

Crearea bazelor de date SQL Server



Gestiunea informației

2016

Scopul lucrării de laborator:

1. Pornirea SQL Server Management Studio-ului (SSMS)

Trebuie reținut că SQL Server-ul este serverul de baze de date, iar Management Studio-ul este un instrument pentru conectarea la server și pentru manipularea bazelor de date gestionate de către server. La un server se pot conecta mai mulți clienți, deci, implicit, ne putem conecta cu mai multe sesiuni de Management Studio.

După instalarea SQL Server-ului și a Management Studio-ului, în funcție de versiunea de Windows, veți găsi ceva similar cu imaginea de mai jos.

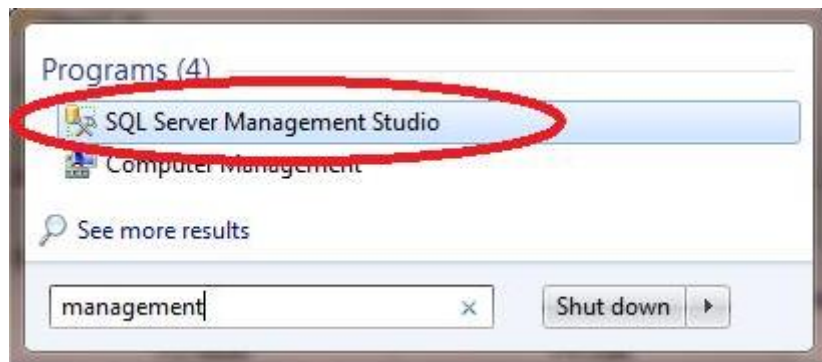


Fig. 1

Dacă instalarea a avut loc cu succes, la pornirea Management Studio-ului apare fereastra de conectare la server. Parametrii sunt cei din imagine. De aici trebuie să reținem numele serverului (MOBIL\SQLEXPRESS în cazul de față) și faptul că ne vom conecta cu Windows Authentication. Pentru pornire, clic pe Connect.



Fig. 2.

2. Crearea bazei de date.

Atunci când vorbim de o bază de date ne referim pe de o parte la crearea, citirea, modificarea și ștergerea **structurii** sale, pe de altă parte, la manipularea **conținutului** său (inserare de date noi, modificarea, respectiv ștergerea datelor existente, precum și la extragerea datelor după anumite criterii într-un timp fezabil – operații **CRUD**).

Presupunând că ne-am conectat deja și că totul e în regulă, o să apară fereastra principală a SSMS-ului. În general, în partea stângă apar elementele serverului: Bazele de date, Securitate, etc.

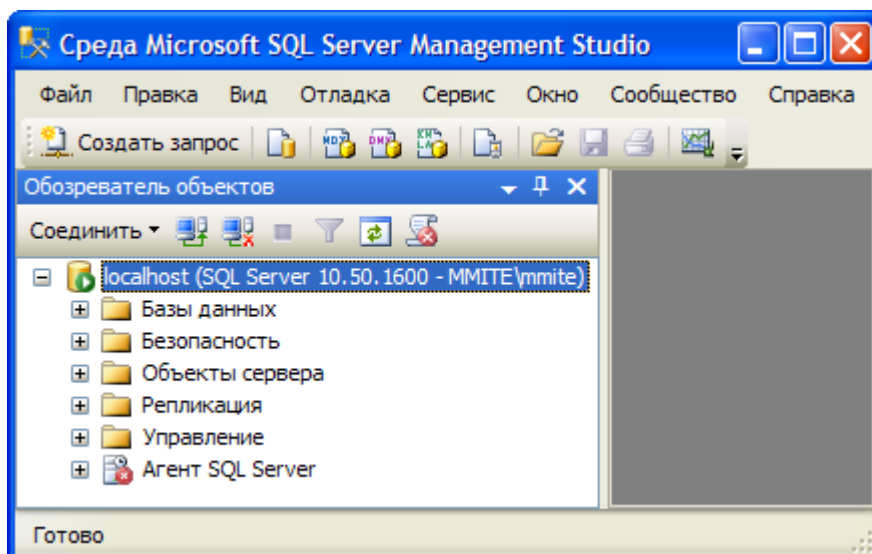


Fig. 3

O bază de date (cu toate obiectele sale) poate fi creată direct în SSMS în designer sau apelând la scrierea de cod SQL.

