Competências, Habilidades e Bases Tecnológicas da disciplina de Banco de Dados II

II.3 BANCO DE DADOS II					
Função: Manipulação de dados em bancos de dados relacionais **Classificação:** Execução					
Atribuições e Responsabilidades					
Manipular dados, utilizando sistemas gerenciadores de bancos de dados relacionais.					
Valores e Atitudes					
Incentivar a criatividade. Desenvolver a criticidade. Estimular o interesse na resolução de situações-problema.					
Competência			**Habilidades**		
1. Efetuar operações em bancos de dados com SQL.			1.1 Executar comandos SQL para manipulação de dados. 1.2 Utilizar transações para garantia de integridade em bancos de dados.		
Bases Tecnológicas					
Práticas de SQL com SGBDR • Linguagem de manipulação de dados (DML); • Linguagem de consulta de dados (DQL); • Transações. DML • Inserção; • Atualização; • Exclusão. DQL • Projeção, seleção, renomeação; • Ordenação; • Agrupamento e funções agregadas; • Junção interna; • Junções externas à esquerda e à direita; • Produto cartesiano (*full/cross join*); • União, interseção e diferença. Transações • Operações *ACID;* • *COMMIT* e *ROLLBACK.*					
Carga horária (horas-aula)					
Teórica	**00**	**Prática Profissional**	**60**	**Total**	**60 Horas-aula**
Teórica (2,5)	**00**	**Prática Profissional (2,5)**	**50**	**Total (2,5)**	**00 Horas-aula**
Possibilidade de divisão de classes em turmas, conforme o item 4.8 do Plano de Curso.					

Observações a considerar:

1-) Comunicação de alunos com alunos e professores:

- Microsoft Teams;
- Criação de grupo nas redes sociais também é interessante.

2-) Uso de celulares:

Para o bom andamento das aulas, recomendo que utilizem os celulares em *vibracall*, para não atrapalhar o andamento da aula.

3-) Material das aulas:

A disciplina trabalha com **NOTAS DE AULA** que são disponibilizadas ao final de cada aula.

4-) Prazos de trabalhos e atividades:

Toda atividade solicitada terá uma data limite de entrega, de forma alguma tal data será postergada ou seja, se não for entregue até a data limite a mesma receberá menção I, isso tanto para atividades entregues de forma impressa, digital ou no Microsoft Teams.

5-) Qualidade do material de atividades:

- Impressas ou manuscritas:
  Muita atenção na qualidade do que será entregue, atividades sem grampear, faltando nome e número de componentes, rasgadas, amassadas, com rebarba de folha de caderno e etc. serão desconsiderados por mim.

- Digitais:
  Ao enviarem atividades para o e-mail, ou inseridas no Microsoft Teams, **SEMPRE** deverá ter o nome da atividade que está sendo enviada, e no corpo do e-mail ou da atividade deverá ter o(s) nome(s) do(s) integrante(s), sem estar desta forma a atividade será **DESCONSIDERADA**.

6-) Menções e critérios de avaliação:

Na ETEC os senhores serão avaliados por MENÇÃO, onde temos:
- MB - Muito Bom;
- B - Bom;
- R - Regular;
- I - Insatisfatório.

Cada trimestres poderemos ter as seguintes formas de avaliação:
- Avaliação teórica;
- Avaliação prática (a partir do 2º trimestre, e as turmas do 2º módulo em diante);
- Seminários;
- Trabalhos teóricos e/ou práticos;
- Assiduidade;
- Outras que se fizerem necessário.

Caso o aluno tenha alguma menção I em algum dos trimestres será aplicada uma recuperação, que poderá ser em forma de trabalho prático ou teórico.

1. Recomendações bibliográficas:

CASTRO, Eduardo. Modelagem Lógica de Dados: construção básica e simplificada. São Paulo. Ciência Moderna, Agosto 2012.

MARTELLI, Richard; FILHO, Ozeas; CABRAL, Alex. Modelagem e banco de dados. São Paulo. Senac, 2017.

PEREIRA, Paloma. Introdução a banco de dados. São Paulo. Senac, 2020.

NACIONAL, Senac. Modelagem de dados. Rio de Janeiro. Senac, 2014.

2. Revisão SQL

2.1. O MySQLWorkBench.

No decorrer desse curso veremos uma série de recursos de gerenciamento ligados ao Sistema Gerenciador de Banco de Dados Relacionais (**SGBDR**) "**Mysql**", esses recursos visam desde a manutenção dos bancos de dados e suas tabelas através de recursos como "**triggers**", "***stored procedures***", "***views***", "***index***" recursos esses fundamentais para a boa "**saúde**" de um banco de dados no que se refere a consistência dos dados e performance, passando também pela parte de segurança referente a criação e gestão de usuários e atribuição e revogação de direitos a esses usuários além de formas de realizar backups das bases de dados existentes.

Tais tarefas podem se mostrar desafiadoras ao "***Database Administrator***" (**DBA**) profissional responsável por tais tarefas, logo se faz necessário o uso de uma ou mais ferramentas que permita um alto grau de produção para esse profissional. Existe uma gama enorme de ferramentas capazes de fornecer esse auxilio para os mais diversos SGBDR corporativos existentes hoje no mercado. No nosso caso estamos trabalhando com o "**MySQL**" e vamos utilizar uma "**Integrated Development Envriroment**" (**IDE**) ou "**Ambiente Integrado de Desenvolvimento**" conhecido como "**MySQL Workbench**".

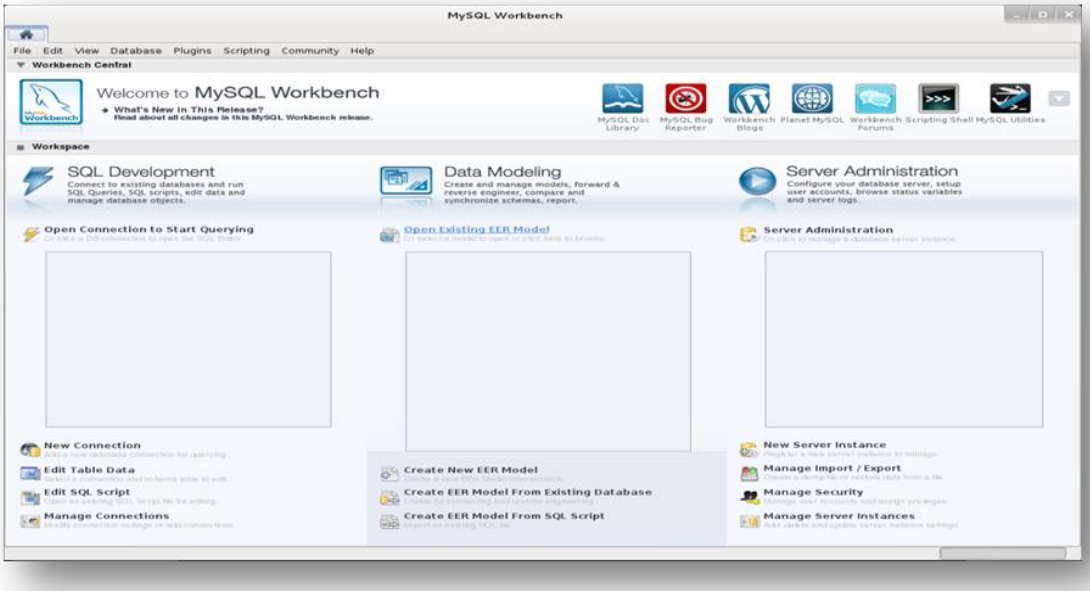
Essa ferramenta reúne dentro de si recursos divididos em três grupos básicos são eles:

- **SQL Development –** Reúne ferramentas ligadas a codificação de comandos **SQL Data Manipulation Language ou Linguagem de Manipulação de Dados** (**DML**), **Data definition Language** ou **Linguagem de Definição de Dados** (**DDL**), **Data Control Language** ou **Linguagem de controle de Dados** (**DCL**), **Data Transaction Language** ou **Linguagem de Transação de Dados** (**DTL**) e **Data Query Language ou Linguagem de Consulta de dados** (**DQL**).

