O SQL Server não é case sensitive, ou seja, ele não faz diferenciação entre letras maiúsculas e minúsculas.

REVISÃO – BANCO DE DADOS 1:

Comandos básicos:

- INSERT: O comando para inclusão de dados, e possuía a seguinte sintaxe:

```
INSERT INTO nome_tabela (lista-de-campos)
VALUES (lista_dados)
```

Onde:

- Nome_tabela: nome da tabela no qual será inserido os dados.
- Lista-de-campos: nome das colunas que receberão os valores.
- Lista-dados: valores que serão inseridos na tabela. Estes campos devem estar na mesma ordem descrita em lista-de-campos, todos separados por vírgula. Se for utilizado um comando SELECT o mesmo deve retornar a mesma quantidade de colunas com os mesmos tipos de dados especificados em lista-de-campos.

Exemplo:

```
INSERT INTO EMPREGADOS (CODIGO, NOME, SALARIO, SECAO)
VALUES (1, "HELBERT CARVALHO", 1.500, 1)
```

- SELECT: Através desde comando passa-se valores:

```
INSERT INTO EMPREGADOS (CODIGO,NOME, SALARIO, SECAO)
SELECT CODIGO,NOME,SALARIO, SECAO
FROM EMPREGADOS_FILIAL
WHERE DEPARTAMENTO = 2
```

Neste comando todos os empregados da tabela EMPREGADOS_FILIAL foram cadastrados na tabela EMPREGADOS. Se o nome dos campos não for citado no comando INSERT, o SELECT deverá retornar valores compatíveis para todos os campos disponíveis.

- UPDATE: comando para atualizar registro. Com a seguinte sintaxe:

```
UPDATE nome_tabela
SET CAMPO = "novo_valor"
WHERE CONDIÇÃO
```

Onde:

- Nome_tabela: nome da tabela que será modificada
- Campo: campo que terá seu valor alterado
- Novo_valor: valor que substituirá o antigo dado cadastrado em campo
- Where: Se não for informado, a tabela intera será atualizada
- Condição: regra que impõe condição para execução do comando

Exemplo de uso do comando update:

```
UPDATE DEPARTAMENTO
SET SALARIO = 1000
WHERE CODIGODEP = 1
```

No trecho acima, todos os colaboradores que fazem parte do departamento 1 terá o salário alterado para 1000.

Podemos combinar o comando SELECT com UPDATE.

Exemplo: Os funcionários de menor salário receberão aumento de 10%.

```
UPDATE EMPREGADOS
SET SALARIO = salario * 1.1
WHERE SALARIO = (SELECT MIN(salario) FROM EMPREGADOS)
```

O comando SELECT também pode ser utilizado na atribuição de valor ao campo:

UPDATE passando SELECT como valor

```
UPDATE EMPREGADOS
SET SALARIO = (SELECT MAX(salario) FROM EMPREGADOS)
WHERE DEPARTAMENTO = 5
```

- DELETE: comando utilizado para apagar dados. Sintaxe:

```
DELETE FROM nome_tabela
WHERE condição
```

Onde:

- Nome_tabela: nome da tabela que será modificada
- Where: cláusula que impõe uma condição sobre a execução do comando

Exemplo:

```
DELETE FROM EMPREGADOS
WHERE CODIGO = 125
```

Create

Após inicializar o SQL Server, clique em New Query.
A sintaxe do comando é: CREATE DATABASE nome_do_banco.
Pressione a tecla F5.
A mensagem (*Command(s)completed successfully*) deve aparecer confirmando que seu banco foi criado com sucesso.

CREATE TABLE é o comando para criação da tabela e deve ser seguida pelo nome que daremos à tabela. Dentro do comando, devemos definir os nomes dos campos de acordo com a conveniência do banco de dados, e determinar o tipo de dado que poderá ser incluído neste campo.
PRIMARY KEY define a chave primária da tabela, isto é, o campo que serve como chave da tabela e que não pode ser repetido.