Варианта 1 – Creare BD în Designer

a. Crearea bazei de date

Dacă deschidem folderul Databases o să observăm că există baze de date sistem și bazele de date ale utilizatorilor.

Pentru exemplificare o să considerăm o bază de date care conține tabelele: **Persoane**, **Telefoane**.

Pentru crearea bazei de date, facem clic dreapta pe *Databases*, apoi clic pe *New Database...* Pot să existe mici diferențe de la o versiune la alta de SQL Server.

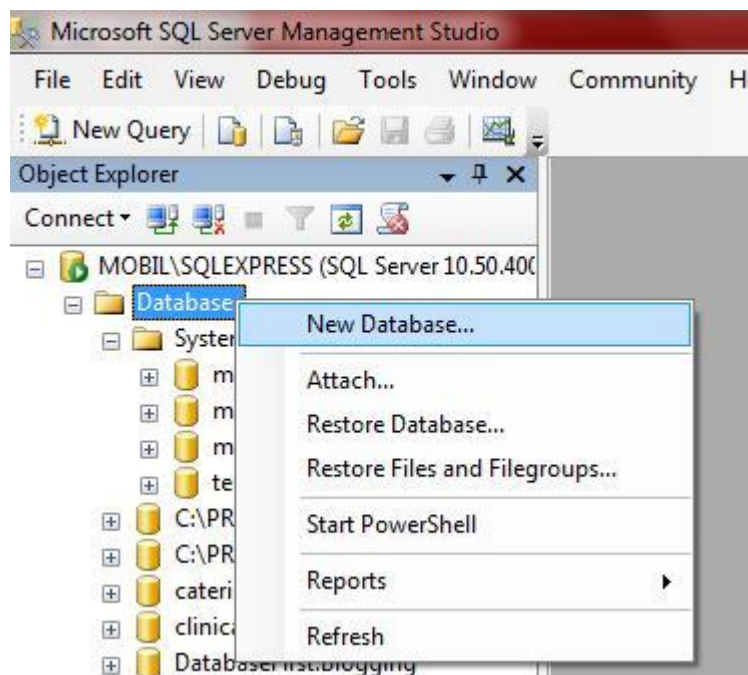


Fig. 4.

În fereastra New Database, scriem numele (denumirea) bazei de date (*TestStudenti* în exemplul de mai jos) și dăm clic pe *OK*. Ceilalți parametri rămân impliciți.

Trebuie reținut că se creează un fișier cu extensia *mdf* (ex: TestStudenti.mdf) și un fișier cu extensia *ldf* (ex: TestStudenti_log.ldf), care vor fi stocate în folderul DATA din SQL Server.

Exemplu: c:\Program Files (x86)\Microsoft SQL Server\MSSQL10_50\SQLEXPRESS\MSSQL\DATA. Datele vor fi stocate în fișierul *mdf*.