- **Data Modeling –** Reúne ferramentas ligadas a modelagem de banco de dados permitindo a criação de diagramas/modelos de entidade relacionamento, além disso é possível converter esses diagramas em projetos físicos (bancos de dados) através de interface gráfica e podemos também realizar processos de engenharia reversa onde podemos "apontar" para um projeto físico de um determinado banco de dados e converter esse em diagramas/modelos de entidade relacionamento isso é muito útil principalmente quando estamos trabalhando em projetos que foram legados (já existiam e nos foram passados a título de continuação) e por alguma razão qualquer a equipe anterior não documentou a base de dados ou até mesmo essa documentação foi perdida.

- **Server Administrator –** Reúne ferramentas uteis para a gerencia de backups e da "saúde" das bases de dados criadas na instancia do "**MySQL**" que se está gerenciando, permite por exemplo verificar quantas conexões estão ativas nas bases de dados existentes, verificar a "saúde" da memória principal do servidor que hospeda a instancia do "**MySQL**", o trafego de dados, índice de eficiência das chaves, criar e associar a usuários diretos ou remover usuários e direitos, realizar backups entre outras ações.

A imagem abaixo ilustra a tela inicial do "**MySQL WorkBench**" onde podemos escolher entre os grupos de ferramentas citadas:



2.2. Aprimorando os comandos vistos

Os comandos da linguagem SQL são subdivididos em algumas categorias de comandos como: DDL, DML e DCL.

Nesta primeira etapa de BDII, usaremos a categoria DDL (*Data Definition Language* – Linguagem de Definição de Dados). Os comandos DDL são usados para definir a estrutura do banco de dados, organizando em tabelas que são compostas por campos (colunas). Comandos que compõem a DDL: CREATE, ALTER, DROP.

2.3. Tipos de Dados

Nas linguagens de programação, há quatro tipos primitivos de dados que são: inteiro, real, literal (texto) e lógico. Em cada SGBD, existem tipos de dados derivados desses tipos primitivos. No SGBD MySQL, é possível destacar os seguintes:

Tipo de Dado	Descrição
INTEGER	Representa um número inteiro
VARCHAR	Texto de tamanho variável. Máximo de 255 caracteres.
CHAR	Texto de tamanho fixo (preenche com espaços em branco os caracteres não preenchidos). Máximo de 255 caracteres.
DATE	Data
DATETIME	Data/Hora
TEXT	Texto de tamanho variável. Máximo de 65535 caracteres.
DECIMAL(p, d)	Número real, sendo que **p** define a precisão e **d** define o número de dígitos após o ponto decimal.

No MySQL, os campos ainda possuem atributos que definem a validação dos valores, tais como:

Atributo	Descrição	Aplica-se a
UNSIGNED	Sem sinal. Define que serão aceitos apenas números positivos.	Números inteiros
BINARY	Usado para diferenciar maiúsculas de minúsculas.	Texto

2.4. CREATE TABLE (com Cláusula DEFAULT)

A cláusula DEFAULT permite definir um valor padrão para um campo, que será utilizado caso não seja informado nenhum valor para esse campo na inserção de um registro na tabela.

Sintaxe:
sexo char(1) **default 'M'**

No exemplo acima, caso o campo "sexo" da tabela não seja preenchido com um valor durante a inserção de um registro, será assumido o valor 'M' para o campo. Para campos do tipo NUMÉRICO, o valor DEFAULT é escrito sem aspas. Exemplo:

salario decimal(10,2) **default 0**

Exemplo:

```
1  CREATE TABLE clientes(
2      cpf integer unsigned not null,
3      nome varchar(100) not null,
4      data_nascimento date not null,
5      sexo char(1) default "M",
6      salario decimal(10,2) default 0,
7      profissao varchar(30),
8      primary key(cpf)
9  );
```

2.4.1. CONSTRAINTS

Constraints são restrições feitas para as colunas nas tabelas contendo diversos tipos, são utilizadas na criação de uma tabela ou mesmo junto com a ***ALTER TABLE***, onde podemos adicionar ou remover *constraints*. Existem os seguintes:

a) *NOT NULL*: define que um campo da tabela é obrigatório (deve receber um valor na inserção de um registro);

b) *PRIMARY KEY*: define que um campo ou conjunto de campos para garantir a identidade de cada registro. Quando um campo é definido como chave primária, seu valor não pode se repetir em registros diferentes. Cada tabela só pode ter uma única chave primária. Podemos ainda as definir como:

- CHAVE PRIMÁRIA SIMPLES: composta por um único campo. Exemplo: se for definido que em um sistema de hotéis não podem existir dois clientes com o mesmo CPF, portanto este campo deverá ser definido como CHAVE PRIMÁRIA.

- CHAVE PRIMÁRIA COMPOSTA: formada por dois ou mais campos. Exemplo: se for definido em um sistema de Agências bancárias que não podem existir duas contas com o mesmo número da mesma agência, então esses dois campos formarão uma CHAVE PRIMÁRIA COMPOSTA, pois a combinação deles não pode se repetir.

Número da Conta	Número da Agência
1234	123
1234	567
3432	123

Observe que o número da conta pode se repetir individualmente, e o mesmo vale para o número da agência, porém a combinação desses dois campos não pode se repetir, garantindo que não existirão duas contas com o mesmo número na mesma agência.

Sintaxe:

- **Criação de uma chave primária simples:**

```
CREATE TABLE contas(
    numero integer not null primary key,
    saldo integer default 0,
    agencia_numero integer not null
);
```

Ou

```
CREATE TABLE contas(
    numero integer not null,
    saldo integer default 0,
    agencia_numero integer not null,
    primary key(numero)
);
```

Ou

```
CREATE TABLE contas(
    numero integer not null,
    saldo integer default 0,
    agencia_numero integer not null,
    constraint pk_conta primary key(numero)
);
```

- **Criação de uma chave primária composta**

```
CREATE TABLE contas(
    numero integer not null,
    saldo integer default 0,
    agencia_numero integer not null,
    primary key(numero,agencia_numero)
);
```

Ou

```
CREATE TABLE contas(
    numero integer not null,
    saldo integer default 0,
    agencia_numero integer not null,
    constraint pk_conta primary key(numero,agencia_numero)
);
```

c) FOREIGN KEY: Uma **chave estrangeira** é definida quando se deseja relacionar tabelas do banco de dados.

Sintaxe:

```
CREATE TABLE contas(
    numero integer not null,
    saldo integer default 0,
    agencia_numero integer not null,
    primary key(numero,agencia_numero),
    foreign key(agencia_numero) references agencias(numero)
);
```

Ou

```
CREATE TABLE contas(
    numero integer not null,
    saldo integer default 0,
    agencia_numero integer not null,
    primary key(numero,agencia_numero),
    constraint fk_contaagencia foreign key(agencia_numero) references
    agencias(numero)
);
```

Na criação da chave estrangeira do exemplo anterior, pode-se ler da seguinte forma: O campo **agencia_numero** da tabela **contas** faz referência ao campo **numero** da tabela **agencias**.

d) UNIQUE: Uma *constraint* **UNIQUE** define que o valor de um campo ou de uma sequência de campos não pode se repetir em registros da mesma tabela. Essa *constraint* é criada de forma implícita quando é definida uma chave primária para uma tabela. Como só é possível ter uma chave primária por tabela, a utilização de

constraints **UNIQUE** é uma solução quando se deseja restringir valores repetidos em outros campos.

