



ENTER

Curso Técnico em Desenvolvimento de Sistemas Online

BANCO DE DADOS II

GEEaD - Grupo de Estudo de Educação a Distância

Centro de Educação Tecnológica Paula Souza

Expediente

*GEEaD – CETEC
GOVERNO DO ESTADO DE SÃO PAULO
EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO
CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS
FUNDAMENTOS DE INFORMÁTICA*

Autor: José Mendes da Silva Neto

Revisão Técnica: Eliana Cristina Nogueira Barion e Lilian Aparecida Bertini

Revisão Gramatical: Juçara Maria Montenegro Simonsen Santos

Editoração e Diagramação: Flávio Biazim

APRESENTAÇÃO

Este material didático do Curso Técnico em Desenvolvimento de Sistemas modalidade EaD foi elaborado especialmente por professores do Centro Paula Souza para as Escolas Técnicas Estaduais – ETECs.

O material foi elaborado para servir de apoio aos estudos dos discentes para que estes atinjam as competências e as habilidades profissionais necessárias para a sua plena formação como Técnicos em Desenvolvimento de Sistemas.

Esperamos que este livro possa contribuir para uma melhor formação e aperfeiçoamento dos futuros Técnicos.

AGENDA 5

**LINGUAGEM DE
CONSULTA DE
DADOS - DQL (DATA
QUERY LANGUAGE)**





MERGULHANDO NO TEMA...

Você agora irá aprender a consultar as informações em um Banco de Dados. Vamos continuar trabalhando com o SGBD MySQL e a parte da SQL chamada DQL, que possui comandos para consultar registros nas estruturas do Banco de Dados.

Assim como nas agendas anteriores, antes de cada comando será apresentada a sua **sintaxe**. Vale ainda lembrar que em linguagem de programação, quando falamos de **sintaxe**, nos referimos à **forma de escrever código fonte (palavras reservadas, comandos, recursos diversos)**. Os conteúdos entre os símbolos <> ou [] encontrados na sintaxe significam que os mesmos devem ser substituídos ou são opcionais, respectivamente. Vamos em frente!!!

Você se lembra do comando select? Agora chegou a vez dele!

Este comando, com certeza, é um dos comandos mais utilizados do SQL. Ele faz parte de uma outra divisão da linguagem SQL, a DQL (Data Query Language), Linguagem de Consulta de Dados, e é utilizado quando precisamos buscar informações no Banco de Dados. Vamos utilizar primeiramente uma sintaxe bem simples:

Sintaxe:

```
select campo_1,  
    ..  
    campo_n
```

```
from <tabela1>
```

tabela de origem das informações

```
[where <condição_1> [and/or]  
    ..  
    <condição_n>]
```

critérios que deverão ser satisfeitos para que o registro seja apresentado no resultado da consulta

```
[order by <campo1> [asc/desc]
```

campo e o tipo de ordem que os registros serão apresentados.

Onde:

ASC: significa que os resultados serão apresentados por ordem ascendente, ou seja, do menor para o maior valor.

DESC: significa que os resultados serão apresentados por ordem descendente, ou seja, do maior para o menor valor.

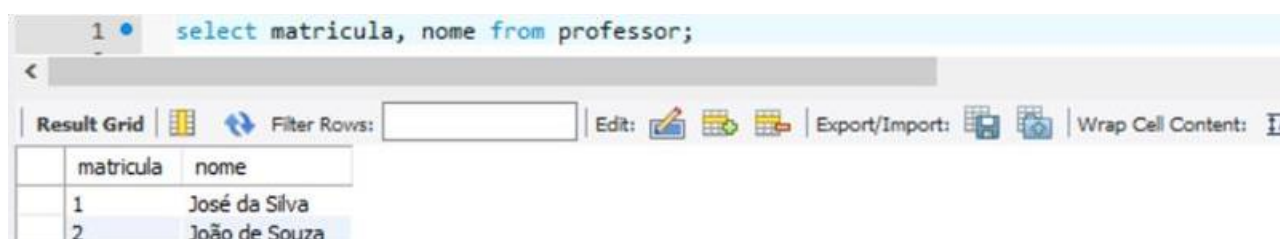
Exemplo 1:

```
select matricula, nome
```

campos que serão apresentados no resultado da consulta

```
from professor;
```

tabela de origem das informações



The screenshot shows the SQL query editor with the query `select matricula, nome from professor;`. Below the editor is the 'Result Grid' tab, which displays the following data:

| | matricula | nome |
|---|-----------|---------------|
| 1 | 1 | José da Silva |
| 2 | 2 | João de Souza |