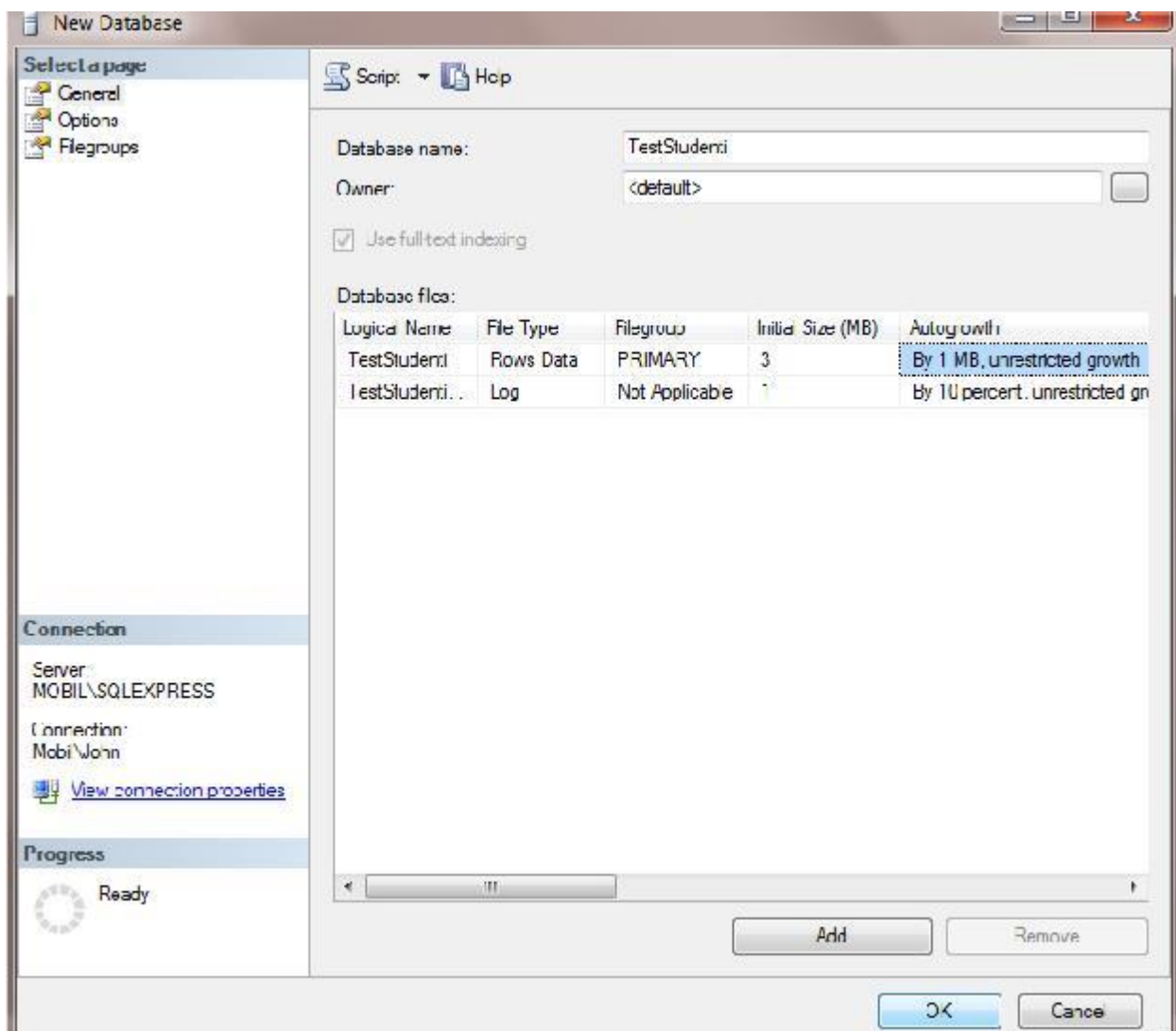


Fig. 5.

În partea stângă o să observăm că a apărut noua bază de date. Dacă extindem cu clic pe + -ul din fața denumirii o să vedem elementele importante ale unei baze de date SQL Server: Diagramele, Tabelele, Vederile, aspectele de programare (procedurile stocate, trigerele, funcțiile), precum și aspectele privind securitatea.

Următorul pas este reprezentat de crearea tabelor și a câmpurilor acestora – structura tabelor.

b. Crearea tabelor

Similar bazelor de date, și în cazul tabelor există tabele sistem pentru fiecare bază de date și tabele utilizator pe care le creăm noi, așa cum se vede în fig. 6.

Clic dreapta pe *Tables*

Clic pe *New Table*

În zona de lucru a SSMS, se deschide o fereastră pentru crearea tabelelor. Aici vom introduce **denumirea** câmpurilor (coloanelor, impropriu zis) și a **tipurilor** de date. Există o asemănare cu denumirea variabilelor și a tipurilor acestora din limbajele de programare.

Înainte de a crea efectiv tabela și câmpurile acesteia trebuie să știm câteva lucruri despre tipurile de date și constrângerile aplicate asupra câmpurilor.

