

Linguagem de Manipulação e Definição de Dados

Prof. Alex Luciano Roesler Rese, MSc.

Adaptado: Prof. Lucas Debatin, MSc.

Conteúdo

- Linguagem de Manipulação de Dados (Select)
- Exercícios

DQL



Conceito

- DQL = Data query language.
- Pode-se buscar dados de uma ou mais tabelas relacionadas dentro do banco de dados.
- Uma das tarefas mais comuns em bancos de dados.
- Saber criá-las da melhor maneira é muito importante para o desempenho do BD e de aplicações.

Conceito

- “A SQL tem uma instrução básica para recuperar informações de um banco de dados: a instrução SELECT”.

Elmasri e Navathe (2011)

SELECT

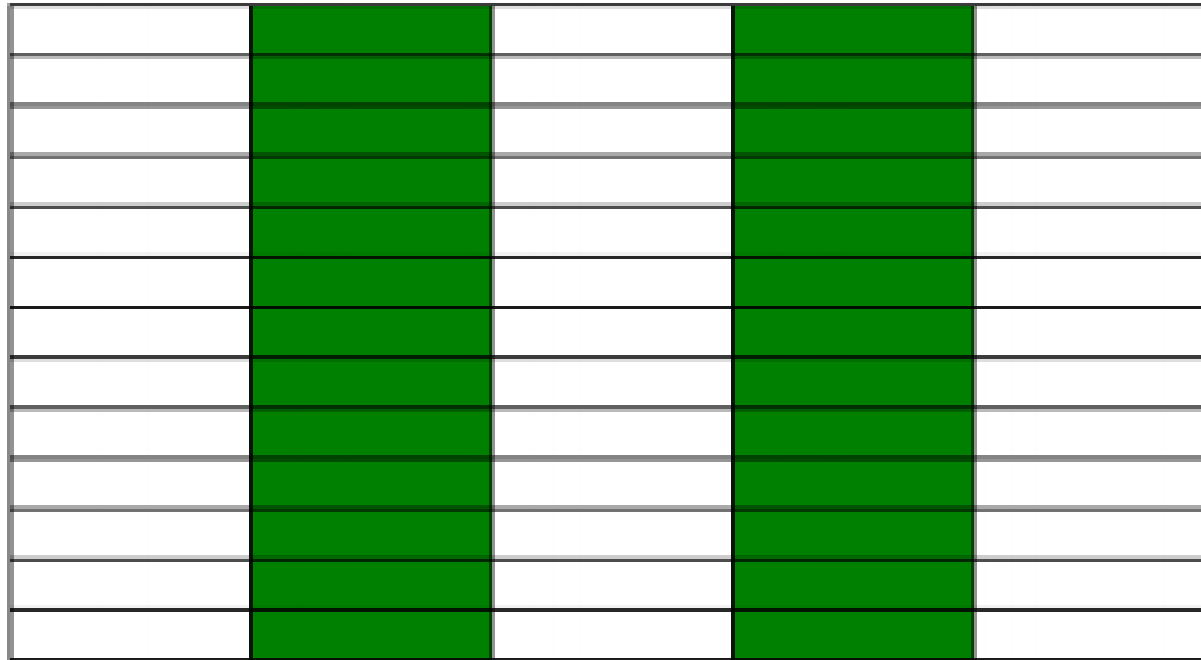
- Principais recursos: ordenação de registros, funções de agregação, junções, entre outros.
- `SELECT * FROM nome_tabela;`
 - `SELECT`: identifica que colunas;
 - `*` = todas;
 - `FROM`: identifica qual tabela.