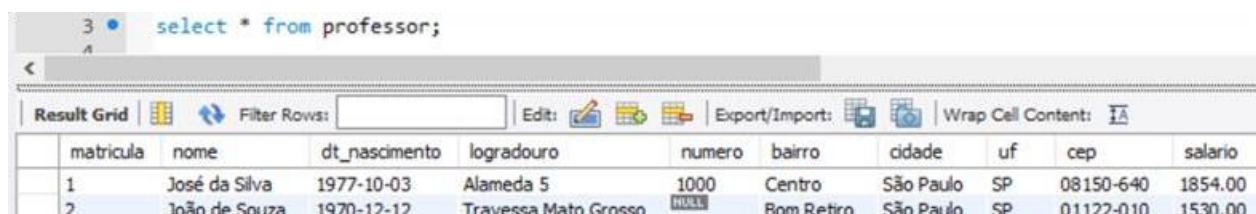
Imagem 04 - Interface Gráfica Workbench – Resultado da consulta do Exemplo 1

No **Exemplo 1**, estamos selecionando apenas os campos `matricula` e `nome` da tabela `professor`, mas, caso você queira selecionar todos os campos da tabela, use o símbolo de asterisco `"*"`. Veja:

Exemplo 2:

```
select *
```

```
from professor;
```



The screenshot shows the SQL query editor with the query `select * from professor;`. Below the editor is the 'Result Grid' tab, which displays the following data:

| | matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|---|-----------|---------------|---------------|----------------------|--------|------------|-----------|----|-----------|---------|
| 1 | 1 | José da Silva | 1977-10-03 | Alameda 5 | 1000 | Centro | São Paulo | SP | 08150-640 | 1854.00 |
| 2 | 2 | João de Souza | 1970-12-12 | Travessa Mato Grosso | 1000 | Bom Retiro | São Paulo | SP | 01122-010 | 1530.00 |

Imagem 05 - Interface Gráfica Workbench – Resultado da consulta do Exemplo 2

Observe a diferença entre os resultados das duas consultas.

A linha de comando do **Exemplo 2** que utilizou o `"*"`, apresentou todos os campos no resultado da consulta, o que não aconteceu com o **Exemplo 1**, onde somente os campos `matricula` e `nome` foram selecionados.

Exemplo 3:

```
select *
from professor
where salario > 1600;
```

critério que deverá ser satisfeito para que o registro seja apresentado no resultado da consulta

| matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|-----------|---------------|---------------|------------|--------|--------|-----------|----|-----------|---------|
| 1 | José da Silva | 1977-10-03 | Alameda 5 | 1000 | Centro | São Paulo | SP | 08150-640 | 1854.00 |

Imagem 06 - Interface Gráfica Workbench – Resultado da consulta do Exemplo 3

Para exemplificar melhor os próximos recursos, vamos incluir mais 3 (três) registros na Tabela **professor** utilizando os conhecimentos obtidos na agenda anterior.

| matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|-----------|---------------|---------------|-------------------------------|--------|---------------|-----------|----|-----------|---------|
| 1 | José da Silva | 1977-10-03 | Alameda 5 | 1000 | Centro | São Paulo | SP | 08150-640 | 1854.00 |
| 2 | João de Souza | 1970-12-12 | Travessa Mato Grosso | 1000 | Bom Retiro | São Paulo | SP | 01122-010 | 1530.00 |
| 3 | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |
| 4 | Marilda Dutra | 1979-01-23 | Rua Adolfo Belini | 621 | São Francisco | São Paulo | SP | 01005-020 | 1480.00 |
| 5 | Acácio Moura | 1985-04-29 | Avenida General Costa e Silva | 908 | Vila Militar | São Paulo | SP | 06442-007 | 1340.00 |

Imagem 07 - Gráfica Workbench – Relação de professores cadastrados

Podemos ainda utilizar outros operadores tais como **between** e **in**, onde:

BETWEEN: quer dizer “entre”. É utilizado para obter intervalos de dados.

IN: é utilizado para obter valores específicos de uma lista.

Exemplo 4:

```
select *
from professor
where salario between 1000 and 1600;
```

| matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|-----------|---------------|---------------|-------------------------------|--------|---------------|-----------|----|-----------|---------|
| 2 | João de Souza | 1970-12-12 | Travessa Mato Grosso | 1000 | Bom Retiro | São Paulo | SP | 01122-010 | 1530.00 |
| 4 | Marilda Dutra | 1979-01-23 | Rua Adolfo Belini | 621 | São Francisco | São Paulo | SP | 01005-020 | 1480.00 |
| 5 | Acácio Moura | 1985-04-29 | Avenida General Costa e Silva | 908 | Vila Militar | São Paulo | SP | 06442-007 | 1340.00 |

Imagem 08 - Interface Gráfica Workbench – Resultado da consulta do Exemplo 4

Exemplo 5:

```
select *
from professor
where matricula in (3, 4);
```

| matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|-----------|---------------|---------------|----------------------|--------|---------------|-----------|----|-----------|---------|
| 3 | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |
| 4 | Marilda Dutra | 1979-01-23 | Rua Adolfo Belini | 621 | São Francisco | São Paulo | SP | 01005-020 | 1480.00 |

Imagem 09 - Interface Gráfica Workbench – Resultado da consulta do Exemplo 5

Qualquer que seja o resultado, você poderá ordená-lo, utilizando a cláusula **order by**, de forma ascendente (**asc**) ou descendente (**desc**).

