***Curso Udemy – SQL***

**Banco de Dados** = É um conjunto organizado de dados, tipicamente armazenado em formato eletrônico. É um local de armazenamento de informações.

**Tipos de Bancos de Dados**

\*Bancos de dados de tipo plano (flat-type databes)

\*Bancos de dados hierárquicos (Hierarchial databases)

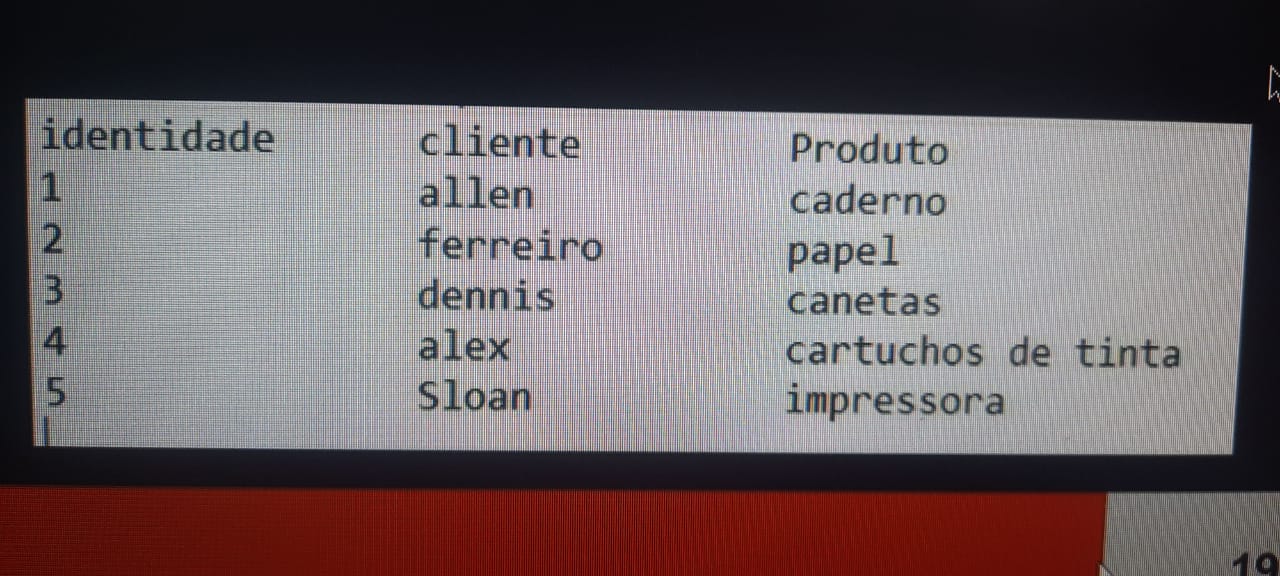
\*Bancos de dados relacionais (Relational databases)

Observação: Cada tipo de Banco possui seus próprios recursos de desing importantes.

**1.Flat-type databases**

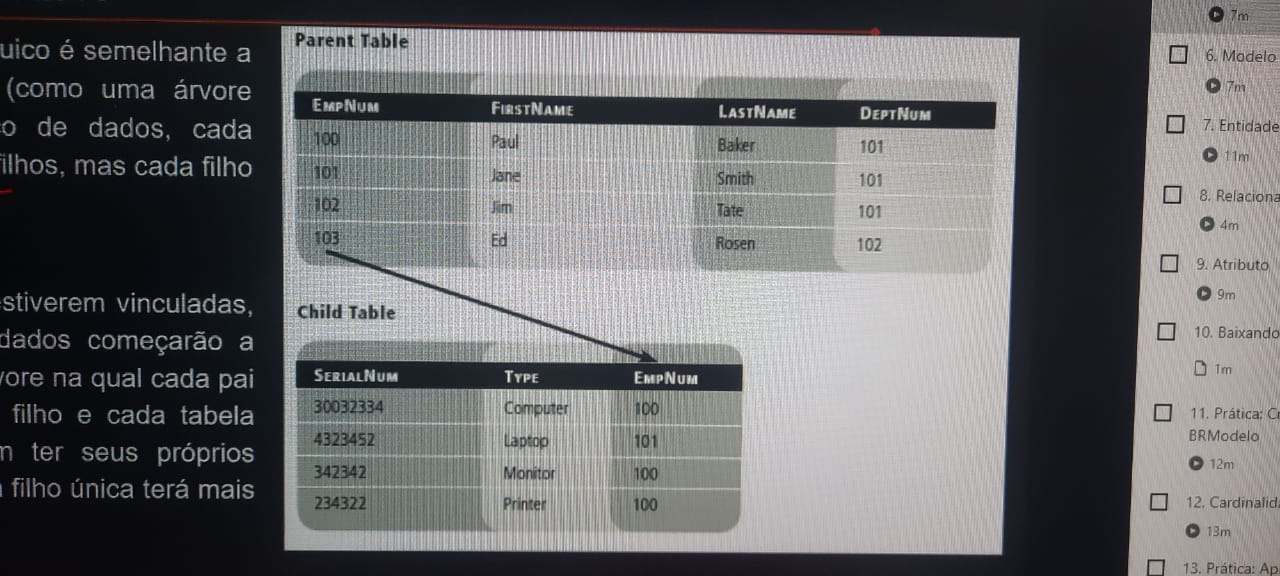
Bancos de Dados de tipo plano são simplistas em design. Eles são mais comumente usados em formatos de textos simples. Como o proposito deles é manter um registro por linha, eles tornam o acesso, o desempenho e as consultas muito rápidos.

Exemplo desse Banco : Seria em arquivo .txt, .csv ou .ini.

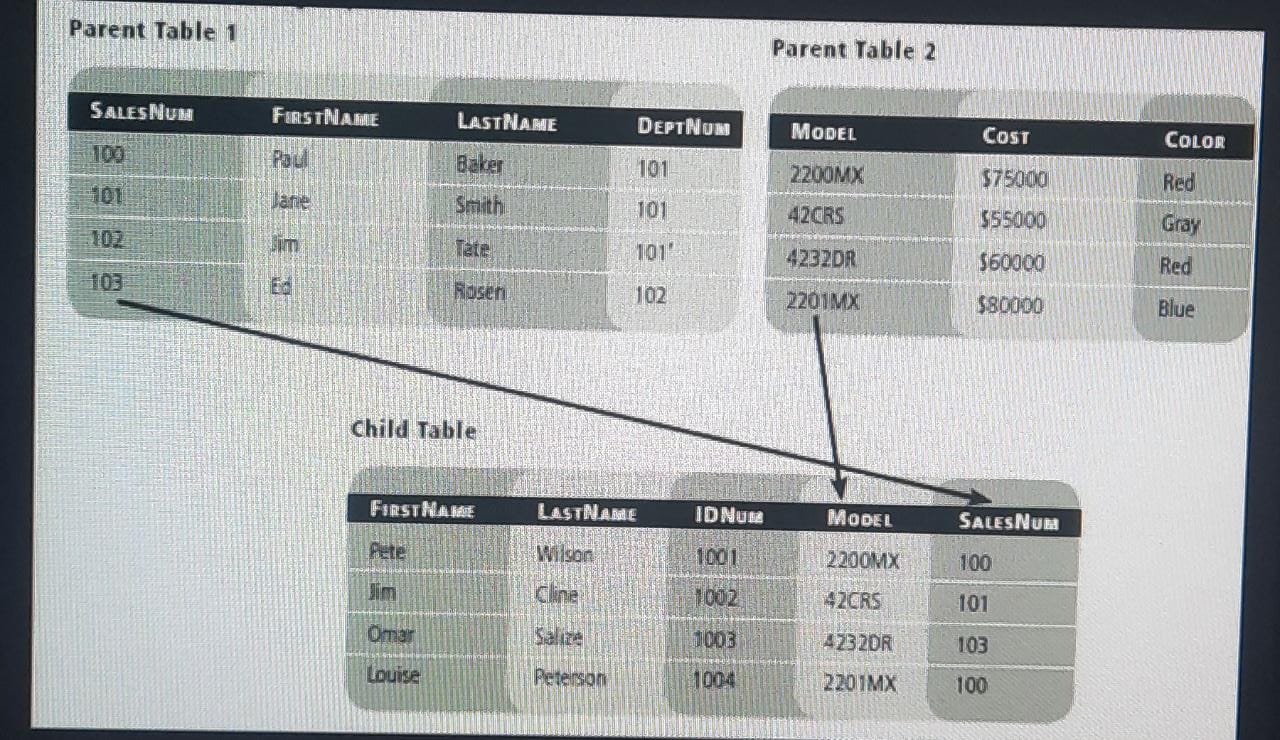
Bancos de dados desse tipo são considerados “planos” porque são tabelas bidimensionais consistindo de linhas e colunas.

**2.Hierarchical databases**

Um bancos de dados hierárquico é semelhante a uma estrutura em árvore (como uma árvore genealógica). Nesse banco de dados , cada tabela “pai” pode ter vários filhos, mas cada filho pode ter apenas um pai.



**3.Relational databases**

É um banco semelhante a um banco de dados hierárquico. Diferentemente dos bancos de dados hierárquicos, no banco relacional, uma tabela em um banco de dados relacional pode ter vários pais.

**Modelos de Dados**

Um modelo de dados mostra a estrutura lógica de um banco de dados, incluindo as relações e restrições que determinam como os dados podem ser armazenados e acessados.