Exemplo:

```
CREATE TABLE clientes(
    cpf integer not null,
    nome varchar(100) not null,
    data_nascimento date not null,
    sexo char(1) default "M",
    salario decimal(10,2) default 0,
    profissao varchar(30),
    rg integer not null,
    estado char(2) not null,
    primary key(cpf),
    unique(rg,estado)
);
```

A criação da *constraint* acima garante que um número de RG não se repetirá em um mesmo estado.

2.5. ALTER TABLE

Para não se apagar uma tabela e recriá-la, é possível fazer alterações em sua estrutura por meio do comando ALTER TABLE. Isso é importante pois a execução do comando ***DROP TABLE*** apaga (obviamente) todos os registros da tabela, já a execução do comando ***ALTER TABLE*** não exclui nenhum registro.

- **Adicionar um campo**

Sintaxe:
ALTER TABLE nome_da_tabela ADD nome_do_campo tipo_de_dado atributos

Exemplo:

```
ALTER TABLE clientes ADD endereco varchar(90) not null;
```

- **Adicionar um campo posicionando depois de um campo já existente**

Sintaxe:
ALTER TABLE nome_da_tabela ADD nome_do_campo tipo_de_dado atributos AFTER nome_do_campo_ja_existente

Exemplo:

```
ALTER TABLE clientes ADD endereco varchar(90) not null AFTER nome;
```

- **Alterar o tipo de dado de um campo**

Sintaxe:
ALTER TABLE nome_da_tabela MODIFY nome_do_campo tipo_de_dado

Exemplo:

```
ALTER TABLE clientes MODIFY endereco varchar(200);
```

- **Renomear um campo e modificar o tipo**

Sintaxe:
ALTER TABLE nome_da_tabela CHANGE COLUMN nome_do_campo novo_nome tipo atributos;

Exemplo para mudar apenas o nome (o tipo do campo é mantido):

```
ALTER TABLE clientes CHANGE COLUMN data_nascimento datanasc date;
```

Exemplo para mudar o nome e o tipo do campo:

```
1 •   ALTER TABLE clientes CHANGE COLUMN data_nascimento datahoranasc datetime;
```

- **Renomear uma tabela**

Sintaxe:
ALTER TABLE nome_da_tabela RENAME TO novo_nome_da_tabela

Exemplo:

```
1 •   ALTER TABLE clientes RENAME TO pessoas_fisicas;
```

- **Apagar um campo**

Sintaxe:
ALTER TABLE nome_da_tabela DROP COLUMN nome_do_campo

Exemplo:

```
1 •   ALTER TABLE clientes DROP COLUMN endereco;
```

- **Adicionar uma PRIMARY KEY**

Sintaxe:
ALTER TABLE nome_da_tabela ADD CONSTRAINT nome_da_constraint PRIMARY KEY(campo1[,campo2,campo3,...,campoN])

Exemplo:

```
1 •   ALTER TABLE clientes ADD CONSTRAINT pk_cpf PRIMARY KEY(cpf);
```

- **Apagar uma PRIMARY KEY**

Sintaxe:
ALTER TABLE nome_da_tabela DROP PRIMARY KEY

Ou

ALTER TABLE nome_da_tabela DROP CONSTRAINT
nome_da_constraint_da_primary_key

Exemplo:

```
1 •   ALTER TABLE clientes DROP PRIMARY KEY;
```

Ou

```
1 •   ALTER TABLE clientes DROP pk_cpf;
```

- **Adicionar uma FOREIGN KEY**

Sintaxe:
ALTER TABLE nome_da_tabela ADD CONSTRAINT nome_da_constraint FOREIGN KEY(campo1[,campo2,campo3,...,campoN]) REFERENCES nome_da_tabela(campo1[,campo2,campo3,...,campoN]);

Exemplo:

```
1 •   ALTER TABLE contas ADD CONSTRAINT fk_contaagencia
2     FOREIGN KEY(agencia_numero) REFERENCES agencias(numero);
```

- **Adicionar uma constraint UNIQUE**

Sintaxe:
ALTER TABLE nome_da_tabela ADD CONSTRAINT nome_da_constraint UNIQUE(campo1[,campo2,campo3,...,campoN])

Exemplo:

```
1 •   ALTER TABLE clientes ADD CONSTRAINT un_rgestado UNIQUE(rg,estado);
```

- **Apagar uma CONSTRAINT qualquer**

Sintaxe:
ALTER TABLE nome_da_tabela DROP CONSTRAINT nome_da_constraint

Exemplo:

```
1 •   ALTER TABLE contas DROP fk_contaagencia;
```

2.6. Comando *SHOW TABLES*

Para visualizar todas as tabelas em um banco de dados, utilize o comando *SHOW TABLES.*

Exemplo:

```
1 •   SHOW TABLES;
```

2.7. Comando DESC

Para visualizar a estrutura de uma tabela, utilize o comando DESC (ou DESCRIBE).

Exemplo:

```
1 •   DESC clientes;
```

3. Manipulando dados das tabelas

3.1 DML (*Data Manipulation Language* – Linguagem de Manipulação de Dados)

Os comandos DML permitem realizar operações de inserção, alteração, exclusão e seleção sobre os registros (linhas) das tabelas. Comandos que compõem a DML: *INSERT, UPDATE, DELETE* e *SELECT*. Alguns autores definem que o comando *SELECT* faz parte de uma subdivisão chamada DQL (*Data Query Language* – Linguagem de Consulta de Dados).

3.2. Comando *INSERT*

Adiciona um ou vários registros a uma tabela. Isto é referido como consulta anexação.

Sintaxe:
INSERT INTO destino [(campo1[, campo2[, ...]])]
VALUES (valor1[, valor2[, ...])

Onde;

- **Destino -** O nome da tabela ou consulta em que os registros devem ser anexados.
- **campo1, campo2** - Os nomes dos campos aos quais os dados devem ser anexados
- **valor1, valor2** - Os valores para inserir em campos específicos do novo registro. Cada valor é inserido no campo que corresponde à posição do valor na lista: Valor1 é inserido no campo1 do novo registro, valor2 no campo2 e assim por diante.

Os valores devem ser separados com uma vírgula e os campos de textos entre aspas duplas ou simples.

Exemplo:

```
1 •   INSERT INTO alunos (Id_aluno, nome, endereco, turma, turno)
2     VALUES (1, "Glaucio", "Av. das Américas", "1101", "manhã");
```

3.3. Comando *SELECT*

Permite recuperar informações existentes nas tabelas.

Sintaxe:
SELECT [DISTINCT] expressao [AS nom-atributo] [FROM from-list] [WHERE condicao] [ORDER BY attr_name1 [ASC | DESC]

onde:

- **DISTINCT** - Para eliminar linhas duplicadas na saída.
- **Expressão** - Define os dados que queremos na saída, normalmente uma ou mais colunas de uma tabela da lista FROM.
- **AS nom-atributo** - um alias para o nome da coluna
- **FROM** - lista das tabelas na entrada

• **WHERE** - critérios da seleção
• **ORDER BY** - Critério de ordenação das tabelas de saída. Podem ser:
• **ASC** - ordem ascendente (crescente);
• **DESC** - ordem descendente (decrescente)

Exemplo:

```
1 •   SELECT cidade AS cid, estado AS est FROM brasil
2     WHERE populacao > 100000 ORDER BY cidade DESC;
```

3.4 Comando *UPDATE*

Cria uma consulta atualização que altera os valores dos campos em uma tabela especificada com base em critérios específicos.

Sintaxe:
UPDATE tabela SET campo1 = valornovo, ... WHERE critério;

onde:
• **Tabela** - O nome da tabela cujos dados você quer modificar.
• **Valornovo** - Uma expressão que determina o valor a ser inserido em um campo específico nos registros atualizados.
• **critério** - Uma expressão que determina quais registros devem ser atualizados. Só os registros que satisfazem a expressão são atualizado.

