SQL

SQL

Bases de dados

Trabalhar no servidor

- Para trabalhar no servidor, entra-se com o comando sqlcmd e os acessos.

sqlcmd –S server –U user –P password

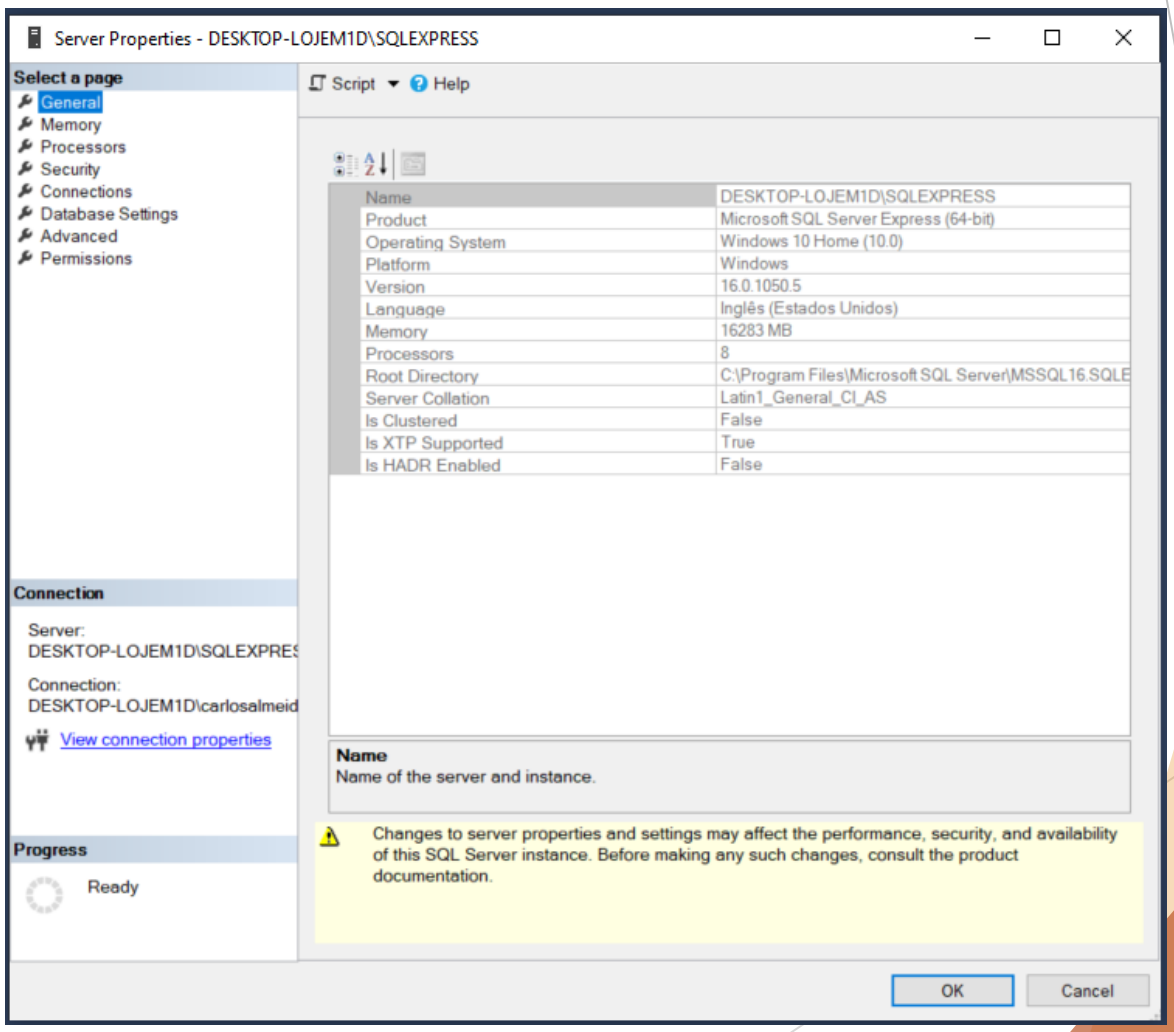
SQLCMD -S DESKTOP-LOJEM1D -U sa -P 1234

- De seguida executam-se os comandos necessários: select, insert, update ou delete:

use cet90_northwind;

go

Propridades do servidor



Propriedades do servidor

- General (Geral): visualizar e modificar informações básicas sobre o servidor, como nome do servidor, versão do SQL Server, número da versão do produto, edição, entre outros.
- Memory (Memória): definir as configurações de alocação de memória para o SQL Server. Pode-se configurar o limite de memória máxima, ajustar o tamanho do cache de buffer e definir configurações relacionadas com o uso da memória pelo SQL Server.
- Processors (Processadores): configurar o uso de recursos de CPU pelo SQL Server. É possível definir o número máximo de processadores que o SQL Server pode utilizar e especificar a afinidade de CPU, ou seja, quais núcleos de CPU devem ser usados pelo SQL Server.

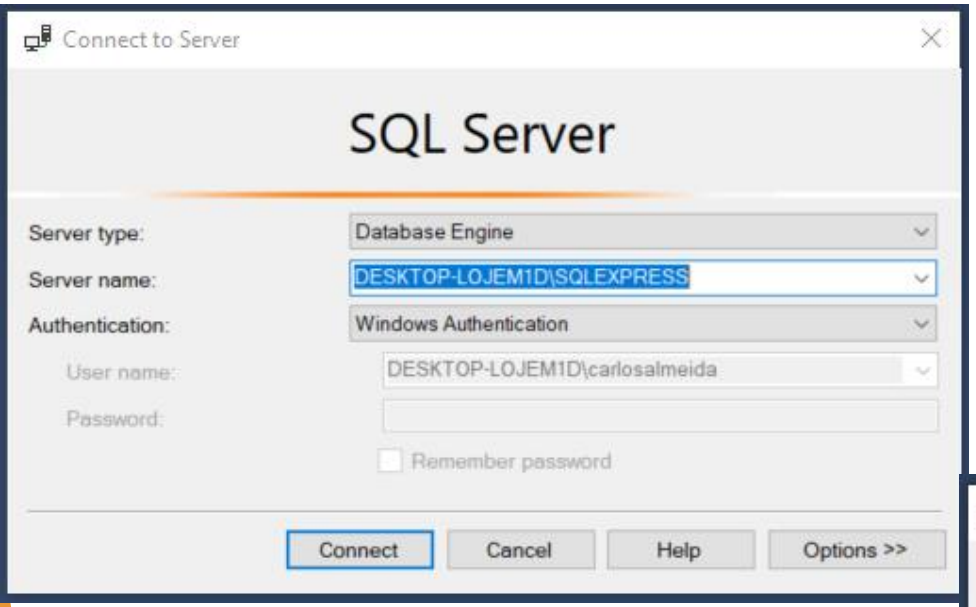
Propriedades do servidor

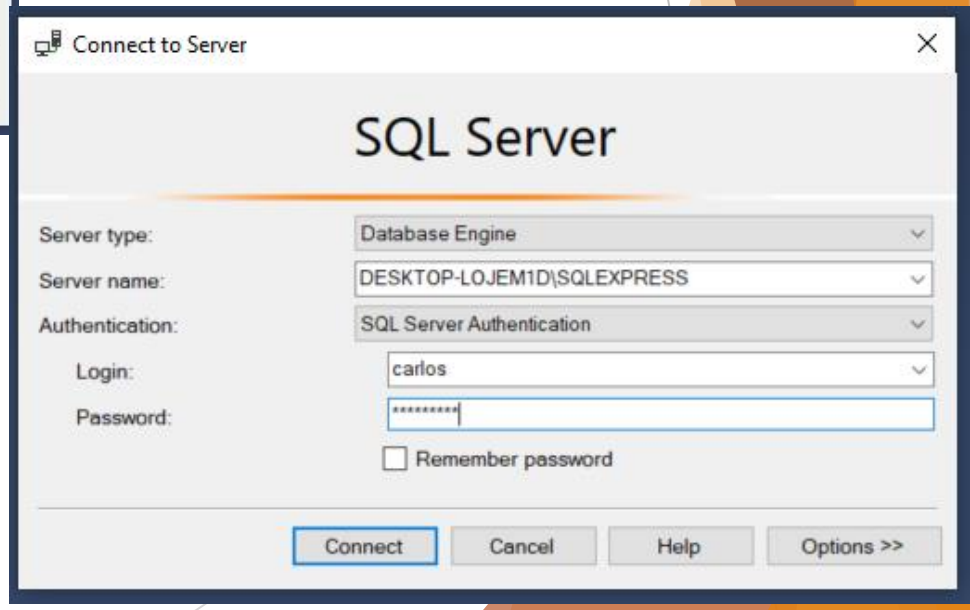
- Connections (Conexões): definir as configurações relacionadas com as ligações ao servidor SQL. Isso inclui o número máximo de ligações simultâneas permitidas, o tempo limite de ligação e as configurações de protocolo de rede.
- Security (Segurança): configurar as opções de segurança do SQL Server. Isso inclui a configuração do modo de autenticação, a definição de regras de password, a habilitação do login do SQL Server, entre outras configurações relacionadas à segurança do servidor.
- Database Settings (Configurações da base de dados): configurações específicas da base de dados, como o seu tamanho máximo, o modo de recuperação, as opções de colação padrão e outras configurações relacionadas ao comportamento das bases de dados no servidor.

Propriedades do servidor

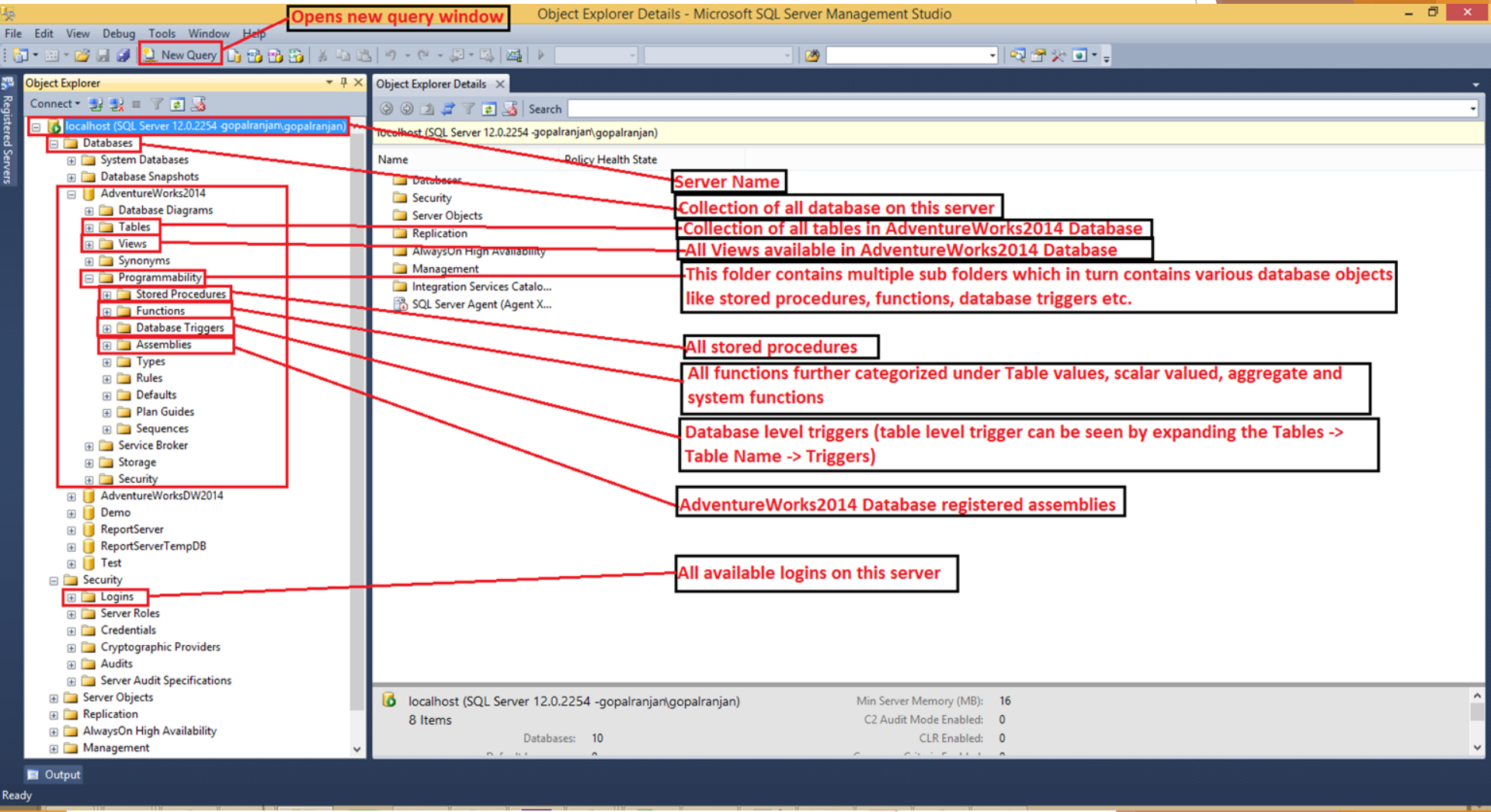
- Advanced (Avançado): configurações avançadas do SQL Server. Essas configurações abrangem várias áreas, como desempenho, comportamento do servidor, recursos específicos, configurações do SQL Server Agent, opções de inicialização, entre outros.
- Permissions (permissões): controlam o acesso e os privilégios ao nível do servidor. Incluem permissões como "sysadmin" (administrador do sistema), "serveradmin" (administrador do servidor), "securityadmin" (administrador de segurança) e outras funções de servidor predefinidas.

