APOSTILA DE SQL

Introdução:

SQL – Structure Query Language – é a linguagem mais comumente utilizada para criar, modificar, recuperar e manipular dados de sistemas gerenciadores de bancos de dados relacionais (SGBD).

Ela foi criada no ano de 1970 pela IBM inicialmente para a recuperação e manipulação de dados do sistema de banco de dados "System R" e em primeiro momento foi chamada de **SEQUEL**. Posteriormente foi abreviada para SQL pois a palavra SEQUEL já era marca registrada de uma companhia aérea inglesa da época.

Após algum tempo, a IBM em paralelo a outra companhia Relational Software (hoje Oracle), visualizou o potencial comercial dos conceitos do SQL e lançaram no verão de 1979 o primeiro banco de dados relacional totalmente baseado no SQL, o "Oracle 2".

Pelo grande sucesso que teve, o SQL foi adotado como padrão ANSI em 1986 e ISO em 1987.

Estrutura da linguagem SQL:

Apesar de definida pelos padrões ANSI e ISO, existem muitas variações da linguagem, como o PL/SQL da Oracle, SQL PL da IBM, e o Transact-SQL da Microsoft. Todas seguem as linhas base, definidas pelo padrão, mas implementam características próprias, como tipos de dados e funções específicas.

O SQL foi planejado para atender a um propósito específico: consultar e manipular dados contidos em bancos de dados relacionais. Por isso, o SQL "puro" não é uma linguagem de programação clássica. Mas por outro lado, não deixa de ser uma linguagem de programação, por ter uma sintaxe específica e o objetivo de ser programável.

Neste material utilizaremos o padrão Transact-SQL, pois o foco será no uso de SGDB's Microsoft (SQL Server 2005 e 2008).

Podemos separar a linguagem em dois subconjuntos de objetivos definidos:

DDL – Data Definition Language (Compõe as funções de manipulação de estrutura das tabelas)

É o subconjunto de comandos que serve para a definição do banco de dados.
Seus principais comando são:

- **CREATE (cria banco de dados, tabelas, índices, etc...);**
- **ALTER (altera a estrutura de uma tabela);**
- **DROP (apaga uma tabela ou um índice existente no banco de dados).**

DML – Data Manipulation Language (Compõe as funções de manipulação de estrutura dos dados)

É o subconjunto de comandos responsável por promover a manutenção dos dados, como inserção, alteração, exclusão e recuperação dos dados e tabelas do banco de dados.

- **INSERT (usado para inserir dados em uma tabela do bando de dados);**
- **UPDATE (usado para modificar valores em um campo ou conjunto de linhas);**
- **DELETE** (**remove linhas existentes no banco de dados);**
- **SELECT (seleciona e exibe dados em uma ou mais tabelas).**

Tipos de Dados:

Ao se criar uma tabela, cada coluna possuirá uma definição que indica o tipo de dado que será armazenado na coluna. Os tipos possíveis são definidos pelo padrão ANSI, mas as implementações diferem de banco para banco.
Como trabalharemos com o SQL Server, seguem abaixo alguns dos tipos de dados mais utilizados:

- BIGINT – números inteiros entre -9223372036854775808 e 9223372036854775807;
- INT - números inteiros entre -2147483648 e 2147483647;
- FLOAT – números de ponto flutuante, ou seja, não inteiros;
- REAL – números de ponto flutuante com o dobro de tamanho do float;
- DECIMAL ou NUMERIC – números com casas decimais entre -10^{38} e $10^{38}-1$;
- MONEY – valores monetários entre -2^{63} e $2^{63}-1$;
- SMALLMONEY – valores monetários entre -2147483648 e 2147483647;
- BIT – números inteiros com valores 1 ou 0;
- BINARY – dados binários de tamanho fixo com capacidade máxima de 8000 bytes;
- IMAGE – dados binários de tamanho variável com capacidade máxima de $2^{31}-1$;
- DATETIME – data e hora entre 01/01/1753 a 31/12/9999, precisão de 3,33 milissegundos;
- SMALLDATETIME – data e hora entre 01/01/1900 a 06/06/2079;
- CHAR – caracteres não-Unicode de tamanho fixo, máxima de 8000 caractéres;
- NCHAR - caracteres Unicode de tamanho fixo, máxima de 4000 caractéres;
- VARCHAR – coluna não-Unicode de tamanho variável, máxima de 8000 caracteres;
- NVARCHAR – coluna Unicode de tamanho variável, máxima de 4000 caracteres;
- TEXT - coluna não-Unicode de tamanho variável, máxima de$2^{31}-1$;
- NTEXT - coluna Unicode de tamanho variável, máxima de$2^{30}-1$;

UTILIZANDO O COMANDO CREATE:

* Criação de banco de dados.