Tipuri de date Microsoft SQL Server

Microsoft SQL Server conține următoarele tipuri de date predefinite, împărțite în grupe:

Tipuri șir de caractere ASCII.

- **char (n)** – șir de caractere de dimensiune fixă de maxim 8000 caractere. Este folosit în câmpurile cu număr fix de caractere (EX: CNP – char (13)). Șirul de caractere se va completa cu caractere spațiu dacă mărimea curentă a șirului este mai mică decât n.
- **varchar (n)** - șir de caractere de dimensiune variabilă de maxim 8000 caractere. Este cel mai des folosit în bazele de date care nu stochează șiruri de caractere ce conțin diacritice (ș,ț,â,ă,î, etc). Avantajul folosirii sale este că nu se stochează inutil spații goale și, deci, nu crește mărimea bazei date (necesită explicație);
- **varchar (MAX)** - șir de caractere de dimensiune variabilă de maxim 1073741824 caractere. Firește, e folosit rar și doar atunci când e necesară stocarea unor volume foarte mari de date;
- **text** - șir de caractere de dimensiune variabilă de max 2 Gb. După cum se vede în acest caz dimensiunea este dată byți și nu în număr de caractere. Șirul de caractere este memorat în pagini de 8ko fiecare.
- **nchar** - șiruri de caractere UNICODE de lungime fixată (maxim 4000 caractere)
- **nvarchar** - șiruri de caractere UNICODE de lungime variabilă (maxim 4000 caractere)
- **ntext** - șiruri de caractere UNICODE de lungime variabilă (lungimea maximală $2^{30} - 1$, sau 1,073,741,823 caractere)

Tipuri șir de caractere Unicode.

Pentru cele 4 tipuri de mai sus există câte un omolog pentru șiruri de date de tip Unicode (care pot stoca și diacritice) (nchar(n), nvarchar(n), nvarchar(MAX), ntext). Deoarece un caracter Unicode este dtocat pe 2 byți, spre deosebire de Ascii care e stocat pe 1 byte, numărul de caractere se reduce la jumătate față de tipurile ASCII. Nu contează acest lucru pentru ntext, care stochează, normal, tot 2 Gb.

În cazul bazelor de date care stochează date în limba română sau alte limbi care au diacritice (maghiară, germană, franceză, etc), precum și alte seturi de caractere (chineză, rusă, greacă, japoneză) vom folosi tipuri din această categorie, cel mai des *nvarchar(n)*.

Tipuri binare.

- **bit** – permite stocarea lui 0, 1 și NULL. E folosit în general pentru valori de tip boolean. (Ex: Activ – 0 sau 1).
- **binary [(n)]** – șir binar de lungime fixată (maximum 8000 octeți). Se folosește pentru stocarea unor secvențe de biți. Valorile de tip binar sunt reprezentate în sistem hexazecimal și se introduc uzual tot în hexazecimal (precedate de 0x).
- **varbinary [(n)]** – șir binar de lungime variabilă (maximum 8000 octeți).
- **image** – șir binar de lungime fixată (maximum $2^{31} - 1$ sau 2,147,483,647 octeți).
- **timestamp** – O valoare de tip timestamp este o valoare specială de tip binary(8). Tipul timestamp garantează unicitatea valorilor coloanei asociate. Un tabel poate avea o singură coloană de tip timestamp. Valoarea coloanei de tip timestamp este modificată automat după fiecare modificare a tuplei. Ea ne arată ordinea operațiilor efectuate de SQL Server. Marcile de timp (timestamp) se pot folosi pentru a împiedica doi utilizatori să modifice aceeași tuplă. Tipul timestamp nu reprezintă data și oră. Valoarea timestamp ce va fi înscrisă ca marcă la următoarea modificare sau inserare de linie poate fi accesată prin intermediul variabilei globale @@DBTS.
- **uniqueidentifier** – reprezintă un identificator unic global (GUID) pe 16 octeți și asigură unicitatea valorilor la nivelul bazei de date. Generarea în Transact SQL a unui nou uniqueidentifier se face cu NEWID()

Tipuri numerice.