Ambiente SQLMS

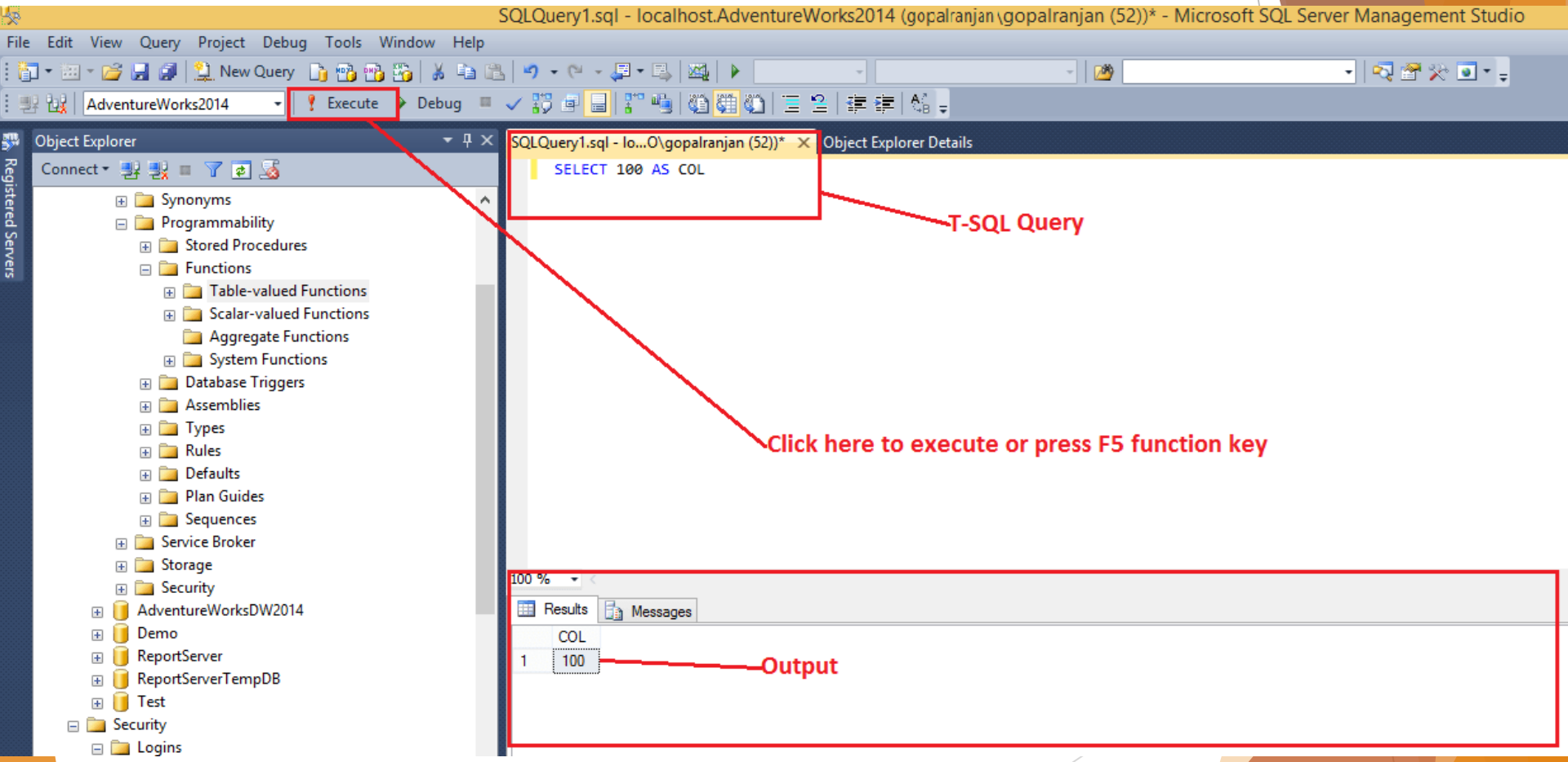




Ambiente SQLMS



Ambiente SQLMS



SQL

Linguagem DDL

Programação em SQL

- As Bases de Dados foram desenvolvidas para:
  - Fornecer acesso facilitado aos dados (DATA);
  - Gestão de grandes volumes de informação (DATA);
  - Responder a todo o tipo de perguntas ao sistema;
  - Relacionar-se com as mais variadas bases dados existentes e tabelas existentes.
- A linguagem padrão dos Bancos de Dados Relacionais é a Structured Query Language, ou simplesmente SQL, como é mais conhecida.

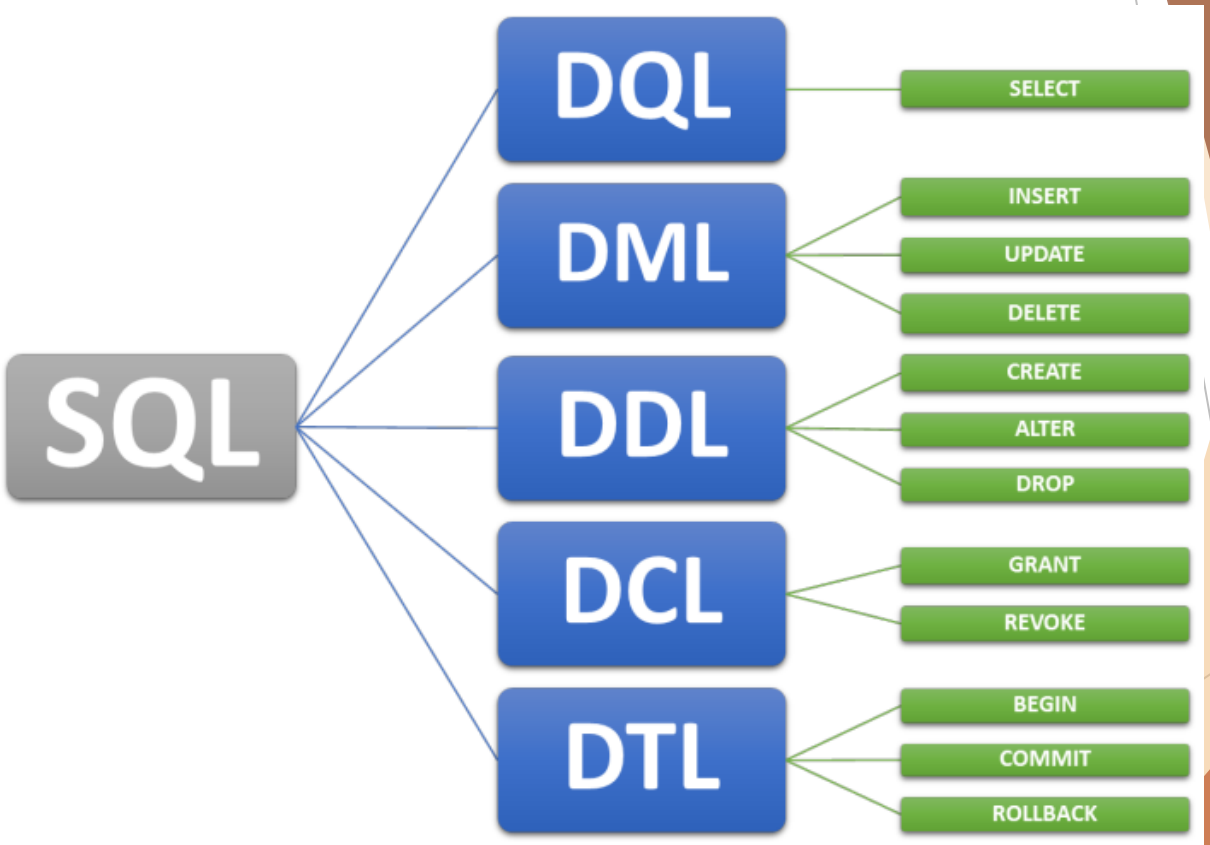
Programação em SQL

- As bases dados são:
  - Conjuntos de dados com uma estrutura regular que organizam informação;
  - Um conjunto de informações relacionadas entre si, referentes a um mesmo assunto e organizadas de maneira útil, com o propósito de servir de base para que o utilizador recupere informações, tire conclusões e tome decisões.
  - (Database) Aplicações cujo objetivo é compilar, organizar e armazenar informações em meio eletrónico de forma estruturada.
  - Conjunto de informações armazenas e ordenadas para consulta imediata por meio de um palavra-chave (primary key)

Componentes da Linguagem

- A linguagem SQL é dividida em subconjuntos de acordo com as operações que queremos efetuar sobre uma base de dados
  - DDL (Data Definition Language) - linguagem de definição de dados (criação, alteração e exclusão de tabelas, indices, views,...)
  - DML (Data Manipulation Language) - linguagem de manipulação de dados (consultas e actualizações)
  - DCL (Data Control Language) – permite conceder diferentes permissões aos utilizadores da BD
  - DTL (Linguagem de Transação de Dados) -
  - DQL (Linguagem de Consulta de Dados) -

Componentes da Linguagem



Base de dados

- master: base de dados primária do sistema. Contém metadados sobre todas as outras bases de dados bem como informações sobre as configurações do servidor.
- model: é usada como um modelo para criar novas bases de dados. Qualquer objeto criado na base de dados "model" estará disponível em todas as novas bases de dados criadas no servidor.

Base de dados

- msdb: usada pelo SQL Server Agent para armazenar informações sobre tarefas agendadas, alertas, operadores e histórico de backup e restauração. Também é usada para outras funcionalidades de gestão, como integração com o Serviços de Relatório (Reporting Services).
- tempdb: base de dados temporária usada para armazenar dados temporários, como tabelas temporárias, variáveis de tabela e outros objetos temporários criados durante a execução de consultas.

Base de dados

- O dbo (Database Owner) é um esquema especial e o proprietário do banco de dados. É utilizado para organizar objetos dentro da base de dados, como tabelas, views, stored procedures e funções.
- Um esquema é um container lógico dentro da base de dados que agrupa objetos (tabelas, views, procedures, etc.), ajudando a organizar e controlar o acesso aos objetos da base de dados.
- Se não se especificar o esquema ao criar um objeto, o SQL Server atribuí-o ao esquema padrão do utilizador.

Base de dados

- dbo é o esquema padrão para objetos criados pelo dono da base de dados.
- O utilizador com a permissão db_owner (ou seja, o dono da base de dados) automaticamente tem objetos atribuídos ao esquema dbo quando os cria.
- Se um utilizador sem permissões de db_owner criar um objeto, ele pode ser atribuído a um esquema diferente.
- Quando usar dbo?
- Em bases de dados pequenos ou médios sem necessidade de separação por esquemas.
- Para garantir compatibilidade e evitar confusão ao aceder objetos.
- Quando criar outros esquemas?
- Quando há necessidade de segmentação por departamentos (Ex: Financeiro, RH, Vendas).
- Para facilitar gestão de permissões (Ex: Restringir acesso a Financeiro_Pagamentos)

Base de dados

- Ver bases de dados existentes

SELECT name FROM sys.databases (contém uma linha para cada BD existente. Fornece informações sobre cada BD, como nome, ID, status de recuperação, versão de compatibilidade

- Criar bases de dados

CREATE DATABASE [IF NOT EXISTS] nome

- Apagar uma base de dados

DROP DATABASE nome

- Ver bases de dados existentes

USE nome

Tabelas

- As **tabelas** implementam na base de dados as principais ideias identificados pelo modelo conceptual.
- São compostas por um **conjunto finito bem definido de atributos.**
- Cada **atributo** possui:
  - um **nome,**
  - um **tipo de dados**,
  - um **tamanho (opcional)**,
  - uma **característica de nulidade** e, possivelmente,
  - **restrições**

Tipos de dados

- Data Types: Texto
  - CHAR(n): Dados de caracteres de comprimento fixo com um comprimento especificado (n).
  - VARCHAR(n): Dados de caracteres de comprimento variável com um comprimento máximo (n).
  - VARCHAR(MAX): Armazena grandes quantidades de caracteres até 2 GB.

- Se os dados excederem o tamanho, o valor é truncado
- Espaços não ocupados não são removidos aquando da inserção

Tipos de dados

- Data Types: Numérico inteiro
  - TINYINT: Armazena pequenos números inteiros dentro do intervalo de 0 a 255.
  - SMALLINT: Representa números inteiros menores dentro do intervalo -32.768 a 32.767.
  - INT: Representa números inteiros dentro do intervalo -2.147.483.648 a 2.147.483.647.
  - BIGINT: Armazena números inteiros maiores dentro do intervalo -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.

Tipos de dados

- Data Types: Numérico com vírgula
  - FLOAT: Armazena números aproximados com precisão variável, entre 1 a 53.
  - REAL: Armazena números com uma precisão fixa de 7 dígitos.
  - DECIMAL(p, s): Representa números decimais de ponto fixo com uma precisão (p) e escala (s).
  - NUMERIC(p, s): Semelhante a DECIMAL, representando números decimais de ponto fixo com precisão e escala especificadas.
- "p" representa o número total de dígitos (precisão)
- "s" representa o número de dígitos após o ponto decimal

Tipos de dados

- Data Types: Datas
  - DATE: Armazena valores de data sem hora.
  - HOUR: Representa uma hora do dia.
  - DATETIME: Armazena os valores de data e hora.
  - DATETIME2: Semelhante a DATETIME, com maior precisão e intervalo de datas maior.
  - TIMESTAMP: Regista automaticamente a última vez que uma linha foi modificada

- Data Types: Booleanos
  - BIT: Representa um valor binário de 0 ou 1, onde 0 representa falso e 1 representa verdadeiro.

Criação de tabelas

CREATE TABLE nome_tabela (
nome_atributo1 < tipo > [CONSTRAINT],
nome_atributo2 < tipo > [CONSTRAINT],
.....
nome_atributoN < tipo > [CONSTRAINT],
PRIMARY KEY(nome_atributo));

- nome_tabela: indica o nome da tabela a ser criada.
- nome_atributo: indica o nome do campo a ser criado na tabela.
- tipo: indica a definição do tipo de atributo (integer(n), char(n), ...).

Criação de tabelas

- CONSTRAINT: permite colocar restrições para limitar o tipo de dados a introduzir numa tabela.
- IDENTITY(1,1): incrementa automaticamente a numeração
- NOT NULL: uma coluna não pode ter o valor NULL
- DEFAULT: valor padrão para uma coluna quando nenhum é especificado
- UNIQUE: todos os valores numa coluna são diferentes
- CHECK: todos os valores numa coluna satisfazem um determinado critério
- PRIMARY KEY: identifica de forma única uma linha na tabela
- FOREIGN KEY: garante a integridade referencial dos dados

Criação de índices

- Os índices devem ser aplicados em colunas de dados quase únicos.
- Podem também ser criados índices para colunas que poderão auxiliar em consultas que tenham a cláusula WHERE, precisando ou não usar os operadores AND, OR ou NOT.
- Um índice CLUSTERED reorganiza fisicamente os dados na tabela com base nas colunas do índice. Cada tabela pode ter apenas um índice CLUSTERED, pois ele define a ordem física das linhas de dados na tabela.
- Um índice NON-CLUSTERED não altera a ordem física das linhas de dados na tabela. Em vez disso, ele cria uma estrutura de dados separada que contém chaves de índice e ponteiros para as linhas de dados correspondentes

Criação de índices

- Devemos considerar os seguintes pontos antes de criar índices:
  - Quando as colunas indexadas são modificadas, o SGBD desloca recurso internamente para manter esses índices atualizados e associados;
  - A manutenção de índices requer tempo e recursos, portanto, não se deve criar índices que não serão usados efetivamente;
  - Quando se tem grandes quantidade de dados duplicados, como por exemplo, sexo.

