4. SQL: Comandos para manipulação de dados (DML)

A linguagem SQL é basicamente dividida em três tipos de comandos:

SQL = DDL + DML + DCL

- DDL (definição de dados): Comandos: CREATE, DROP, ALTER
- DML (manipulação de dados): Comandos: SELECT, INSERT,UPDATE e DELETE
- DCL(controle de dados): Comandos: GRANT e REVOKE

Os comandos de manipulação de dados(DML) em SQL são representados por:

- **INSERT:** permite a inclusão de novas linhas nas tabelas
- **UPDATE:** altera os valores de dados já cadastrados
- **DELETE:** remove dados já cadastrados
- **SELECT:** usado para consultar o BD e retornar dados que satisfazem a determinada expressão em um comando

4.1. Comando INSERT

O comando INSERT permite inserir uma linha de dados na tabela e possui a seguinte sintaxe abaixo:

```
INSERT INTO NOME DA TABELA(coluna1,coluna2,coluna3) VALUES
(valor1, valor2, valor3)
```

Exemplos:

```
INSERT INTO Cliente(codigo,nome,sexo)
VALUES("200810", "Regilan Meira", "Masculino")
```

```
INSERT INTO Disciplina(codigo,nome,ementa)
VALUES("01", "Banco de Dados", "DER,Modelo Relacional,SQL")
```

4.2. Comando UPDATE

O comando UPDATE é usado para mudar valores de linhas de dados que já foram cadastrados anteriormente e que obedecem a determinados critérios, especificados em condições. Este comando pode alterar mais de uma linha ao mesmo tempo, caso mais de uma linha obedeça a determinada condição. As condições podem também ser representadas utilizando os operadores: AND,OR e NOT

O comando UPDATE, contém a cláusula WHERE, de forma a restringir o conjunto dos registros que serão processados pelo comando. **Se não for colocada a cláusula WHERE no comando UPDATE, as alterações serão realizadas em todos os registros da tabela.**

Sintaxe:

```
UPDATE NOME DA TABELA
SET coluna1 = valor1, coluna2 = valor2
WHERE condições
```

Exemplos:

```
UPDATE  Avaliacao SET media = 10
```

```
UPDATE  Avaliacao SET media = 10
WHERE nome_aluno = "João"
```

```
UPDATE Compras SET preco = 105,
forma_pagamento = "Cartão de Crédito"
WHERE numero_compra = "2008708"
```

Para melhor exemplificar o funcionamento do comando UPDATE, veremos as seguintes situações:

Situação 01: Aumentar o salário de todos os funcionários em 10%

Como se pretende aumentar o salário de todos os elementos da tabela FUNCIONÁRIO, o comando UPDATE não usará a cláusula WHERE.

```
UPDATE Funcionario SET salario = salario * 1.1
```

Situação 02: Aumentar o salário do funcionário Regilan Meira e adicionar 1 ano ao tempo de serviço.

Nessa situação, estamos restringindo a atualização para o funcionário REGILAN MEIRA, sendo assim faz-se necessário o uso da cláusula WHERE.

```
UPDATE Funcionario SET salario = salario * 1.1, idade =
idade + 1
WHERE nome = "Regilan Meira"
```

Situação 03: Adicionar o prefixo 55 ao telefone de todos os hospedes que residem no Brasil

Nessa situação, estamos restringindo a atualização para indivíduos Brasileiros, sendo assim faz-se necessário o uso da cláusula WHERE.

```
UPDATE Hospedes SET Telefone = "55" + Telefone
WHERE pais = "Brasil"
```

Situação 04: Adicionar R$ 150 no salário das mulheres que possuem filhos.

Nessa situação, estamos restringindo a atualização dos dados para duas condições. Sendo assim, utilizaremos a cláusula **WHERE** e o operador **AND**.

```
UPDATE Funcionarios SET Salario = Salario + 150
WHERE Sexo = "F" and Filhos > 0
```

Situação 05: Adicionar R$ 150 no salário das mulheres que possuem filhos, ou homens que são casados.

Nessa situação, utilizaremos a cláusula **WHERE** , juntamente com o operador **AND e OR**.

```
UPDATE Funcionarios SET Salario = Salario + 150
WHERE (Sexo = "F" and Filhos > 0) OR (Sexo = "M" and
EstadoCivil = "Casado")
```

Situação 06: Conceder desconto de 5% nos preços dos veículos que possuírem cor diferente de preto e branco.