SELECT - Seleção

- `SELECT * FROM nome_tabela WHERE condicoes;`

SELECT - Projeção

- SELECT **lista_colunas** FROM nome_tabela;



Alias

- Podemos utilizar a palavra-chave AS para criar um apelido para coluna ou função de agregação:
 - `SELECT id, data_venda AS data FROM venda;`
 - `SELECT COUNT(*) AS qtd_vendas FROM venda;`

DISTINCT

- Trazer informações únicas.
 - Removendo duplicados.
- DISTINCT:
 - `SELECT DISTINCT cidade FROM cliente;`

ORDER BY

- Organiza os resultados de acordo com uma ou mais colunas da tabela.
- Pode-se definir a ordem dos resultados:
 - Crescente (ASC) ☐ padrão;
 - Decrescente (DESC).
- `SELECT * FROM cliente ORDER BY cidade, nome;`

ORDER BY

- `SELECT * FROM aluno ORDER BY nome;`

id	nome	email
1	João Fernando	joao@gmail.com
2	Maria Carvalho	maria@gmail.com
3	Bruna dos Santos	bruna@gmail.com
4	Pedro da Silva	pedro@gmail.com
5	Tiago	tiago@gmail.com

id	nome	email
3	Bruna dos Santos	bruna@gmail.com
1	João Fernando	joao@gmail.com
2	Maria Carvalho	maria@gmail.com
4	Pedro da Silva	pedro@gmail.com
5	Tiago	tiago@gmail.com

LIMIT

- Especifica o número de linhas que devem ser retornadas no resultado de uma consulta.
 - `SELECT * FROM alunos LIMIT 0, 3;`

id	nome	email	Id_turma
1	João Fernando	joao@gmail.com	1
2	Maria Carvalho	maria@gmail.com	1
3	Bruna dos Santos	bruna@gmail.com	2
4	Pedro da Silva	pedro@gmail.com	2
5	Tiago Pereira	tiago@gmail.com	1

id	nome	email	Id_turma
1	João Fernando	joao@gmail.com	1
2	Maria Carvalho	maria@gmail.com	1
3	Bruna dos Santos	bruna@gmail.com	2

UNION

- Combina os resultados de duas ou mais queries em um único result set, retornando todas as linhas pertencentes a todas as queries envolvidas na execução.
 - O número e a ordem das colunas precisam ser idênticos em todas as queries e os data types precisam ser compatíveis.
- Existem dois tipos: UNION e UNION ALL.

UNION

- Combina o resultado de execução das duas queries e elimina as linhas duplicadas.
- `SELECT nome, curso FROM turma WHERE semestre = 1 UNION
SELECT nome, curso FROM turma WHERE semestre = 2;`

UNION ALL

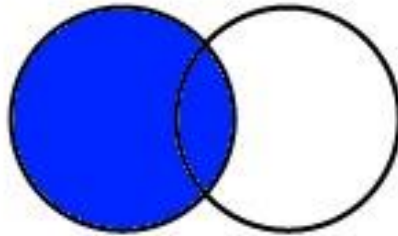
- Combina o resultado de execução das duas queries, entretanto, não elimina as linhas duplicadas.
- `SELECT nome, curso FROM turma WHERE semestre = 1 UNION ALL SELECT nome, curso FROM turma WHERE semestre = 2;`

JOIN

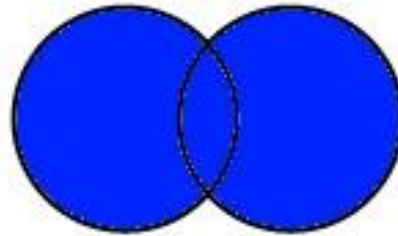
- As tabelas devem ser associadas em pares, embora seja possível usar um único comando para juntar várias tabelas.
- Uma das formas mais usadas é a associação da chave primária da primeira tabela com a chave estrangeira da segunda.