Criação de tabelas

CREATE TABLE < nome_tabela > (

nome_atributo < tipo > IDENTITY (1,1),

nome_atributo < tipo > NOT NULL,

nome_atributo < tipo > DEFAULT 80,

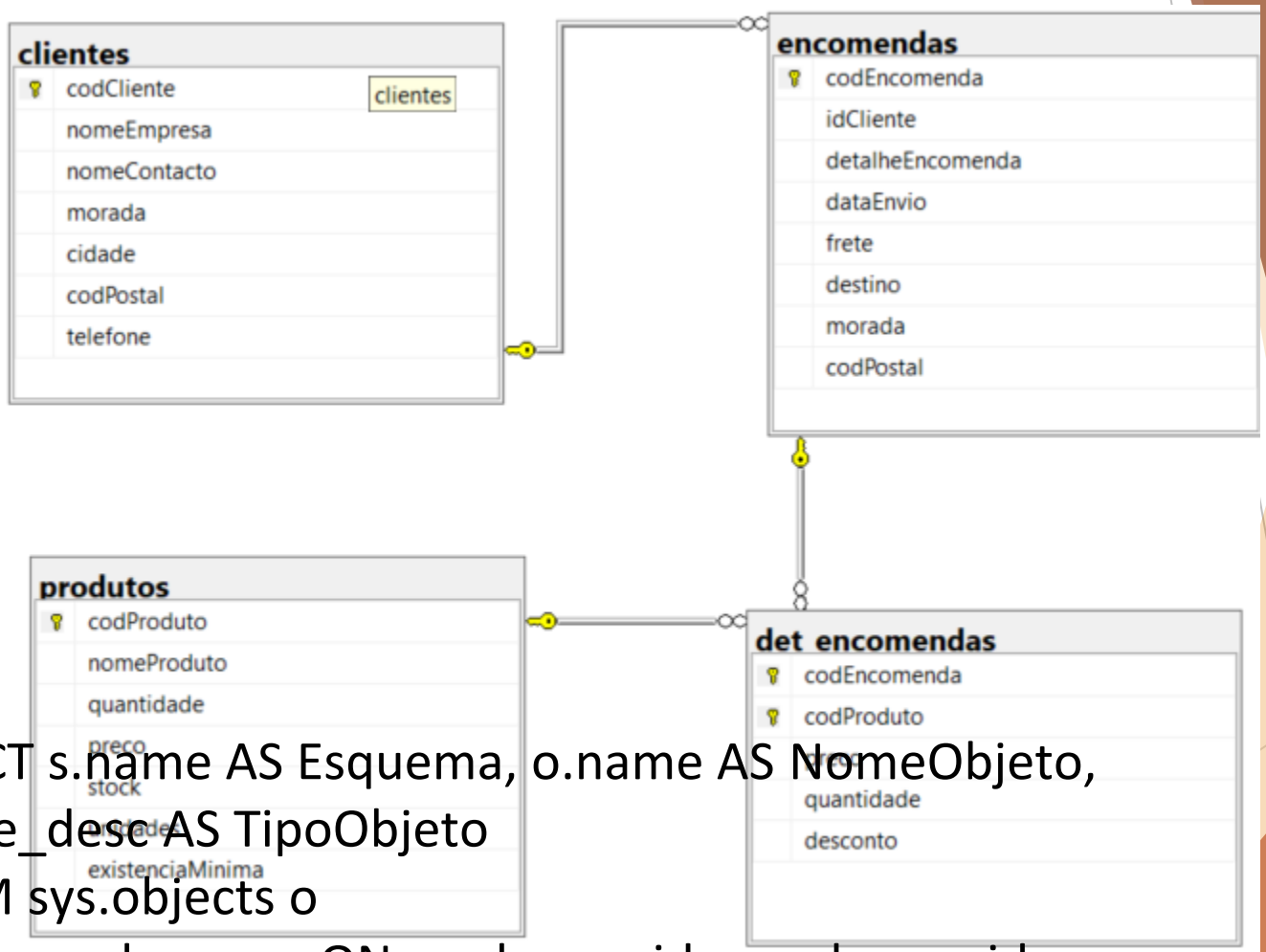
nome_atributo < tipo > UNIQUE,

nome_atributo < tipo > CHECK (nome_atributo>=???),

INDEX (nome) (nome_atributo),

FOREIGN KEY (nome_atributo) REFERENCES <outra tabela> (nome_atributo))

Criação de tabelas



- SELECT s.name AS Esquema, o.name AS NomeObjeto, o.type_desc AS TipoObjeto
- FROM sys.objects o
- JOIN sys.schemas s ON o.schema_id = s.schema_id
- WHERE s.name = 'dbo

Criação de tabelas

```
CREATE TABLE clientes(
    codCli INT IDENTITY (1,1) PRIMARY KEY,
    nomeCli VARCHAR (50) UNIQUE NOT NULL,
    nomeContato VARCHAR (40) NOT NULL,
    morada VARCHAR (50) NOT NULL,
    cidade VARCHAR (20) NOT NULL ,
    codPostal VARCHAR (8) NOT NULL,
    telefone VARCHAR (15) NOT NULL,
    mail VARCHAR(50) NOT NULL,
    INDEX nome_constraint (mail))
```

Criação de tabelas

```
CREATE TABLE produtos(
    codProd INT IDENTITY(1,1) PRIMARY KEY,
    nomeProd VARCHAR(50) NOT NULL,
    quantUnidade TINYINT DEFAULT 0,
    precoUnitario DECIMAL(5,2) NOT NULL,
    stock SMALLINT DEFAULT 1,
    stockMinimo TINYINT)
```

Criação de tabelas

```
CREATE TABLE encomendas(
    codEnc INT IDENTITY(1,1) PRIMARY KEY,
    codCli INT NOT NULL,
    dataEnvio DATE NOT NULL,
    frete DECIMAL(5,2) NOT NULL,
    morada VARCHAR(50) NOT NULL,
    CONSTRAINT fk_enc_cli FOREIGN KEY (codCli)
    REFERENCES clientes (codCli))
```

Criação de tabelas

```
CREATE TABLE detalhesdaencomenda(
    codEnc INT NOT NULL,
    codProd INT NOT NULL,
    precoUnit DECIMAL(5,2) NOT NULL,
    quantidade INT NOT NULL CHECK(quantidade>0),
    desconto DECIMAL(4,2) DEFAULT 0,
    PRIMARY KEY(codEnc, codProd),
    CONSTRAINT fk_de_pr FOREIGN KEY (codProd) REFERENCES
            produtos(codProd),
    CONSTRAINT fk_de_en FOREIGN KEY (codEnc) REFERENCES
            encomendas(codEnc))
```

Index vs Unique

- Diferenças entre campos UNIQUE e campos INDEX:
  - **UNIQUE** garante que todos os valores numa coluna sejam diferentes entre si.
  - Nenhum valor duplicado é permitido na coluna ou no conjunto de colunas definido como UNIQUE.
  - **INDEX** é uma estrutura organizada que melhora a velocidade de recuperação de dados numa tabela.
  - Não impõe necessariamente a restrição de unicidade como um campo UNIQUE, sendo utilizado principalmente para otimizar consultas.

Index vs Keys

- Diferenças entre campos KEYS e campos INDEX:
  - Chaves são utilizadas para garantir a consistência dos dados e manter integridade referencial entre tabelas.
  - **Primary Key** define uma coluna (ou um conjunto de colunas) que identifica cada registo numa tabela.
  - **Foreign Key** estabelece uma relação entre duas tabelas, onde a chave estrangeira na tabela filha referencia a chave primária (ou uma chave única) na tabela pai.
  - **Unique Key** garante que todos os valores numa coluna (ou conjunto de colunas) sejam únicos, permitindo valores nulos.

Index vs Keys

- Diferenças entre campos KEYS e campos INDEX:
  - Índices: Utilizados para acelerar consultas e operações de busca em grandes volumes de dados.
  - **Clustered Index** reorganiza fisicamente os dados na tabela com base nas colunas do índice.
  - Uma tabela pode ter apenas um índice clusterizado. Geralmente é associado à chave primária, se não especificado explicitamente em outra coluna.
  - **Non-Clustered Index** cria uma estrutura de dados separada para armazenar chaves de índice e ponteiros para as linhas de dados correspondentes.
  - Pode ser criado em colunas que não são a chave primária. Pode haver vários índices não clusterizados numa tabela.

Eliminar tabelas

- Eliminar a tabela. Se a tabela contiver dados, esses são perdidos.
- Caso haja um chave estrangeira apontando para a tabela, tem que se eliminar o CONSTRAINT (ALTER TABLE.. DROP …)
- Só pode ser realizado por utilizadores com permissões: owner, grupo sysadmin.

DROP TABLE <nome_tabela>

Alteração da estrutura

- Renomear uma base de dados

```
ALTER DATABASE empresa MODIFY NAME=companhia
```

- Renomear uma tabela

```
EXEC SP_RENAME 'encomendas', 'detalhes’
```

- Renomear um campo

```
EXEC SP_RENAME 'detalhes.desconto', 'novo_preco', 'COLUMN’
```

- Ver tabelas existentes numa base de dados

```
SELECT table_schema, table_name, table_type FROM
information_schema.tables
```

Alteração da estrutura

- Ver detalhes de uma base de dados

```
SELECT table_name, column_name, column_default, is_nullable,
numeric_precision, data_type, character_maximum_length,
datetime_precision
FROM information_schema.columns
```

- Ver informações detalhadas sobre um objeto específico na base de dados

```
EXEC sp_help 'utente';
```

Alteração de tabelas

- É possível alterar a estrutura dos campos, acrescentando, alterando ou excluindo as suas definições.
- O comando ALTER implica a alteração numa tabela.

ALTER TABLE <nome_tabela>

ESPECIFICAÇÃO

- Logo a seguir, as alterações pretendidas terão que ser declaradas no comando.

Alteração de tabelas

- Adicionar campos

ADD nome_atributo <TIPO_DADOS>

- Adicionar chave estrangeira

ADD CONSTRAINT <nome_constraint>

FOREIGN KEY (atributo) REFERENCES <outra_tabela> (atributo)

- Adicionar restrições

ADD CONSTRAINT (atributo)

Alteração de tabelas

- Apagar tabela

DROP TABLE <nome_tabela>

- Apagar campo

DROP COLUMN atributo

- Apagar uma restrição

DROP CONSTRAINT [nome_constraint]

SQL

Linguagem DML

Inserir valores

- Uma das operações mais importantes numa base de dados é a inserção de registos (dados).
- Faz-se isso com a utilização do comando INSERT INTO.

INSERT INTO tabela (coluna1, coluna2,...) VALUES (valor1, valor2,...);

- Inserir valores com o IDENTITY manualmente

SET IDENTITY_INSERT produtos ON;

INSERT INTO tabela (coluna1, coluna2,...) VALUES (valor1, valor2,...);

SET IDENTITY_INSERT produtos OFF;

Inserir valores

- Devemos especificar a tabela e quais colunas dessa tabela que receberão os dados
- Após a palavra-chave VALUES, especificamos os dados em si, na mesma ordem em que as colunas foram especificadas.
- Caso haja uma coluna com IDENTITY ou CURRENT_TIMESTAMP, não deve ser incluída na lista de colunas do comando, pois os seus dados serão gerados e inseridos automaticamente pelo SQL quando um novo registo for adicionado.

Inserir valores

- A lista de colunas pode ser eliminada se todos os dados forem fornecidos na ordem de definição da tabela:

INSERT INTO tabela VALUES (‘cccc’, 1233, ’2002-12-05’,12.4, 45)

- Dados de tipo caracter entre aspas;
- Dados de tipo data, como os caracteres;
- Dados numéricos sem aspas;
- As colunas não fornecidas assumem valor nulo ou valor default

- Colocar a PK a zero

DBCC CHECKIDENT ('Professor', RESEED, 0);

Alterar valores

- Para alterar um registo de uma tabela utilizamos o comando UPDATE.

UPDATE tabela SET coluna = novo_valor

WHERE coluna = valor_antigo;

- Caso não seja utilizada a cláusula WHERE para filtrar os registos, todos os dados da coluna serão alterados.
- Todas as linhas que qualificam para o predicado têm seus valores alterados nas colunas referenciadas.

UPDATE empregado

SET salario=salario*1.2

Alterar valores

- Pode-se fazer atualizações baseadas em cálculos matemáticos:

```
UPDATE empregado
SET salario=salario*1.2
WHERE cddept=“d1”;
```

- A atualização pode incluir a junção com outras tabelas, desde que só se atualizem colunas de uma das tabelas

```
UPDATE empregado
SET vlsalario=vlsalario*1.2
FROM empregado e,departamento d
WHERE e.nrmatric = d.nrchefe
```

Apagar valores

- Uma das tarefas mais comuns na manutenção de tabelas é a exclusão de registos. É normal, por exemplo, excluir um produto de uma tabela que não é mais vendido por uma loja, ou um cliente que apagou o registo.
- Em algumas situações especiais, pode ser necessário excluir todos os registos de uma tabela – ou seja, limpar a tabela.
- Podemos excluir registos de uma tabela por meio de duas declarações: DELETE FROM e TRUNCATE TABLE.

Apagar valores

- Com a cláusula DELETE pode-se excluir registos específicos, indicados (filtrados) por meio de uma condição na cláusula WHERE.

DELETE FROM tabela WHERE coluna = valor;

- Devemos sempre utilizar a cláusula WHERE para evitar a perda de dados da tabela, caso contrário todos os registos serão excluídos, um a um!

Apagar valores

- O comando TRUNCATE TABLE permite remover todas as linhas de uma tabela numa única operação, sem registar as exclusões de linhas individuais.
- O comando TRUNCATE TABLE equivale a executar a instrução DELETE, porém sem utilizar a cláusula WHERE. Portanto, é utilizada para apagar completamente o conteúdo de uma tabela.
- O comando TRUNCATE TABLE é mais rápido e utiliza menos recursos de sistema durante sua execução.