Exemplo 6:

```
select *
from professor
where matricula in (3, 4)
order by salario;
```

campo definido para ordenação dos registros que serão apresentados no resultado da consulta

Neste exemplo, estão sendo selecionados todos os campos da tabela **professor** onde a **matricula** do **professor** seja **3** ou **4**, por ordem crescente do campo **salario**.

| | matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|--|-----------|---------------|---------------|----------------------|--------|---------------|-----------|----|-----------|---------|
| | 4 | Marilda Dutra | 1979-01-23 | Rua Adolfo Belini | 621 | São Francisco | São Paulo | SP | 01005-020 | 1480.00 |
| | 3 | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |

Imagem 10 - Interface Gráfica Workbench – Resultado da consulta do Exemplo 6

Veja outro exemplo:

Exemplo 7:

```
select *
from professor
order by salario desc;
```

campo e tipo de ordenação definidos para os registros que serão apresentados no resultado da consulta

Neste exemplo, estão sendo selecionados todos os campos da tabela **professor** por ordem decrescente do campo **salario**.

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

| | matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|---|-----------|---------------|---------------|-------------------------------|--------|---------------|-----------|----|-----------|---------|
| 1 | | José da Silva | 1977-10-03 | Alameda 5 | 1000 | Centro | São Paulo | SP | 08150-640 | 1854.00 |
| 3 | | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |
| 2 | | João de Souza | 1970-12-12 | Travessa Mato Grosso | HULL | Bom Retiro | São Paulo | SP | 01122-010 | 1530.00 |
| 4 | | Marilda Dutra | 1979-01-23 | Rua Adolfo Belini | 621 | São Francisco | São Paulo | SP | 01005-020 | 1480.00 |
| 5 | | Acácio Moura | 1985-04-29 | Avenida General Costa e Silva | 908 | Vila Militar | São Paulo | SP | 06442-007 | 1340.00 |

Imagem 11 - Interface Gráfica Workbench – Resultado da consulta do Exemplo 7

Observe que a ordenação **ascendente** é padrão, cláusula **asc**, ou seja, do **menor para o maior**, se você a omitir, os registros serão apresentados nessa ordem, conforme demonstrado no **Exemplo 6**.

Quando você quiser alterar isso, utilize a cláusula **desc**, ou seja, do **maior para o menor**, conforme apresentado no Exemplo 7.

Tudo certo? Vamos seguir!!!

Um outro operador SQL muito utilizado é o **like** que faz a busca de conteúdos parecidos ou semelhantes.

Veja este exemplo:

Exemplo 8:

```
select *
from professor
where nome like 'A%';
```

utilização do operador like para pesquisas aproximadas de conteúdos em campos

| matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|-----------|--------------|---------------|-------------------------------|--------|---------------|-----------|----|-----------|---------|
| 3 | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |
| 5 | Acácio Moura | 1985-04-29 | Avenida General Costa e Silva | 908 | Vila Militar | São Paulo | SP | 06442-007 | 1340.00 |

Imagem 11 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 8

Neste exemplo, foram selecionados todos os **professores** cujo campo **nome** inicia-se com a letra “A”. O caractere “%”, significa que não importa quais serão os próximos caracteres, ou seja, neste exemplo o importante é que o nome comece com “A”, independente do conteúdo após essa letra.

Exemplo 9:

```
select *
from professor
where nome like '%O';
```

| matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|-----------|--------------|---------------|-------------------------------|--------|---------------|-----------|----|-----------|---------|
| 3 | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |
| 5 | Acácio Moura | 1985-04-29 | Avenida General Costa e Silva | 908 | Vila Militar | São Paulo | SP | 06442-007 | 1340.00 |

Imagem 12 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 9

Já no Exemplo 9, o que importa é o **final do conteúdo**, neste caso, que seja “O”, independente do que está preenchido antes. Como pôde notar, não existe nenhum **professor** cujo conteúdo do campo nome termine com a letra “O”.

Exemplo 10:

```
select *
from professor
where nome like '%ri%';
```

| matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|-----------|---------------|---------------|----------------------|--------|---------------|-----------|----|-----------|---------|
| 3 | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |
| 4 | Marilda Dutra | 1979-01-23 | Rua Adolfo Belini | 621 | São Francisco | São Paulo | SP | 01005-020 | 1480.00 |

Imagem 13 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 10

No **Exemplo 10**, foi utilizado o caractere curinga “%” duas vezes, **no início e no final**, isso significa, que o importante é encontrar um **nome que contenha** os caracteres “ri” independentemente do que está antes ou depois de deles.