- **tinyint** – nr întregi cuprinse între 0 și 255 (1 byte)
- **smallint** - nr întregi cuprinse între -32768 și 32767 (2 byte)
- **int** – nr întregi cuprinse între aprox -2 mld și 2 mld (4 octeți)
- **bigint** - nr întregi cuprinse între aprox -9 mld de mld și 9 mld de mld (1018) (8 octeți)
- **Decimal[(p[,s])]** – stochează numere cu precizie și magnitudine fixă, unde **p** este precizia, **s** – nr maxim de cifre zecimale, valori posibile între $-10^{38} - 1$ și $10^{38} - 1$.
- **numeric[(p[,s])]** – echivalent cu tipul decimal.
p (precizia) – numărul total de cifre care pot fi stocate, inclusiv partea întreagă și partea zecimală. Precizia poate lua valori de la 1 la 38. Valoarea implicită a lui p este 18.
s (scale) – numărul de cifre zecimale. Poate lua valori de la 0 la p. Valoare implicită 0.

Numărul de octeți alocați tipului decimal/numeric depinde de precizie după cum urmează:

Precizia	• Nr octeți
1 - 9	5
10-19	9
20-28	13
29-38	17

- **real** – Poate să rețină numere zecimale numere pozitive și negative în virgula flotantă din intervalul de la $3.4E -38$ până la $3.4E + 38$ cu o precizie de 7 cifre. Este reprezentat pe 4 octeți.
- **float[(n)]** – Dacă se specifică o valoare între 1 și 7 pentru n, tipul definit este similar cu tipul real, iar dacă nu se specifică nicio valoare pentru n sau se specifică o valoare între 8 și 15, numerele stocate se pot afla în intervalul de la $-1.79E -308$ până la $1.79E + 308$ (pozitive și negative).

Observație: Valorile în virgulă mobilă sunt supuse erorilor de rotunjire. Ele asigură acuratețe numai până la numărul de cifre specificat ca precizie. De exemplu, în cazul unei precizii de 7 cifre este posibilă stocarea unui număr cu mai mult de 7 cifre, dar nu se garantează că cifrele începând cu a 8-a mai reprezintă exact numărul stocat.

Tipuri folosite în calculele financiare.

- **smallmoney** – stocat pe 4 octeți, stochează valori de până la 214 mii, puțin utilizabil, valori în intervalul $[-214,748.3648, +214,748.3647]$.
- **money** – stocat pe 8 octeți, stochează date de până a 922 de mii de mld, suficient pentru cele mai multe tipuri de calcule contabile, financiare, etc. (intervalul de la -2^{63} ($-922,337,203,685,477.5808$) până la $2^{63} - 1$ ($+922,337,203,685,477.5807$)).

Tipuri pentru stocarea timpului și a datei calendaristice.

- **datetime** – de la 1 ian 1753 la 31 dec 9999, cu o rezoluție (acuratețe) de 3,33 ms). Este cel mai des utilizat tip de date pentru date calendaristice, suficient pentru majoritatea aplicațiilor. Este reprezentat pe 8 octeți și păstrează data și ora. Timpul se definește cu exactitate de sutimi de secunde ($1/300$ dintr-o secundă).
- **smalldatetime** – Este reprezentat pe 4 octeți și păstrează data și ora. Data poate lua o valoare din intervalul de la 1 ianuarie anul 1900 până la 6 iunie anul 2097. Timpul se păstrează cu o acuratețe de 1 minut.

- date – 1 ian 0001 la 31 dec 9999, rezoluție: 1 zi. E folosit atunci când dorim să stocăm doar data și nu ne interesează ore, min, etc. E frecvent utilizat în aplicații uzuale
- time – stochează orele cu o rezoluție de 100 ns. Folosit pentru câmpuri în care suntem interesați numai de stocarea timpului fără a interesa data.
- timestamp – amprentă de timp. E folosit atunci când dorim să stocăm momentul în care cineva a făcut o modificare, ștergere, etc din baza de date, precum și în alte cazuri similare. Într-o tabelă putem avea un singur câmp de acest tip.