A sintaxe básica para criarmos é:

```
CREATE TABLE nome_tabela
(
nome_campo_1  tipo_1,
nome_campo_2  tipo_2,
...
nome_campo_n tipo_n,
PRIMARY KEY ( campo_x,...));
```

Se desejamos que um campo seja de preenchimento obrigatório,

devemos inserir NOT NULL na frente do campo determinado.

```
CREATE TABLE nome_tabela
(
nome_campo_1 tipo_1 NOT NULL,
nome_campo_2 tipo_2,
...
nome_campo_n tipo_n,
PRIMARY KEY(campo_x,...));
```

Se desejamos que um campo seja de auto-incremento,
devemos inserir AUTO_INCREMENT na frente do campo determinado.

```
CREATE TABLE nome_tabela
(
nome_campo_1 tipo_1 NOT NULL  AUTO_INCREMENT,
nome_campo_2 tipo_2,
...
nome_campo_n tipo_n,
PRIMARY KEY (campo_x,...));
```

- >Drop: O comando DROP DATABASE é utilizado para a remoção de um determinado banco de dados.

Eliminando todas as tabelas e estruturas que possam estar associadas a ele.
A sintaxe para a execução deste desde comando é aseguinte:

```
DROP DATABASE nome_do_banco.
```

~ BANCO DE DADOS 2 ~

JOIN:

O primeiro passo é criar as tabelas A e B:

```
CREATE TABLE TabelaA(
  Nome varchar(50) NULL
)

GO

CREATE TABLE TabelaB(
  Nome varchar(50) NULL
)
```

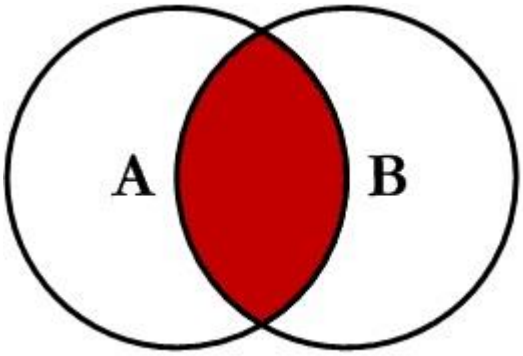
O segundo passo inclui a inserção de valores nas tabelas A e B:

```
INSERT INTO TabelaA VALUES('Fernanda')
INSERT INTO TabelaA VALUES('Josefa')
INSERT INTO TabelaA VALUES('Luiz')
```

```
INSERT INTO TabelaA VALUES('Fernando')

INSERT INTO TabelaB VALUES('Carlos')
INSERT INTO TabelaB VALUES('Manoel')
INSERT INTO TabelaB VALUES('Luiz')
INSERT INTO TabelaB VALUES('Fernando')
```

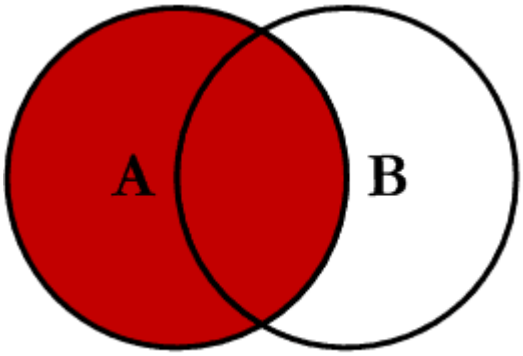
Agora, vamos analisar cada um dos joins:

- INNER JOIN: Usando o inner join, teremos como resultado todos os registros comuns nas duas tabelas.



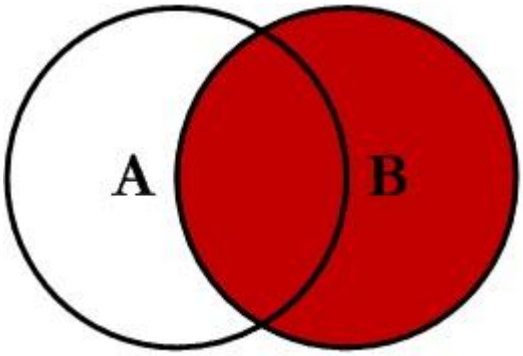

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
INNER JOIN TabelaB as B
                on a.Nome = b.Nome
```

- LEFT JOIN: Usando o Left Join, teremos como resultado todos os registros que estão na tabela A (mesmo que não estejam na tabela B) e os registros da tabela B que são comuns na tabela A.



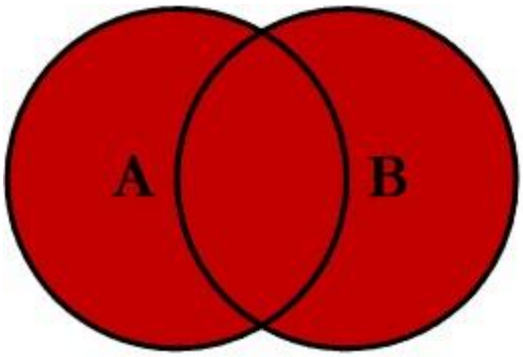

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
LEFT JOIN TabelaB as B
                on a.Nome = b.Nome
```

- RIGHT JOIN: Usando o Right Join, teremos como resultado todos os registros que estão na tabela B (mesmo que não estejam na tabela A) e os registros da tabela A que são comuns na tabela B.