Cláusula **Distinct**

Em SQL você pode ainda **distinguir** conteúdos, ou mesmo verificar quais conteúdos foram definidos para um ou mais campos, como os filtros utilizados no **Microsoft Excel**.

| matricula | nome | cidade |
|-----------|---------------|-------------|
| 1 | José da Silva | São Paulo |
| 2 | João de Souza | São Paulo |
| 3 | Ana Maria | São Paulo |
| 4 | Marilda Dutra | São Paulo |
| 5 | Acácio Moura | Santo André |

Imagem 14 – Microsoft Excel – Utilização de filtro de dados

Para exemplificarmos melhor essa cláusula, suponhamos que o Professor Acácio Moura tenha mudado de endereço. Seu novo endereço é **Avenida São Bernardo, número 127, Vila Luzita**, município de **Santo André** e o **CEP 09171-195**. Vamos utilizar os conhecimentos da agenda anterior para alterar o registro do **professor**.

update professor

```
set logradouro = ' Avenida São Bernardo',
    numero = 127,
    bairro = 'Vila Luzita',
    cidade = 'Santo André',
    uf = 'SP',
    cep = '09171-195'
```

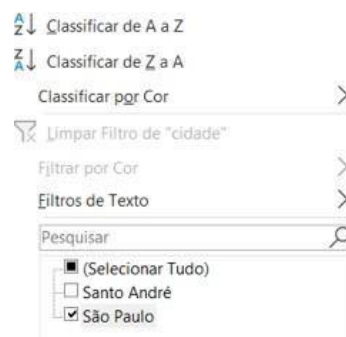
where matricula = 5;

Com a alteração, os registros dos **professores** ficaram assim:

| matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|-----------|---------------|---------------|----------------------|--------|---------------|-------------|----|-----------|---------|
| 1 | José da Silva | 1977-10-03 | Alameda 5 | 1000 | Centro | São Paulo | SP | 08150-640 | 1854.00 |
| 2 | João de Souza | 1970-12-12 | Travessa Mato Grosso | 432 | Bom Retiro | São Paulo | SP | 01122-010 | 1530.00 |
| 3 | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |
| 4 | Marilda Dutra | 1979-01-23 | Rua Adolfo Belini | 621 | São Francisco | São Paulo | SP | 01005-020 | 1480.00 |
| 5 | Acácio Moura | 1985-04-29 | Avenida São Bernardo | 127 | Vila Luzita | Santo André | SP | 09171-195 | 1340.00 |

Imagem 15 – Interface Gráfica Workbench – Relação de Professores cadastrados após a alteração do registro

Mesmo que na tabela tenhamos 5 registros, somente é apresentada uma **ocorrência de cada conteúdo**, neste caso, como quatro **professores** residem na **cidade de São Paulo**, para filtro dos registros por este campo, serão disponibilizados para seleção somente os valores '**Santo André**' e '**São Paulo**', o suficiente para filtrar todos os registros. Quando o filtro utilizando a **cidade São Paulo** for selecionado, ele apresentará 4 registros como resultado da busca.



Em SQL, aplicando o mesmo exemplo demonstrado em Excel, utilizamos a cláusula **distinct** da seguinte forma:

| matricula | nome | cidade |
|-----------|---------------|-----------|
| 1 | José da Silva | São Paulo |
| 2 | João de Souza | São Paulo |
| 3 | Ana Maria | São Paulo |
| 4 | Marilda Dutra | São Paulo |

Exemplo 11:

```
select distinct cidade
from professor;
```

clausula distinct irá desconsiderar conteúdos duplicados para o campo

| Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|--------------|---------|--------------------|
| cidade | | | |
| São Paulo | | | |
| Santo André | | | |

Imagem 16 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 11

O MySQL apresentará todos os conteúdos utilizados no preenchimento do campo **cidade**, desconsiderando os repetidos.

Você se lembra do vídeo Curso MySQL #14 - Modelo Relacional, disponível em <https://www.youtube.com/watch?v=8fxKJWJcRTw>, que vimos na Agenda 9 do Módulo 1? Vamos trabalhar na prática seus conceitos sobre relacionamentos entre entidades.

Para exemplificar melhor esses conceitos vamos incluir registros nas Tabelas **curso_professor** e **professor_telefone**, baseado na representação gráfica a seguir:

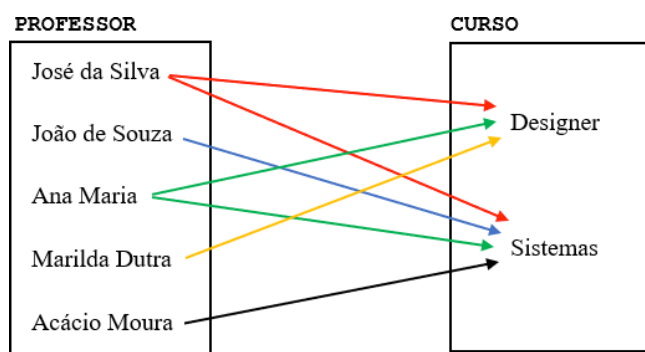


Imagem 17 – Representação Gráfica do relacionamento entre os registros de professor e curso

O relacionamento entre as Tabelas professor e curso, será representado pelos registros na Tabela **curso_professor** que possui a seguinte estrutura:

92 • describe curso_professor;

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

| Field | Type | Null | Key | Default | Extra |
|----------------|---------|------|-----|---------|----------------|
| curso_prof_id | int(10) | NO | PRI | NULL | auto increment |
| codigo_curso | int(3) | NO | MUL | NULL | |
| matricula_prof | int(5) | NO | MUL | NULL | |

Imagem 18 – Interface Gráfica Workbench – Estrutura da Tabela curso_professor

Note que os campos **codigo_curso** e **matricula_prof** são **chaves estrangeiras** na Tabela **curso_professor** e chaves primárias na Tabelas **curso** e **professor** respectivamente. Vamos fazer o relacionamento entre os registros dessas duas Tabelas com base no valor desses dois campos.

