Baze de date
Sistemul SQL Server
2022 Express

Prefaţă
- Definiţii

SQL Server
- Instalarea unor componente ale sistemului de accesare a bazelor de date

SQL Server Management Studio: Crearea unei baze de date

SQL Server Management Studio: Crearea unui tabel şi plasarea sa într-o bază de date

Extragerea de informaţii dintr-o bază de date: comanda SELECT

Operaţiuni simple cu informaţia dintr-o bază de date: TOP, MIN, MAX

Gruparea unor informaţii: GROUP BY, SUM, AVG

Înregistrări similare: LIKE

Ordonarea înregistrărilor rezultate după o interogare SELECT: ORDER BY

Alăturarea înregistrărilor din mai multe tabele: JOIN

Adăugarea de înregistrări: INSERT – proiectarea unei baze de date dedicată bibliotecii personale

Modificarea unor înregistrări: UPDATE

Ştergerea unor înregistrări: DELETE

Crearea unei copii de rezervă a unei baze de date; restaurarea/recuperarea unei baze de date

Exportarea de informaţie dintr-o bază de date

Sintaxa pentru crearea unui tabel

Prefaţă

O definiţie simplă a Bazei De Date ar putea fi:

- un fişier sau un grup de fişiere în care se află informaţii de unul sau de mai multe feluri şi...
    - grupate în mulţimi distincte în acord cu una sau mai multe diverse caracteristici comune ale fiecărei secvenţe de informaţie şi care mulţimi...
        - pot fi reunite şi intersectate după aplicarea unuia sau mai multor criterii pe elementele conţinute care astfel pot fi *relaţionate*, ori *alăturate*

Astfel, se poate spune că prin Bază De Date se poate înţelege şi un fişier XML care poate susţine informaţii grupate astfel; şi, în general, că orice fişier al cărui conţinut este structurat asemănător se constituie într-o Bază De Date.

Totuşi, în mod uzual acest termen se foloseşte ca referindu-se la un ansamblu compus din:

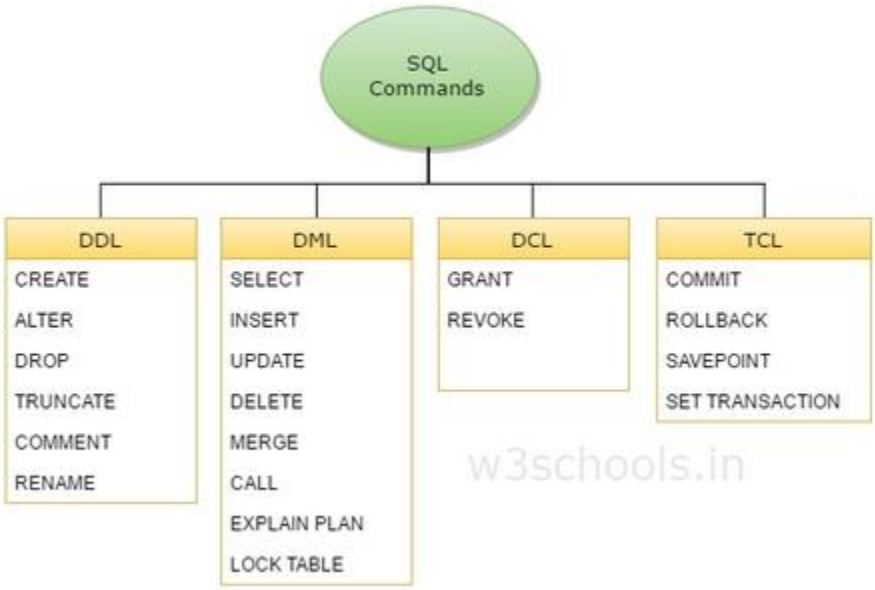
- fişierele înseşi care conţin informaţie
- un sistem *software* capabil de operaţiuni Input şi Output efectuate pe acele fişiere
- un set de instrucţiuni/comenzi şi o sintaxă alcătuind un limbaj prin care poate fi accesată informaţia din acele fişiere

Acel limbaj se numeşte
"Structured Query Language"
ori,
"**SQL**"

şi poate fi considerat drept compus la rându-i din entităţi ce pot fi privite ca fiind grupate de obicei în două categorii:

- "***Data Manipulation Language***", ori "***DML***".
    - Acestea sunt acele comenzi care permit ***accesarea informaţiilor*** dintr-o bază de date: selectare, modificare, adăugare, ştergere
- "***Data Definition Language***", ori "***DDL***".
    - Acestea sunt acele comenzi care permit ***modificarea structurii înseşi a bazei de date***

Imaginea de pe pagina următoare prezintă numele comenzilor SQL din aceste două categorii,
însă deoarece există mai multe variante/versiuni/dialecte ale limbajului SQL,
unele comenzi din imagine pot lipsi din dialectul folosit la un moment dat –
la fel cum alte comenzi pot lipsi din imagine:



Informaţia dintr-o bază de date este structurată arborescent, astfel:

- pe primul nivel se află baza de date însăşi
    - pe nivelul următor, în baza de date se află:
        - Tabele ("*Table*s")
            - pe nivelul următor, în fiecare Tabel în care este stocată informaţie de un anumit tip (de exemplu, detaliile comune despre un număr oarecare de persoane ar fi: numele, prenumele, adresa) se află informaţiile/datele propriu-zise structurate desigur tabelar în:
                - Înregistrări ori Linii ("*Record*s" ori "*Row*s"), alcătuite din ***secvenţele de informaţie care denotă caracteristicile comune ale înregistrărilor*** şi care sunt indicate de:
                - Câmpuri ori Coloane ("*Field*s" ori "*Column*s")

Deci, ***în mare***, o bază de date este alcătuită din:

- tabele cu linii şi coloane definind
    - înregistrări întinse pe mai multe Celule ale fiecărui tabel
        - Celule în care sunt secvenţele de informaţie comune înregistrărilor dintr-un tabel

O bază de date în sensul acesta al noţiunii nu este necesară de fiecare dată când se intenţionează manipularea unor informaţii grupate/structurate într-un anumit fel.

De obicei, bazele de date se folosesc atunci când se lucrează cu un număr de înregistrări de la ordinul sutelor până la cel al milioanelor şi dincolo de acesta.

Sistemul *software* capabil de operaţiuni Input (de intrare: întrebări) şi Output (de ieşire, răspunsuri) efectuate pe acele fişiere, menţionat mai sus şi care susţine toate bazele de date dintr-un anumit loc (prin aceasta înţelegând unul sau mai multe computere şi/sau medii de stocare ca SSD, hard-disc ş.a.) se numeşte generic "***Relational Database Management System***", ori "***RDBMS***" – "Sistem de Administrare a Bazelor de Date Relaţionale.

Un RDBMS este alcătuit din module *software* care oferă posibilitatea de accesare a informaţiilor din bazelor de date pe diverse sisteme de operare (de exemplu: *Windows*, *Linux*) şi folosindu-se de funcţionalitatea unor diverse platforme de rulare a programelor (de exemplu: *.NET*, *Mono*).

Pentru înlesnirea lucrului unui operator (utilizator) uman cu bazele de date administrate, un RDBMS poate avea asociat un ansamblu de programe care au interfaţă cu operatorul.

În această prezentare vor fi folosite:

- RDBMS-ul "*SQL Server*", ediţia *2022*, versiunea *Express*, construit de Microsoft
- asociat acestuia – programul "*SQL Server Management Studio*" care facilitează interacţiunea utilizatorului cu RDBMS-ul
- câteva comenzi *SQL* cu sintaxă simplă, comune mai multor dialecte ale SQL

SQL Server

Pentru a putea face operaţiuni cu o bază de date – de la a o crea până la a-i modifica structura – trebuie instalate aceste componente ale RDBMS-ului SQL Server:

- Serverul SQL în sine
  (versiunea 2022, specificată aici):
  Microsoft® SQL Server® 2022 Express
  poate fi preluat de la adresa pe Internet (la 2023.05.13):

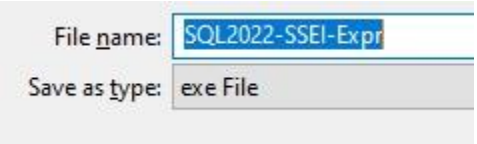
  https://www.microsoft.com/en-us/download/details.aspx?id=104781

- programul care facilitează lucrul cu SQL Server
- (versiunea 19.0.2, specificată aici):
  Microsoft SQL Server Management Studio (SSMS)
  poate fi preluat de la adresa pe Internet (la 2023.05.13):

  https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

Instalarea acestora este prezentată în imaginile următoare.

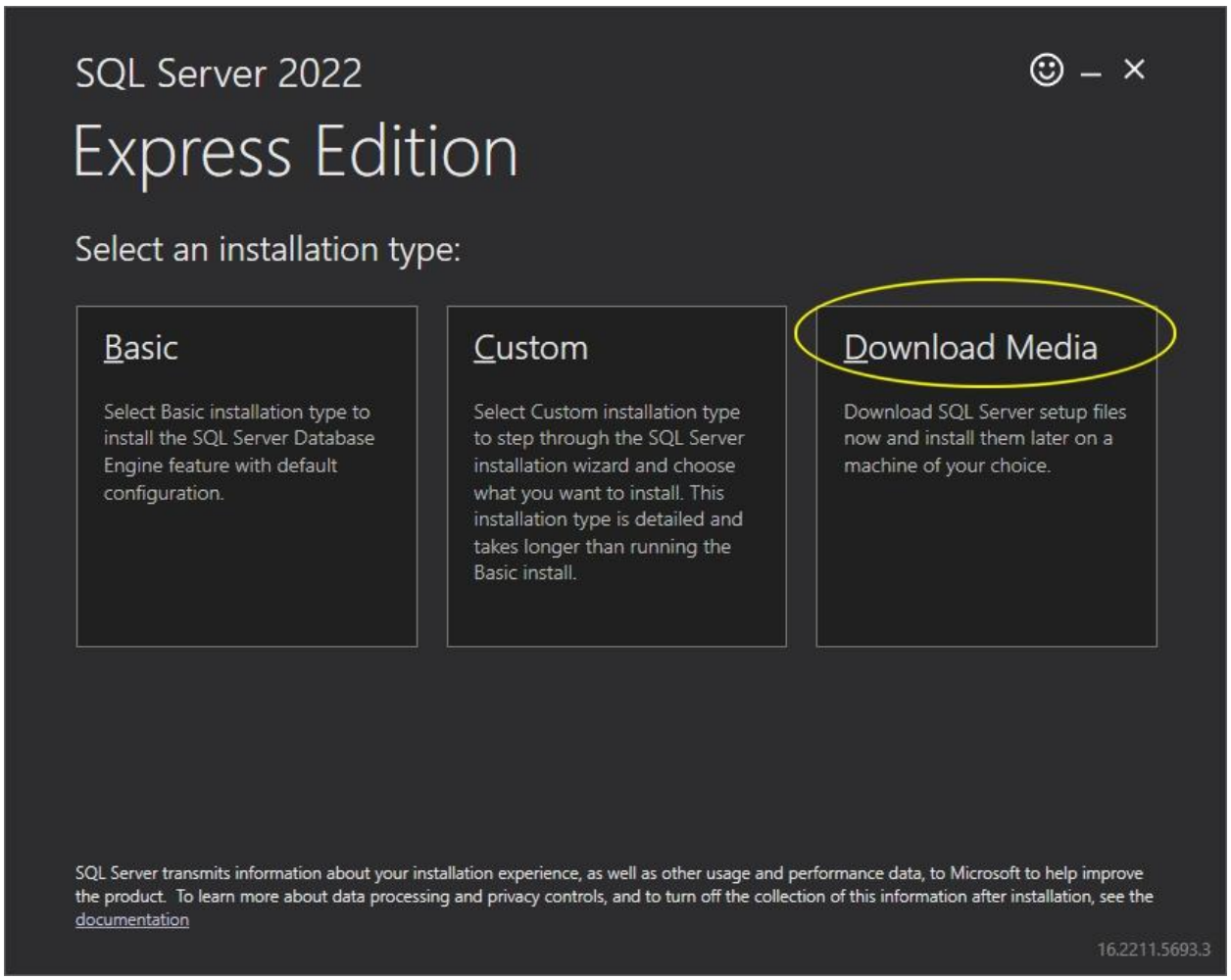
Întâi se instalează SQL Server:
De la adresa de mai sus se preia acest fişier executabil:



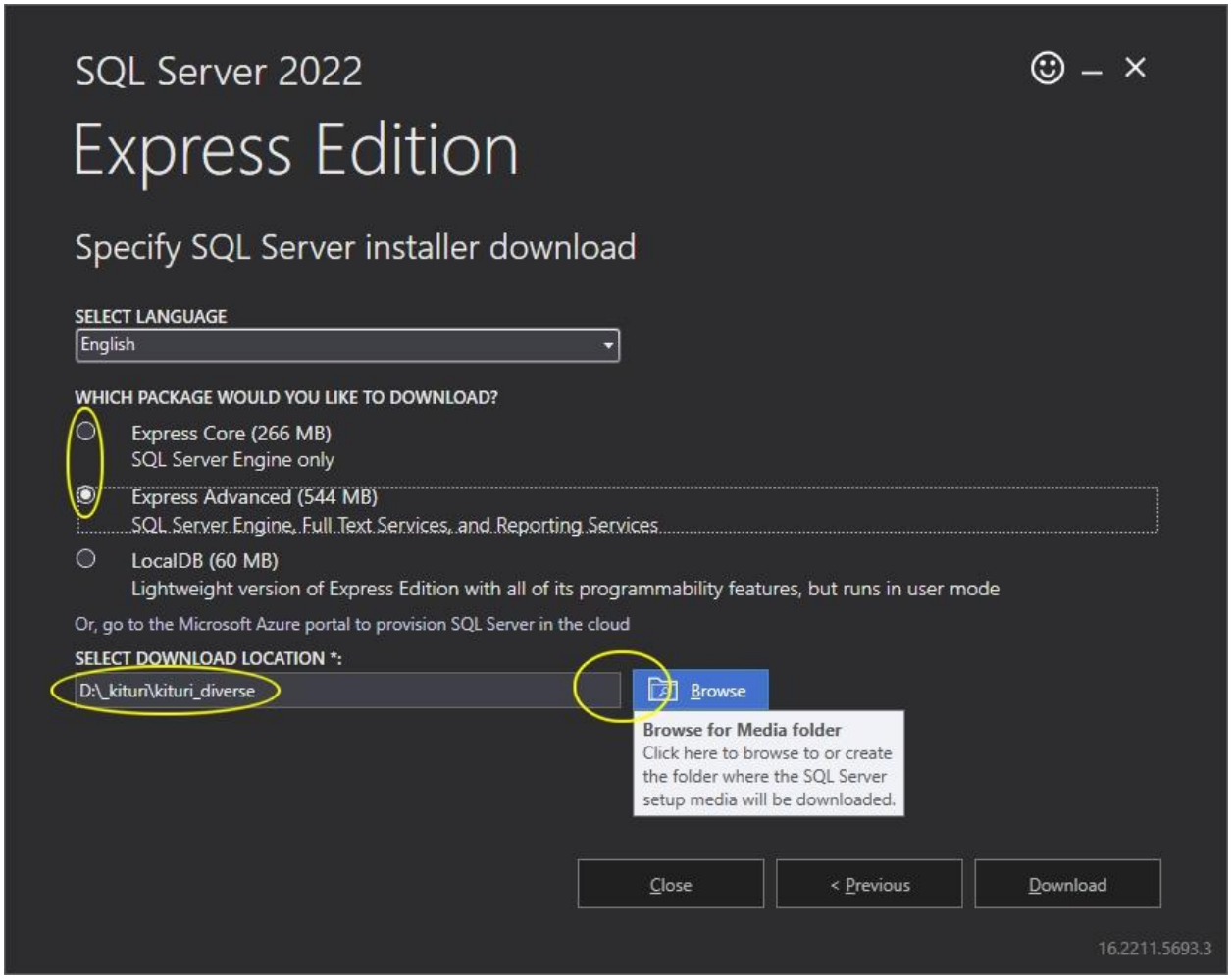
Acesta este programul care iniţiază instalarea SQL Server,
însă depinzând de accesul la Internet – anume,
va prelua la rându-i prin Internet componentele de instalat

Din opţiunile oferite de acest program de instalare (de *setup*) se alege "Download Media":

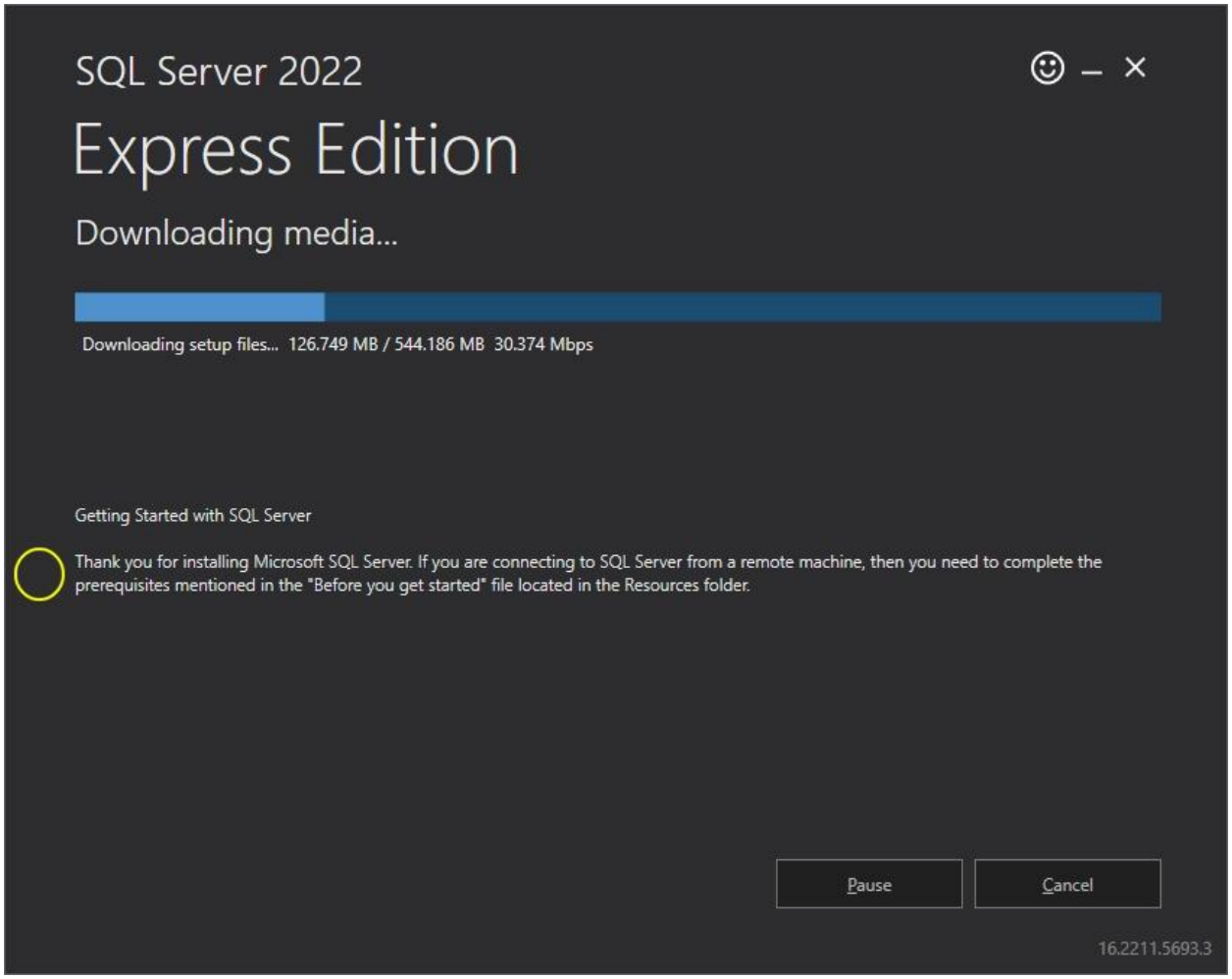
Aceasta înseamnă ca vrem să preluăm/descărcăm kitul de instalare a SQL Server
pentru ca, în eventualitatea când nu există acces la Internet,
să-l putem totuşi instala:

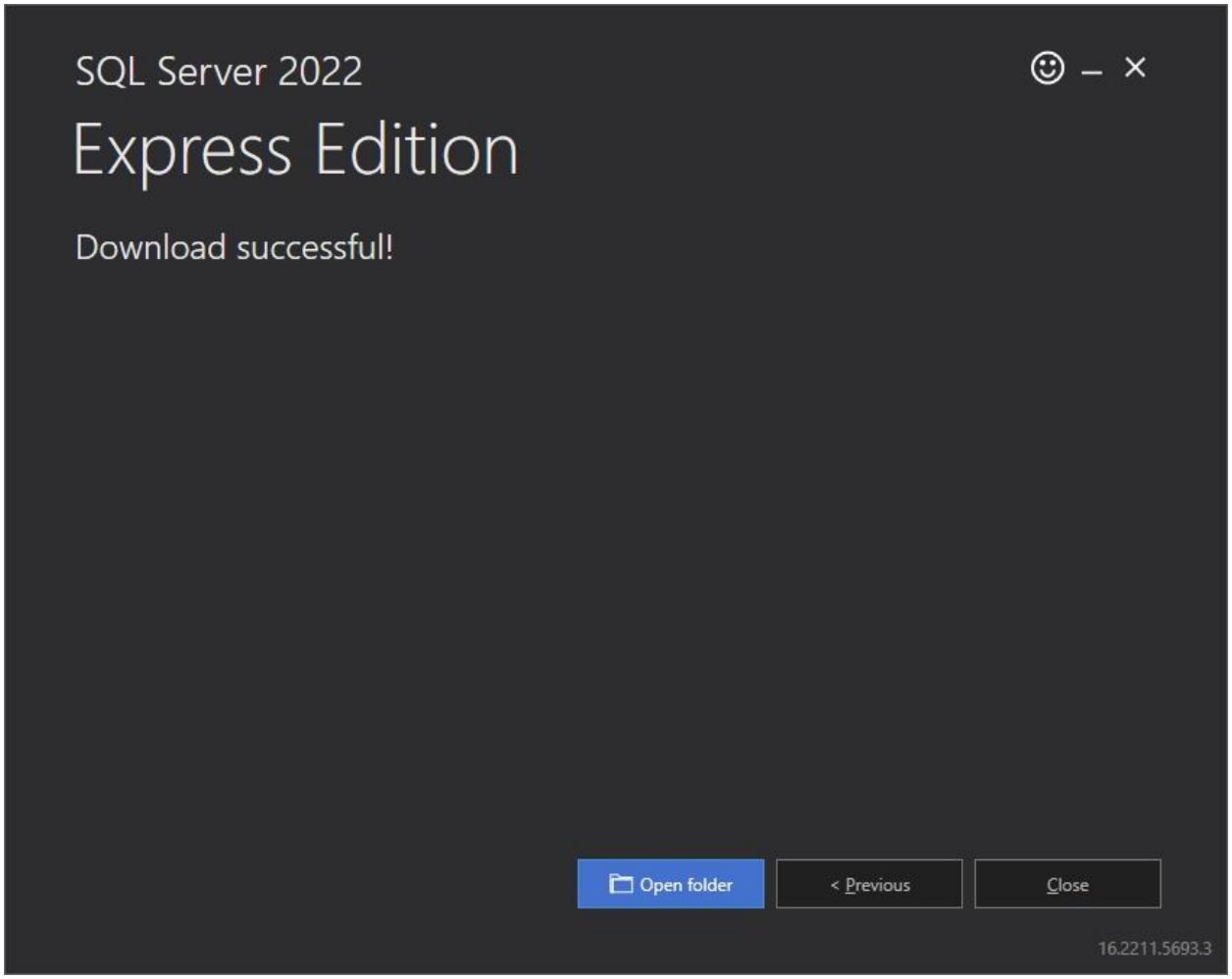


Alegem să instalăm un pachet relativ complex: "*Express Advanced*":
Specificăm destinaţia unde va fi plasat/descărcat kitul de instalare a acestuia:

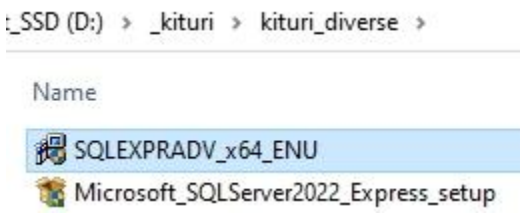


De acum aşteptăm până se termină descărcarea kitului cu acel pachet SQL Server:

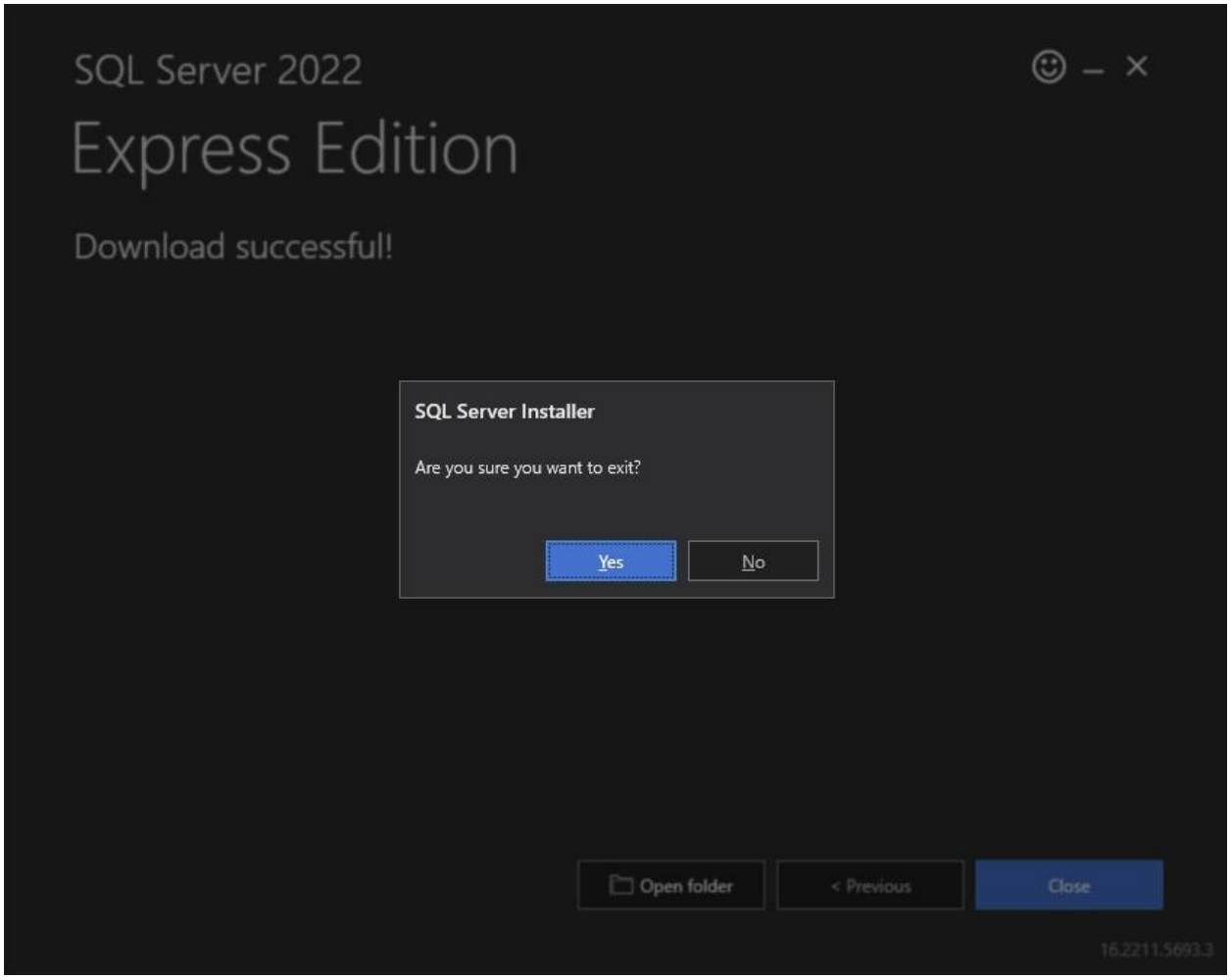

SQL Server 2022
Express Edition
Downloading media...
Downloading setup files... 126.749 MB / 544.186 MB 30.374 Mbps
Getting Started with SQL Server
Thank you for installing Microsoft SQL Server. If you are connecting to SQL Server from a remote machine, then you need to complete the prerequisites mentioned in the "Before you get started" file located in the Resources folder.
Pause
Cancel
16.2211.5693.3



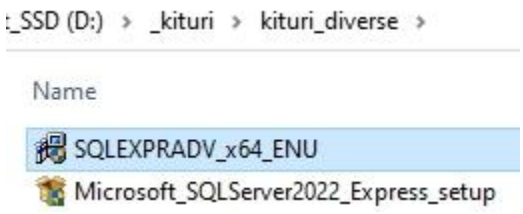
Fişierul cu kitul de instalare al SQL Server a fost plasat în directorul-destinaţie specificat mai devreme:



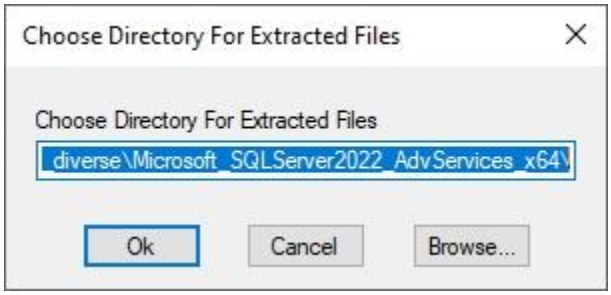
Întrucât kitul a fost descărcat în directorul/folderul ales înainte:
Închidem acest program, cu care am preluat kitul:
Apăsăm butonul marcat "Close"

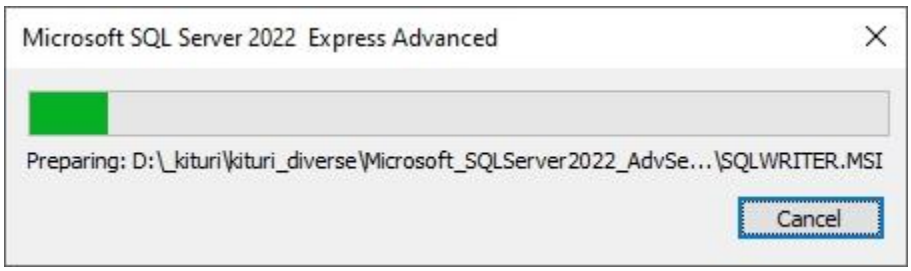


Acum putem lansa în execuţie programul-kitul tocmai descărcat:
Dublu-click sau Enter pe numele acelui fişier:

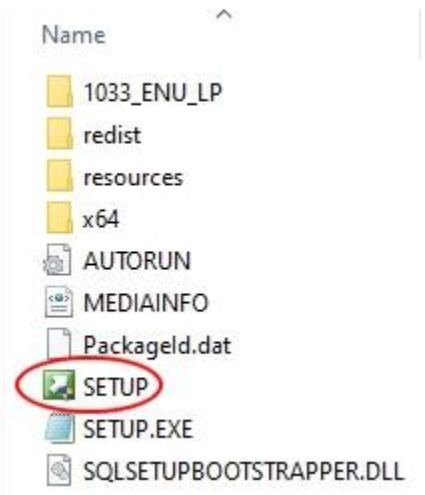


Apoi pe butonul OK al dialogului referitor la ***extragerea fişierelor din kitul de instalare*** a SQL Server:
Atenţie la directorul/folderul unde vor fi extrase fişierele din kit:
probabil că după instalarea SQL Server vom vrea să ştergem aceste componente ale kitlui:



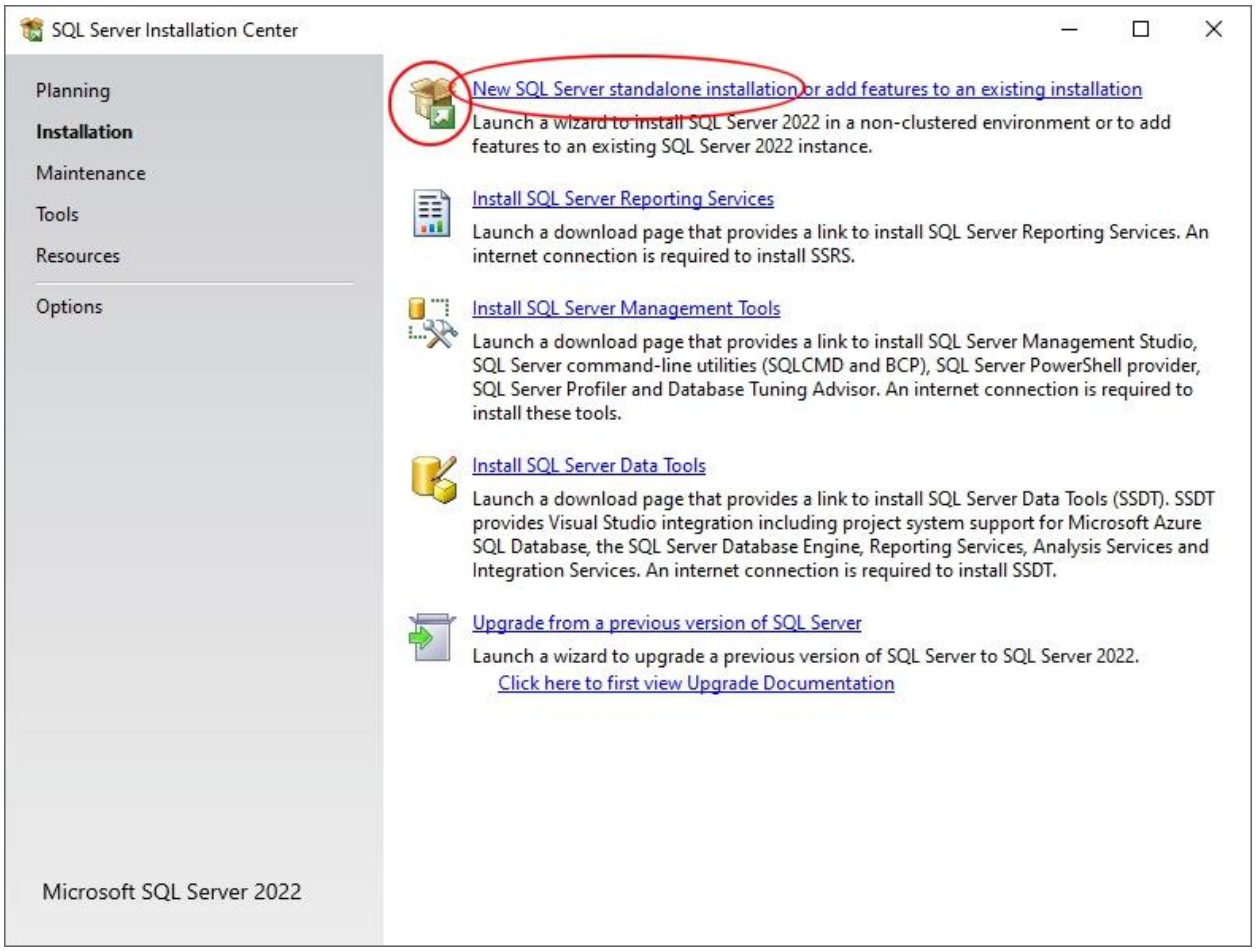


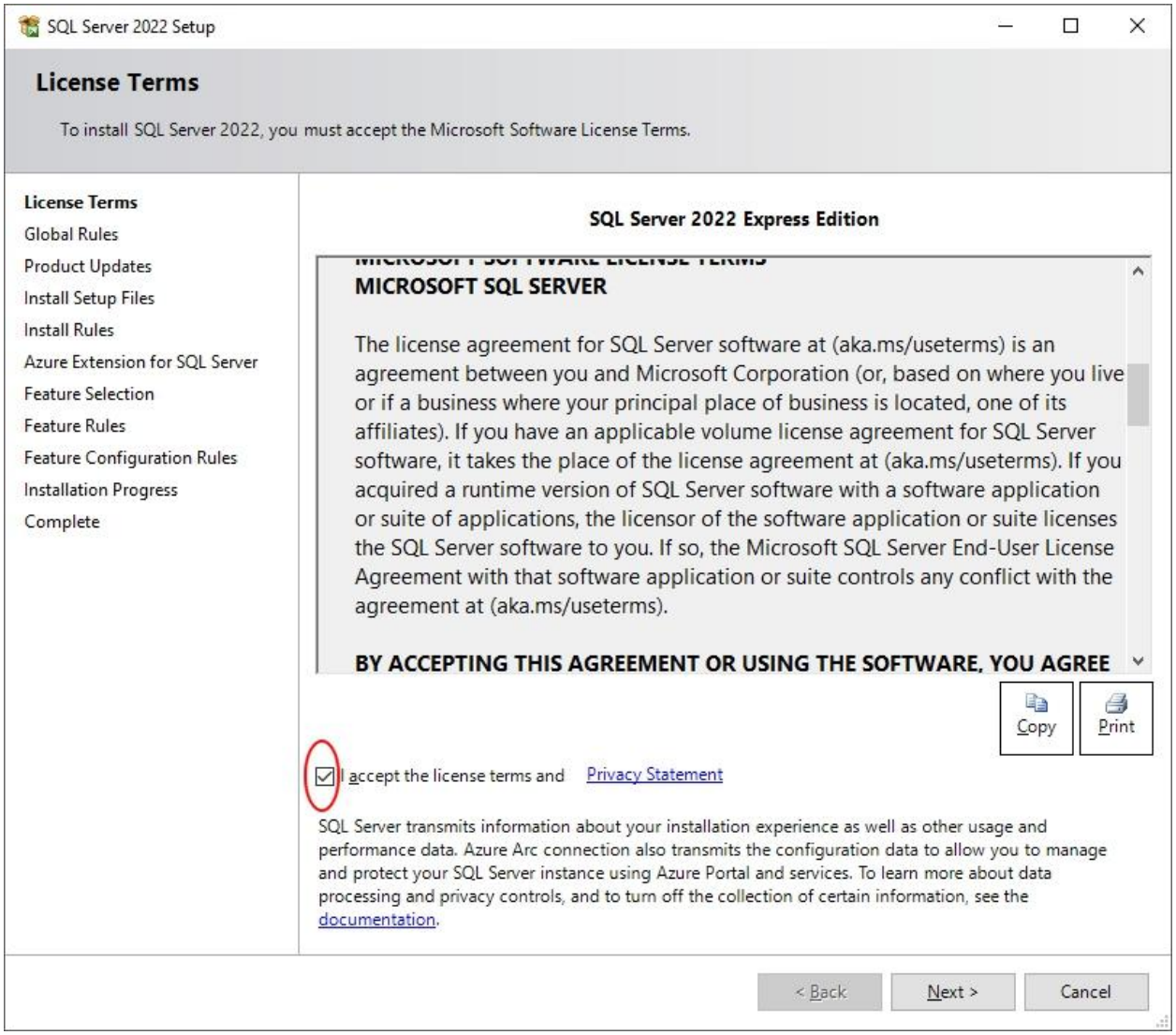
Mergem în acel director/folder unde identificăm ***programul de instalare propriu-zisă a SQL Server:***