TRUNCATE TABLE tabela;

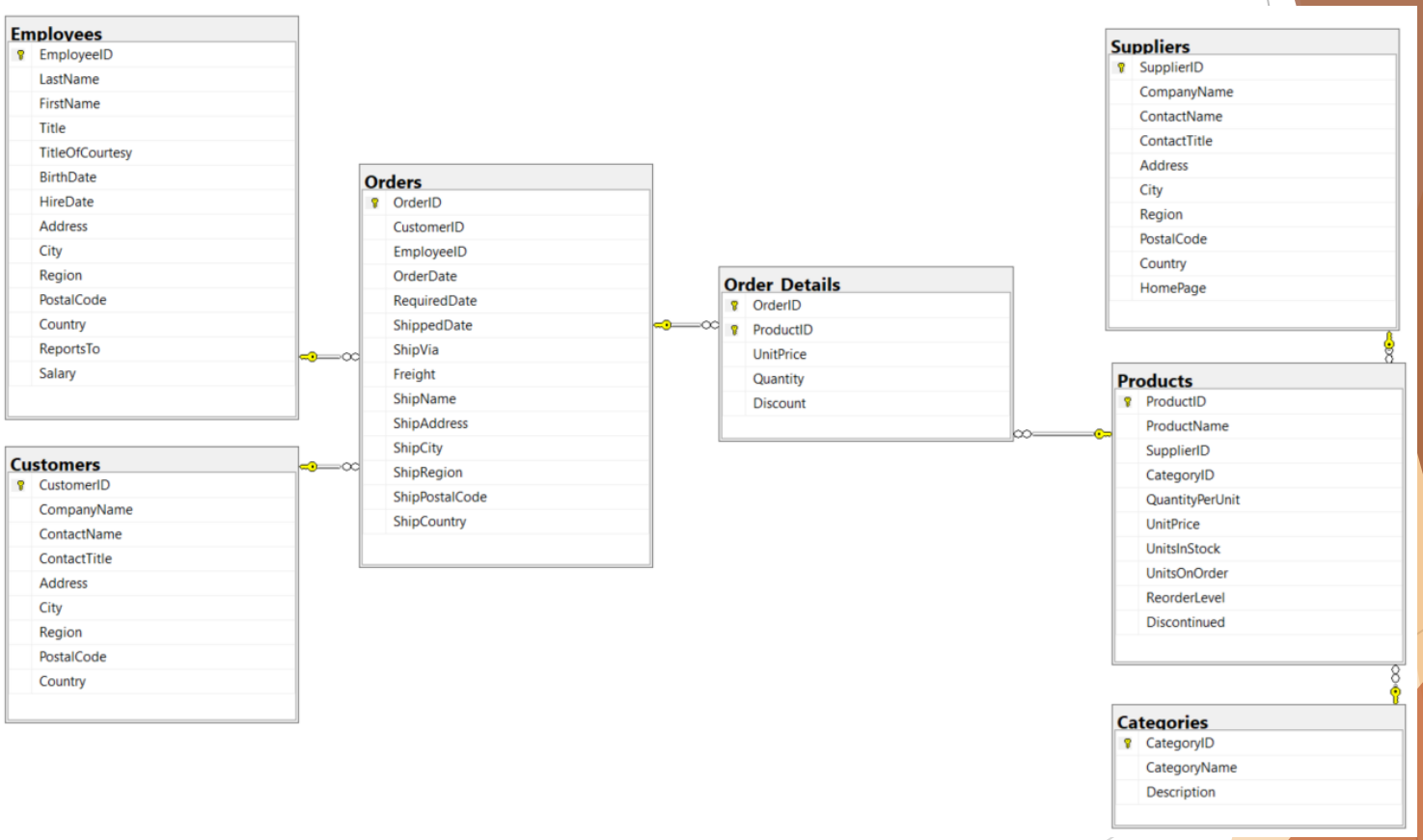
Criação de índices

- A utilização de índices pode trazer grandes melhorias para o dsempenho da base de dados.
- Estes são bons em termos de performance das pesquisas, mas, por outro lado, consomem muito espaço em disco.
- Dentro da arquitetura de um SGBD, existem dois métodos para acesso aos dados:
  - Consulta de tabela, que examina todas as páginas de dados das tabelas, começando do início da tabela passando por todos os registos, e extraindo aqueles que satisfazem os critérios da consulta.
  - Através de índices, percorrendo a estrutura da árvore do índice para localizar os registos, por comparação, extraindo somente os registos necessários para satisfazerem os critérios passados pela consulta.

SQL

Queries

Consultas


Employees
EmployeeID
LastName
FirstName
Title
TitleOfCourtesy
BirthDate
HireDate
Address
City
Region
PostalCode
Country
ReportsTo
Salary
Customers
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Orders
OrderID
CustomerID
EmployeeID
OrderDate
RequiredDate
ShippedDate
ShipVia
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode
ShipCountry
Order Details
OrderID
ProductID
UnitPrice
Quantity
Discount
Suppliers
SupplierID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
HomePage
Products
ProductID
ProductName
SupplierID
CategoryID
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued
Categories
CategoryID
CategoryName
Description

Consultas

- Quando um utilizador necessita de alguma informação de uma base de dados, utiliza uma query.
- Uma query é um pedido do utilizador para a obtenção de dados ou de informações mediante certas condições.
- SQL é uma linguagem que cria querys e que permite ao utilizador especificar essas condições.

SELECT campos FROM tabela

- select, especifica a coluna que se quer consultar
- from, específica a tabela ou as tabelas que se quer consultar

Consulta simples

- Lista todas as colunas da tabela clientes

SELECT * FROM customers

- Lista uma coluna da tabela clientes

SELECT companyname FROM customers

- Lista diversas colunas da tabela clientes

SELECT companyname, address FROM customers

- Reordena as colunas

SELECT address, companyname FROM customers

Consulta com DISCTINT

- Distinct é uma palavra chave que elimina colunas duplicadas na saída

SELECT [DISTINCT] campos FROM tabelas

- Sem DISTINCT, lista todas as localidades da tabela clientes

SELECT city FROM customers

- Com DISTINCT, lista somente uma ocorrência para cada cidade

SELECT DISTINCT city FROM customers

Consulta com WHERE

- WHERE é uma cláusula que determina exatamente que linhas devem ser apresentadas.

SELECT campos FROM tabelas

WHERE condições_de_pesquisa

Consulta com WHERE

- Condições na cláusula WHERE
  - Operadores de comparação (=,>,<, >=, <=, !=);
  - Operadores lógicos (AND,OR)
  - Intervalos (BETWEEN e NOT BETWEEN);
  - Listas(IN e NOT IN);
  - Caracteres semelhantes (LIKE e NOT LIKE);
  - Valores desconhecidos (IS NULL e IS NOT NULL);

Consulta com WHERE

- NOT pode negar qualquer expressão booleana e palavras chave, tais como “LIKE”, “NULL”, “BETWEEN”, e “IN”.
- Na comparação de datas, “<” significa antes e “>” significa depois;
- Pode-se utilizar aspas simples ou duplas para dados CHAR, VARCHAR e DATETIME
- BETWEEN é uma palavra chave utilizada para especificar valores que se encontrem entre duas condições

Consulta com WHERE

- LIKE é uma palavra chave utilizada para selecionar linhas que contém campos que se assemelham numa determinada parte da string.
- É utilizado somente com CHAR, VARCHAR e DATETIME;
- Podem-se utilizar wildcards:
  - % (percentagem), qualquer string de zero ou mais caracteres
  - _(underscore), qualquer caracter;

Consulta com WHERE

- IN, NOT IN são palavras chave que permite que se selecione valores que se assemelham, ou não, com um valor específico de uma lista.
- AND une duas os mais condições. Retorna resultados somente quando todas condições são verdadeiras.
- OR liga duas ou mais condições. Retoma resultados quando qualquer uma das condições é verdadeira.

Consulta com comparação

- Com igual

SELECT * FROM customers WHERE city='Tokyo'

- Com diferente

SELECT * FROM customers WHERE city!='Tokyo'

- Com maior que

SELECT * FROM employees WHERE EmployeeID>5

- Com menor que

SELECT * FROM employees WHERE EmployeeID <=5

Consulta com (NOT) BETWEEN

- Com BETWEEN

SELECT * FROM employees WHERE EmployeeID BETWEEN 5 AND 8

- Com NOT BETWEEN

SELECT * FROM employees WHERE EmployeeID NOT BETWEEN 5 AND 8

Consulta com (NOT) LIKE

- Lista os nomes das localidades que comecem com a letra L

SELECT * FROM customers WHERE city LIKE ‘L%’

- Lista os nomes das localidades que não comecem com a letra L

SELECT * FROM customers WHERE city NOT LIKE ‘L%’

- Lista os ids de cliente cuja primeira letra seja A

SELECT * FROM customers WHERE city LIKE ‘A_’

Consulta com (NOT) IN

- Lista os clientes que têm, diversos critérios

SELECT * FROM customers WHERE city IN (‘Braga’, ‘Porto’)

SELECT * FROM customers WHERE city IN (SELECT DISTINCT city FROM customers)

- Lista os clientes que não correspondem aos critérios

SELECT * FROM customers WHERE city NOT IN (‘Tokyo’, ‘London’)

Valores nulos

- Um valor nulo implica em um valor desconhecido:
  - Um valor nulo não implica em zeros ou brancos, não existe valor designado;
  - is null (= null) pode ser usado para selecionar colunas que contém valores nulos.
- Um valor nulo nunca é igual a outro valor nulo.
- Algumas colunas são definidas para permitir valores nulos.
- Operações envolvendo nulos resultam em nulos.

Consulta com AND ou OR

- Lista os clientes que satisfaçam as duas condições

SELECT * FROM customers WHERE city =Tokyo' AND country=‘Japan’

- Lista os clientes que satisfaçam pelo menos uma condição

SELECT * FROM customers WHERE city =Tokyo’ OR country=Japan’

SELECT employeeid, firstname, lastname, city FROM employees WHERE firstname LIKE '%A%' AND city='London' OR city ='Seattle'

Renomear colunas

- Permite que o utilizador atribua outro nome a ser usado na saída do comando select ao invés do nome da coluna.

SELECT coluna AS novo_nome … FROM tabela AS novo_nome

SELECT firstname AS nome, lastname AS apelido

FROM employees AS e

Ordenar campos

- A clausula ORDER BY ordena os resultados da query (em ordem ascendente por default)
  - itens colocados no ORDER BY não precisam aparecer no SELECT;
  - utilizando a cláusula ORDER BY, os nulos são listados primeiro.

SELECT campos FROM tabelas [WHERE condições] [ORDER BY {coluna} [ASC/DESC]

Ordenar campos

SELECT contactname, address FROM customers WHERE city='Lisbon' ORDER BY contactname ASC

SELECT contactname, address FROM customers WHERE city='Lisbon' ORDER BY contactname DESC

SELECT contactname, address FROM customers WHERE city='Lisbon' ORDER BY contactname DESC, address ASC

Número de registos

- A cláusula TOP é utilizada para especificar o número de registos a serem retornados.
- É útil em grandes tabelas com milhares de registos.
- Retornar um grande número de registos pode afetar o desempenho da base de dados.
- Por norma está associado a uma ordenação.

```
SELECT TOP numero coluna
FROM tabela
WHERE condição;
```

```
SELECT TOP 3 * FROM Customers;
```

Rotinas Armazenadas

- **VIEW** é uma tabela virtual derivada do resultado de uma consulta SQL e serve como uma consulta armazenada que se pode referenciar e usar como se fosse uma tabela física.
- A finalidade das **VIEWS** no SQL é fornecer um mecanismo para simplificar consultas complexas, aprimorar a segurança dos dados e melhorar o desempenho da consulta.

```
SELECT *
FROM sys.views
WHERE type = 'V';
```

Associações de tabelas

- As associações de tabelas podem ser utilizadas para diversas finalidades, como converter em informação os dados encontrados em duas ou mais tabelas.
- Esse tipo de operação pode ser feito por meio das cláusulas WHERE e JOIN.
- Além disso, as tabelas podem ser combinadas por meio de uma condição ou um grupo de condições de junção.
- Por exemplo, podemos usar as chaves estrangeiras como condição para relacionar as tabelas.

Associações de tabelas

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

SELECT campos

FROM nome_primeira_tabela

TIPO_DE_ASSOCIAÇÃO nome_segunda_tabela

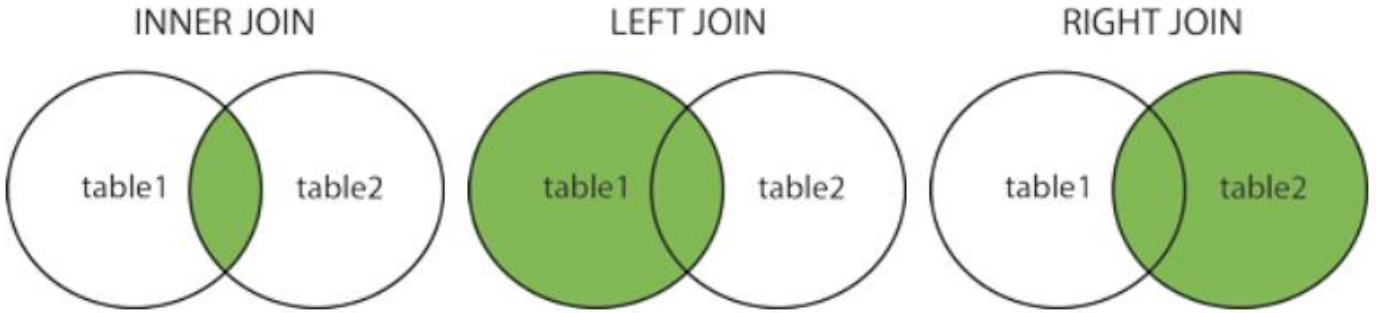
ON (condição_associação)

Associações de tabelas

- tipo_de_associação: Permite identificar uma das seguintes associações: LEFT JOIN, INNER JOIN e RIGHT JOIN
- condição_de_associação: Define um critério para avaliar duas linhas de dados que já estão associadas.
- A forma mais indicada para a especificação de associações é a cláusula FROM, que permite que as condições JOIN sejam identificadas em relação às condições de busca referenciadas na cláusula WHERE.
- As tabelas podem ser associadas de modo que sejam gerados não apenas dados relacionados entre elas, mas também dados não relacionados da tabela encontrada à esquerda ou à direita da cláusula JOIN.

Associações de tabelas

- Existem 3 tipos de associação:
  - (INNER) JOIN: Retorna registos que possuem valores correspondentes em ambas as tabelas
  - LEFT JOIN: Retorna todos os registos da tabela da esquerda e os registos correspondentes da tabela da direita
  - RIGHT JOIN: Retorna todos os registos da tabela da direita e os registos correspondentes da tabela da esquerda



Cláusula (INNER) JOIN

- A cláusula INNER JOIN seleciona registos que possuem valores correspondentes em ambas as tabelas.

```
SELECT campo
FROM tabela1
INNER JOIN tabela2
ON tabela1.campo = tabela2.campo
```

Cláusula LEFT JOIN

- A cláusula LEFT JOIN retorna todos os registos da tabela da esquerda (tabela1) e os registos correspondentes da tabela direita (tabela2).
- O resultado é 0 registos do lado direito, se não houver correspondência.

```
SELECT campo
FROM tabela1
LEFT JOIN tabela2
ON tabela1.campo = tabela2.campo
```

Cláusula RIGHT JOIN

- A cláusula RIGHT JOIN retorna todos os registos da tabela direita (tabela2) e os registos correspondentes da tabela esquerda (tabela1).
- O resultado é 0 registos do lado esquerdo, se não houver correspondência.

```
SELECT campo
FROM tabela1
RIGHT JOIN tabela2
ON tabela1.campo = tabela2.campo
```

Cláusula UNION

- Assim como um JOIN, uma operação UNION permite combinar dados provenientes de duas ou mais tabelas (ou da mesma tabela, com condições diferentes).
- Porém, em vez de combinar as colunas dessas tabelas, a UNION combina as linhas de dois ou mais conjuntos de resultados.
- Assim, enquanto um INNER JOIN é uma operação de intersecção entre conjuntos, o UNION é uma operação de soma de conjuntos.
- Cada instrução SELECT na clásula UNION deve ter o mesmo número de colunas, o mesmo tipo de dados e as colunas devem estar na mesma ordem.

