem alfabética de produtos
select produtos.codgrupo, grupo.grupo, produtos.produto
from produtos, grupo
where produtos.codgrupo = grupo.codgrupo
order by grupo.grupo, produtos.codgrupo;

-- mesma sintaxe anterior
-- filtrando preços menores que 5
select grupo.grupo, produtos.produto, produtos.preco
from produtos, grupo
where (produtos.codgrupo = grupo.codgrupo) and produtos.preco < 5
order by grupo.grupo, produtos.codgrupo;

-- mesma sintaxe anterior
-- classificando preços em ordem decrescente
select grupo.grupo, produtos.produto, produtos.preco
from produtos, grupo
where (produtos.codgrupo = grupo.codgrupo) and produtos.preco < 5
order by grupo.grupo, produtos.preco desc;
```

```
--- outro exemplo
--- http://carros.uol.com.br/tabela-fipe/

create table fabricantes
(
cod_fabricante  int          not null,
fabricante      varchar(20)  not null,
primary key(cod_fabricante)
);

insert into fabricantes values(1,'fiat');
insert into fabricantes values(2,'ford');
insert into fabricantes values(3,'gm');
insert into fabricantes values(4,'honda');

--mostrar fabricantes
select * from fabricantes;
```

```sql
create table categoria
(
cod_categoria   int          not null,
categoria       varchar(20) not null,
primary key(cod_categoria)
);

insert into categoria values(1,'carro');
insert into categoria values(2,'moto');
insert into categoria values(3,'caminhão');

-- categorias cadastradas
select * from categoria;
```

```
create table veiculos
(
cod_veiculo     int          not null,
veiculo         varchar(30) not null,
preco           decimal(8,2)not null,
cor             varchar(15) not null,
ano             int          not null,
cod_fabricante  int          not null,
cod_categoria   int          not null,
primary key(cod_veiculo),
foreign key(cod_fabricante) references fabricantes(cod_fabricante),
foreign key(cod_categoria)  references categoria(cod_categoria),
);
```

```sql
insert into veiculos values(1,'fiesta sedan',32700,'prata',2014,2,1);
insert into veiculos values(2,'Ka 1.0',13755,'azul',2007,2,1);
insert into veiculos values(3,'Camaro v8',39565,'amarelo',1994,3,1);
insert into veiculos values(4,'Camaro Sport',45628,'amarelo',1994,3,1);
insert into veiculos values(5,'Camaro Z-28',62360,'amarelo',1994,3,1);
insert into veiculos values(6,'Uno Way',23274,'prata',2014,1,1);
insert into veiculos values(7,'Uno Mile',15195,'azul',2007,1,1);

-- veiculos cadastrados
select * from veiculos;
```

```sql
-- fabricantes cadastrados utilizados
select distinct cod_fabricante
from veiculos;

-- categorias cadastradas utilizadas
select distinct cod_categoria
from veiculos;

-- quantidade de veiculos de cada fabricante
select cod_fabricante,count(*)
from veiculos
group by cod_fabricante;

-- quantidade de veiculos de cada categoria
select cod_categoria,count(*)
from veiculos
group by cod_categoria;
```

```sql
-- quantidade de veiculos de cada categoria
select cod_categoria,count(*)
from veiculos
group by cod_categoria;

-- O preço médio de cada produto por
select cor,avg(preco)
from veiculos
group by cor;

-- o total de veiculos de cada ano e o maior preço de cada ano
select ano,count(*),max(preco)
from veiculos
group by ano;
```

```sql
-- grupos e subgrupos da tabela
select distinct veiculos.cod_categoria, categoria.categoria,fabricantes.fabricante
from veiculos,categoria,fabricantes
where veiculos.cod_categoria = categoria.cod_categoria and
veiculos.cod_fabricante = fabricantes.cod_fabricante;

-- união de tabelas
select veiculos.cod_categoria, categoria.categoria,fabricantes.fabricante
from veiculos,categoria,fabricantes
where veiculos.cod_categoria = categoria.cod_categoria and
veiculos.cod_fabricante = fabricantes.cod_fabricante
order by veiculos.cod_fabricante;
```

Cláusula HAVING com GROUP BY

A cláusula **HAVING** determina uma condição de busca para um grupo ou um conjunto de registros, definindo critérios para limitar os resultados obtidos a partir do agrupamento de registros. É importante lembrar que essa cláusula só pode ser usada em parceria com **GROUP BY**.

A cláusula **GROUP BY** pode ser empregada, entre outras finalidades, para agrupar os registros de acordo com cada **condição** existente. Dentro de cada um dos grupos, a cláusula **HAVING** pode ser usada para restringir apenas os registros que possuem uma determinada **condição**.

Obs: O **HAVING** é diferente do **WHERE**. O **WHERE** restringe os resultados obtidos **sempre** após o uso da cláusula **FROM**, ao passo que a cláusula **HAVING** filtra o retorno do agrupamento.