Exemplos 12:

```
insert into curso_professor
(codigo_curso, matricula_prof)
values (2, 1);
```

No **Exemplo 12**, vinculamos o Professor José da Silva, que possui o campo **matricula** = 1 ao Curso Designer que possui **codigo** = 2. Entendeu? Tenho certeza que sim!!! Vamos, agora, inserir os demais relacionamentos entre professores e cursos.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

| curso_prof_id | codigo_curso | matricula_prof |
|---------------|--------------|----------------|
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 2 | 3 |
| 4 | 1 | 3 |
| 5 | 2 | 4 |
| 6 | 1 | 5 |

Imagem 19 – Interface Gráfica Workbench – Relação de registros que relacionam as Tabelas professor e curso

Obs.: lembrando que o conteúdo do campo **curso_prof_id** não consta no comando **insert** porque ele foi definido como **auto_increment** quando da criação da Tabela **curso_professor**.

Agora é a vez dos telefones, vamos incluir os números aos professores de acordo com a seguinte lista:

| Professor | Telefone |
|---------------|----------------|
| José da Silva | (11) 2324-2345 |
| João de Souza | (11) 3456-2397 |
| Ana Maria | (11) 2304-4854 |
| Marilda Dutra | (11) 2348-3984 |
| Acácio Moura | (11) 3471-4857 |
| | (11) 9872-3647 |

Imagem 20 – Lista de Telefones dos professores

Para vincular os números dos telefones aos professores utilizaremos a Tabela **professor_telefone** que possui a seguinte estrutura:

| Field | Type | Null | Key | Default | Extra |
|-------------------|-------------|------|-----|---------|----------------|
| professor_fone_id | int(10) | NO | PRI | NULL | auto increment |
| matricula | int(5) | NO | MUL | NULL | |
| numero | varchar(15) | NO | | NULL | |

Imagem 21 – Interface Gráfica Workbench – Estrutura da Tabela professor_telefone

Note que o campo **matricula** é **chave estrangeira** na Tabela **professor_telefone** e chave primária na Tabela **professor**. Vamos inserir aos professores seus respectivos números de telefone.

Exemplos 13:

```
insert into professor_telefone
(matricula, numero)
values
(1, '(11) 2324-2345');
```

No Exemplo 13, inserimos para o **Professor** José da Silva, que possui o campo **matricula** = 1 o número '(11) 2324-2345'. Entendeu? Tenho que certeza que sim!!! Vamos, agora, inserir os demais telefones.

| professor_fone_id | matricula | numero |
|-------------------|-----------|----------------|
| 1 | 1 | (11) 2324-2345 |
| 2 | 2 | (11) 3456-2397 |
| 3 | 3 | (11) 2304-4854 |
| 4 | 4 | (11) 2348-3984 |
| 5 | 5 | (11) 3471-4857 |
| 6 | 5 | (11) 9872-3647 |

Imagem 22 – Interface Gráfica Workbench – Relação de registros com os telefones dos professores

Agora que já temos os registros dos relacionamentos entre as Tabelas **professor** e **curso** e os telefones dos **professores**, vamos aprender agora a consultar múltiplas Tabelas.

Quando necessitamos apresentar, em uma mesma consulta, campos de mais de uma Tabela como por exemplo, o nome do Professor e o Curso que ele leciona, será necessário utilizarmos um mecanismo que reúna os dados dessas duas Tabelas. Esse mecanismo é conhecido como **junção (join)**.

O que é uma junção?

As consultas a uma única Tabela certamente não são raras, mas você verá que a maioria de suas consultas requer duas, três ou mais tabelas. Com base nas estruturas nas Tabelas `professor` e `professor_telefone` vamos definir uma consulta que recupere dados de ambas as tabelas.

Suponhamos que foi solicitado a você o desenvolvimento de uma consulta que apresente o nome do professor e seu telefone. Sua consulta, portanto, precisa apresentar o conteúdo dos campos nome e numero das Tabelas `professor` e `professor_telefone` respectivamente. Mas como você faria para apresentar os dados de ambas as tabelas e, além disso, os **números de telefones** relacionados de cada professor?

A resposta está no campo matricula da Tabela `professor_telefone`, é ele que vincula o professor a seu(s) telefone(s). O campo matricula na Tabela `professor_telefone` é a **chave estrangeira**, ela referencia a chave primária da Tabela `professor`, ou seja, o servidor a utiliza como ponte entre as duas Tabelas, permitindo assim que os campos de ambas as Tabelas sejam incluídos no resultado da consulta. Esse tipo de operação é conhecido como junção.