Nessa situação, utilizaremos a cláusula WHERE , juntamente com o operador AND.

```
UPDATE Veiculos SET Preco = Preco – Preço * 0.05
WHERE Cor <> "Branco" AND Cor <> "Preto"
```

Situação 07: Conceder desconto de 5% nos preços dos produtos a base de leite.

Nessa situação, utilizaremos o operador LIKE. O operador LIKE permite fazer comparações de partes da string. Para isso utilizaremos dois curingas (“ % “)

```
UPDATE Produto SET Preco = Preco – Preço * 0.05
WHERE Nome Like “Leite%”
```

4.3. Comando DELETE

O comando DELETE é usado para remover linhas de uma tabela. Este comando pode remover mais de uma linha ao mesmo tempo, caso mais de uma linha obedeça a uma certa condição. As condições podem ser representadas utilizando os operadores AND, OR e NOT.

O comando DELETE, contém a cláusula WHERE, de forma a restringir o conjunto dos registros que serão processados pelo comando. Se não for colocada a cláusula WHERE no comando DELETE, serão apagados todos os registros de uma tabela.

Assim como no comando UPDATE, podemos utilizar os operadores relacionais (>,>=,<,<=, =, <>, like) e os operadores lógicos(AND, OR) para especificar as condições de exclusão de dados.

Sintaxe:

```
DELETE FROM NOME DA TABELA
WHERE <condições>
```

Exemplos:

```
DELETE FROM ESCOLA
```

```
DELETE FROM ESCOLA WHERE ALUNO = “TIAGO PEREIRA”
```

```
DELETE FROM PRODUTOS
WHERE NOME Like “LEITE%”
```

```
DELETE FROM CLIENTES
WHERE QuantidadeCompras <= 3
```

4.4. Comando SELECT

O comando SELECT é usado para consultar o banco de dados e retornar dados recuperados que satisfazem a determinada condição expressa no comando.

Sua sintaxe é representada da seguinte forma:

```
SELECT <lista de atributos>
FROM NOME DA TABELA
WHERE <condições>
```

Exemplos:

```
SELECT codigo,aluno,media
FROM NOTAS
WHERE aluno = "Tiago"
```

```
SELECT matricula,nome,responsavel,data_nascimento,cpf,rg,
endereco,codigo_curso,observacoes FROM ALUNOS
WHERE nome = "Camila"
```

```
SELECT * FROM ALUNOS
WHERE nome = "Camila"
```

Obs: o símbolo * na clausula SELECT indica que deverá ser selecionado todos os campos de uma tabela.

Para melhor exemplificar o funcionamento do comando SELECT, veremos as seguintes situações:

Situação 01: Escrever o comando SQL que permite obter o RG, Nome e o Código Postal de todos os clientes registrados no banco de dados.

```
SELECT Rg, Nome, CodigoPostal
FROM Cliente
```

Situação 02: Selecionar todos os dados de todos os pacientes cadastrados no Hospital

Nesta situação usaremos o curinga(*) , ao invés de escrever todos os campos da tabela Paciente no comando SQL.

```
SELECT *
FROM Pacientes
```

Situação 03: Selecionar o ID, Nome, Idade e o Salário de todos os Funcionários com Idade entre 30 e 40 anos

Nesta situação usaremos o operador WHERE, juntamente com operadores lógicos e relacionais.

```
SELECT Id,Nome,Idade,Salario
FROM Funcionario
WHERE Idade >= 30 AND Idade <= 40
```

Situação 04: Selecionar o ID, Nome, Idade e o Salário de todos os Funcionários cuja a idade não está entre 30 e 40 anos.

Nesta situação usaremos o operador WHERE, juntamente com operadores lógicos (AND e NOT) e relacionais.

```
SELECT Id,Nome,Idade,Salario
FROM Funcionario
WHERE Not(Idade >= 30 AND Idade <= 40)
```

Situação 05: Selecionar todos os indivíduos que possuem sobrenome "Silva"

Nesta situação usaremos o operador Like e o curinga "%"

```
SELECT *
FROM Pessoa
WHERE Nome Like "%Silva%"
```

Situação 06: Selecionar a quantidade de votos dos Partidos: PT, PSDB, PSB e DEM nas eleições de 2012.

```
SELECT QuantidadeVotos
FROM Votos
WHERE (Partido = "PT" OR Partido = "PSDB" OR Partido =
"PSB" OR Partido = "DEM") AND AnoEleicao = 2012
```

Podemos incluir no comando SQL uma cláusula que permita solicitar a ordenação dos resultados fornecidos por um comando SELECT. A ordenação é feita da seguinte forma: primeiro os dígitos, depois os caracteres maiúsculos e por último os caracteres minúsculos.