***Normalização de Dados e de sistemas***

**Normalização de Dados** = Técnica da análise de Dados que visa determinar a melhor formação para uma estrutura de dados.

Objetivos: \*Eliminar detalhes que dificultam as operações sobre os dados. \*minimizar as redundâncias e os consequentes riscos de inconsistências. \*Reduzir e facilitar manutenções.

**Análise da Organização do Sistema** = É super, importante pois através disso pode-se evitar problemas futuros que podem gerar vários gastos a empresa.

**Como evitar problemas futuros** = Usando a técnica de normalização pois produz sistemas estáveis e confiáveis.

**Sistema Normalizado** = Significar normalizar ou seja dividir e cada divisão conterá um tipo de informação.

\*Organizar de forma coerente cada Informação do sistema\*

**1FN (1ª Forma Normal)**

Refere-se ao formato de um registro. Esta diretriz exclui a possibilidade de haverem campos que possuem mais de um atributo, ou seja, um vetor ou grupo de atributos.

(É a forma que só pode possuir atributo simples e não pode possuir atributos compostos e mais de uma PK)

**2FN (2ª Forma Normal)**

Boson :

\*Está na 1FN

\*Todos os atributos não-chave são funcionalmente dependentes de todas as partes da chave primária.

\*Não existem dependências parciais, e atributos não dependem de chaves candidatas.

Caso contrário, deve-se gerar uma nova tabela com os dados.

Um atributo-chave é um que é uma PK ou parte de uma PK composta.

-----------------------------------------------------------------------------------------

Esta forma normal refere-se ao relacionamento entre atributos dentro de uma determinada tabela, mais especificamente entre atributos chave e não-chave, tomaremos daqui em diante “chave” como chave primária, não considerando chaves estrangeiras.

Um esquema está na Segunda Forma Normal (2FN) se está na 1FN e se cada atributo não chave de uma tabela refere-se à chave da mesma **como um todo.**

(É a forma em que os campos na tabela precisão está ligado uma única PK e por contrario é necessário organizar e torna-la organizável).

**3FN (3ª Forma Normal)**

Boson:

\*Estiver na 2FN

\*Não existirem dependências transitivas.

\*Uma tabela está na Terceira Forma Normal se ela estiver na segunda forma normal e se nenhuma coluna não-chave depender de outra coluna não-chave.

Dependência transitiva é uma dependência funcional entre dois ou mais atributos não-chave.

Caso contrário, deve-se gerar uma nova tabela.

A Terceira Forma Normal (3FN) é bastante semelhante a segunda e também refere-se a chaves. Além disso, ela também requer que antes a Primeira Forma Normal seja satisfeita. Esta diretriz é violada quando um atributo não chave refere-se a outro atributo que também não é chave dentro de uma tabela.

Um esquema está na Segunda e Terceira Forma Normal se todo campo é parte da chave ou provê um fato sobre toda a chave, nunca uma parte dela.

(É a forma quase idêntica a segunda , ela não fica formal a 3FN quando o atributo não chave esta com ligação a outro atributo na mesma tabela sendo que ele também não é PK. Pra ficar correto o campo deveria está ligado a PK e não metade).

***Categoria dos tipos de dados do SQL Server***

No SQL Server, cada coluna, variável, expressão e parâmetro tem um tipo de dados relacionado. O tipo de dado especifica o que pode ser colocado no objeto. A integridade do banco de dados depende decisivamente dos tipos de dados com escopo apropriado.

Os tipos de dados que o objeto pode assumir são: dados inteiros, dados de caractere, dados monetários, data e hora, cadeias de caracteres binárias etc.

\* Os tipos de dados em SQL Server são organizados nas seguintes categorias:

- **Númericos**:

-Númericos exatos

# bigint # numeric # bit # smallint # decimal

# smallmoney # int # tinyint # money

-Númericos aproximados

# float # Real

-**String**

-Cadeias de caracteres

# char #vachar #text

-Cadeias de caracteres Unicode

# nchar #nvchar # ntext

-Cadeia de caracteres binária

# binary # varbinary # imagem

**-Data e hora**

# date # datetimeoffset # datetime2

# smalldatetime # datetime # time

**-Outros tipos de dados**

# cursor # rowversion # hierarchyid

# uniqueidentifier # sql\_variant # xml

# Tipos de geometria espacial

# Tipos de geografia espacial # Table

\***Conhecer mas os dados** = <https://docs.microsoft.com/pt-br/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-2017>

***SQL***

Structured Query Language ou linguagem de Consulta Estrutura ou SQL, é a linguagem de pesquisa declarativa padrão para banco de dados relacional (base de dados relacional). Muita das características originais do SQL forma inspiradas na álgebra relacional. Tipicamente a linguagem SQL ANSI é a linguagem de SQL padrão.

Pode ser migrada de plataforma, para plataforma sem mudanças estruturais principais.

**Variações de SQL ANSI**

SQL Server: Transact -SQL

Oracle: PL/SQL

PostegreSQL: PL/pgSQL

***Grupos de comandos do SQL***

A linguagem de SQL se subdivide em cinco grupos de comandos. Alguns autores apenas consideram 3 ou 4 grupos. Cada grupo tem seus comandos específicos, tendo cada um uma finalidade específica.

\*DDL

- USE

-CREATE

-ALTER

-DROP

-TRUNCATE

\*DML

-INSERT

-UPDATE

-DELETE

-MERGE

\*DCL

-GRENT

-REVOKE

-DENY

\*DTL

-BEGIN

TRANSACTION

-COMMIT

-ROLLBACCK

\*DQL

-SELECT

Observação:

- O comando SELECT para prova é designado como DML;

***Definição dos Comandos***

**DDL – DATA DEFINITION LANGUAGE**

O DDL é usado para criar, alterar e deletar objetos do banco de dados. Podendo ser Banco de Dados, tabelas, views, procedures, funtions, index, e etc.

Principais Comandos:

- USE = Muda o contexto da janela de consulta;

- CREATE = Criar objetos no banco de dados;

- ALTER = Altera objetos no banco;

- DROP = Apaga objetos do banco;

- TRUNCATE = Ele é DML também, apaga todos os registros de uma tabela, reiniciando o índice das colunas autoincrementais.

**Diferenças entre os comandos SQL Server DELETE e TRUNCATE**

\*Truncar redefine os valores de identidade, enquanto a exclusão não;

\*Truncar remove todos os registros e não aciona gatilhos;

\*Truncar é mas rápido do que excluir, pois faz menos uso dos log da transação;

\*Truncar não é possível quando uma tabela é referenciada por uma Chave Estrangeira (Foreing Key) ou tabelas são usadas na replicação ou com visualizações indexadas.

**DML – DATA MANIPULATION LANGUAGE**

O DML é usado para a manipulação de registros do banco de dados.

Seus comandos são:

-INSERT = Insere registros no banco;

-UPDATE = Atualiza registros no banco.

-DELETE = Deleta/Apaga registro no banco.

-MERGE = Executa operações de inserção, atualização ou exclusão em uma tabela de destino com base nos resultados de uma junção com uma tabela de origem.

Obs.: Para a certificação MTA Database Fundamentais o comando SELECT é caracterizado como um comando DML (Verificar o livro oficial).

**DCL – DATA CONTROL LANGUAGE**

O DCL controla a segurança dos dados, definido quem tem acesso a determinado tipo de operação.

Seus comandos são:

-GRANT = Usado para a permissão a um determinado usuário para executar um procedimento.

-REVOKE = Desfaz uma permissão , seja uma GRANT ou DENY.

-DENY = Negar permissão de acessos de dados.

**DTL – DATA TRANSACTION LANGUAGE**

O DTL é usado para a manipulação de transações (inserção, alteração, remoção e consulta) no banco de dados.

Seus comandos são:

-BEGIN TRAN OU (BEGIN TRANSACTION) = Utilizado para iniciar uma transação.

-COMMIT = Utilizado para gravar as transações no banco.

-ROLLBACK = Utilizado para desfazer as alterações de uma transação no banco.

Obs1: Alguns SGBDs são auto-commit, ou seja, uma vez executado comandos do tipos DML os dados são gravados automaticamente no banco de dados.

Obs2: Ao realizar a abertura de uma transação com o BEGIN TRAN, a transação deve ser finalizada com os comandos COMMIT ou ROLLBACK.

Alguns sistemas travam quando uma transação é aberta e não é finalizada.

**DQL – DATA QUERY LANGUAGE**

O DQL é usado para a consultar dados no banco.

Seu comando é:

-SELECT = Usado para consultar informações ao registro das tabelas do banco de dados.

***Dicas sobre os comandos SQL***

É importante criar um padrão de codificação e nomenclatura dos objetos do banco de dados. Esse processo facilita a leitura do código e futuras manutenções.

Dicas:

- Os tipos de dados string (texto) e data são delimitados por aspas simples.

- Ex: ‘João’

- Ex: ‘2019-01-01’

- Comentários no SQL Server

- Linha: --

- Bloco: /\*\*/

- Dependendo do idioma que o SQL Server foi configurado a data pode vir no formato ANO-MÊS-DIA ou DIA-MÊS-ANO.

- Virgula na linguagem SQL é separador de campos.

- Ex.: Campo1, campo2, campo3, ....

- Para dados números utilizar o ponto para separar os valores.