Para assimilar melhor esses conceitos, vamos fazer mais alguns exemplos!!!

Exemplos 14:

```
select p.nome, pt.numero

from professor p inner join professor_telefone pt
    on p.matricula = pt.matricula

order by p.matricula;
```

Define para o servidor que ele deverá juntar as tabelas `professor` e `professor_telefone` utilizando o campo `matricula`

| Result Grid | | | Filter Rows: | Export: | Wrap Cell Content: |
|-------------|---------------|----------------|--------------|---------|--------------------|
| | nome | numero | | | |
| | José da Silva | (11) 2324-2345 | | | |
| | João de Souza | (11) 3456-2397 | | | |
| | Ana Maria | (11) 2304-4854 | | | |
| | Marilda Dutra | (11) 2348-3984 | | | |
| | Acácio Moura | (11) 3471-4857 | | | |
| | Acácio Moura | (11) 9872-3647 | | | |

Imagem 23 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 14

Obs.: Como o Professor Acácio Moura tem dois telefones, ele é apresentado em duas linhas.

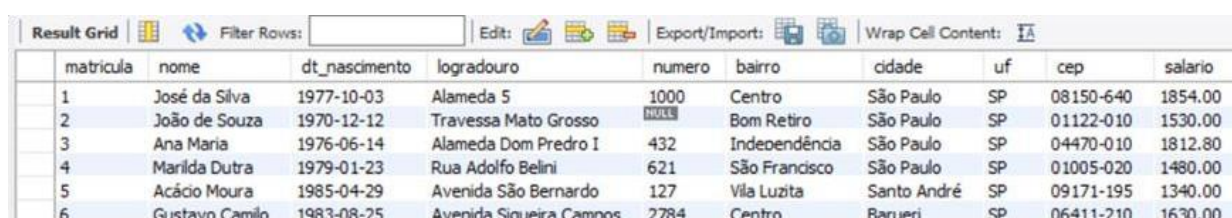
Onde:

Inner join: é o tipo de junção interna mais utilizado e significa que se um valor existe em uma Tabela mas não existe na outra, a junção falha, e esses registros não são exibidos no resultado.

p: também chamado de **alias**, é um apelido dado à Tabela, neste caso a Tabela professor.

pt: apelido dado à Tabela **professor_telefone**.

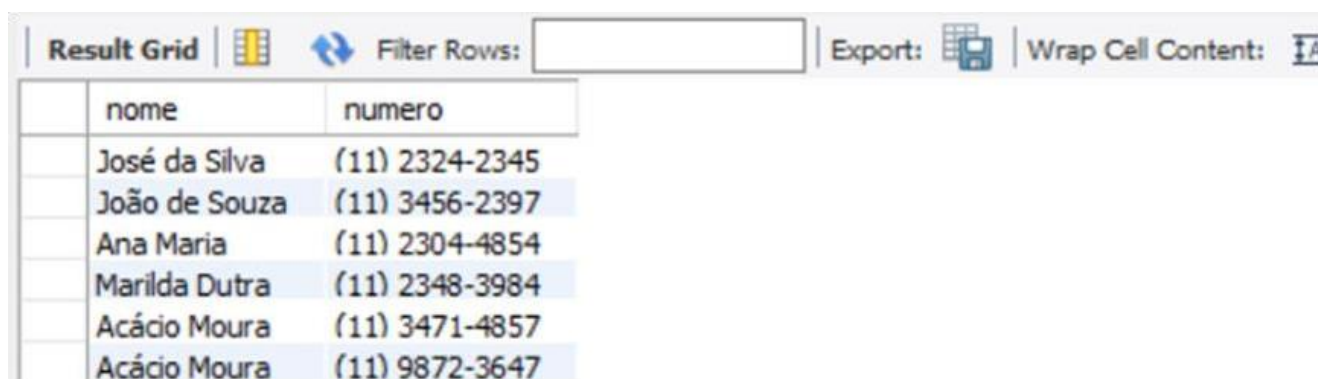
Vamos inserir mais um registro na Tabela **professor** para simular a falha de uma junção interna (**inner join**).



| | matricula | nome | dt_nascimento | logradouro | numero | bairro | cidade | uf | cep | salario |
|---|-----------|----------------|---------------|-------------------------|--------|---------------|-------------|----|-----------|---------|
| 1 | | José da Silva | 1977-10-03 | Alameda 5 | 1000 | Centro | São Paulo | SP | 08150-640 | 1854.00 |
| 2 | | João de Souza | 1970-12-12 | Travessa Mato Grosso | 1000 | Bom Retiro | São Paulo | SP | 01122-010 | 1530.00 |
| 3 | | Ana Maria | 1976-06-14 | Alameda Dom Predro I | 432 | Independência | São Paulo | SP | 04470-010 | 1812.80 |
| 4 | | Marilda Dutra | 1979-01-23 | Rua Adolfo Belini | 621 | São Francisco | São Paulo | SP | 01005-020 | 1480.00 |
| 5 | | Acácio Moura | 1985-04-29 | Avenida São Bernardo | 127 | Vila Luzita | Santo André | SP | 09171-195 | 1340.00 |
| 6 | | Gustavo Camilo | 1983-08-25 | Avenida Siqueira Camoos | 2784 | Centro | Barueri | SP | 06411-210 | 1630.00 |