Deasemenea tot în acest moment putem preciza tipurile de constrângeri la care e supus fiecare câmp în parte. Exemple de **constrângeri**:

- -PRIMARY KEY – precizează câmpul (câmpurile) care va fi cheie primară. Cheia primară poate fi simplă (formată dintr-un singur câmp), sau compusă (formată din mai multe câmpuri). Foarte important în cazul cheii primare e faptul că în acest câmp (câmpuri) se stochează valori unice la nivelul întregii tabele. Scopul său este să se identifice „în mod unic” fiecare rând din tabelă. Orice câmp cheie primară va fi setat la NOT NULL și la UNIQUE.
- NOT NULL – nu permite introducerea de valori null în câmpul respectiv. Cu excepția câmpurilor care sunt obligatorii (ID, Nume, în exemplul de mai sus, de obicei celelalte câmpuri sunt setate la NULL).
- UNIQUE – nu permite introducerea de valori egale în câmpul unic pentru rânduri diferite.
- CHECK – forțează introducerea într-un câmp a unui domeniu precizat de valori. Ex: valori pozitive; pentru un câmp numit Luna am putea forța să permită numai valori cuprinse între 1 și 12.
- DEFAULT – este o valoare specifică unui câmp, pe care o precizăm în timpul creării tabelei. De fiecare dată când vom introduce un rând nou în tabelă, dacă pentru câmpul respectiv nu precizăm nicio valoare, atunci în câmpul respectiv va fi stocată implicit valoarea precizată.
- FOREIGN KEY – precizează câmpul (câmpurile) cheie străină pentru a putea crea asocieri între tabele.

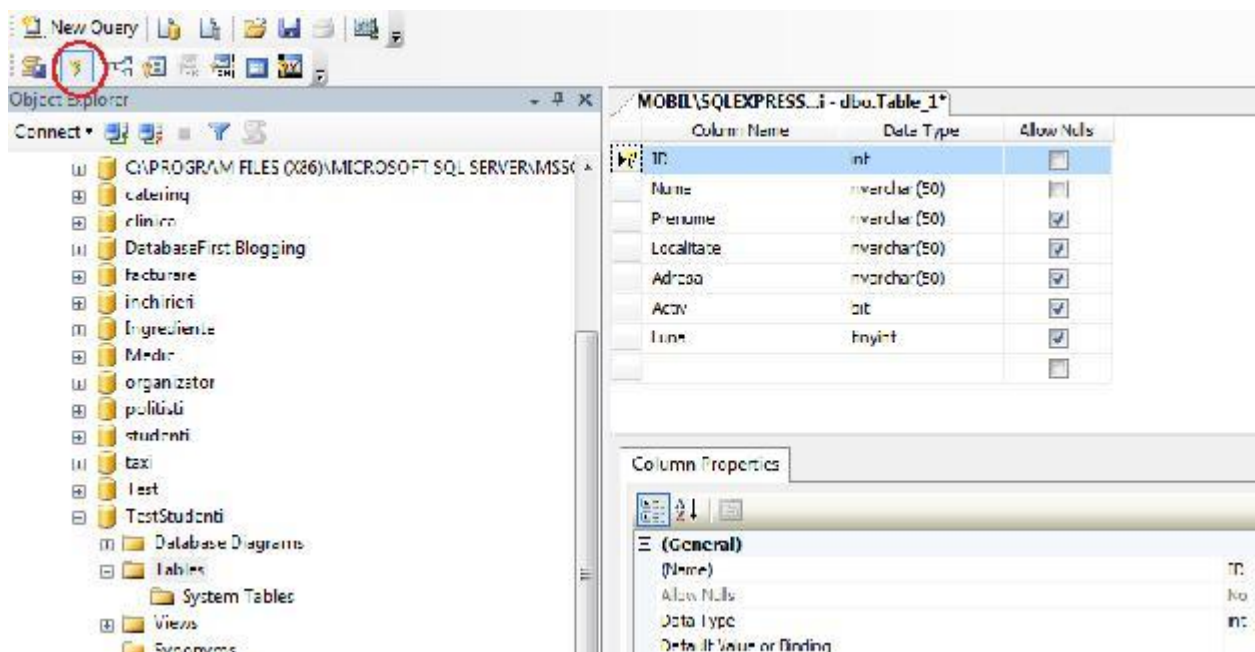


Fig. 7.