- Ex.: 149.99

- No comando SELECT o asterisco (\*) significa que todos os campos da tabela serão exibidos.

***Comandos – DDL***



- **CREATE**

- Sintaxe:

- CREATE OBJETO NOME\_OBJETO;

- Criando banco de dados

- CREATE DATABASE BD\_DDL;

- **USE**

- Carregando o banco de dados no contexto da janela de consulta do SQL Server

Sintaxe:

- USE BD\_DDL

-Criando uma Tabela

CREATE TABLE TB\_DDL(MATRICULA INT PRIMARY KEY, NOME VARCHAR(100), SEXO CHAR(1))

**-ALTER**

Sintaxe:

- ALTER OBJETO NOME\_OBJETO;

Alterando a tabela, adicionando um novo campo

Sintaxe:

- ALTER TABLE TB\_DDL ADD DTNASC DATETIME, CPF CHAR(11)

Alterando a tabela, alterando o tipo de dados campo SEXO de CHAR(1) para VACHAR (15)

Sintaxe:

- ALTER TABLE TB\_DDL

- ALTER COLUMN DTNASC DATE

Alterando a tabela, excluindo o campo SEXO.

Sintaxe:

- ALTER TABLE TB\_DDL DROP COLUMN SEXO;

**- DROP**

Sintaxe

- DROP OBJETO NOME\_OBJETO

Apagando tabela do banco de dados

Sintaxe

- DROP TABLE TB\_DDL

Obs.: Se existir ligação de chave estrangeira não será possível deletar a tabela.

Apagando Objetos do banco de dados

- DROP VIEW VW\_TESTE

- DROP FUNCTION FN\_TESTE

- DROP PROCEDURE USP\_TESTE

- DROP TRIGGER TR\_TESTE

**- TRUNCATE**

Sintaxe

- TRUNCATE TABLE NOMETABELA;

Apagando todos os registros da tabela

- TRUNCATE TABLE TB\_DDL

***Comandos – DML***



**- INSERT**

Sintaxe:

Maneira 1

- INSERT INTO TABELA (CAMPOS) VALUES (VALORES);

Maneira 2

- INSERT INTO NOMETABELA VALUES (VALORES,VALORES)

Obs.: Quando não se especifica os campos na hora do insert, deve-se assumir a ordem de criação dos mesmo na tabela.

Inserindo um registro especificando os campos

- INSERT INTO TB\_TML ( MATRICULA, NOME, DTNASC, SEXO) VALUES (1, ‘PESSOA\_A’, ‘2000-12-10’, ‘M’)

Utilizando o mesmo insert para inserir vários registros

- INSERT INTO TB\_DML (MATRICULA, NOME, DTNASC, SEXO) VALUES (2, ‘PESSOA\_B’, ‘2000-12-10’, ‘M’), (3, ‘PESSOA\_B’, ‘2000-01-01’, ‘F’)

Inserindo um registro na tabela sem especificar os campos. Lembrar da ordem de inserção das colunas no momento da criação da tabela.

Inserindo um registro nas tabelas sem especificar os campos. Lembrar da ordem de inserção das colunas no momento da criação das tabelas.

- INSERT TB\_DML VALUES (4, ‘PESSOA\_D’, ‘2000-01-10’, ‘M’)

Inserindo dados em uma tabela que foi resultado de n x n.

- INSERT INTO LISTA\_COMPRAS (COD\_PRODUTO, CODNF, QTD) VALUES(‘PRD0001’, ‘NF000001’, 3)

**UPDATE**

Sintaxe:

- UPDATE TABELA SET CAMPO = NOVO\_VALOR WHERE CONDICAO = VALOR

Obs.: Sempre utilizar filtros para utilizar este comando

Atualizando um registro de um campo de uma tabela funcionário

Sintaxe:

- UPDATE TB\_DML SET SEXO = ‘F’ WHERE NOME = ‘PESSOA\_B’

Atualizando dois campos de um registro de uma tabela

Sintaxe:

- UPDATE TB\_DML SET SEXO = ‘F’, DATNASC = ‘1997-07-01’ WHERE NOME = ‘PESSOA\_D’

**DELETE**

Sintaxe:

- DELETE FROM TABELA CONDICAO = VALOR

Obs.: Sempre utilizar filtros para utilizar este comando

Apagando registros da tabela funcionário

Sintaxe:

- DELETE FROM DB\_DML WHERE NOME = ‘PESSOA\_C’

***Comando – DQL***



**SELECT**

Sintaxe

- SELECT \* FROM TABELA ou VIEW ou FUNCTION

Selecionado todos os dados da tabela

Sintaxe

- SELECT \* FROM TB\_DML

Selecionando apenas alguns campos da tabela

Sintaxe

- SELECT NOME, DATNASC, FROM TB\_DML WHERE SEXO = ‘F’

Selecionando todos os campos da tabela filtrando por várias informações

Sintaxe:

- SELECT \* FROM TB\_DML WHERE DATNASC > ‘1999-10-25’ AND SEXO = ‘F’

***Comando – DTL***

**BEGIN TRANSACTION / COMMIT / ROLLBACK**

Sintaxe:

- BEGIN TRAN / BEGIN TRANSACTION

COMANDO SQL

COMMIT OU ROLLBACK

Abrindo uma transação no banco de dados para utilização do comando UPDATE

Sintaxe:

- BEGIN TRAN

UPDATE TB\_DML SWET SEXO = ‘M’ WHERE NOME = ‘PESSOA\_D’

COMMIT : Salvará a alteração na base de dados definitivamente e fechará a transação.

ROLLBACK : Irá desfazer a alteração realizada e fechará a transação.

***Comando – DCL***



**GRANT / REVOKE / DENY**

Sintaxe:

- GRANT / REVOKE / DENY <Privilegio>

ON OBJETO\_BD TO <Usuario>.

Dando permissão para o usuário executar o comando SELECT na tabela.

Sintaxe:

- CREATE USER USER\_DCL WITHOUT LOGIN;

-GRANT SELECT, INSERT ON DBO.TB\_DML TO USER\_DCL

Desfazendo permissão dada do comando GRANT

Sintaxe:

- REVOKE INSERT ON DBO.TB\_DML TO USER\_DCL

Negando permissão do usuário dar SELECT na tabela

Sintaxe:

- DENY SELECT ON DBO.TB\_DML TO USER\_DCL

Executando consulta com usuário criado

Sintaxe:

- EXECUTE AS USER = ‘USER\_DCL’ SELECT \* FROM TB\_DML REVERT;

Lista de Privilégios

SELECT – Capacidade de executar instruções SELECT na tabela.

INSERT – Capacidade de executar instruções INSERT na tabela.

UPDATE – Capacidade de executar instruções UPDATE na tabela.

DELETE – Capacidade de executar instruções DELETE na tabela.

REFERENCES – Capacidade de criar uma restrição que se refere à tabela.

ALTER – Capacidade de executar instruções ALTER TABLE para alterar a definição da tabela.

ALL – ALL não concede todas as permissões para a tabela. Em vez disso, concede as permissões ANSI-92 que são SELECT, INSERT, UPDATE, DELETE, e REFERENCES.

**CONTRAINTS**

**IDENTITY – EXTRA**

- Faz com o atributo selecionado seja incrementado automaticamente, ou seja, irá gerar uma numeração automática sem que seja necessário gerar o valor no insert.

- É indicado ser utilizado com campos do tipo inteiro.

- No identity você pode informar qual o inicio da operação e quando será o incremento. Caso não seja informado implicitamente o SQL SERVER aplica o valor (1,1). Iniciando o número 1 e incrementando 1.

- Criando uma tabela com o campo incremental.

Sintaxe:

CREATE TABLE tb\_identity( cod INT IDENTITY PRIMARY KEY, NOME VACHAR(50))

- Criando uma tabela começando a numeração a partir do número 1000 e incrementando de 1 em 1.

Sintaxe:

CREATE TABLE tb\_identity( cod INT IDENTITY (1000,1), NOME VACHAR(50))

- Habilitando e desligando uma Identity

Sintaxe:

SET IDENTITY\_INSERT tb\_identity ON – Desliga a identidade

SET IDENTITY\_INSERT tb\_identity OFF – Habilita a identidade

**PRIMARY KEY – PK**

- Restrição que identifica o atributo com identificador da tabela.

- Quando uma tabela possui mais de uma chave primária , é chamada de chave composta.

- Ao criar uma tabela sempre definir um campo com chave primária.

- Criando chave primária de uma tabela com cosntraint

Sintaxe:

- CREATE TABLE tb\_pk ( cod INT CONSTRAINT pk\_teste PRIMARY KEY, NOME Vachar(50))

-Criando uma chave primária no final do comando. Deve-se referenciar os campos nos parênteses.

Sintaxe:

- CREATE TABLE tb\_pk ( Cod INT, Nome VACHAR(50), PRIMARY KEY (cod)).

- Criando tabela com duas chaves primárias

Sintaxe:

- CREATE TABLE tb\_pk ( Cod INT, Cod2 INT, Nome VACHAR(50), PRIMARY KEY (cod1,cod2))

- Adicionando chave primária, após criação de uma tabela sem pk.

Sintaxe:

- ALTER TABLE tb\_pk ADD PRIMARY KEY (cod);

- Adicionando chave primária , após a criação de uma tabela sem pk, informando o nome da CONSTRAINT.