Representação:

0 < 1 < ... < 8 < 9 < ... < A < B < .... < Z < ... a < b < ... < z

A ordenação pode ser realizada através da cláusula **ORDER BY** no comando SELECT. Esta cláusula aparece sempre posicionada no final do comando SELECT.

Veja a sintaxe:

```
SELECT Campo1,Campo2,Campo3
FROM Tabela
WHERE Condição
ORDER BY Campo ASC | DESC
```

- ASC indica ordenação ASCendente
- DESC indica ordenação DESCendente

Para melhor compreensão, veremos os exemplos abaixo:

Situação 07: Selecionar todos os dados da tabela Pessoa, ordenado pela Idade.

```
SELECT *
FROM Pessoa
ORDER BY Idade
```

OBS: Quando a ordenação for ASCendente não é necessário incluir a cláusula ASC em ORDER BY, já que a ordenação padrão é ascendente

Situação 08: Selecionar o Nome e o Salário de todas as pessoas, ordenando o resultado pelo Salário, de tal forma que os maiores salários fiquem no topo da lista.

Nesta situação usaremos a cláusula DESC de maneira que os maiores salários fiquem no topo da lista.

```
SELECT Nome,Salario
FROM Pessoa
ORDER BY Salario DESC
```

Situação 09: Resolver o mesmo problema anterior, mas só para as pessoas com Cargo de Assistente Administrativo.

Nesta situação usaremos a cláusula WHERE, antes da ordenação.

```
SELECT Nome,Salario
FROM Pessoa
WHERE Cargo = "Assistente Administrativo"
ORDER BY Salario DESC
```

Situação 10: Selecionar todos os dados da tabela Pessoa, ordenado pela Idade e pelo Salário.

```
SELECT *
FROM Pessoa
ORDER BY Idade,Salario
```

Situação 11: Selecionar da tabela Pessoa o Nome e a Idade que irão ter daqui a dois anos. O resultado deverá vir pela nova idade.

```
SELECT Nome, Idade + 2 as Nova_Idade
FROM Pessoa
ORDER BY Idade + 2
```

Situação 12: Selecionar na tabela Vendas, a Nota Fiscal, o campo Valor, o montante do Imposto (17%) do Valor, e 0 Valor total com Imposto.

É possível também incluir na cláusula de ordenação o número da posição, dentre as colunas a serem apresentadas, da coluna ou colunas pelas quais se pretende ordenar o resultado.

```
SELECT Nota_Fiscal, Valor, Valor * 0.17 As Imposto, Valor +
Valor * 0.17 as Valor_Total
FROM VENDAS
ORDER BY 1,4
```

OBS: 1 está se referindo ao campo Nota_Fisca, 4 está se referindo ao valor total da venda(valor + imposto)

Situação 13: Selecionar o conjunto das Localidades existente na tabela Postal. OBS: Não exibir localidades iguais.

A cláusula DISTINCT permite eliminar repetições de linhas no resultado de um comando SELECT. A cláusula DISTINCT só pode ser colocada imediatamente depois do SELECT.

```
SELECT DISTINCT Localidade
FROM Postal
```

É possível incluir em comandos SELECT as funções de agragação. As funções de agregação (funções estatísticas) têm por objetivo obter informações sobre o conjunto de linhas especificados na cláusula **WHERE** ou sobre grupos de linhas indicados na cláusula **GROUP BY**.

