6 – Introdução ao SQL

- Assim como os comandos DDL, vistos anteriormente, são responsáveis pela criação dos objetos do banco de dados, os comandos DML possibilitam que as informações sejam inseridas e manipuladas no banco de dados.

- A linguagem de manipulação de dados (DML) é utilizada para manipular os dados, ou seja, inserir, alterar, excluir e recuperar as informações no banco de dados.

- Os comandos de manipulação de dados são de grande importância no desenvolvimento de aplicações.

- Algumas linguagens de programação utilizam os comandos DML embutidos dentro de seu próprio código, possibilitando ao desenvolvedor manipular dados inseridos no banco de dados dentro da aplicação que está sendo desenvolvida.

6.1 – Inserção de Registros

- Para inserir registros em uma tabela, utiliza-se o comando **SQL INSERT,** seguido do nome da tabela e dos respectivos valores para cada campo**.**

- A sintaxe do comando INSERT é mostrada no quadro abaixo:

INSERT INTO <Tabela> (<Campo 1>, <Campo 2>, ...) **VALUES** (<Valor 1>, <Valor 2>, ...);

Onde:

<Tabela>: Nome da tabela que será inserido o registro.

<Campos 1>, <Campos 2>, ...: Nome dos campos cujas informações serão inseridas.

<Valor 1>, <Valor 2>, ...: Valores que serão inseridos para os respectivos campos.

6.1 – Inserção de Registros

- Para inserção de registros seguindo esta sintaxe, os valores devem estar na mesma ordem de declaração dos campos.

- Os campos não precisam estar declarados no comando INSERT na mesma ordem em que estão dispostos na tabela.

- Não é obrigatório a declaração de todos os campos da tabela, apenas os campos NOT NULL. Os campos que não forem declarados no comando INSERT ficarão com valor NULL.

- Os valores devem estar dentro do conjunto de valores aceitos pelo tipo do campo, do contrário o banco de dados não aceitará o registro.

- Também é importante verificar se o valor a ser inserido nos campos de chave estrangeira possuem correspondentes nas suas tabelas de origem (Ex: valor do id_cidade na tabela Cliente deve existir na tabela Cidade)

6.1 – Inserção de Registros

- Como exemplo, o quadro abaixo ilustra o comando para inserir um registro na tabela de cliente. Neste exemplo, todos os campos foram declarados na ordem de criação dentro do comando INSERT.

```
CREATE TABLE IF NOT EXISTS cliente(
  `id_cliente` INT PRIMARY KEY AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `endereco` VARCHAR(45) NULL,
  `id_cidade` INT NULL,
  `telefone` VARCHAR(11) NULL
);
```

```
INSERT INTO `cliente` (`id_cliente`, `nome`, `endereco`, `id_cidade`, `telefone`) VALUES (NULL, 'Gabriel Jesus', 'Rua Juca, 45', '1', '44999566878');
```

6.1 – Inserção de Registros

- No exemplo abaixo, um registro é inserido na tabela de cliente. Porém, os campos do comando INSERT não se encontram na ordem de criação e o campo endereço (que pode ser nulo) não receberá nenhum valor.

```
CREATE TABLE IF NOT EXISTS cliente(
  `id_cliente` INT PRIMARY KEY AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `endereco` VARCHAR(45) NULL,
  `id_cidade` INT NULL,
  `telefone` VARCHAR(11) NULL
);
```

```
INSERT INTO  `cliente` (`nome`, `id_cidade`, `telefone`)
VALUES ('Fernando Prass', '1' , '44999564378')
```

6.1 – Inserção de Registros

A linguagem SQL permite também que mais de um registro seja inserido dentro do mesmo comando, como ilustra o código abaixo:

```
CREATE TABLE IF NOT EXISTS cliente(
  `id_cliente` INT PRIMARY KEY AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `endereco` VARCHAR(45) NULL,
  `id_cidade` INT NULL,
  `telefone` VARCHAR(11) NULL
);
```

```
INSERT INTO `cliente` (`id_cliente`, `nome`, `endereco`, `id_cidade`, `telefone`) VALUES
(NULL, 'Felipe Melo', 'Rua Brasil, 52', '3', '23999566878'),
(NULL, 'Roger Guedes', 'Rua José, 15', '4', '47988566878'),
(NULL, 'William Bigode', 'Rua 19, 21', '5', '25999566878');
```

6.1 – Inserção de Registros

- A linguagem SQL também possui uma forma resumida de inserir registros, que consiste em omitir a declaração dos campos.