Cláusula UNION

- O operador UNION seleciona apenas valores distintos por defeito. Para permitir valores duplicados, utiliza-se a cláusula UNION ALL.
- Assim, uma UNION une duas ou mais declarações SELECT. O resultado de cada SELECT deve possuir o mesmo número de colunas, e o tipo de dado de cada coluna correspondente deve ser compatível.

```
SELECT campo FROM tabela1 UNION [ALL]
SELECT campo FROM tabela2;
```

Cláusula UNION

- A principal vantagem do UNION é que ele permite combinar os resultados de múltiplas tabelas em uma única saída. Isso é útil quando se tem dados semelhantes em diferentes tabelas.
- Se uma empresa tem Clientes e Fornecedores armazenados separadamente, o UNION pode criar uma única lista de contatos.
- Pode-se unificar Pedidos de Compra e Pedidos de Venda numa única consulta.
- Mostrar todas as transações financeiras (vendas, compras, pagamentos, reembolsos) num único relatório.

SQL

Funções agregadoras

Operações aritméticas

- É possível realizar operações matemáticas simples nos valores de uma coluna e retornar os resultados numa outra coluna.
- Para isso usamos os operadores matemáticos comuns de soma, subtração, divisão e multiplicação, além dos operadores de divisão inteira e módulo (resto da divisão inteira)

```
SELECT firstname, lastname, salary, ROUND((salary+100),2)
FROM employees
```

```
SELECT quantity, unitprice, (quantity * unitprice) AS total
FROM order_details
```

Funções agregadoras

- Uma função de agregação processa um conjunto de valores contidos numa única coluna e retorna um único valor como resultado

Palavra chave da função	Valor calculado
SUM	Total
AVG	Valor médio (média)
MIN	Valor mínimo
MAX	Valor máximo
COUNT(*)	Número de linhas
COUNT	Número de (únicos)
([DISTINCT] nome_da_coluna)	Valores válidos

SELECT função(coluna(s)) FROM tabela

Funções agregadoras

- A função COUNT() retorna o número de linhas numa tabela.
- Permite contar todas as linhas ou apenas as linhas que correspondem a uma condição especificada.
  - Agregados ignoram valores nulos (exceto COUNT(*));
  - Só uma coluna é retornada (se a cláusula GROUP BY não for utilizada);
  - Não podem ser utilizadas na cláusula WHERE;
  - Retorna 0 se nenhuma linha correspondente for encontrada.

SELECT COUNT(expressão)

FROM tabelas

[WHERE ... condições]

Funções agregadoras

- A função COUNT() tem três formas:
- COUNT(*)
- COUNT(atributo)
- COUNT(DISTINCT atributo)

```
SELECT COUNT(*) AS conta FROM [Order Details];
```

```
SELECT COUNT(OrderID) AS conta FROM [Order Details];
```

```
SELECT COUNT(DISTINCT OrderID) AS conta FROM [Order Details];
```

Funções agregadoras

- COUNT (*)
  - A função COUNT (*) retorna o número de linhas num conjunto de resultados retornado por uma instrução SELECT.
  - O COUNT (*) retorna o número de linhas incluindo linhas duplicadas, NOT NULL e NULL.
- COUNT (campo)
  - O COUNT (campo) retorna o número de linhas que não contêm valores NULL como resultado da expressão.
- COUNT (DISTINCT expressão)
  - O COUNT (DISTINCT expressão) retorna o número de linhas distintas que não contêm valores NULL como resultado da expressão.

Funções agregadoras

- SUM retorna a soma de valores de uma coluna.

SELECT SUM(nome_da_coluna) FROM nome_da_tabela

- AVG retorna a média aritmética dos valores de uma coluna.

SELECT AVG(nome_da_coluna) FROM nome_da_tabela

- MIN retorna o valor mínimo de uma coluna.

SELECT MIN(nome_da_coluna) FROM nome_da_tabela

- MAX retorna o valor máximo de uma coluna.

SELECT MAX(nome_da_coluna) FROM nome_da_tabela

Cláusula GROUP BY

- A cláusula GROUP BY agrupa os registos em grupos de valores.
- Essa mudança faz com que se tenha mais que uma linha como resultado, pois o processamento será realizado uma vez sobre cada um desses grupos.

```
SELECT atributo1, max(atributo)
FROM tabela
GROUP BY atributo1
```

- Desta maneira os dados serão divididos através do atributo1

Cláusula HAVING

- Pode-se utilizar a cláusula HAVING em conjunto com GROUP BY para filtrar os resultado que serão submetidos a agregação.

```
SELECT atributo1, max(coluna) as maximo
FROM tabela
GROUP BY atributo1
HAVING max(coluna) > 10
```

- Desta maneira os dados serão filtrados através da clausula HAVING

SQL

Subqueries

Subqueries

- Uma subquery é uma consulta dentro de uma consulta.

Query principal

“Que empregados têm um salário maior que a média dos salários?”

Subquery

“Qual é a media dos salários?”

Subqueries

- A query de dentro é executada antes da query principal.
- O resultado da subquery é utilizado pela query principal como uma condição.
- Uma subquery podem ser feitas nas cláusulas SELECT, FROM, WHERE e HAVING.
- A parte interna da subquery pode retornar um único valor, uma coluna ou uma tabela derivada.

```
SELECT campos
FROM   tabela
WHERE  condição
                    (SELECT campos
                    FROM tabela);
```

Subqueries

- Tem que estar sempre entre parêntesis.
- Deve incluir uma cláusula SELECT e uma cláusula FROM.
- Pode incluir cláusulas WHERE, GROUP BY e HAVING.
- Não pode ter a cláusula ORDER BY.
- Utilizadas com funções de agregação retornam somente um valor.

Subqueries

SELECT campos, [subquery]

FROM [subquery]

JOIN [subquery]

[WHERE condição_de_pesquisa] =

[subquery]

[GROUP BY expressão_agregada]

[HAVING condição_de_pesquisa] =

[subquery]

Subquery no SELECT

- A subquery deve retornar um único valor. Se retornar uma coluna deverá ser ordenada e aplicada a cláusula TOP 1.
- Podem ser utilizadas funções de agregação, como SUM(), COUNT(), MAX(), MIN(), ou AVG(). Essas funções resumem um conjunto de valores num único valor.
- Pode fazer referência a colunas da tabela da consulta principal. Isso é comum e útil para cálculos que dependem de cada linha da consulta principal.
- Pode ser independente e não fazer referência à tabela da consulta principal.

Subquery no WHERE

- O uso mais básico para subqueries é filtrar dados na cláusula WHERE.
- Os operadores convencionais como = e >, são usados para ligar o comando que o contém e a subquery.
- O operador IN é provavelmente o mais usado, sendo frequentemente combinado com NOT para operar em todos os registos que não estão nos resultados da subquery.
- Os valores retornados devem ser compatíveis com o join da cláusula WHERE da consulta externa.

Subquery no FROM

- A subquery pode retornar várias linhas e colunas.
- Os resultados retornados são chamados tabela derivada.
- Uma tabela derivada é útil quando se quer trabalhar com um subconjunto de dados de uma ou mais tabelas sem a necessidade de criar uma view ou tabela temporária.
- A subquery tem que ter um alias.

Subqueries no HAVING

- As subqueries na cláusula HAVING permitem filtrar grupos resultantes de uma cláusula GROUP BY com base numa condição especificada na subquerie.
- É executada para cada grupo e retorna um valor que é usado na condição do HAVING para determinar quais grupos serão incluídos no resultado final.

SQL

Programação

Variáveis

- Em scripts SQL, é possível utilizar variáveis para armazenar valores durante a execução de uma sequência de comandos e utilizá-los em vez de literais.
- O T-SQL reconhece diferentes tipos de variáveis.
  - Variáveis de sistema contêm informações sobre o ambiente de execução e configurações do sistema. Fornecem acesso a informações úteis durante a execução de scripts ou consultas e são identificadas pelos símbolo @@
  - Variáveis definidas pelo utilizador, identificadas por um símbolo @ utilizado como um prefixo.

Variáveis de sistema

select @@VERSION - versão do SQL Server.

select @@SERVERNAME – nome do servidor

select @@LANGUAGE - idioma atualmente configurado no servidor

Variáveis definidas pelo utilizador

- Para declarar ou inicializar uma variável definida pelo utilizador, é necessário utilizar uma declaração DECLARE ou SET.
- É possível inicializar muitas variáveis ao mesmo tempo, separando cada declaração de atribuição com uma vírgula.

```
DECLARE @FirstVar int
SET @FirstVar=1
SELECT @FirstVar
```

Variáveis definidas pelo utilizador

- A duração de uma variável definida pelo utilizador dura enquanto a sessão estiver ativa, e é invisível para outras sessões. Uma vez encerrada a sessão, a variável desaparece.
- Os tipos de dados que se podem atribuir a uma variável definida pelo utilizador são os mesmos que vimos anteriormente.

Estruturas de repetição

- While - Cria um loop que executa repetidamente um bloco de código enquanto uma condição especificada for verdadeira.

```
WHILE Condição
BEGIN
        --Código SQL
        --Incremento
END
```

Estruturas de decisão

- If – Executa um bloco de código se uma determinada condição for verdadeira.

```
IF condição
BEGIN
        --Código
END
ELSE
BEGIN
        --Código
END
```

Estruturas de decisão

- Case – utilizado para realizar avaliações de múltiplas condições e executar diferentes blocos de código com base no valor de uma expressão. CASE simples:

```
CASE expressão
        WHEN valor1 THEN
                --Código
        WHEN valor2 THEN
                --Código
        ELSE
                --Código
END
```

Estruturas de decisão

- CASE de pesquisa:

```
CASE
        WHEN condição1 THEN
                --Código
        WHEN condição2 THEN
                --Código
        ELSE
                --Código
END
```

Rotinas Armazenadas

- Rotinas armazenadas são um conjunto de comandos SQL armazenados num SGBD.
- **STORED PROCEDURES** são um bloco de código SQL, armazenado no servidor, que não retorna valor.
- As STORED PROCEDURES são passivas, ou seja, para serem executadas é necessário uma aplicação pedir a sua execução.
- **FUNCTION** é um bloco SQL que retorna valores.
- Também as FUNCTION necessitam de ser executadas através de uma aplicação.

Rotinas Armazenadas

- **TRIGGERS** também são procedimentos SQL armazenados.
- A diferença é que os TRIGGERS são ativos, ou seja, são acionados automaticamente a partir de um evento que representa uma ação sobre uma tabela.
- Esses eventos estão relacionados com comandos INSERT, UPDATE ou DELETE.
- Sempre que um registo for movimentado numa tabela, o código do TRIGGER será executado, automaticamente.
- Um TRIGGER pode chamar uma STORED PROCEDURE, que por sua vez pode chamar outra STORED PROCEDURE.

Stored Procedures

- Stored Procedures são blocos de código SQL, armazenados no servidor, que não retornam valor.
- As Stored Procedures são passivas, ou seja, para serem executadas é necessário uma aplicação pedir a sua execução.
- Este procedimento armazenado poderá reduzir o tráfego na rede, melhorar a performance, criar mecanismos de segurança, etc.
- Dependendo da rotina a ser executada, isso pode requerer várias consultas e atualizações na base, o que acarreta um maior consumo de recursos pela aplicação.

Stored Procedures

- Para declarar uma variável local, pode-se utilizar a declaração DECLARE ou utilizá-la dentro de uma declaração PROCEDURE.
- Quando se declara uma variável local, opcionalmente, pode ser-lhe atribuído um valor por defeito. Se não se atribuir qualquer valor por defeito, a variável é inicializada com um valor NULL.
- Cada variável vive dentro de um âmbito, delimitado pelo bloco BEGIN ... END que contém a sua declaração.

Stored Procedures

```
CREATE [REPLACE] PROCEDURE nome
AS
BEGIN
   Bloco de códigos SQL
END;
```

Stored Procedures

```
CREATE [REPLACE] PROCEDURE nome
        @variavel1 tipo_dados,
        @variavel2 tipo_dados
AS
BEGIN
  Bloco de códigos SQL
END;
```

Stored Procedures

- A execução de uma Stored Procedure pode ser feita diretamente através do comando:

EXEC nome_procedure

EXEC nome_procedure @variavel1=‘valor’, @variavel2=‘valor’

Stored Procedures

- A execução de uma Stored Procedure pode ser feita diretamente através do comando:

Exec Nome;

EXEC Nome @Variavel1 = ‘’, @Variavel2 = ‘’

Triggers

- Utilização de TRIGGERS
  - Aplicação das regras do negócio;
  - Validação de dados de entrada;
  - Consulta de outros arquivos para fins de referência cruzada;
  - Acesso a funções do sistema;
  - Replicação de dados em arquivos diferentes para obtenção de consistência de dados;
  - Impor integridade de dados;
  - Consultar outras tabelas;
  - Incluir instruções Transact-SQL complexas.

Triggers

- TRIGGERS são procedimentos especiais executados automaticamente em resposta aos eventos de objeto da base de dados e do servidor.
- Um TRIGGER automatiza a execução de uma determinada ação ou conjunto de ações sempre que um evento específico ocorre.
- Um TRIGGER consiste em três partes principais: o evento que dispara o TRIGGER, o corpo do TRIGGER que contém as ações a serem executadas e a tabela a que o TRIGGER está vinculado.

Triggers

- TRIGGERS de linguagem de manipulação de dados (DML) que são “chamados” automaticamente em resposta a eventos de INSERT, UPDATE e DELETE nas tabelas.
- TRIGGERS de linguagem de definição de dados (DDL) que são acionados em resposta às instruções CREATE, ALTER e DROP.
- TRIGGERS de LOGON que disparam em resposta a eventos LOGON.
- O TRIGGER e a instrução que o aciona são tratados como uma única transação, que pode ser revertida dentro do TRIGGER.
- Se um erro grave for detetado (por exemplo, espaço em disco insuficiente), toda a transação será revertida automaticamente.