CREATE DATABASE **CURSOSQL**;

Com o comando acima, criamos o banco de dados de nome CURSOSQL. Onde abrimos o comando com o CREATE, e logo após informamos o que será criado DATABASE e em seguida o nome da base de dados.

* Criação de tabelas.

```
CREATE TABLE CLIENTES
(
        COD_Cliente INT  Not Null,
        Nome VARCHAR(70) Not Null,
        Fone VARCHAR(20),
        Sexo CHAR(1) default 'M',
        Endereco VARCHAR(70),
        Bairro VARCHAR(30),
        Cidade VARCHAR(30),
        UF CHAR(2),
        CEP VARCHAR(9)
);


CREATE TABLE PEDIDOS
(
        COD_Pedido INT Not Null,
        COD_Cliente INT,
        Data_Pedido DATETIME Not Null,
        Valor NUMERIC(9,2)
);
```

Repare que nas tabelas criadas acima, não existe nenhuma identificação que permite ao usuário identificar de maneira única cada linha constante.
Essa identificação é chamada de **Chave Primária** (Primary Key) e pode ser composta de um ou vários campos da tabela.
Se for composta de um único campo, a informação contida nele não poderá se repetir em mais nenhuma outra linha da tabela. Caso seja composta de vários campos, a composição dos dados não poderá ser repetir.
Existe ainda uma condição para que a chave primária seja informada, a coluna deve ser definida com não nula, ou seja, **Not Null**.

Ainda nos exemplos acima, não foi definido o relacionamento entre as duas tabelas.
Esse conceito da modelagem de dados é chamado de **Chave Estrangeira** (Foreign Key) e serve para indicar uma coluna ou conjunto de colunas de uma tabela, que representa a chave primária de outra tabela.
Basicamente é o conceito que indica que uma tabela está relacionada com outra (ou outras), ou seja, no exemplo acima, quando um pedido for inserido, devemos informar na coluna da chave estrangeira, um código de cliente válido, pois assim cada pedido será relacionado ao cliente correspondente e evitará que exista linhas órfãs (pedido sem clientes).

Tanto a Chave Primária quanto a Chave Estrangeira, podem ser definidas na criação das tabelas pelo comando CREATE TABLE, ou podem ser definidas após a criação das tabelas, em um processo de alteração, utilizando o comando ALTER TABLE.

Veja abaixo como deverá ficar o código de criação das tabelas já com a chave primária e a chave estrangeira definida:

```
CREATE TABLE CLIENTES
(
        COD_Cliente INT  Not Null,
        Nome VARCHAR(70) Not Null,
        Fone VARCHAR(20),
        Sexo CHAR(1) default 'M',
        Endereco VARCHAR(70),
        Bairro VARCHAR(30),
        Cidade VARCHAR(30),
        UF CHAR(2),
        CEP VARCHAR(9)
        PRIMARY KEY (COD_Cliente)
);

CREATE TABLE PEDIDOS
(
        COD_Pedido INT Not Null,
        COD_Cliente INT,
        Data_Pedido DATETIME Not Null,
        Valor NUMERIC(9,2)
        PRIMARY KEY (COD_Pedido)
        FOREIGN KEY (COD_Cliente) REFERENCES CLIENTES
);
```

Agora veja como deveriam ser criados os comandos de alteração das tabelas para adicionar as chaves primárias e chave estrangeira do primeiro exemplo:

Alteração da tabela CLIENTES – inclusão da chave primária.

```
ALTER TABLE CLIENTES ADD PRIMARY KEY (COD_Cliente)
```

Alteração da tabela PEDIDOS – inclusão da chave primária e da chave estrangeira

```
ALTER TABLE PEDIDOS ADD PRIMARY KEY (COD_Pedido)

ALTER TABLE PEDIDOS ADD FOREIGN KEY (COD_Cliente) REFERENCES CLIENTES
```

OBS: Na criação das tabelas foram utilizadas as designações **Not Null** e **Default.**
A cláusula Not Null indica que ao inserir uma linha na tabela, esse campo não poderá ter um valor nulo ou vazio. A utilização da designação Null / Not Null é opcional na maioria dos servidores, porém, a designação padrão pode ser diferente dependendo do padrão do banco de dados utilizado. Como exemplo, temos o Interbase/Firebird que acompanha a determinação do padrão ANSI que define como Null as colunas onde a designação não é informada. Já o SQL Server, define como Not Null quando a designação não é informada, sendo assim, devemos ter atenção no uso delas.