JOIN

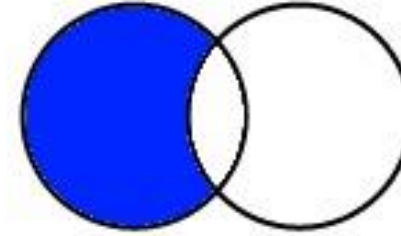
LEFT JOIN



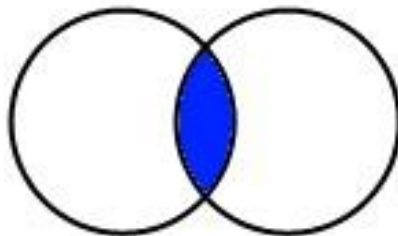
FULL OUTER JOIN



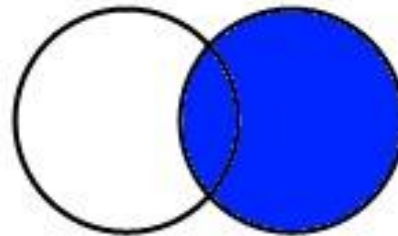
**LEFT JOIN
(if NULL)**



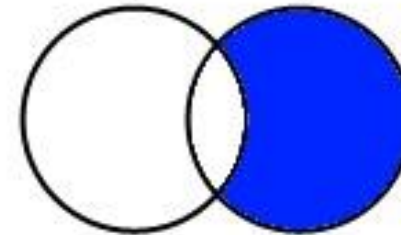
INNER JOIN



RIGHT JOIN

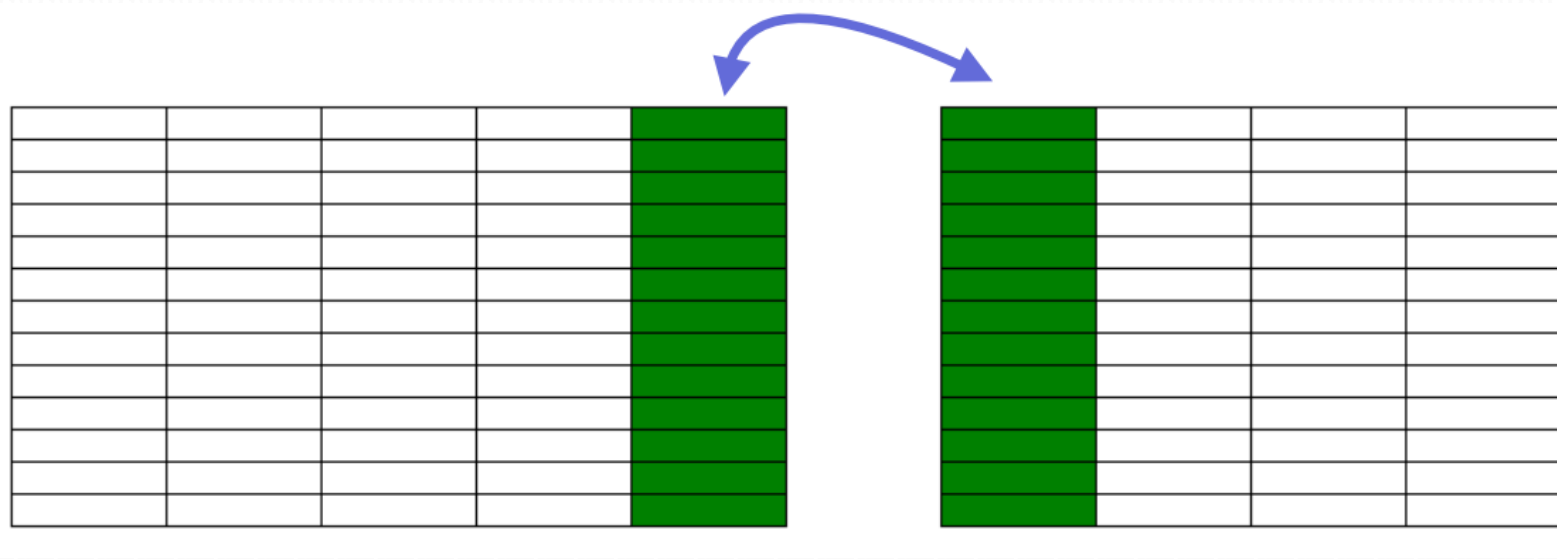


**RIGHT JOIN
(if NULL)**



JOIN

- `SELECT t1.status, t2.id FROM nome_tabela1 t1 INNER JOIN nome_tabela2 t2 ON (t2.id_tabela1 = t1.id);`