```sql
-- mostrar a quantidade em estoque por tipos com saldo maior que 200
select tipo, sum(quantidade) as 'Quantidade em Estoque'
from produtos
group by tipo
having sum(quantidade) > 200;

-- mostrar o valor médio dos preços por tipo
select tipo, avg(vlunitario) as 'Valor medio'
from produtos
group by tipo
having tipo = 'console';

-- mostrar a quantidade em estoque por tipos e fabricantes com saldo maior que 200
select tipo, fabricante, sum(quantidade) as 'Quantidade em Estoque'
from produtos
group by tipo, fabricante
having sum(quantidade) > 200
```

Atividade 5

Utilizar as tabelas do grupo 3

-- mostrar as vendas agrupadas por clientes

-- mostrar as vendas agrupadas por categoria

-- mostrar as vendas agrupadas por fabricante

-- mostrar as vendas por período

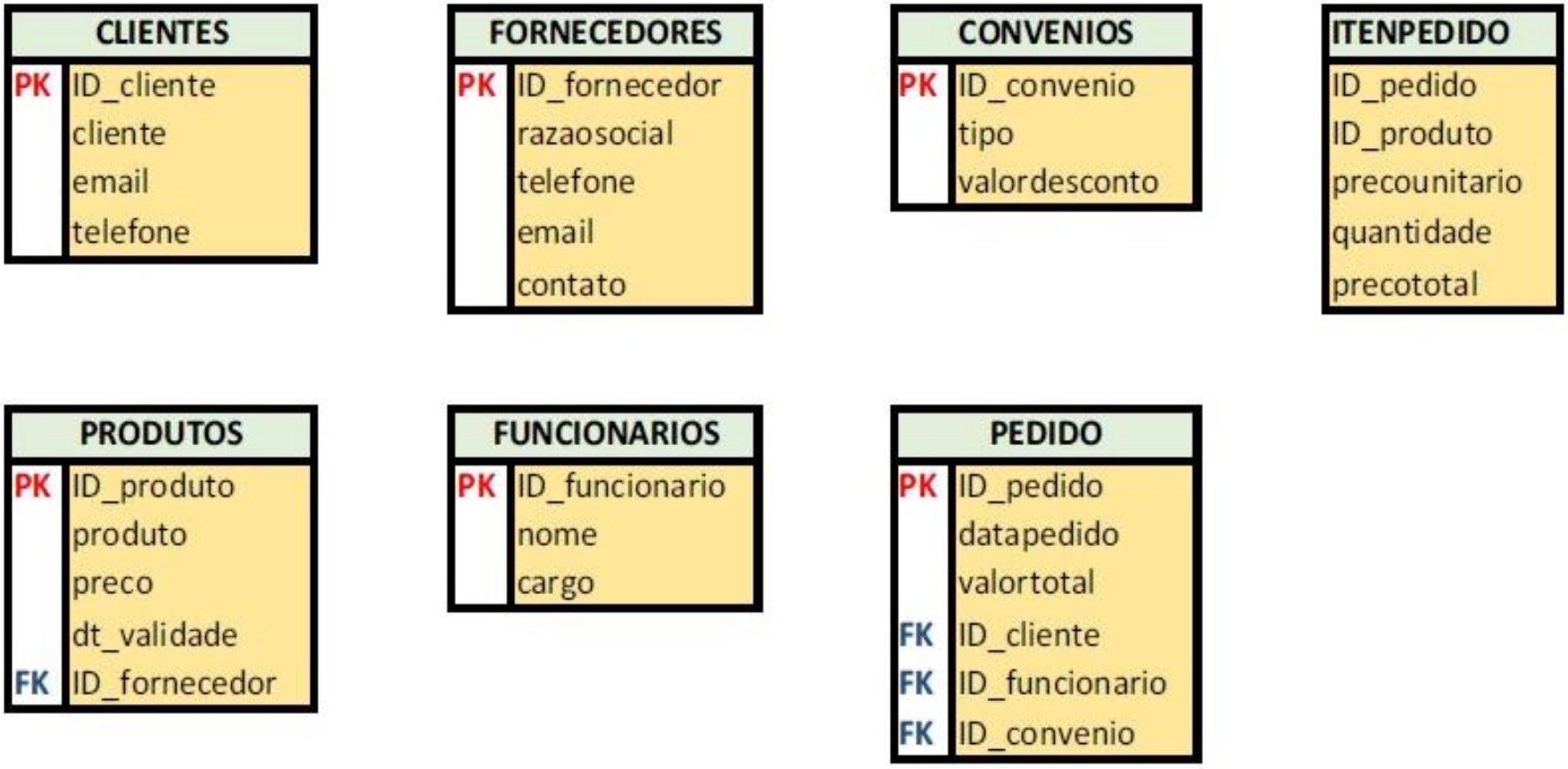
-- elaborar scripts de agrupamento com as funções SUM, AVG e COUNT

Exercício Final para Avaliação

Banco de dados **MERCADO**

Identifique as tabelas, os campos, as chaves primárias e estrangeiras

Crie os relacionamentos necessários e insira registros.



Enviar script para prof_Antunes@outlook.com

Assunto – POS FIB Aula 1