Já a cláusula Default, é utilizada para atribuir automaticamente um valor padrão para a coluna, quando este não for definido.

ALTERANDO TABELAS EXISTENTES:

* Adicionando colunas na tabela

ALTER TABLE PEDIDOS **ADD** Data_Entrega DATETIME, Situacao VARCHAR(20) Default 'Aberto'

Veja no exemplo acima, que a tabela Pedidos teve duas colunas adicionadas: Data_Entrega e Situacao.

* Alterando colunas na tabela

ALTER TABLE PEDIDOS **DROP** Situacao, **ADD** Posicao VARCHAR(20) Default 'Aberto'

Repare no exemplo acima que a coluna Situacao foi apagada e em seu lugar foi criada a coluna Posicao com um valor padrão (Aberto).

* Apagando colunas na tabela

ALTER TABLE PEDIDOS **DROP** Data_Entrega, Posicao

Repare no exemplo acima que as colunas Data_Entrega e Situacao foram apagadas.

OBS: Pode-se apagar também uma tabela inteira do banco de dados. Para isso, basta executar o comando **DROP TABLE** e informar o nome da tabela a ser excluída. Devemos ter cuidado ao executar este comando, pois ele excluirá toda a tabela junto com os seus dados.

INSERINDO DADOS NAS TABELAS EXISTENTES:

INSERT INTO CLIENTES
(COD_Cliente, Nome, Fone, Sexo, Endereco, Bairro, Cidade, UF, CEP)
VALUES
(01, 'Joao Silva', '21-2345-5678', 'M', 'Rua Uruguaiana 500', 'Centro', 'Rio de Janeiro', 'RJ', '20003-009')

INSERT INTO PEDIDOS
(COD_Pedido, COD_Cliente, Data_Pedido, Valor)
VALUES
(10001, 01, '2010-07-18', 275.50)

OBS: Valores em string ou data devem conter aspas (simples ou duplas dependendo do padrão do SQL utilizado) antes e depois do valor, e os valores numéricos devem ser passados diretamente sem as aspas.
Repare que os valores de do tipo data, devem ser informado na seguinte ordem: Ano-Mês-Dia.
Os valores numéricos reais devem ser informados com ponto ao invés de vírgula.

ALTERANDO / ATUALIZANDO DADOS DAS TABELAS:

UPDATE CLIENTES **SET** UF = 'RJ'

UPDATE PEDIDOS **SET** VALOR = 500

OBS: Nos exemplos acima, o comando UPDATE atualizará a UF de todos os clientes para o RJ, e o valor de todos os pedidos para 500.
Na maioria das vezes, isto não é o esperado, ou seja, queremos atualizar valores dentro de um determinado conjunto específico de dados. Para isto deve-se utilizar o comando WHERE como veremos abaixo.

UTILIZANDO A CLÁUSULA WHERE:

A cláusula WHERE é utilizada para limitar um conjunto de dados, seja em uma pesquisa, alteração ou exclusão de dados.
Ela deve ser inserida logo após o comando SQL da operação desejada.

Aproveitando os exemplos de atualização mostrados acima, vamos reescrevê-los de forma que a alteração acontece somente a um determinado conjunto.

UPDATE CLIENTES **SET** UF = 'RJ' **WHERE** COD_Cliente = 01

UPDATE PEDIDOS **SET** VALOR = 500 **WHERE** COD_Pedido = 10001

Repare nos exemplos acima, que agora as alterações somente serão feitas nos registros que atenderem a limitação da cláusula WHERE, ou seja, a UF alterada será somente a do cliente com o código 01 e somente o pedido de número 10001 terá ser valor alterado para 500.