În imagine se vede că am creat 7 câmpuri.

Câmpul ID, de tip int, a fost setat să fie cheie primară. Setarea, respectiv desetarea se face prin clic pe butonul cheie primară din bara Table Designer, buton încercuit cu roșu în imagine. În momentul în care setăm un câmp cheie primară, este automat setat să nu permită NULL-uri și de asemenea este setată constrângerea UNIQUE. Câmpul fiind de tip int putem să gestionăm noi valorile sale sau putem să setăm ca în momentul inserării unui rând nou valorile să fie gestionate automat (autonumber). Pentru aceasta, în partea de jos, la Proprietăți (*Column Properties*), căutăm *Identity Specification* pe care îl expandăm cu clic pe plus și setăm *Is Identity* la *Yes*. Se vor activa *Seed* și *Increment*. Seed reprezintă valoarea de start, iar Increment pasul cu care va crește valoarea cheii primare.

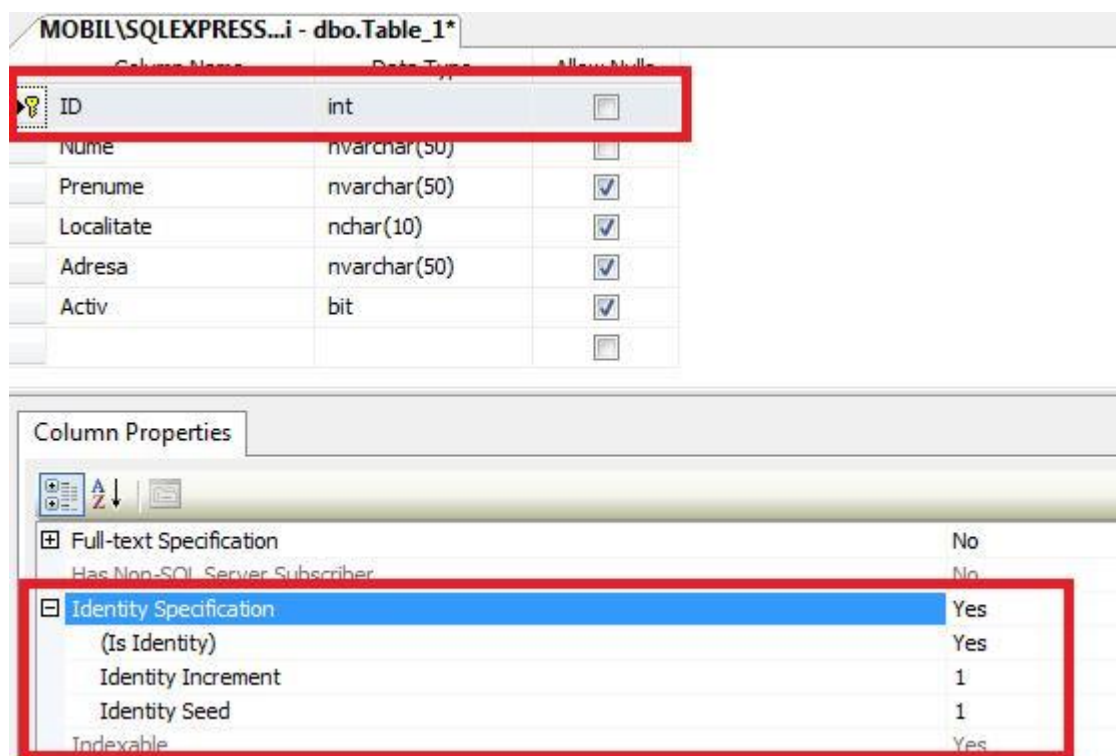


Fig. 8.