- Neste caso, é obrigatório que os valores estejam na ordem que os campos foram criados na tabela. Todo campo deve receber um valor, mesmo que este seja nulo.

```
CREATE TABLE IF NOT EXISTS cliente(
  `id_cliente` INT PRIMARY KEY AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `endereco` VARCHAR(45) NULL,
  `id_cidade` INT NULL,
  `telefone` VARCHAR(11) NULL
);
```

```
INSERT INTO cliente VALUES (NULL, 'Miguel Broja', 'Rua João, 40', '6', '');
```

6.2 – Alteração de Registros

- Para alterar registros que já foram cadastrados em uma tabela, utiliza-se o comando **SQL UPDATE,** seguido do nome da tabela e dos respectivos valores para cada campo**.**

- A sintaxe do comando UPDATE é mostrada no quadro abaixo:

UPDATE <Tabela> **SET**<Campo 1> = ‘<Valor1>’,
<Campo2>= ‘<Valor2>’,
<CampoN>= ‘<ValorN>’
WHERE <Campo>=‘<Valor Campo Identificador>’;

6.2 – Alteração de Registros

- Vejamos um exemplo de alteração de registros na tabela de Clientes:

```
CREATE TABLE IF NOT EXISTS cliente(
  `id_cliente` INT PRIMARY KEY AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `endereco` VARCHAR(45) NULL,
  `id_cidade` INT NULL,
  `telefone` VARCHAR(11) NULL
);
```

```
UPDATE `cliente` SET `endereco` = 'Rua Tiradentes, 56',
`telefone` = '4232316765',
`id_cidade` = '2' WHERE `id_cliente`=2;
```

6.2 – Alteração de Registros

Como visto no exemplo, a instrução WHERE é responsável por filtrar os registros que serão alterados.

No exemplo anterior, foi utilizado o campo id_cliente, que é chave primária do registro. Logo, somente o cliente com código igual a 12 foi alterado.

Quando a comparação da cláusula **WHERE** for feita com o campo chave primária da tabela, somente um registro será alterado. Mas é possível efetuar alterações em um conjunto de registros. Como ilustra a tabela abaixo:

Sinal	Valor
=	Igual
>=	Maior ou Igual
<=	Menor ou Igual
>	Maior
<	Menor
<>	Diferente

6.2 – Alteração de Registros

- No exemplo abaixo, será alterado o salário (campo salario_fixo) de todos os funcionários que não estão alocados no departamento 1:

```
UPDATE `funcionario` SET `salario_fixo` =600 WHERE `id_departamento`<>1;
```

Observação: Se o comando **UPDATE** for feito sem a cláusula **WHERE**, todos os registros da tabela serão alterados. Por isso, a não ser que se queira alterar todos os registros, a cláusula WHERE sempre deve ser colocada.

6.3 – Excluindo Registros

- Para excluir registros de uma tabela, utiliza-se o comando **SQL DELETE,** seguido do nome da tabela e condição de exclusão**.**

- A sintaxe do comando DELETE é mostrada no quadro abaixo:

DELETE FROM <Tabela**> WHERE** <Campo> = ‘<Valor Campo Identificador>’;

6.3 – Excluindo Registros

- O exemplo abaixo ilustra a exclusão de um registro cujo cliente tem o codigo igual a 10:

```
CREATE TABLE IF NOT EXISTS cliente(
  `id_cliente` INT PRIMARY KEY AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `endereco` VARCHAR(45) NULL,
  `id_cidade` INT NULL,
  `telefone` VARCHAR(11) NULL
);
```

```
DELETE FROM `cliente` WHERE `id_cliente`=10;

Ou

DELETE FROM `cliente` WHERE `nome`='Felipe Melo';
```