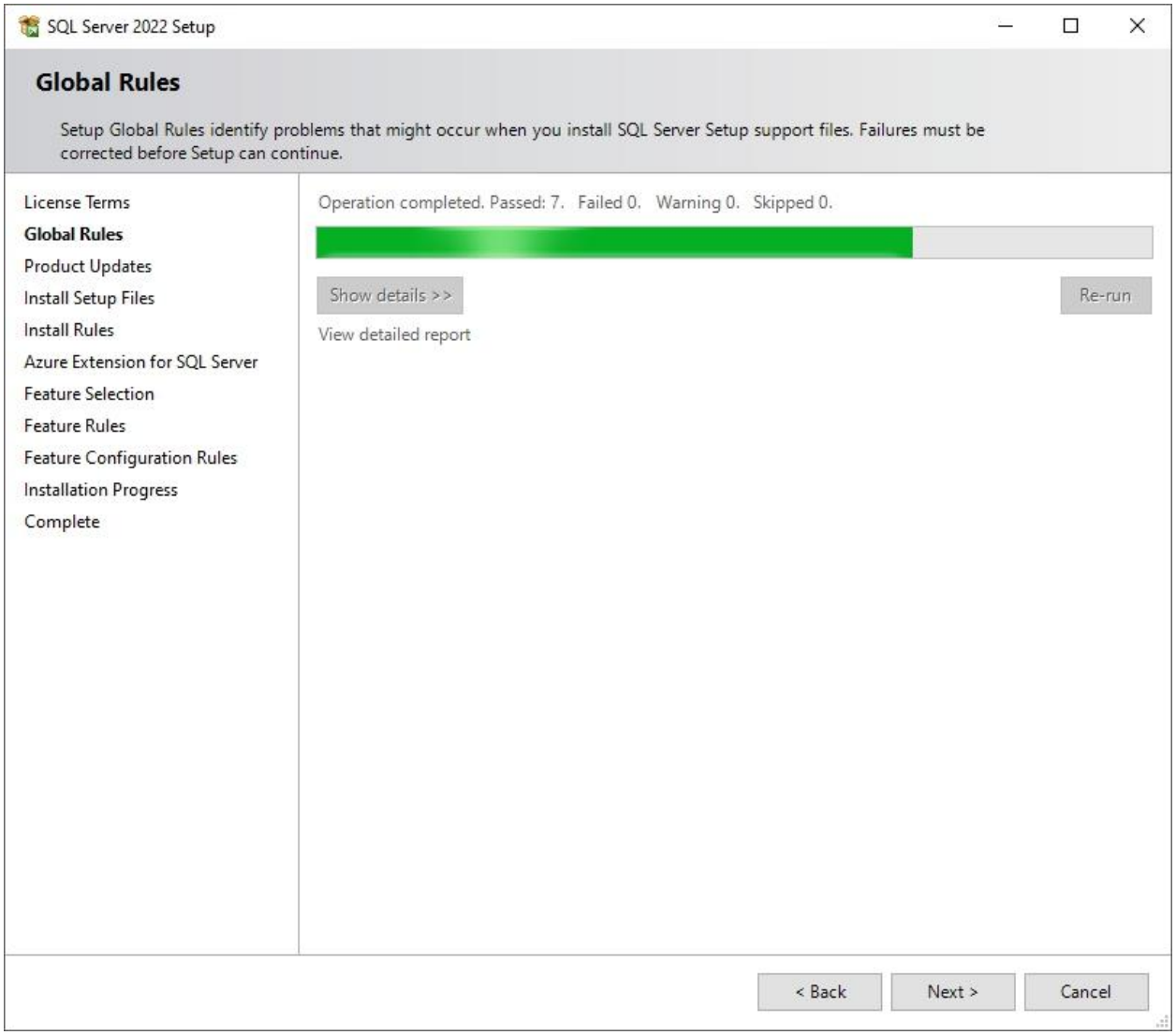


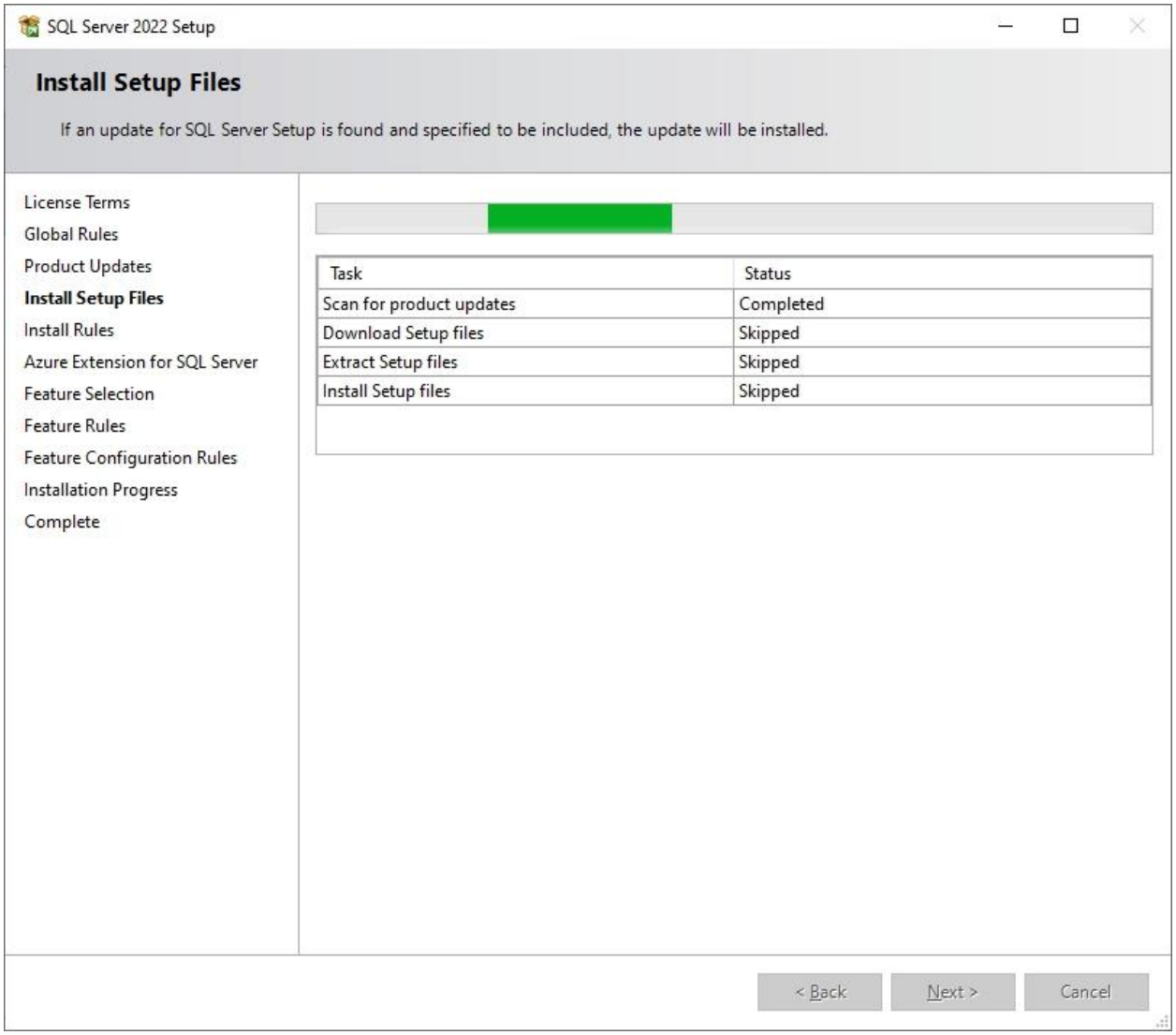
Imaginile următoare sunt elocvente pentru cele ce trebuie făcute în continuare:

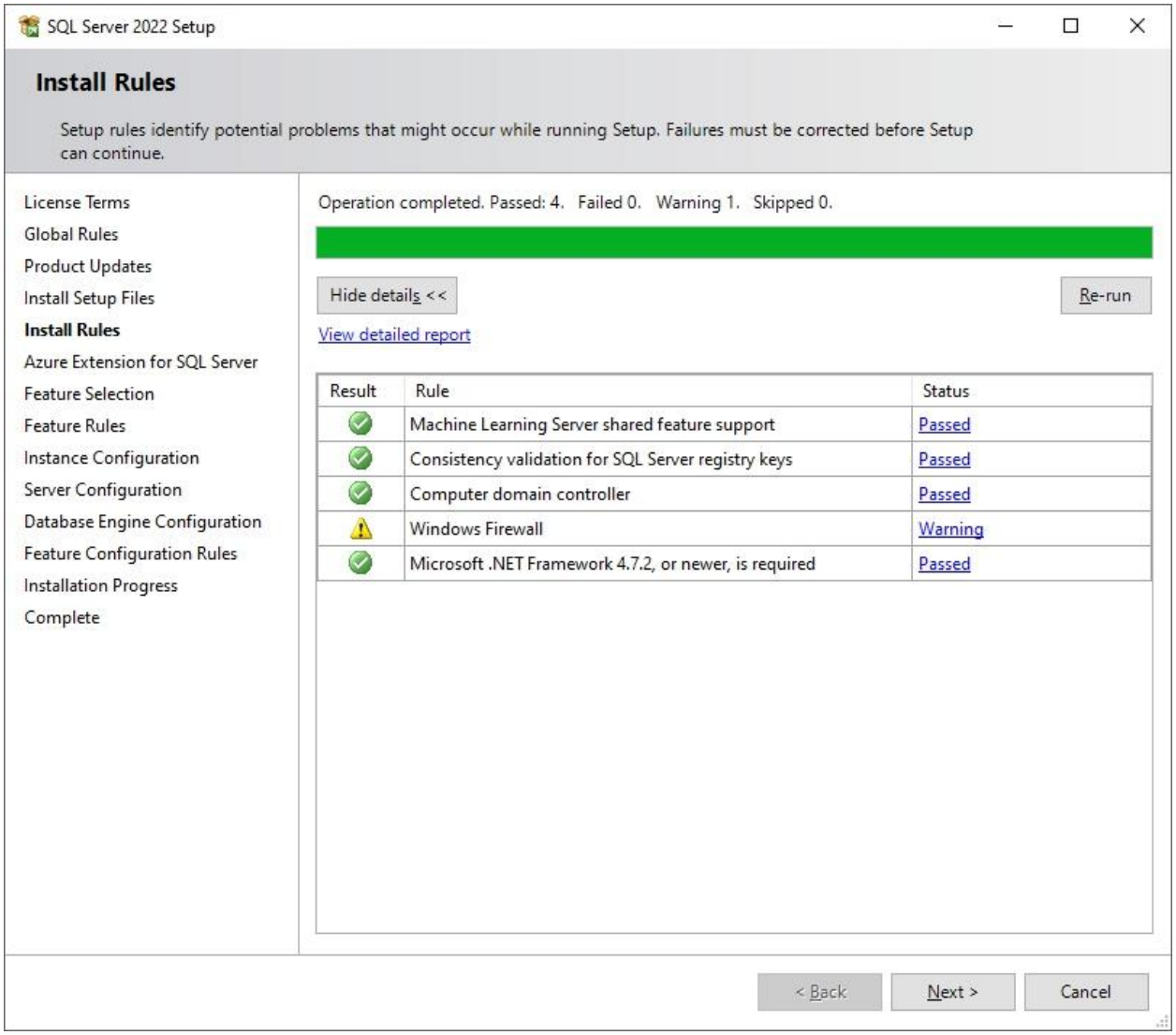




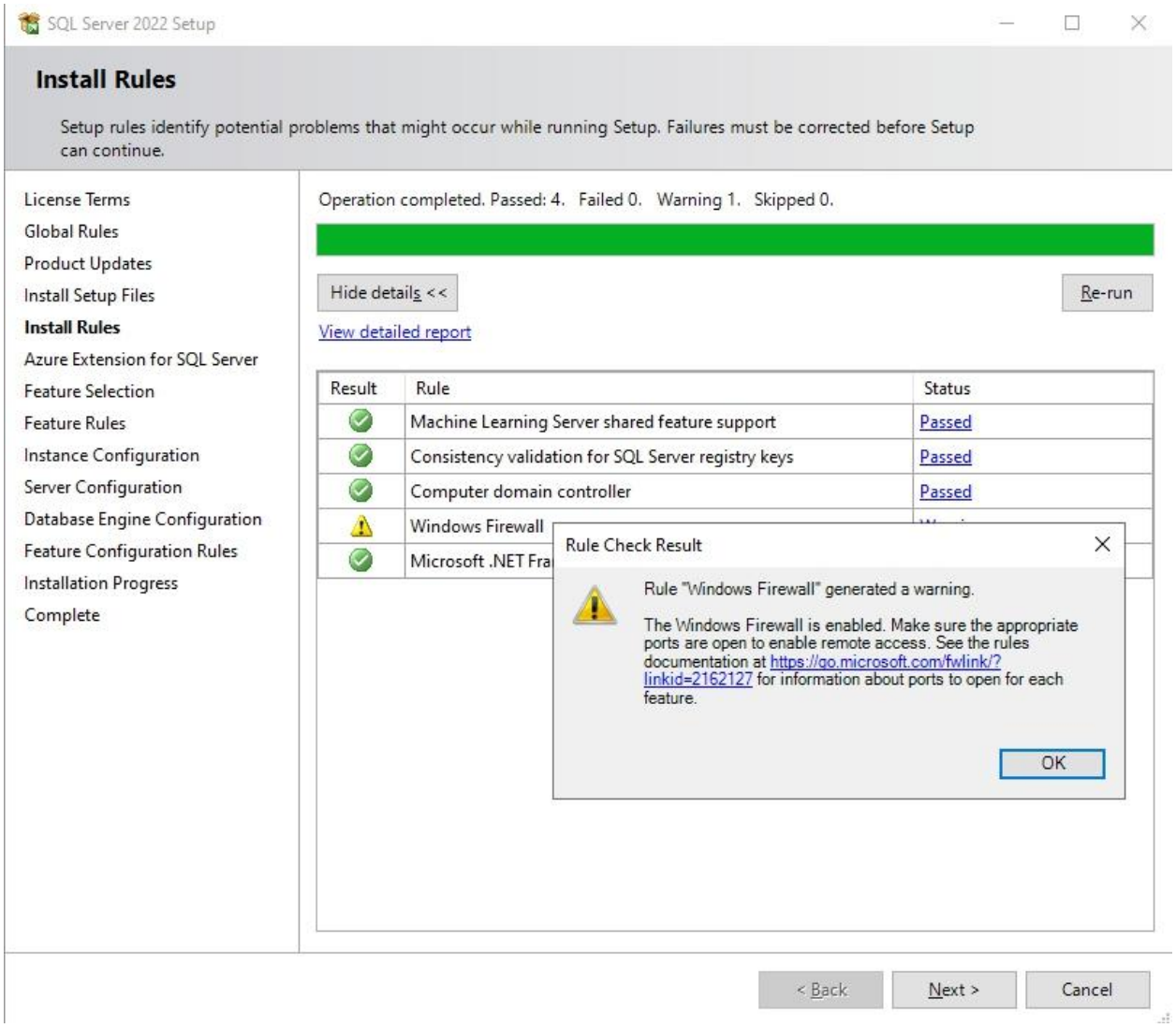

SQL Server 2022 Setup
License Terms
To install SQL Server 2022, you must accept the Microsoft Software License Terms.
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Azure Extension for SQL Server
Feature Selection
Feature Rules
Feature Configuration Rules
Installation Progress
Complete
SQL Server 2022 Express Edition
MICROSOFT SQL SERVER
The license agreement for SQL Server software at (aka.ms/useterms) is an agreement between you and Microsoft Corporation (or, based on where you live or if a business where your principal place of business is located, one of its affiliates). If you have an applicable volume license agreement for SQL Server software, it takes the place of the license agreement at (aka.ms/useterms). If you acquired a runtime version of SQL Server software with a software application or suite of applications, the licensor of the software application or suite licenses the SQL Server software to you. If so, the Microsoft SQL Server End-User License Agreement with that software application or suite controls any conflict with the agreement at (aka.ms/useterms).
BY ACCEPTING THIS AGREEMENT OR USING THE SOFTWARE, YOU AGREE
Copy
Print
I accept the license terms and
Privacy Statement
SQL Server transmits information about your installation experience as well as other usage and performance data. Azure Arc connection also transmits the configuration data to allow you to manage and protect your SQL Server instance using Azure Portal and services. To learn more about data processing and privacy controls, and to turn off the collection of certain information, see the documentation.
< Back
Next >
Cancel

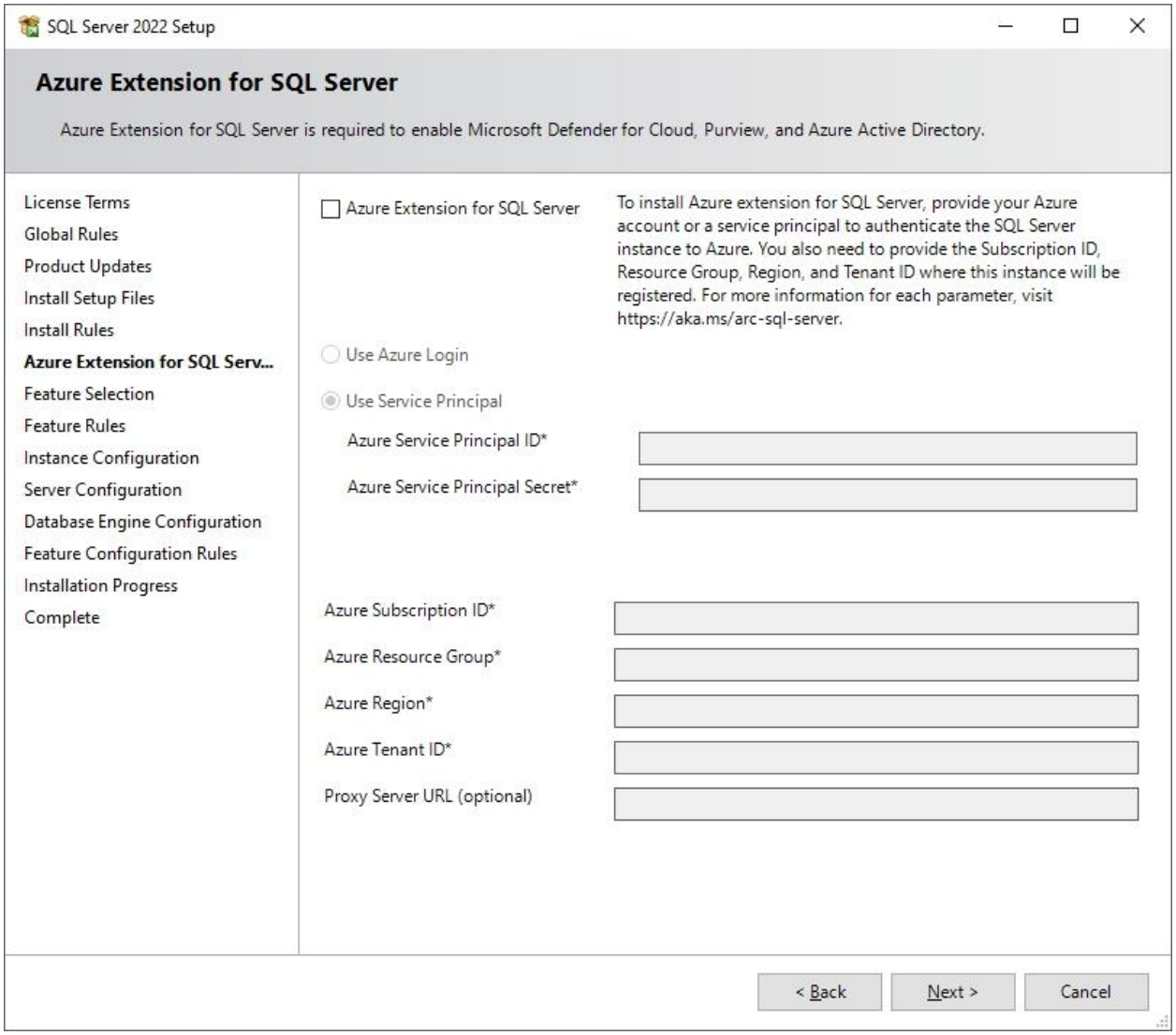

SQL Server 2022 Setup
Global Rules
Setup Global Rules identify problems that might occur when you install SQL Server Setup support files. Failures must be corrected before Setup can continue.
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Azure Extension for SQL Server
Feature Selection
Feature Rules
Feature Configuration Rules
Installation Progress
Complete
Operation completed. Passed: 7. Failed 0. Warning 0. Skipped 0.
Show details >>
Re-run
View detailed report
< Back
Next >
Cancel


SQL Server 2022 Setup
Install Setup Files
If an update for SQL Server Setup is found and specified to be included, the update will be installed.
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Azure Extension for SQL Server
Feature Selection
Feature Rules
Feature Configuration Rules
Installation Progress
Complete
Task
Status
Scan for product updates
Completed
Download Setup files
Skipped
Extract Setup files
Skipped
Install Setup files
Skipped
< Back
Next >
Cancel

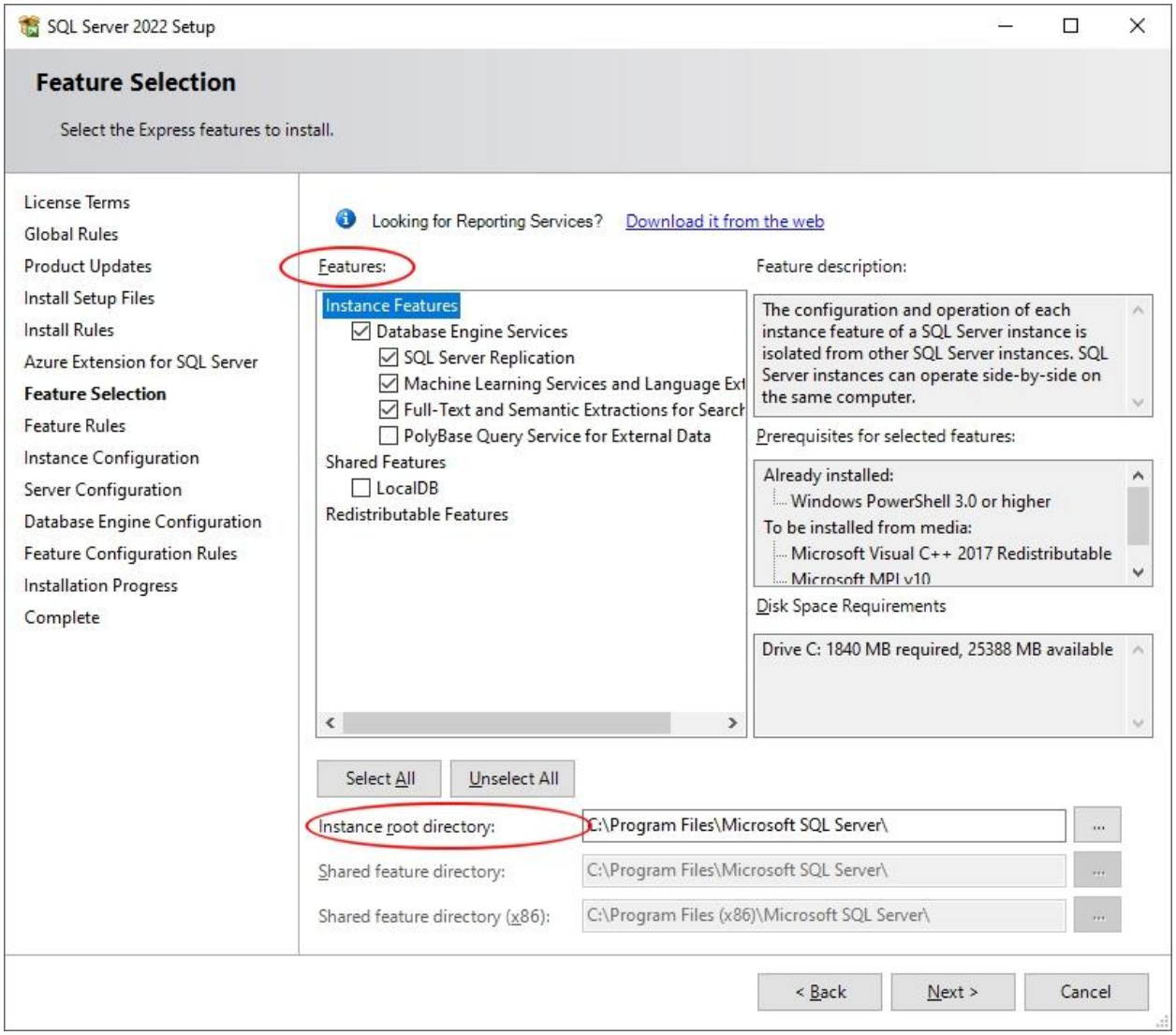

SQL Server 2022 Setup
Install Rules
Setup rules identify potential problems that might occur while running Setup. Failures must be corrected before Setup can continue.
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Azure Extension for SQL Server
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Installation Progress
Complete
Operation completed. Passed: 4. Failed 0. Warning 1. Skipped 0.
Hide details <<
Re-run
View detailed report
Result
Rule
Status
Machine Learning Server shared feature support
Passed
Consistency validation for SQL Server registry keys
Passed
Computer domain controller
Passed
Windows Firewall
Warning
Microsoft .NET Framework 4.7.2, or newer, is required
Passed
< Back
Next >
Cancel

Mesajul despre *Firewall* poate fi trecut cu vederea, acum:



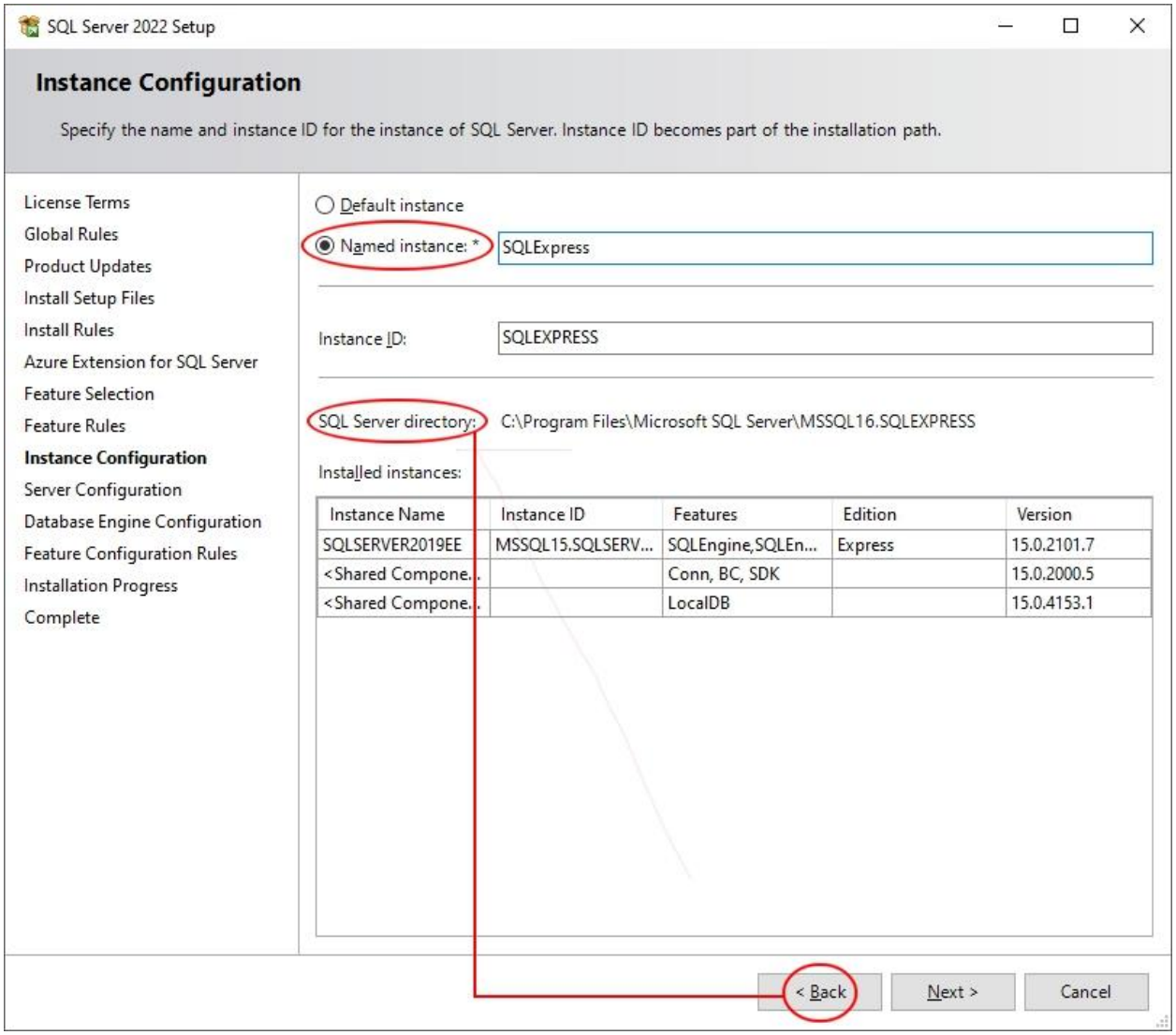

SQL Server 2022 Setup
Azure Extension for SQL Server
Azure Extension for SQL Server is required to enable Microsoft Defender for Cloud, Purview, and Azure Active Directory.
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Azure Extension for SQL Serv...
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Installation Progress
Complete
Azure Extension for SQL Server
To install Azure extension for SQL Server, provide your Azure account or a service principal to authenticate the SQL Server instance to Azure. You also need to provide the Subscription ID, Resource Group, Region, and Tenant ID where this instance will be registered. For more information for each parameter, visit https://aka.ms/arc-sql-server.
Use Azure Login
Use Service Principal
Azure Service Principal ID*
Azure Service Principal Secret*
Azure Subscription ID*
Azure Resource Group*
Azure Region*
Azure Tenant ID*
Proxy Server URL (optional)
< Back
Next >
Cancel

Putem specifica directorul/folderul unde va fi instalat / care va susţine sistemul SQL Server:
("Instance root directory"):



Putem da un nume sistemului SQL Server pe care îl instalăm:
("Named instance"):

Dacă ne răzgândim în privinţa directorului/folderului unde va fi instalat / care va susţine acest sistem SQL Server, apăsăm butonul marcat "Back":

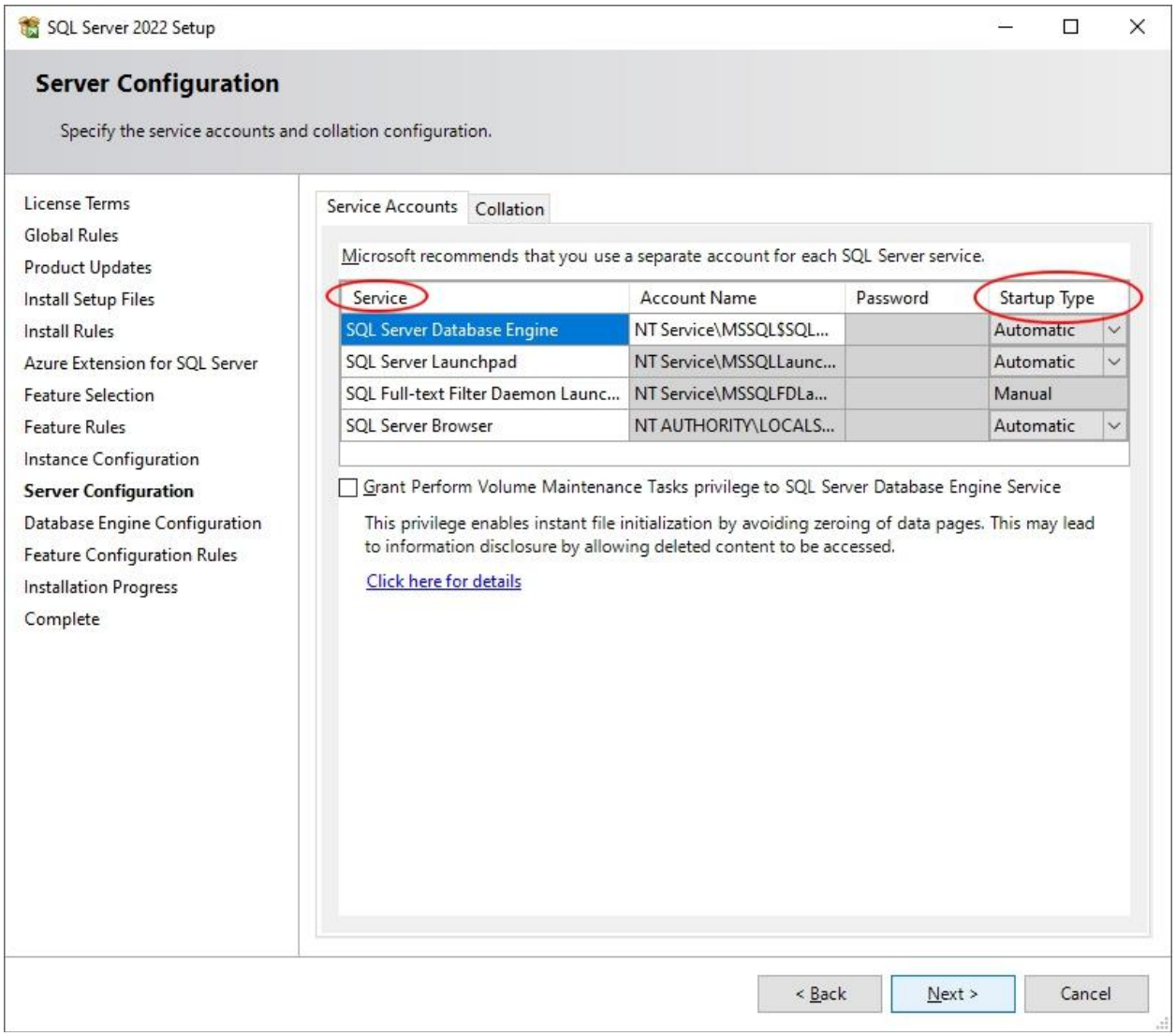


Modulele care asigură funcţionarea corectă a SQL Server (în primul rând găsirea/identificarea sa pe computer) pot fi lansate în execuţie odată cu pornirea *Windows*.

Aceasta se face astfel:

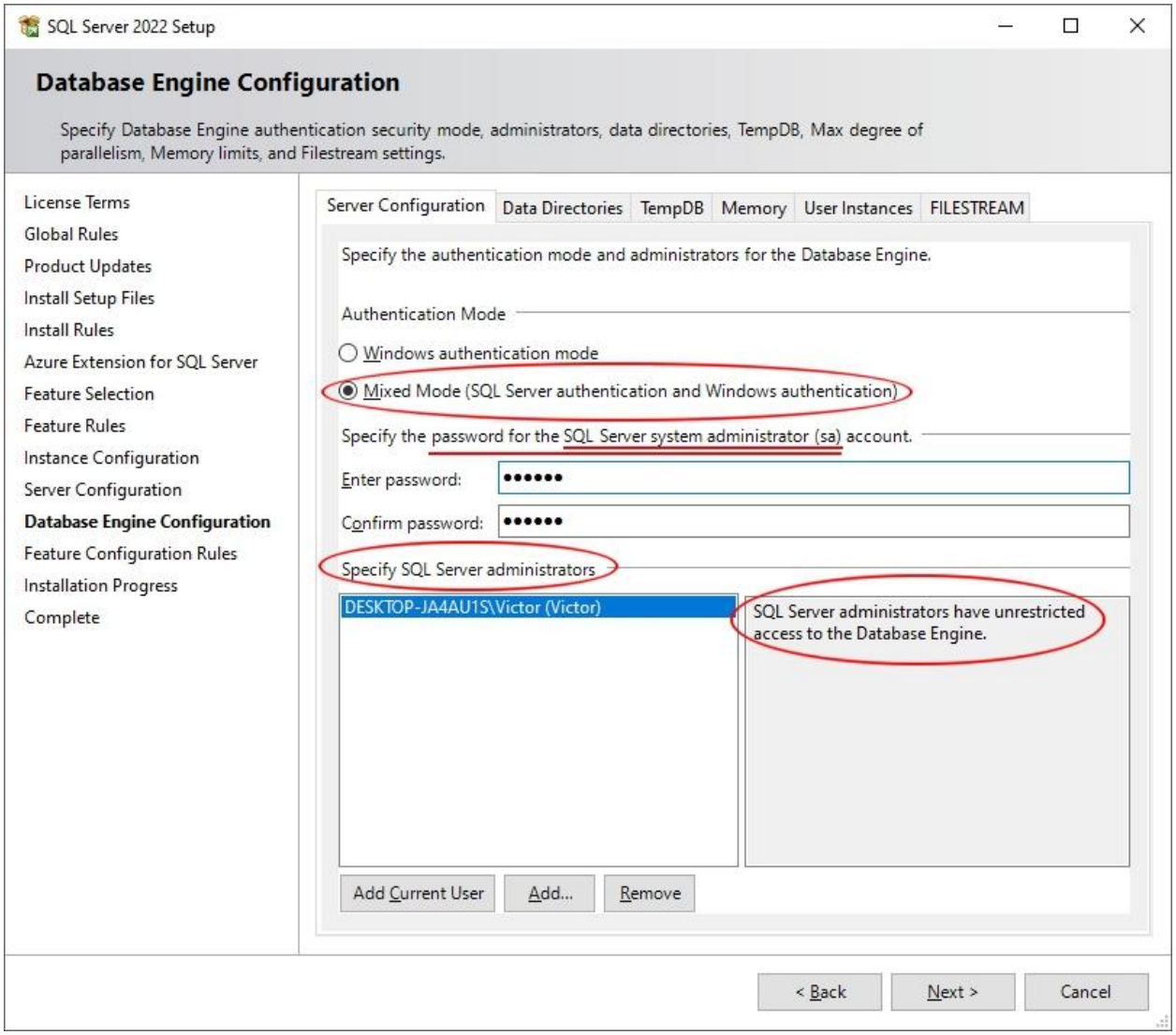
- specificând modul de pornire ("Startup Type") al acelor *servicii* (adică, în sens larg, programe fără interfaţă cu utilizatorul) la "Automatic"
- - selectând un utilizator (un "User Account") al computerului care are cât mai multe drepturi, inclusiv pe cel de **Administrare**. De obicei în *Windows*, astfel de utilizatori sunt: **SYSTEM** şi **NETWORK_SERVICE**

Deocamdată, opţiunile selectate implicit pot fi lăsate aşa:

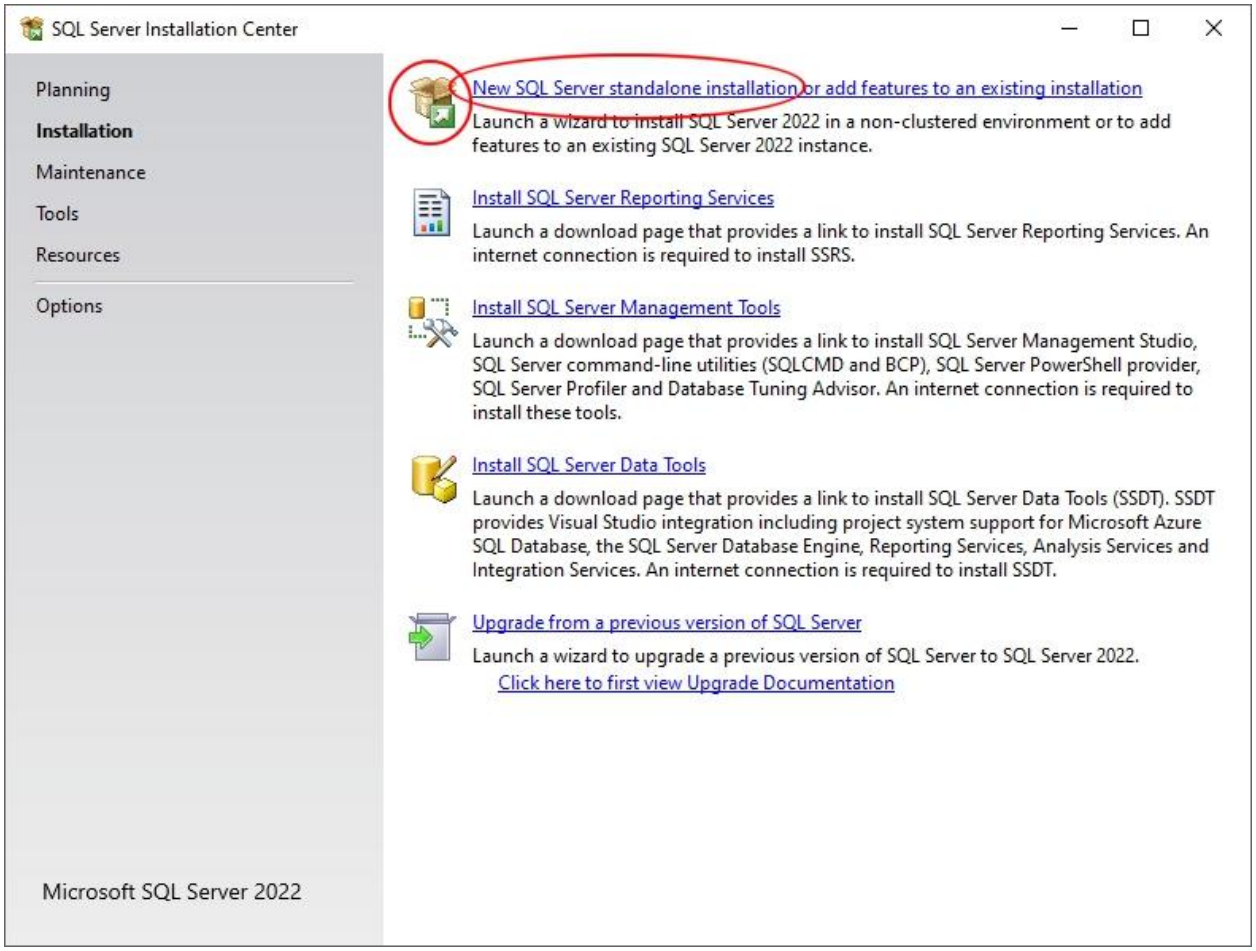


Selectăm opţiunea care oferă posibilitatea ca Serverul SQL instalat acum să poată fi accesat prin nume şi parolă specificate de către un utilizator cunoscut de/autentificat la *Windows* de pe acelaşi computer: "***Mixed Mode...***":

Uzual, parola pentru administratorul "**sa**" al acestui server SQL este stabilită ca una simplă – de exemplu, "**sa123**".



Aşteptăm completarea instalării SQL Server, apoi închidem dialogul de la început:



Aşadar, până acum am instalat serverul SQL propriu-zis, *pe acest computer*.

Deşi această constatare este trivială şi se subînţelege, este important
ca atunci când instalăm mai multe servere SQL într-o reţea de computere
să reţinem şi eventual să notăm undeva pentru fiecare astfel de instalare:

- denumirea serverului SQL
- denumirea reţelei
- denumirea computerului pe care există un server SQL
- adresa IP a acelui computer

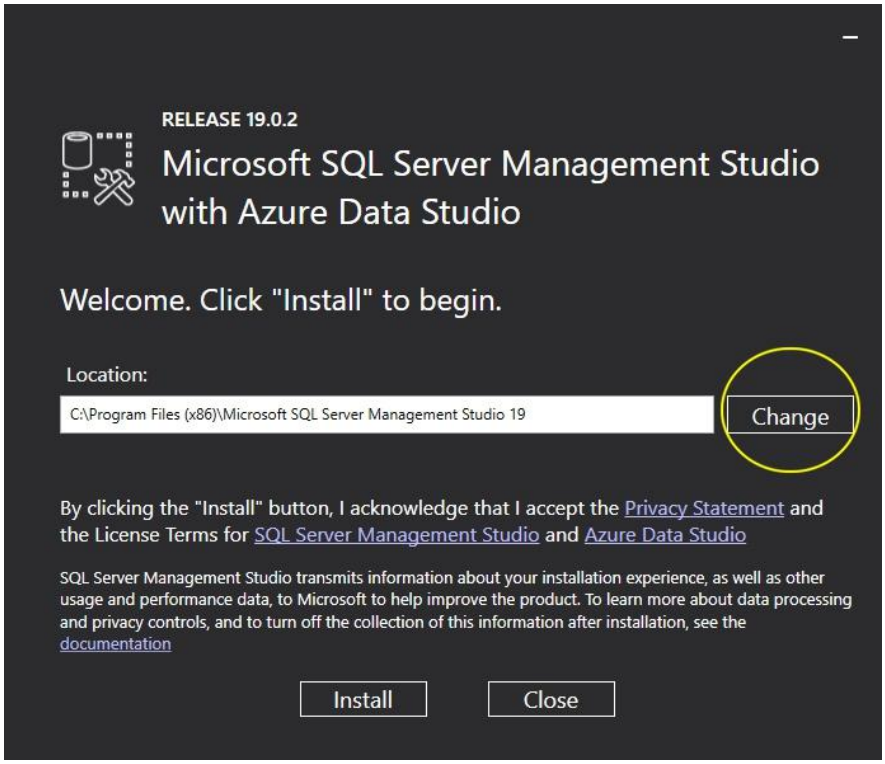
Încă nu putem interacţiona (vizual) cu serverul SQL tocmai instalat.