As funções disponíveis são:

- COUNT: número de linhas
- MAX: o maior valor da coluna
- MIN: o menor valor da coluna
- SUM: soma de todos os valores da coluna
- AVG: média de todos os valores da coluna

Para melhor compreensão veremos as seguintes situações:

Situação 14: Quantos Funcionários existem e quantos têm telefone fixo.

```
SELECT COUNT(Nome) AS Total_Funcionario, COUNT(Telefone) as
Total_Funcionario_Com_Telefone FROM Funcionarios
```

Situação 15: Quantos Funcionários existem em nosso Banco de Dados

```
SELECT COUNT(*) AS Total
FROM Funcionarios
```

Situação 16: Qual o maior salário e a menor idade dos funcionários da empresa.

```
SELECT MAX(Salario) AS Maior_Salario, MIN(Idade) AS
Menor_Idade FROM Funcionarios
```

Situação 17: Qual o menor e maior valor de comissão superior a 1000 e inferior a 10000 na tabela Venda

```
      SELECT MAX(Comissao) AS Maior_Comissao, MIN(Comissao) AS
Menor_Comissao
      FROM Venda
      WHERE Comissao >= 1000 AND Comissao <= 10000
```

Situação 18: Qual é o valor total das comissões a pagar?

```
      SELECT SUM(Comissao) AS Total_Comissao
      FROM Venda
```

Situação 19: Qual é o salário médio dos Funcionários com mais de 40 anos.

```
      SELECT AVG(Salario) AS Media_Salario
      FROM Funcionario
      WHERe Idade > 40
```

O comando SELECT também permite agrupar resultados. As cláusulas de agrupamento estão relacionadas com as funções de agregação, e são úteis no tratamento de informações de forma agrupada. As cláusulas **GROUP BY** e **HAVING** fazem parte do comando SELECT e são representadas de acordo com a sintaxe a seguir:

```
      SELECT Campos
      FROM Tabela
      WHERE Condição
      GROUP BY ...
      HAVING ...
```

A cláusula GROUP BY divide o resultado de um SELECT em um grupo de resultados que serão processados pelas funções de agregação.

Situação 20: Mostrar a quantidade de CARROS vendidos agrupados pelo Modelo.

```
      SELECT Modelo_Carro, Count(Nota_Fiscal) AS
Quantidade_Carros_Vendidos
      FROM Vendas
      GROUP BY Modelo_Carro
```

Situação 21: Mostrar para cada funcionário o valor total das vendas realizadas.

```
SELECT Nome, Sum(Valor) AS Total_Vendas
FROM Vendas
GROUP BY Nome
```

A cláusula **HAVING** faz restrições ao nível dos grupos que são processados. É comum surgir a dúvida sobre **WHERE** ou **HAVING**. A diferença entre HAVING e WHERE é que a cláusula WHERE é usada para restringir os registros a serem considerados na seleção. A Cláusula HAVING restringe os grupos que foram formados depois da aplicação da cláusula WHERE.

Situação 22: Mostrar para cada funcionário o valor total das vendas superiores a 80000

```
SELECT Nome, Sum(Valor) AS Total_Vendas
FROM Vendas
GROUP BY Nome
HAVING Sum(Valor) > 80000
```

Situação 23: Selecionar a quantidade de cada produto vendido em uma compra, ordenados em ordem descendente.

```
SELECT Codigo_Produto,Count(Codigo_Produto) AS QUANTIDADE
FROM Compra
GROUP BY Codigo_Produto
ORDER BY 2 DESC
```

4.4.1. Comando SELECT: ligação entre tabelas e subsconsultas

A ligação entre tabelas (join) permite extrair, através de um único SELECT, informações contidas em diferentes tabelas.

A junção entre tabelas é feita colocando-se na cláusula FROM, as tabelas que pretende-se juntar. Veja a sintaxe:

```
SELECT Campo1, Campo2, Campo3, CampoN
FROM Tabela1,Tabela2
```

A estrutura mostrada acima representa o produto cartesiano entre as tabelas, que associa a cada linha de uma tabela, o conjunto de linhas de outra tabela.

O **Inner Join** (ligação entre tabelas), ocorre quando se juntam duas ou mais tabelas, ligando-as através da Chave Primária de uma e da Chave Estrangeira da outra. Num Inner Join são exibidos apenas os registros em que exista ligação entre as tabelas. A junção de duas ou mais tabelas, é feita através das chaves estrangeiras, na cláusula WHERE.

Considere o seguinte comando:

```
SELECT Pessoa.Nome,Postal.Cod_Postal,Postal.Localidade
FROM Pessoa,Postal
WHERE Pessoa.Cod_Postal = Postal.Codigo
```

O comando acima retornará o Nome da Pessoa (tabela Pessoa), o código Postal e a Localidade (tabela Postal), quando o código Postal que existe na tabela PESSOA for igual ao código Postal existente na tabela Postal.