Triggers

CREATE TRIGGER nome

ON tabela

GATILHO (AFTER/INSTEAD OF)

COMANDO (INSERT/UPDATE/DELETE)

[DECLARE lista de variáveis]

AS

BEGIN

Bloco de códigos SQL

END

Triggers

- Nome: Nome do TRIGGER a ser criada/alterada;
- Gatilho: Momento de disparo do TRIGGER: depois (AFTER) ou em vez da execução do comando (INSTEAD OF);
- Comando: Comando que acionará o TRIGGER: INSERT, UPDATE ou DELETE;
- Bloco de códigos SQL: Código SQL que será executado.

Triggers

- AFTER
- Execução: Executa após a operação DML (INSERT, UPDATE, DELETE) ter sido concluída.
- Uso Comum: Operações que precisam ocorrer depois que os dados foram inseridos, atualizados ou apagados.
- Contexto Típico: Manutenção de auditoria ou log; Atualizações de agregados ou resumos; Verificações de integridade referencial que não são cobertas por chaves estrangeiras.

Triggers

- INSTEAD OF
- Execução: Substitui a operação DML, executando a lógica definida no TRIGGER em vez da operação DML original.
- Uso Comum: Quando se deseja substituir a operação DML padrão por outra lógica personalizada.
- Contexto Típico: Validações complexas antes de permitir a operação DML; Controlo de operações DML onde as operações padrão não são possíveis; Implementação de lógica condicional ou complexa que determina se e como a operação deve prosseguir.

Triggers

- A tabela especial "INSERTED" é uma tabela virtual disponível nos TRIGGERS.
- Contém as linhas que são inseridas ou modificadas durante uma operação de inserção ou atualização.
- Quando um TRIGGER é acionado em resposta a uma operação de inserção, a tabela "INSERTED" é preenchida com as linhas que estão a ser inseridas na tabela-alvo do TRIGGER. Cada linha na tabela "INSERTED" representa uma nova linha a ser inserida ou a linha modificada que será inserida como uma nova versão.

Triggers

- A tabela "INSERTED" possui a mesma estrutura da tabela-alvo do TRIGGER, ou seja, as mesmas colunas e tipos de dados. Isso permite que se aceda aos valores das colunas das linhas inseridas ou modificadas dentro do código do TRIGGER.
- É importante observar que a tabela "INSERTED" está disponível apenas dentro do contexto do TRIGGER e não pode ser acedida diretamente fora do TRIGGER.
- É exclusiva para cada execução do TRIGGER e contém apenas as linhas relacionadas à operação que acionou o TRIGGER.

Triggers

- A tabela "DELETED" é uma tabela temporária disponível em SQL Server que armazena os registos que foram excluídos ou modificados numa operação de exclusão ou atualização.
- Pode ser referenciada dentro do corpo do gatilho para aceder aos dados antigos antes da operação.
- Quando um gatilho é acionado em resposta a uma operação de exclusão ou atualização, a tabela "DELETED" é preenchida automaticamente com os registos afetados pela operação.
- A tabela possui a mesma estrutura (colunas) da tabela original na qual o gatilho está associado.

Triggers

- O comando COMMIT é usado para confirmar as alterações realizadas numa transação, mas somente num evento AFTER.
- Uma transação é uma sequência de uma ou mais instruções SQL que são executadas como uma unidade.
- O comando ROLLBACK é utilizado para desfazer as alterações feitas na transação que não foram confirmadas.
- Se ocorrer um erro ou se a transação não puder ser concluída com sucesso por qualquer motivo, pode ser utilizado o ROLLBACK para reverter todas as alterações feitas durante a transação.
- O ROLLBACK pode ser emitido explicitamente ou automaticamente pelo sistema em caso de erro.

Triggers

- A função RAISERROR é usada em T-SQL para gerar uma mensagem de erro personalizada durante a execução de uma instrução ou um bloco de código.
- Permite que se informem ou utilizadores ou aplicações sobre condições de erro ou eventos específicos.

RAISERROR (mensagem, gravidade, estado)

- mensagem: É a mensagem de erro personalizada em formato de texto.
- gravidade: É o nível de gravidade do erro. Pode variar de 0 a 25. Valores mais altos indicam erros mais graves.
- estado: É o estado do erro. Pode variar de 0 a 255. É útil para fornecer informações adicionais sobre o erro.

Triggers

- Gravidade:
- 0 a 10: Informações ou mensagens que não são erros. São utilizados para mensagens informativas.
- 11 a 16: Erros que podem ser corrigidos pelo utilizador. Normalmente não interrompem a execução do script ou do procedimento.
- 17 a 19: Erros mais graves que podem ter um impacto significativo no processamento. Geralmente requerem atenção imediata.
- 20 a 25: Erros graves que geralmente resultam no fim da execução do script. São problemas sérios que podem exigir intervenção administrativa.

Triggers

- Estado:
- O estado é um número personalizado que pode ser utilizado para fornecer informações adicionais sobre o erro.
- Ele varia de 0 a 255, e é opcional.
- O estado pode ser útil para categorizar ou identificar diferentes cenários de erro dentro do mesmo nível de gravidade.
- Por exemplo, pode-se utilizar diferentes estados para indicar diferentes condições ou origens de um mesmo tipo de erro. Isso ajuda na identificação precisa da causa do problema.

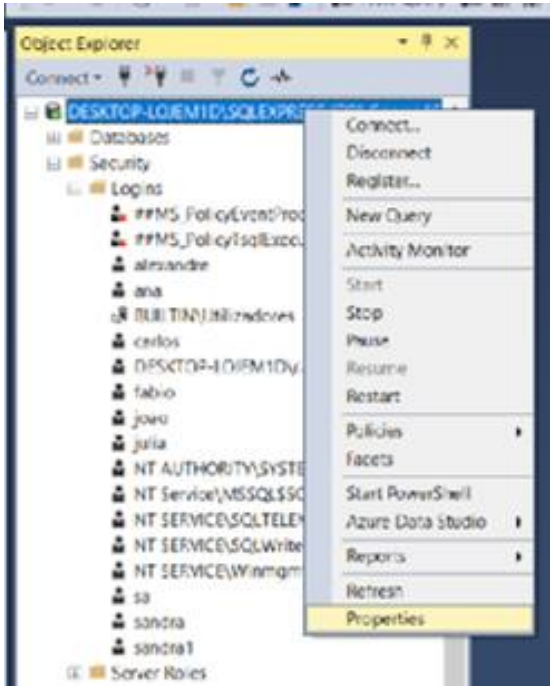
SQL

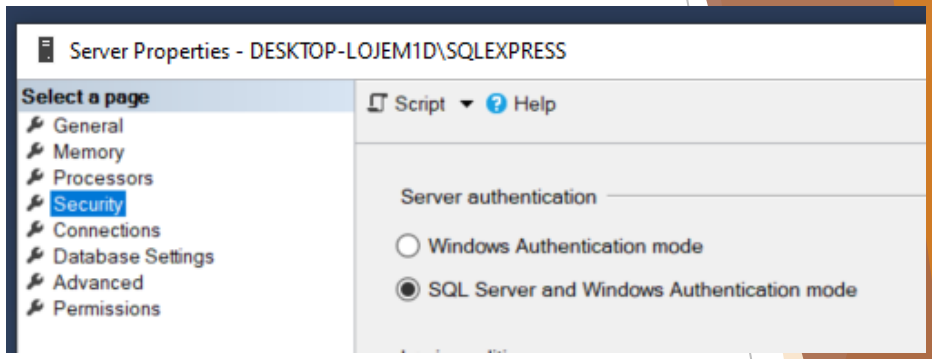
Utilizadores

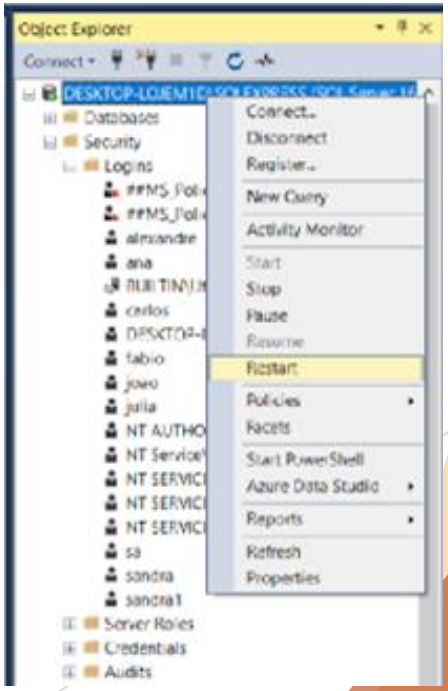
Utilizadores

- Uma das principais tarefas que devem ser realizadas para garantir a segurança de uma base de dados é a gestão de contas dos utilizadores.
- Isso inclui a criação de utilizadores, configuração de permissões de acesso (privilégios) às bases de dados e objetos relacionados, e a eventual exclusão de utilizadores do sistema.
- Todos os comandos inerentes a estas tarefas devem ser executados dentro do sistema do SQL Server, sendo necessário possuir privilégios adequados para isso.
- Geralmente, o utilizador sa do SQL é quem executa as ações de gestão de utilizadores, se não houver outro utilizador configurado com permissões administrativas.

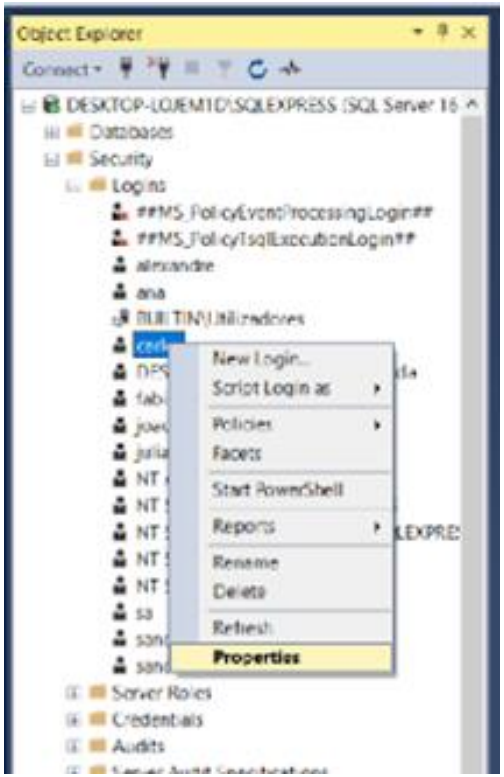
Fazer login com novo user

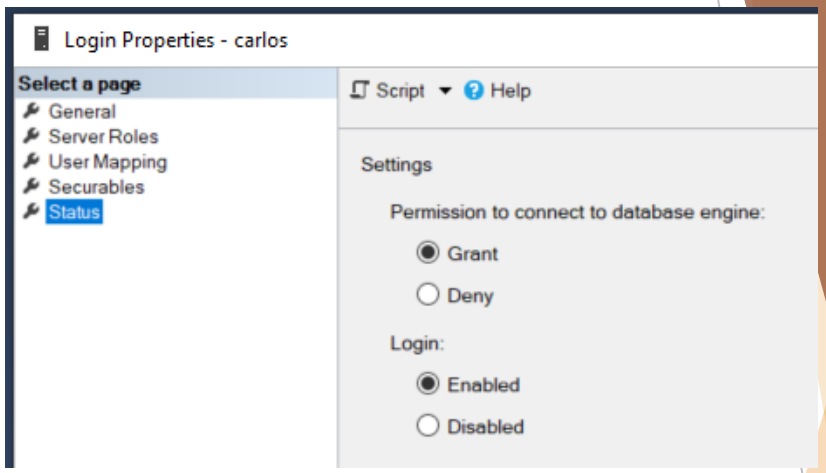


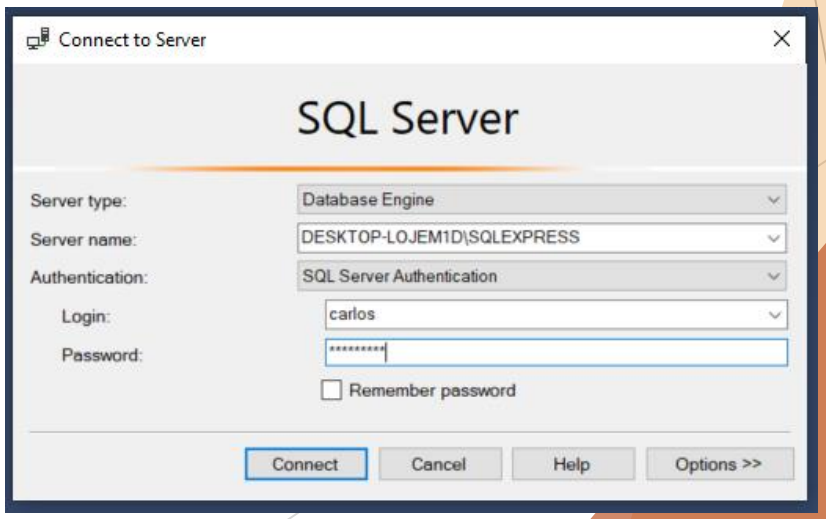




Fazer login com novo user







Utilizadores

- O sys é um esquema de sistema que contém tabelas e views sobre a configuração do servidor, objetos e segurança.
- É essencial para administrar o SQL Server, fazer auditoria, e obter detalhes de configuração.
- sys.database_principals - informações sobre a base de dados, indicando quem tem permissões e acessos.
- sys.database_users - informações sobre utilizadores
- sys.server_principals - informações sobre logins, roles de servidor
- sys.server_role_members - informações sobre membros de roles
- sys.database_permissions - informações sobre as permissões concedidas aos utilizadores, roles.

Utilizadores

- sys.sysobjects: informações sobre objetos
- sys.views - contém uma linha para cada view
- sys.procedures - contém uma linha para cada stored procedure
- sys.triggers - informações sobre triggers, como tipo, tabela de origem e o código de trigger.
- sys.sql_modules - contém o código de objetos como stored procedures, triggers e funções no banco de dados.

Utilizadores

- Efetuar uma consulta à tabela interna sys.database_principals para descobrir os utilizadores existentes.

```
SELECT d.name, d.create_date
FROM sys.database_principals d
WHERE type = 'S' AND authentication_type = 1;
```

type = 'S': utilizadores **criados** diretamente no SQL.

authentication_type = 1: utilizadores **autenticados** no SQL (em vez de autenticação do Windows)

Utilizadores

- No SQL Server, permissões são o mecanismo que define o que um utilizador pode ou não fazer dentro do sistema.
- Elas podem ser atribuídas a diferentes níveis — servidor, base de dados ou objeto (como tabelas e procedimentos).
- As roles são grupos de permissões predefinidas a nível do servidor ou da base de dados. Em vez de dar permissões individuais a cada login, adiciona-se o login a uma dessas roles e ele herda as permissões automaticamente.
- As permissões diretas são aquelas concedidas especificamente a um utilizador ou login, fora de um role.