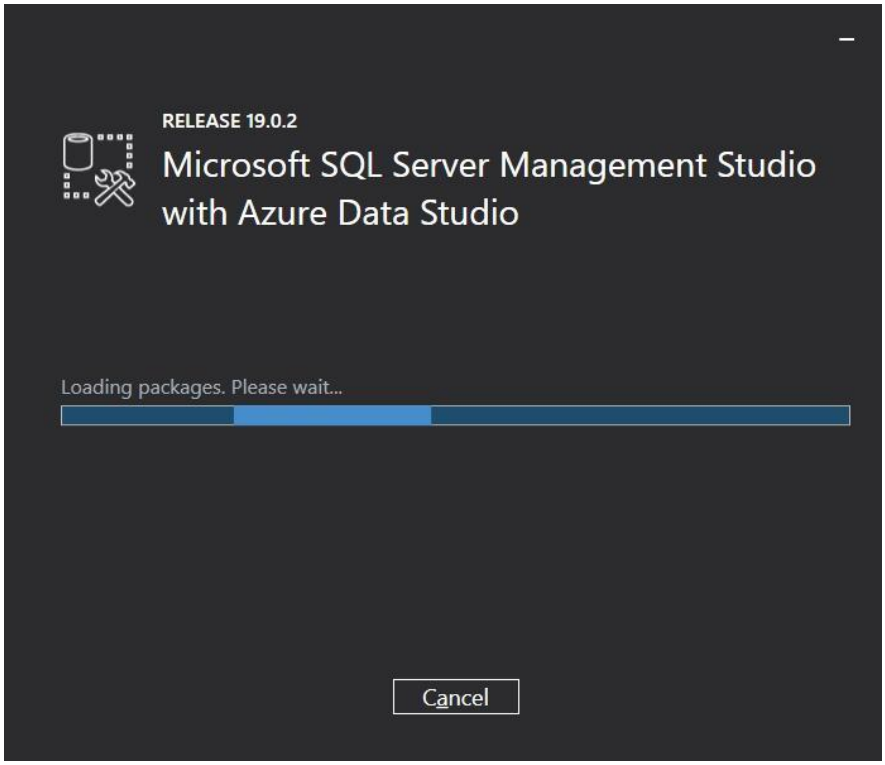
Interacţiunea cu entităţi aferente SQL Server făcută prin tastatură şi mouse uşurează mult lucrul cu astfel de obiecte complexe. Pentru aceasta există o unealtă care tocmai asta oferă operatorului, la fel cum Visual Studio®™ oferă programatorului posibilitatea de a crea programe de asemenea având interfaţă vizuală.

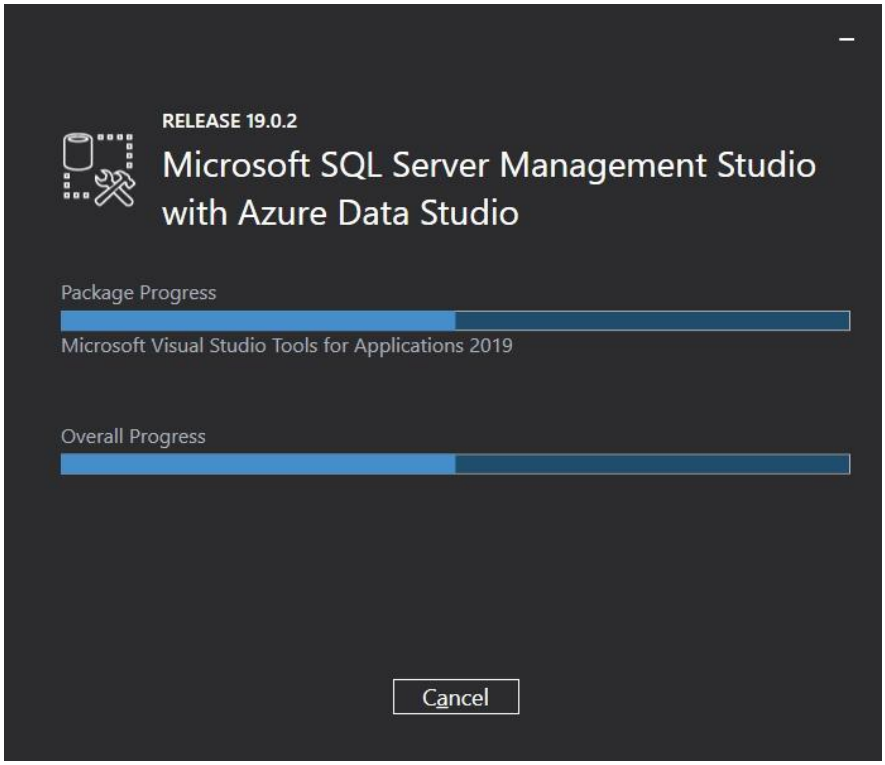
Această unealtă, dedicată lucrului cu un SQL Server, este ***SQL Server Management Studio ("SSMS")***.

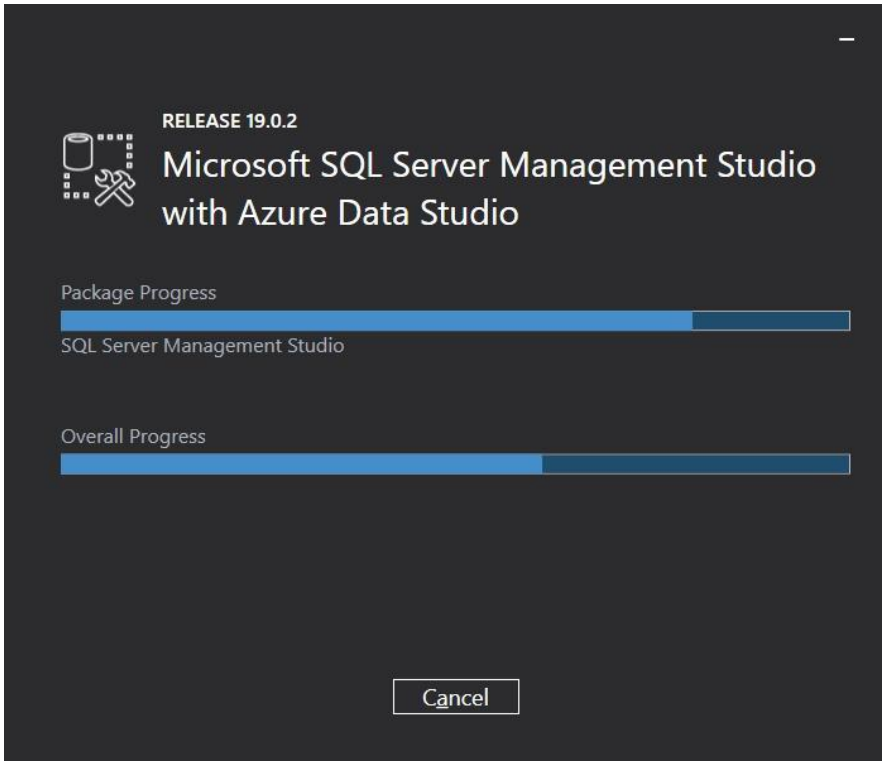
Aşa că instalăm acest pachet de programe.

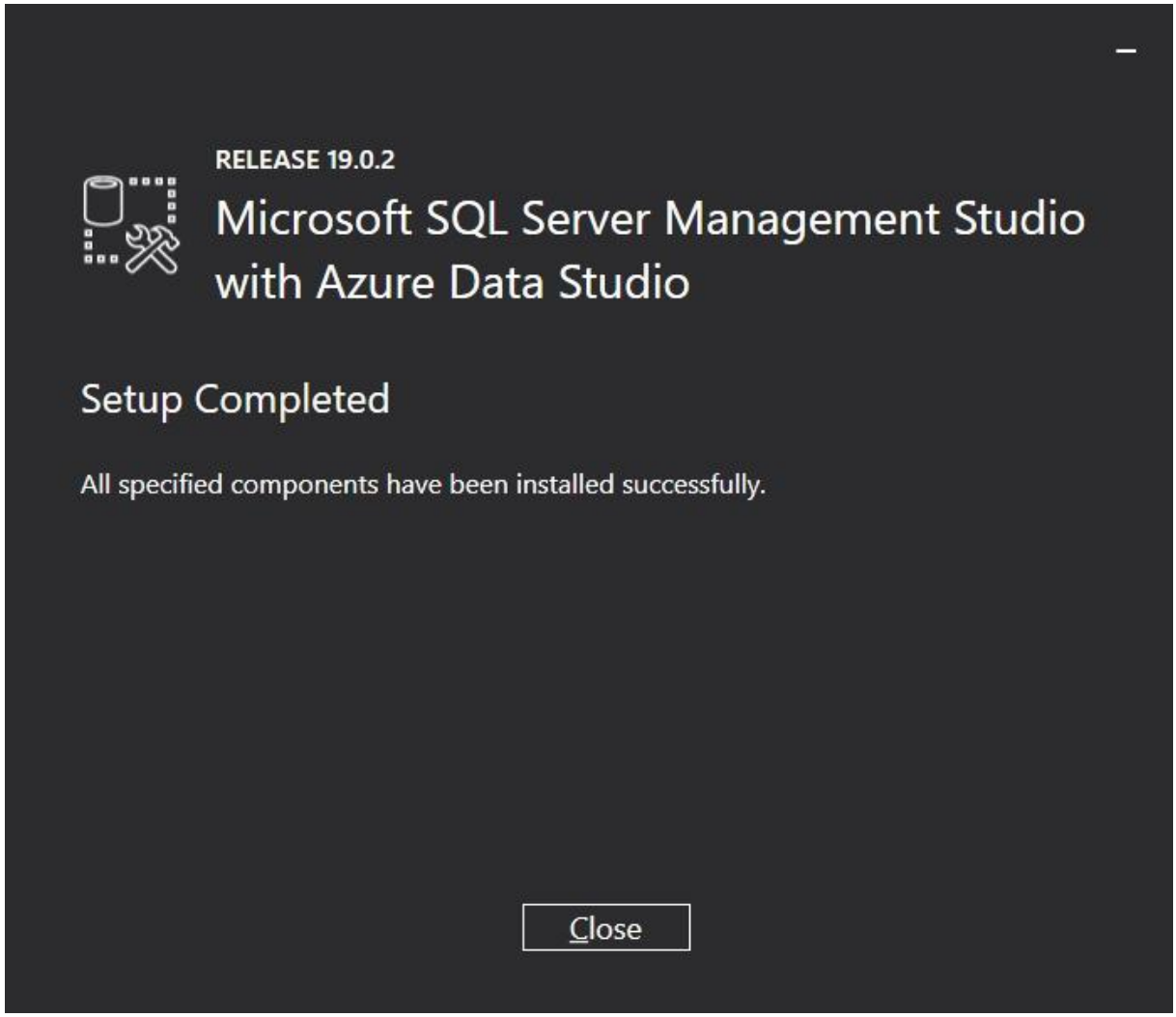
Această procedură este mult mai uşoară şi mai scurtă decât instalarea SQL Server, fiindcă nu solicită utilizatorul decât la începutul instalării:




RELEASE 19.0.2
Microsoft SQL Server Management Studio
with Azure Data Studio
Loading packages. Please wait...
Cancel


RELEASE 19.0.2
Microsoft SQL Server Management Studio
with Azure Data Studio
Package Progress
Microsoft Visual Studio Tools for Applications 2019
Overall Progress
Cancel


RELEASE 19.0.2
Microsoft SQL Server Management Studio
with Azure Data Studio
Package Progress
SQL Server Management Studio
Overall Progress
Cancel

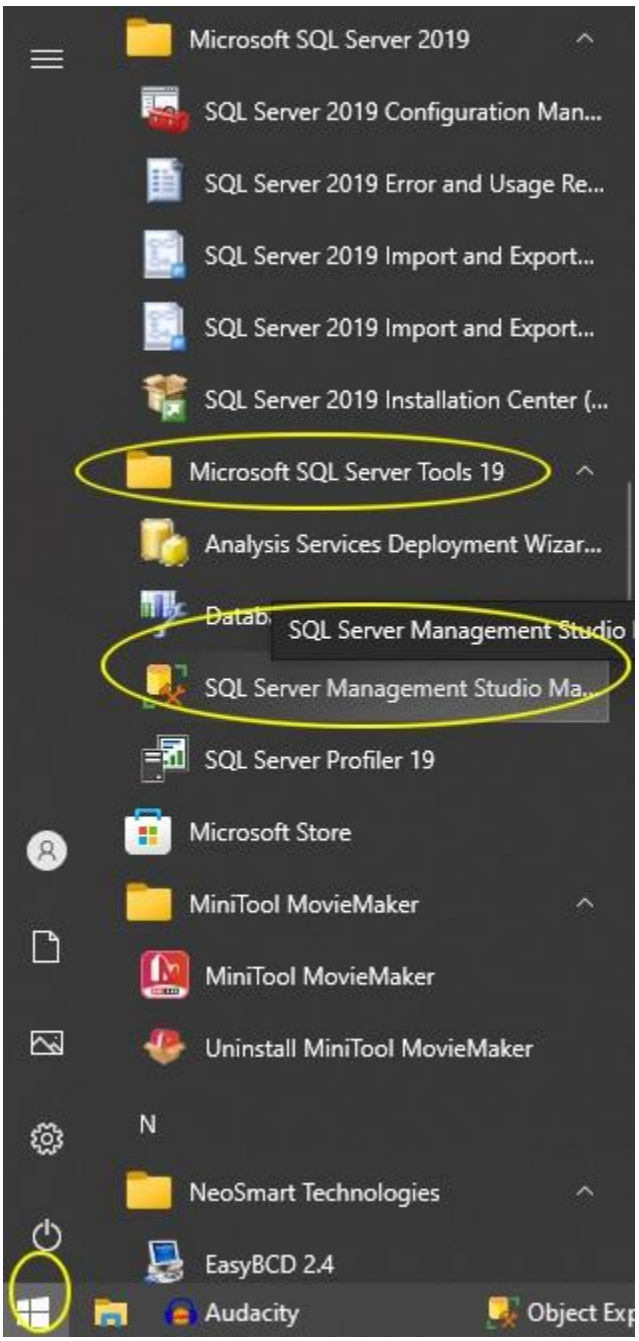


După ce facem şi aceasta, pornim SSMS:

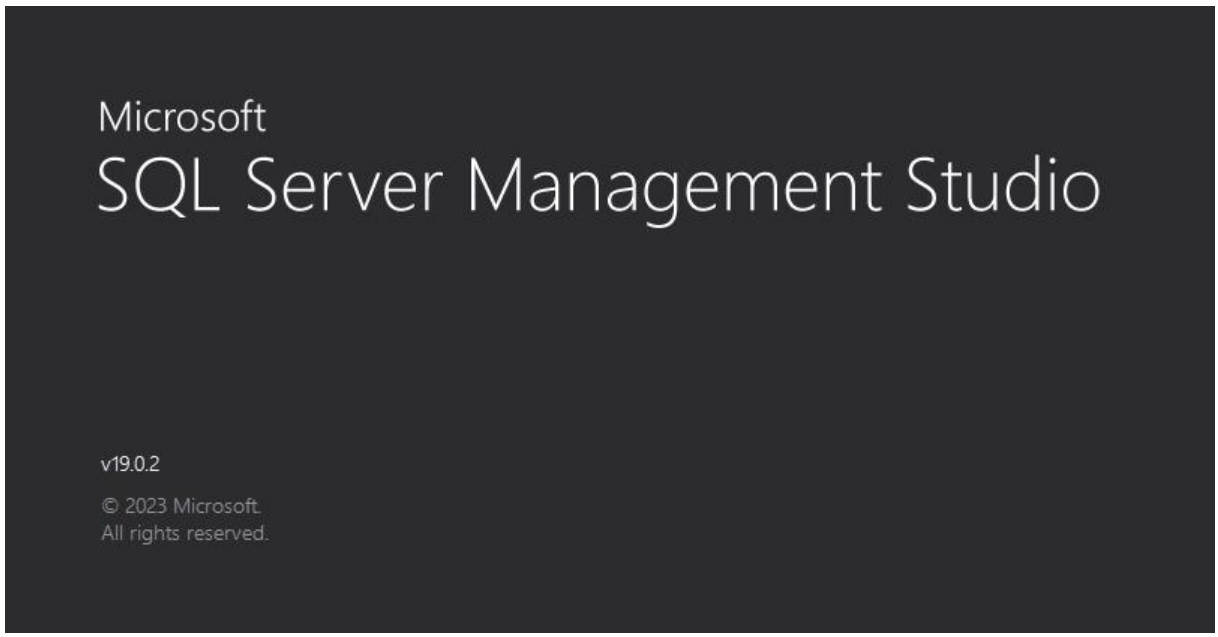
Imaginea următoare este cea care apare după ce în *Windows 10* se apasă butonul *Start* şi se scrie "ssms":



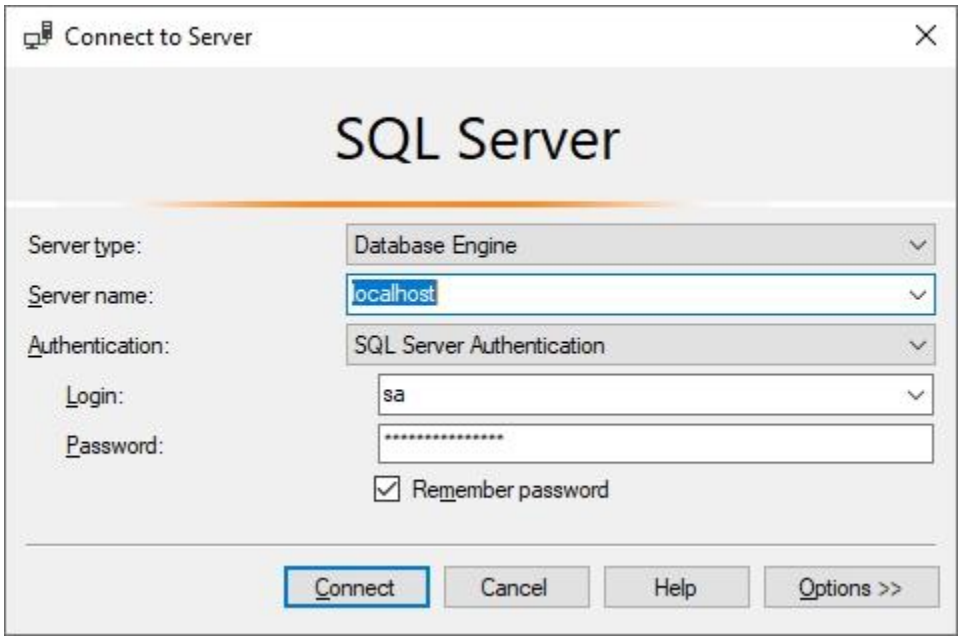
În unele versiuni de *Windows 10*, SSMS este în grupul "Microsoft SQL Server Tools 19" unde poate fi găsit după ce se apasă butonul *Start* şi se derulează până acolo – cum se vede în imaginea următoare:



Click pe intrarea de meniu a butonului *Start* din imaginile precedente:
SSMS va porni:



Imediat suntem întrebaţi la care anume server SQL *cunoscut acestui SSMS/computer*
vrem să ne conectăm:
Observăm că în dialogul de conectare la un server SQL este propus numele computerului curent.
Îl înlocuim cu "localhost":

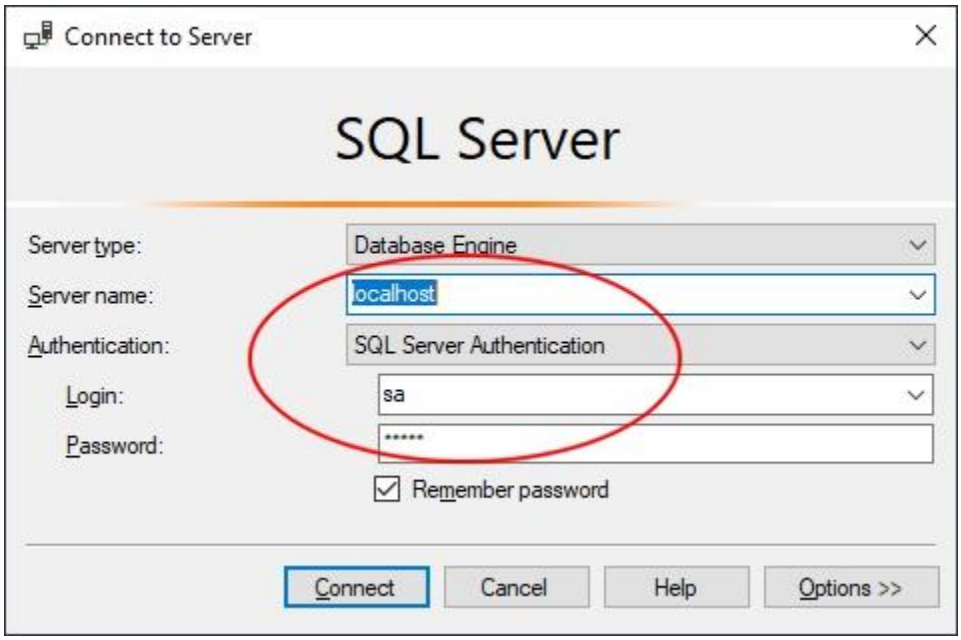


localhost şi punctul **.** se referă la ***acest computer***

- este numele ***subînţeles al serverului SQL de pe acest computer*** *unde* l-am instalat mai devreme.

Din caseta *ComboBox* marcată "Authentication" selectăm: ***SQL Server Authentication***:

Specificăm ***numele de utilizator/administrator al serverului SQL***, de obicei "**sa**"
(acest nume a fost stabilit în cursul instalării serverului SQL) şi
parola de acces la serverul SQL:



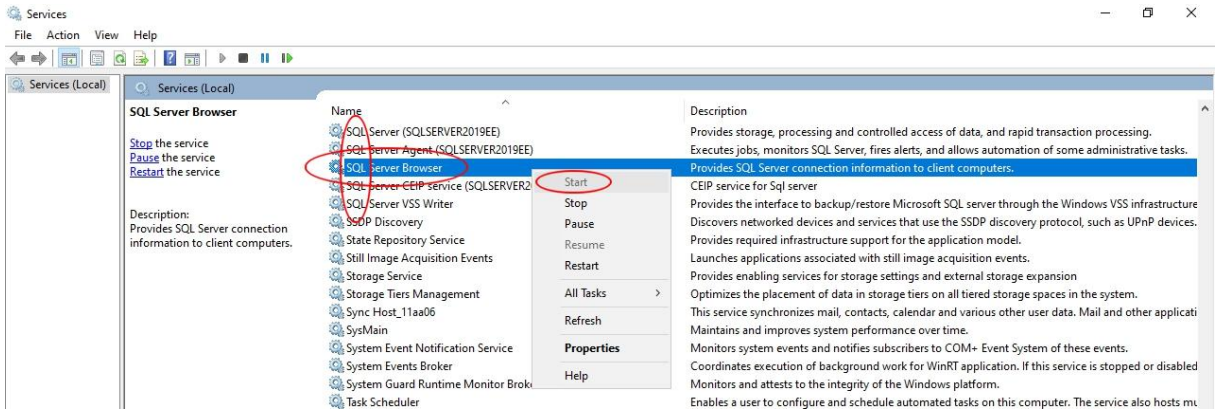
Dacă nu este acceptată solicitarea de conectare, facem acestea:

Pornim programul "*Services*" al *Windows*:
Acest program poate fi găsit în/cu *Control Panel*.
Imaginea următoare este cea care apare după ce în unele versiuni de *Windows 10* se apasă butonul *Start* şi se scrie "Services":

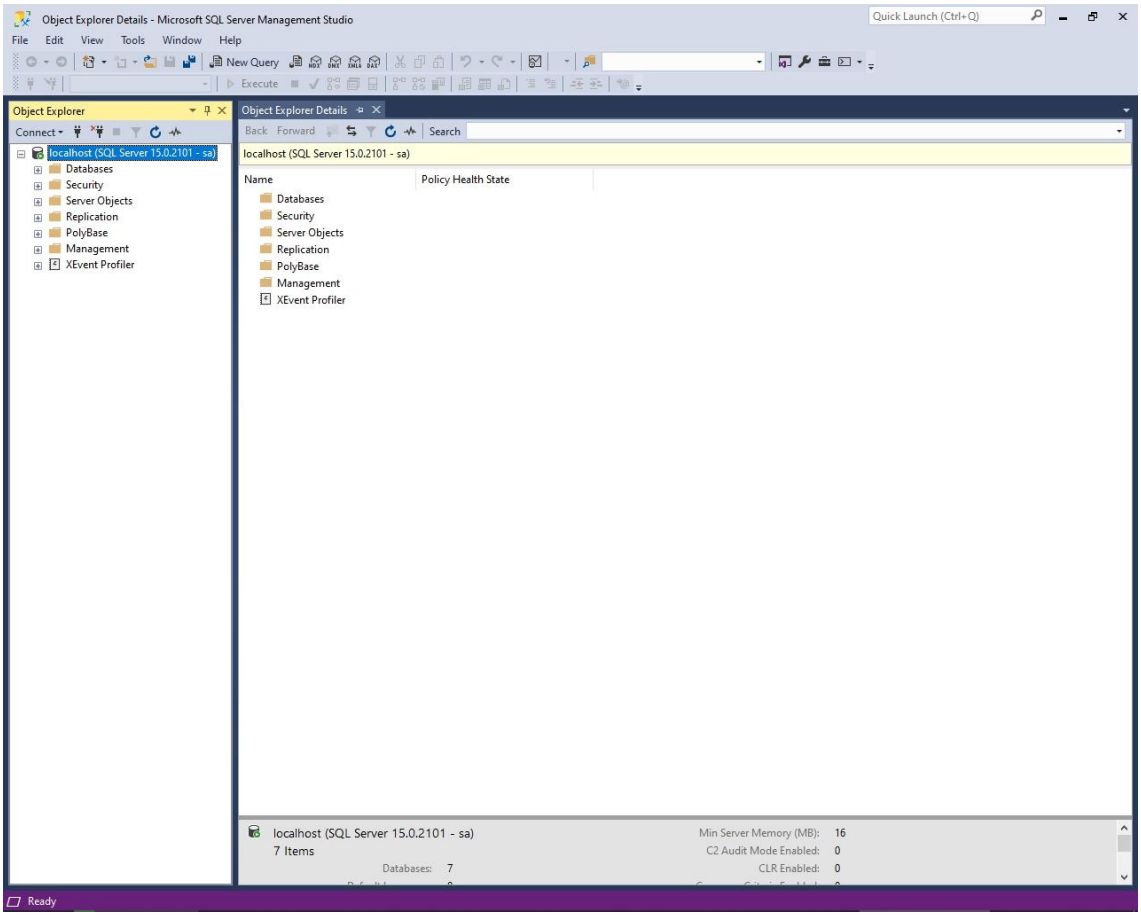


În acest program:

- căutăm grupul "***SQL***" de *Servicii*
- ne asigurăm că Serviciul "***SQL Server Browser***" este **pornit**:
    - click-dreapta pe numele acestuia, apoi din meniu se alege comanda *Start*:



Apoi ne întoarcem în SSMS:
După ce furnizăm numele ("Login") şi parola de acces ***la SQL Server-ul specificat***,
SSMS ne prezintă obiectele cunoscute din sistemul SQL Server la care ne-am conectat:

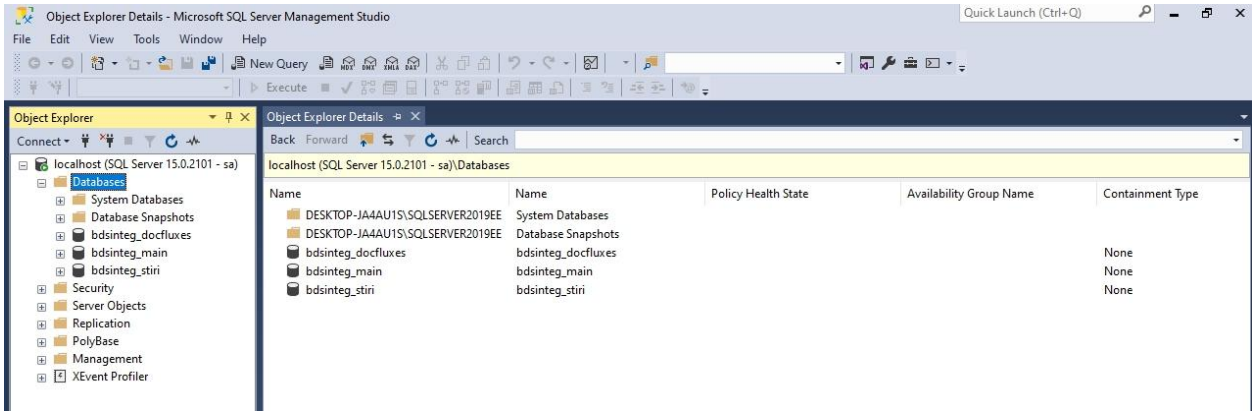


Dacă nu apare această imagine, atunci apăsăm tasta **F7** ("Object Explorer Details").

În această imagine, în stânga vedem primele două niveluri (superioare) din structura arborescentă a SQL Server-ului la care ne-am conectat:

- localhost
  - Databases
  - Security
  - Server Objects
  - ...

Vrem să ştim ce baze de date există acum în SQL Server:
aşa că desfăşurăm nodul "Databases" din acel *TreeView*:
Am putea vedea ceva asemănător imaginii urm ătoare:



Nodul "Databases" are mai multe noduri-descendent, printre care "System Databases".

Pe acelaşi nivel cu "System Databases" se află noduri cu numele
bazelor de date create de utilizatorii SQL Server-ului,
nu cele create automat.

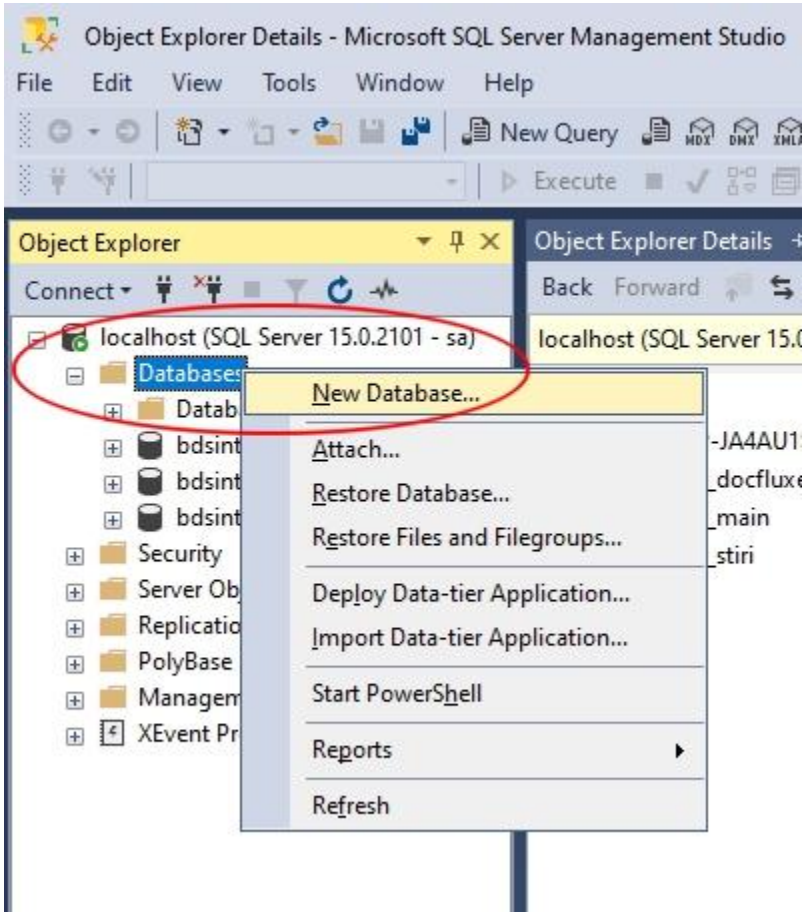
De obicei când se lucrează cu SSMS, acele baze de date "ale sistemului" nu prezintă interes.
Pentru ascunderea lor trebuie setată opţiunea corespunzătoare,
din meniul *Tools -> Options -->*
Startup -> Hide system objects in Object Explorer, sau similar

De acum putem crea baze de date folosind SSMS.

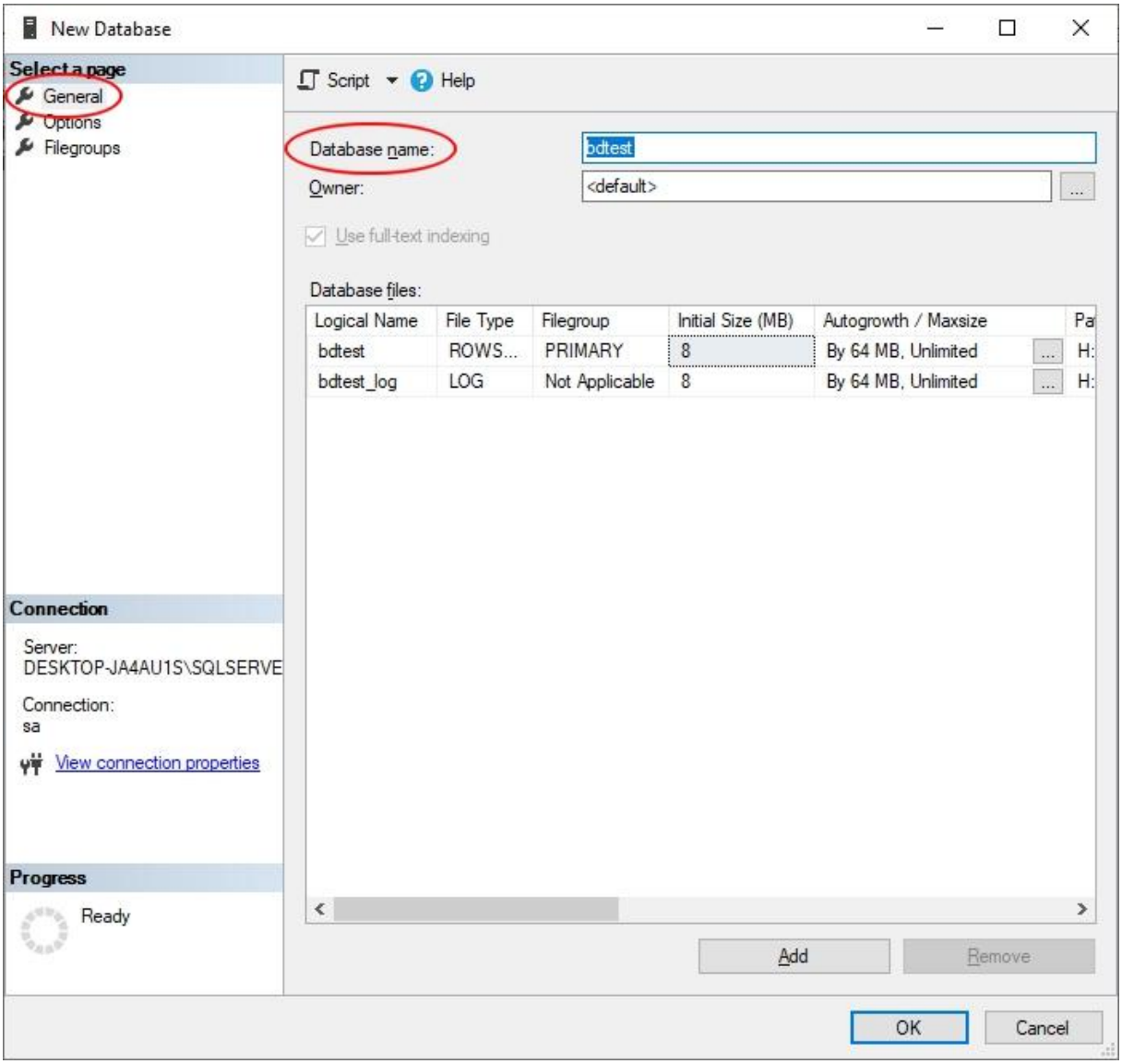
Să creăm o bază de date pe care s-o şi populăm cu tabele în care să punem înregistrări.

SQL Server Management Studio: Crearea unei baze de date

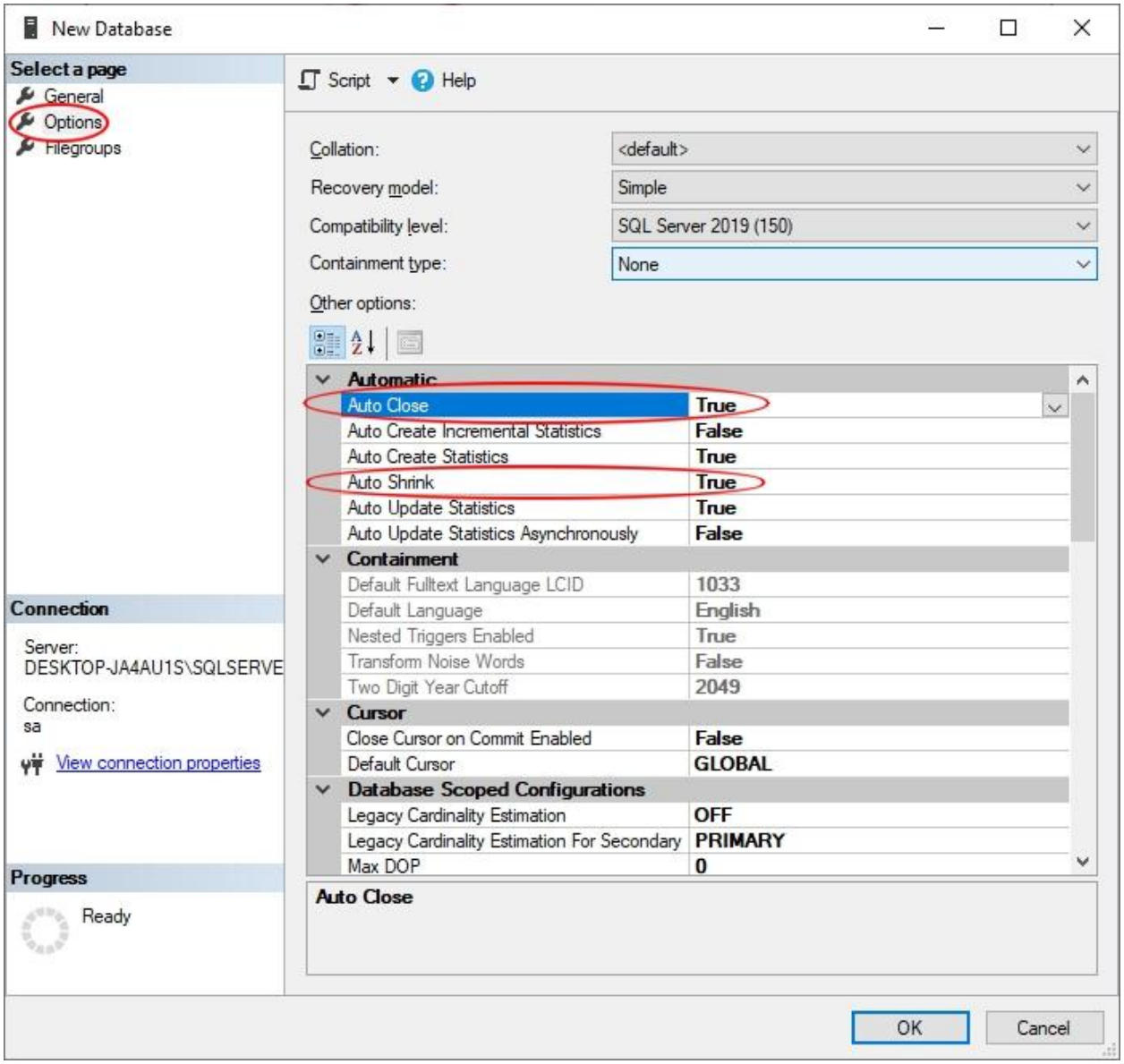
În SSMS:
În *TreeView*-ul din stânga:
Click-dreapta pe nodul "Databases":
Apoi, pe intrarea de meniu "New Database...":



Dăm un nume noii baze de date:

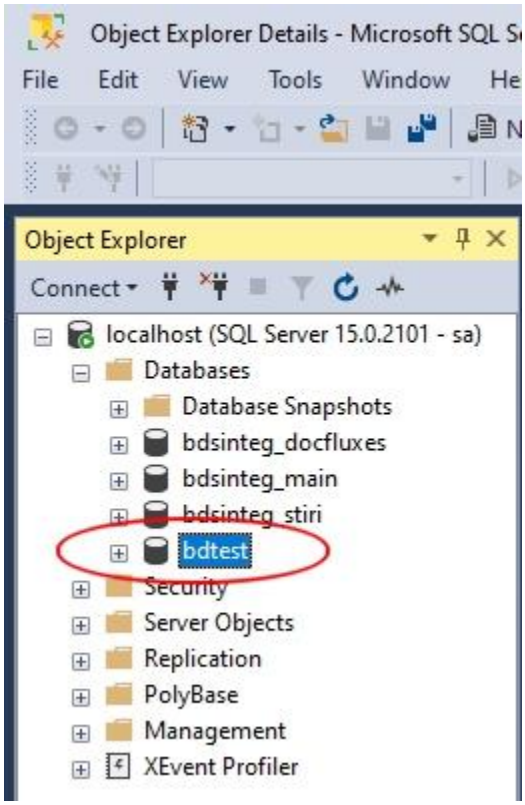


Stabilim câteva opţiuni importante pentru noua bază de date:



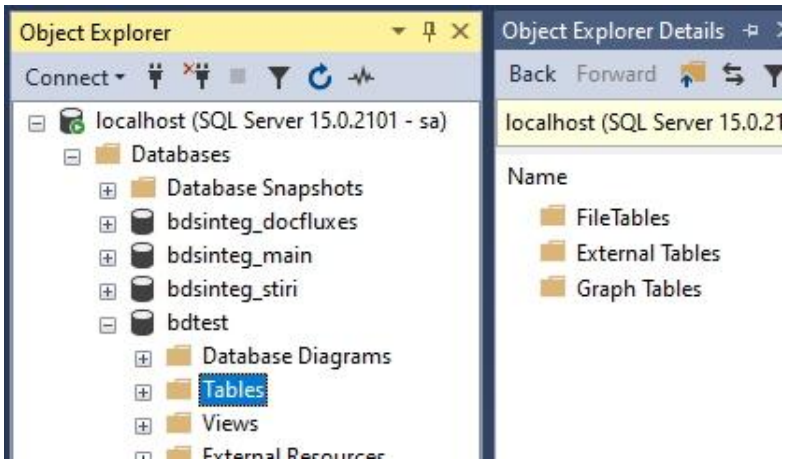
Apoi apăsăm butonul marcat "OK".