Exemplo:

```
1 •   UPDATE alunos SET turno = "tarde" WHERE turma = 1101;
```

UPDATE é especialmente útil quando você quer alterar muitos registros ou quando os registros que você quer alterar estão em várias tabelas. Você pode alterar vários campos ao mesmo tempo.

UPDATE não gera um conjunto de resultados. Se você quiser saber quais resultados serão alterados, examine primeiro os resultados da consulta seleção que use os mesmos critérios e então execute a consulta atualização.

3.5 Comando *DELETE*

Remove registros de uma ou mais tabelas listadas na cláusula *FROM* que satisfaz a cláusula *WHERE*.

Sintaxe:
DELETE [tabela.*] FROM tabela WHERE critério

onde:
tabela.* - O nome opcional da tabela da qual os registros são excluídos.
tabela - O nome da tabela da qual os registros são excluídos.
critério - Uma expressão que determina qual registro deve ser excluído.

Exemplo:

```
1 •   DELETE FROM alunos WHERE turno = "Manhã";
```

DELETE é especialmente útil quando você quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados, você pode usar o método Execute com uma instrução *DROP*. Entretanto, se você eliminar a tabela, a estrutura é perdida. Por outro lado, quando você usa *DELETE*, apenas os dados são excluídos.

A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos. Você pode usar *DELETE* para remover registros de tabelas que estão em uma relação um por vários com outras tabelas. Operações de exclusão em cascata fazem com que os registros das tabelas que estão no lado "vários" da relação sejam excluídos quando os registros correspondentes do lado "um" da relação são excluídos na consulta.
Por exemplo, nas relações entre as tabelas Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz

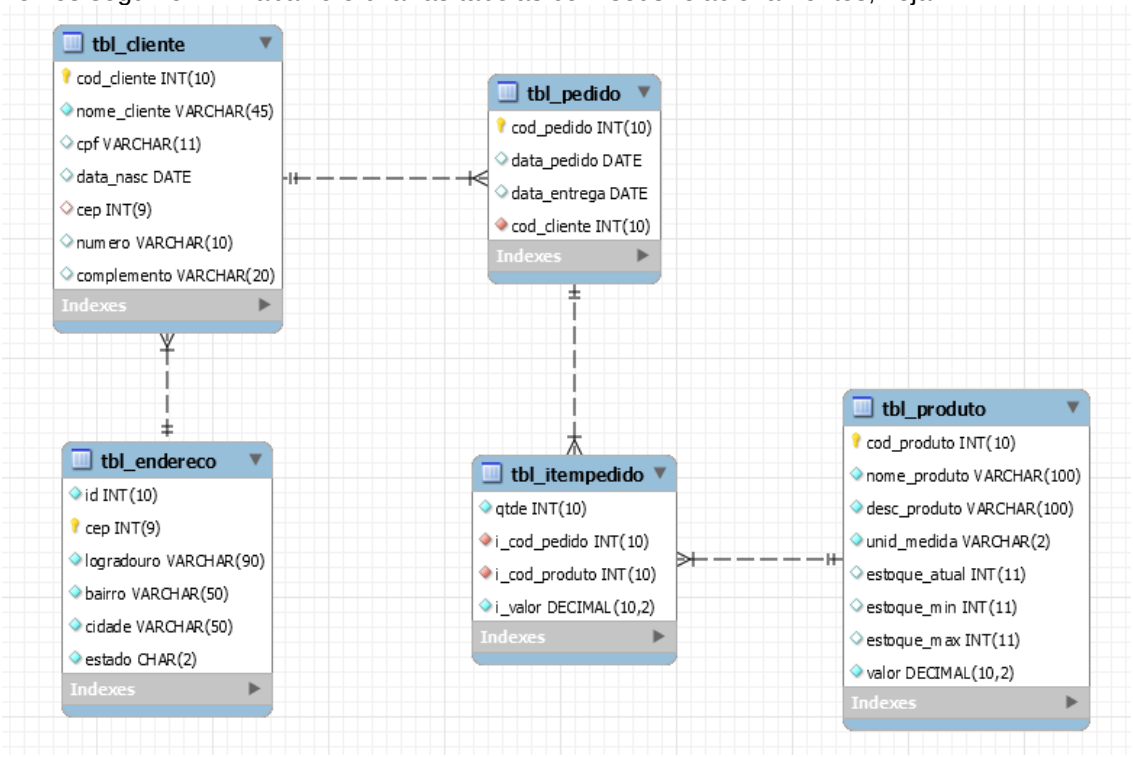
com que os registros correspondentes em Pedidos sejam excluídos se a opção de exclusão em cascata for especificada.

O *DELETE* exclui registros inteiros e não apenas dados em campos específicos. Se você quiser excluir valores de um campo específico, crie uma consulta atualização que mude os valores para *Null*.

Após remover os registros usando uma consulta exclusão, você não poderá desfazer a operação. Se quiser saber quais arquivos foram excluídos, primeiro examine os resultados de uma consulta seleção que use o mesmo critério e então, execute a consulta exclusão. Mantenha os *backups* de seus dados. Se você excluir os registros errados, poderá recuperá-los a partir dos seus backups.

4. SQL

Vamos aplicar os conceitos vistos, fazendo um banco de dados de pedidos onde iremos seguir o DER abaixo e criar as tabelas com seus relacionamentos, veja:



Mas iremos fazer esse banco de dados através de comandos SQL, primeiramente vamos organizar as coisas, iremos criar um Script SQL chamado “bd_vendas DDL” onde irá conter a criação do banco de dados e as tabelas mostradas no DER anterior.

- Criando e habilitando a base de dados:

```
/*
CRIAÇÃO DO BANCO DE DADOS BD_VENDAS - TLBD III
*/
create database bd_vendas;
-- -----------------------------------------------------

/*
HABILITANDO O BANCO DE DADOS PARA USO
*/
use bd_vendas;
-- -----------------------------------------------------
```

- Criando a tabela de produtos:

```
/*
CRIAÇÃO DA TABELA DE PRODUTOS
*/
create table tbl_produto (
   cod_produto   int unsigned auto_increment,
   nome_produto  varchar(100) not null,
   desc_produto  varchar(100) not null,
   unid_medida   varchar(2) not null,
   estoque_atual int default 0,
   estoque_min   int default 0,
   estoque_max   int default 0,
   valor         decimal(10,2) not null,
   primary key (cod_produto));
-- -----------------------------------------------------
```

- Criando a tabela de endereços para armazenar os CEPs:

```
/*
CRIAÇÃO DA TABELA DE ENDEREÇO - CEP
*/
create table tbl_endereco (
   id int(10) not null,
   cep int(9) not null,
   logradouro varchar(90) not null,
   bairro varchar(50) not null,
   cidade varchar(50) not null,
   estado char(2) not null,

   constraint pk_endereco primary key (cep)
);
```

- Criando a tabela de clientes:

```
/*
CRIAÇÃO DA TABELA DE CLIENTES
*/
create table tbl_cliente (
   cod_cliente  int unsigned auto_increment,
   nome_cliente varchar(45) not null,
   cpf          varchar(11) default '',
   data_nasc    date,
   cep          int(9) default 0,
   numero       varchar(10) default '',
   complemento  varchar(20) default '',

   primary key (cod_cliente),
   constraint foreign key fk_cliencep (cep) references tbl_endereco(cep)
   );

-- -----------------------------------------------------------------------
```

- Criando a tabela de pedidos:

```
/*
CRIAÇÃO DA TABELA DE PEDIDOS
*/
create table tbl_pedido (
   cod_pedido   int unsigned auto_increment,
   data_pedido  date,
   data_entrega date,
   cod_cliente  int unsigned not null,
   primary key (cod_pedido),
   constraint fk_cliente foreign key (cod_cliente)
   references tbl_cliente(cod_cliente));
-- -----------------------------------------------------------
```

- Criando a tabela de itens do pedido:

```
/*
CRIAÇÃO DA TABELA DE ITENS DO PEDIDO
*/
create table tbl_itempedido (
  qtde          int unsigned not null,
  i_cod_pedido  int unsigned not null,
  i_cod_produto int unsigned not null,
  i_valor         decimal(10,2) not null,
  constraint fk_pedido1
    foreign key (i_cod_pedido)
    references tbl_pedido (cod_pedido),
  constraint fk_tbl_produto1
    foreign key (i_cod_produto)
    references tbl_produto(cod_produto));
-- -----------------------------------------------------
```

Feito isso salvem esse arquivo e mantenha sempre com vocês em nossas aulas, pois iremos incrementar mais tabelas no decorrer do curso.