Utilizadores

- Para efetuar a autenticação no servidor, é necessária a criação de um login:

CREATE LOGIN NomeLogin WITH PASSWORD = 'SenhaLogin’

- Após a criação do login, não há nenhum privilégio em nenhuma base de dados, somente no servidor.
- Para fazer a autenticação ao nível do domínio (a password é definida ao nível da AD):

CREATE LOGIN [DOMÍNIO\Utilizador] FROM WINDOWS

CHECK_POLICY = OFF (não aplica a política de complexidade da password)

Utilizadores

- Para testar com o novo login:

SELECT name FROM sys.databases;

- E dá resultados! acontece porque mesmo que o login não tenha permissões explícitas atribuídas, ele ainda pode ver os nomes das bases de dados (mas não entrar nelas) por causa do comportamento padrão de visibilidade.

Utilizadores

- Para alterar a senha a um utilizador, utiliza-se o comando ALTER LOGIN:

```
ALTER LOGIN NomeLogin WITH PASSWORD = 'NovaSenha’;
```

- Quando se quer testar sem ser no SSMS

```
EXECUTE AS USER = 'ana’;
select * from [order details]
SELECT USER_NAME();
revert
```

Utilizadores

- Para eliminar o login do servidor:

DROP LOGIN NomeLogin;

- Caso o login tenha sessões ativas, tem que se verificar o número da sessão

```
SELECT session_id, login_name, host_name, status
FROM sys.dm_exec_sessions
WHERE login_name = 'nomedologin’;
```

- Depois tem que se terminar essas sessões

KILL n_sessao

- Só então se pode apagar o login

Utilizadores

- Server roles são funções de servidor fixas que incluem conjuntos predefinidos de permissões e são atribuídas a logins.
- Sysadmin - Poder total sobre tudo no SQL Server.
- Serveradmin - Gerir configurações do servidor, iniciar/parar o serviço.
- Securityadmin - Gerir logins e permissões a nível de servidor.
- Processadmin - Terminar processos (ou seja, sessões / ligações).
- Setupadmin - Adicionar/remover linked servers e configurar replication.
- Dbcreator - Criar, alterar, restaurar e apagar bases de dados.
- Bulkadmin - Correr operações de importação em massa.

Utilizadores

- Adicionar um login a uma server role:

```
ALTER SERVER ROLE sysadmin ADD MEMBER carlos;
```

- Apagar um login de uma server role:

```
ALTER SERVER ROLE sysadmin DROP MEMBER carlos;
```

Utilizadores

- Lista, para cada função de servidor (como sysadmin, securityadmin, etc.), quais logins que pertencem a uma role.

```
SELECT r.name AS RoleName, m.name AS MemberName
FROM sys.server_role_members rm
JOIN sys.server_principals r ON rm.role_principal_id = r.principal_id
JOIN sys.server_principals m ON rm.member_principal_id = m.principal_id;
```

- Útil para auditorias de segurança ou controlo de acessos, ajudando a perceber se há logins com mais privilégios do que deviam ter.

Utilizadores

- Database Roles são atribuídas a nível de base de dados e gerem o que os utilizadores podem fazer dentro de uma base de dados específica.
- db_owner - controlo total sobre a base de dados.
- db_securityadmin - gere roles e permissões na base de dados.
- db_accessadmin - adiciona ou remove acessos.
- db_backupoperator - fazer backups da base de dados.
- db_ddladmin - criar, alterar ou eliminar objetos.
- db_datawriter - inserir, atualizar e eliminar dados.
- db_datareader - selecionar dados de todas as tabelas e vistas.

Utilizadores

- Adicionar um user a uma db role:

```
ALTER ROLE db_datareader ADD MEMBER carlos;
```

- Apagar um user de uma db role:

```
ALTER ROLE db_datareader DROP MEMBER carlos;
```

Utilizadores

- Lista, para cada função de servidor (como sysadmin, securityadmin, etc.), quais logins que pertencem a uma role.

```
SELECT dp.name AS RoleName, mp.name AS MemberName
FROM sys.database_role_members drm
JOIN sys.database_principals dp ON drm.role_principal_id = dp.principal_id
JOIN sys.database_principals mp ON drm.member_principal_id = mp.principal_id;
```

- Útil para auditorias de segurança ou controlo de acessos, ajudando a perceber se há logins com mais privilégios do que deviam ter.

Utilizadores

- Os privilégios diretos são atribuídos em três níveis diferentes:
  - Nível de servidor – O utilizador tem acesso ao servidor SQL conforme as permissões atribuídas.
  - Nível da base de dados – O utilizador tem acesso a uma base de dados específica, podendo interagir com os seus objetos dependendo da permissão.
  - Nível de objeto – O utilizador tem acesso a objetos individuais da base de dados, como tabelas, colunas, views ou stored procedures, com permissões específicas (ex: SELECT, INSERT, EXECUTE).

Utilizadores

- A declaração GRANT é utilizada para atribuir privilégios de acesso a um utilizador ou grupo, permitindo-lhe realizar operações específicas sobre objetos dentro da base de dados.

GRANT privilégio TO utilizador;

GRANT privilégio ON objeto TO utilizador;

GRANT privilégio ON objeto TO utilizador WITH GRANT OPTION

Utilizadores

- Tipos de privilégios para trabalhar ao nível do servidor:
  - CONTROL SERVER: controlo total sobre o servidor.
  - ALTER ANY LOGIN: alterar qualquer login.
  - SHUTDOWN: desliguar o servidor SQL.
  - VIEW SERVER STATE: ver processos, waits, estatísticas, etc.
  - ALTER SERVER STATE: ver/alterar estado do servidor, sessões, processos, etc.
  - CREATE ANY DATABASE: criar qualquer base de dados
  - VIEW ANY DATABASE: ver a lista de bases de dados
  - CONNECT SQL: ligar ao servidor SQL

- Estas permissões são permissões explícitas que se podem conceder diretamente a um login

Utilizadores

```
GRANT CONTROL SERVER TO NomeLogin;

GRANT ALTER ANY LOGIN TO NomeLogin;

GRANT SHUTDOWN TO NomeLogin;

GRANT VIEW SERVER STATE TO NomeLogin;

GRANT ALTER SERVER STATE TO NomeLogin;

GRANT CREATE ANY DATABASE TO NomeLogin;

GRANT VIEW ANY DATABASE TO NomeLogin;

GRANT CONNECT SQL TO NomeLogin;
```

Utilizadores

- Para verificar as permissões atribuídas ao login:

```
-- Verificar permissões do login
SELECT * FROM fn_my_permissions(NULL, 'SERVER');
```

Utilizadores

- No SQL Server, criar um login e um utilizador são passos distintos, mas interdependentes, que servem para configurar a segurança e o acesso às bases de dados.
- Login é uma entidade de segurança ao nível do servidor, responsável por autenticar o utilizador no SQL Server.
- Quando um login é criado, ele permite que o utilizador se ligue ao servidor SQL.
- Utilizador é uma entidade de segurança ao nível da base de dados.
- Permite que um login autenticado aceda a uma base de dados específica dentro do servidor SQL.

Utilizadores

- Depois de criar um login ao nível do servidor, é necessário criar um utilizador (user) ao nível da base de dados específica, para que este possa aceder aos objetos.
- O utilizador é criado com base no login previamente definido e liga o acesso do servidor à segurança interna da base de dados.

```
USE BaseDados;

CREATE USER NomeLogin FOR LOGIN NomeLogin;
```

Utilizadores

- A declaração DROP USER é utilizada para remover um utilizador (user) de uma base de dados específica.
- Este comando não remove o login do servidor, apenas o user daquela base de dados.

```
DROP USER NomeLogin;
```

Utilizadores

- Tipos de privilégios para trabalhar com a estrutura:
  - CONTROL ON DATABASE: concede todos os privilégios sobre um objeto de base de dados, incluindo a capacidade de modificar a estrutura do objeto (como tabelas, views, procedimentos, etc.), além de poder gerir a segurança e alterar a base de dados.
  - CREATE ANY DATABASE: criar novas bases de dados (só no master). Utilizadores com esta permissão podem criar novas bases de dados, mas não têm necessariamente permissão para criar objetos dentro de bases de dados específicas.

Utilizadores

- Tipos de privilégios para trabalhar com a estrutura:
  - ALTER ANY DATABASE: permite ao utilizador realizar alterações na estrutura das bases de dados (como renomear, alterar propriedades, etc.), mas também somente no nível do banco de dados master.
  - DROP DATABASE: Para apagar uma base de dados, o utilizador precisa de permissões administrativas. Normalmente, apenas utilizadores com funções de Administrador (como sysadmin ou db_owner) têm a capacidade de remover bases de dados.
  - Para apagar uma base de dados, o utilizador deve ter a permissão de ALTER DATABASE ao nível da base de dados e o privilégio para remover a base de dados. A permissão de DROP DATABASE normalmente é dada a administradores.

Utilizadores

- CREATE TABLE: criar novas tabelas.
- ALTER TABLE: realizar alterações numa tabela existente.
- DROP TABLE: excluir uma tabela existente.
- CREATE PROCEDURE: criar novas stored procedures.
- CREATE VIEW: criar novas views.

Utilizadores

- Tipos de privilégios para trabalhar com dados:
  - SELECT: consulta dos dados de uma tabela.
  - INSERT: insere novos registos numa tabela.
  - UPDATE: atualiza os registos existentes em uma tabela.
  - DELETE: exclui registos de uma tabela.
  - EXECUTE: executa procedimentos armazenados, funções e pacotes.

Utilizadores

```
GRANT SELECT ON Customers TO carlos;
GRANT INSERT, UPDATE, DELETE ON Orders TO carlos;
GRANT EXECUTE ON sp_GetOrderDetails TO carlos;
GRANT REFERENCES ON Orders(OrderID) TO carlos;
GRANT ALTER ON Products TO carlos;
```

Utilizadores

- Para retirar (revogar) privilégios dos utilizadores utiliza-se a declaração REVOKE ou DENY.

REVOKE lista_privilégios ON tabela FROM utilizador1;

DENY lista_privilégios ON tabela TO utilizador1;

- REVOKE é usado para remover permissões previamente concedidas, mas se concedidas por outra pessoa tem acesso a elas
- DENY é usado para negar explicitamente permissões, mesmo que alguém lhe tente dar acesso.

Roles

- As roles (funções de segurança) são utilizadas para agrupar permissões e conceder acesso a múltiplos utilizadores ou outras roles.
- São uma forma eficiente de gerir permissões de forma centralizada, evitando a necessidade de atribuí-las individualmente a cada utilizador.
- Ao conceder permissões a uma role, todos os seus membros herdam automaticamente essas permissões.
- Isto simplifica significativamente a administração da segurança, especialmente quando existem vários utilizadores com necessidades de acesso semelhantes.

Roles

- Selecionar a base de dados onde se pretende criar a ROLE
- Criar uma função chamada "role_name"

```
CREATE ROLE role_name;
```

- Conceder permissão SELECT para a função "role_name"

```
GRANT SELECT ON nome_tabela TO role_name;
```

- Adicionar o utilizador "user_name" à função "role_name"

```
ALTER ROLE role_name ADD MEMBER user_name;
```

SQL

Manutenção do servidor

Manutenção do servidor

- A manutenção do SQL Server pode ser dividida em várias categorias essenciais para garantir desempenho, segurança e disponibilidade:
  - Segurança, protegendo a base de dados contra acessos não autorizados e ataques, através da criação de utilizadores e regras.
  - Backup e Restore, garantindo a integridade e disponibilidade dos dados em caso de falha.
  - Agendamento e Automação, automatizando e verificando processos para reduzir o trabalho manual.

Manutenção do servidor

- Desempenho, otimizando a performance da base de dados e reduzir bottle necks.
- Espaço e Armazenamento, evitando que as bases de dados cresçam descontroladamente e prejudiquem o desempenho.
- Atualizações e Patching, mantendo o SQL Server atualizado e seguro.

SQL

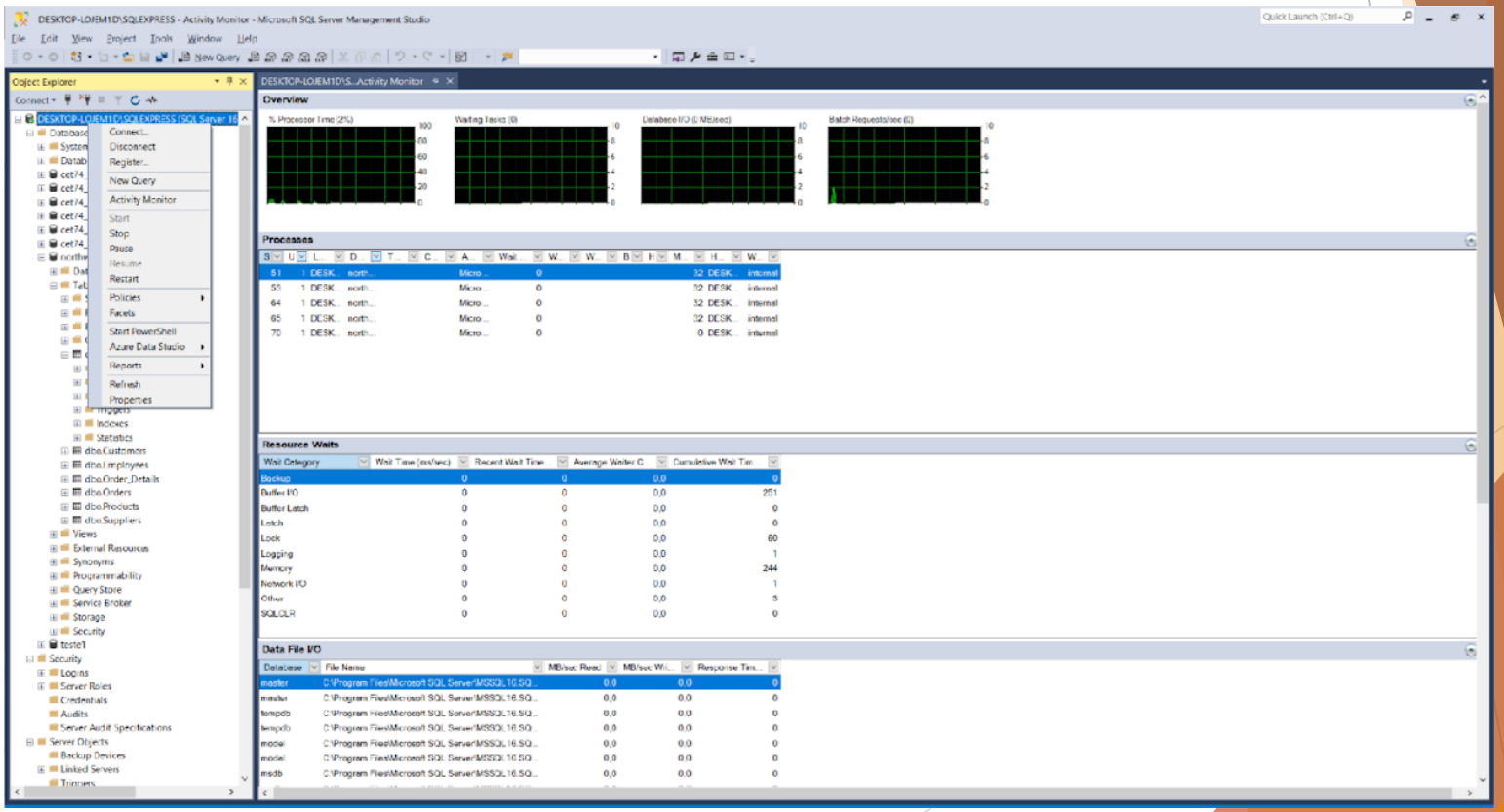
Backup e Restore

SQL

Administração

Activity monitor

- Activity Monitor fornece uma interface gráfica para monitorizar e analisar a atividade do servidor em tempo real, como sessões ativas, uso de CPU, I/O, bloqueios.
- Botão direito no servidor → Activity Monitor