Baza de date propusă va fi creată
şi va apărea în *TreeView*-ul din stânga, în nodul "Databases":



Desfăşurând (click pe semnul +) nodul cu numele bazei de date,
vedem obiectele accesibile acum din aceasta:

Click-stânga (selectare) pe "Tables"
pentru a afla ce tabele conţine baza de date:
Rezultatul ar trebui să fie cel din imaginea următoare; dacă nu este acesta, atunci apăsăm tasta **F7** care invocă afişarea în SSMS a secţiunii sale "Object Explorer Details":

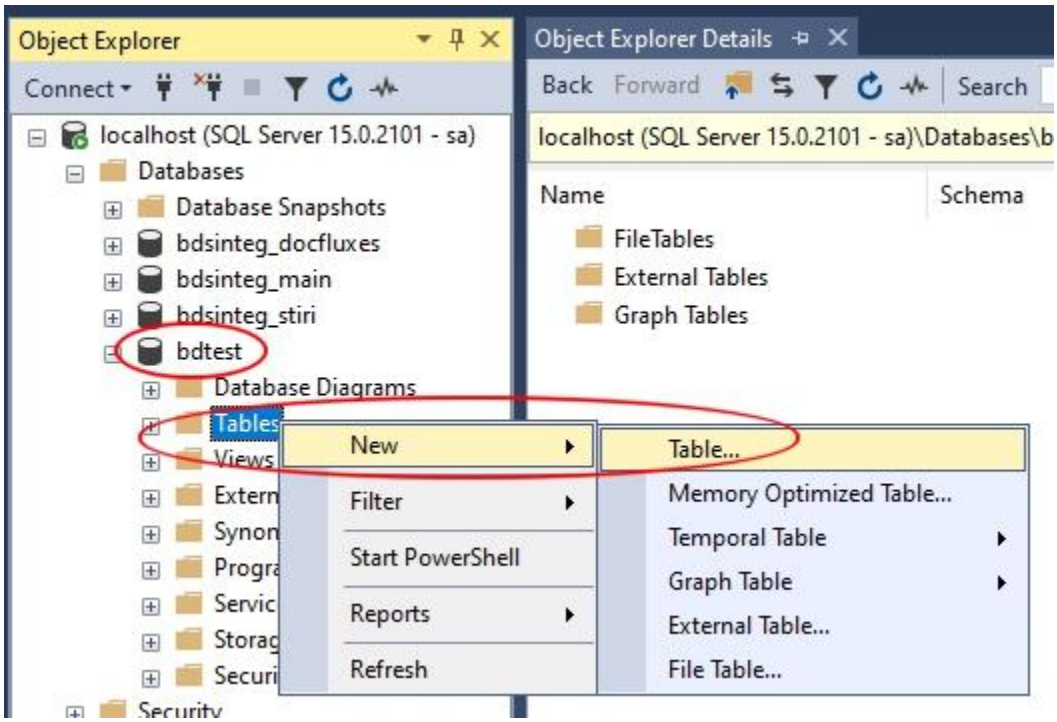


SQL Server Management Studio: Crearea unui tabel şi plasarea sa într-o bază de date

Deocamdată, imediat după crearea bazei de date, aceasta nu conţine niciun tabel care să fie populat cu informaţii utile nouă, ca utilizatori obişnuiţi ai acesteia.

Dar vrem să stocăm informaţii în această bază de date, ceea ce înseamnă că
trebuie să creăm ***tabele***
care să conţină acele ***înregistrări***.

Pentru aceasta:
click-dreapta pe nodul "Tables", apoi pe intrarea de meniu "New", apoi pe "Table...":



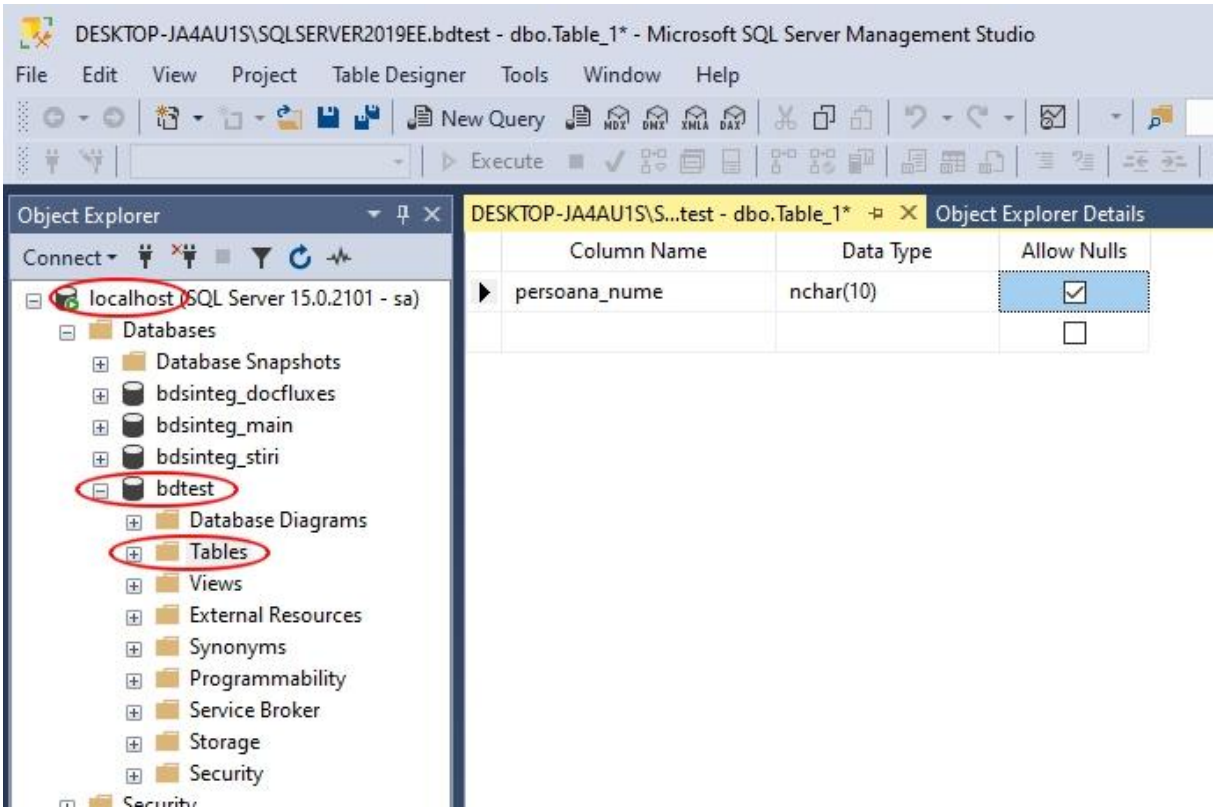
Aşadar, vom crea un tabel:

SSMS solicită ***coloanele***, ori ***câmpurile*** (***Field*s**) tabelului:

- numele fiecărei coloane: ***"Column Name"***
- tipul valorilor (particulelor de informaţie) care vor fi plasate pe o coloană: "***Data Type***"
- dacă pe această coloană pot fi valoir nule (celule goale): "***Allow Nulls***"
    - dacă ştim că poate sau că va surveni această situaţie, atunci bifăm caseta "Allow Nulls"
    - în această versiune – 19.0.2. a SSMS, imediat ce se bifează această casetă se observă cum *Data Type* este stabilit la "nchar(10)"; acest tip de valori denotă un text, ori un *String* (cunoscut din limbajul C#) a cărui lungime este 10 (caractere).

În imaginea următoare se vede cum adăugăm o coloană având:

- numele (denumirea): "persoana_nume"
- tipul valorilor de pe această coloană a fost stabilit automat la "nchar(10)" fiindcă am bifat caseta "*Allow Nulls*" care face încât această coloană să accepta valori nule (ori, celule goale):
- capacitatea de a susţine/accepta valori nule: "*Allow Nulls*" este/are valoarea *bool*eană: *True*



Este posibil să vrem ca valorile din această coloană să fie de alt tip ("*Data Type*")
decât cel propus – *nchar(10)*:

Un tabel trebuie proiectat încât să poată susţine valori neprevăzute, dar posibile.

Dacă nu se procedează aşa, atunci în acel tabel nu vor putea fi trimise valori care...

- ori au o dimensiune (atât ca text, cât şi ca numere) mai mare decât "*Data Type*" al coloanei-destinaţie a lor
- ori sunt nule (adică, efectiv nimic – nicio valoare, nici măcar textul nul ""), dar coloana-destinaţie a lor să nu poată susţine valori nule ("Allow Nulls")

De exemplu:

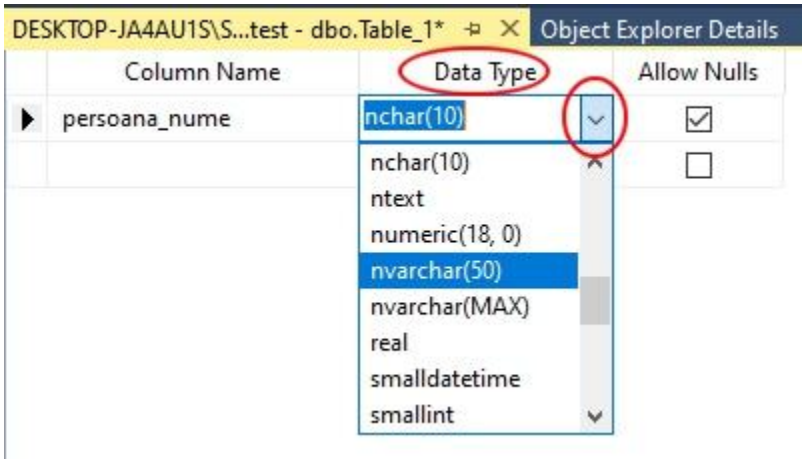
- când creăm o coloană pe care intenţionăm a plasa ca text numele unor persoane din mai multe ţări, stabilim lungimea maximă a textului acceptat de acea coloană la un număr presupus a fi mai mare decât cel mai mare folosit în toate ţările, fiindcă în unele ţări numele de familie este mai lung de 15 caractere/litere, în alte ţări prenumele lipseşte, iar în alte ţări numele de familie este compus din două sau mai multe nume – fiecare mai lung de 10 caractere/litere; astfel că putem stabili tipul acelei coloane la, de exemplu, nchar (100).
- când creăm o coloană pe care intenţionăm a plasa numere – de exemplu, temperatura aerului la o anumită oră - presupunem că pot fi situaţii când valoarea temperaturii lipseşte la sursă, astfel că pe acea coloană nu va fi plasat nimic, cu alte cuvinte – acea celulă va fi lăsată goală; astfel că bifăm caseta "Allow Nulls" – proiectăm/pregătim acea coloană încât să accepte valori nule

În acest fel,
încă de la proiectarea unui tabel
se ia o măsură de precauţie referitor la
valorile care vor fi plasate în acel tabel.

Măsuri de precauţie pentru trimiterea unor valori valide într-un tabel pot fi luate după ce (coloanele) tabelul(ui) a(u) fost proiectat(e), bineînţeles înainte de a i se trimite acele valori.
De exemplu, dintr-un program valorile nule (nimic) pot fi trimise astfel:

- ca textul nul "", în coloanele de tip text
- ca "-999" sau o valoare foarte improbabilă, în coloanele de tip numeric
- ca "1900-01-01" sau o dată calendaristică improbabilă/stranie, în coloanele de tip *datetime*
- şamd.

În timpul proiectării unui tabel
tipul unei coloane poate fi stabilit/schimbat
selectând tipul "corect" al coloanei din lista "Data Type",
aşa cum se vede în imaginea următoare:



Este recomandat ca pentru valori de tip text despre care nu se ştie ce caractere/litere vor conţine să fie selectat tipul "***nvarchar***" cu lungimea fiecărui astfel de text fiind în paranteză - în imaginea precedentă a fost ales tipul "nvarchar(50)". Acea lungime poate fi schimbată după selectarea acelui tip, de exemplu pentru acele nume lungi poate fi schimbată la "nvarchar(100)".

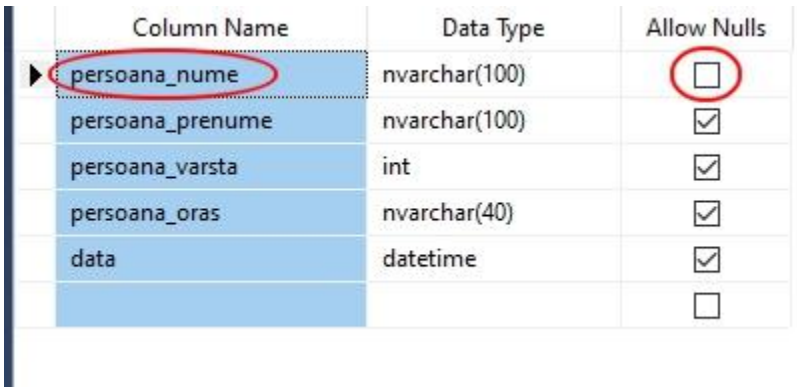
Deşi mai ales tipurile de date *nvarchar* sunt consumatoare de memorie şi de spaţiu pe mediul de stocare a bazei de date, este bine ca atunci când creăm o coloană unde vor fi trimise valori de tip text să proiectăm/pregătim acea coloană ca fiind de tipul *nvarchar* şi cu o lungime rezonabilă şi posibil mai mare (cu 5% - 50%) decât cea reală, de exemplu *nvarchar(100)* pentru nume de persoane şi *nvarchar (50)* pentru denumirile anotimpurilor în limba ţării unde va fi folosită baza de date.

Chiar dacă valorile trimise într-o coloană de tip text nu vor avea toate lungimea maximă a acelui câmp (adică, a acelei coloane), SQL Server alocă/rezervă atât spaţiu de stocare pentru toate valorile, cât este specificat ca lungimea mximă acceptată de acel câmp; de exemplu, dacă trimitem un text de 5 litere într-un câmp *nvarchar(100)*, acel text va fi plasat într-o celulă pentru care s-a rezervat spaţiu pentru 100 litere.

Pentru coloanele de tip numeric nu trebuie specificată o lungime anume a valorilor acceptate, findcă de aceasta se ocupă SQL Server.

Pot fi create coloane unde să fie trimise informaţii diferite de cele indicate de numele coloanelor.
De exemplu, poate fi creată o coloană dedicată unor texte şi având numele "anotimp", iar valorile trimise să fie denumirile unor zile ale săptămânii, sau numele unor persoane.
Aceasta este exclusiv responsabilitatea celui care populează un tabel – fie acesta un individ sau un program de calculator.

Având în vedere aceste precauţii şi observaţii, să creăm cinci coloane în acest tabel:



Column Name	Data Type	Allow Nulls
persoana_nume	nvarchar(100)	☐
persoana_prenume	nvarchar(100)	☑
persoana_varsta	int	☑
persoana_oras	nvarchar(40)	☑
data	datetime	☑
		☐

Am definit coloanele astfel:

- "**persoana_nume**": este o coloană...
  - unde vor fi trimise numele de familie ale unor persoane, deci dedicată numai acelei particule de informaţie
  - de un tip adecvat stocării unor texte despre care nu ştim nimic (ce fel de alfabet, ce caractere din alfabet ş. a.), anume ***nvarchar***
  - în care pot fi stocate texte lungi de maximum 100 litere/caractere: ***nvarchar(100)***
  - care nu acceptă valori nule – cu menţiunea că
    - textul nul "" nu este valoare nulă, ci un text de lungime 0

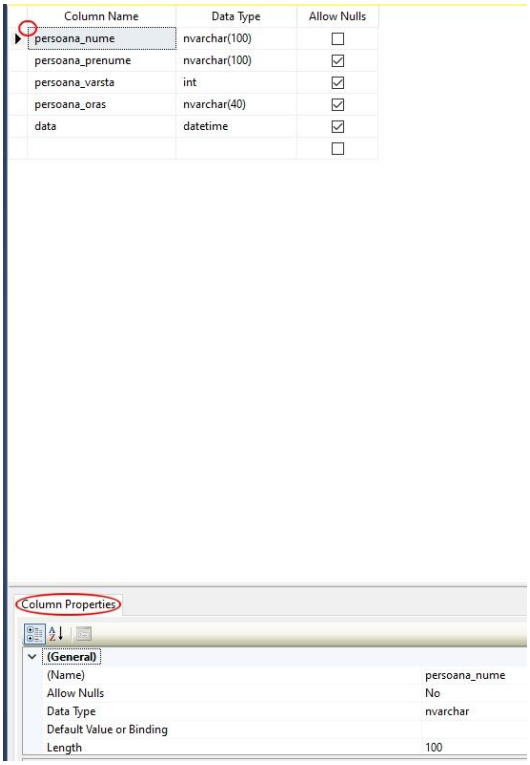
- "**persoana_prenume**": este o coloană...
  - unde vor fi trimise prenumele unor persoane
    - fiecare prenume se află pe aceeaşi linie (*row*), adică în aceeaşi înregistrare despre o persoană, cu numele corespunzător
  - care are acelaşi tip cu tipul coloanei unde vor fi stocate numele: *nvarchar*
  - în care pot fi stocate texte lungi de maximum 100 litere/caractere: ***nvarchar(100)***
  - care acceptă valori nule – cu aceeaşi menţiune ca pentru coloana dedicată localităţilor de reşedinţă/de origine ("persoana_oras")...
    - fiindcă este posibil ca prenumele acelei persoane să nu fie cunoscut

- "**persoana_varsta**": este o coloană..
  - unde vor fi trimise numerele care indică vârsta persoanei cu numele si prenumele de pe o
    - fiecare vârstă se află pe aceeaşi linie (*row*), adică în aceeaşi înregistrare despre o persoană, cu numele şi prenumele corespunzătoare
  - de un tip adecvat stocării unor numere întregi – anume, aici, ***int***
  - care acceptă valori nule – adică, nimic...
    - fiindcă este posibil ca vârsta acelei persoane să nu fie cunoscută

- "**persoana_oras**": este o coloană...
  - unde vor fi trimise numele localităţilor de reşedinţă/de origine a fiecărei persoane
    - fiecare astfel de toponim se află pe aceeaşi linie (*row*), adică în aceeaşi înregistrare despre o persoană, cu numele, prenumele şi vârsta corespunzătoare
  - care are acelaşi tip cu tipul coloanelor unde vor fi stocate numele şi prenumele: *nvarchar*
  - în care pot fi stocate texte lungi de maximum 40 litere/caractere: ***nvarchar(40)***
  - care acceptă valori nule – cu aceeaşi menţiune ca pentru coloanele dedicate numelor şi vârstei...
    - fiindcă este posibil ca toponimul acelei localităţi să nu fie cunoscut

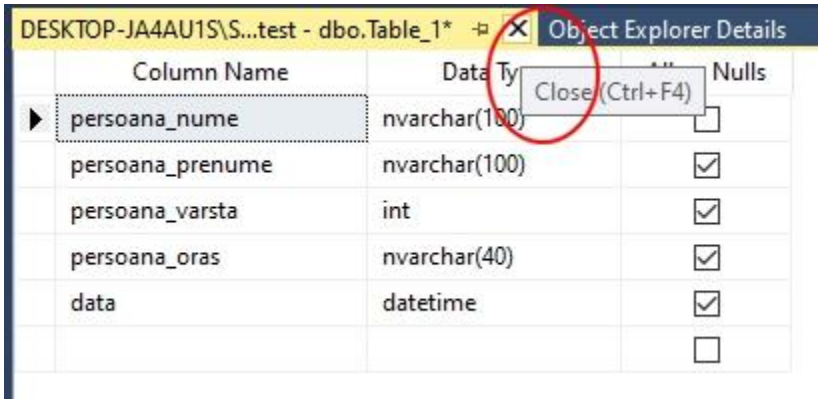
- "**data**": este o coloană...
  - unde vor fi trimise date calendaristice relevante pentru fiecare persoană
    - fiecare astfel de dată calendaristică se află pe aceeaşi linie (*row*), adică în aceeaşi înregistrare despre o persoană, cu numele, prenumele, vârsta şi localitatea corespunzătoare
  - care are tipul ***datetime*** – acesta reprezentând date calendaristice: anul, luna, ziua şi eventual ora, minutele, secundele, milisecundele
  - care acceptă valori nule – cu aceeaşi menţiune ca pentru coloanele dedicate numelor şi prenumelor
    - fiindcă este posibil ca nicio dată calendaristică relevantă pentru acea persoană să nu fie cunoscută

În SSMS, sub secţiunea dedicată proiectării coloanelor se observă o secţiune unde sunt prezentate şi solicitate alte proprietăţi ale coloanei selectate prin click-stânga:



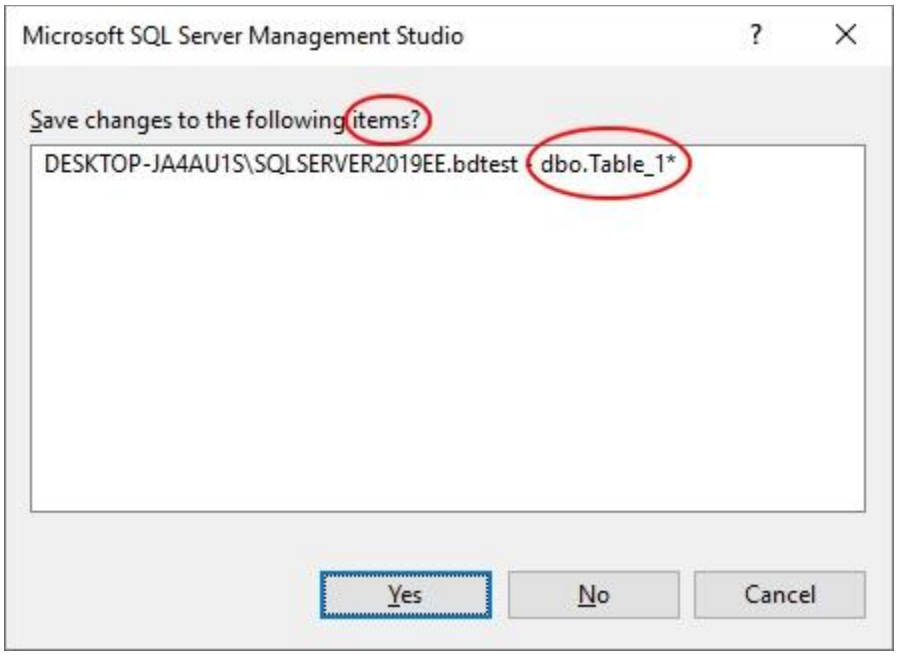
Despre (unele din) aceste proprietăţi se va vorbi mai târziu.
Deocamdată, pentru ceea ce vrem să facem cu acest tabel este suficient ce am făcut până acum.

Odată ce am terminat cu adăugarea şi proiectarea coloanelor acestui tabel de adăugat în baza de date, închidem procedura de proiectare a tabelului:



SSMS ne solicită confirmare pentru închiderea modului de proiectare a acestui tabel:

Zonele încercuite denotă, respectiv:
- "items": ceea ce poate fi salvat sau abandonat, în acest caz, tabelul pe care tocmai l-am proiectat
- "dbo": este numele acestei baze de date: cel ***generic, curent, cunoscut acum, în timpul procedurii acesteia, de către SQL Server***
- "Table_1": este numele tabelului pe care îl putem salva: cel generic, implicit propus în SSMS



Pentru a salva în forma curentă acest tabel de adăugat în baza de date - deci, conţinând coloanele pe care le-am definit mai devreme, apăsăm butonul marcat "Yes":

În dialogul următor se solicită numele (denumirea) acestui tabel:

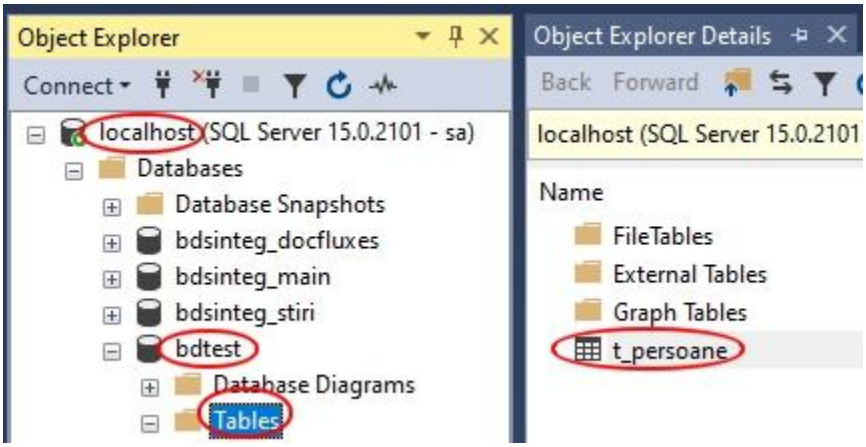
Specificăm un nume care să indice natura/caracteristicile informaţiilor care vor fi stocate în acest tabel:

De exemplu, dacă în acest tabel vor fi trimise informaţii despre persoane,
tabelul va putea primi numele: "t_persoane"

- "t" indicând un tabel în baza de date
- "_persoane" indicând, desigur, că tabelul va conţine informaţii despre persoane



Pentru a salva efectiv tabelul şi a-l adăuga în baza de date, apăsăm butonul marcat "OK", după care numele său va fi afişat în SSMS, în nodul "Tables" al bazei de date unde a fost adăugat:



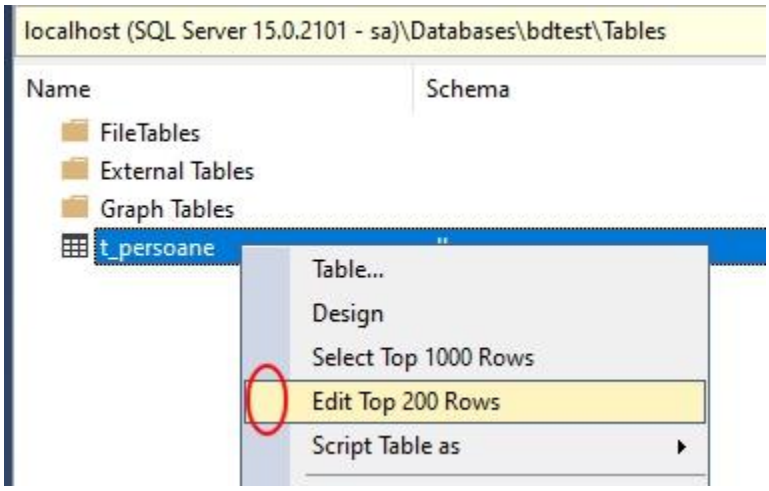
Dacă numele noului tabel nu apare nici în nodul "Tables", nici în secţiunea "Object Explorer Details" afişabilă cu tasta **F7**, atunci: click-stânga (adică, selectare) pe nodul "Tables" şi se apasă tasta **F5** ("*Refresh*" – reafişarea obiectelor din entitatea selectată).

Acum tabelul nou creat şi adăugat/plasat în baza de date este gol de conţinut relevant pentru utilizator.

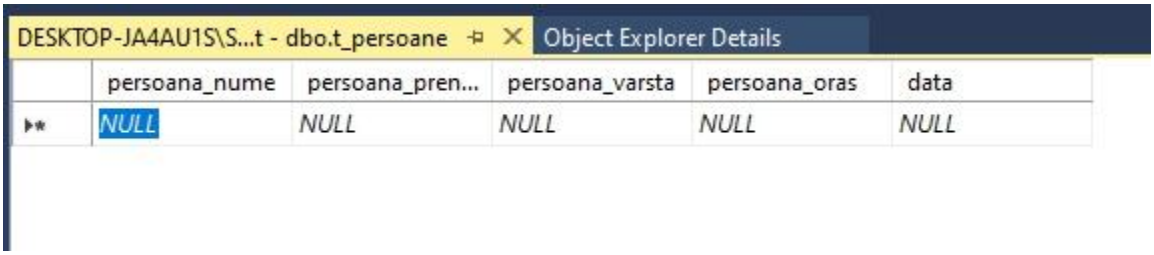
Pentru a servi totuşi unui scop, tabelul trebuie să conţină fie şi o singură înregistrare.

Să punem câteva înregistrări în acest tabel.

Click-dreapta pe numele tabelului, apoi se selectează opţiunea "Edit Top 200 Rows":



SSMS ne oferă posibilitatea de a **edita** – iar acum, tabelul fiind gol, de a **adăuga manual** înregistrări (linii, *Row*s).
Oferta este de a edita primele 200 înregistrări; acum, tablelul fiind gol, nu ne interesează acest număr.



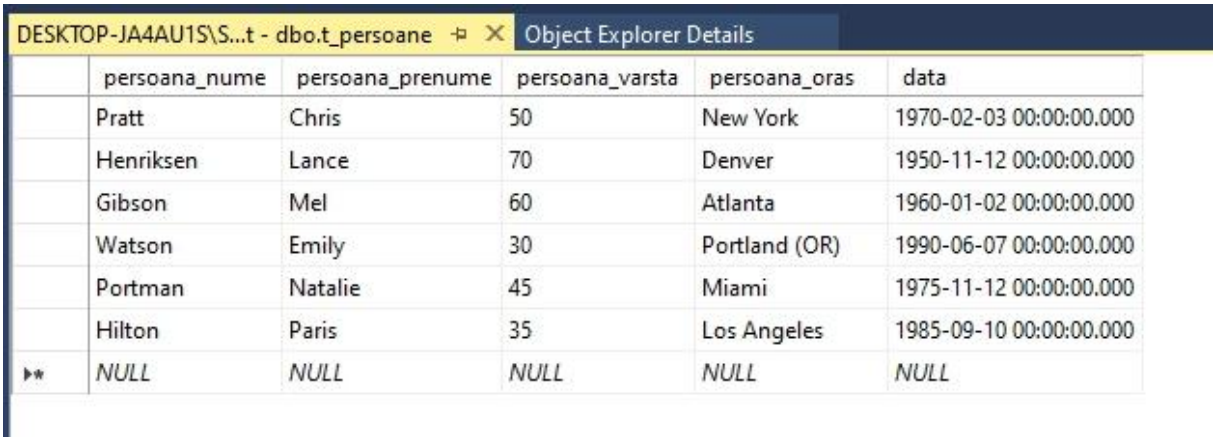
Vedem cum celulele goale sunt marcate "NULL": anume, in acele celule nu se află nimic.

Aceasta nu înseamnă că acele celule acceptă valori nule, ci numai că ***acum*** nu conţin nimic.

Să scriem detaliile cunoscute despre câteva persoane:

Scriem în fiecare celulă particula de informaţie care ***teoretic*** trebuie să se afle acolo –
adică, putem scrie orice indiferent de numele unei coloane;
de exemplu – SQL Server nu verifică dacă un număr
plasat pe o coloană pentru preţuri reprezintă într-adevăr un preţ:

contează numai tipul particulelor de informaţie:
adică, nu poate fi scris un text de tip nvarchar într-un câmp de tip datetime


DESKTOP-JA4AU1S\S...t - dbo.t_persoane | Object Explorer Details

	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
	Pratt	Chris	50	New York	1970-02-03 00:00:00.000
	Henriksen	Lance	70	Denver	1950-11-12 00:00:00.000
	Gibson	Mel	60	Atlanta	1960-01-02 00:00:00.000
	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000
	Portman	Natalie	45	Miami	1975-11-12 00:00:00.000
	Hilton	Paris	35	Los Angeles	1985-09-10 00:00:00.000
▶*	NULL	NULL	NULL	NULL	NULL

SSMS va atenţiona cu mesaj de eroare atunci când:

- ***într-o coloană scriem un text care nu este acceptabil în acea coloană...***
  - ***de exemplu, aici, dacă scriem "abc" în coloana vârstei care are tip numeric întreg***

- ***vrem să trecem pe linia următoare (să scriem ceva în înregistrarea ce-i urmează celei curente), dar nu am scris nimic în coloanele care nu acceptă valori nule (în acest caz, coloana numelui)***

După ce adăugăm astfel, manual, câteva înregistrări într-un tabel al unei bazei de date,
tabelului şi într-adevăr bazei de date li se dau un rost şi sunt gata de treabă.

Extragerea de informaţii dintr-o bază de date: comanda SELECT

Devreme ce ştim că într-o bază de date se află informaţii de interes şi totodată cunoaştem structura acelei baze de date –

- numele tabelelor de interes şi
- numele câmpurilor de interes din fiecare tabel

putem extrage informaţii din acea bază de date.

Comanda, ori instrucţiunea din limbajul SQL care foloseşte la această procedură este

SELECT

Forma, ori sintaxa comenzii SELECT
prin care sunt accesate ***toate înregistrările dintr-un tabel*** este:

SELECT
*
FROM
numele_tabelului

Caracterul *
denotă toate înregistrările din tabel.

Acest caracter, când este folosit în domeniul *software* este cunoscut şi ca *"wildcard"* –
adică, ceva care poate fi folosit în locul tuturor celorlaltor lucruri
din aceeaşi colecţie, ori în acelaşi grup

Acest caracter are acelaşi rol ca în Controalele din Visual Studio®™:
OpenFileDialog şi *SaveFileDialog*
unde când este scris în caseta dedicată numelui fişierului de selectat
determină afişarea tuturor fişierelor din directorul/folderul prezentat în acel dialog

Această formă a lui **SELECT** trebuie utilizată cu moderaţie şi judicios,
întrucât un tabel poate conţine foarte multe (milioane) înregistrări
şi ca atare accesarea tuturor înregistrărilor poate dura mai mult decât este convenabil.

Pentru a accesa ***anumite câmpuri din toate înregistrările*** se foloseşte forma/sintaxa:

SELECT
ValoareaDinCampulNumitCutare1, V*aloareaDinCampulNumitCutare2*, ..., V*aloareaDinCampulNumitCutareN*
FROM
NumeleTabelului

Adică, dintr-un tabel vor fi selectate/extrase/accesate/prezentate:
numai acele particule de informaţie *din toate înregistrările*
aflate pe coloanele de tabel (în câmpurile) enumerate după **SELECT**,

Poate fi accesat şi un singur câmp, bineînţeles reducând lista numelor câmpurilor la un singur element:

SELECT
ValoareaDinCampulNumitCutare
FROM
NumeleTabelului

*Cu formele de mai sus ale comenzii **SELECT** –*
şi chiar solicitând valorile numai dintr-un câmp sau numai din câteva câmpuri –
încă *ne referim la toate înregistrările din tabel*

Unele înregistrări pot fi duplicat,
adică toate câmpurile unei/unor înregistrări au aceleaşi valori
cu valorile tuturor câmpurilor altei/altor înregistrări.

De asemenea, pe o coloană a unui tabel (într-un câmp) pot fi valori duplicat –
indiferent de valorile celorlaltor câmpuri care pot face duplicat toată înregistrarea.

Pentru preluarea numai a înregistrărilor distincte sau numai a valorilor distincte din câmpuri se foloseşte clauza:

DISTINCT

Astfel, pentru aceste cazuri sintaxa comenzii SELECT este:

SELECT DISTINCT
...

Pentru a reduce mai mult setul înregistrărilor sau al unor valori anume,
în sintaxa comenzii **SELECT** se includ
criterii de filtrare sau **de evaluare**
a rezultatelor – anume, prin clauza:

WHERE

Forma, ori sintaxa comenzii **SELECT**
prin care sunt accesate
toate înregistrările dintr-un tabel
care se supun anumitor criterii / care îndeplinesc anumite condiţii
este:

SELECT
*
FROM
NumeleTabelului
WHERE
ValoareaDinCampulNumitCutare1 /este evaluată /

De exemplu, lucrând cu baza de date creată mai devreme:

Din tabelul "t_persoane"
vrem să accesăm *numai acele înregistrări*
care pe coloana/în câmpul "persoana_varsta"
au valori mai mici de 60.

Vom folosi această instrucţiune:

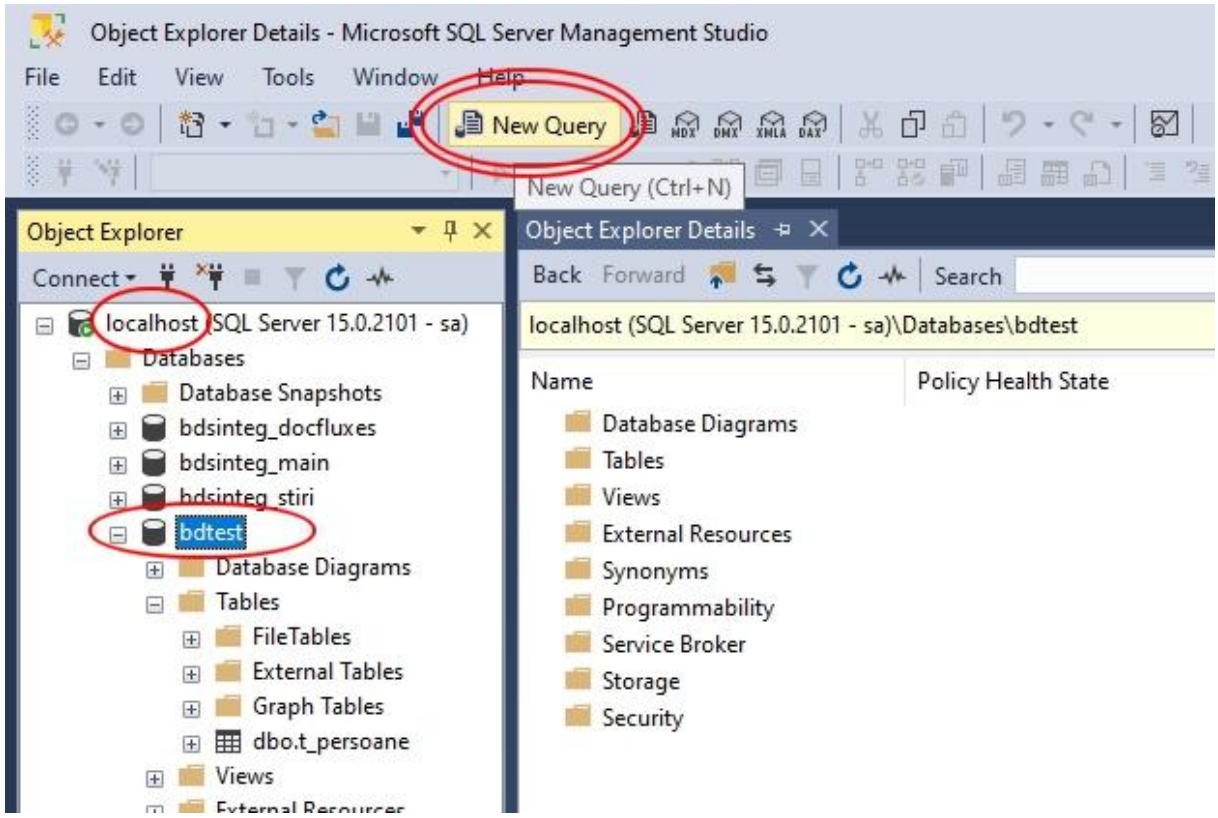
SELECT
*
FROM
t_persoane
WHERE
persoana_varsta < 60

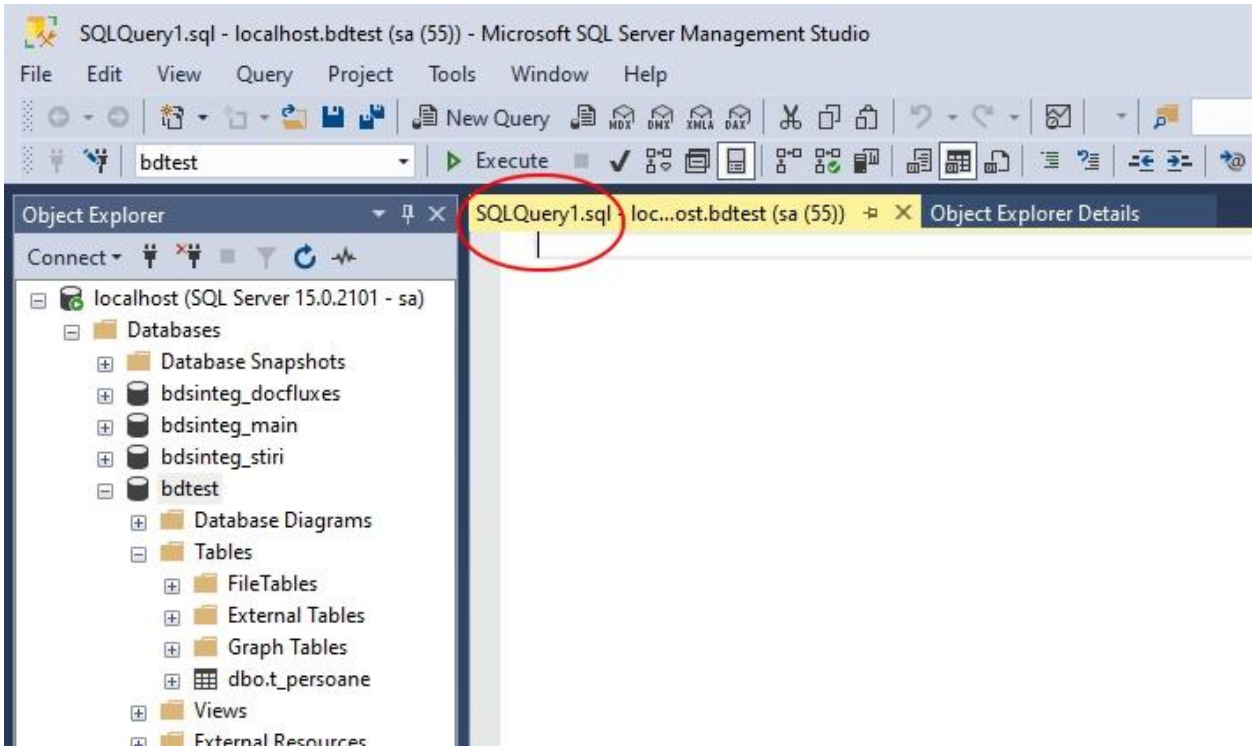
În SSMS, aceasta se face astfel:

- În *TreeView*-ul din stânga:

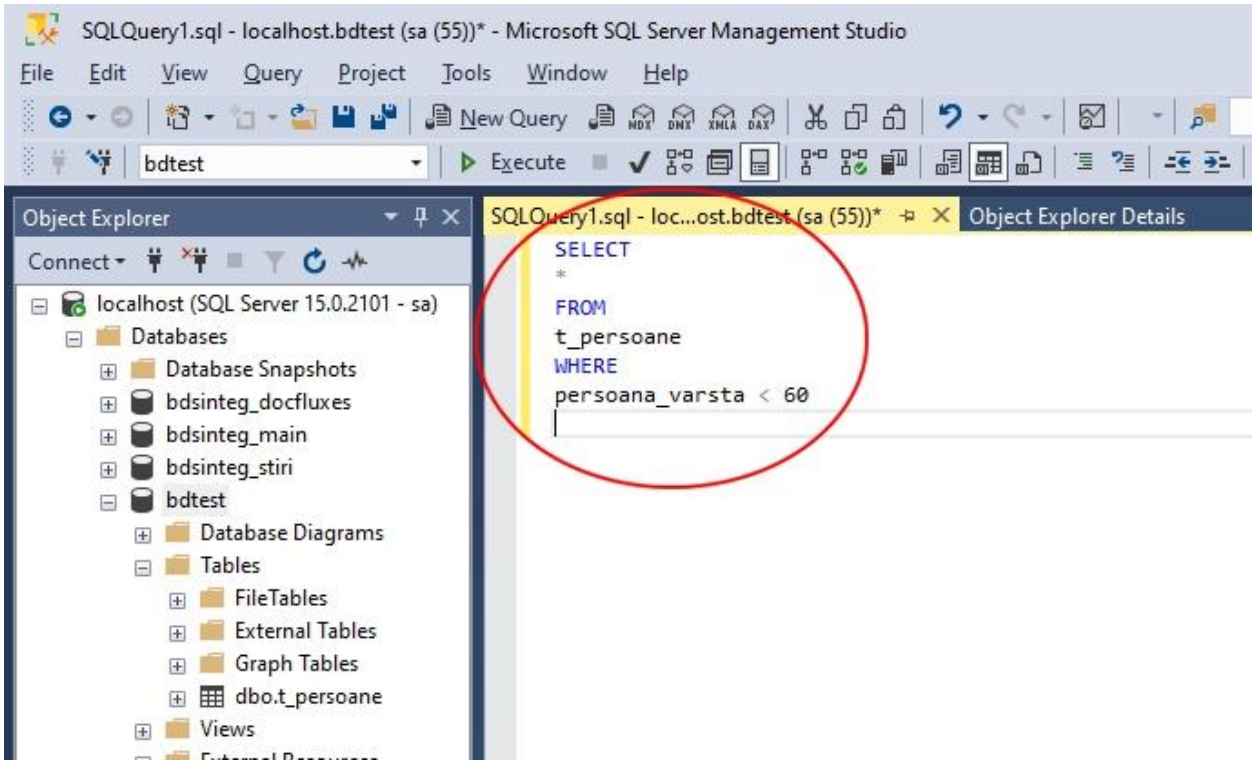
se selectează (click-stânga) baza de date de interes – în acest caz, "bdtest"
(pentru siguranţă, se selectează şi numele tabelului din care vrem să extragem înregistrări –
în acest caz, "t_persoane"

- Pe bara SSMS de comenzi se caută butonul marcat "New Query" ("Interogare Nouă") şi se apasă:

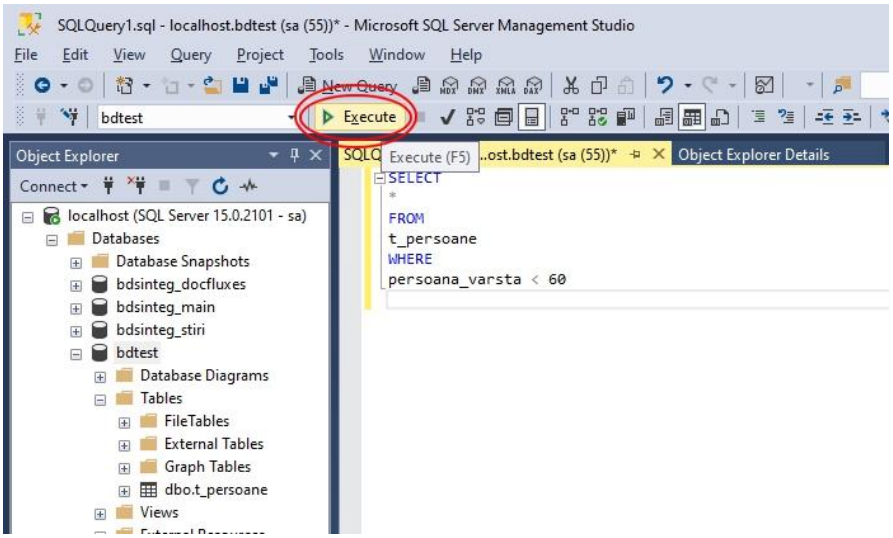

Object Explorer Details - Microsoft SQL Server Management Studio
File Edit View Tools Window Help
New Query
New Query (Ctrl+N)
Object Explorer
Connect
localhost (SQL Server 15.0.2101 - sa)
Databases
Database Snapshots
bdsinteg_docfluxes
bdsinteg_main
bdsinteg_stiri
bdtest
Database Diagrams
Tables
FileTables
External Tables
Graph Tables
dbo.t_persoane
Views
Object Explorer Details
Back Forward Search
localhost (SQL Server 15.0.2101 - sa)\Databases\bdtest
Name
Policy Health State
Database Diagrams
Tables
Views
External Resources
Synonyms
Programmability
Service Broker
Storage
Security


SQLQuery1.sql - localhost.bdtest (sa (55)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query
bdtest
Execute
Object Explorer
Connect
localhost (SQL Server 15.0.2101 - sa)
Databases
Database Snapshots
bdsinteg_docfluxes
bdsinteg_main
bdsinteg_stiri
bdtest
Database Diagrams
Tables
FileTables
External Tables
Graph Tables
dbo.t_persoane
Views
SQLQuery1.sql - loc...ost.bdtest (sa (55))
Object Explorer Details

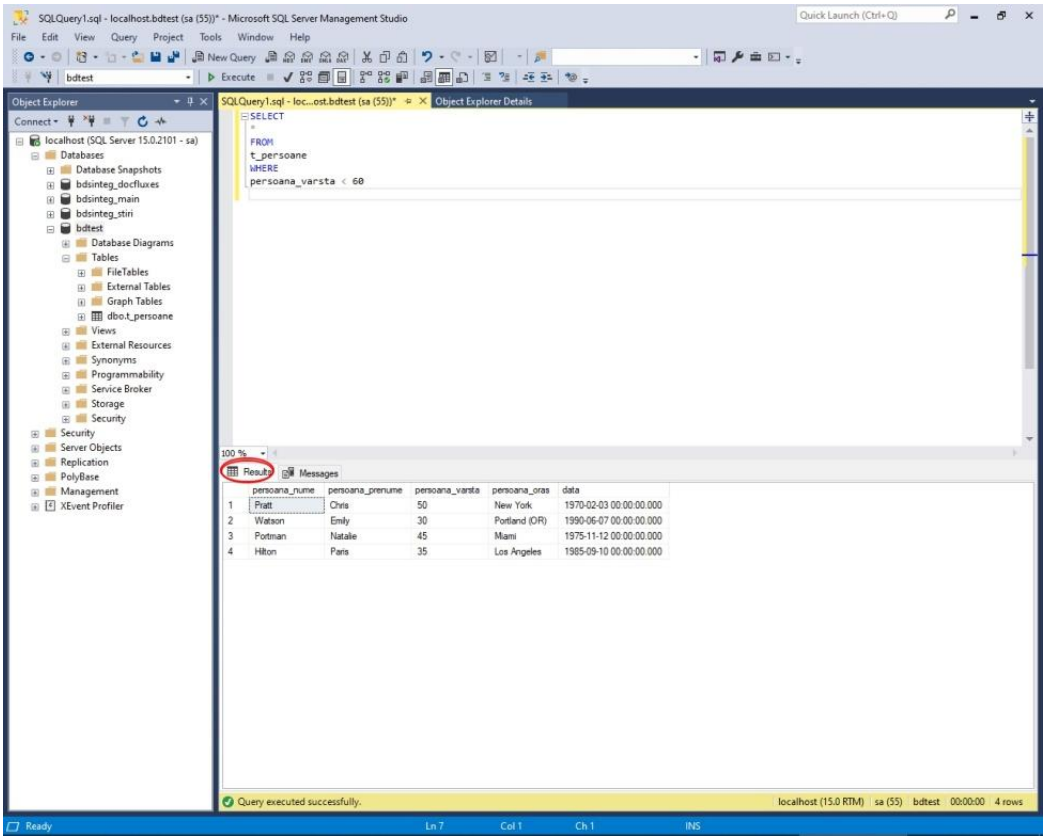
Scriem "interogarea" – aici, instrucţiunea **SELECT** de mai sus:



Pentru a trimite interogarea către SQL Server şi a obţine rezultatul său:
Tot pe bara SSMS de comenzi, apăsăm butonul marcat "Execute":
Sau apăsăm tasta **F5**:



Rezultatul interogării va fi afişat jos, sub secţiunea din SSMS unde am scris instrucţiunea SQL:



Încercaţi cu diverse criterii, de exemplu evaluări ale unor câmpuri de tip text, ca:

```
SELECT
*
FROM
t_persoane
WHERE
persoana_prenume =  'emily'
```

În acest exemplu se vede cum pentru prenume nu am folosit majuscula;
totuşi, acea înregistrare va fi afişată.

Aşa ceva, însă, nu este recomandat, fiindcă:
- se poate să nu ştim cum va interpreta acest criteriu dialectul SQL folosit de către serverul SQL
- în general, nu cunoaştem în întregime forma înregistrărilor/conţinutul din/acelui tabel, aşa că
- nu vrem să riscăm obţinerea unor rezultate false sau/şi inutile
sau/şi care pot cauza erori în cascadă

La fel ca în mediul de programare Visual Studio®™, într-o interogare pot fi incluse **comentarii**.
În SQL, este bine să comentăm măcar instrucţiunile complexe sau folosite rareori.

În SSMS, o linie de comentariu începe cu două semne minus, ca în exemplul următor:

```
-- vom cauta textul 'emily' in campul persoana_prenume:
SELECT
*
FROM
t_persoane
WHERE
persoana_prenume =  'emily'
-- putem cauta 'emily' in alt camp care se poate sa nu contina acest text
-- de exemplu, in campul persoana_varsta
-- caz in care SQL Server nu va intoarce niciun rezultat
-- adica nicio inregistrare - in cazul comenzii SELECT
```

***Mai multe criterii de evaluare pot fi ală*turate**:

- folosind cuvintele-cheie:
  - **AND**
  - **OR**
- şi încadrând fiecare evaluare în paranteze rotunde ( )

De exemplu, interogarea care solicită înregistrările unde:

- valoarea câmpului *persoana_varsta* este mai mică decât 60
- **ŞI – AND**
- valoarea câmpului *persoana_prenume* este "Chris"

va arăta astfel:
se observă că:

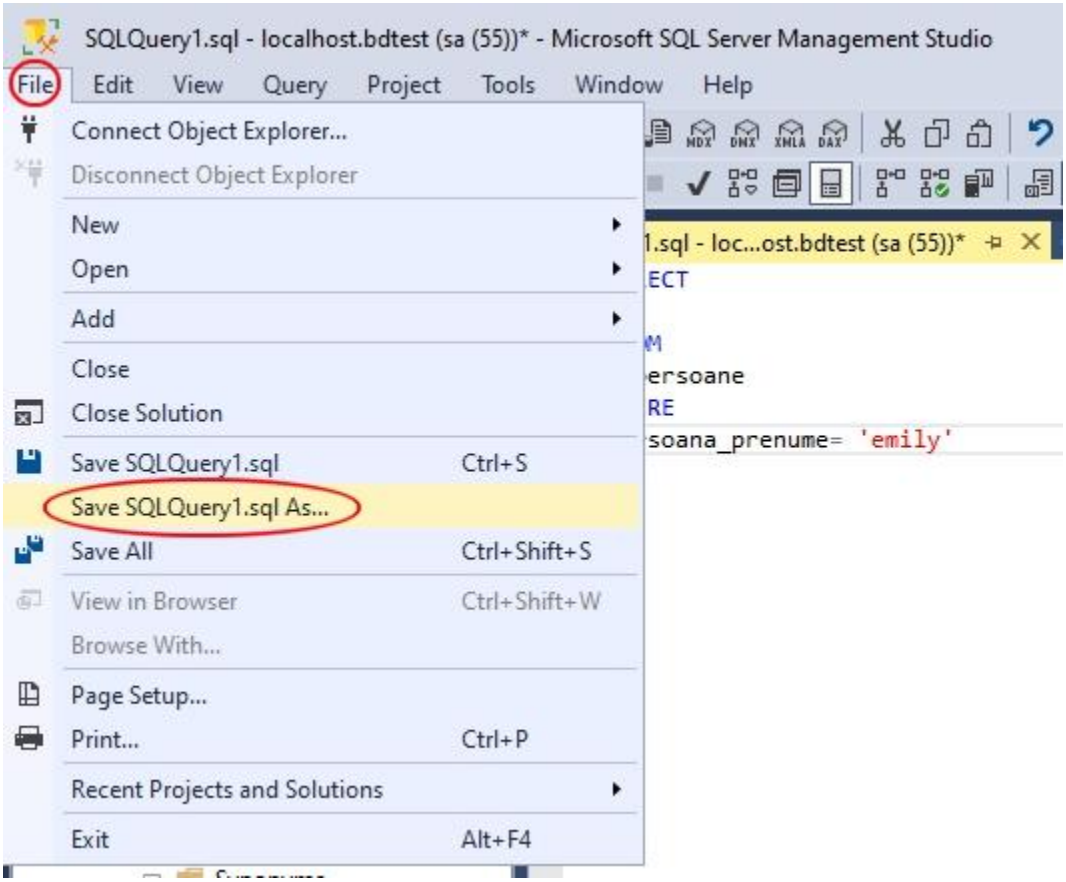
- fiecare evaluare este cuprinsă între paranteze şi
- ***ambele (toate) evaluări(le) sunt cuprinse între paranteze***

```
SELECT
*
FROM
t_persoane
WHERE
(
(persoana_varsta < 60)
AND
(persoana_prenume = 'Chris')
)
```

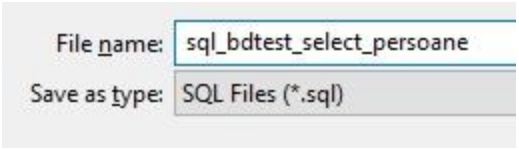
- **AND determină întoarcerea unui set relativ mic de înregistrări**
  - **fiindcă este restrictiv, impunând îndeplinirea tuturor condițiilor**

- **OR determină întoarcerea unui set relativ mare de înregistrări (mai mare decât o face AND)**
  - **fiindcă este mai tolerant, verificând îndeplinirea uneia sau unor anumite condiții din mai multe**

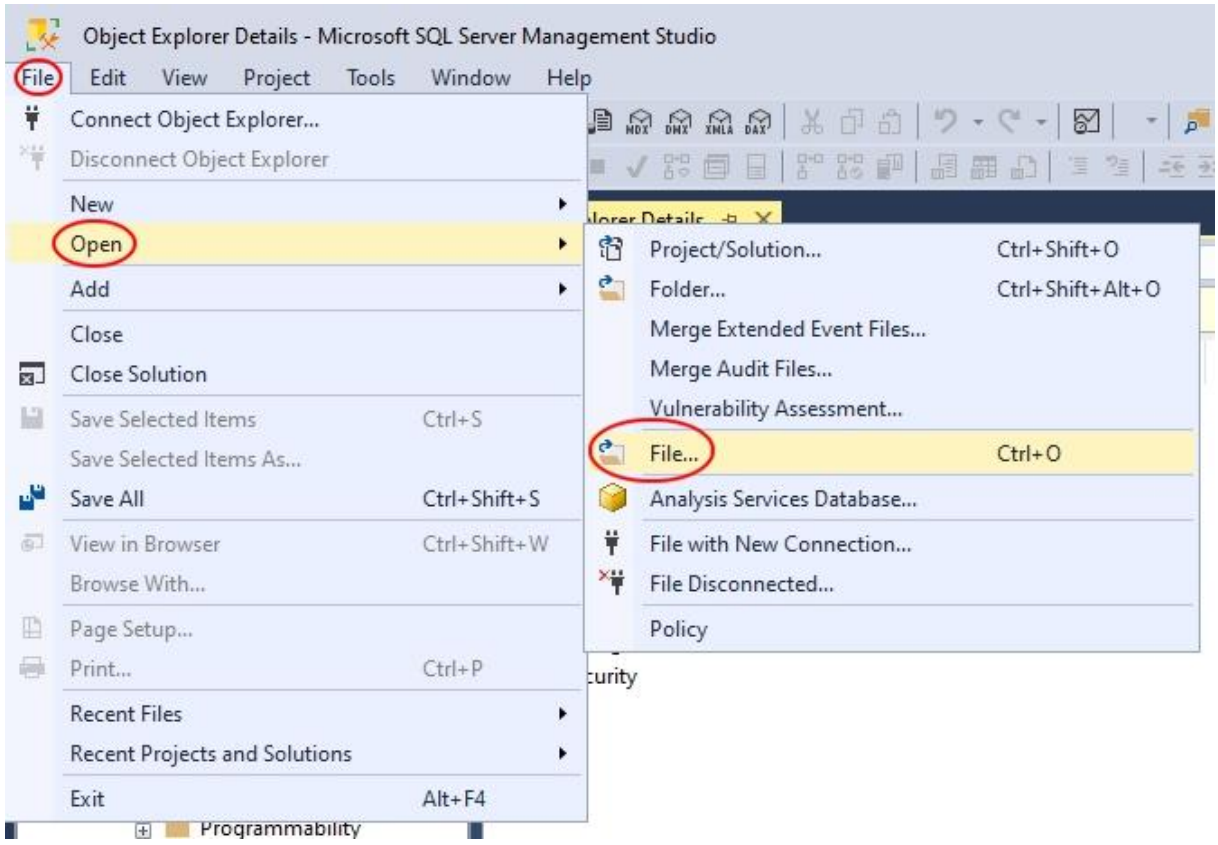
O interogare scrisă în SSMS poate fi salvată pentru ca mai târziu să fie refolosită sau modificată, astfel:
Din meniul "File" al SSMS se alege intrarea "Save SQLQuery....sql As...":

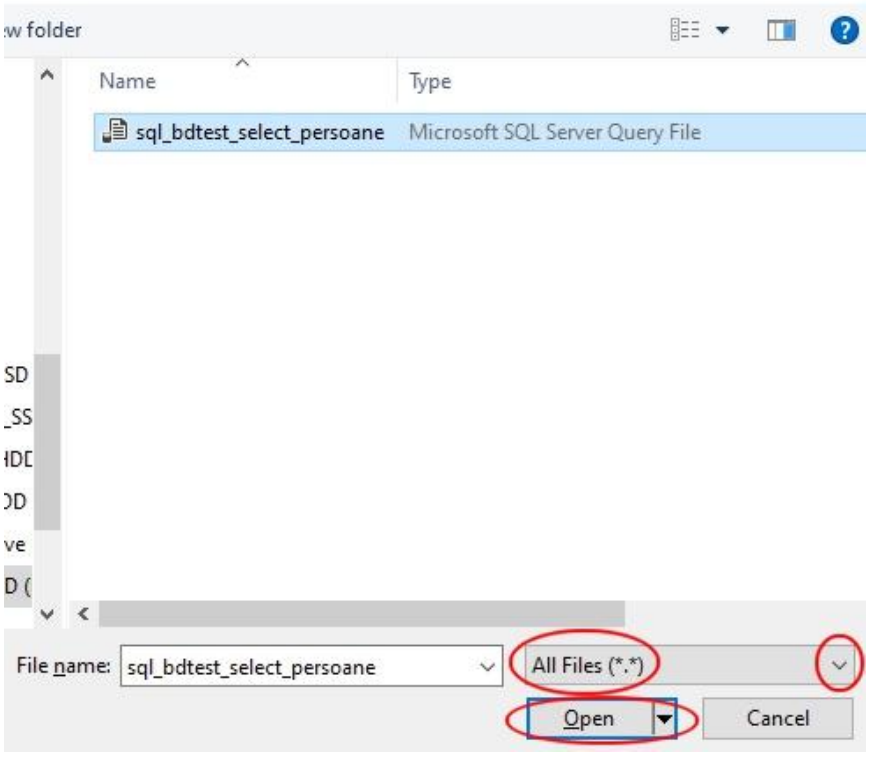


În *SaveFileDialog*-ul care solicită numele fișierului unde va fi salvată interogarea se specifică acel nume:



Mai târziu, în SSMS putem încărca pentru rulare şi/sau modificare interogări existente:





Până acum am făcut interogări pe o bază de date al cărui nume îl cunoaştem – fiindcă
am lucrat tot timpul cu baza de date şi tabelele create împreună, într-o singură sesiune.

Însă, după ce închidem SSMS şi îl deschidem
pentru a refolosi sau modifica o interogare adecvată pentru o anumită bază de date,
este foarte probabil ca SSMS să nu poată executa interogarea fiindcă
nu este cunoscut numele bazei de date pe care se face acea interogare.

Pentru a ne asigura că o interogare se face numai pe o anumită bază de date:

Înantea oricărei comenzi SQL o punem pe aceasta:

USE *numelebazeidedate*

În acest caz, această comandă va fi:

USE bdtest

Odată cu această nouă comandă am mărit numărul lor la două.

Pentru a trimite serverului SQL interogări formate din mai multe comenzi/instrucţiuni:
după fiecare instrucţiune se pune semnul de punct şi virgulă ;

Astfel că modul corect întotdeauna de a scrie interogarea de mai sus –
în care am solicitat înregistrările unde valoarea din/câmpul/ui persoana_varsta este mai mică decât 60
este acesta:

```
USE bdtest;
-- ne-am asigurat că instrucţiunile urmatoare, din aceasta interogare
-- vor fi executate pe baza de date bdtest
--
SELECT
*
FROM
t_persoane
WHERE
persoana_varsta < 60;
```

Interogarea poate sau nu să fie terminată cu ;

Acum putem salva această interogare explicită pentru a o refolosi oricând după pornirea SSMS.

În timp ce modificăm o interogare SQL, ne asigurăm că nu ştergem
comanda **USE** *numelebazeidedate*;
care trebuie să existe în orice interogare.

Operaţiuni simple cu informaţia dintr-o bază de date: TOP, MIN, MAX

Ne reamintim că o înregistrare (zisă şi *"Record"*) este compusă din (valori situate în) câmpuri (*Field*s).
Ori, cu aceeaşi semnificaţie – că pe o linie (*"Row"*) se află valori de pe coloane (*Column*s).

Valorile dintr-un câmp (de pe o coloană) având aproape orice tip
pot fi supuse unor operaţiuni care separă...

- unele valori de restul valorilor din acel câmp
- înregistrările care conţin anumite valori de restul care nu conţin acea valoare în acel câmp.

Cele mai simple astfel de operaţiuni sunt:

- aflarea primelor *N* valori dintr-un câmp (de pe o coloană)
- aflarea celei mai mici valori dintr-un câmp (de pe o coloană)
    - aceasta este, desigur, aplicabilă tipurilor celor mai frecvent folosite:
      numeric, text, dată calendaristică
- aflarea celei mai mari valori dintr-un câmp (de pe o coloană)

Sintaxa fiecărei din aceste comenzi este, respectiv:

- aflarea primelor *N* valori dintr-un câmp (de pe o coloană)

TOP *NumeleCampului*

- aflarea celei mai mici valori dintr-un câmp (de pe o coloană):

MIN (*NumeleCampului*)

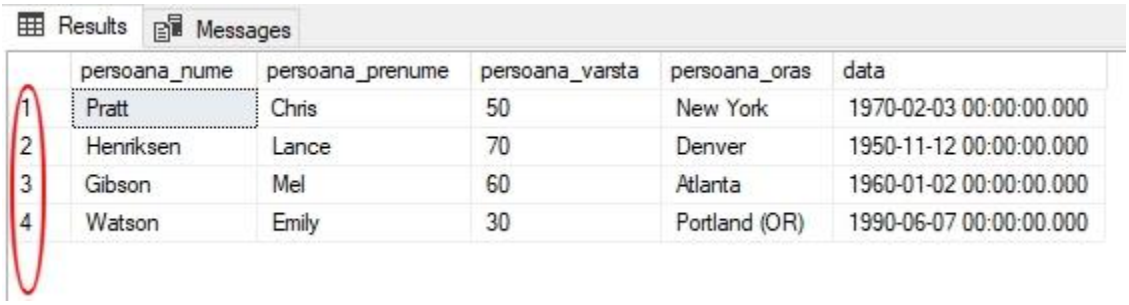
- aflarea celei mai mari valori dintr-un câmp (de pe o coloană):

MAX (*NumeleCampului*)

Să scriem astfel de interogări adresate bazei noastre de date, *bdtest*:

1. Solicităm primele 4 înregistrări complete (caracterul *, toate câmpurile) din tabelul *t_persoane*:

```
USE bdtest;
SELECT
TOP 4 *
FROM
t_persoane
```

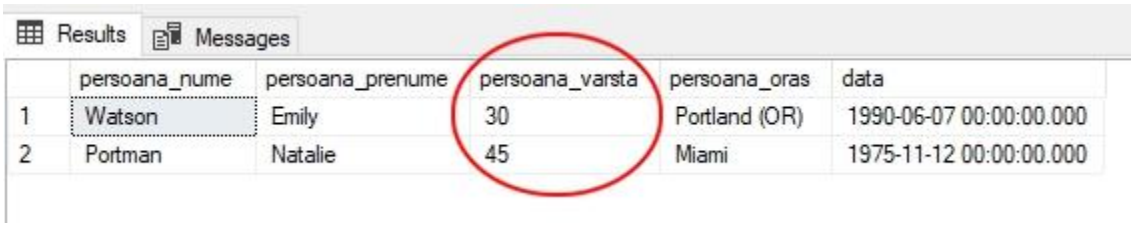


Results | Messages

	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
1	Pratt	Chris	50	New York	1970-02-03 00:00:00.000
2	Henriksen	Lance	70	Denver	1950-11-12 00:00:00.000
3	Gibson	Mel	60	Atlanta	1960-01-02 00:00:00.000
4	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000

2. Solicităm primele 2 înregistrări complete (caracterul *) din tabelul *t_persoane* unde valoarea din câmpul *persoana_varsta* este mai mică decât 50):

```
USE bdtest;
SELECT
TOP 2 *
FROM
t_persoane
WHERE
persoana_varsta < 50
```

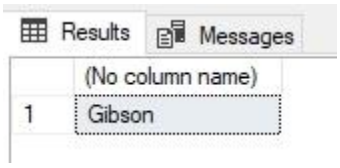


Results | Messages

	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
1	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000
2	Portman	Natalie	45	Miami	1975-11-12 00:00:00.000

3. Solicităm cea mai mică valoare din câmpul *persoana_nume* al tabelului *t_persoane*:

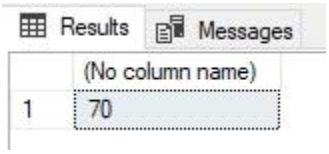
```
USE bdtest;
SELECT
MIN (persoana_nume)
FROM
t_persoane
```



Aici vedem titlul coloanei care conţine acea valoare ca fiind "(No column name)".
Aceasta denotă faptul că în interogare nu am specificat un nume pentru coloana cu rezultate.
Deocamdată, asta nu contează.

4. Solicităm cea mai mare valoare din câmpul *persoana_varsta* al tabelului *t_persoane*:

```
USE bdtest;
SELECT
MAX (persoana_varsta)
FROM
t_persoane
```



Gruparea unor informaţii: GROUP BY, SUM, AVG

Adesea, unele informaţii aflate într-un tabel pot fi **grupate** în colecţii/mulţimi/seturi/liste de elemente asemănătoare dar care îndeplinesc anumite condiţii.

Întreg grupul de valori/înregistrări/rezultate va fi asociat unuia sau mai multor câmpuri ale cărui/căror valori vor fi evaluate.

De exemplu:
- mai multe zile – ca date calendaristice – cu însemnătate pot fi grupate după luna care le conţine
- mai multe valori numerice provenind din aceeaşi sursă pot fi grupate după numele sursei (cum este setul valorilor temperaturii aerului măsurate de staţia cutare)

Funcţia sau clauza SQL care grupează valori este:

GROUP BY

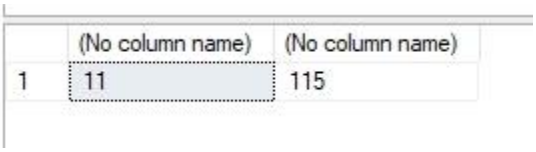
În SSMS, deschidem fişierul cu una din interogările de mai sus.

Scriem o interogare cu care vrem să obţinem:
- o înregistrare care să conţină:
  - luna 11
  - **suma** vârstelor celor născuţi în luna 11, ştiind că:
- vârstele se află în câmpul "*persoana_varsta*"
- luna anului se află în câmpul "*data*"
- câmpurile de interes fiind în tabelul "*t_persoane*"

Aşa că scriem:

```
USE bdtest;
SELECT
MONTH (data),
SUM(persoana_varsta)
FROM
        t_persoane
WHERE
        MONTH (data) = 11
GROUP BY
        MONTH(data)
```

Trimitem această interogare serverului SQL: apăsăm tasta **F5**. Rezultatul:



	(No column name)	(No column name)
1	11	115

În această interogare am folosit cuvântul-cheie sau operatorul sau funcţia

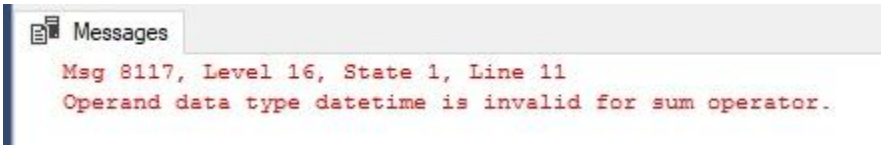
SUM

care efectuează suma valorilor câmpului având numele specificat/primit ca argument (adică, aflat între paranteze).

Această funcţie necesită neapărat asociere cu clauza GROUP BY,
adică nu se poate solicita **SUM**(...) fără a grupa câmpurile referite, cum se arată în interogarea de mai sus.

Totodată, această funcţie poate acţiona asupra câmpurilor de tip numeric.

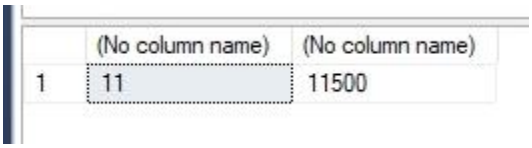
Nu se poate face **SUM** pe valori de tip text, nici pe unele de tip dată calendaristică (*datetime*);
iată mesajul de eroare din partea SSMS după ce în interogarea de mai sus am scris**SUM**(data):



Valoarea rezultată în urma **SUM**(...) fiind una numerică,
se poate modifica dintru bun început - în interogarea care conţine o astfel de operaţiune
de exemplu, aici înmulţim cu 100 valoarea rezultată din **SUM**(...):

```
USE bdtest;
SELECT
MONTH (data),
100 * SUM(persoana_varsta)
FROM
        t_persoane
WHERE
        MONTH (data) = 11
GROUP BY
        MONTH(data)
```

Cu rezultatul aşteptat:



	(No column name)	(No column name)
1	11	11500

Acum scriem o interogare cu care vrem să obţinem:

- o înregistrare care să conţină:
  - **media aritmetică** a vârstelor celor născuţi în luni mai mici decât 11,

    ştiind la fel ca la interogarea precedentă că:
- vârstele se află în câmpul "*persoana_varsta*"
- luna anului se află în câmpul "*data*"
- câmpul de interes fiind în tabelul "*t_persoane*"

Pentru aceasta vom folosi cuvântul-cheie sau operatorul sau funcţia

AVG

care efectuează media aritmetică a valorilor ***câmpului de tip numeric*** având numele specificat – primit ca argument (adică, aflat între paranteze).

Acestei funcţii i se aplică observaţiile la **SUM**.

Nu vom folosi clauza **GROUP BY**, fiindcă *teoretic (adică, presupunem că nu ştim cum)* nu ne referim la o particulă anume din *toate* înregistrările întoarse de SQL Server căreia să îi asociem o singură valoare produsă de **AVG**

Aşa că scriem:

```
USE bdtest;
SELECT
AVG (persoana_varsta)
FROM
        t_persoane
WHERE
        MONTH (data) < 11
```

Trimitem această interogare serverului SQL: apăsăm tasta **F5**. Rezultatul:

	(No column name)
1	43

Astfel de interogări **SELECT** relativ complexe – care solicită informaţii *grupate*, sau *agregate*, la fel cu cele simple **SELECT** ... **FROM** –
pot include mai multe criterii **WHERE** de evaluare, desigur alăturate/alipite/înlănţuite cu **AND** şi **OR**.

Înregistrări similare: LIKE

*Unele înregistrări **pot semăna** cu altele* fie din acelaşi tabel, fie din alte tabele.

Căutarea unor astfel de înregistrări se face folosind clauza de ***evaluare***:

LIKE

Fiind o clauză de evaluare, **LIKE** este neapărat asociată clauzei **WHERE**.

LIKE primeşte ca argument, ori parametru
o valoare de un tip acceptat, ori inteligibil de către SQL Server
pe care o va căuta în câmpurile specificate ale înregistrărilor de interes
care înregistrări se poate să fie filtrate **şi cu alte**
clauze **WHERE-AND/OR**

Sintaxa generică este:

LIKE(*ValoareDeUnTipAcceptatInSQLServer*)

Particulele de ***text*** astfel căutate trebuie – în această versiune de SQL Server şi în unele anterioare) –
să fie marcate cu caracterul apostrof (').

Textul care urmează, care precedă sau care conţine particula de text căutată este indicat de caracterul procent (%).

Să vedem **LIKE** în acţiune, pe baza de date "*bdtest*" pe care am creat-o mai devreme.

1. Scriem o interogare cu care vrem să obţinem:
- toate înregistrările care ***la sfârşitul***
valorii-text din câmpul "*persoana_nume*" să conţină
particula de text "**on**"
- câmpul de interes fiind în tabelul "*t_persoane*"

Se observă cum ***particula căutată la sfârşitul valorii-text a acelui câmp***:
- ***este precedată de caracterul % care denotă un text oarecare precedând particula căutată***
- ***nu este urmată de aceeaşi secvenţă, ceea ce înseamnă că se caută chiar la sfârşitul textului***
- ***textul este specificat prin-încadrat de apostrof, nu de ghilimele***

```
USE bdtest;
SELECT
*
FROM
t_persoane
WHERE
persoana_nume
LIKE ('%on')
```

	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
1	Gibson	Mel	60	Atlanta	1960-01-02 00:00:00.000
2	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000
3	Hilton	Paris	35	Los Angeles	1985-09-10 00:00:00.000

2. Scriem o interogare cu care vrem să obţinem:
- toate înregistrările care ***la începutul***
valorii-text din câmpul "*persoana_nume*" să conţină
particula de text "**P**"
- câmpul de interes fiind în tabelul "*t_persoane*"

Se observă cum ***particula căutată la începutul valorii-text a acelui câmp***:
- ***este urmată de caracterul % care denotă un text oarecare urmând particula căutată***
- ***nu este precedată de acest caracter, ceea ce înseamnă că se caută chiar la începutul textului***

```
USE bdtest;
SELECT
*
FROM
t_persoane
WHERE
persoana_nume
LIKE ('P%')
```



	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
1	Patt	Chris	50	New York	1970-02-03 00:00:00.000
2	Portman	Natalie	45	Miami	1975-11-12 00:00:00.000

3. Scriem o interogare cu care vrem să obţinem:
- toate înregistrările care ***oriunde***
  în valoarea-text din câmpul "*persoana_nume*" să conţină ***oriunde***
  particula de text "**at**"
- câmpul de interes fiind în tabelul "*t_persoane*"

Se observă cum ***particula căutată oriunde în valoarea-text a acelui câmp***:
- ***se află între două caractere % care denotă un text oarecare conţinând particula căutată***

```
USE bdtest;
SELECT
*
FROM
t_persoane
WHERE
persoana_nume
LIKE ('%at%')
```

	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
1	Pratt	Chris	50	New York	1970-02-03 00:00:00.000
2	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000

4. Scriem o interogare cu care vrem să obţinem:
- toate înregistrările care:
    - ***oriunde*** în valoarea-text din câmpul "*persoana_nume*" să conţină particula de text "or"
    - **SAU (OR)**
    - ***oriunde*** în valoarea-text din câmpul "*persoana_oras* " să conţină particula de text "or"
- câmpul de interes fiind în tabelul "*t_persoane*"

Se observă cum ***particula căutată oriunde în valoarea-text a acelor câmpuri***:
- ***se află între două caractere % care denotă un text oarecare conţinând particula căutată***

Desigur, "or" este textul căutat, *nu* operatorul **OR**.

Textul căutat cu **LIKE** poate fi un cuvânt-cheie SQL, fiindcă apostrofurile îl fac un parametru/argument de tip text, întocmai cum este aşteptat.

Aici căutăm acelaşi text în ambele câmpuri.
Bineînţeles că putem căuta orice text.

```
USE bdtest;
SELECT
*
FROM
t_persoane
WHERE
(
(persoana_nume LIKE('%or%'))
OR
(persoana_oras LIKE('%or%'))
)
```

	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
1	Pratt	Chris	50	New York	1970-02-03 00:00:00.000
2	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000
3	Portman	Natalie	45	Miami	1975-11-12 00:00:00.000

Ordonarea înregistrărilor rezultate după o interogare SELECT: ORDER BY

Înregistrările rezultate din executarea unei comenzi **SELECT** pot fi afişate/prezentate//accesibile ordonate:

- în cazul câmpurilor de tip text: alfabetic:
  - de la caracterul cu cel mai mic cod *Unicode* sau *ASCII*
    - până la caracterul cu cel mai mare cod – adică **ASC**endent,
    - sau invers – adică **DESC**endent
- în cazul câmpurilor de tip numeric:
  - de la cea mai mică spre cea mai mare valoare – adică **ASC**endent,
    - sau invers – **DESC**endent
- în cazul câmpurilor de tip dată calendaristică, *datetime*:
  - de la cea mai de demult dată calendaristică până la cea mai recentă – adică **ASC**endent,
    - sau invers – **DESC**endent
- şamd.

Aceasta se face folosind clauza:

ORDER BY

În clauza **ORDER BY** pot fi specificate mai multe câmpuri, nu neapărat unul singur – deşi această din urmă abordare este mult mai frecventă decât altele.

În acest caz, ordonarea se va face după toate câmpurile specificate – de obicei, descendent după primul câmp şi ascendent după următoarele.

Aşadar, ordonarea poate fi făcută:

- **ASC**endent – de la cea mai mică la cea mai mare valoare din câmpul/câmpurile de interes
  - ***acesta este modul implicit de ordonare şi poate lipsi*** *din clauza* ***ORDER BY***
- **DESC**endent – de la cea mai mare la cea mai mică valoare din câmpul/câmpurile de interes

Modul în care solicităm a se face ordonarea se specifică astfel:

- după clauza **ORDER BY**
- cu particula **ASC** sau cu particula **DESC** urmând lui **ORDER BY**

Clauza SQL care determină ordonarea înregistrărilor rezultate din **SELECT** se plasează astfel:

- după clauzele **WHERE**
- înainte sau după clauza **GROUP BY**
- în orice caz, după instrucţiunea **SELECT**... **FROM** *NumeleTabelului*

De exemplu:
Să trimitem o astfel de instrucţiune **SELECT** bazei noastre de date, *bdtest*:

1. Vrem să ne fie prezentate ordonat persoanele din tabelul "*t_persoane*",
   înregistrările fiind ordonate **DESC**endent după vârsta fiecărei persoane –
   deci, de la cea mai bătrână persoană până la cea mai tânără

```
USE bdtest;
SELECT
*
FROM
        t_persoane
ORDER BY
        persoana_varsta
DESC
```



	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
1	Henriksen	Lance	70	Denver	1950-11-12 00:00:00.000
2	Gibson	Mel	60	Atlanta	1960-01-02 00:00:00.000
3	Pratt	Chris	50	New York	1970-02-03 00:00:00.000
4	Portman	Natalie	45	Miami	1975-11-12 00:00:00.000
5	Hilton	Paris	35	Los Angeles	1985-09-10 00:00:00.000
6	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000

2. Vrem să ne fie prezentate ordonat numai ***numele şi vârsta*** persoanelor din tabelul "*t_persoane*", înregistrările fiind ordonate **ASC**endent după ***prenumele*** fiecărei persoane

```
USE bdtest;
SELECT
        persoana_nume, persoana_varsta
FROM
        t_persoane
ORDER BY
        persoana_prenume
-- nu am mai specificat ASC fiindca ordonarea se face implicit ASCendent
```

	persoana_nume	persoana_varsta
1	Pratt	50
2	Watson	30
3	Henriksen	70
4	Gibson	60
5	Portman	45
6	Hilton	35

Aici, prenumele persoanei = valoarea din câmpul "*persoana_prenume*" fiind, respectiv, de sus în jos:

- Chris
- Emily
- Lance
- Mel
- Natalie
- Paris

Deşi câmpul după care se face ordonarea ORDER BY poate lipsi din înregistrările rezultate/prezentate/accesibile, ordonarea se poate face după valorile din acel câmp

Alăturarea înregistrărilor din mai multe tabele: JOIN

Adesea, într-o bază de date se află mai multe tabele – în fiecare fiind înregistrări despre entităţi distincte.

De exemplu, o bază de date cu informaţii despre cetăţenii care locuiesc într-un sector al unei capitale (deci, mai multe entităţi distincte - de exemplu, Bucureşti cu 6 sectoare şi Viena – unde capitala e împărţită în 23 districte) poate avea aceste tabele cu înregistrări despre acei cetăţeni:

- un tabel cu câmpuri pentru:
  - codul numeric personal – câmp de tip text
    - care conţine valori dintre care unele sunt identice cu unele dintr-un câmp din celălalt tabel***, indiferent de numele câmpurilor unde se află***
  - adresa – câmp de tip text
  - seria cărţii de identitate – câmp de tip text
  - numărul cărţii de identitate – câmp de tip text
- un tabel cu câmpuri pentru:
  - codul numeric personal – câmp de tip text
    - care conţine valori dintre care unele sunt identice cu unele dintr-un câmp din celălalt tabel, ***indiferent de numele câmpurilor unde se află***
  - numărul de înmatriculare al autovehicului – câmp de tip text
  - datele calendaristice ale amenzilor primite în trafic – câmp de tip text
  - tipul fiecărei astfel de amenzi – câmp de tip text
  - starea fiecărei astfel de amenzi: achitată sau neachitată - câmp *bool*-ean, adevărat/fals

Câmpul codului numeric personal existând ***eventual*** *cu aceeaşi denumire* în cele două tabele,
înregistrările din cele două tabele pot fi combinate/ unite/alipite
în acord cu valorile din acel câmp.