Caso queiram para verificar se a construção do seu banco de dados ficou igual a mencionada inicialmente, façam a engenharia reversa (*Reverse Engineer*) no menu *DATABASE* para verificar se ficou igual.

4.1. Popular as tabelas

Iremos agora inserir dados nas tabelas para que possamos dar seguimento aos nossos estudos, mas a primeira pergunta que vem a nossa mente é: “Por qual tabela devo começar?”. Para essa resposta precisamos analisar o DER e verificar quais tabelas não possuem dependências de chaves estrangeiras e começar pelas mesmas, assim as tabelas de endereço e produto devem ser as primeiras a serem populadas, pois são tabelas que fornecem suas chaves para o preenchimento de outras, se fizermos ao contrário, a inserção dará erro.

Para isso, vamos criar outro Script SQL chamado “bd_vendas DML”, onde iremos colocar todos os *INSERTs* que iremos fazer para popular essas tabelas, esse também deverá acompanhar os senhores para nossas aulas.

Para darmos início, vamos popular a tabela de endereço com os CEPs, vou disponibilizar um Script com esses *inserts* com as cidades de Taboão da Serra e Embu das Artes, darão cerca de 2000 registros nessa tabela.

Na sequência vamos inserir os produtos:

```
insert into tbl_produto (nome_produto, desc_produto, unid_medida,
estoque_atual,estoque_min, estoque_max,valor) values
('Arroz', 'Arroz agulhinha tipo 1','SC', 10,2,20, 12.50),
('Feijão', 'Feijão carioquinha com casca','SC', 25,5,60, 7.50),
('Macarrão', 'Macarrão Adria espaguete','PC', 50,10,80, 5.50),
('Óleo', 'Óleo Lisa','LT', 15,10,45, 6.50),
('Vinagre', 'Vinagre Castelo','GR', 30,10,50, 7.89),
('Batata', 'Batata lavada','KG', 100,50,200, 4.50),
('Tomate', 'Tomate vermelho','KG', 80,8,160, 6.90),
('Cebola', 'Cebola com casca','KG', 50,5,100, 6.99),
('Leite', 'Leite Leco','CX', 25,10,90, 2.50),
('Café', 'Café do Ponto','SC', 500,100,200, 11.50);
```

Feito isso, podemos realizar as inserções dos clientes, pois sem os dados da tabela de CEPs não seria possível chegar a essa etapa. **Só lembrando que para a inserção dessa tabela, o CEP informado deverá estar cadastrado na tabela de CEPs.**

```
/*
  Clientes
*/
insert into tbl_cliente(nome_cliente,cpf,data_nasc,cep,numero,complemento) values
('Marcos Costa de Sousa','12345678901','1981-02-06',6768100,'1525','apto 166C'),
('Zoroastro Zoando','01987654321','1989-06-15',6757190,'250',''),
('Idelbrandolâncio Silva','54698721364','1974-09-27',6753001,'120',''),
('Cosmólio Ferreira','41368529687','1966-12-01',6753020,'25','apto 255 F'),
('Conegunda Prado','54781269501','1950-10-06',6753020,'50','apto 166C'),
('Brogundes Asmônio','41256398745','1940-05-10',6753400,'100',''),
('Iscruência da Silva','12457965823','1974-11-25',6803040,'5',''),
('Zizafânio Zizundo','54123698562','1964-08-14',6803140,'25',''),
('Ricuerda Zunda','21698534589','1934-10-14',6803045,'123',''),
('Aninoado Zinzão','25639856971','1976-12-25',6803070,'50','');
```

4.2. Exercício para menção:

"Populem" as tabelas de acordo com o MER desenvolvido por vocês, com a massa de dados consistente de produtos e clientes, que passei para vocês anteriormente, e após isso, implementem:

1º - Criar no mínimo 5 pedidos;

2º - Para cada pedido, pelo menos 3 itens em cada;

Podem realizar em duplas, prazo para entrega, próxima semana, valendo **menção**. Irei verificar toda a codificação do banco de dados e os *INSERTs* feitos por vocês.

5. *Transactions* em banco de dados

Transação ou *Transaction* é uma única unidade de trabalho processada pelo Sistema de Gerenciamento de Banco de Dados (SGBD). Imagine que precisamos realizar em uma única transação duas operações de manipulação de dados (DML, do inglês Data Manipulation Language), como *INSERT, DELETE* e *UPDATE*. Estas operações só podem se tornar permanentes no banco de dados se todas forem executadas com sucesso. Em caso de falhas em uma das duas é possível cancelar a transação, porém todas modificações realizadas durante a mesma serão descartadas, inclusive a que obtive sucesso. Assim, os dados permanecem íntegros e consistentes.

Podemos caracterizar as transações conforme abaixo:

O *BEGIN TRANSACTION* indica onde ela deve começar, então os comando SQL a seguir estarão dentro desta transação.

O *COMMIT TRANSACTION* indica o fim normal da transação, o que tiver de comando depois já não fará parte desta transação. Neste momento tudo o que foi manipulado passa fazer parte do banco de dados normalmente e operações diversas passam enxergar o que foi feito.

O *ROLLBACK TRANSACTION* também fecha o bloco da transação e é a indicação que a transação deve ser terminada, mas tudo que tentou ser feito deve ser descartado porque alguma coisa errada aconteceu e ela não pode terminar normalmente. Nada realizado dentro dela será perdurado no banco de dados.

Ao contrário do que muita gente acredita *rollback* no contexto de banco de dados não significa reverter e sim voltar ao estado original. Um processo de reversão seria de complicadíssimo à impossível.

A maioria dos comandos SQL são transacionais implicitamente, ou seja, ele por si só já é uma transação. Você só precisa usar esses comandos citados quando precisa usar múltiplos comandos.

6. Comandos DML

Vamos neste momento alternar nossas aulas de acordo com o que foi visto até agora, neste capítulo iremos ver *UPDATE, INSERT, DELETE* e principalmente o *SELECT*, nos aprofundando na DQL, que é uma ramificação da DML, tudo isso para melhor fixação do conteúdo, que é vital neste momento do nosso curso. Apesar de termos as tabelas criadas e populadas conforme aula anterior, neste primeiro momento iremos nos ater em trabalhar somente com a tabela de Clientes **TBL_CLIENTE** e Endereço **TBL_ENDERECO**, em seguida passamos para as outras tabelas até finalizarmos fazendo os todos os procedimentos vistos com as chaves estrangeiras criadas nestas tabelas.

6.1 – SELECT

De acordo com a tabela “populada”, podemos retirar informações ricas da mesma, para isso iremos ver a DQL, que nos ajudará muito nesta extração de informações.