Ou seja, só será mostrado os dados da tabela Postal quando eles forem relacionados com a tabela Pessoa.

Para melhor compreensão veremos as seguintes situações:

Situação 24: Selecionar as localidades das pessoas que não tem telefone.

```
SELECT Postal.Localidade
FROM Pessoa,Postal
WHERE Pessoa.Cod_Postal = Postal.Codigo AND Pessoa.Telefone
IS NULL
```

OBS: A utilização do nome da tabela antes do nome da coluna não é obrigatória (Pessoa.Telefone). Somente quando uma coluna, tem o mesmo nome em várias tabelas, é necessário essa identificação.

Situação 25: Selecionar a data da venda, o valor e o nome do vendedor.

A tabela Venda e Vendedor estão relacionadas através das chaves estrangeiras. (Venda.CPFVendedor = Vendedor.CPF)

```
SELECT Venda.Data,Venda.Valor,Vendedor.Nome
FROM Venda,Vendedor
WHERE Venda.CPFVendedor = Vender.CPF
```

Situação 26: Em um clube, existe sócios titulares e dependentes. Selecionar o nome e a data de nascimento de todos os dependentes. Incluir na consulta o nome do titular de cada um dos dependentes. Ordenar o resultado pelo nome dos Dependentes.

A tabela Titular e Dependente estão relacionadas através das chaves estrangeiras. (Titular.Matricula = Dependente.MatriculaTitular)

```
      SELECT
Titular.Nome,Dependente.Nome,Dependente.Data_Nascimento
      FROM Titular,Dependente
      WHERE Titular.Matricula = Dependente.MatriculaTitular
      ORDER BY Dependente.Nome
```

É possível ainda em alguns sistemas, escrever a consulta acima usando o comando **INNER JOIN**.

```
      SELECT
Titular.Nome,Dependente.Nome,Dependente.Data_Nascimento
      FROM Titular INNER JOIN Dependente ON Titular.Matricula =
Dependente.MatriculaTitular
      ORDER BY Dependente.Nome
```

Situação 27: Selecionar todos os produtos vendidos pelo vendedor de nome Fulano de Tal

Essa relação envolve 4 tabelas: Venda, Itens_Da_Venda, Produto e Vendedor. Na consulta SELECT deve-se especificar todos os relacionamentos entre as tabelas.

```
      SELECT Produto.Nome
      FROM Venda,Itens_Da_Venda,Produto,Vendedor
      WHERE Vendedor.Nome = 'Fulano de Tal'
      AND Venda.MatriculaVendedor = Vendedor.Matricula
      AND Venda.Codigo = Itens_Da_Venda.Codigo_Venda
      AND Itens_Da_Venda.Codigo_Produto = Produto.Codigo
```

Utilizando SQL, podemos ainda utilizar o comando de união (UNION) que permite juntar o conteúdo de dois ou mais comandos SELECT.

Situação 28: Em uma base de dados, o cadastro de todos os alunos, professores e funcionários estão localizados em tabelas diferentes. Como obter o CPF e o Nome, de todos os indivíduos cadastrados na base dados?

Para obter os dados solicitados no slide anterior, será necessário criar 3 comandos SELECT e usar o comando UNION para unir as consultas. Veja o resultado:

```
SELECT cpf,nome FROM Professor
UNION
SELECT cpf,nome FROM Funcionario
UNION
SELECT cpf,nome FROM Aluno
```

Em uma união (UNION), o número de campos a serem selecionados em cada um dos comandos SELECT tem de ser IGUAL. O nome das colunas apresentado no resultado é o nome das colunas selecionadas na primeira instrução SELECT. Além disso, são eliminadas do resultado as linhas duplicadas, do mesmo modo que no DISTINCT, a não ser que seja utilizado UNION ALL.

Situação 29: Obter o CPF e o Nome de todos os Funcionários, Professores e Alunos cuja a cidade natal seja Ilhéus. Por fim ordenar o resultado pela coluna Nome.

Cada comando SELECT pode conter sua própria cláusula WHERE. Porém, só poderá existir uma única cláusula ORDER BY, que estará localizada no último comando SELECT, e será aplicada a todo o resultado. Veja a resolução:

```
SELECT cpf,nome FROM Professor WHERE naturalidade = 'Ilhéus
UNION
SELECT cpf,nome FROM Funcionario WHERE naturalidade =
'Ilhéus'
UNION
SELECT cpf,nome FROM Aluno WHERE naturalidade = 'Ilhéus'
ORDER BY nome
```

O operador **INTERSECT** permite juntar o resultado de dois comandos SELECT, apresentando as linhas que resultam de ambos os comandos. As linhas duplicadas são eliminadas, a não ser que seja utilizado INTERSECT ALL.