Sintaxe:

- ALTER TABLE tb \_pk ADD CONSTRAINT pk\_test PRIMARY KEY (cod)

- Apagando uma CONSTRAINT PK

Sintaxe:

- ALTER TABLE tb\_pk DROP CONSTRAINT pk\_teste;

**Foreing Key – Fk**

Restrição que define que um atributo A faz referência ao atributo B que é chave de outra tabela.

A Chave Estrangeira é definida no momento da normalização.

O atributo deve ser do mesmo tipo do campo referência de sua tabela de origem.

O atributo (FK) deve ser referenciado a qual atributo da tabela de origem ele faz correlação.

Para a chave estrangeira funcionar, a tabela que contém o atributo chave deve estar criada.

Para um registro ser referenciado na tabela com a FK, o registro tem que existir na tabela de origem.

- Criando chave estrangeira na criação da tabela com constraint.

Sintaxe:

- CREATE TABLE tb\_fk\_marca ( cod INT PRIMARY KEY IDENTITY, Nome VARCHAR(50) )

- CREATE TABLE tb\_fk\_carro( Placa varchar (6) PRIMARY KEY, modelo VARCHAR(50), fk\_marca INT FOREING KEY REFERENCES test\_fk\_marca(Cod) )

- Adicionado uma chave estrangeira após criação de tabela. Supondo que o campo não foi referenciado com fk.

Sintaxe:

- ALTER TABLE tb\_fk\_carro ADD FOREING KEY (fk\_marca) REFERENCES tb\_fk\_marca (COD);

- Adicionando uma chave estrangeira com constraint após criação de tabela. Supondo que o campo não foi referenciado como fk.

Sintaxe:

- ALTER TABLE tb\_fk\_carro ADD CONSTRAINT fk\_marca\_carro FOREING KEY (fk\_marca) REFERENCES tb\_fk\_marca (COD);

- Apagando constraint

Sintaxe:

- ALTER TABLE tb\_fk\_carro DROP CONSTRAINT fk\_marca\_carro

**UNIQUE**

Restrição que garante que o valor de um determinado campo não se repita dentro de uma tabela, ou seja, os valores são únicos.

A restrição PRIMARY KEY possui uma restrição UNIQUE implícita.

Em uma tabela pode haver várias restrições Unique, porém somente uma restrição chave de PK por tabela.

- Criando tabela com a restrição UNIQUE

Sintaxe:

- CREATE TABLE tb\_unique( cod INT PRIMARY KEY IDENTITY, Nome VARCHAR (50), CPF CHAR(11) CONSTRANIT UQ\_CPF UNIQUE)

- Adicionando unique em um campo já criado

Sintaxe:

- ALTER TABLE tb\_unique ADD UNIQUE (cpf);

- Adicionando unique em um campo já criado especificando o nome da cosntraint.

Sintaxe:

- ALTER TABLE tb\_unique ADD CONSTRAINT UQ\_CPF UNIQUE (cpf);

- Apagando uma constraint

- ALTER TABLE tb\_unique DROP CONSTRAINT UQ\_CPF;

**NOT NULL**

Not Null é a restrição que impede a coluna de receber valores nulos no momento da inserção ou atualização.

O SQL Server coloca como padrão NULL quando não é informado se o campo não pode receber valores nulos.

Boa prática: Sempre explicitar se o campo é NULL ou NOT NULL na criação da tabela.

- Criando tabela com campos são nulos e campos nulos

Sintaxe:

- CREATE TABLE tb\_notnull( Cod INT PRIMARY KEY IDENTITY, Nome VARCHAR (50) NOT NULL, DT DATE NULL )

- Colocando not null em um campo já criado

Sintaxe:

- ALTER TABLE tb\_notnull ALTER COLUMN DT DATE NOT NULL

- Removendo o not null em um campo já criado

Sintaxe:

- ALTER TABLE tb\_not null ALTER COLUMN DT DATE NULL

**DEFAULT**

Restrição que coloca um valor como padrão em uma determinada coluna.

Ao realizar um Insert E o valor de um determinado campo não for informado, a restrição Default ira preencher o campo o valor padrão definido pela Constraint

- Criando tabela com a restrição default

Sintaxe

- CREATE TABLE tb\_default(Cod INT PRIMARY KEY IDENTITY, Nome VARCHAR(50), dtInsert DATE CONSTRAINT DF\_DATA DEFAULT (GETDATE())

- Adicionando default em uma coluna já criada

Sintaxe

- ALTER TABLE tb\_default ADD CONSTRAINT df\_dt\_insert DEFAULT getdate() FOR dtInsert

- Apagando constraint

Sintaxe

- ALTER TABLE tb\_default ALTER COLUMN dtInsert DROP DEFAULT

**CHECK**

A restrição CHECK é usada para limitar os valores que pode ser inseridos em uma coluna.

Se você definir uma restrição CHECK em uma única coluna, ela permitirá apenas inserir determinados valores que atendam a lógica de checagem.

- Criando tabela com o CHECK

Sintaxe

- CREATE TABLE tb\_check ( Cod INT PRIMARY KEY IDENTITY, NOME VARCHAR(50) NOT NULL, VALOR NUMERIC (6,2) CHECK (VALOR >0), LETRA CHAR(1) CHECK ( LETRA IN (‘A’,’B’,’C’))

)

- Adicionando check a um campo já criado

Sintaxe

- ALTER TABLE tb\_check ADD CHECK (valor>0)

- Adicionando check atribuindo nome a constraint

Sintaxe

- ALTER TABLE tb\_check ADD CONSTRAINT CHEK\_VALOR(valor>0)

- Apagando uma constraint check

Sintaxe

- ALTER TABLE tb\_check DROP CONSTRAINT CHECK\_VALOR

**Desenvolvendo Consultas no SQL**

**Comando SELECT**

O comando SELECT possui uma estrutura e é executado em uma ordem lógica para retornar as informações.

Cláusulas

- SELECT – Seleciona a lista de resultados

- FROM – Aonde vou buscar meus dados

- WHERE – O que eu quero filtrar dos dados

- GROUP BY – Agrupa os dados

- HAVING – Filtra o valor do agrupamento

- ORDEN BY – Ordena o resultado

- Como escreve uma consulta simples

Sintaxe

- SELECT <\* / campos\_da\_tabela> FROM <tabela/view/functions\_tablevalue> WHERE <Filtros> ORDER BY <campos> ASC/DESC

- Uma consulta é executada logicamente na ordem como é mostrada abaixo.

1: FROM < table souce>

2: WHERE < search condition>

3: GROUP BY <group by list>

4: HAVING < search condition>

5: SELECT <select list>

6: ORDER BY <order by list>

- ORDER BY : Maneira em que os dados são ordenados após a realização de uma consulta a uma tabela. Quando não for informado como se quer ordenar, o SQL coloca como padrão o ASC.

- ACS – Ascendente – Crescente – (menor para o maior)

Sintaxe

- SELECT \* FROM TB\_FUNC ORDER BY NOME

Ex.: 2

- SELECT \* FROM TB\_FUNC ORDER BY NOME,SALARIO

- DESC – Descendente – Decrescente – (maior para o menor)

Sintaxe

- SELECT \* FROM TB\_FUNC ORDER BY NOME DESC

Ex.:2

- SELECT \* FROM TB\_FUNC ORDER BY NOME ASC, SALARIO DESC

**Operadores Lógicos**

São utilizados quando queremos trabalhar com o relacionamento de duas ou mais condições ao mesmo tempo.

Operadores lógicos

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Operador - Descrição**

AND - E lógico

OR - OU lógico

NOT - Negação

----------------------------------------

- **AND:** E lógico. Avalia as condições e devolve um valor verdadeiro caso ambos sejam corretos. Ambos os valores devem ser verdadeiro para que o operador AND traga o resultado.

Sintaxe

- SELECT MATRICULA, NOME FROM TB\_FUN WHERE DATAEMISSAO IS NULL NA STATU\_FUNC = ‘ATIVO’

- **OUR:** OU lógico. Avalia as condições e devolve um valor verdadeiro se algum for correto.

- Uma das condições pode ser verdadeira ou não.

Sintaxe

- SELECT MATRICULA, NOME FROM TB\_FUNC WHERE DATADEMISSAO IS NULL OR STATU\_FUNC = ‘ATIVO’

- **NOT:** Negação Lógica. Devolve o valor contrario de expressão, ou seja, a negação do filtro. O NOT inverte o resultado lógico de uma condição, ou seja se a condição for verdadeira o resultado é falso, e se a condição for falsa o resultado é verdadeiro.

- Negando a condição do filtro utilizado

Sintaxe

- SELECT MATRICULA, NOME, SALARIO FROM TB\_FUNC WHERE NOT STATUS\_FUNC = ‘ATIVO’

**Operadores Aritméticos**

São responsáveis pelas operações e pelos cálculos matemáticos simples obtidos com T-SQL.

Tabela de operadores aritméticos

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Operador - Descrição

Adição - +

Subtração - -

Multiplicação - \*

Divisão - /

Resto da divisão - %

--------------------------------------

- Utilizando operadores para realizar operações com os campos da tabela

Sintaxe

- SELECT NOME, SALARIO + 50 AS ADICAO, SALARIO – AS SUBTRAÇÃO, SALARIO \* 1.2 AS AJUSTE\_20\_PCENTO, SALARIO / 2 AS DIVISAO FROM TB\_FUNC WHERE STATUS\_FUNC = ‘ATIVO’

**Operadores Relacionais**

São responsáveis pelo estabelecimento de ações de comparação, são utilizadas para a definição de condições em que há necessidade de comparação entre dois ou mais valores.