6.1.1 – CLÁUSULA DISTINCT

A cláusula DISTINCT permite que os dados repetidos de uma tabela sejam exibidos uma única vez, vamos observar o exemplo abaixo:

```
select distinct bairro
from tbl_endereco;
```

Neste caso somente serão mostrados os bairros distintos existentes na **TBL_ENDERECO,** ou seja, aparecerá no resultado 202 linhas, então podemos dizer que a tabela de endereço da nossa base de dados possui 202 bairros diferentes em seu cadastro.

Como qualquer SELECT, podemos refinar mais a consulta, por exemplo se quisermos ver a quantidade distintas de bairro por município por exemplo:

```
select distinct bairro
from tbl_endereco
where cidade = 'Taboão da Serra';
```

Nesse outro exemplo filtramos pelo município de Taboão da Serra, onde foram apresentadas 98 linhas, ou seja, em Taboão da Serra a nossa tabela possui 98 bairros diferentes.

6.1.2 – CLÁUSULA LIMIT

A cláusula LIMIT é usada para determinar um limite para a quantidade e para a faixa de linhas que podem ser retornadas, vamos neste nosso exemplo mostrar as linhas referentes aos clientes ISCRUÊNCIA e ZIZAFÂNIO observar o exemplo abaixo:

```
select *
from tbl_cliente
limit 6,2
```

Observe que o primeiro parâmetro de limite especificado (6) refere-se ao deslocamento, ou seja, ao ponto inicial, ao passo que o segundo parâmetro (2) refere-se a quantidade de linhas a serem retornadas.

Quando um único parâmetro é informado, este se refere a quantidade de linhas retornadas.

6.1.3 – CLÁUSULA ORDER BY

A cláusula ORDER BY **pode** ser usada juntamente com a cláusula LIMIT quando for importante que as linhas sejam retornadas de acordo com uma determinada ordem, observar o exemplo abaixo:

```
select *
from tbl_cliente
order by nome_cliente
limit 6,2;
```

Esta ordem também poderá ser ASCENDENTE ou DESCENDENTE dependendo da minha necessidade, veja:

```
select *
from tbl_cliente
order by nome_cliente desc
limit 6,2;
```

Ou

```
select *
from tbl_cliente
order by nome_cliente asc
limit 6,2;
```

Quando não informamos a classificação depois da cláusula ORDER BY por default é considerado ASC.

6.1.4 – CLÁUSULA WHERE

A cláusula *WHERE* serve para determinar quais os dados de uma tabela que serão afetados pelos comandos *SELECT, UPDATE* e *DELETE*, podemos dizer então que esta cláusula determina o escopo de uma consulta a algumas linhas, realizando uma filtragem dos dados que estejam de acordo com as condições definidas, vamos observar o exemplo abaixo:

```
select *
from tbl_cliente
where cpf = '12345678901';
```

Neste caso trará os dados do cliente “MARCOS COSTA DE SOUSA” que possui o CPF 12345678901.

```
select *
from tbl_endereco
where bairro = 'Jardim Caner'
```

Neste outro caso os dados de endereço que são do Bairro JD CANER serão mostrados.

```
select *
from tbl_endereco
where cidade = 'Taboão da Serra'
```

Neste outro caso os dados de endereço que são da Cidade de TABOÃO DA SERRA serão mostrados.

6.1.5 – OPERADORES AND E OR

Os operadores lógicos AND e OR são utilizados junto com a cláusula WHERE quando for necessário utilizar mais de uma condição de comparação, vamos observar o exemplo abaixo:

```
select * from tbl_endereco
where estado = 'SP'
  and cidade = 'Taboão da Serra'
```

Neste caso todos endereços da UF de SP e a CIDADE de TABOÃO DA SERRA serão mostrados.

```
select * from tbl_endereco
where estado = 'SP'
   or cidade = 'Taboão da Serra';
```

Neste caso todos os endereços da UF de SP ou da CIDADE de TABOÃO DA SERRA serão mostrados, em nosso exemplo irão aparecer todos os dados cadastrados, em caso de uma base de dados de todo o Brasil por exemplo, se existisse a cidade de Taboão da Serra na Bahia por exemplo, seria mostrada no resultado.

6.1.6 – OPERADOR IN

O operador IN permite checar se o valor de uma consulta está presente em uma lista de elementos.

```
select * from tbl_endereco
where bairro in ('Jardim Kuabara','Jardim Pazini');
```

Neste caso todos os Endereços dos BAIRROS Jardim Kuabara e Jardim Pazini serão mostrados.

Com o NOT antecedido do IN, faz a operação inversa, ou seja não trará os endereços dos BAIRROS Jardim Kuabara e Jardim Pazini.

```
select * from tbl_endereco
where bairro not in ('Jardim Kuabara','Jardim Pazini');
```

6.1.7 – OPERADOR LIKE

O operador LIKE é empregado quando a base para realizar pesquisa forem as colunas que estão no formato char ou varchar, vamos observar o exemplo abaixo:

```
select * from tbl_cliente
where nome_cliente like 'M%'
```

Neste caso todos os clientes que possuem a letra M no início do seu nome serão mostrados.

```
select * from tbl_cliente
where nome_cliente like '%M%';
```

Neste caso todos os clientes que possuem a letra M em qualquer parte do seu nome serão mostrados.

```
select * from tbl_cliente
where nome_cliente like '%M';
```

Neste caso todos os clientes que possuem a letra M no fim do seu nome serão mostrados.

Com o NOT antecedido do LIKE, faz a operação inversa, ou seja não trará os clientes dos nas situações acima, veja:.

```
select * from tbl_cliente
where nome_cliente not like 'M%';
```

Neste caso todos os clientes que possuem a letra M no início do seu nome não serão mostrados.

```
select * from tbl_cliente
where nome_cliente not like '%M%';
```

Neste caso todos os clientes que possuem a letra M em qualquer parte do seu nome não serão mostrados.

```
select * from tbl_cliente
where nome_cliente not like '%M';
```

Neste caso todos os clientes que possuem a letra M no fim do seu nome não serão mostrados.

```
select * from tbl_cliente
where nome_cliente like 'M_R__S%';
```

Neste caso todos os clientes que possuem na 1ª posição de seus nomes letra M, na 3ª posição a letra R e na 5ª posição a letra S e não importando o restante do nome serão mostrados, nesse caso adivinhem o nome de quem aparecerá!!!

6.1.8 – OPERADOR BETWEEN

O operador BETWEEN tem a função de permitir a consulta de uma determinada faixa de valores. Este operador permite checar se o valor de uma coluna encontra-se em um determinado intervalo, vamos observar o exemplo abaixo:

```
select * from tbl_produto
where estoque_min between 10 and 100
```

Neste caso serão mostrados todos os registros que possuem o campo ESTOQUE_MIN entre 10 e 100.

Outro exemplo, podemos usar o NOT BETWEEN

```
select * from tbl_produto
where estoque_min not between 10 and 100
```

Neste caso é feito o contrário do exemplo dado, ele trará os registro que NÃO estejam com o campo ESTOQUE_MIN entre 10 e 100.

6.1.9 – CLÁUSULA COUNT

A cláusula *COUNT* serve para contarmos quantas ocorrências existe um determinado *SELECT*, vamos observar o exemplo a seguir:

```
select count(*) from tbl_produto
where estoque_min between 10 and 100
```

Neste caso é realizada a contagem de quantas ocorrências existem de acordo com o campo ESTOQUE_MIN entre 10 e 100, ele mostrará o número 6.