EXCLUINDO DADOS DAS TABELAS:

DELETE FROM CLIENTES **WHERE** COD_Cliente = 99

DELETE FROM PEDIDOS **WHERE** Valor = 275.50

Observe que nos exemplos acima, a exclusão somente acontecerá nos registros que satisfizerem a condição da cláusula WHERE.
Se for necessário excluir 'todos' os registros da tabela, basta não utilizar a cláusula WHERE.

DELETE FROM CLIENTES

DELETE FROM PEDIDOS

Devemos ter muito cuidado ao utilizar o DELETE dessa maneira, pois uma vez esses dados excluídos, eles não poderão ser restaurados, a não ser que antes tenhamos utilizado uma transação.

RECUPERANDO DADOS DAS TABELAS:

Como já foi dito anteriormente, o SQL foi planejado com o objetivo de permitir a consulta de dados contidos em tabelas de banco de dados relacionais. Por isso o comando SELECT é o mais importante e elaborado da linguagem.

SELECT * FROM CLIENTES

SELECT * FROM PEDIDOS

Nos comandos acima, a consulta retornará uma listagem com todos os registros e campos contidos nas tabelas. Mas como geralmente precisamos consultar um conjunto de dados específicos, podemos utilizar várias cláusulas para refinar este resultado.
Veja nos exemplos abaixo:

Consulta com ordenação

SELECT * FROM CLIENTES **ORDER BY** Nome

SELECT * FROM PEDIDOS **ORDER BY** Data_Pedido

Consulta com limitação de dados

SELECT * FROM CLIENTES **WHERE** UF = SP

SELECT * FROM PEDIDOS **WHERE** COD_Cliente = 05

Consulta de colunas específicas

SELECT COD_Cliente, Nome, Sexo FROM CLIENTES

SELECT COD_Pedido, COD_Cliente, Data_Pedido FROM PEDIDOS

Consulta com agrupamentos

SELECT COD_Cliente, Valor FROM PEDIDOS **GROUP BY** COD_Cliente

Consulta com colunas de agregação

SELECT COUNT (COD_Cliente) FROM PEDIDOS – conta os registros

SELECT MAX (VALOR) FROM PEDIDOS – extrai o maior valor

SELECT SUM (VALOR) FROM PEDIDOS – soma os valores

Consulta inteligente

SELECT * FROM CLIENTES WHERE NOME **LIKE** ‘A%’

SELECT * FROM PEDIDOS WHERE Data_Pedido **BETWEEN** ‘2010-05-01’ AND ‘2010-05-31’

UTILIZANDO JOINS PARA CONSULTAS COMPOSTAS:

A cláusula WHERE também pode ser utilizada para unir tabelas no comendo SELECT, criando um resultado composto. Para "unir" tabelas em uma cláusula SELECT você deve fazer duas modificações básicas: Especificar tabelas adicionais após a cláusula FROM, e ligar os campos relacionados. Veja o exemplo abaixo:

```
SELECT CLIENTES.COD_Cliente, CLIENTES.Nome, PEDIDOS.Valor
FROM CLIENTES
INNER JOIN PEDIDOS ON
(CLIENTES.COD_Cliente = PEDIDOS.COD_Cliente)
```

O comando acima é conhecido por INNER JOIN ou por LEFT JOIN. Ele é o comando mais comum nas aplicações baseadas em SGDB. O retorno desse comando exibe uma listagem com código do cliente, nome e valores de pedidos existentes para ele, ou seja, ele somente exibe o cliente (tabela a esquerda conhecida com OUTER TABLE) se existir referência na tabela da direita (conhecida como INNER TABLE).

Existe também outros comandos como:

Left Outer Join – Retorna todos os valores das duas tabelas, mesmo que não haja igualdade entre eles.

```
SELECT CLIENTES.COD_Cliente, CLIENTES.Nome, PEDIDOS.Valor
FROM CLIENTES
LEFT OUTER JOIN PEDIDOS ON
(CLIENTES.COD_Cliente = PEDIDOS.COD_Cliente)
```

Right Outer Join – Retorna os valores onde a chave existe na tabela da direita, mas não existe na tabela da esquerda, ou seja, pode-se dizer que é o oposto do INNER JOIN. Geralmente ele é utilizado para achar registros orfãos no banco de dados.

```
SELECT CLIENTES.COD_Cliente, CLIENTES.Nome, PEDIDOS.Valor
FROM CLIENTES
RIGHT OUTER JOIN PEDIDOS ON
(CLIENTES.COD_Cliente = PEDIDOS.COD_Cliente)
```