Activity monitor

- Overview (Visão geral): Apresenta uma visão geral da atividade do servidor, incluindo informações sobre a utilização da CPU, E/S de disco, bloqueios e espera.
- Processes (Processos): Mostra os processos em execução no servidor, incluindo consultas, status, tempo de execução, recursos utilizados.
- Resource Waits (Esperas de Recursos): Exibe informações sobre os tipos de espera que os processos estão encontrando, permitindo que se identifique gargalos e problemas de desempenho.

Activity monitor

- Data File I/O (E/S de Arquivos de Dados): Mostra estatísticas de E/S para os ficheiros de dados do SQL Server, como leituras e gravações.
- Recent Expensive Queries (Consultas Morosas Recentes): Apresenta as consultas mais morosas em termos de utilização de recursos recentemente executadas no servidor.

Backups

- Para criar uma backup da base de dados, clica-se com o botão do lado direito na base de dados pretendida > Tasks > Backup.

C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup\

Backups

- Tipos de Backup no SQL Server
- Full Backup: cria uma cópia exata de todos os dados incluindo a parte do log de transações necessária para garantir a consistência da recuperação.
- Differential Backup)Definição: contém todas as alterações feitas desde o último backup completo. Ou seja, grava as diferenças entre o estado da base de dados no momento do último backup completo e o momento em que o backup diferencial é realizado.
- Transaction Log Backup: captura todas as transações registadas no log de transações desde o último backup de log.

Backups

- Como planear o Backup
- Um bom plano de backup deve combinar os diferentes tipos de backup para balancear desempenho, segurança e facilidade de recuperação. A combinação ideal geralmente envolve:
- Backup completo diário ou semanal.
- Backups diferenciais diários ou em intervalos mais curtos.
- Backups de log de transações em intervalos regulares (por exemplo, a cada hora ou conforme a carga de trabalho).
- Essa estratégia permite garantir que existam cópias completas e rápidas de recuperação (completas ou diferenciais) e que os logs de transações permitem uma recuperação detalhada até pontos específicos no tempo.

Agendamento de tarefas

- O SQL Server Agent é um componente que ajuda na automatização de tarefas e manutenção da base de dados. É uma ferramenta essencial para a administração, permitindo que se criem e agendem tarefas automatizadas, como backups, manutenção de índices, executar consultas, enviar alertas, executar jobs, e muito mais.
- SSMS → Object Explorer → SQL Server Agent

```
-- Mostrar as opções avançadas
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;

-- Habilitar o 'Agent XPs'
EXEC sp_configure 'Agent XPs', 1;
RECONFIGURE;
```

Agendamento de tarefas

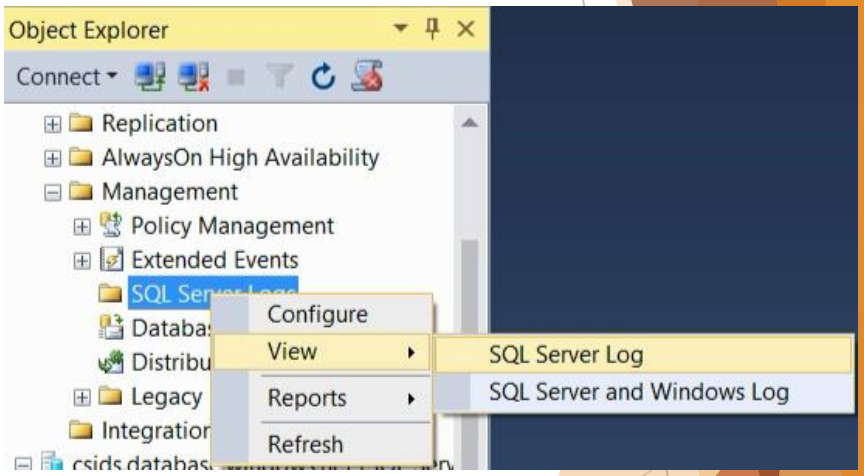
- Jobs: tarefa ou conjunto de tarefas que são executadas de forma automatizada. Essas tarefas podem incluir a execução de comandos Transact-SQL, manutenção da base de dados, execução de pacotes SSIS, ou outras atividades programadas, como agendar backups.
- Alerts: monitorar eventos e condições específicas e disparar alertas quando essas condições forem atendidas. Os alertas podem ser configurados para responder a uma série de situações, como erros de SQL Server ou problemas de desempenho, e podem enviar notificações por email ou executar ações específicas, como a falha de um backup.

Agendamento de tarefas

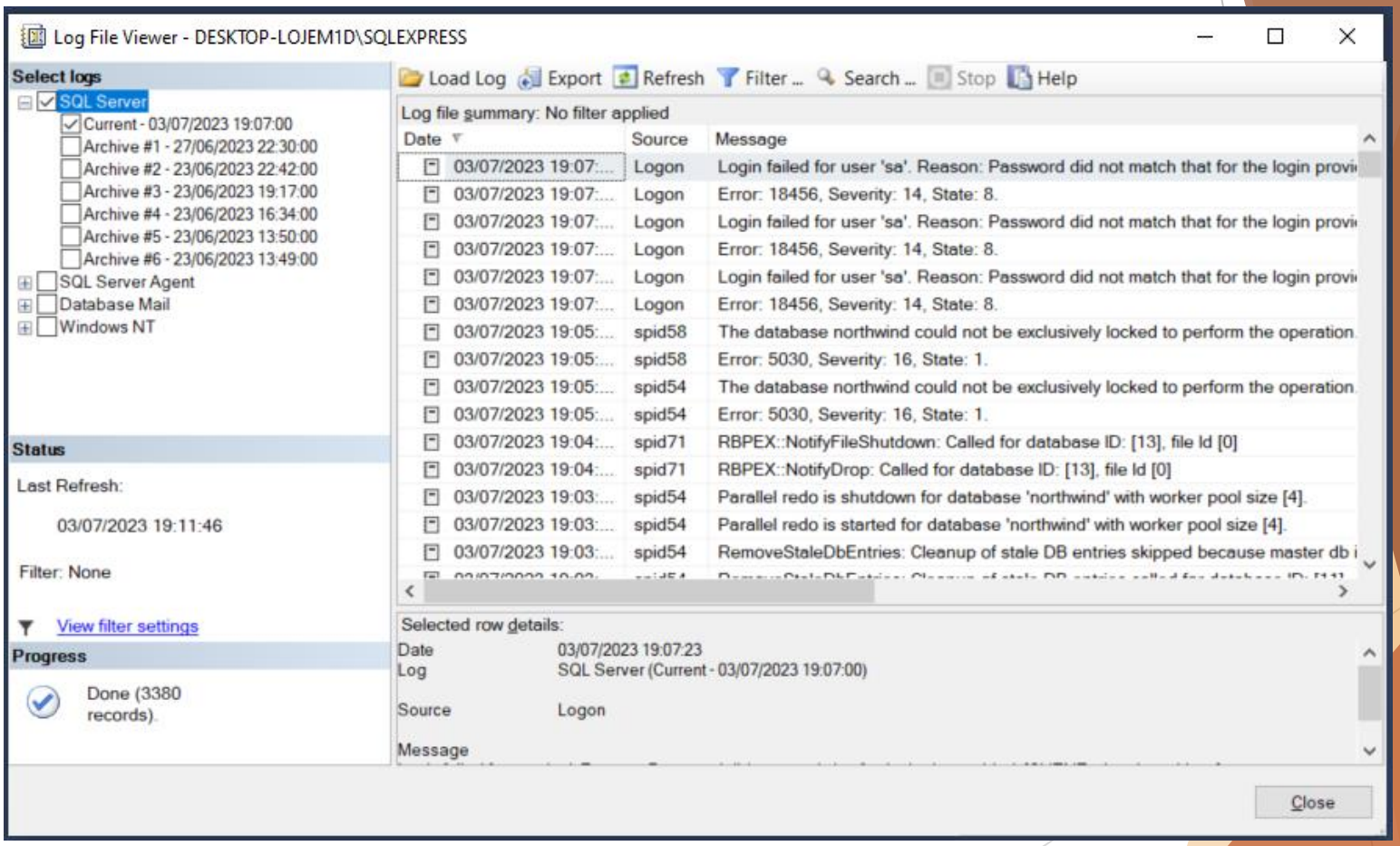
- Operators: utilizador ou grupo que pode receber alertas e notificações do SQL Server Agent. Esses operadores podem ser configurados para receber e-mails ou mensagens, dependendo das configurações do alerta.
- Proxies: usados para permitir que o SQL Server Agent execute tarefas em nome de um utilizador diferente (que tem permissões específicas). Isso é útil quando um job precisa de permissões adicionais ou se o trabalho precisa ser executado com privilégios mais elevados.

Ver logs do servidor

- O log de erros do SQL Server contém eventos definidos pelo utilizador e alguns eventos do sistema que podem ser usados para solução de problemas.

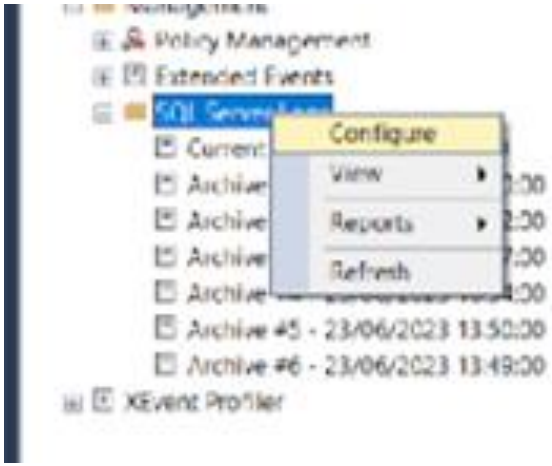


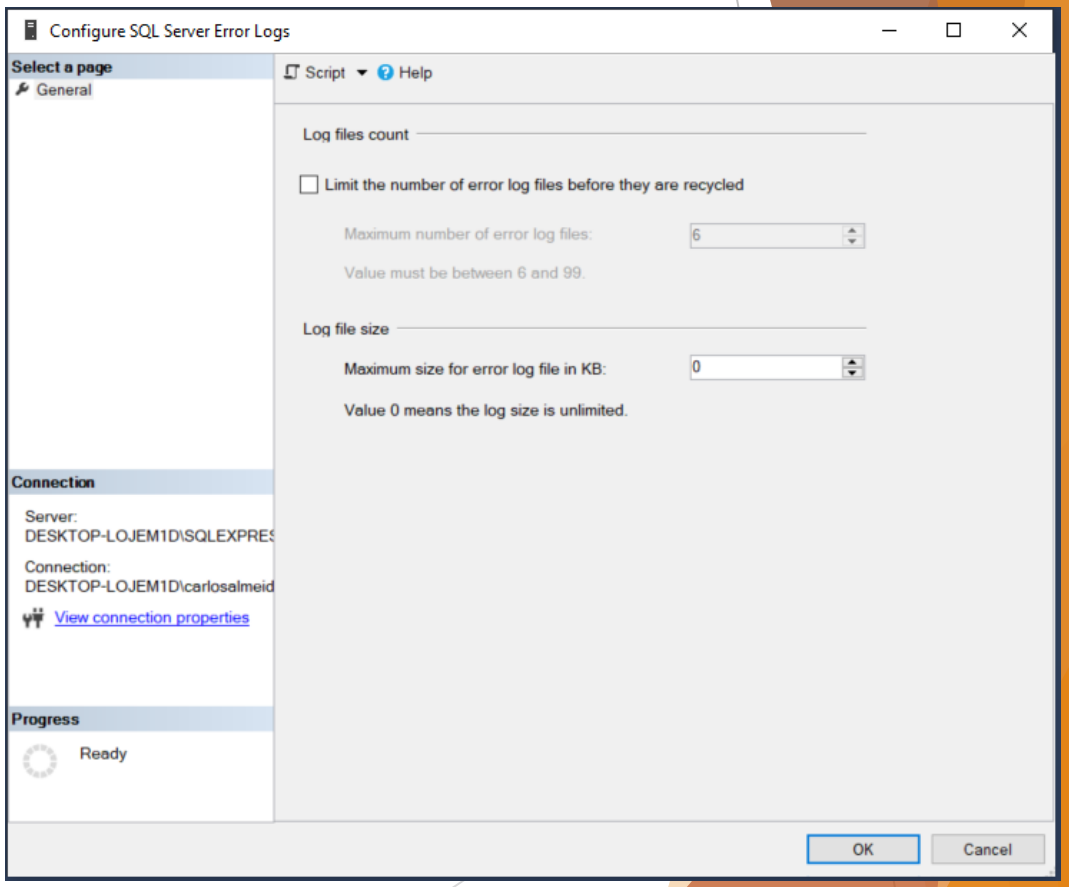
Ver logs do servidor



Ver logs do servidor

- Para configurar os logs do servidor





Ver logs do servidor

- Limitar o número dos arquivos de log de erros antes que eles sejam reciclados
- Tamanho máximo do arquivo de log de erros em KB