JOIN

- `SELECT c.id, c.nome, e.sigla FROM cidade c LEFT JOIN estado e ON c.id_estado = e.id;`
- `SELECT c.nome, v.data_venda, i.quantidade FROM cliente c INNER JOIN venda v ON (v.id_cliente = c.id) INNER JOIN item i ON (i.id_venda = v.id);`

Tabela Dual

- Não há necessidade de usar dual no MySQL e na maioria dos bancos de dados SQL.
 - DUAL é introduzido pela Oracle para fornecer resultados que retornam sempre um registro (uma linha e coluna).
- Oracle:
 - `SELECT 'Lucas Debatin' AS nome FROM dual;`
- MySQL:
 - `SELECT 'Lucas Debatin' AS nome;`
 - `SELECT 6*6+3/2;`
 - `SELECT coluna_nome FROM table_nome LIMIT 1;`

Funções de Agregação

- MAX:
 - `SELECT MAX(preco venda) FROM produtos;`
- MIN:
 - `SELECT MIN(preco venda) FROM produtos;`
- SUM:
 - `SELECT SUM(preco venda) FROM produtos WHERE categoria = 1;`

	max(preco venda)
	14.00

	min(preco venda)
	1.00

	sum(preco venda)
	6.65

Funções de Agregação

- AVG:

- SELECT AVG(preco venda) FROM produtos;

	avg(preco venda)
	5.412500

- COUNT:

- SELECT COUNT(preco venda) FROM produtos WHERE categoria = 1;

	count(preco venda)
	5

Funções de String

- CHAR_LENGTH:
 - `SELECT CHAR_LENGTH('Lucas');`
 - Output: 5
- CONCAT:
 - `SELECT CONCAT('Lucas', 'Debatin');`
 - Output: Lucas Debatin
- CONCAT_WS:
 - `SELECT CONCAT_WS('/', '04', '02', '2023');`
 - Output: 04/02/2023

Funções de String

- INSTR:
 - `SELECT INSTR('Lucas Debatin 28 anos', '29');`
 - Output: 0
 - `SELECT INSTR('Lucas Debatin 29 anos', '29');`
 - Output: 15
 - Posição inicial.

Funções de String

- LCASE ou LOWER:
 - `SELECT LCASE('LUCAS');`
 - Output: lucas
 - `SELECT LOWER('LUCAS');`
 - Output: lucas
- LEFT:
 - `SELECT LEFT('Lucas', 4);`
 - Output: Luca

Função de String

- LPAD/RPAD:
 - `SELECT LPAD('51',6, '0');`
 - Output: 000051
 - `SELECT LPAD('51',6, '0');`
 - Output: 510000

Função de String

- LTRIM/TRIM/RTRIM:
 - SELECT LTRIM(' Lucas');
Output: Lucas
 - SELECT RTRIM('Lucas ');
Output: Lucas
 - SELECT CONCAT('[', TRIM(' Lucas '), ']');
 - Output: [Lucas]

Funções de String

- SUBSTRING:
 - SELECT SUBSTRING ('Lucas Debatin', 7, 4);
 - Output: Deba
- REPEAT:
 - SELECT REPEAT('Lucas ', 3);
 - Output: Lucas Lucas Lucas

Funções de String

- REPLACE:
 - `SELECT REPLACE('www.dlweb.com', '.com', '.com.br');`
 - Output: `www.dlweb.com.br`
- RIGHT:
 - `SELECT RIGHT('ALEXANDRE',5);`
 - Output: `ANDRE`

Funções de String

- UCASE/UPPER:
 - `SELECT UCASE('lucas');`
 - Output: LUCAS
 - `SELECT UPPER('lucas');`
 - Output: LUCAS
- REVERSE:
 - `SELECT REVERSE('1234');`
 - Output: 4321

Funções de Cálculos

- `CEILING()`: Arredondar para cima
- `FLOOR()`: Arredondar para baixo
- `PI()`: Retorna o valor de Pi
- `POW(x,y)`: Retorna x elevado a y
- `SQRT()`: Raiz quadrada de um argumento
- `SIN()`: Retorna o seno de um número dado em radianos
- `HEX()`: Retorna a representação hexadecimal de um valor decimal.