Tabela de operadores Relacionais

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Operador - Descrição

> Maior que

< Menor que

= Igual a

>= Maior ou igual a

<= Menor ou igual a

<> ou != Diferente de

------------------------------------------------

Ex.:

Sintaxe

- SELECT \* FROM TB\_FUNC WHERE STATUS\_FUNC = ‘Ativo’

- SELECT \* FROM TB\_FUNC WHERE DATANASCIMENTO >= ‘1999-01-01’

- SELECT \* FROM TB\_FUNC WHERE SALARIO >= 2000 AND SALARIO <= 4000

- SELECT \* FROM TB\_FUNC WHERE STATUS\_FUNC <> ‘Ativo’

**Operadores Auxiliares**

São operadores que facilitam a definição de condições a serem utilizados com o comando DQL

Tabela de operadores auxiliares

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Operador - Descrição

IS NULL Verifica se o campo é Nulo

IS NOT NULL Verifica se o campo não está nulo

BETWEEN Verifica se o valor esta numa faixa de valores

IN Verifica se i valor existe em uma lista de valores

LIKE Verifica um valor buscando seus semelhantes

**IS NULL e IS NOT NULL**

IS NULL: Verifica se o campo em questão tem valores nulos.

ISD NOT NULL: verifica se o campo em questão não possui valores nulos.

Vazio diferente de nulo. O campo vazio tem informação enquanto o nulo não.

Um campo tem o valo nulo em um SGBD quando em uma determinada coluna é exibida o valor NULL.

Ex.:

Sintaxe

- SELECT \* FROM TB\_FUNC WHERE DATAEMISSAO IS NULL

- SELECT \* FROM TB\_FUNC WHERE DATAEMISSAO IS NOT NULL

**BETWEEN**

Utilizado para especificar um intervalo de valores.

Ex.:

Sintaxe

- SELECT \* FROM TB\_FUNC WHERE DATAADMISSAO BETWEEN ‘2012-01-01’ AND ‘2018-12-20’

- SELECT \* FROM WHERE DATAADMISSAO NOT BETWEEN ‘2017-01-01’ AND ‘2017-12-20’

- SELECT \* FROM TB\_FUNC WHERE CODDEPTO BETWEEN 5 NA 15

**IN**

Utilizado para verificar se o valor procurado está dentro de uma lista.

Ex.:

Sintaxe

- SELECT \* FROM TB\_FUNC WHERE STATUS\_FUNC IN(‘ATIVO’,’FÉRIAS’)

- SELECT \* FROM TB\_FUNC WHERE MONTH(DATAADMISSAO) IN (1,2)

- SELECT \* FROM TB\_FUNC WHERE MONTH(DATAADMISSAO) NOT IN(10,11)

**LIKE**

Utilizado para fazer busca em campos com valores texto.

Podemos utilizar a porcentagem para ajudar na busca. É considerado um caractere coringa no LIKE.

Palavra + % - trás todos registros que começam com o nome da palavra passada

% + - trás todos os registros que terminam com a palavra passada.

% + palavra +% - trás todos os registros que contenham a palavra no inicio, meio ou final.

EX.:

Sintaxe

- SELECT \* FROM TB\_FUNC WHERE NOME LIKE’JOSE’

- SELECT \* FROM TB\_FUNC WHERE NOME LIKE ‘JOSE%’

- SELECT \* FROM TB\_FUNC WHERE NOME ‘%JOSE’

- SELECT \* FROM TB\_FUNC WHERE NOME LIKE ‘%JOSE%’

**Extra – Clausulas que podem ser utilizadas em um comando SELECT**

Clausulas Extra

- ALIAS (Apelidos para tabelas e campos)

- CASE

- DISTINCT

- TOP

**ALIAS =** São apelidos utilizados para renomear tabelas e colunas a nível de consulta, ou seja, somente no momento do SELECT. Os apelidos ficam ativos durante a consulta assumindo o valor do campo ou tabela.

Caso for utilizar ALIAS de tabela, o mesmo deve ser utilizado durante toda a construção da consulta.

\* Facilitam a leitura da Query quando utilizamos JOINS.

Em casos de ALIAS de coluna não se torna obrigatório, pois são utilizados para exibir um nome amigável para a coluna em questão.

É possível realizar ALIAS de coluna de três maneiras diferentes

Sintaxe :

- Coluna AS Apelido

NOME AS ‘NOME FUNCIONARIO’

- Apelido = Coluna

‘NOME AS ‘NOME FUNCIONARIO’

- Coluna Apelido

NOME ‘NOME FUNCIONARIO’

Utilizado alias de coluna e tabela numa consulta.

Sintaxe:

SELECT F.NOME AS ‘NOME FUNCIONARIO’,’DATA DE ADMISSÃO’ = F.DATAADMISSAO,F.SALARIO ‘SALÁRIO FUNCIONÁRIO FROM TB\_FUNC AS F

**CASE =** Utilizado para fazer uma analise lógica do valor da coluna e retornar caso seja verdadeiro ou falso.

Sintaxe:

- SELECT NOME, STATUS\_FUNC, CASE STATUS\_FUNC

WHEN ‘ATIVO’ THEN ‘A’

WHEN ‘INATIVO’ THEN ‘I’

WHEN ‘FÉRIAS’ THEN ‘F’

ELSE

‘O’

END SIGLA STATUS

FROM TB\_FUNC

- SELECT NOME, SALARIO, CASE

WHEN SALARIO > 1200 AND SALARIO < 1500 THEN SALARIO \* 1.1

WHEN SALARIO > 1501 AND SALARIO < 2000 THEN SALARIO \* 1.2

WHEN SALARIO < 1000 THEN SALARIO END SALARIO\_COM\_REAJUSTE

FROM TB\_FUNC

**DISTINCT:** Elimina linhas duplicadas de um determinado campo, retornando apenas 1 registro daqueles dados.

Ao adicionar outro campo a consulta o DISTINCT ira somente remover as duplicatas caso os dois campos possuam valores iguais.

Sintaxe:

- SELECT DISTINCT NOME FROM TB\_FUNC

- SELECT DISTINCT NOME, SALARIO FROM TB\_FUNC

- SELECT DISTINCT NOME, SALARIO, DATANASCIMENTO FROM TB\_FUNC

TOP: Objetivo permitir limitar o número de registros serão mostrados pela instrução SELECT.

Obs.: A cláusula ORDER BY irá influenciar diretamente no retorno dos registros quando é utilizada a cláusula TOP.

O comando recebe como parâmetro o numero de registros a serem mostrados na query.

Sintaxe:

- SELECT

TOP 5 MATRICULA, NOME FROM TB\_FUNC

- SELECT

TOP 5 MATRICULA, NOME FROM TB\_FUNC

ORDER BY MATRICULA DESC

***JOINS***

As cláusulas JOIN são usadas para realizar a combinação das tabelas do banco de dados. Toda essa combinação é baseada no processo de modelagem do negócio. A ligação das tabelas é geralmente feita através dos campos de chave primária e estrangeira.

Os JOIN possuem uma clausula chamada ON, que é responsável iniciar a combinação dos campos utilizados na junção.

Temos os seguintes tipos de Joins

- INNER JOINN

- LEFT OUTER JOIN

- RIGHT OUTER JOIN

- FULL OUTER JOIN

- CROSS JOIN

- SELF JOIN

Forma de utilizar as junções no SQL

Sintaxe

\*ANSI 89 – SELECT <CAMPOS> FROM A,B WHERE A.Coluna = B.Coluna\*

ANSI 92 – SELECT <CAMPOS> FROM A INNER JOIN B ON A.Coluna = B.Coluna

**INNER JOIN =** Retorna linhas quando há uma correspondência em ambas as tabelas. Somente exiba o que é correspondente entre A e B

Sintaxe

SELECT F.NOME, D.DESCRICAO FROM FUNCIONARIO as F

INNER JOIN DEPARTAMENTO as D ON F.CODDEPTO = D.CODIGO

**LEFT JOIN ou LEFT OUTER JOIN =** Retorna todas as linhas da tabela à esquerda (tabela de cima), mesmo se não houver nenhuma correspondência na tabela Á direita. Exiba todas as linhas de A que tenham ou não correspondente com B.

Sintaxe

SELECT F.NOME, D.DESCRICAO FROM FUNCIONARIO as F

LEFT JOIN DEPARTAMENTO as D ON F.CODDEPTO = D.CODIGO

Exibir somente registros de A que não possuem correspondência com B.

SELECT F.NOME, D.DESCRICAO FROM FUNCIONARIO as F

LEFT JOIN DEPARTAMENTO as D ON F.CODDEPTO = D.CODIGO

WHERE D.CODIGO IS NULL

**RIGHT JOIN ou RIGHT OUTER JOIN=** Retorna todas as linhas da tabela da direita (tabela de baixo), mesmo que não haja correspondências na tabela à esquerda. Exiba todas as linhas de B que tenham ou não correspondente com A.