Imagem 24 – Interface Gráfica Workbench – Relação de professores cadastrados após a inserção de mais um registro

Foi inserido o Professor Gustavo Camilo. Agora, execute novamente o **select** do Exemplo 14.



| | nome | numero |
|--|---------------|----------------|
| | José da Silva | (11) 2324-2345 |
| | João de Souza | (11) 3456-2397 |
| | Ana Maria | (11) 2304-4854 |
| | Marilda Dutra | (11) 2348-3984 |
| | Acácio Moura | (11) 3471-4857 |
| | Acácio Moura | (11) 9872-3647 |

Imagem 25 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 14 após a inserção de mais um registro

Note que o resultado continua o mesmo, isso porque o **Professor Gustavo Camilo** não possui nenhum **telefone**, ou seja, o conteúdo do campo **matricula**, utilizado pela junção para relacionar as duas Tabelas só existe em uma delas, neste caso, somente na Tabela professor.

Juntando três ou mais tabelas

Para juntar duas tabelas utilizamos somente um tipo de junção na cláusula **from**, além de uma única subcláusula **on**. Quando temos que juntar três tabelas, devemos utilizar dois tipos de junção na cláusula **from**, além de duas subcláusulas **on**.

Exemplo 15:

```
select p.matricula, p.nome, c.descricao

from professor p
inner join curso_professor cp
    on p.matricula = cp.matricula_prof
inner join curso c
    on cp.codigo_curso = c.codigo

where c.codigo = 1

order by p.matricula;
```

Dois tipos de junção
professor e curso_professor
curso_professor e curso
Duas subcláusulas on
p.matricula =
cp.matricula_prof
cp.codigo_curso = c.codigo

O resultado da consulta traz todos os **professores** que lecionam no **curso** de **Sistemas**, apresentando o **professor**, os dados **matricula** e **nome**, o curso e o dado **descricao**. Para conseguirmos esse resultado foram utilizadas as **chaves estrangeiras** **codigo_curso** e **matricula_prof** da Tabela **curso_professor** para relacioná-la com as Tabelas **professor** e **curso** por meio das **chaves primárias** **matricula** e **codigo** respectivamente.

| matricula | professor | curso |
|-----------|---------------|----------|
| 1 | José da Silva | Sistemas |
| 2 | João de Souza | Sistemas |
| 3 | Ana Maria | Sistemas |
| 5 | Acácio Moura | Sistemas |

Imagem 26 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 15

Funções de agregação

A história do surgimento dos computadores está muito ligada à necessidade da execução de cálculos aritméticos, com maior rapidez e eficiência. Em SQL, alguns cálculos são facilmente executados a partir de funções pré-definidas, como contar, somar entre outras. Estou falando sobre as funções de agregação.

As funções comuns de agregação implementadas por todos os principais servidores incluem:

max () : retorna o maior valor dentro de um conjunto.

min () : retorna o menor valor dentro de um conjunto.

avg () : retorna o valor médio de um conjunto.

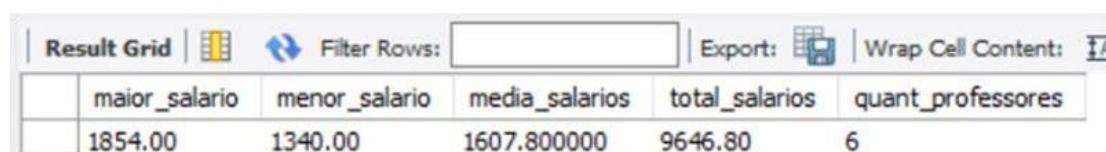
sum () : retorna a soma dos valores de um conjunto.

count () : retorna a quantidade de valores em um conjunto.

Vamos fazer um exemplo que usa todas as funções comuns de agregação para analisar os salários pagos aos professores da escola:

Exemplos 16:

```
select max(p.salario) maior_salario,
       min(p.salario) menor_salario,
       avg(p.salario) media_salarios,
       sum(p.salario) total_salarios,
       count(*) quant_professores
from professor p;
```



| | maior_salario | menor_salario | media_salarios | total_salarios | quant_professores |
|--|---------------|---------------|----------------|----------------|-------------------|
| | 1854.00 | 1340.00 | 1607.800000 | 9646.80 | 6 |

27 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 16

Obs.: Assim como foi definido o alias “p” para a Tabela professor, note que também foi definido para os campos que serão apresentados como resultado da consulta.