Expressões

- Aritméticas:
 - + - * /
 - `SELECT id, data_venda - data_contato AS tempo_venda FROM venda;`
- Lógicas:
 - AND, OR
 - `SELECT id, data_venda - data_contato AS tempo_venda FROM venda WHERE id > 100 AND id < 150;`

Funções de Data

- NOW(): Retorna a data e hora atuais
- CURDATE(): Retorna a data atual
- CURTIME(): Retorna a hora atual
- DATE() : Extrai a parte da data de uma data ou expressão/hora
- EXTRACT(): Retorna uma única parte de uma data/hora (MONTH, YEAR, ...)
- DATE_ADD(): Adiciona um intervalo de tempo especificado para uma data
- DATE_SUB(): Subtrai um intervalo de tempo especificado a partir de uma data
- DATEDIFF(): Retorna o número de dias entre duas datas
- DATE_FORMAT(): Exibe a data/data e hora em diferentes formatos

GROUP BY

- DISTINCT != GROUP BY.
- É usado em conjunto com as funções de agregação.
- Agrupa linhas baseado em semelhanças entre elas.
 - `SELECT COUNT(id_cidade) FROM cliente GROUP BY id_cidade;`

HAVING

- Podemos usar a cláusula HAVING em conjunto com GROUP BY para filtrar os resultado que serão submetidos a agregação.
 - `SELECT categoria, MAX(preco venda) FROM produtos GROUP BY categoria HAVING MAX(preco venda) > 10;`

SUBQUERIES

- É um comando SELECT que é embutido dentro de outro comando.
 - Utiliza-se normalmente com as cláusulas IN() ou EXISTS().
 - Exemplos:
 - `SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);`
 - `SELECT * FROM pedido WHERE id IN (SELECT pedido_id FROM venda WHERE cliente_id = 2);`
 - `SELECT * FROM pedido pe WHERE EXISTS (SELECT pedido_id FROM venda WHERE pedido_id = pe.id AND cliente_id = 2);`
 - `DELETE FROM t1 WHERE s11 > (SELECT COUNT(*) FROM t2);`

INSERT + SELECT

- INSERT INTO tabela1 (coluna1, coluna2, coluna3, ...)
SELECT coluna1, coluna2, coluna3, ...
FROM tabela2
[WHERE condição];

UPDATE + SELECT

- UPDATE tabela1 SET coluna1 = (
 SELECT coluna1
 FROM tabela2
 [WHERE condição]);
- UPDATE tabela1 SET coluna1 = valor1 WHERE coluna2 = (
 SELECT coluna1
 FROM tabela2
 [WHERE condição]);

CREATE TABLE + SELECT

- Uma cópia de uma tabela existente poderá ser criada usando CREATE TABLE.
 - A nova tabela obtém as mesmas definições de coluna.
 - Todas as colunas ou colunas específicas podem ser selecionadas.
 - Ao criar uma nova tabela usando uma tabela existente, a nova tabela será preenchida com os valores existentes da tabela antiga.

CREATE TABLE + SELECT

- CREATE TABLE nova_tabela1 AS
SELECT coluna1, coluna2,...
FROM tabela1
[WHERE condição];