Clauza SQL care efectuează astfel de alipiri ale unor înregistrări din tabele distincte este:

JOIN

Sintaxa generică a **JOIN** într-o instrucţiune **SELECT** care solicită înregistrări din două tabele având un câmp ***eventual*** *comun ca denumire* este:

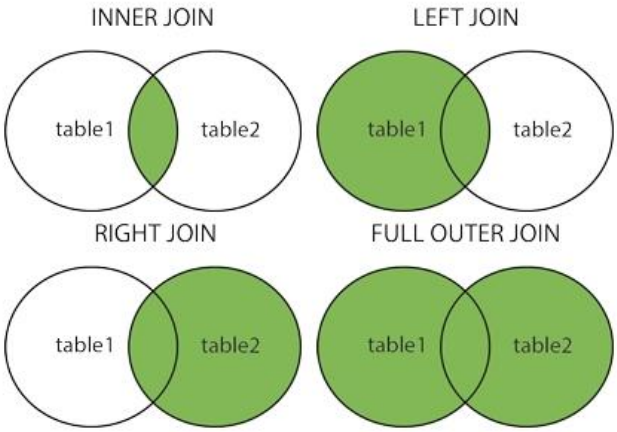
SELECT
NumeleTabelului1.CâmpulCutareDinTabelul1,
NumeleTabelului2.CâmpulCutareDinTabelul2,
FROM *NumeleTabelului1*
JOIN *NumeleTabelului* 2
ON
NumeleTabelului1.CâmpulCutareXDinTabelul1 = NumeleTabelului2.CâmpulCutareYDinTabelul2

Punctul (.) separă
numele câmpurilor unde s-ar afla valorile identice
de numele tabelelor corespunzătoare:
NumeleTabelului.CâmpulCutareDinTabel

Mai departe, acestei clauze i se poate asocia modul în care se va efectua alipirea înregistrărilor, utilizând una din aceste particule care să preceadă "**JOIN**", individual sau unele din ele alăturate:

- LEFT (OUTER) JOIN: întoarce toate înregistrările din primul tabel (cel "din stânga") şi pe cele care se potrivesc din cel de-al doilea tabel (cel "din dreapta")
- RIGHT (OUTER) JOIN: întoarce toate înregistrările din cel de-al doile tabel (cel "din dreapta") şi pe cele care se potrivesc din primul tabel (cel "din stânga")
- (INNER) JOIN: întoarce din ambele tabele toate înregistrările care corespund criteriului
- FULL (OUTER) JOIN: întoarce toate înregistrările care corespund indiferent în ce sens (stânga -> dreapta sau dreapta -> stânga) se caută în cele două tabele

O imagine care prezintă aceste moduri este următoarea:



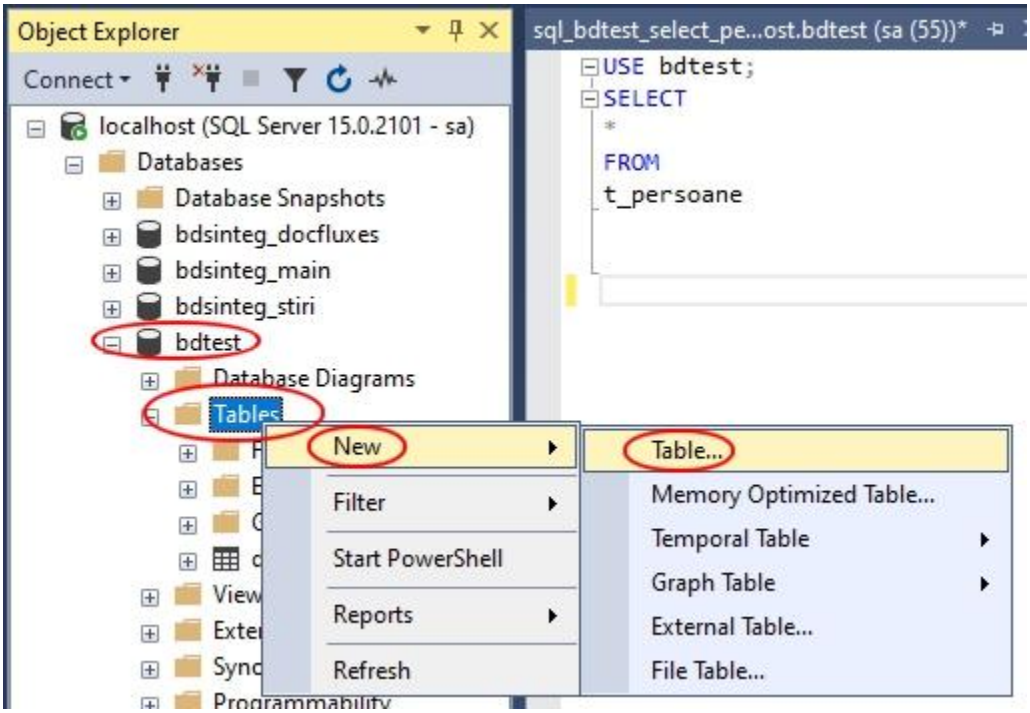
sursa: https://www.w3schools.com/sql/sql_join.asp

Să aflăm ce putem face cu **JOIN**:

Pornim SSMS.

În baza noastră de date "*bdtest*" mai creăm un tabel:
Vom încerca să alipim înregistrări din noul tabel unora din tabelul deja existent, "t_persoane"

În *TreeView*-ul unde apare numele bazei de date ("bdtest"):
desfăşurăm nodul numit "bdtest"
apoi click-dreapta pe nodul "Tables"
şi selectăm "New -> Table...":



Ştim cum să proiectăm un tabel – cum să adăugăm câmpuri, ce tip să stabilim pentru fiecare câmp.

Aşa că alcătuim structura de bază a noului tabel:
Adăugăm câmpurile din imaginea urmatoare:

Column Name	Data Type	Allow Nulls
oras_nume	nvarchar(50)	☐
oras_obiectiv	nvarchar(100)	☑
		☐

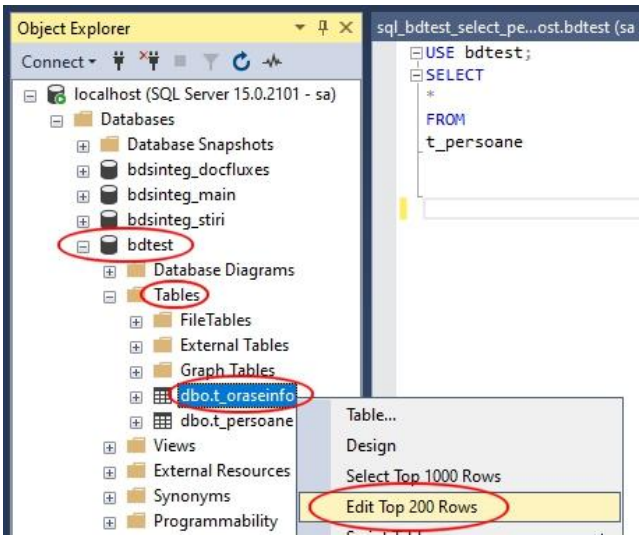
Salvăm acest tabel sub numele "t_oraseinfo".

În imaginea precedentă se observă că:
- tabelul conţine un câmp numit "oras_nume" de tip text cu lungimea maximă de 50 caractere/litere
- în câmpul "oras_nume" nu pot exista valori nule
- în câmpul "oras_obiectiv" pot exista valori nule, adică în acel oraş e posibil să nu existe obiective de interes (turistic)

Să populăm acest tabel cu câteva înregistrări
dintre care unele vor avea ca valoare a câmpului "oras_nume"
o valoare din câmpul "persoana_oras" din tabelul "t_persoane"

Pentru a vedea noul tabel în *TreeView*-ul cu entităţile cunoscute acestui server SQL:
Click-stânga pe nodul "Tables" al bazei de date "bdtest"
Apoi tasta **F5 (Refresh)**

Adăugăm înregistrări la noul tabel la fel cum am făcut pentru celălalt:
Click-dreapta pe numele tabelului celui nou, "t_oraseinfo"
Apoi se selectează opţiunea "Edit Top 200 Rows":



Pentru exemplificarea funcţionalităţii **JOIN** pe cele două tabele din baza noastră de date, "bdtest" la tabelul "t_oraseinfo" adaugăm înregistrările din imaginea următoare:

	oras_nume	oras_obiectiv
	New York	Statue of Liberty
	San Francisco	Golden Gate Bridge
	Miami	Miami Beach
	Denver	Downtown Aquarium
	Portland (OR)	Municipal Fountains

Acum scriem o instrucţiune (o interogare) **SELECT** care să combine/alipească:
înregistrările din tabelul cu informaţii despre persoane ("tabelul 1") cu
înregistrările din tabelul cu informaţii despre oraşe ("tabelul 2")
care în valoarea câmpului cu numele oraşului din tabelul 2
au valoarea câmpului cu numele oraşului din tabelul 1:

```
USE bdtest;
SELECT
*
FROM
	t_persoane
JOIN
	t_oraseinfo
ON
	t_persoane.persoana_oras = t_oraseinfo.oras_nume
```

Executăm această instrucţiune: click-stânga în interiorul textului său, apoi tasta **F5** sau click pe comanda "Execute" de pe bara de unelte a SSMS.

Rezultatul:

Results | Messages

	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data	oras_nume	oras_obiectiv
1	Pratt	Chris	50	New York	1970-02-03 00:00:00.000	New York	Statue of Liberty
2	Portman	Natalie	45	Miami	1975-11-12 00:00:00.000	Miami	Miami Beach
3	Henriksen	Lance	70	Denver	1950-11-12 00:00:00.000	Denver	Downtown Aquarium
4	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000	Portland (OR)	Municipal Fountains

Cu această instrucţiune am obţinut
înregistrări compuse din înregistrările din cele două tabele
care ca element comun au numele oraşului
ca valoare a câmpului care conţine numele oraşului
indiferent cum se numeşte acel câmp

Adăugarea de înregistrări: INSERT

Până acum am adăugat manual, din SSMS, înregistrări la un tabel, deci la o bază de date.

La fel cum am scris instrucţiuni **SELECT** cu care solicitam înregistrări dintr-un tabel,
aşa putem scrie instrucţiuni cu care să trimitem înregistrări noi într-un tabel.

Comanda SQL dedicată acestei proceduri este:

INSERT

Sintaxa generică a lui **INSERT**, prin care într-un tabel este trimisă o înregistrare nouă –
specificând câmpurile de completat ale înregistrării
şi valorile corespunzătoare – este:

INSERT
NumeleTabelului
(
NumeleCampuluiDeCompletat1,
NumeleCampuluiDeCompletat2,
...,
NumeleCampuluiDeCompletatN
)
VALUES
(
ValoareaPentruCampulDeCompletat1,
ValoareaPentruCampulDeCompletat2,
...,
ValoareaPentruCampulDeCompletatN
)

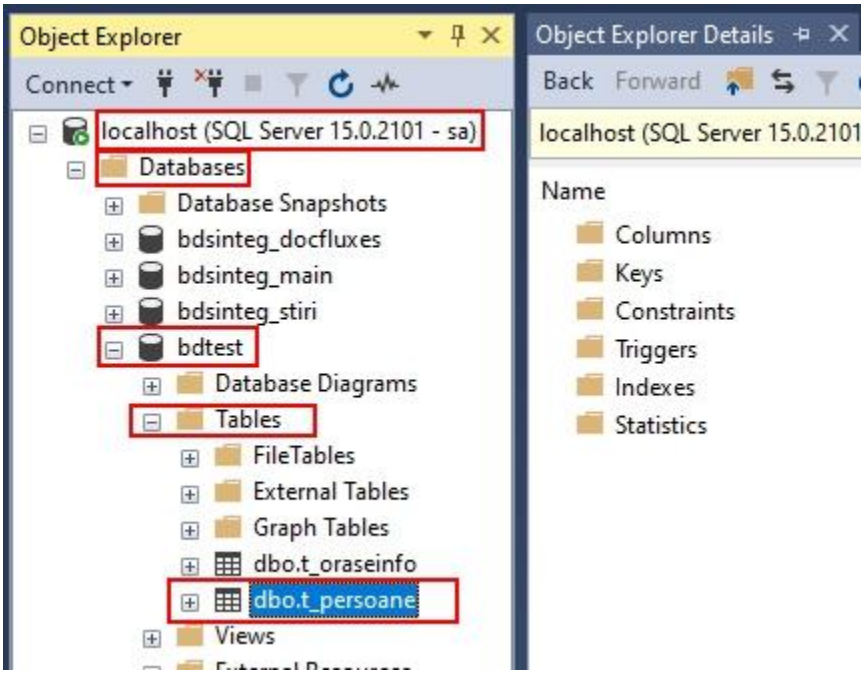
Valorile trebuie să fie specifcate în aceeaşi ordine cu cea a câmpurilor în care sunt trimise

Să adăugăm o înregistrare la tabelul "*t_persoane*" din baza noastră de date, "*bdtest*":

Pornim SSMS,
iar înainte de a scrie instrucţiuni SQL
ne asigurăm că serverul SQL la care ne-am conectat cu nume şi parolă
conţine entităţile la care ne vom referi:
(fiindcă de la ultima accesare prin SSMS se poate ca altcineva să fi şters unele din acele entităţi)

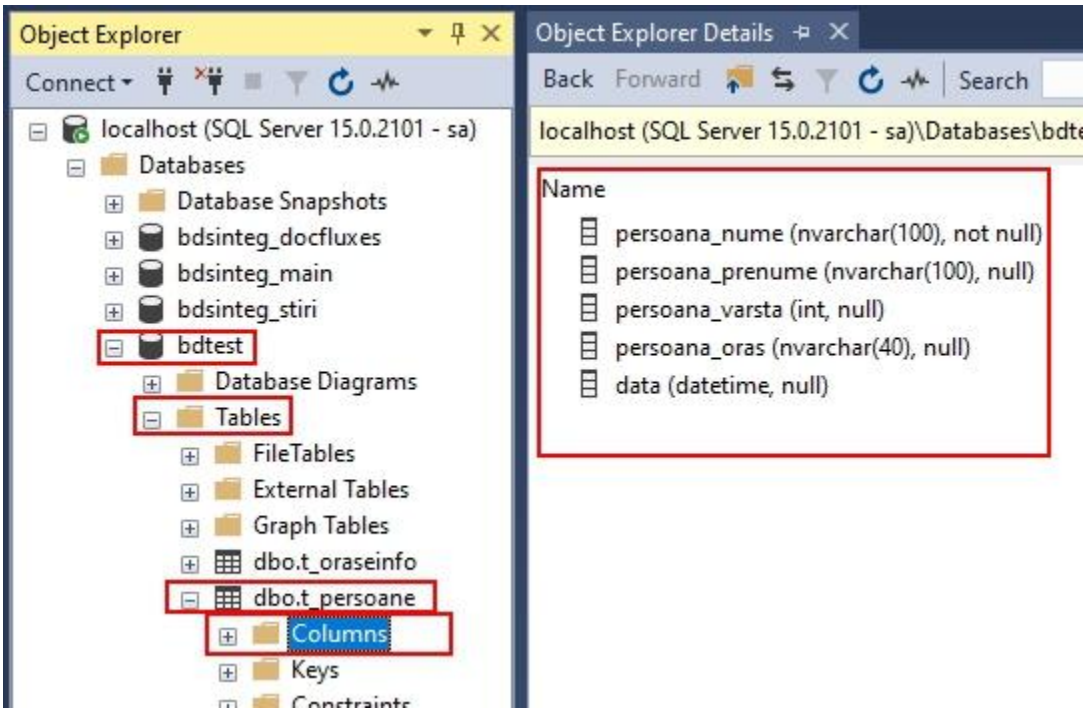
Aşadar, ne asigurăm că în serverul SQL (care în imagine este numit "*localhost*", deci *acest computer*):

- există baza de date "*bdtest*"
- în care există tabelul "*t_persoane*":



Mai departe, pentru a ne asigura că în comanda INSERT ne vom referi la câmpuri existente în tabel:

- desfăşurăm nodul cu numele acestuia
- click-stânga (selectare) pe nodul "*Columns*":

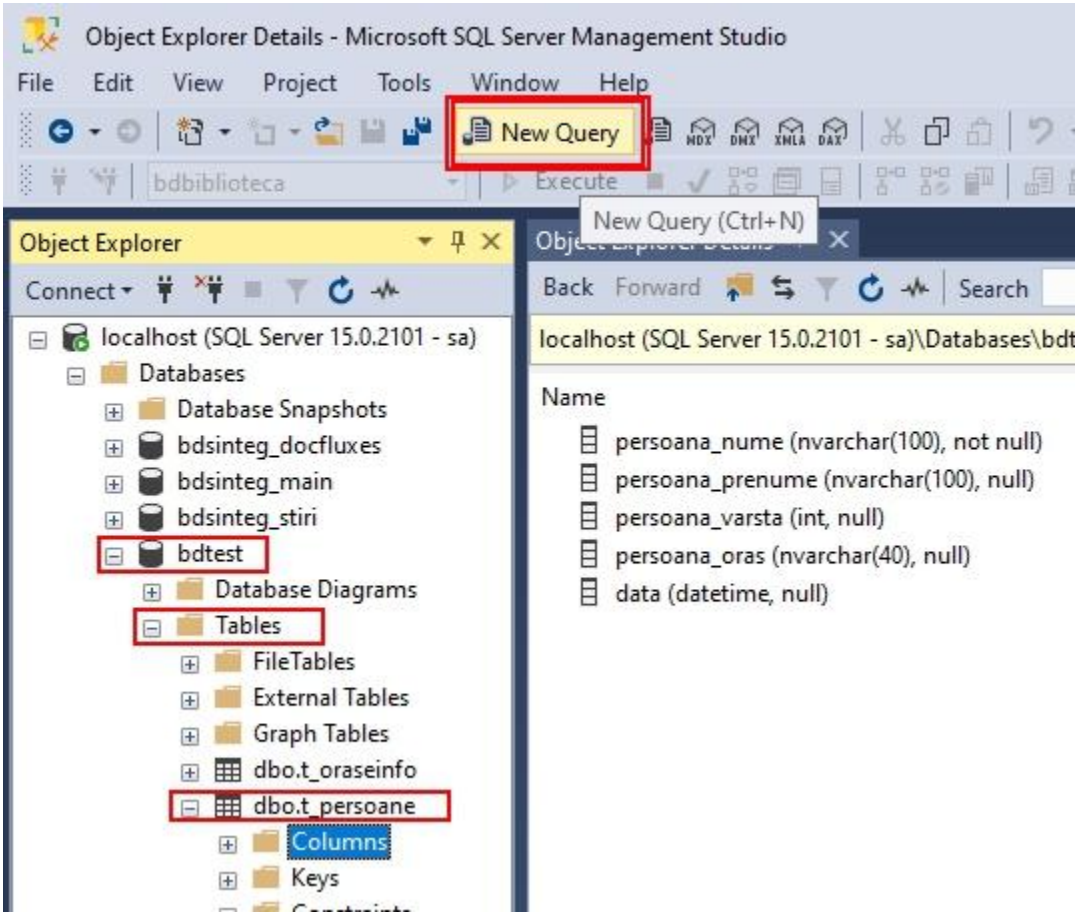


Dacă în dreapta nu apar numele câmpurilor (coloanelor)) tabelului selectat, atunci:

- se apasă tasta **F7** ("*Object Explorer Details*")
- şi tasta **F5** (Refresh)

Acum scriem instrucţiunea **INSERT** cu care vom adăuga o înregistrare la cele din tabelul "*t_persoane*":

Pe bara de unelte a SSMS, click pe "*New Query*":
(deci, la fel ca atunci când scriem o comandă **SELECT** şi de altfel orice instrucţiuni SQL)



Scriem această instrucţiune SQL:

```
USE bdtest;
INSERT
t_persoane
        (
        persoana_oras,
        data,
        persoana_varsta,
        persoana_nume,
        persoana_prenume
        )
VALUES
        (
        'Houston',
        '1940-08-05',
        '80',
        'Brooks',
        'Mel'
        )
```

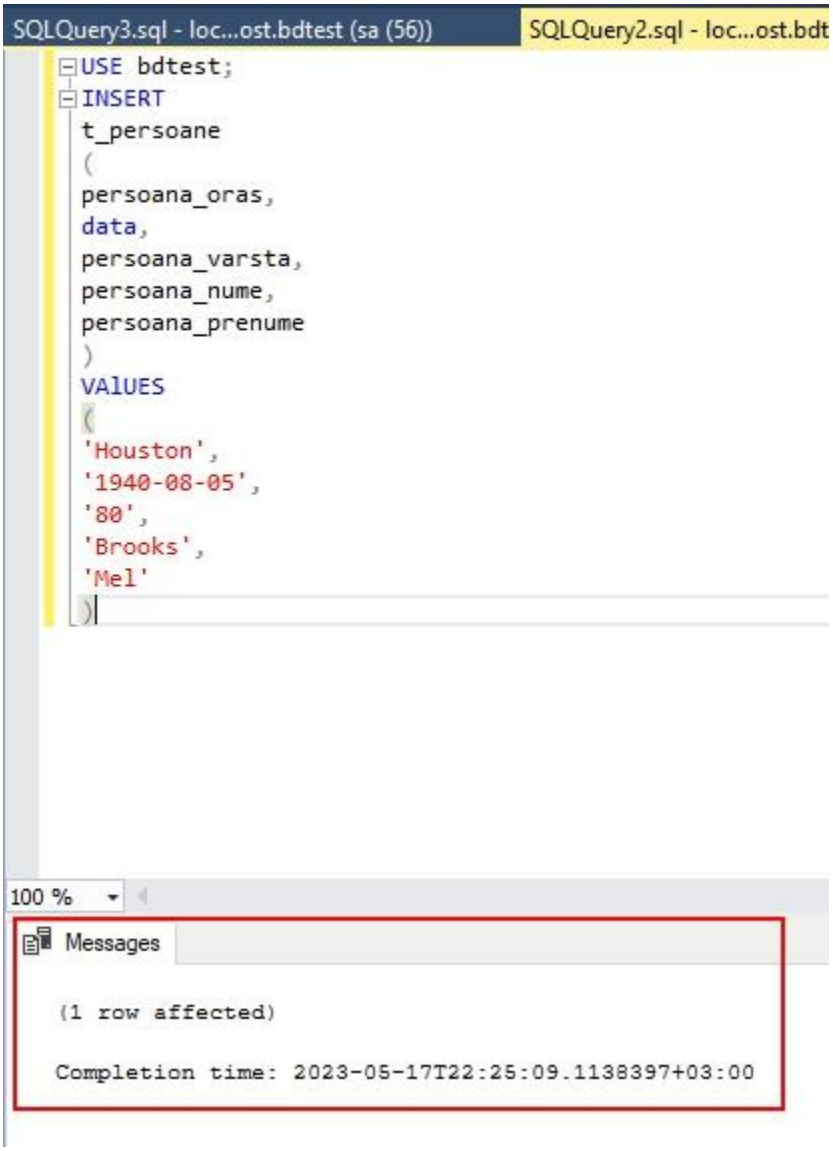
Se observă cum am specificat câmpurile de completat în altă ordine decât cea în care sunt în tabel.

Cum s-a spus mai devreme –

La scrierea unei instrucţiuni INSERT
contează respectarea simultană a
- ***ordinii câmpurilor de completat care ordine impune***
- ***ordinea valorilor trimise ca parametri ai VALUES***

Executăm această instrucţiune: apăsăm tasta **F5**, sau click pe "*Execute*" de pe bara de unelte a SSMS.

La fel ca în cazul unei instrucţiuni **SELECT** şi de altfel ca în cazul oricăror instrucţiuni SQL, rezultatul va fi afişat jos – sub secţiunea SSMS unde am scris instrucţiunea:



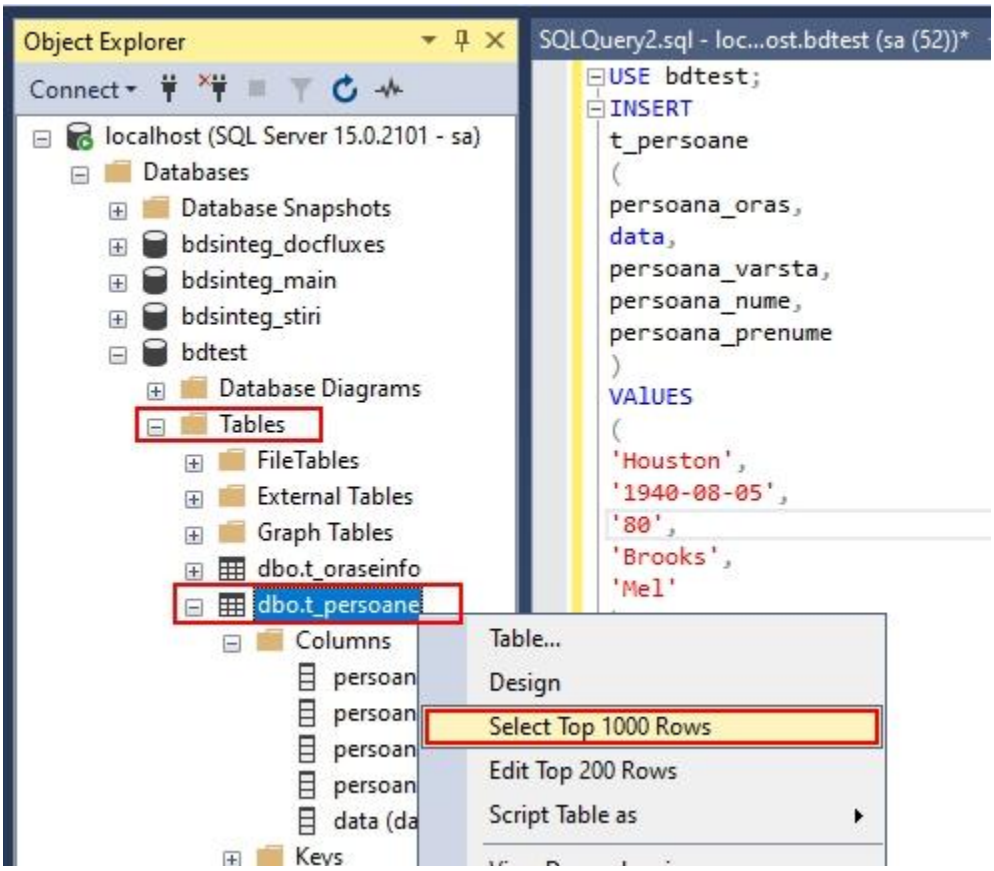
Mesajul "*1 row affected*" sau similar indică faptul că a fost "afectată" 1 înregistrare (1 linie din tabel) – în acest caz, că a fost adăugată 1 înregistrare.

Să verificăm dacă înregistrarea trimisă există în tabel:

SSMS oferă posibilitatea de a afişa "primele" o mie de înregistrări dintr-un tabel
fără ca utilizatorul să trebuiască a scrie o comandă **SELECT**

Această funcţionalitate este apelabilă astfel:

În *TreeView*-ul din stânga, unde se află entităţile cunoscute serverului SQL la care ne-am conectat:
Se desfăşoară până la nodul cu numele tabelului din care vrem să **SELECT**-ăm înregistrări, în cazul de faţă pentru a verifica dacă înregistrarea a fost adăugată cu **INSERT**
Apoi click-dreapta pe acel nod
Apoi se alege intrarea de meniu "*Select Top 1000 Rows*" sau similar:



Tabelul nostru "*t_persoane*" nu conţine o mie de înregistrări, dar rezultatul va fi afişat jos în SSMS:



Results | Messages

	persoana_nume	persoana_prenume	persoana_varsta	persoana_oras	data
1	Pratt	Chris	50	New York	1970-02-03 00:00:00.000
2	Henriksen	Lance	70	Denver	1950-11-12 00:00:00.000
3	Gibson	Mel	60	Atlanta	1960-01-02 00:00:00.000
4	Watson	Emily	30	Portland (OR)	1990-06-07 00:00:00.000
5	Portman	Natalie	45	Miami	1975-11-12 00:00:00.000
6	Hilton	Paris	35	Los Angeles	1985-09-10 00:00:00.000
7	Brooks	Mel	80	Houston	1940-08-05 00:00:00.000

Se vede înregistrarea trimisă mai devreme cu instrucţiunea **INSERT** şi cum

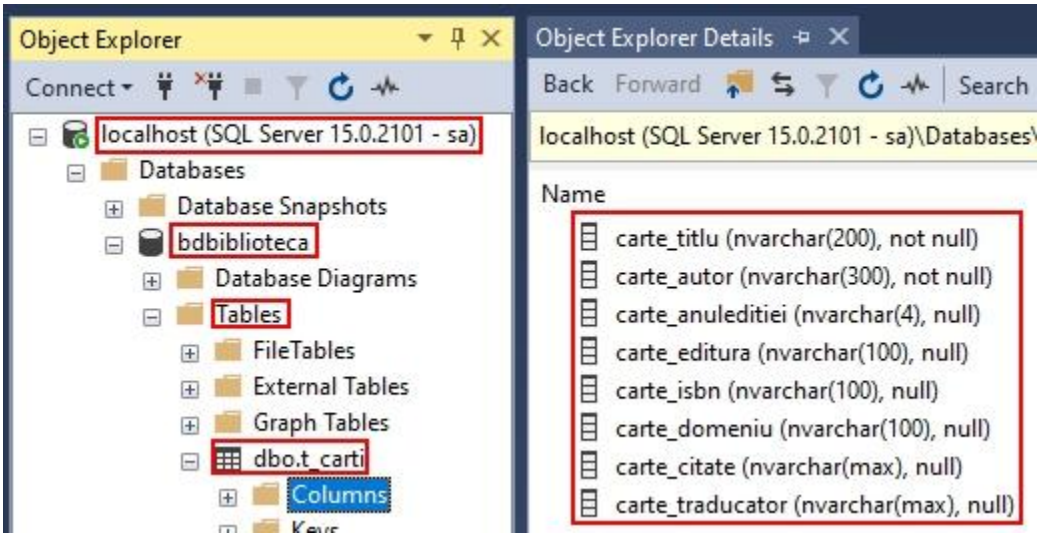
- ***se află după celelalte înregistrări, la urmă***
- fiecare valoare trimisă într-un câmp
  este în câmpul corespunzător
  fiindcă:
  - deşi am trimis numele câmpurilor în altă ordine decât cea în care sunt în tabel
  - am trimis fiecare valoare în câmpul corespunzător
    *respectând, deci, ordinea din instrucţiunea **INSERT**, nu pe cea din tabel*

Se subînţelege că valori adecvate unui câmp de un anume tip pot fi trimise în alt câmp de acelaşi tip. Aceasta, însă, cauzează confuzie şi denaturează scopul comenzii **INSERT** care, totuşi, presupune popularea corectă a unui tabel şi astfel a bazei de date care conţine acel tabel.

Să proiectăm o bază de date dedicată inventarului cărţilor din biblioteca personală.

În SSMS:

- creăm o bază de date numită "*bdbiblioteca*"
- la aceasta adăugăm un tabel numit "*t_carti*"
- ale cărui coloane/câmpuri se numesc şi au tipurile prezentate în imaginea următoare:
    - "carte_titlu"
        - text-*nvarchar* cu lungimea maximă 200
        - câmp obligatoriu de completat ("Allow Nulls" = False)
    - "carte_autor"
        - text-*nvarchar* cu lungimea maximă 300
        - câmp obligatoriu de completat ("Allow Nulls" = False)
    - "carte_anuleditiei"
        - text-*nvarchar* cu lungimea maximă 4
        - opţional = permite valori nule: "Allow Nulls" = True)
    - "carte_editura"
        - text-*nvarchar* cu lungimea maximă 100
        - opţional = permite valori nule: "Allow Nulls" = True)
    - "carte_isbn"
        - text-*nvarchar* cu lungimea maximă 100
        - opţional = permite valori nule: "Allow Nulls" = True)
    - "carte_domeniu"
        - text-*nvarchar* cu lungimea maximă 100
        - opţional = permite valori nule: "Allow Nulls" = True)
    - "carte_citate"
        - text-*nvarchar* cu lungimea maximă a tipului *nvarchar*
        - opţional = permite valori nule: "Allow Nulls" = True)
    - "carte_traducator"
        - text-nvarchar cu lungimea maximă a tipului *nvarchar*
        - opţional = permite valori nule: "Allow Nulls" = True)



Deci, în această bază de date vom plasa
cel puţin o parte din informaţiile pentru care am pregătit tabelul "*t_carti*" –
despre cărţile din biblioteca personală.

Întrucât numărul cărţilor poate fi mai mare de câteva sute,
nu vom adăuga manual, cu comanda **INSERT** tot atâtea înregistrări câte cărţi avem.

Pentru aceasta vom folosi un program creat în limbajul C#, în mediul de programare Visual Studio®™.

Prezentarea unui astfel de program se află în documentul care ilustrează alcătuirea unor programe pentru diverse scopuri.

Însă, pentru a folosi acel program este neapărat necesar să fi pregătit această bază de date, "*bdbiblioteca*".

Acum este recomandat să se citească prezentarea acelui program, să fie compilat şi lansat în execuţie pentru a plasa informaţii despre câteva cărţi în această bază de date.

Modificarea unor înregistrări: UPDATE

Înregistrările – adică, în esenţă, valoarea unui câmp sau valorile mai multor câmpuri dintr-una sau din mai multe înregistrări - pot fi modificate cu comanda:

UPDATE

La fel cu comenzile **SELECT** şi **INSERT**, aceasta implică specificarea:

- denumirea bazei de date unde se află
  - tabelul care conţine
    - înregistrările de modificat
    - *opţional*: având
      - câmpurile ale căror
        - valori vor fi modificate

Sintaxa generică este:

UPDATE [numele tabelului]
SET [numele câmpului]
=
[valoarea nouă]
WHERE
[numele câmpului]
[operator de evaluare: = , IS , <= , IS NOT , >= , LIKE , etc]
[valoarea curentă]

Comanda **UPDATE** va fi executată chiar dacă niciun câmp nu are valoarea care trebuie modificată - bineînţeles, în această situaţie nu se va schimba nimic.

ATENŢIE
la clauza WHERE unde restrângem setul de înregistrări pe care le vom modifica

Să trimitem o astfel de comandă către baza de date "bdtest" unde se află informaţii despre câteva persoane.

1. Vom modifica numele oraşului "Denver"
   în toate înregistrările unde valoarea câmpului persoana_oras este "Denver".
   Aici, vom schimba textul *Denver* la *Phoenix*.

Comanda este:

USE bdtest
UPDATE t_persoane
SET persoana_oras = 'Phoenix'
WHERE persoana_oras = 'Denver'

2. Să restrângem setul înregistrărilor de modificat
   la cele unde valoarea câmpului "persoana_nume" este "Henriksen"

Comanda este:

```
USE bdtest
UPDATE t_persoane
        SET persoana_oras = 'Phoenix'
                WHERE
                (
                        persoana_oras = 'Denver'
                AND
                        persoana_nume = 'Henriksen'
                )
```

3. Să mărim setul înregistrărilor de modificat
   folosind clauza LIKE

   Vom modifica ***toate înregistrările*** unde câmpul "persoana_varsta"
   are valoarea mai mică decât, sau egală cu 50.2

   Desigur, putem adăuga criterii cu clauze **AND, astfel restrângând setul înregistrărilor**
   La fel, putem folosi clauze **OR, astfel mărind setul înregistrărilor**

Când comparăm cu un număr nu este nevoie să încadrăm numărul între apostrofuri

Comanda este:

```
USE bdtest
UPDATE t_persoane
        SET persoana_oras = 'Columbus'
                WHERE
                (
                        persoana_varsta <= 50.2
                )
```

4. Să modificăm ***toate înregistrările*** dintr-un tabel

ATENŢIE ATENŢIE ATENŢIE

Comanda este:

```
-- vor fi MODIFICATE TOATE INREGISTRARILE din tabelul t_persoane
-- valoarea campului persoana_oras din TOATE INREGISTRARILE va fi Atlantic City
--
USE bdtest;
UPDATE t_persoane
SET persoana_oras =  'Atlantic City"
```

Clauza AND restrânge setul înregistrărilor
pe care se va efectua o operaţiune
fiindcă trebuie îndeplinită ŞI condiţia aceea.

Clauza OR măreşte setul înregistrărilor
pe care se va efectua o operaţiune
fiindcă trebuie îndeplinită
FIE condiţia precedentă lui OR,
FIE condiţia specificată cu OR.

Este bine ca dacă este posibil şi sintactic corect
să încadrăm între paranteze rotunde ( )
clauze identice (AND/OR, AND/OR...LIKE, <= etc.)
Astfel, să existe un bloc pentru AND... AND... AND...
unul pentru OR... OR... OR...
unul pentru AND/OR...LIKE, <= etc.
şi ca toată acea clauză complexă WHERE (... (...)... (...) ...)
să fie încadrată la rându-i între paranteze rotunde.

Ştergerea unor înregistrări: DELETE

Înregistrări pot fi şterse cu comanda:

DELETE

La fel cu comenzile **SELECT**, **INSERT** şi **UPDATE**, aceasta implică specificarea:

- denumirea bazei de date unde se află
  - tabelul care conţine
    - înregistrările de şters
      - *opţional*: având
      - câmpurile ale căror
        - valori vor fi evaluate pentru ştergerea înregistrărilor respective

Sintaxa generică este:

DELETE
FROM [numele câmpului]
WHERE
[numele câmpului]
[operator de evaluare: = , IS , <= , IS NOT , >= , LIKE , etc]
[valoarea curentă]

Comanda **DELETE** va fi executată chiar dacă niciun câmp nu are valoarea evaluată - bineînţeles, în această situaţie nu se va şterge nicio înregistrare.

ATENŢIE
la clauza WHERE unde restrângem setul de înregistrări pe care le vom şterge

Să trimitem o astfel de comandă către baza de date "bdtest" unde se află informaţii despre câteva persoane.

1. Vom şterge înregistrările unde numele oraşului este "Denver"

Comanda este:

USE bdtest
DELETE FROM t_persoane
WHERE persoana_oras = 'Denver'

2. Să restrângem setul înregistrărilor de şters
   la cele unde
- ***valoarea câmpului "persoana_oras" este "Denver"***
  şi - AND
- ***valoarea câmpului "persoana_nume" este "Henriksen"***

Comanda este:

```
USE bdtest
DELETE FROM t_persoane
        WHERE
                (
                        persoana_oras = 'Denver'
                AND
                        persoana_nume = 'Henriksen'
                )
```

3. Să mărim setul înregistrărilor de şters
   folosind clauza LIKE

   Vom şterge ***toate înregistrările*** unde câmpul "persoana_varsta"
   are valoarea mai mare decât 37

   Desigur, putem adăuga criterii cu clauze **AND, astfel restrângând setul înregistrărilor**
   La fel, putem folosi clauze **OR, astfel mărind setul înregistrărilor**

Când comparăm cu un număr nu este nevoie de a folosi apostroful

Comanda este:

```
USE bdtest
DELETE FROM t_persoane
        WHERE
                (
                        persoana_varsta > 37
                )
```

4. Să ştergem ***toate înregistrările dintr-un tabel***

ATENŢIE ATENŢIE ATENŢIE

Comanda este:

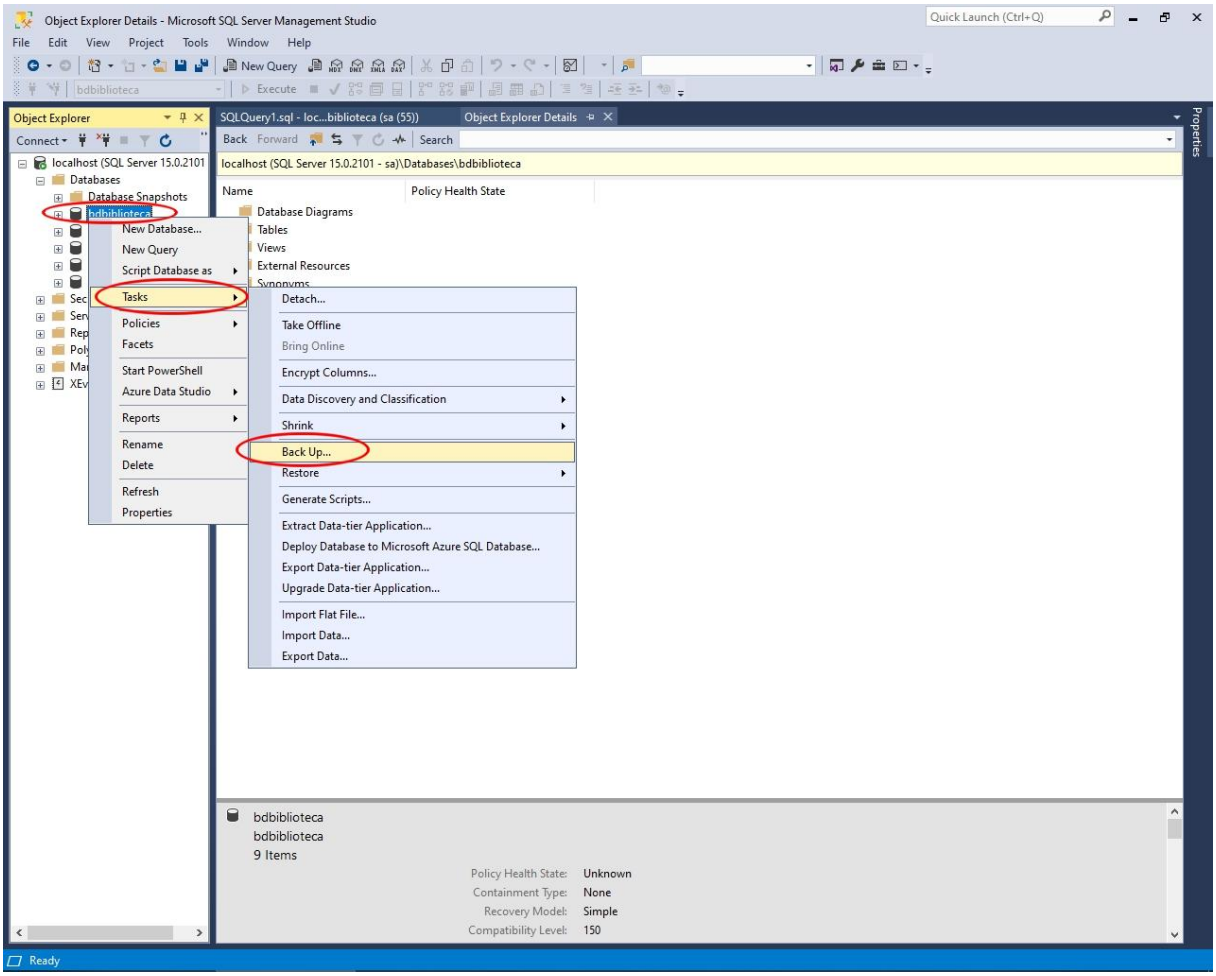
```
-- vor fi STERSE TOATE INREGISTRARILE din tabelul t_persoane
--
USE bdtest;
DELETE FROM t_persoane
```

Crearea unei copii de rezervă a unei baze de date; restaurarea/recuperarea unei baze de date

O bază de date poate fi clonată, ori duplicată într-o copie (de rezervă) astfel:

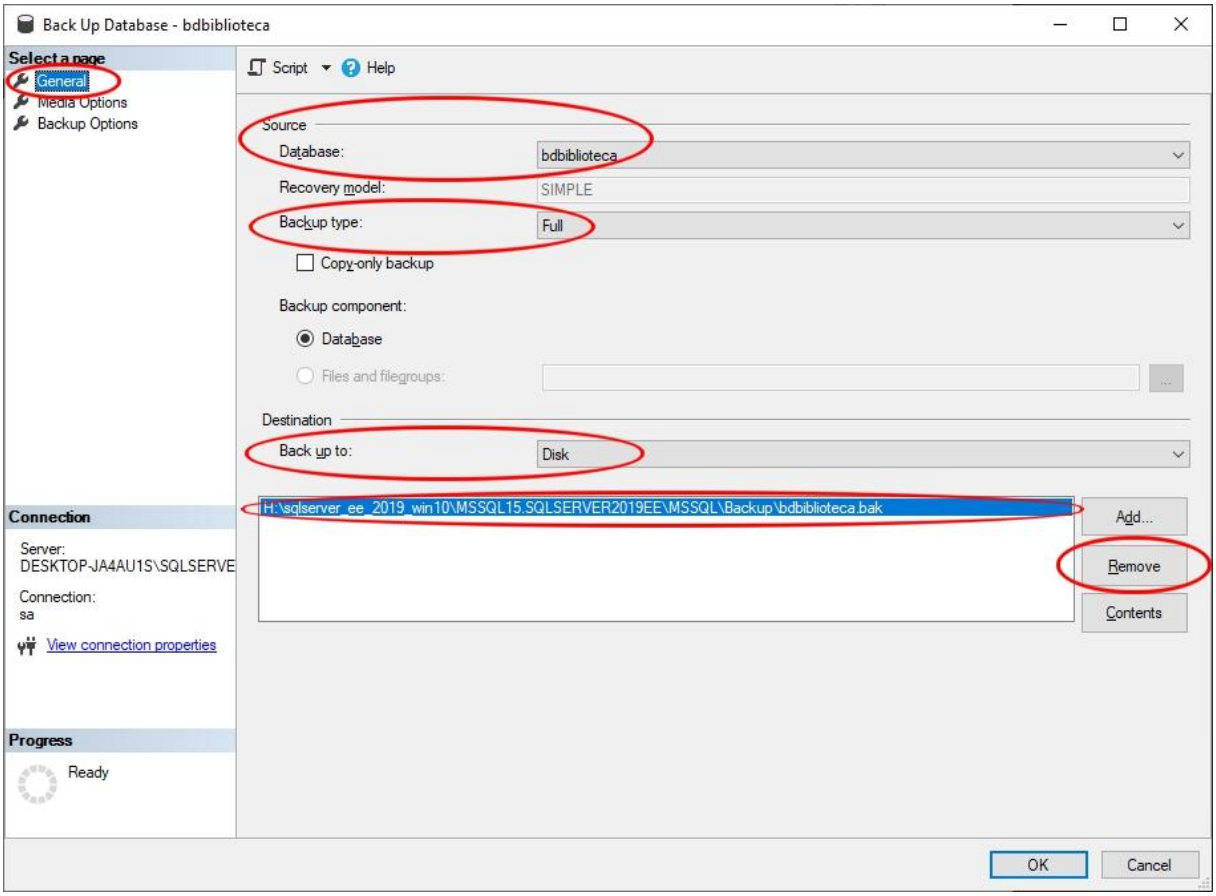
În SQL Server Management Studio:

Click pe numele bazei de date
Click pe intrarea de meniu "Tasks"
Se alege comanda "Back Up..."