TRABALHANDO COM SUBQUERIES:

Uma SubQuery é um comando SELECT embutido em um outro comando SQL. Pode ser utilizado como condição em uma cláusula WHERE; pode ser utilizado para retornar um valor como se fosse uma coluna de outro SELECT; e também pode ser utilizado para retornar um valor a ser atribuído a uma coluna no momento de uma alteração ou inclusão. Veja os exemplos abaixo:

<u>Selecionar todos os clientes que tiveram pedidos no mês de Maio.</u>

```
SELECT * FROM CLIENTES
WHERE COD_Cliente IN
(SELECT COD_Cliente FROM PEDIDOS WHERE Data_Pedido BETWEEN '2010-05-01'
AND '2010-05-31')
```

Selecionar todos os clientes e exibir a soma dos valores de pedidos de cada um deles.

```
SELECT COD_Cliente, Nome, (SELECT SUM (Valor) FROM PEDIDOS WHERE
PEDIDOS.COD_Cliente = CLIENTES.COD_Cliente ) AS TOTAL FROM CLIENTES
```

AGRUPANDO E ORDENANDO O RESULTADO DO SELECT:

- **GROUP BY**

A cláusula GROUP BY permite que se agrupe dados e se obtenha somatórios através de funções de agregação. Abaixo reescreveremos o exemplo com subquery mas utilizando o GROUP BY:

```
SELECT CLIENTES.COD_Cliente, CLIENTES.Nome, SUM (PEDIDOS.Valor) AS TOTAL
FROM CLIENTES, PEDIDOS
WHERE CLIENTES.COD_Cliente = PEDIDOS.COD_Cliente
GROUP BY CLIENTES.COD_Cliente, CLIENTES.Nome
```

- **HAVING**

A cláusula HAVING é usada para limitar as linhas retornadas pelo GROUP BY. Ela funciona como o WHERE da cláusula SELECT norma, porém somente se aplica ao resultado obtido pelo GROUP BY. Veja abaixo o comando que retornará somente as linhas cujo valor seja maior que 500.

```
SELECT CLIENTES.COD_Cliente, CLIENTES.Nome, SUM (PEDIDOS.Valor) AS TOTAL
FROM CLIENTES, PEDIDOS
WHERE CLIENTES.COD_Cliente = PEDIDOS.COD_Cliente
GROUP BY CLIENTES.COD_Cliente, CLIENTES.Nome
HAVING SUM (PEDIDOS.Valor) > 500
```

- **ORDER BY**

A cláusula ORDER BY é utilizada para ordenar os resultados da seleção por um de seus campos.

```
SELECT * FROM CLIENTES ORDER BY Nome
```

- **DISTINCT**

Essa cláusula evita a repetição da mesma informação em uma seleção.
Exemplo: Extrair da tabela de clientes uma listagem das UF's aonde possui cliente cadastrado.

```
SELECT DISTINCT UF FROM CLIENTES
```

TRABALHANDO COM TRANSAÇÕES NA MANIPULAÇÃO DOS DADOS:

Quando falamos de manipulação de dados, é extremamente necessário que antes de pense na segurança e integridade dos dados. Por estes motivos, antes de executar qualquer comando em SQL de inserção, alteração e/ou exclusão, devemos sempre iniciar a transação de dados do SGDB. Essa transação permite que se execute o comando SQL, veja o seu resultado, e confirme se ele for o esperado ou retorne ao estado anterior caso o resultado não lhe agrade.

Veja abaixo a estrutura da transação:

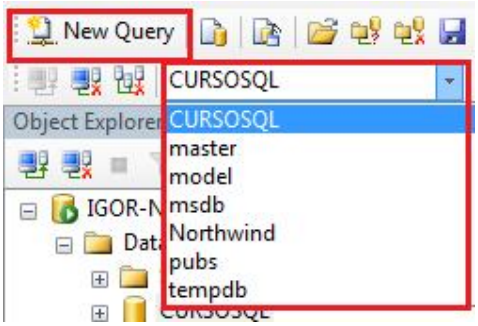
BEGIN TRANSACTION - Deve ser inserido antes do comando SQL;

COMMIT - Confirma o comando SQL executado;