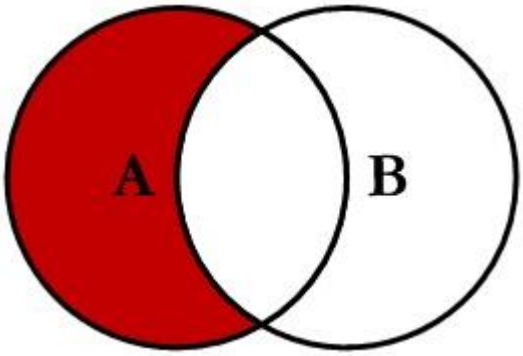

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
RIGHT JOIN TabelaB as B
                on a.Nome = b.Nome
```

- OUTER JOIN: Usando o Outer Join (conhecido por Full Outer Join ou Full Join), teremos como resultado todos os registros que estão na tabela A e todos os registros da tabela B.



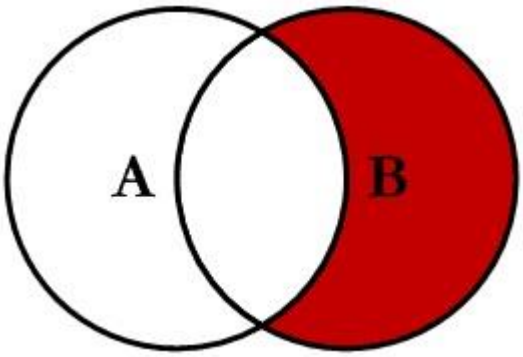

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
FULL OUTER JOIN TabelaB as B
                on a.Nome = b.Nome

SELECT a.Nome, b.Nome
FROM TabelaA as A
FULL JOIN TabelaB as B
                on a.Nome = b.Nome
```

- LEFT EXCLUDING JOIN: Left Excluding Join, que retorna como resultado todos os registros que estão na tabela A e que não estejam na tabela B



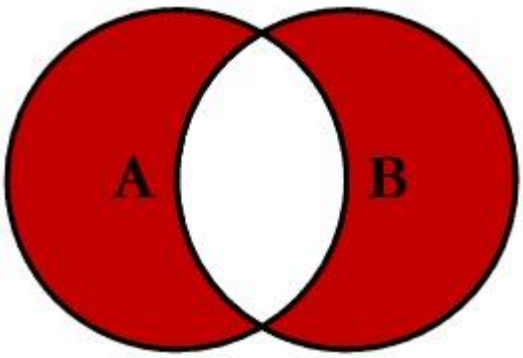

```
SELECT a.Nome, b.Nome
FROM TabelaA as A
LEFT JOIN TabelaB as B
                on a.Nome = b.Nome
WHERE b.Nome is null
```

- RIGHT EXCLUDING JOIN: Right Excluding Join, teremos como resultado todos os registros que estão na tabela B e que não estejam na tabela A.




```
SELECT a.Nome, b.Nome
FROM TabelaA as A
RIGHT JOIN TabelaB as B
                on a.Nome = b.Nome
WHERE a.Nome is null
```

- RIGHT EXCLUDING JOIN: Outer Excluding Join, teremos como resultado todos os registros que estão na tabela B (que não estejam na tabela A) e todos os registros que estão na tabela A (que não estejam na tabela B).



```
SELECT a.Nome, b.Nome
FROM TabelaA as A
FULL OUTER JOIN TabelaB as B
                on a.Nome = b.Nome
WHERE a.Nome is null or b.Nome is null

SELECT a.Nome, b.Nome
FROM TabelaA as A
FULL JOIN TabelaB as B
                on a.Nome = b.Nome
WHERE a.Nome is null or b.Nome is null
```