În dialogul care apare specificăm caracteristicile copiei de rezervă ca în imaginile următoare:

- în pagina "***General***":



Source:

Database (aici putem duplica orice bază de date accesibilă):

- selectăm baza de date pe care o vom duplica

Backup type:

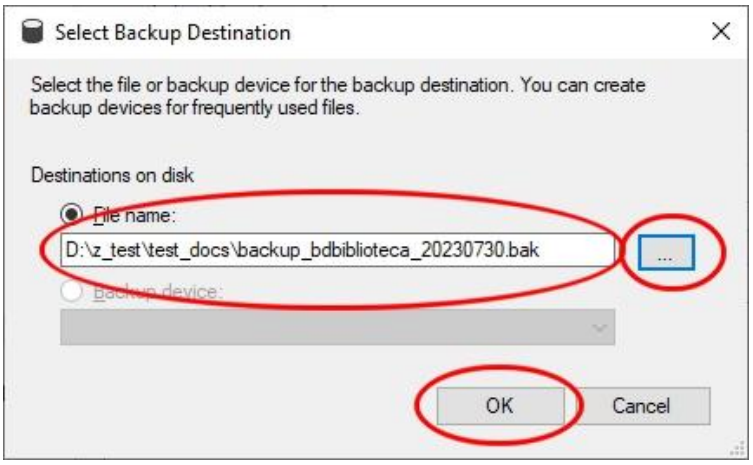
- Full (selectăm această opţiune)
- Differential

Back up to (Copierea se va face la:

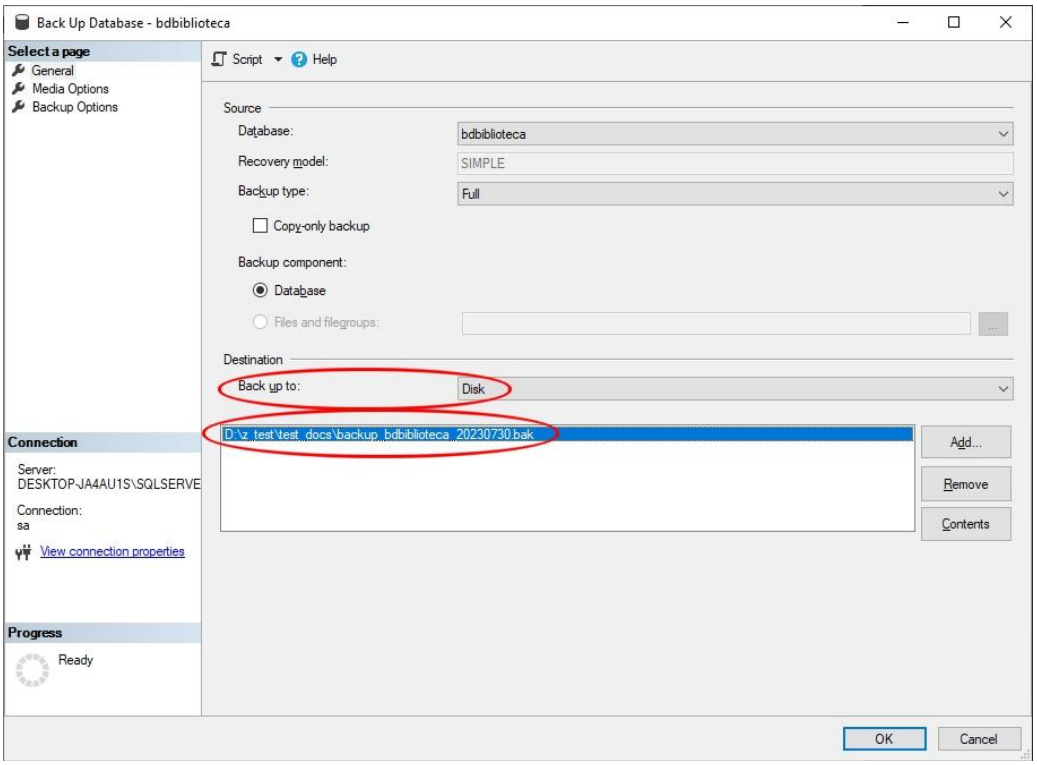
- Disk (selectăm această opţiune)
- URL (o adresă pe Internet)

Locaţia pe Disk poate fi:

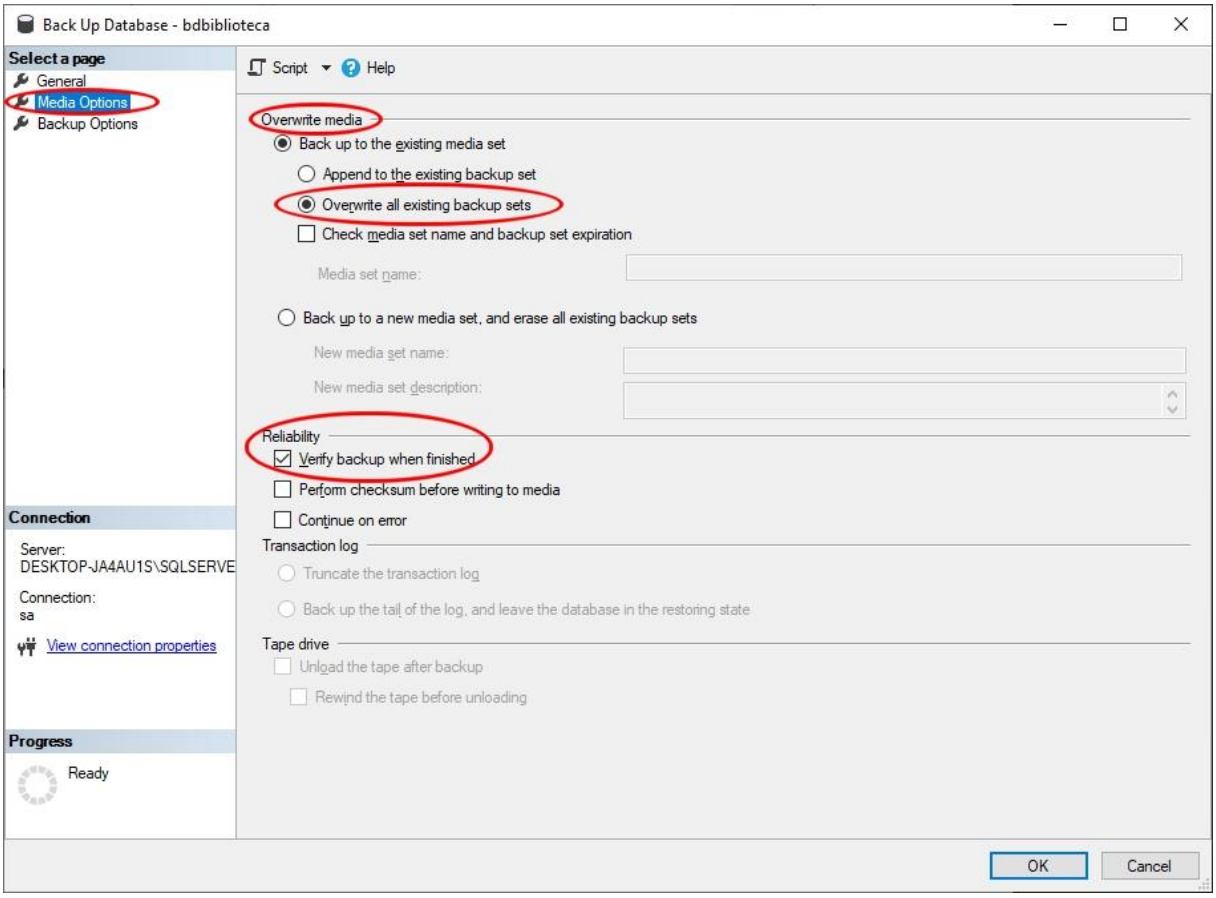
- îndepărtată, dacă nu o vom folosi:
  - o selectăm în listă
    - apăsăm butonul "Remove"
- adăugată, dacă nu este prezentă în listă:
  - apăsăm butonul "Add..."
    - selectăm locaţia şi numele fişierului unde va fi stocată copia:



Locaţia şi numele fişierului-destinaţie vor fi afişate în listă:



- acum trecem la pagina "***Media Options***":

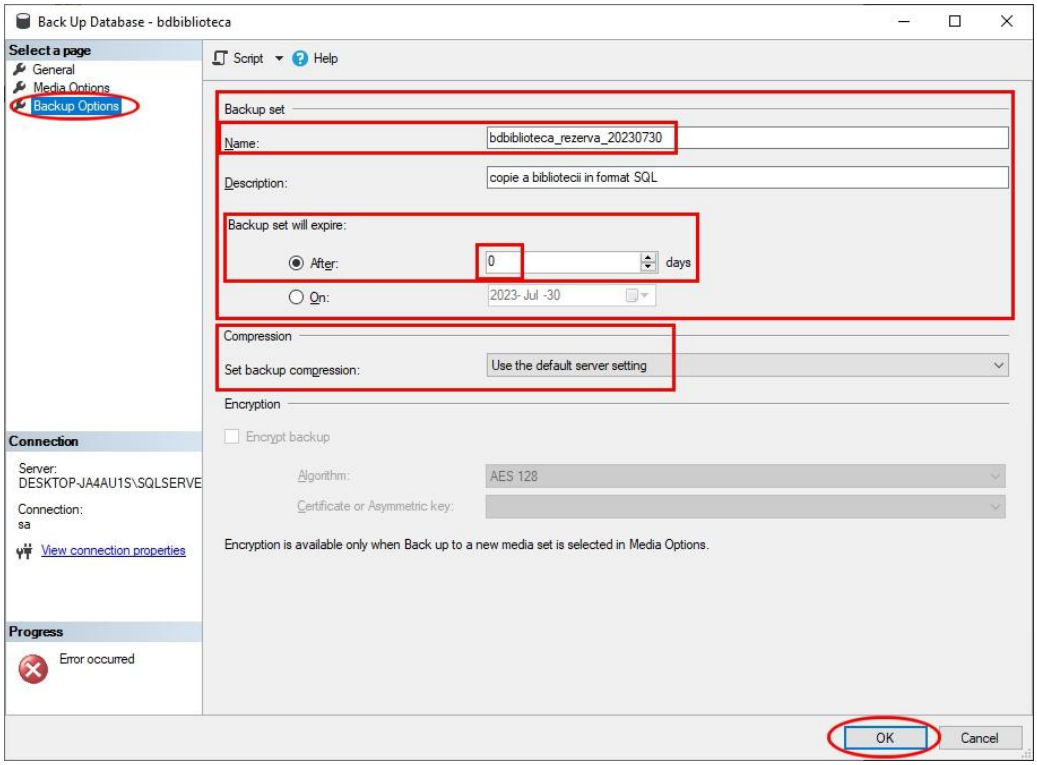


Selectăm-bifăm:

Overwrite all existing backup sets
Dacă la destinaţie exist ă deja o copie, atunci acaea va fi suprascrisă-înlocuită cu noua copie.

Verify backup when finished
Copia va fi verificată - e posibil să apară erori, de exemplu defecte ale mediului de stocare.

- trecem la pagina "***Backup Options***":



Aici specificăm:

- numele copiei, aşa cum va fi cunoscută în ansamblul SQL Server:
  - aici, "bdbiblioteca_rezerva_20230730"
- opţional, descrierea copiei
- când anume va expira copia:
  - 0 zile înseamnă niciodată
- tipul de comprimare a bazei de date
  - este posibil ca SQL Server să nu poată efectua operaţiunea de comprimare
    - astfel că este rezonabilă alegerea opţiunii "Use the default server setting" – modul implicit de comprimare suportat

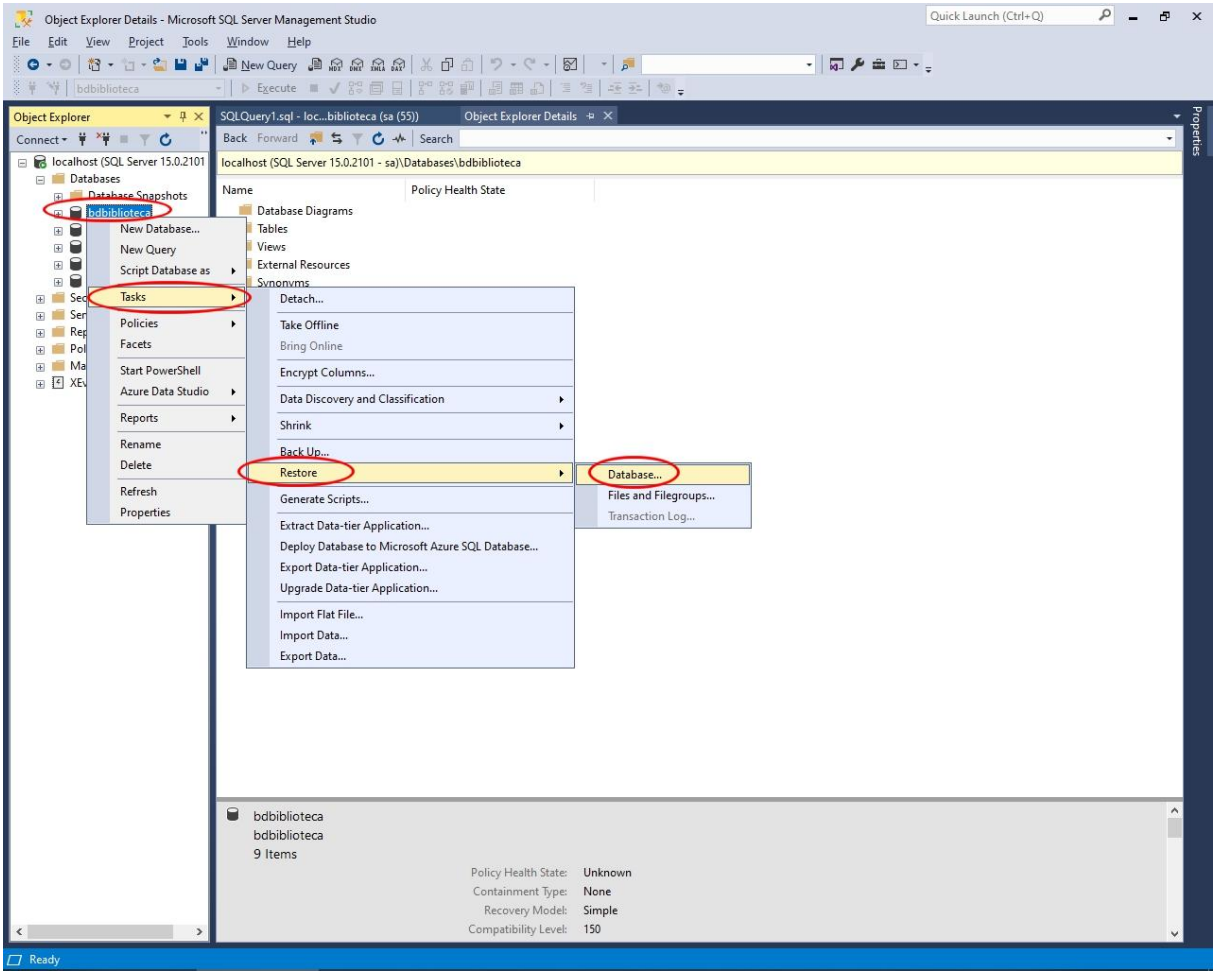
Apăsăm butonul "OK". Dacă totul merge bine, este creată copia bazei de date selectate şi rezultatul este anunţat printr-un mesaj în acest dialog:



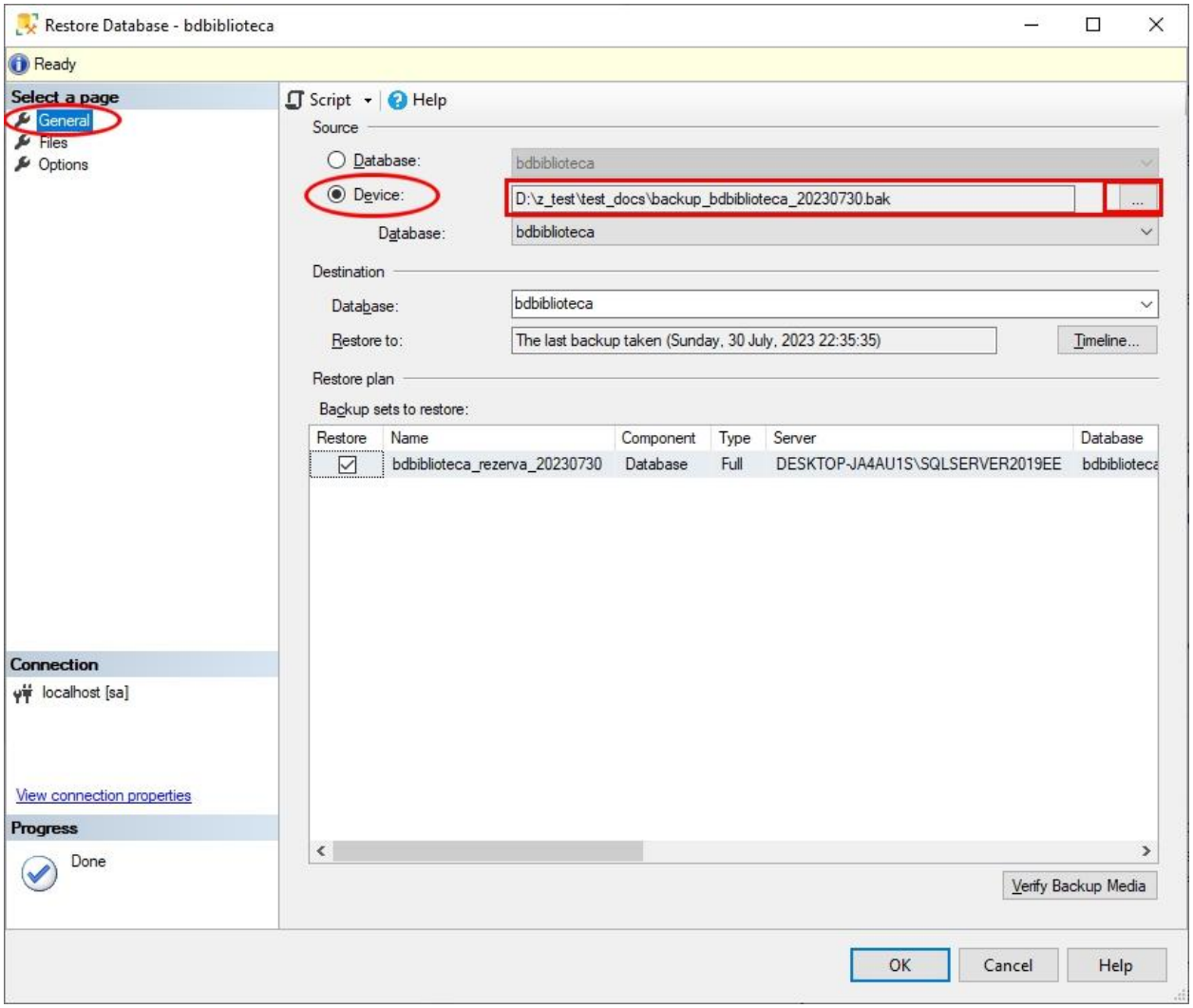
O bază de date poate fi şi restaurată, ori recuperată dintr-o copie de rezervă astfel:

În SQL Server Management Studio:

Click pe numele bazei de date
Click pe intrarea de meniu "Tasks"
Se alege comanda "Restore"
Se alege intrarea de meniu "Database...":



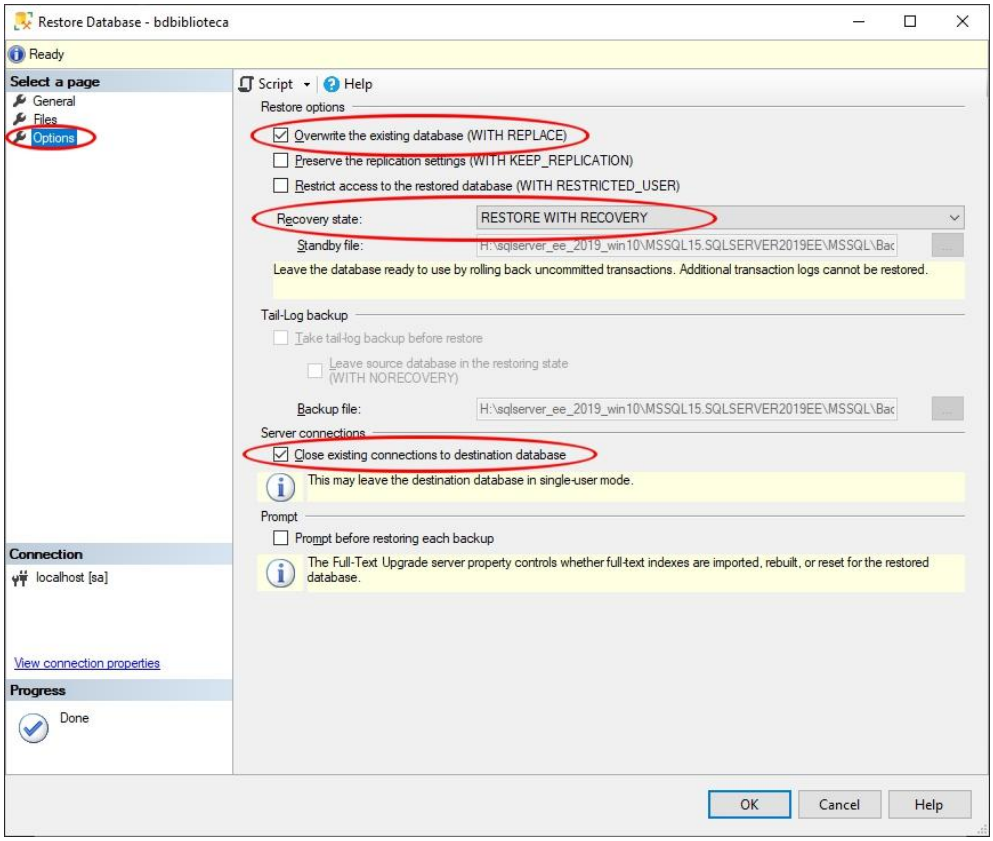
- în pagina "**General**":



Vrem să recuperăm-rest aurăm copia aflată într-o locaţie pe un mediu de stocare ("*Disk*"):

- astfel că
  - selectăm "*Device*"
    - navigăm spre fişierul-copie de rezervă pe care vrem s-o restaurăm
    - ne asigurăm că este bifată în lista marcată "*Backup sets to restore*"

- pe pagina "**Options**":



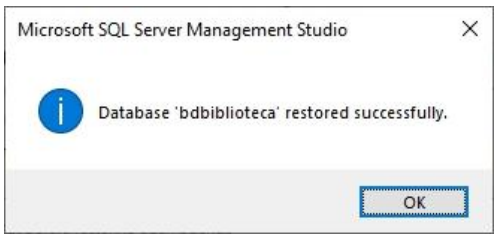
Selectăm-bifăm:

- "*Overwrite the existing database (WITH REPLACE)*"
  - aceasta înseamnă ca baza de date existentă acum va fi înlocuită/suprascrisă cu copia
- "*Close existng connections to destination database*"
  - aceasta înseamnă că înainte de înlocuire-recuperare-restaurare vor fi închise toate conexiunile către baza de date curentă

Selectăm modul de restaurare:

- *Recovery state*: RESTORE WITH RECOVERY

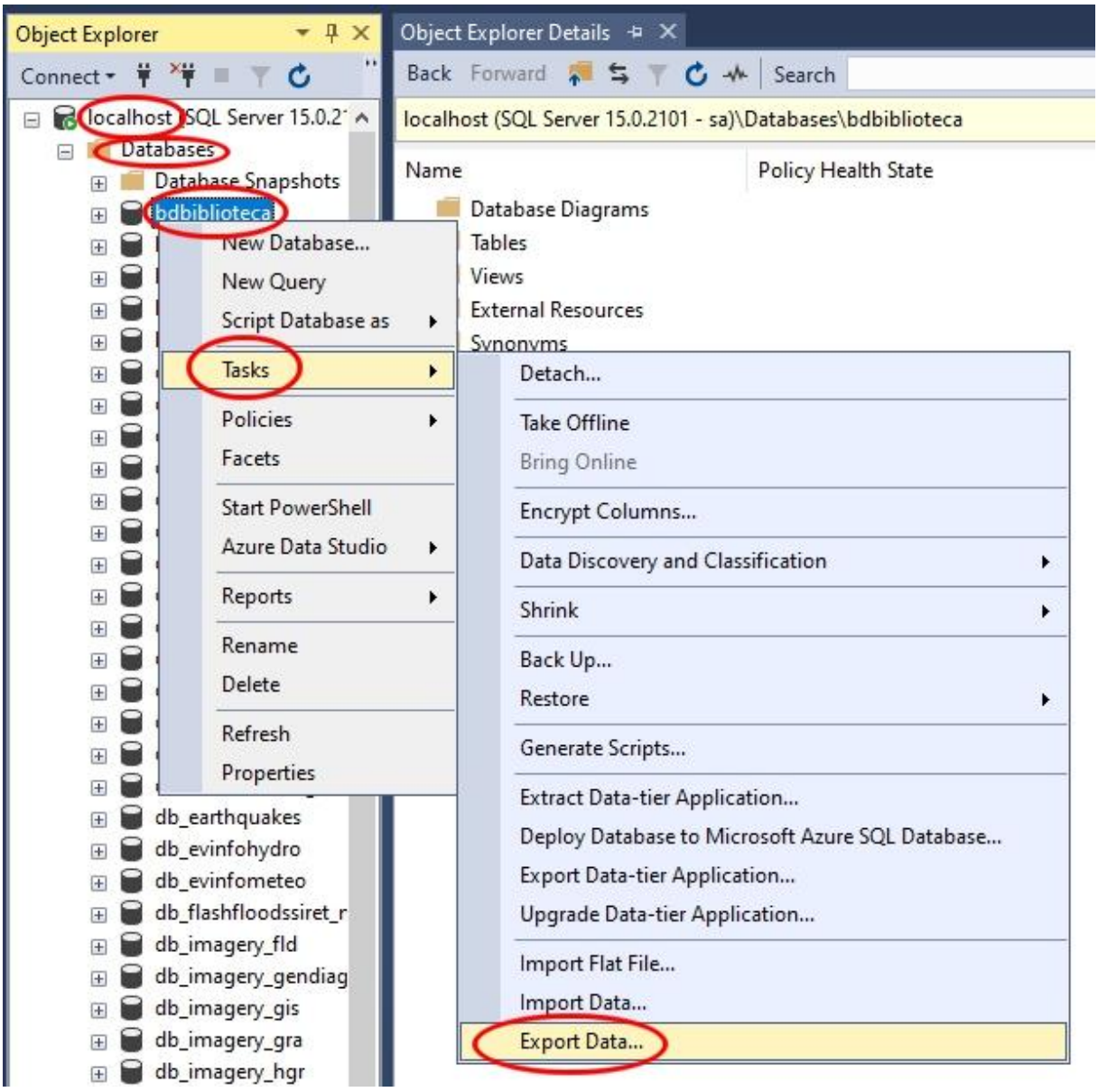
Apăsăm butonul "OK". Dacă totul merge bine, este creată copia bazei de date selectate şi rezultatul este anunţat printr-un mesaj în acest dialog:

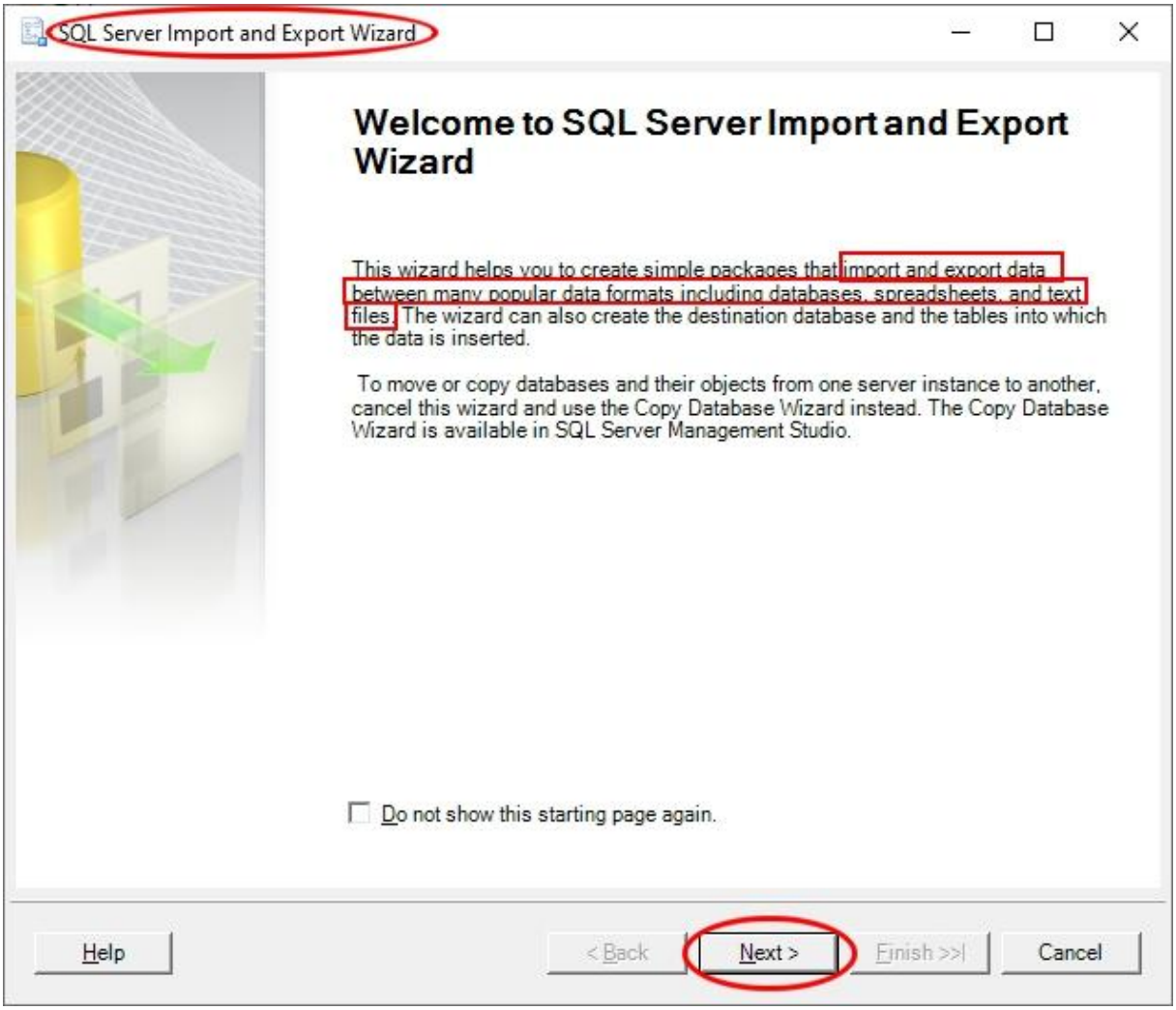


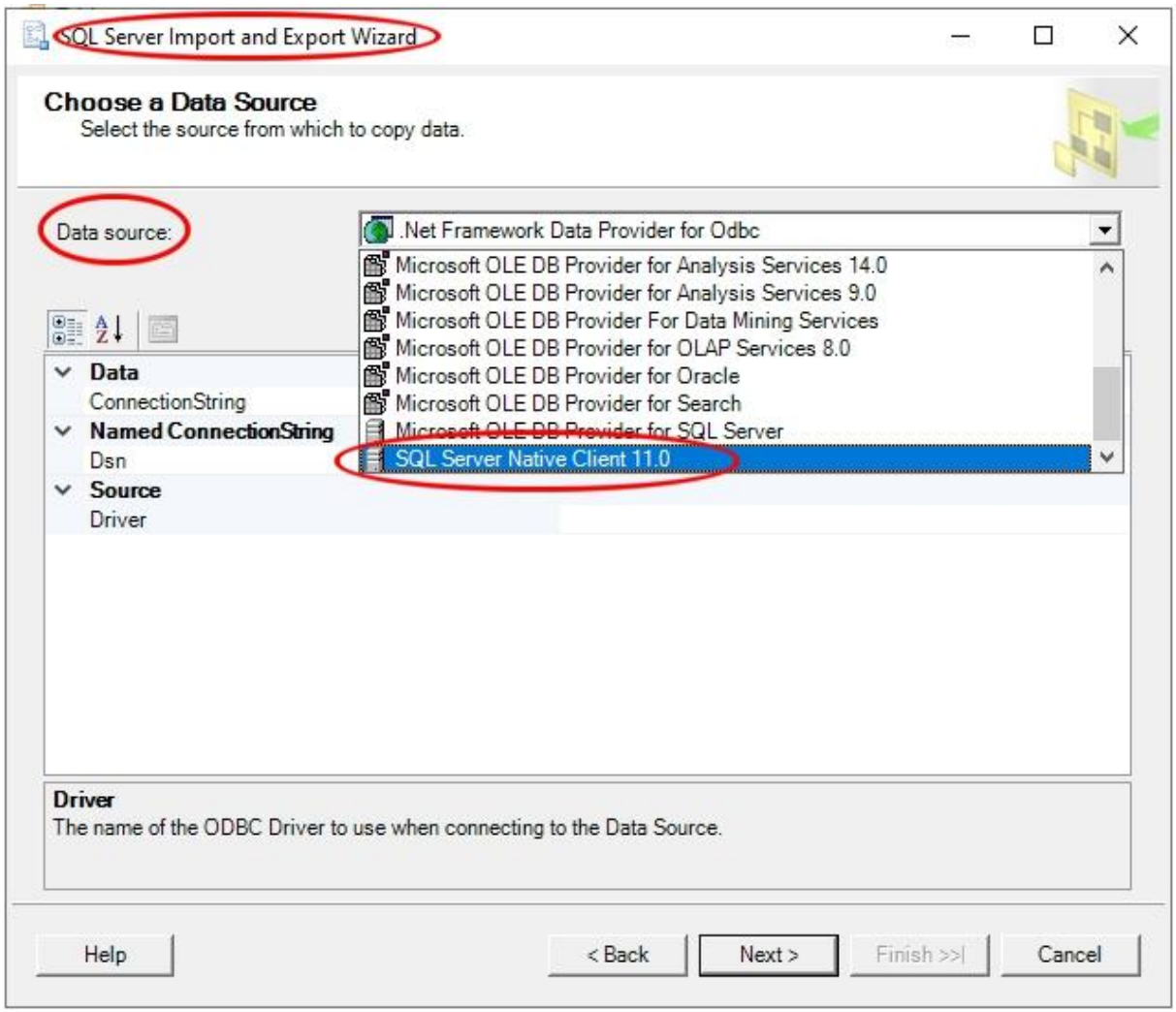
Exportarea de informaţie dintr-o bază de date

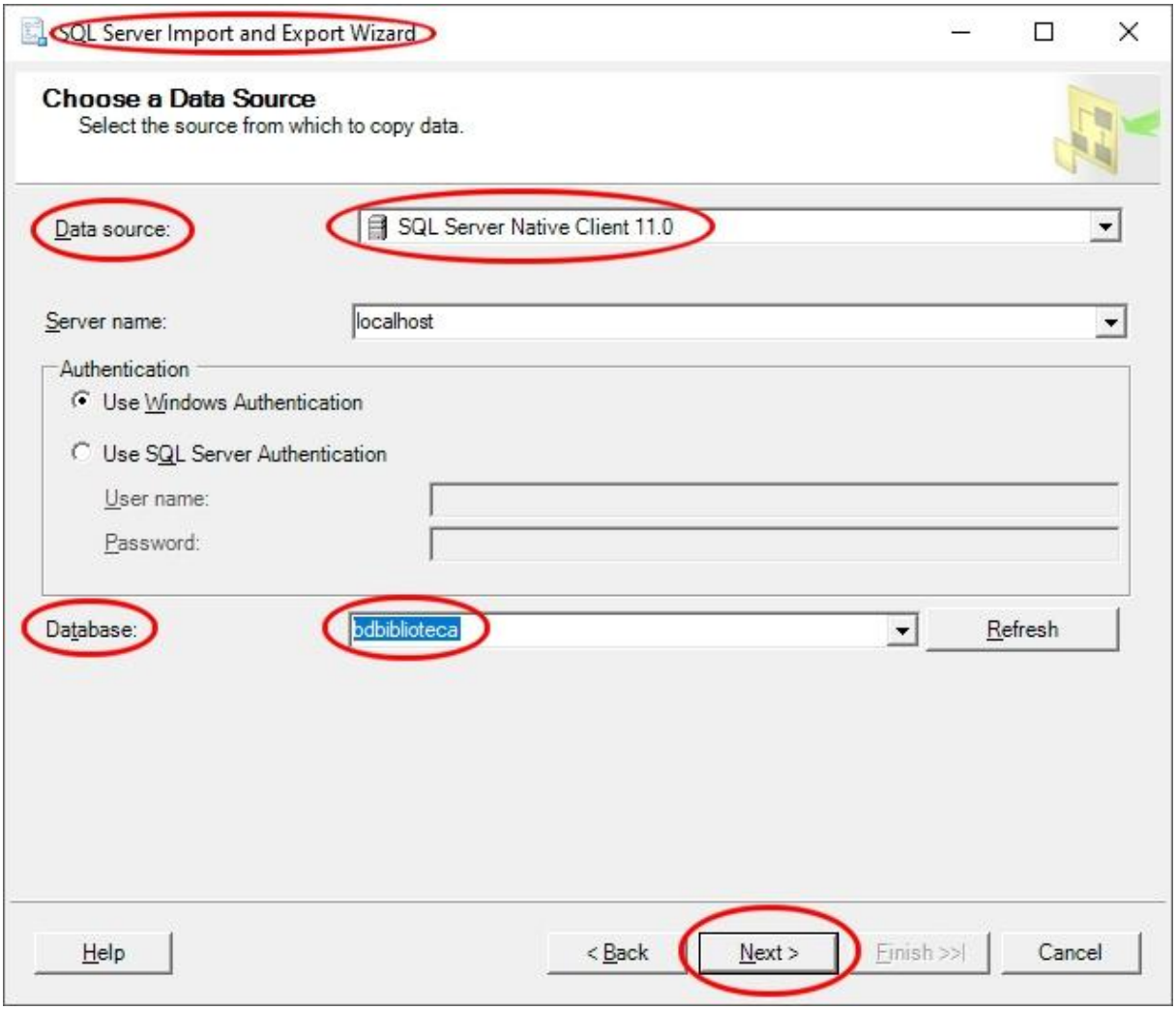
Din *SQL Server Management Studio* putem exporta informaţii/date dintr-o bază de date, totodată transformând setul de astfel de rezultate într-unul lizibil/asimilabil în/de către alte programe.