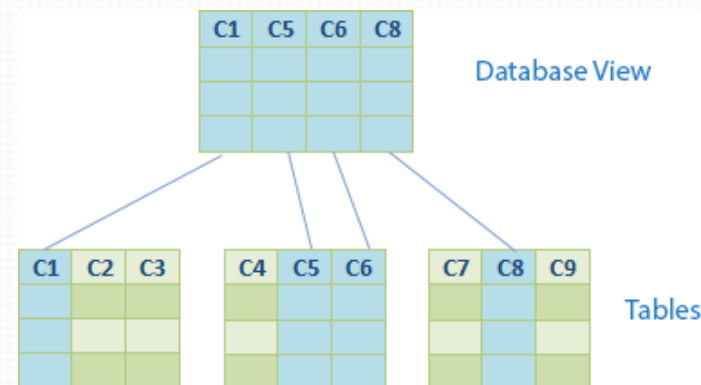
← Poderá usar apenas *
para todas as colunas

VIEW

- Retorna uma determinada visualização de dados de uma ou mais tabelas.
- Promover restrições em dados para aumentar a segurança e definir políticas de acesso em nível de tabela e coluna.
- Conjunto de tabelas que podem ser unidas a outros conjuntos de tabelas com a utilização de JOIN's ou UNION.

VIEW

- É uma ferramenta valiosa para impor políticas de segurança.
- Pode ser usado para criar uma “janela” sobre uma coleção de dados, apropriada para algum grupo de usuários.
- Permitem limitar o acesso a dados sigilosos, fornecendo acesso a uma versão restrita desses dados, em vez do acesso aos dados em si.



Conceito

- “Em um sistema de banco de dados multiusuário, o SGBD precisa oferecer técnicas para permitir que certos usuários ou grupos de usuários acessem partes selecionadas de um banco de dados sem que obtenham acesso ao restante dele”.

Elmasri e Navathe (2011)

VIEW

- `CREATE [OR REPLACE] VIEW nome_view AS comando_select;`
- Exemplo:
 - `CREATE VIEW v_aluno AS SELECT nome FROM aluno;`
- Executar a VIEW:
 - `SELECT * FROM v_aluno;`

VIEW

- Funcionario (id, id_departamento, id_cidade, nome, salario, logradouro, numero, bairro, data_admissao, status)
- CREATE VIEW View_Funcionario AS SELECT id, id_departamento, id_cidade, nome, data_admissao FROM funcionario;
- SELECT * FROM View_Funcionario;

INDEX

- É uma estrutura de dados utilizada para melhorar o tempo de execução das consultas.
- São estruturas que organizam referências da localização dos dados nas tabelas.

Índice	
1. Introdução	5
2. Arquitetura Barroca	
2.1. O Grande Barroco	7
2.2. Palácio de Mafra	11
2.3. Norte de Portugal	15
3. Escultura barroca	19
4. Pintura Barroca	23
5. Conclusão	27
6. Bibliografia	29

INDEX - Dicas

- Adicionar índices aos campos de cláusulas WHERE;
- Evitar muitos índices e índices com mais de um campo;
- Evitar índice em tabela muito pequena (poucos registros, não compensa).

INDEX

- Vantagem:
 - As pesquisas são realizadas de maneira mais rápida, quando estes são adicionados em campos únicos, por exemplo, o CPF de um cliente.
- Desvantagens:
 - Os dados são manipulados (INSERT, UPDATE, DELETE) de forma mais lenta com base nos índices criados, devido a reorganização dos índices;
 - Os índices podem levar muito tempo para serem criados.

TABELA HEAP (Padrão)

- Os dados são armazenados sem uma ordem.
 - São manipulados sem uma reorganização dos dados existentes na tabela.
- Melhor desempenho no momento de manipulação, mas é lento ao recuperar as informações.
 - É necessário uma busca por cada registro na tabela.

TABELA com ÍNDICES

- É necessário ter cuidado com a quantidade de índices por tabela, pois sobrecarrega as operações de manipulação (INSERT, UPDATE e DELETE).
 - Para cada uma das operações realizadas, é necessária a atualização dos índices.