Sintaxe

SELECT F.NOME, D.DESCRICAO FROM FUNCIONARIO as F

RIGHT JOIN DAPARTAMENTO as D ON F.CODDEPTO = D.CODIGO

Sintaxe

- Exibir somente registros de B que não possuem correspondência com A

SELECT F.NOME, D.DESCRICAO FROM FUNCIONARIO as F

RIGHT JOIN DAPARTAMENTO as D ON F.CODDEPTO = D.CODIGO

WHERE F.CODEPTO IS NULL

**FULL JOIN ou FULL OUTER JOIN =** Combina os resultados de esquerda e direita. A tabela conterá todos os registros de ambas as tabelas. Caso uma linha de dados não esteja associada a qualquer linha da outra tabela, os valores das colunas a lista de seleção serão nulos.

Sintaxe

SELECT F.NOME, D.DESCRICAO FROM FUNCIONARIO as F

FULL JOIN DAPARTAMENTO as D ON F.CODDEPTO = D.CODIGO

- Exibir registros de A e de B que não possuem correspondência de chave.

Sintaxe

SELECT F.NOME, D.DESCRICAO FROM FUNCIONARIO as F

RIGHT JOIN DAPARTAMENTO as D ON F.CODDEPTO = D.CODIGO

WHERE F.CODEPTO IS NULL OR D.CODIGO IS NULL

**CROSS JOIN =** Combina dos dados das duas tabelas formando um produto cartesiano. Este tipo de JOIN não utiliza a cláusula ON. Ele faz uma multiplicação dos registros de uma tabela com outra.

Sintaxe

SELECT F.NOME, D.DESCRICAO FROM FUNCIONARIO as F

CROSS JOIN DAPARTAMENTO as D

SELF JOIN = É um conceito de join e não um comando propriamente dito. Ele é caracterizado na modelagem como auto relacionamento, ou seja, a tabela se relacionado com ela mesma. Pode-se utilizar qualquer tipo de JOIN.

Sintaxe

SELECT F. NOME AS FUNCIONARIO, S.NOME AS SUPERVISOR

FROM FUNCIONARIO F

INNER JOIN FUNCIONARIO S ON F.CODSUPERVISOR = S.MATRICULA

**SELF JOIN:** É um conceito de join e não um comando propriamente dito. Ele é caracterizado na modelagem como auto relacionamento, ou seja, a tabela se relacionando com ela mesma. Pode-se utilizar qualquer tipo de JOIN.

Sintaxe

SELECT F.NOME AS FUNCIONARIO, S.NOME AS SUPERVISOR

FROM FUNCIONARIO F

INNER JOIN FUNCIONARIO S ON F.CODSUPERVISOR = S.MATRICULA

***UNION***

UNION / UNION ALL: Cria um único conjunto de resultados baseado na reunião de varias consultas.

\*Requisitos para utilização do Union

- As duas consultas devem conter o mesmo número de colunas;

- Os tipos de dados devem responder para cada coluna;

- A clausula ORDER BY deve-se ser utilizada apenas no ultimo SELECT a ser feito;

- A primeira consulta é o cabeçalho do resultado;

\*Principal diferença entre o UNION e o UNIN ALL

- UNION: Remove qualquer linha duplicada no conjunto de resultados. Ele possui um DISTINCT implícito;

- UNION ALL: Inclui os registros duplicados

Sintaxe:

- SELECT coluna1, coluna2, FROM TABLEA1 UNION / UNION ALL

SELECT coluna1, coluna2 FROM TABELA2

EXCEPT

**EXCEPT:** Realiza a diferença dos conjuntos A,B (A – B), ou seja, o conjunto de elementos que pertencem a A e não pertencem a B.

O EXCEPT opera nos conjuntos de resultados de duas consultas de entrada e retorna as linhas que aparecem na primeira entrada, mas não na segunda, ou seja, quando se quer que apareçam os registros da tabela A que não sejam iguais aos da tabela B.

O EXCEPT utiliza a mesma regra do UNION referente numero de campos, cabeçalho de tipo de dado.

Sintaxe:

- SELECT coluna1, coluna2, FROM TABLEA1

EXCEPT

-SELECT coluna1, coluna2 FROM TABELA2

**INTERSECT**

INTERSECT: Retorna apenas linhas que são comuns a ambos os conjuntos. Em outras palavras, se uma linha aparecer pelo menos uma vez no primeiro conjunto e pelo menos uma vez no segundo conjunto, ela aparecerá uma vez no resultado do operador INTERSECT;

O INTERSECT assemelha-se ao INNER JOIN.

O INTERSECT utiliza a mesma regra do UNION referente numero de campos, cabeçalho e tipo de dado.

- SELECT coluna1, coluna2, FROM TABLEA1

INTERSECT

-SELECT coluna1, coluna2 FROM TABELA2

**VARIÁVEIS**

Para declarar variáveis no SQL-SERVER deve-se seguir alguns itens

- Primeiro, utilizar a palavra reservada DECLARE

- Segundo, nome da variável é composta de @ + nome = @nome

- Terceiro, tipo de variável (INT,CHAR,DATE)

- Quarto, valor que a variável vai receber, deve ser declarado com a palavra reservada SET @nome = valor,

Sintaxe da declaração de uma variável

DECLARE @nome VARCHAR(20)

SET @NOME = ‘CURSO MTA DF’

Exibindo o valor de uma variável

SELECT @NOME

OBS: São importantes para programa procedures, Trigres, Indices

***Funções de Conversões***

Funções de conversão são utilizadas quando precisamos transformar um tipo de dados em outro.

\*Conversão Implícita: Quando um determinado tipo de dado é convertido sem a utilização das funções de conversões.

-Utiliza a precedência dos tipos de dados

A precedência do tipo de dados determina qual tipo de dados será escolhido quando as expressões de tipos de dados diferentes forem combinadas.

Por padrão, os tipos de dados com precedências mais baixa é convertido no tipo de dados com a precedência mais alta. Isso é importante para entendermos as conversões implícitas.

\*Conversão Explicita: É necessário a utilização das funções CAST e CONVERT para alterar o tipo de dado.

**Data e Hora do Sistema**

GETDATE(): Data e hora atuais do sistema. Função da engine do SQL SERVER

CURRENT\_TIMESTAMP: Data e hora atuais do sistema. Padrão SQL ANSI.

**Conversão de Dados**

CAST: Função de conversão padrão ANSI

TRY\_CAST: Realiza o tratamento dos dados retornando o valor NULL quando não conseguem converter o valor para o tipo desejado. A partir da versão 2012 do SQL SERVER.

CAST: É mais performático do que a função CONVERT.

Ambas recebem dois parâmetros;

- Primeiro Parâmetro: Campo a ser convertido

- Segundo Parâmetro: Tipo de dado para qual será convertido o campo.

Sintaxe:

CAST(campo AS novo\_tipo)

TRY\_CAST(campo AS novo\_tipo)

CONVERT : Função de conversão da engine do SQL Server

TRY\_CONVERT: Realiza o tratamento dos dados retornando o valor NULL quando não conseguem converter o valor para o tipo desejado . A partir da versão 2012 do SQL Server.

Convert é essencial para converter data para String

Ambas podem receber 2 ou 3 parâmetros

- Primeiro parâmetro: tipo de dados para qual será convertido o campo

- Segundo parâmetro: Campo que será convertido

- Terceiro parâmetro: Opcional, utilizado para formatação de campos de data

Sintaxe;

CONVERT(TIPO\_DE\_DADO, parâmetro2, parâmetro3(opcional))

TRY\_CONVERT(TIPO\_DE\_DADO, parâmetro2, parâmetro3(opcional))

***Funções de Texto***

**SUBSTRING:** Extrai da parte de um texto existente.

- Esta função utiliza 3 parâmetrox

1º Texto que será extraído

2º Posição inicial do texto

3º posição final no texto

Sintaxe

SUBSTRING (texto(ou campo), parâmetro 1, parâmetro 2 )

DECLARE @TXT VARCHAR (10) = ‘ABCDE’

SELECT SUBSTRING (@TXT,1,3 ) AS TESTE\_SUBSTRING

LEFT e RIGHT: Retorna os caracteres solicitados da esquerda ou da direita do parâmetro de entrada.

Estas funções recebem 2 parâmetros:

- Primeiro parâmetro: texto ( ou campo)

- Segundo parâmetro: número de palavras a serem extraídas do texto.

Sintaxe:

LEFT (texto(oucampo), parâmetro 2)

RIGHT (texto(oucampo), parâmetro2)

DECLARE @TXT VACHAR(10) = ‘ABCDE’

SELECT LEFT (@TXT,2) TESTE\_LEFT

SELECT RIGHT (@TXT,2) TEST\_RIGHT

**LEN**

Retorna o número de caracteres de um texto passado como parâmetro de entrada.

O único parâmetro desta função é o texto a ser contado

Sintaxe

LEN(texto(campo))

DECLARE @TEXT VARCHAR(10) = ‘ABCDE’

SELECT LEN (@TXT) AS TEST\_LEN

**CHARINDEX**

Retorna a posição da primeira ocorrência de uma palavra ( ou parte da palavra) de uma texto completo.

Recebe 2 Parâmetros:

Primeiro parâmetro: texto o que queremos encontrar

Segundo parâmetro: palavra ou texto que será realizada a pesquisa da item do parâmetro 1.

Sintaxe:

CHARINDEX(parametro1, texto(ou campo))

DECLARE @TXT VARCHAR(20) = ‘CONSULTA BD’

SELECT CHARINDEX(‘’, @TXT) TEST\_CHARINDEX

**REPLACE**

Substitui todas as ocorrências de uma palavra com outra.