6.1.10 – CLÁUSULA GROUP BY

A cláusula *GROUP BY* serve para agrupar diversos registros com base em uma ou mais colunas da tabela, vamos observar o exemplo abaixo:

```
select count(*), unid_medida
from tbl_produto
group by unid_medida
```

Neste caso serão mostrados os registros agrupados pelo campo UNID_MEDIDA, aparecerão CX = 1, GR = 1, KG = 3, LT = 1, PC = 1 e SC = 3.

```
select count(*), month(data_nasc)
from tbl_cliente
where year(data_nasc) > 1930
group by month(data_nasc)
order by 2
```

Neste caso será mostrada a quantidade de ocorrências existentes de acordo o mês do campo DATA_NASC da tabela TBL_CLIENTE, pois estamos contando e agrupando pelo mês, com o ano do mesmo campo maior que 1930.

```
select count(*) as qtde, month(data_nasc) as mes,
year(data_nasc) as ano
from tbl_cliente
where year(data_nasc) > 1930
group by month(data_nasc), year(data_nasc)
order by 2
```

Neste caso será mostrada a quantidade de ocorrências existentes igualmente ao caso anterior, só com a diferença de colocarmos o ano e agrupando o mesmo, isso nos trará mais uma linha no resultado.

6.2. Funções agregadas

Podemos resumir as informações dos dados de uma tabela através de funções agregadas, que são aplicadas sobre as colunas de uma tabela. Devemos passar como argumento de entrada uma coluna que precisamos extrair uma simples informação da mesma, um único resultado, para que assim, não seja preciso analisar informação por informação, com uso da nossa força bruta.

Você sabe como obter um valor resultante de uma simples soma dos valores de uma coluna específica?

Sabe extrair qual é o maior ou menor valor entre muitos em um conjunto?

Sabe quantas linhas uma tabela possui?

Sabe fazer a média entre um conjunto de valores?

Iremos ver 4 diferentes funções agregadas que geram diferentes tipos de resultado, são elas:

- SUM(): computa e retorna o resultado da soma de todos os valores de uma coluna informada;
- AVG(): retorna a média dos valores de uma coluna;
- MIN(): encontra o menor valor dentre aqueles contidos na coluna de entrada;
- MAX(): encontra o maior dentre todos os valores de um conjunto;

2.2.1 Função SUM()

A função soma computa a soma dos valores de uma coluna. Essa função serve para manipularmos dados numéricos. O tipo numérico do dado pode ser *integer e decimal*. O resultado da função *SUM()* será do mesmo tipo da coluna que está sendo utilizada por esta função, vamos observar o exemplo abaixo:

```
select sum(estoque_min) as soma_estoque
from tbl_produto;
```

Neste caso será mostrada a soma do campo ESTOQUE_MIN existente na tabela.

2.2.2 Função AVG()

A função *AVG()* computa a média entre os valores de uma coluna de dados. Assim como a função *SUM()*, esta função deve ser aplicada sobre dados numéricos. Porque esta função direciona a soma dos valores, da coluna informada, para uma coluna temporária e divide o resultado da soma pela quantidade de valores que foram usadas na soma. Esse resultado final pode ser de um tipo diferente do tipo da coluna usada. Por exemplo, se você aplicar AVG em uma coluna onde só contenha inteiros, o resulta será *decimal*, isso porque ocorre uma divisão antes de finalizar o processo desta função, vamos observar o exemplo abaixo:

```
select avg(estoque_max) as media_qtdemax
from tbl_produto;
```

Neste caso será mostrada a média do campo ESTOQUE_MAX existente na tabela.

2.2.3 Função MIN()

A função MIN serve para obtermos o valor mínimo existente em uma determinada coluna, vamos observar o exemplo abaixo:

```
select min(valor) as vlrmin_preco
from tbl_produto;
```

Neste caso será mostrado o mínimo valor do campo VALOR existente na tabela.

2.2.4 Função MAX()

A função MAX serve para obtermos o valor máximo existente em uma determinada coluna, vamos observar o exemplo abaixo:

```
select max(valor) as vlrmax_preco
from tbl_produto;
```

Neste caso será mostrado o máximo valor do campo VALOR existente na tabela.

2.3 Operações matemáticas no *SELECT*

Podemos realizar “contas” no próprio SELECT para agilizar o processo de consulta, vejamos os exemplos:

```
select qtde * i_valor as valor_produto
from tbl_itempedido
where i_cod_produto = 2
  and i_cod_pedido = 1
```

Neste caso será mostrado o resultado da multiplicação dos dados das colunas VALOR e QTDE.

2.4 Prática de *Querys* SQL.

Com o objetivo de aprimorar e exercitar o conteúdo visto, vamos simular que a gestão de nossa loja queira algumas informações de nosso banco de dados, tudo isso, com o objetivo de verificar se o negócio está em bom andamento, ou, de mal a pior, para isso, é pedido para nós algumas informações como vamos ver abaixo:

1º - Queremos fazer uma panfletagem na região, e gostaria de saber quantas ruas temos nos municípios que atendemos, mas quero saber por cidade, para termos a ideia de quantas pessoas precisaremos para isso.

2º - Qual é o bairro que possui mais ruas? Queria saber para colocar mais gente trabalhando na panfletagem nestes bairros.

3º - Gostaria de saber quem são os clientes que nasceram entre as décadas de 60 e 90.

4º - Qual o mês que temos mais clientes nascidos?

5º - Qual o mês que temos mais pedidos realizados?

6º - Gostaria de saber o menor preço que vendi.

7º - Gostaria de saber o código do produto que mais vendi.

8º - Gostaria de saber a soma do meu estoque atual.

9º - Gostaria de saber a média do meu estoque atual.

10º – Gostaria de saber o valor total do meu estoque atual.

Pessoal, desenvolvam estas ***querys*** para que possamos validar na próxima aula.

2.4.1. Correção da prática de *Querys* SQL.

Vamos fazer uma correção da prática anterior, vale lembrar, que temos outras formas de atingirmos o mesmo resultado, esta é uma das formas:

01 - Correção desafio 10 querys

Limit to 50000 rows

```
1   #CORREÇÃO DE EXERCÍCIOS PRÁTICOS
2   #NÚMERO 01
3 • select count(*) as qtde, cidade
4   from tbl_endereco
5   group by cidade;
6
7   #NÚMERO 02
8 • select count(*) as qtde, bairro
9   from tbl_endereco
10  group by bairro
11  order by 1 desc
12  limit 5;
13
14  #NÚMERO 03
15 • select * from tbl_cliente
16  where year(data_nasc) between 1960 and 1999;
17
18  #NÚMERO 04
19 • select count(*) as qtde, month(data_nasc) mes
20  from tbl_cliente
21  group by 2
22  order by 1 desc;
```

```
23
24   #NÚMERO 05
25 • select count(*) as qtde, month(data_pedido)
26   from tbl_pedido
27   group by 2
28   order by 2 desc;
29
30   #NÚMERO 06
31 • select min(i_valor) as valor_minimo
32    from tbl_itempedido;
33
34   #NÚMERO 07
35 • select sum(qtde) as qtde, i_cod_produto
36   from tbl_itempedido
37   group by 2
38   order by 1  desc;
39
40   #NÚMERO 08
41 • select sum(estoque_atual) as estoque_atual
42   from tbl_produto;
43
44   #NÚMERO 09
45 • select avg(estoque_atual) as media_estoque
46   from tbl_produto;
47
48   #NÚMERO 10
49 • select sum(estoque_atual * valor) as valor_estoque
50   from tbl_produto;
```

2.5 *Joins* de tabelas

Quando queremos relacionar mais de uma tabela no *SELECT*, podemos contar com alguns tipos de JOINS que veremos a seguir:

2.5.1 *Inner Join*

É o método de junção mais conhecido e, pois retorna os registros que são comuns às duas tabelas, precisamos fazer a amarração das tabelas com as chaves respectivas, veja os exemplos:

```
select *
from tbl_cliente as c
inner join  tbl_pedido as p
on c.cod_cliente = p.cod_cliente
```

Ou

```
select *
from tbl_cliente as c, tbl_pedido as p
where c.cod_cliente = p.cod_cliente
```