TABELA com ÍNDICES

- CREATE INDEX nome_indice ON tabela (colunas);

- Exemplos:

```
CREATE TABLE t1 (  
  col1 VARCHAR(10),  
  col2 VARCHAR(20),  
  INDEX (col1, col2(10))  
);
```

```
CREATE INDEX idx1 ON t1 ((col1 + col2));
```

```
CREATE INDEX idx2 ON t1 ((col1 + col2), (col1 - col2), col1);
```

```
ALTER TABLE t1 ADD INDEX ((col1 * 40) DESC);
```

Vamos Praticar

Softwares utilizados em aula:

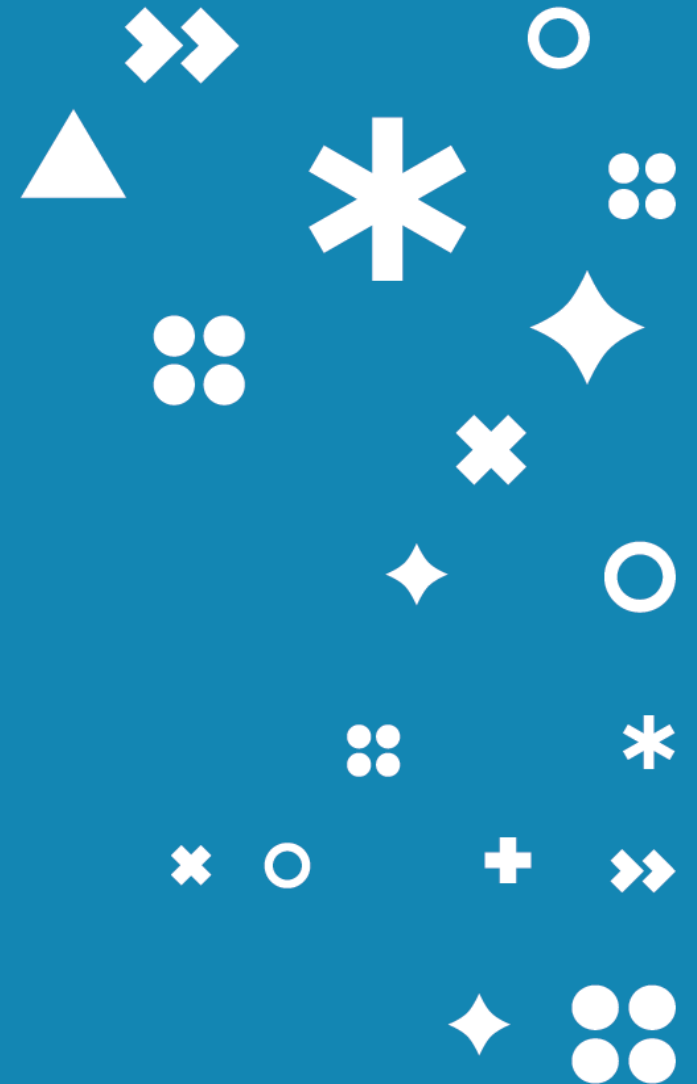
MySQL Server;

MySQL Workbench.

Disponível em:

<https://dev.mysql.com/downloads/mysql/>

Tutorial para instalação no material da Aula 7.



Ferramentas

- https://www.w3schools.com/sql/exercise.asp?filename=exercise_database7
- <https://www.sql-practice.com/>
- <http://sqlweb.com.br/aluno/classificacao.php?e=48&p=50>

Questão 1

- Com base no banco de dados criado na aula anterior:
- Mostre o nome das cidades em ordem alfabética.
- Mostre o Nome do Cliente e Nome da Cidade dos clientes cadastrados do estado 'SC'.
- Mostre apenas os clientes que começam com a letra B.
- Mostre a quantidade de clientes que existem por cidade. Colocar o nome da cidade e a quantidade.
- Mostre a quantidade de clientes da UF = 'SC'.
- Mostre o nome dos clientes concatenando com o nome da cidade e UF (exemplo: Lucas-Brusque-SC).
- Mostre o nome dos clientes de Santa Catarina e do Paraná, em dois comandos separadamente, e junte-os utilizando o comando UNION.
- Crie uma visão para mostrar apenas o nome dos clientes de São Paulo.