Recebe 3 parâmetros:

Primeiro parâmetro: texto

Segundo parâmetro: palavra ou caractere que será alterado na palavra

Terceiro parâmetro: palavra, texto ou informação que substituirá a palavra informado no segundo parâmetro.

Sintaxe:

REPLACE(texto(oucampo), parametro2, parametro3)

DECLARE @TXT VARCHAR(50)

SET @TXT = ‘APELIDO: PADAWAN’

SELECT REPLACE(@TXT, ‘:’,’>>>’) TESTE\_REPLACE

**UPPER e LOWER**

Retorna as palavras com todos os caracteres maiúsculos ou minúsculos.

UPPER: Maiúsculo

LOWER: Minúsculo

Recebe como parâmetro somente o texto

Sintaxe

UPPER (texto(ou campo)

LOWER (texto(ou campo)

DECLARE @TXT VARCHAR(10) = ‘t-SQL’

SELECT UPPER(@TXT) TEST\_UPPER

SELECT LOWER(@TXT) TEST\_LOWER

**RTRIM e LTRIM**

Remove os espaços da esquerda ou da direita da palavra passada com parâmetro.

TRIM: Remove os espaços em banco de dados dos dois lados, mas está função só esta disponível a partir da versão 2017 do SQL Server.

Sintaxe:

TRIM, RTRIM ou LTRIM (texto(ou campo))

**CONCAT**

Retorna uma cadeia de caracteres que é o resultado da concatenação de dois ou mais valores. A partir da versão 2012 do SQL Server.

Substitui o uso do “+” no momento de concatenar os campos numa consulta.

Também reduz a utilização do Cast ou Convert, pois ele convert os valores em string automaticamente.

Sintaxe:

CONCAT(‘valor1’, ‘valor2’,valor3)

***Funções para tratamento de não nulos***

**ISNULL**

Verifica se primeiro parâmetro é nulo e substitui pelo valor utilizado no segundo parâmetro.

**NULLIF**

Retorna nulo se os dois parâmetros passados foram iguais, caso sejam diferentes, ele retorna o valor primeiro parâmetro.

**COALESCE**

É uma função do SQL ANSI, que retorna o primeiro valor Não nulo em uma lista de valores.

***Funções Agregadas***

\*Usa-se essa funções para uma consulta no BD, então sempre usar o comando SELECT;

\*Uso em Cálculos;

MIN = Valor Mínimo

MAX = Valor Máximo

AVG = Média Aritmética

SUM = Total (Soma)

COUNT = Contar quantidade de itens

Sintaxe:

SELECT (nome da função) (nome da coluna em () ) FROM (nome da tabela)

***VIEW***

É um objeto (ou tabela virtual) cujo o conjunto de resultado é extraído de uma query.

As views são usadas como uma camada de abstração por desenvolvedores de banco de dados. As vezes, também são usadas para dar segurança aos dados de diversas maneiras.

Não se deve utilizar SELECT\* em views.

O mesmo somente é utilizado quando houver alguma alteração em algum campo da tabela, sendo replicado no código da view.

A cláusula ORDER BY não é válida em views.

Só será válida com uso dos comandos TOP ou OFFSET. Nesse caso, ORDER BY é usado para determinar quais linhas serão retornadas.

A criação de uma View:

Primeiro o comando CREATE VIEW;

Seguido do nome da view

Após o nome o prefiro AS

O ultimo item é a consulta que queremos armazenar dentro da view

Boas práticas: Criar o nome dos objetos de programação T-SQL com um prefixo seguido de Underline. Para View (uVW\_), para Funcions (uFN\_), Triggers (uTR\_) e Store Procedures (uSP\_).

Sintaxe de criação

CREATE VIEW uVW\_NAME

AS

QUERY

Consultando uma VIEW

SELECT \* FROM uVW\_NAME

Alteração na estrutura da View

ALTER VIEW uVW\_NAME

AS

QUERY

Apagando objeto view

DROP VIEW uVW\_NAME

Visualizando código de uma view já existente

SP\_HELPTEXT uVW\_NAME

***STORED PROCEDURE***

São um conjunto de instruções (uma ou mais) SQL normalmente agrupadas para executar uma rotina específica.

AS stored procedures são comparáveis com as funções, mas possuem flexibilidade e recursos impossíveis dentro de funções.

Vantagens da utilização de Stored Procedures

Oferecem maior desempenho, devido ao código compilado.

São fáceis de manter, pois as alterações são centralizadas, em vez de serem feitas com código.

Oferecem um alto nível de segurança. Pois em vez de o acesso ser concedido ao objeto, a permissão pode ser dada somente para o stored procedure.

Basicamente, as stored procedures cria um nível de abstração para permissões – em vez de se conceder ao usuário direitos SELECT, INSERT, UPDATE ou DELETE, ele pode receber direitos EXECUTE para um stored procedure

AS procedures podem receber ou não parâmetros.

Podemos criar procedures que irão nos retornar os dados de uma tabela (SELECT) ou para inserir/atualizar/remover dados de uma tabela.

**Sintaxe com parâmetro**

CREATE PROC OU PROCEDURE SP\_NAME

(@PARAMETER TYPE\_PARAMETER)

AS

BEGIN

QUERY

END

**Sintaxe sem parâmetro**

CREATE PROC OU PROCEDURES SP\_NAME

AS

BEGIN

QUERY

END

**Sintaxe Alterar Procedure**

ALTER PROC OU PROCEDURES SP\_NAME

(@PARAMETER TYPE\_PARAMETER)

AS

BEGIN

QUERY

END

**Sintaxe para remover o objeto procedure:**

DROP PROC OU PROCEDURE SP\_NAME

**Visualizando código de uma procedure já existente**

SP\_HELPTEXT SP\_NAME

Executando uma Stored Procedure

Comando EXEC / EXECUTE + nome da procedure criada + parâmetros passados na criação se existirem.

**Sintaxe**

EXEC/EXECUTE SP\_NAME PARAMETERS

**Criar um procedimento Armazenado**

Sintaxe:

CREATE PROCEDURE nome\_procedimento

(@Parâmetro Tipo\_dados)

AS

Bloco de códigos

Exemplo 1

CREATE PROCEDURE teste

AS

SELECT ‘Bóson Treinamentos’ AS Nome

Para executar:

EXEC(UTE) teste

Obs.: Se o procedimento armazenado for o primeiro comando de um batch, não é necessário usar a palavra EXEC.

Exemplo 2

CREATE PROCEDURE p\_LivroValor

AS

SELECT Nome\_Livro, Preco\_livro FROM tbl\_Livro

**Visualizar conteúdo de SP**

Use o procedimento armazenado sp\_helptext para extrair o conteúdo de texto de uma stored procedure:

EXEC sp\_helptext nome\_procedimento

Ex.:

EXEC sp\_helptext p\_LivroValor

**Criptografar Stored Procedure**

CREATE PROCEDURE p\_LivroISBN

WITH ENCRYPTION

AS

SELECT Nome\_Livro, ISBN

FROM tb\_Livro

Agora tente visualizar seu contéudo com sp\_helptext:

EXEC sp\_helptext p\_LivroISBN

**Modificar Stored Procedure**

ALTER PROCEDURE

ALTER PROCEDURE <nomeprocedimento> bloco de código da sp

**Parâmetros de Entrada**

ALTER PROCEDURE teste (@par1 AS int)

AS

SELECT @part

Executar passando um parâmetro:

EXEC teste 22 –22 é o valor do parâmetro passado

***TRIGGER***

São procedimentos armazenados, disparados pela execução de comando DML (insert, update ou delete), comandos DDL (create table, alter procedure, drop index, etc) ou logon do usuário.

Podem ser executados em views ou tabelas.

Categorias de Triggers

FOR/AFTER

INSTEAD OF

FOR = É o valor padrão e faz com o que o gatilho seja disparado junto da ação.

AFTER = Faz com que o disparo se dê somente após a ação que o gerou ser concluída.

Quando um comando insert/update/delete é executado em uma tabela que tenha uma trigger FOR ou AFTER associada a ela, primeiro é feita a inserção/alteração/remoção na tabela, para que a trigger seja disparada.

Caso o comando cause um erro, a trigger não é disparada;

**INSTEADOF**

Faz com que a trigger seja executado no lugar da ação que o gerou, ou seja, se um UPDATE for executado em uma tabela que tenha uma trigger associada, antes de realizar a atualização na tabela o comando DML vai disparar a trigger e a mesma além de executa os procedimentos, também executará o comando DML que foi realizado.

Podem ser utilizadas sobre tabelas e visões;

Utilizada para permitir ou estender as possibilidades de update sobre visões.

***User Functions***

Funções de usuário

Funções em T-SQL podem ou não aceitar parâmetros, efetuar cálculos lógicos e complexos, e retorna dados.

Tipos de funções:

- Escalares: Retornam um único valor

- Table-Valued: Retornam um conjunto de resultados.

***ÍNDICES***

Um índice é uma estrutura em disco, baseada em tabela ou views. Seu objetivo é tornar a recuperação dos dados mais rápidas e eficientes, na maioria dos casos.

Os índices trabalham sobre uma estrutura chamada B-Tree (Binary Tree – Árvore Binária). Uma B-Tree identifica os dados semelhantes e os agrupa. O rápido acesso aos dados proporcionado por um índice pode ser atribuído ao fato de que as pesquisas feitas em uma B-Tree são baseadas em valores de chave. Como uma B-Tree agrupa registros com chaves semelhantes, o banco de dados precisará navegar por apenas algumas páginas para encontrar os registros.