Ambos darão o resultado:

cod_cliente	nome_cliente	cpf	data_nasc	cep	numero	complemento	cod_pedido	data_pedido	data_entrega	cod_cliente
1	Marcos Costa de Sousa	12345678901	1981-02-06	6768100	1525	apto 166C	1	2019-01-01	2019-01-04	1
2	Zoroastro Zoando	01987654321	1989-06-15	6757190	250		2	2019-01-05	2019-01-07	2
3	Idelbrandolâncio Silva	54698721364	1974-09-27	6753001	120		3	2019-01-12	2019-01-15	3
5	Conegunda Prado	54781269501	1950-10-06	6753020	50	apto 166C	4	2019-01-15	2019-01-16	5
7	Iscruência da Silva	12457965823	1974-11-25	6803040	5		5	2019-01-20	2019-01-22	7

2.6.2 *Left Join*

Tem como resultado todos os registros que estão na tabela A (mesmo que não estejam na tabela B) e os registros da tabela B que são comuns à tabela A, precisamos fazer a amarração das tabelas com as chaves respectivas, veja os exemplos:

```
select *
from tbl_cliente as c left join tbl_pedido as p
on c.cod_cliente = p.cod_cliente
```

O resultado será esse:

cod_cliente	nome_cliente	cpf	data_nasc	cep	numero	complemento	cod_pedido	data_pedido	data_entrega	cod_cliente
1	Marcos Costa de Sousa	12345678901	1981-02-06	6768100	1525	apto 166C	1	2019-01-01	2019-01-04	1
2	Zoroastro Zoando	01987654321	1989-06-15	6757190	250		2	2019-01-05	2019-01-07	2
3	Idelbrandolâncio Silva	54698721364	1974-09-27	6753001	120		3	2019-01-12	2019-01-15	3
5	Conegunda Prado	54781269501	1950-10-06	6753020	50	apto 166C	4	2019-01-15	2019-01-16	5
7	Iscruência da Silva	12457965823	1974-11-25	6803040	5		5	2019-01-20	2019-01-22	7
4	Cosmólio Ferreira	41368529687	1966-12-01	6753020	25	apto 255 F	NULL	NULL	NULL	NULL
6	Brogundes Asmônio	41256398745	1940-05-10	6753400	100		NULL	NULL	NULL	NULL
8	Zizafânio Zizundo	54123698562	1964-08-14	6803140	25		NULL	NULL	NULL	NULL
9	Ricuerda Zunda	21698534589	1934-10-14	6803045	123		NULL	NULL	NULL	NULL
10	Aninoado Zinzão	25639856971	1976-12-25	6803070	50		NULL	NULL	NULL	NULL

2.6.3 *Right Join*

Teremos como resultado todos os registros que estão na tabela B (mesmo que não estejam na tabela A) e os registros da tabela A que são comuns à tabela B, precisamos fazer a amarração das tabelas com as chaves respectivas, veja os exemplos:

```
select *
from tbl_pedido as p right join tbl_cliente as c
on p.cod_cliente = c.cod_cliente
```

O resultado será:

cod_pedido	data_pedido	data_entrega	cod_cliente	cod_cliente	nome_cliente	cpf	data_nasc	cep	numero	complemento
1	2019-01-01	2019-01-04	1	1	Marcos Costa de Sousa	12345678901	1981-02-06	6768100	1525	apto 166C
2	2019-01-05	2019-01-07	2	2	Zoroastro Zoando	01987654321	1989-06-15	6757190	250	
3	2019-01-12	2019-01-15	3	3	Idelbrandolâncio Silva	54698721364	1974-09-27	6753001	120	
4	2019-01-15	2019-01-16	5	5	Conegunda Prado	54781269501	1950-10-06	6753020	50	apto 166C
5	2019-01-20	2019-01-22	7	7	Iscruência da Silva	12457965823	1974-11-25	6803040	5	
NULL	NULL	NULL	NULL	4	Cosmólio Ferreira	41368529687	1966-12-01	6753020	25	apto 255 F
NULL	NULL	NULL	NULL	6	Brogundes Asmônio	41256398745	1940-05-10	6753400	100	
NULL	NULL	NULL	NULL	8	Zizafânio Zizundo	54123698562	1964-08-14	6803140	25	
NULL	NULL	NULL	NULL	9	Ricuerda Zunda	21698534589	1934-10-14	6803045	123	
NULL	NULL	NULL	NULL	10	Aninoado Zinzão	25639856971	1976-12-25	6803070	50	

3. VIEWS (Visualizações).

3.1 Introdução.

"**Views**" também conhecidas em português como visualizações nada mais são do que tabelas virtuais que são criadas a partir de seleções de dados, essas tabelas virtuais reúnem apenas os campos de uma ou mais tabelas físicas que são listados em uma instrução "**SELECT**" associada a "**VIEW**", diferente das tabelas físicas "reais" que são armazenadas em disco e são permanentes, ou seja se o servidor de dados for reinicializado tanto as tabelas quanto o seus dados permanecerão em disco as "**VIEWS**" são, como já foi dito virtuais, logo seus dados permanecem armazenados na memória principal do servidor de dados, isso traz algumas vantagens a mais óbvia é o acesso extremamente rápido a esses dados, logo podemos afirmar que que as "**VIEWS**" são um recurso altamente recomendado em casos onde essas seleções de dados são muito requisitadas o caso mais comum de aplicação é em

relatórios, e sendo consideradas como “tabelas” tudo que vimos em DQL pode ser usado em VIEWS.

Abaixo podemos ver a sintaxe de como criar uma “**view**”:

CREATE VIEW <NOME_DA_VIEW> **AS** <CONSULTA>;

Para praticarmos um pouco vamos criar três “**views**” em nossa base de dados “**VENDAS**” é necessário que a massa de dados que pedi no início do semestre esteja feita. Uma vez feito isso vamos criar três “views”, observe as imagens abaixo:

A primeira “**view**” lista clientes e pedidos:

```
#1ª VIEW - RELAÇÃO DE CLIENTE COM PEDIDO
create view vw_cliped as
   select c.cod_cliente as codigo, c.nome_cliente as nome, p.cod_pedido as pedido,
          p.data_pedido as data_requisicao
   from tbl_cliente c, tbl_pedido p
   where c.cod_cliente = p.cod_cliente;
```

A segunda “**view**” lista clientes, pedidos e itens dos pedidos:

```
#2ª VIEW - RELACIONA CLIENTES, PEDIDOS E ITEM DOS PEDIDOS
create view vw_clipedprod as
   select c.cod_cliente as codigo, c.nome_cliente as nome, p.cod_pedido as pedido,
          p.data_pedido as data_requisicao,
          i.i_cod_produto as produto, i.qtde
   from tbl_cliente c, tbl_pedido p, tbl_itempedido i
   where c.cod_cliente = p.cod_cliente
     and i.i_cod_pedido = p.cod_pedido;
```

A terceira “**view**” lista clientes, pedidos, itens dos pedidos e descrição dos produtos:

```
#3ª VIEW - RELACIONA CLIENTES, PEDIDOS, ITENS DOS PEDIDOS E PRODUTOS
create view vw_prodtudototal as
   select c.cod_cliente as codigo, c.nome_cliente as nome, p.cod_pedido as pedido,
          p.data_pedido as data_requisicao,
          i.i_cod_produto as produto, pr.nome_produto as descricao, i.qtde,
          pr.valor, i.qtde * pr.valor as totalcomprado
   from tbl_cliente c, tbl_pedido p, tbl_itempedido i, tbl_produto pr
   where c.cod_cliente = p.cod_cliente
     and i.i_cod_pedido = p.cod_pedido
     and i.i_cod_produto = pr.cod_produto;
```

Para visualização de “VIEWS” usamos o comando SELECT convencional, e para apagar uma “VIEW” usamos o comando DROP VIEW.