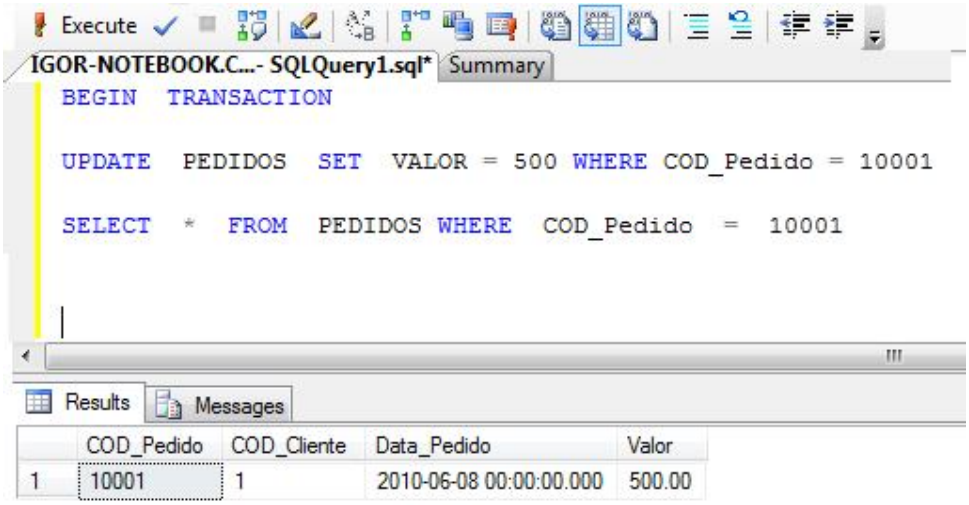
ROLLBACK - Cancela o comando SQL executado.

Exemplo prático:

Abra o Management Studio escolha a database e clique em New Query:



Será aberta a tela de scripts para a database informada. Nela digite o comando a ser executado precedido da abertura da transação, e após clique em Execute:



Veja que o comando acima abriu a transação; executou uma alteração no valor do Pedido 10001 para R$ 500.00; leu a tabela de pedidos selecionando o pedido 10001 exibindo já a sua alteração.

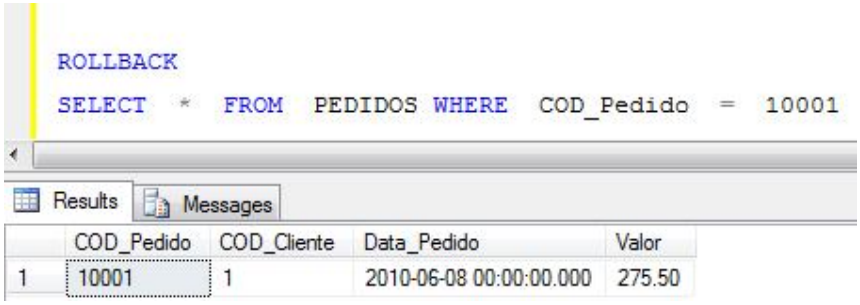
Como abrimos uma transação temos que confirmar essa alteração ou desistir dela, para isso temos que executar logo após o primeiro bloco de comandos, o comando COMMIT para confirmar a alteração, ou o ROLLBACK para desistir dela. Veja abaixo:

```
COMMIT
SELECT * FROM PEDIDOS WHERE COD_Pedido = 10001
```

Results | Messages

	COD_Pedido	COD_Cliente	Data_Pedido	Valor
1	10001	1	2010-06-08 00:00:00.000	500.00

Acima executamos a confirmação do bloco de comandos de alteração e exibimos novamente o registro alterado, repare que o valor realmente foi alterado para R$ 500.00.



```
ROLLBACK
SELECT * FROM PEDIDOS WHERE COD_Pedido = 10001
```

Results | Messages

	COD_Pedido	COD_Cliente	Data_Pedido	Valor
1	10001	1	2010-06-08 00:00:00.000	275.50

Agora veja que desistimos da alteração, ou seja, mandamos desfazer o bloco de comandos executado. Logo após, exibimos o registro aonde a alteração foi desfeita. Repare que o seu valor voltou a ser R$ 275.50.

OBS: Na prática, a utilização de transações é muito útil para realizar manutenção nos bancos de dados, pois dá segurança ao usuário de poder desfazer a alteração caso o resultado não seja o esperado.