Para incrementar a utilização das funções de agregação, podemos incluir outras cláusulas:

group by: usada para agrupar os registros por meio de conteúdos comuns de campos.

having: filtra grupos indesejados.

Exemplos 17:

```
select p.cidade,
       count(*) quant_professores
from professor p
group by p.cidade;
```

campo definido para o agrupamento dos registros

| cidade | quant_professores |
|-------------|-------------------|
| Barueri | 1 |
| Santo André | 1 |
| São Paulo | 4 |

27 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 16

A consulta do **Exemplo 16**, apresenta como resultado a quantidade de **professores** cadastrados **agrupados** pelo campo **cidade**.

Exemplos 18:

```
select p.cidade,
       sum(p.salario) total_salarios
from professor p
group by p.cidade
having count(*) > 1;
```

Define a condição dos registros que serão considerados

A consulta do **Exemplo 17**, apresenta como resultado o total pago de **salários** para os **professores** agrupados pelo campo **cidade**, mas a condição da cláusula **having** (**count**(*) > 1) faz com que sejam consideradas somente as **cidades** que possuem mais de um **professor**.

| cidade | total_salarios |
|-----------|----------------|
| São Paulo | 6676.80 |

Imagem 29 – Interface Gráfica Workbench – Resultado da consulta do Exemplo 18

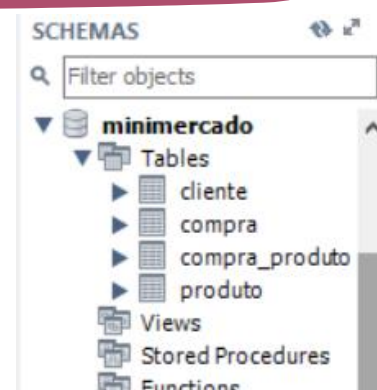
Agora é com você!!!



VOCÊ NO COMANDO

Vamos continuar utilizando o Projeto do Banco de Dados do Sistema do minimercado. Adriano precisa de relatórios que o ajudem a gerenciar seu negócio.

Além da relação de clientes e produtos cadastrados, e das compras efetuadas pelos clientes, esses relatórios serão gerados diariamente, semanalmente ou mensalmente, isso significa que devem possuir filtros com intervalos entre datas. Utilize as funções de agregação para obter o total de clientes e produtos cadastrados e análises por período baseadas nas compras efetuadas pelos clientes, tais como: valor total de produtos vendidos em um dia, o valor médio das vendas durante um mês e maior venda da semana.



Obs.: Não se esqueça de selecionar o Banco de Dados antes de iniciar a execução  das instruções.

Tudo certo? Você conseguiu? Tenho certeza que sim!!!! Vamos juntos conferir!!!

Para obtermos a relação de clientes e produtos cadastrados você poderá utilizar as seguintes consultas:

```
select *
from cliente
order by nome;
```

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

Para incrementar as consultas acrescentamos nas duas ordenações baseadas nos campos **nome** e **descricao**, das Tabelas **cliente** e **produto** respectivamente.

Para obtermos a relação das compras efetuadas pelos clientes podemos utilizar as seguintes consultas:

a) Diariamente.

```
select cp.codigo_compra,
       cp.data,
       c.cpf_cliente,
       c.nome
from cliente c inner join compra cp
on c.cpf_cliente = cp.cpf_cliente
where cp.data = '2019-10-01';
```

b) Semanalmente, quinzenalmente ou mensalmente.

```
select cp.codigo_compra,
       cp.data,
       c.cpf_cliente,
       c.nome
from cliente c inner join compra cp
on c.cpf_cliente = cp.cpf_cliente
where c.data between '2019-10-01' and '2019-10-15';
```

Obs.: altere o intervalo entre as datas para gerar qualquer um dos relatórios.

Chegou a vez das funções de agregação, começamos pelo total de **clientes** e **produtos** cadastrados:

```
select count(*) total_clientes
from cliente;
```

```
select count(*) total_produtos
from produto;
```

Vamos agora para as análises das compras dos clientes:

a) valor total de produtos vendidos em um dia.

```
select sum(quantidade * preco) total_compra
from compra c inner join compra_produto cp
on c.codigo_compra = cp.codigo_compra
where c.data = '2019-10-01';
```

Obs.: na Tabela `compra_produto` não temos o total da compra, mas sim, a quantidade comprada e o preço unitário, por esse motivo antes de somar, foi necessário multiplicar um campo pelo outro.

b) o valor médio das vendas durante um mês.

```
select avg(quantidade * preco) total_compra
from compra c inner join compra_produto cp
on c.codigo_compra = cp.codigo_compra
where c.data between '2019-10-01' and '2019-10-31';
```

Obs.: foi utilizado a cláusula `between` para implementação de filtro com intervalo entre datas.

c) maior venda da semana.

```
select max(quantidade * preco) total_compra
from compra c inner join compra_produto cp
on c.codigo_compra = cp.codigo_compra
where c.data between '2019-10-01' and '2019-10-07';
```

É isso aí!!! Vamos agora finalizar essa agenda colocando a mão na massa.