Os termos bitmap, b-tree, heap e hash são associados a estruturas de índice.

**Índices Clustered**

Só poderá existir um índice clusterizado por tabela ou view.

Um índice clusterizado muda o modo como os dados são armazenados e classificados.

Quando se cria uma chave primaria na tabela, automaticamente o SQL Server cria um índices clustered.

Um índice clusterizado classifica e armazena as linhas de dados da tabela ou exibição em ordem com base na chave de índice clusterizado. O índice clusterizado é implementado como uma estrutura de índice de árvore B que oferece suporte à recuperação rápida das linhas, com base em seus valores de chave de índice clusterizado.

**Índices Nom-Clustered**

Podemos ter quantos forem necessários por tabela ou view.

Quando colocamos uma coluna com a restrição do tipo UNIQUE, ela cria um índice não clusterizado na tabela automaticamente.

Este tipo de índice não altera a ordem da tabela. Ao invés disso, os registros estarão armazenados em uma ordem aleatória, entretanto, poderão ser facilmente localizados através do ponteiro.

Um índice não clusterizado pode ser definido em uma tabela ou exibição com um índice clusterizado ou em um heap. Cada linha de índice no índice não clusterizado contém o valor da chave não clusterizada e um localizador de linha. Esse localizador aponta para a linha de dados no índice clusterizado ou heap com o valor da chave. As linhas no índice são armazenadas na ordem dos valores de chave de índice, mas não há garantia de que as linhas de dados estejam em nenhuma ordem específica, a menos que um índice clusterizado seja criado na tabela.

Índices Clustered :Ocupa fisicamente espaço no Banco

Índices Nom-Clustered:Ele é logico

**Hash**

Com um índice de hash, os dados são acessados por meio de uma tabela de hash na memória. Os índices de hash consomem uma quantidade fixa de memória, que é uma função da contagem de buckets.

**Memory optimized Nomclusterd**

Para índices não clusterizados com otimização de memória, o consumo de memória é uma função da contagem de linhas e do tamanho das colunas de chave de índice.

***Backup***

O principal objetivo do backup é proteger o banco de dados contra perda de dados e inatividade. Contendo em algum tipo de repositória, uma cópia funcional do banco de dados.

Com eles, um DBA pode fazer uma restauração a partir do último backup ou em um ponto de tempo específico.

SQL Server possui 3 tipos de backup:

-Backup Completo

-Backup Diferencial

-Backup de logs de transação

**Backup completo ou Full:** É realizado uma copia do banco de dados inteiro, ou seja, toda sua estrutura de criação e os dados.

**Backup Diferencial:** Os backups diferenciais são menores do que os completos, pois capturam apenas o que foi alterado no banco de dados desde o seu último backup completo. Para realizar um backup diferencial é necessário ter realizado um backup full.

**Backup de logs de transação:** São utilizados para restaurar todos as transações realizadas no banco de dados, podendo ser utilizado para restaurar um ponto específico no tempo.

Para os mesmos funcionarem de forma eletiva, é necessário ter realizado um backup full e o banco deve estar em modo de recuperação full ou bulk-logged.

**Modelos de Recuperação**

O modelo determina a precisão que um banco de dados pode ser restaurado:

**Modelo Simples**: requer menos administração, pois os backups do log de transação são truncados regularmente.

**Modelo Full**: permite restaurar em um determinado momento, pois arquivos de log registram todas as transações SQL e a hora em que foram executadas. As desvantagens do modelo full é que os logs podem crescer muito.

**Modelo Bulk-logged**: É o modelo menos usado. Permite um bom desempenho enquanto utiliza o menor espaço de log. No entanto, você não pode fazer uma recuperação algum ponto no tempo.

**RESTORE**

É o processo de restauração de um backup em uma instância do SQL SERVER.

É possível restaurar um backup para:

- Tornar disponível um banco inativo ou com anormalidades.

- Criar um ambiente de testes / desenvolvimentos

- Realizar migração de ambientes

- Restaurar dados que foram perdidos no ambiente de produção

- Entre outras coisas

Podemos restaurar os arquivos de backup

- Completo

- Diferencial

- Logs de transação

Cenários de restaurações possíveis no SQL Server;

**\* Restauração completa do banco de dados:**

- Restaura um banco de dados inteiro, começando com um backup full, que pode ser seguido pela restauração de um backup diferencial e posteriormente backups de log.

**\* Restauração de arquivo:**

- Restaura um arquivo ou um grupo de arquivos em um banco de dados. Após uma restauração completa do arquivo, um backup diferencial de arquivos pode ser restaurado.

**\* Restauração de página:**

- Restaura páginas individuais.

**\* Restauração fragmentada:**

- Restaura um banco de dados em estágios, começando com o grupo de arquivos primário e um ou mais grupos de arquivos secundários.

**\* Apenas recuperação:**

- Recupera dados que já são consistentes com o banco de dados e precisam apenas ser disponibilizados.

**\* Restauração do log de transação:**

- No modelo de recuperação deve estar como full ou buck-logged, uma vez que os logs registram cada transação, é necessária a restauração dos backups de log para atingir o ponto de recuperação desejado.

**\* Criar um banco de dados espelho:**

- Quando você tem um banco de dados espelho, você tem banco de dados duplicados em vários servidores. Quando as informações são gravadas em um servidor, elas são replicadas automaticamente para o segundo servidor.

**\* Criar e manter um servido em espera:**

- Quando você tem um servidor em espera, está usando cluster ativo-passivo que consiste em dois ou mais servidores. Quando o servidor ativo falha, o servidor passivo se torna o servidor ativo, permitindo um tempo de inatividade mínimo.

**Segurança do SQL SERVER**

\* Um login ou logon é o processo pelo qual o acesso individual a um sistema de computador é controlado por identificação do utilizador através das credenciais que ele ou ela oferece. O método de login mais comum envolve o fornecimento tanto um nome de usuário e senha.

Contas de serviço do SQL Server:

**- Local account**: Boa para ser utilizada quando nenhum acesso a rede é necessário, restrito a uma máquina local.

**- System Account:** Boa para ser utilizada quando nenhum acesso a rede é necessário e é bom para instalações de um único servidor.

**- Domain account**: Boa para ser utilizado em todos outros cenários e é geralmente o recomendado para ambientes de produção.

**SEGURANÇA A NÍVEL DE SERVIDOR**

**LOGIN**

Conecta você ao servidor e pode atribuir certos tipos de permissões a nível de servidor.

As portas padrão para conexão com o SQL são: 1433 e 1434

Existem dois tipos de autenticação para logins:

- Windows Authetication: Autenticação via Active Directory. Habilitada por padrão na instalação do Sql Server e pode ser permitida para um usuário ou um grupo.

- SQL Authetication: Autenticação por logins criados diretamente no SQL Serve. Desasbilitada por padrão na instalação. Nesse tipo de autenticação o SQL Server mantém sua própria política de segurança de senhas.

Logins podem ter uma ou mais roles(regras de login), que vão indicar quais as permissões e quais as atividades podem ser feitas por eles.

**SERVER ROLES:**

- Bulkadmin: Pode executar operações de inserção em massa (bulk insert);

- Dbcreator: Pode criar, alterar, excluir e restaurar bancos de dados.

- Diskadmin: Pode criar, alterar e excluir arquivos de disco.

- Processadmin: Pode matar um processo do SQL Server em execução;

- Securityadmin: Pode gerenciar os logins do servidor;

- Serveradmin: Pode definir as configurações de todo o servidor, incluindo a pesquisa de texto completo (full text Search) e o desligamento do servidor.

- Setupadmin: Pode configurar Linked Servers, Extended Stored Procedures e o inicializar Stored Procedures;

- Sysadmin: pode executar qualquer atividade na instalação do SQL Server, independentemente de qualquer outra configuração de permissão. Tem acesso a todas as permissões de servidor.

**DATABASE USER**

- Logins são mapeados dentro de cada banco de dados como usuários.

- Um login pode mapear diferentes usuários em diferentes bancos de dados.

- Usuários dentro de banco de dados, recebem permissões para atuar dentro dos bancos de dados.

- Usuários podem receber atribuições de uma ou mais database roles ou user-defined roles.

**DATABASE ROLES**

- db\_acessadmin: Autoriza um usuário a acessar o banco de dados , mas não gerencia a segurança no nível do banco de dados.

- db\_backupoperator: Permite que um usuário execute backups, pontos de verificação e comandos DBCC, mas não restaura. (Apenas administradores de sistema do servidor podem executar restaurações}).

- db\_datareader: Autoriza um usuário a ler todos os dados no banco de dados.

- db\_datawriter: Permite que um usuário grave todos os dados no banco de dados,

- db\_ddladmin: Autoriza um usuário a emitir comandos DDL (CREATE, ALTER, DROP)

- db\_denydatareader: Bloqueia um usuário de ler qualquer tabela no banco de dados.

- db\_denydatawriter: Bloqueia um usuário de modificar dados em qualquer tabela no banco de dados

- db\_owner: Esta é uma função especial, que possui todas as permissões no banco de dados. Essa função inclui todos os recursos das outras funções e difere da função de usuário dbo.

- db\_securityadmin: Permite que um usuário gerencie a segurança no nível do banco de dados, incluindo funções e permissões.