Situaçã0 30: Selecionar o nome, cpf e data de nascimento de todos os funcionários que são alunos da universidade onde trabalham. Exibir os dados ordenados de forma decrescente pela coluna nome.

```
SELECT nome,cpf,data_nascimento FROM Funcionario
INTERSECT
SELECT nome,cpf,data_nascimento FROM Aluno
ORDER BY nome DESC
```

O operador **EXCEPT** retorna todas as linhas presentes no resultado da consulta1, mas que não estão presentes no resultado da consulta2 (às vezes isto é chamado de diferença entre duas consultas).

As linhas duplicadas são eliminadas a não ser que seja utilizado EXCEPT ALL.

Situaçã0 31: Selecionar o nome, cpf e data de nascimento de todos os funcionários que não são alunos da universidade onde trabalham. Exibir os dados ordenados de forma decrescente pela coluna nome.

```
SELECT nome,cpf,data_nascimento FROM Funcionario
EXCEPT
SELECT nome,cpf,data_nascimento FROM Aluno
ORDER BY nome DESC
```

O comando SQL permite o uso de subconsultas. Uma subconsulta consiste em um SELECT dentro de outro SELECT.

Considere o seguinte problema:

Situaçã0 32: Qual é o nome da Pessoa com menor salário na empresa?

Para conseguir resolver a consulta acima, será necessário resolver dois problemas:

1. Qual é o valor do menor salário?
2. Qual o nome da pessoal a que esse salário corresponde?

Para resolver o primeiro problema (Qual é o valor do menor salário?) podemos usar a função de agregação MIN.

```
SELECT MIN(salario) From Funcionario
```

Com base no valor retornado acima, podemos realizar a consulta para o outro problema (Qual o nome da pessoal a que esse salário corresponde?). Considere que a primeira consulta resultou um salário de R$ 698.

```
SELECT Nome FROM Funcionario
WHERE salario = 698
```

Podemos obter o mesmo resultado através de comandos SELECT encadeados.

```
SELECT Nome FROM Funcionario
WHERE salario = (SELECT MIN(salario) FROM Funcionario)
```

Com o comando acima, o salário de cada um dos indivíduos existentes na tabela Pessoa é comparado diretamente com o resultado obtido do comando SELECT interior.

Sendo assim, primeiro é executado a consulta interior (SELECT MIN (salario) FROM Funcionarios), e em seguida a consulta exterior pode ser executada pelo resultado que o SELECT interior devolveu.

Situação 33: Qual o nome das pessoas cujo salário é menor que a soma de suas comissões durante o mês de Janeiro/2013.

```
SELECT Nome, Salario FROM Funcionario
WHERE salario < (SELECT SUM(valor) FROM Comissao WHERE
Mês = 1 and ano = 2013 and Comissao.CPF = Funcionario.CPF)
```

Na situação apresentada acima, o sentido da execução é de fora para dentro, ou seja, o SELECT exterior envia o Salário a fim de ser comparado com o total das Comissões associadas ao CPF a que pertence o Salário.

Quando o SELECT interior depende dos dados que lhe são fornecidos pelo SELECT exterior, damos o nome de Consulta Relacionada.

4.5. Exercícios práticos

a) Considere as seguintes tabelas:

- **Curso** (**Codigo (PK),** Nome, NumeroVagas)
- **Instrutor** (**Codigo (PK)**, Nome, Apelido, Fone, Celular).
- **Horario** (**Codigo (PK)**, Sala, Hora)
- **Ministrado** (DataCurso (PK), CodigoHorario (PK,FK), CodigoCurso (FK) , CodigoInstrutor (FK))

  Através de consultas SQL, faça:

- Mostre todas as tabelas do seu BD
- Insira pelo menos 5 (cinco) registros em cada tabela
- Adicione o campo Autorizado char(1) na tabela cursos
- Mostre o nome de todos os instrutores
- Mostre todos os campos da tabela horário
- Mostre o nome do curso ordenado pela quantidade de vagas