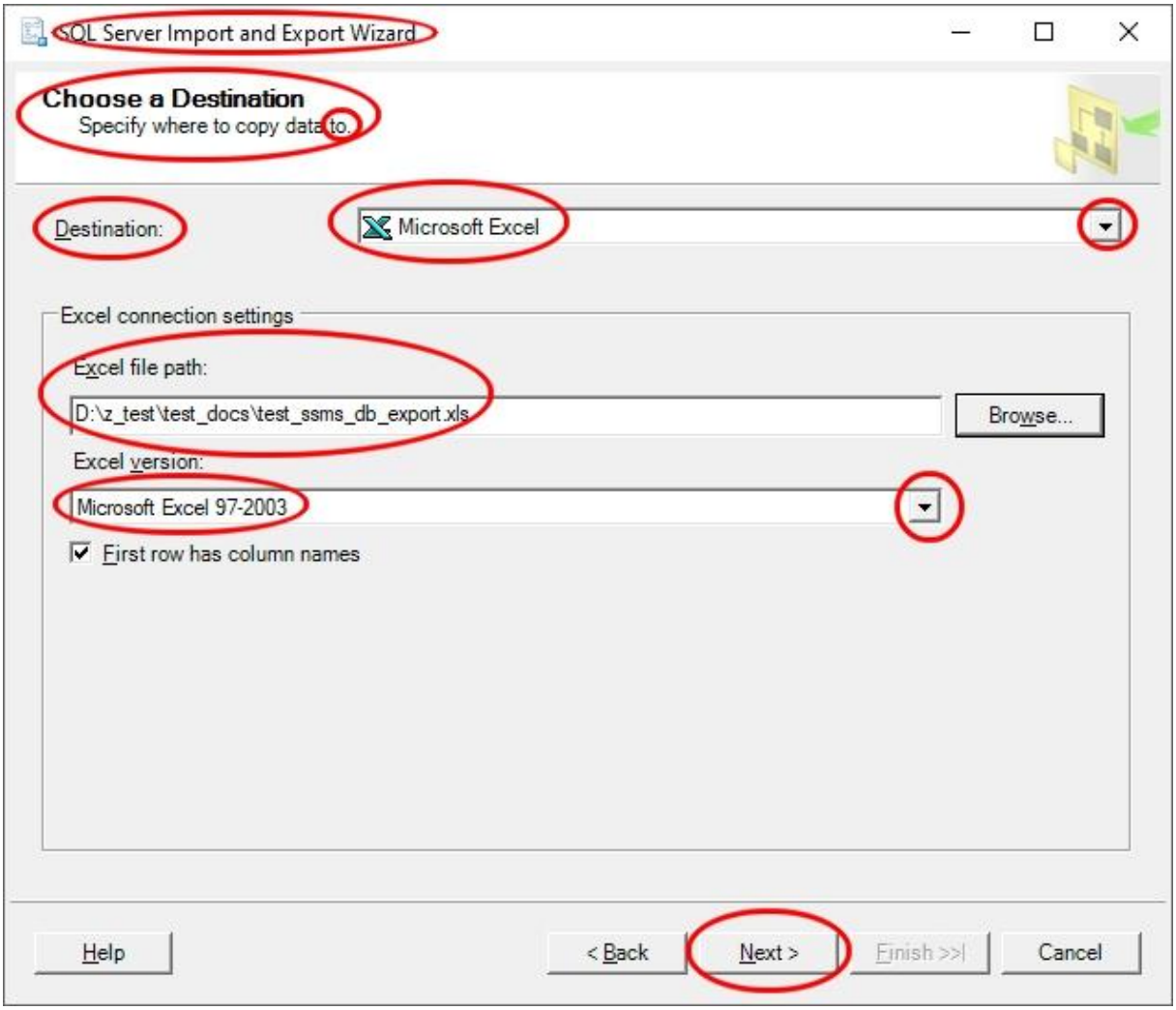
Click-dreapta pe deemirea unei baze de date,
Apoi din meniul "Tasks" se alege comanda "Export Data":

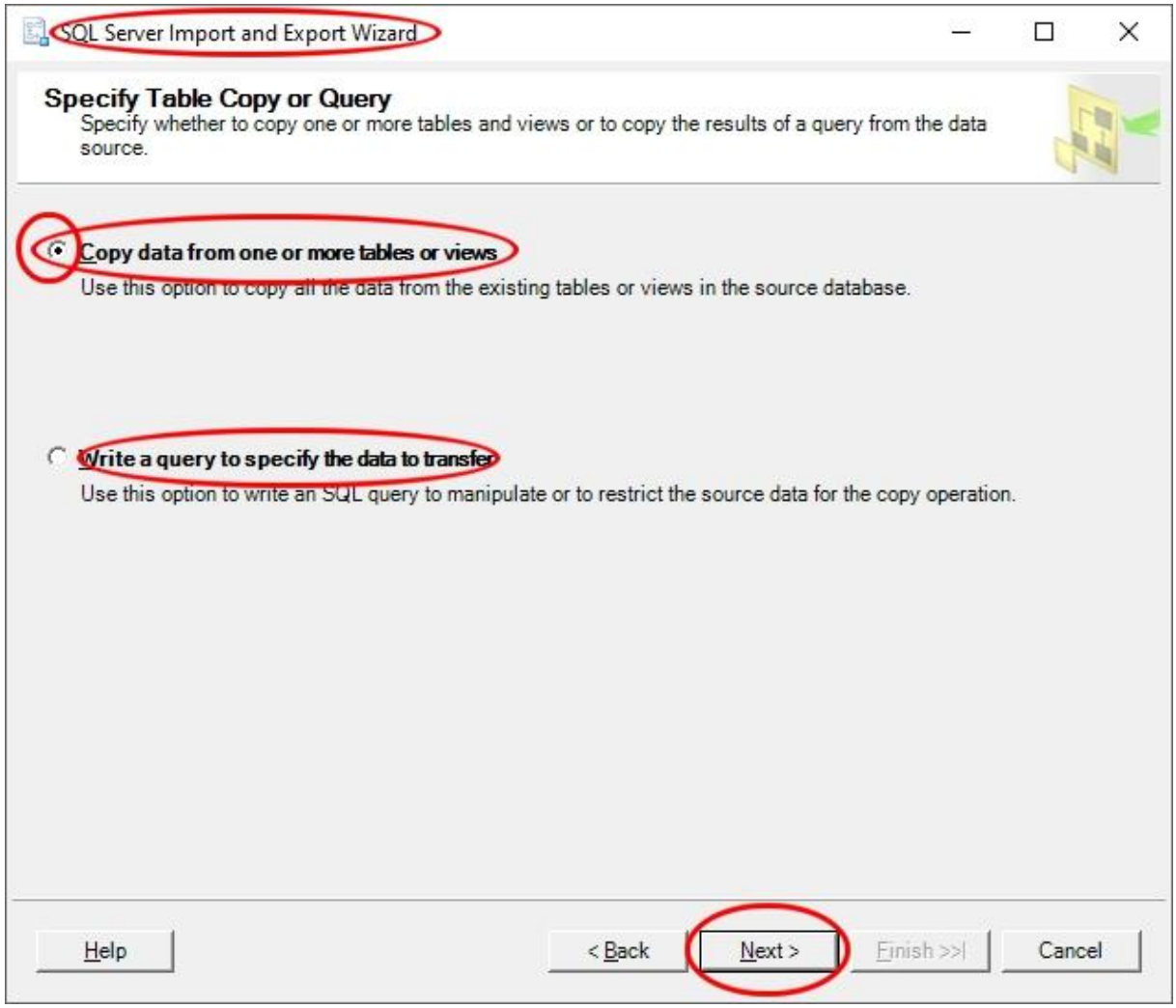


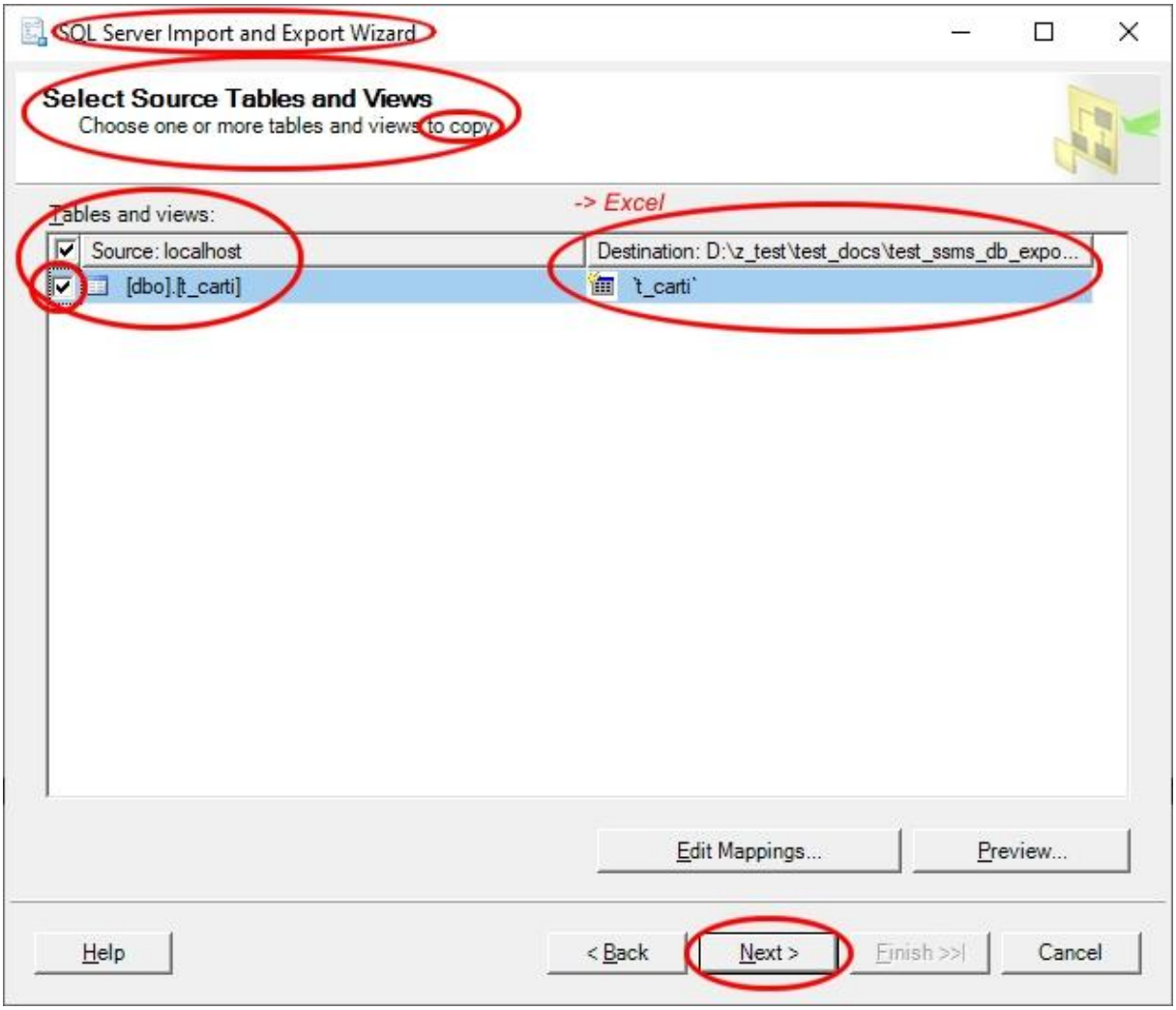

SQL Server Import and Export Wizard
Welcome to SQL Server Import and Export Wizard
This wizard helps you to create simple packages that import and export data between many popular data formats including databases, spreadsheets, and text files. The wizard can also create the destination database and the tables into which the data is inserted.
To move or copy databases and their objects from one server instance to another, cancel this wizard and use the Copy Database Wizard instead. The Copy Database Wizard is available in SQL Server Management Studio.
Do not show this starting page again.
Help
< Back
Next >
Finish >>|
Cancel

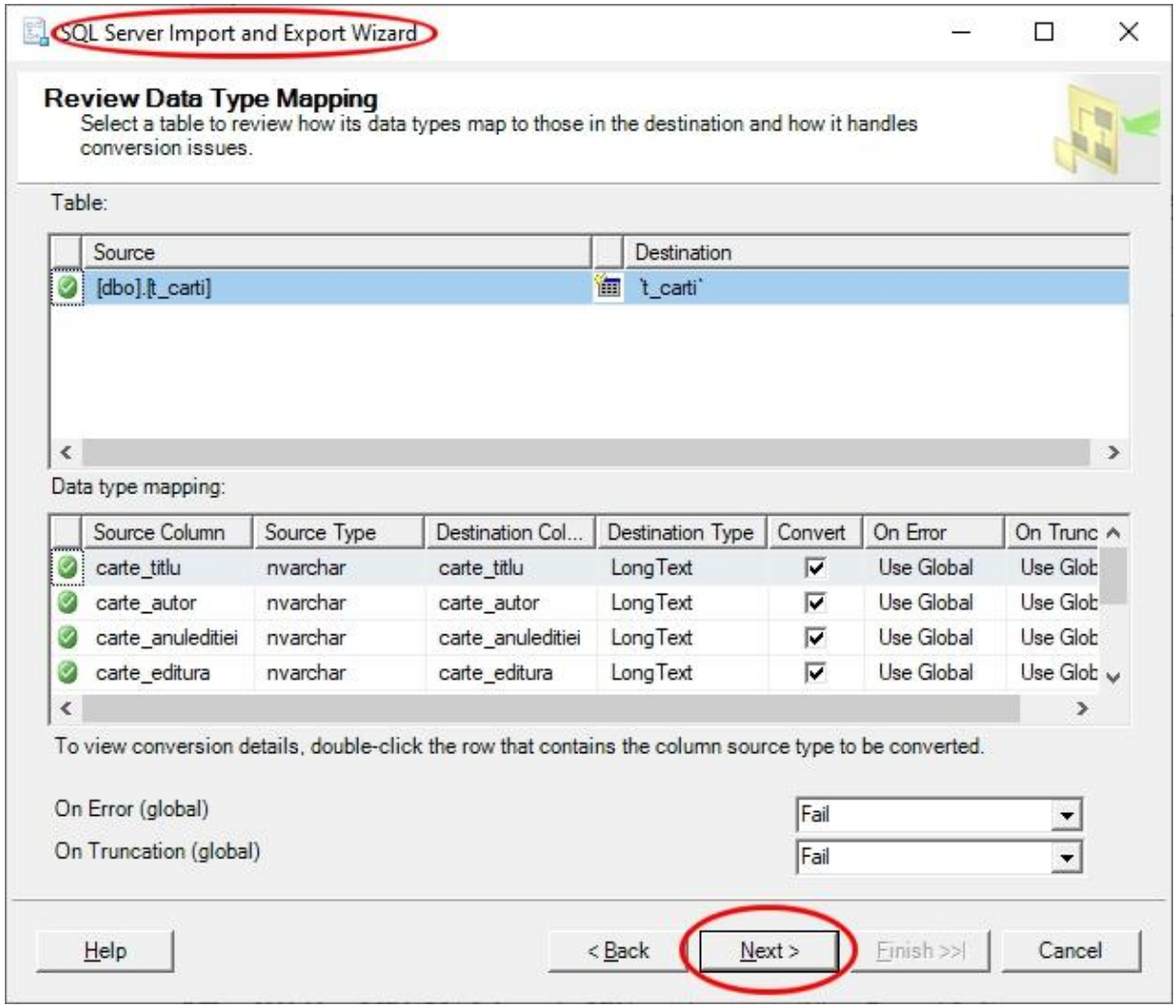

SQL Server Import and Export Wizard
Choose a Data Source
Select the source from which to copy data.
Data source:
.Net Framework Data Provider for Odbc
Microsoft OLE DB Provider for Analysis Services 14.0
Microsoft OLE DB Provider for Analysis Services 9.0
Microsoft OLE DB Provider For Data Mining Services
Microsoft OLE DB Provider for OLAP Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for Search
Microsoft OLE DB Provider for SQL Server
SQL Server Native Client 11.0
Data
ConnectionString
Named ConnectionString
Dsn
Source
Driver
Driver
The name of the ODBC Driver to use when connecting to the Data Source.
Help
< Back
Next >
Finish >>|
Cancel

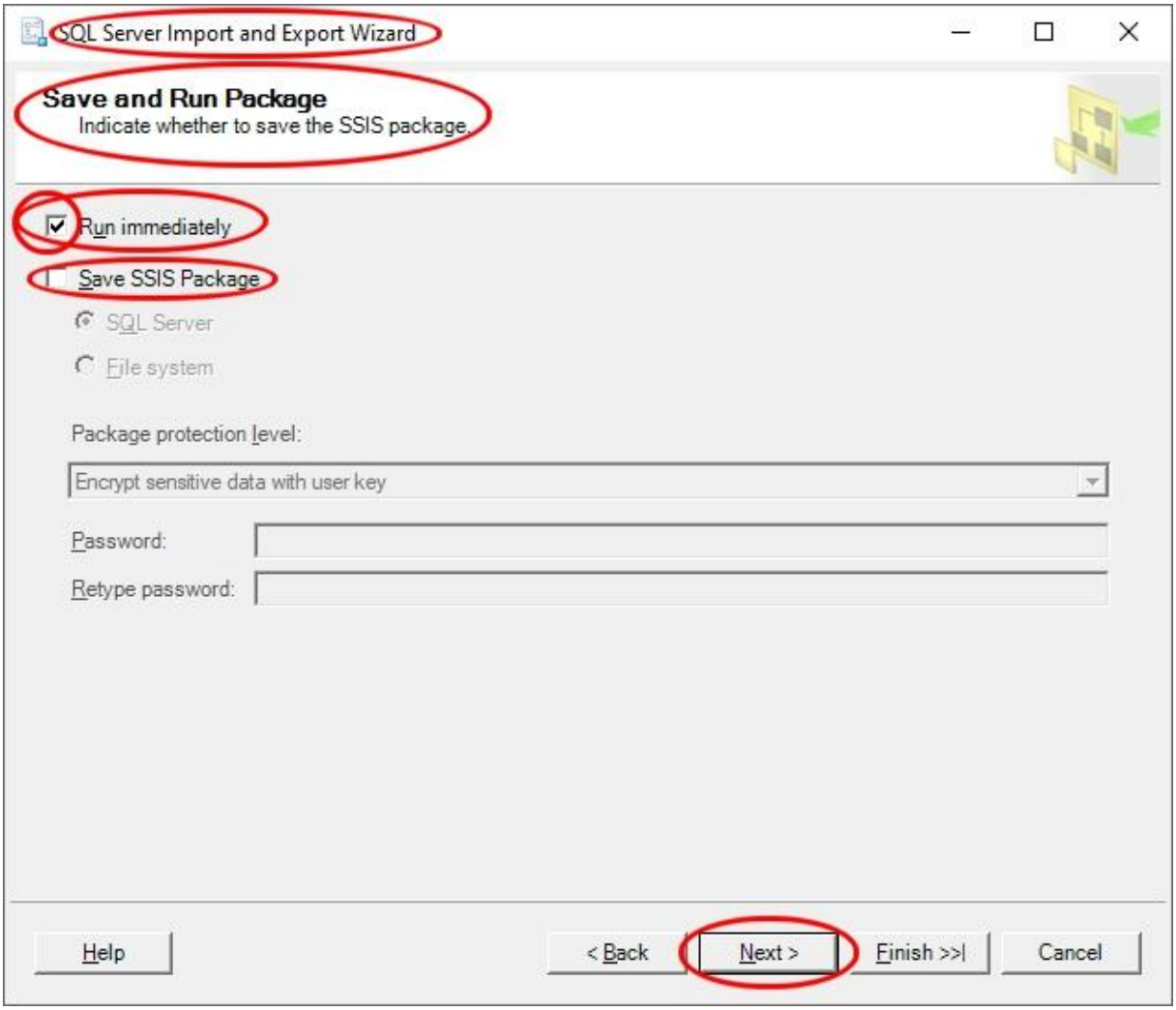

SQL Server Import and Export Wizard
Choose a Data Source
Select the source from which to copy data.
Data source:
SQL Server Native Client 11.0
Server name:
localhost
Authentication
Use Windows Authentication
Use SQL Server Authentication
User name:
Password:
Database:
bdbiblioteca
Refresh
Help
< Back
Next >
Finish >>|
Cancel

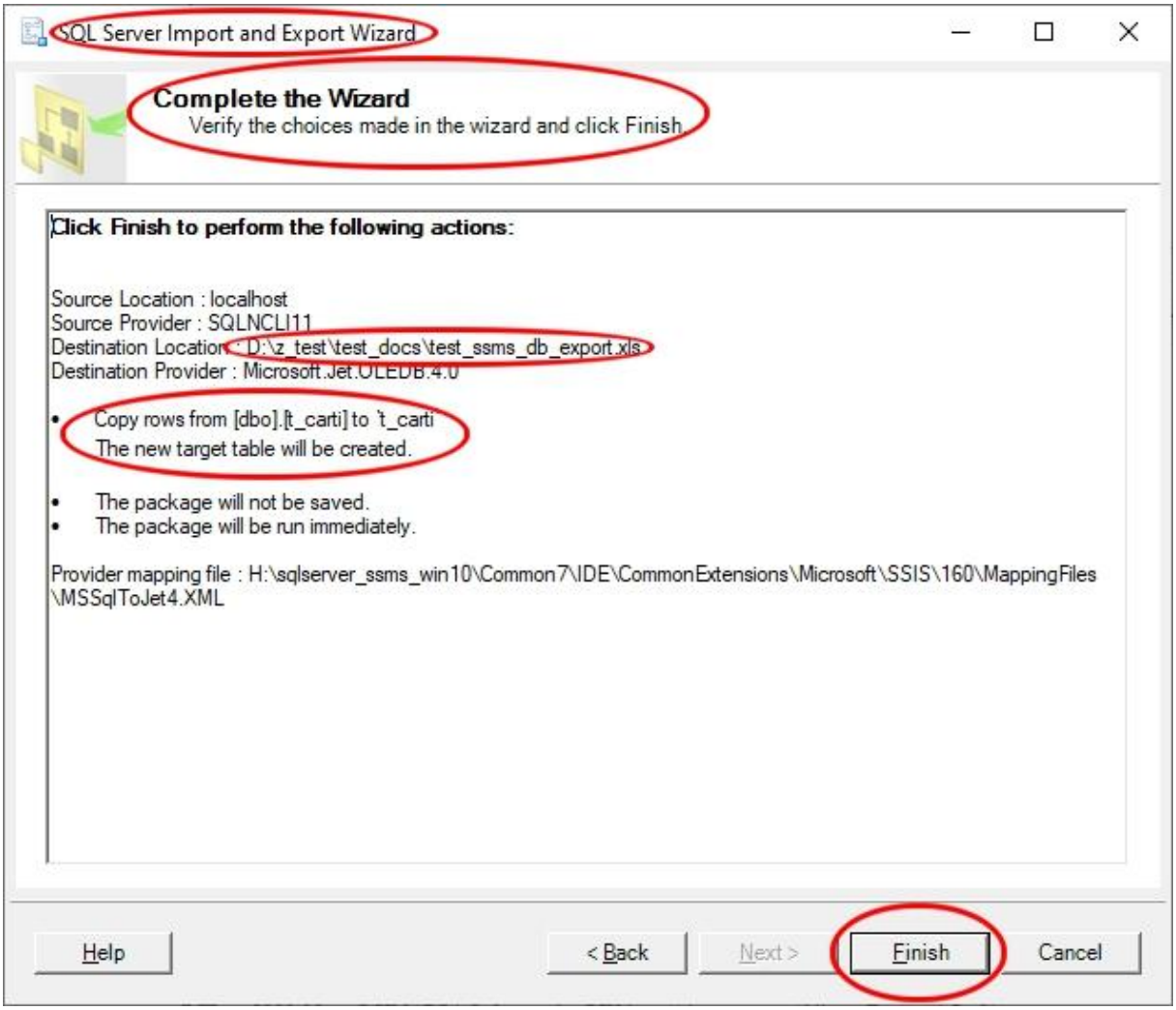

SQL Server Import and Export Wizard
Choose a Destination
Specify where to copy data to.
Destination:
Microsoft Excel
Excel connection settings
Excel file path:
D:\z_test\test_docs\test_ssms_db_export.xls
Browse...
Excel version:
Microsoft Excel 97-2003
First row has column names
Help
< Back
Next >
Finish >>|
Cancel

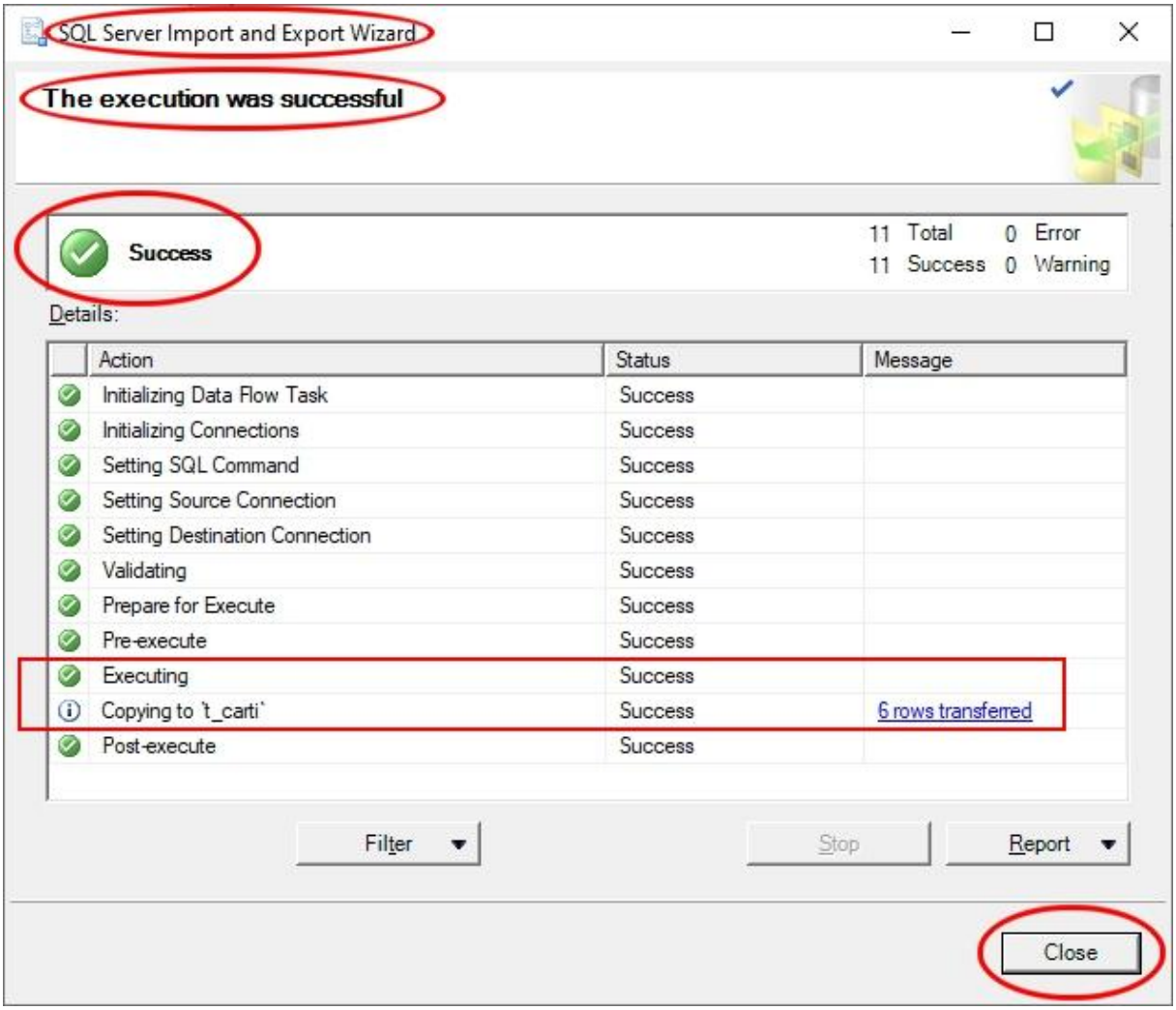

SQL Server Import and Export Wizard
Specify Table Copy or Query
Specify whether to copy one or more tables and views or to copy the results of a query from the data source.
Copy data from one or more tables or views
Use this option to copy all the data from the existing tables or views in the source database.
Write a query to specify the data to transfer
Use this option to write an SQL query to manipulate or to restrict the source data for the copy operation.
Help
< Back
Next >
Finish >>|
Cancel


SQL Server Import and Export Wizard
Select Source Tables and Views
Choose one or more tables and views to copy.
-> Excel
Tables and views:
Source: localhost
Destination: D:\z_test\test_docs\test_ssms_db_expo...
[dbo].[t_carti]
`t_carti`
Edit Mappings...
Preview...
Help
< Back
Next >
Finish >>|
Cancel


SQL Server Import and Export Wizard
Review Data Type Mapping
Select a table to review how its data types map to those in the destination and how it handles conversion issues.
Table:
Source
Destination
[dbo].[t_carti]
`t_carti`
Data type mapping:
Source Column
Source Type
Destination Col...
Destination Type
Convert
On Error
On Trunc
carte_titlu
nvarchar
carte_titlu
LongText
Use Global
Use Glob
carte_autor
nvarchar
carte_autor
LongText
Use Global
Use Glob
carte_anuleditiei
nvarchar
carte_anuleditiei
LongText
Use Global
Use Glob
carte_editura
nvarchar
carte_editura
LongText
Use Global
Use Glob
To view conversion details, double-click the row that contains the column source type to be converted.
On Error (global)
Fail
On Truncation (global)
Fail
Help
< Back
Next >
Finish >>|
Cancel

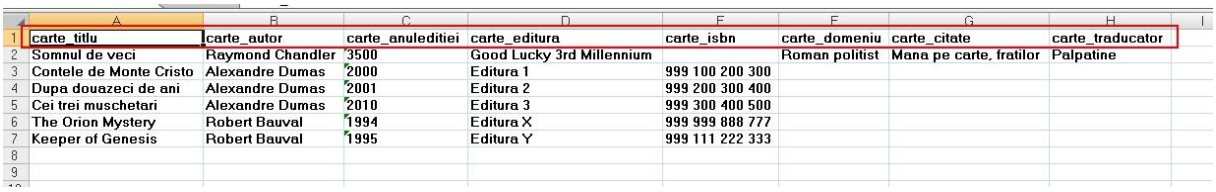

SQL Server Import and Export Wizard
Save and Run Package
Indicate whether to save the SSIS package.
Run immediately
Save SSIS Package
SQL Server
File system
Package protection level:
Encrypt sensitive data with user key
Password:
Retype password:
Help
< Back
Next >
Finish >>|
Cancel


SQL Server Import and Export Wizard
Complete the Wizard
Verify the choices made in the wizard and click Finish.
Click Finish to perform the following actions:
Source Location : localhost
Source Provider : SQLNCLI11
Destination Location : D:\z_test\test_docs\test_ssms_db_export.xls
Destination Provider : Microsoft.Jet.OLEDB.4.0
Copy rows from [dbo].[t_carti] to `t_carti
The new target table will be created.
The package will not be saved.
The package will be run immediately.
Provider mapping file : H:\sqlserver_ssms_win10\Common7\IDE\CommonExtensions\Microsoft\SSIS\160\MappingFiles\MSSqlToJet4.XML
Help
< Back
Next >
Finish
Cancel


SQL Server Import and Export Wizard
The execution was successful
Success
11 Total 0 Error
11 Success 0 Warning
Details:
Action
Status
Message
Initializing Data Flow Task
Success
Initializing Connections
Success
Setting SQL Command
Success
Setting Source Connection
Success
Setting Destination Connection
Success
Validating
Success
Prepare for Execute
Success
Pre-execute
Success
Executing
Success
Copying to `t_carti`
Success
6 rows transferred
Post-execute
Success
Filter
Stop
Report
Close

După ce închidem acest dialog final care anunţă succesul operaţiunii
verificăm transferarea informaţiilor (înregistrărilor) din baza de date şi tabelul specificate:

Navigăm la fişierul Excel®™ în care au fost trimise informaţiile şi îl deschidem:



	A	B	C	D	E	F	G	H	I
1	carte_titlu	carte_autor	carte_anuleditiei	carte_editura	carte_isbn	carte_domeniu	carte_citate	carte_traducator	
2	Somnul de veci	Raymond Chandler	3500	Good Lucky 3rd Millennium		Roman politist	Mana pe carte, fratilor	Palpatine	
3	Contele de Monte Cristo	Alexandre Dumas	2000	Editura 1	999 100 200 300				
4	Dupa douazeci de ani	Alexandre Dumas	2001	Editura 2	999 200 300 400				
5	Cei trei muschetari	Alexandre Dumas	2010	Editura 3	999 300 400 500				
6	The Orion Mystery	Robert Bauval	1994	Editura X	999 999 888 777				
7	Keeper of Genesis	Robert Bauval	1995	Editura Y	999 111 222 333				
8									
9									

Obsrevăm că fiecare coloană
este intitulată la fel cu câmpul din care provin valorile din acea coloană - adică, din acel câmp.

Sintaxa pentru crearea unui tabel

În SQL Management Studio (SSMS) se poate afla sintaxa instrucţiunilor – şi, bineînţeles, instrucţiunile – folosite pentru crearea unui tabel al unei baze de date.

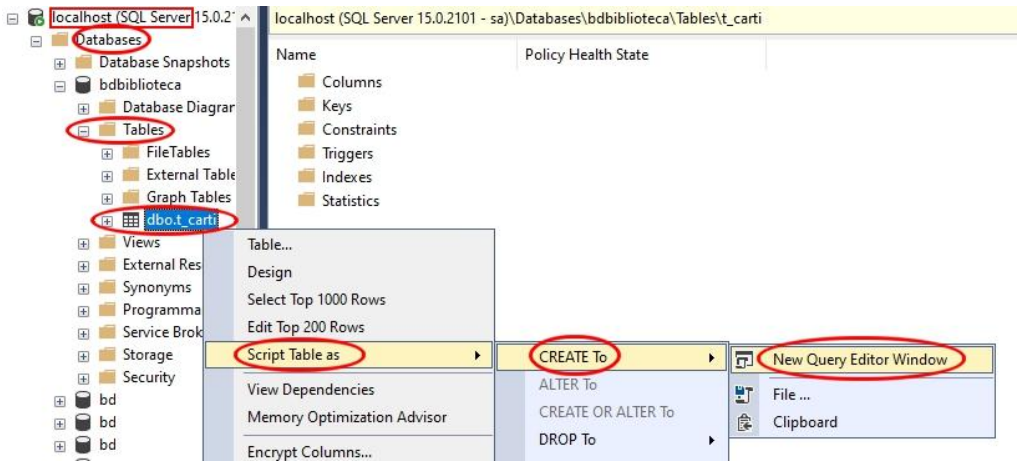
Acestea sunt instrucţiuni care
- pot fi folosite atât în SSMS ca o interogare distinctă pentru a crea un tabel
- pot fi trimise dintr-un program, de exemplu unul creat cu Visual Studio®™
    - ca proprietatea *CommandText*
      al unui obiect de tipul *SqlCommand*
      asociat unui obiect de tipul *SqlConnection*

Aici, vom vedea cum tabelul "*t_carti*" a fost creat în baza de date "*bdbiblioteca*".

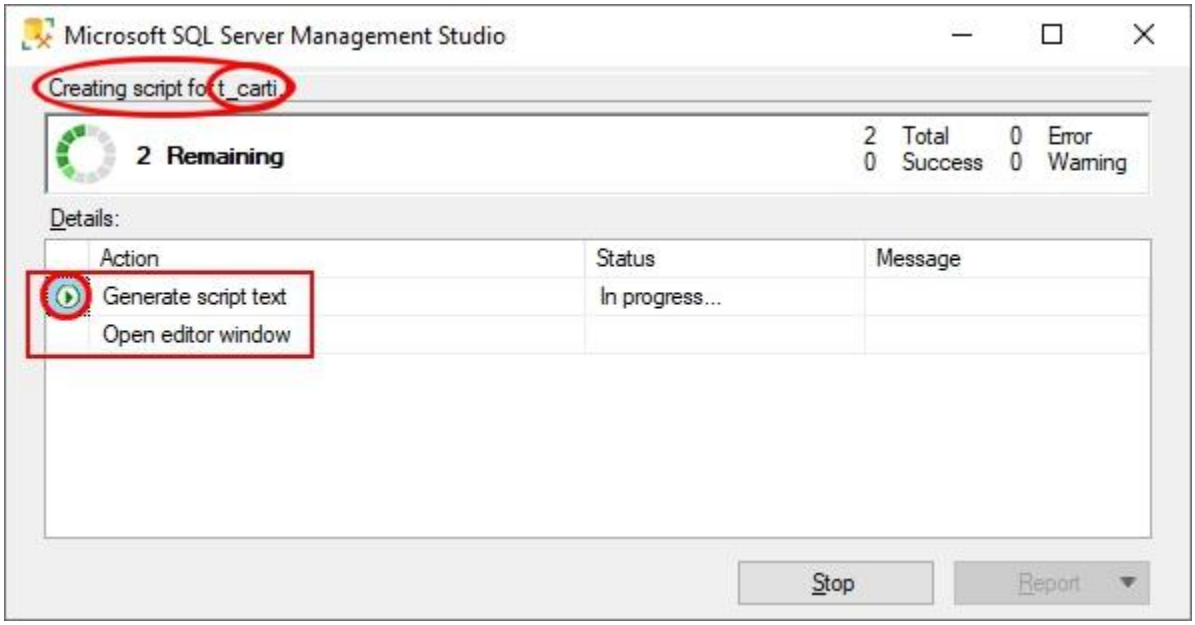
Deci, nu vom crea un tabel, ci vom afla cum anume se creează unul.

Astfel, putem refolosi instrucţiunile afişate pentru a crea un tabel
cu denumire şi câmpuri proprii –
în instrucţiuni vom schimba
"t_carti" la numele noului tabel
şi numele şi tipul câmpurilor la cele care trebuie să se afle în noul tabel

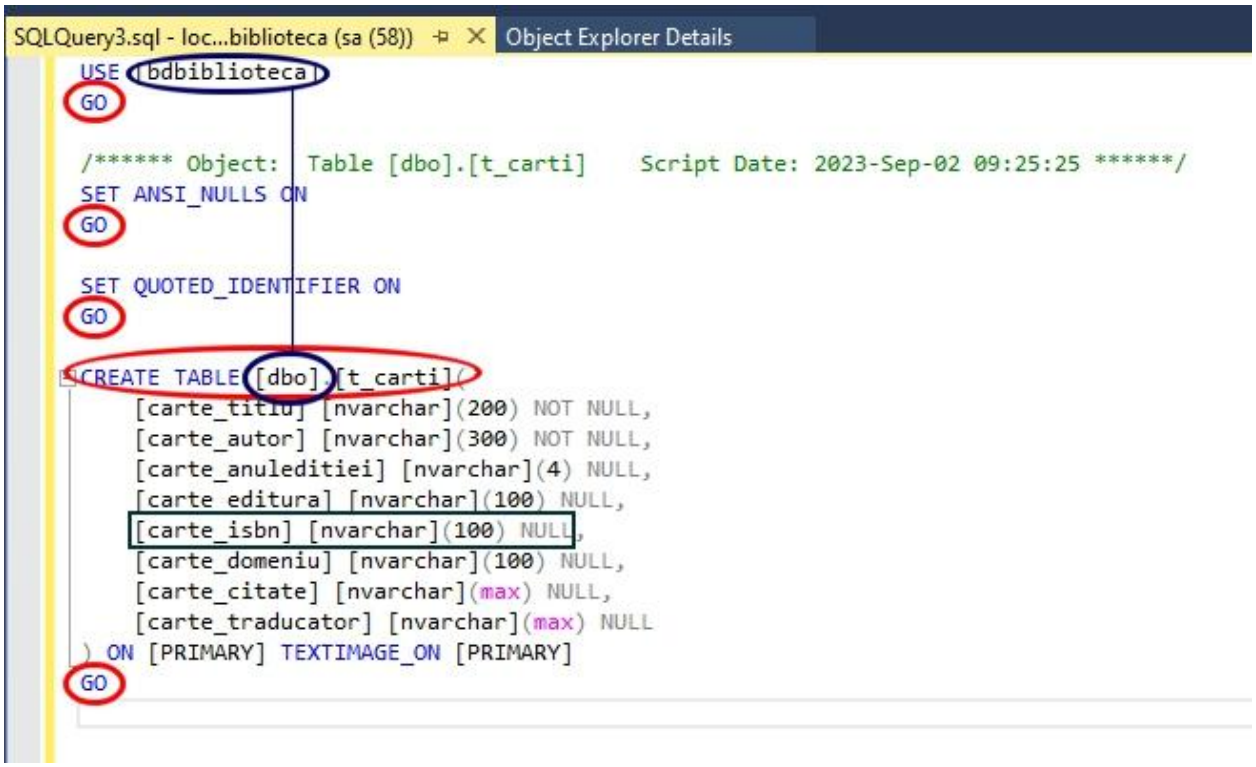
Pornim SSMS.
- desfăşurăm ramura serverului SQL de interes
- apoi ramura "Databases"
- apoi ramura "Tables"
    - unde click-dreapta pe tabelul "t_carti"
        - desfăşurăm meniul "Script Table as" –
          căutăm instrucţiuni SQL asociate acestui tabel
            - apoi meniul "CREATE TO" –
              *solicităm instrucţiunile pentru **Crearea** acestui tabelul*
                - apoi selectăm "New Query Editor Window" –
                  solicităm afişarea instrucţiunilor într-o fereastră/casetă de editare a unei interogări obişnuite



Aşteptăm executarea procedurilor solicitate:



După aceasta vedem blocul de instrucţiuni-comenzi SQL pe care SQL Server le-a folosit "intern" pentru crearea tabelului "t_carti":



În imaginea precedentă remarcăm:

- comanda specificând baza de date pe care se execută comenzile din acest bloc:
  USE bdbiblioteca

- comanda **"GO"**
  - care lansează în execuţie comanda/comenzile premergătoare, de la comanda "GO" precedentă (dacă există)

- comanda **"CREATE"**
  - care iniţiază crearea unui obiect cunoscut în SQL Server
    - aici, un obiect de tipul "**TABLE**"

- **[dbo]**
  semnifică ***baza de date curentă*** - aici, cea tocmai specificată cu comanda **USE**

- punctul separator de niveluri – **caracterul .**
  având acelaşi rost/fiind folosit ca punctul cunoscut din/folosit în Visual Studio®™ –
  anume, separator de niveluri între
  un membru al unui spaţiu de nume şi
  obiecte/proprietăţi ale acelui membru
  aici, tabelul denumit "t_carti" este cu *un nivel mai jos decât baza de date şi aparţine acesteia*:
  [dbo].[t_carti]

- denumirea fiecărui câmp din tabel, de exemplu
  [carte_isbn]
  - parantezele drepte indică un text/o valoare delimitată strict –
    care în comanda SQL trebuie considerată integral, aşa cum este scrisă între paranteze

- tipul şi lungimea (dacă valorile de acel tip au o lungime anume)
  fiecărui câmp, de exemplu un câmp de tipul *nvarchar*, deci cu valori-text
  de exemplu, valori/texte care ***pot fi*** lungi de ***maximum*** (***până la***) 100 caractere/litere:
  [nvarchar](100)
  sau valori-texte care ***pot fi*** lungi cât ***până la*** lungimea maximă, acceptabilă de acel tip valoric:
  [nvarchar](max)

- toleranţa fiecărui câmp la valori nule –
  anume, dacă în acel câmp
  - pot exista valori nule:
    - **NULL**
  - sau nu pot exista valori nule:
    - **NOT NULL**

  altfel spus, dacă din acel câmp pot lipsi valori