Pe câmpurile Nume și Prenume le vom face de tip nvarchar de 50 de caractere. Pentru că dorim să nu existe rânduri fără nici un fel de date (cu excepția cheii primare care e un câmp surogat, deci nu are nici un sens fizic), vom stabili ca cel puțin Numele să fie obligatoriu. Pentru aceasta vom debifa Permite valori Null (Allow Nulls). Observăm că toate câmpurile sunt implicit SET NULL. În cazul câmpului din cheia primară (ID), debifarea aceasta s-a făcut automat.

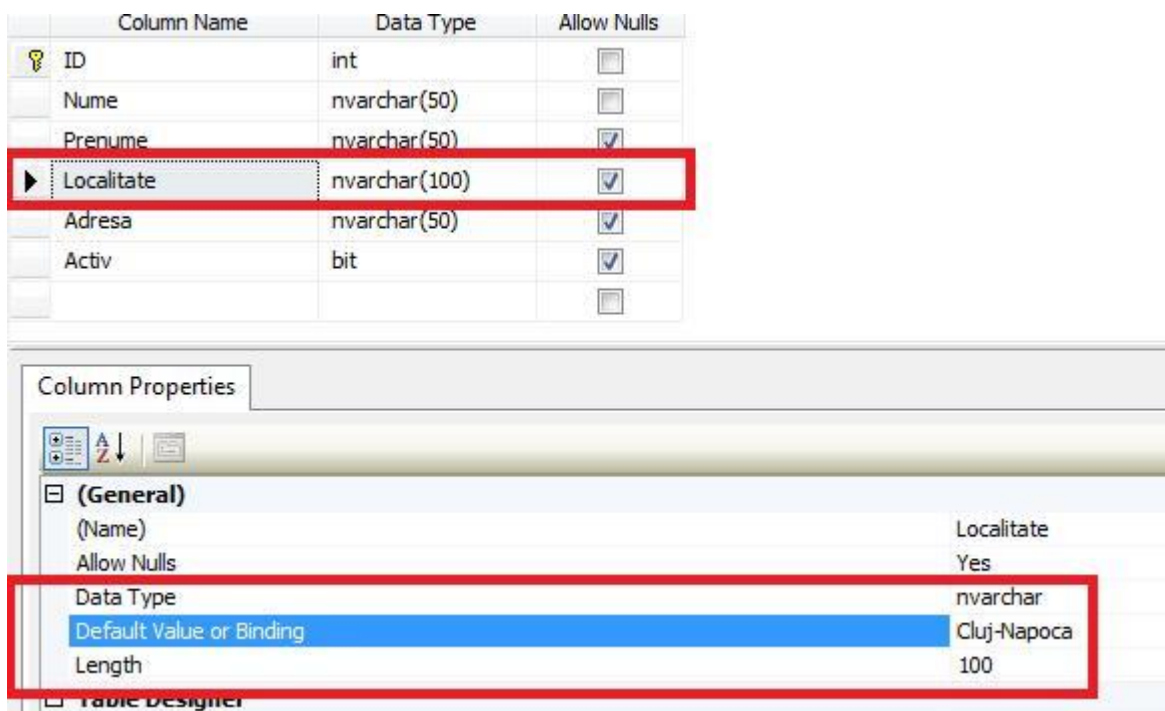


Fig. 9.

În cazul câmpului Localitate vom stabili tipul tot nvarchar, dar de 100 de caractere și suplimentar vom stabili o constrângere DEFAULT la *Cluj-Napoca* așa cum se vede în fig. 9. Deoarece tipul este Unicode, valoarea *Cluj-Napoca* va fi transformată în *N'Cluj-Napoca'*. Dacă tipul ar fi fost ASCII, de ex varchar, valoarea ar fi fost transformată în *'Cluj-Napoca'*.

Deoarece câmpul Activ este boolean l-am setat la tipul bit.

În cazul câmpului Luna (pe care l-am adăugat doar pentru exemplificare) vom crea o constrângere CHECK în care forțăm să permită numai numere de lună valide (de la 1 la 12).

Clic dreapta pe câmp, apoi clic pe Check Constraints... În fereastra care apare clic pe Add Schimbăm numele (Name) ca în fig. 10. În Expression introducem: *Luna>0 and Luna<13* Apoi clic pe Close.

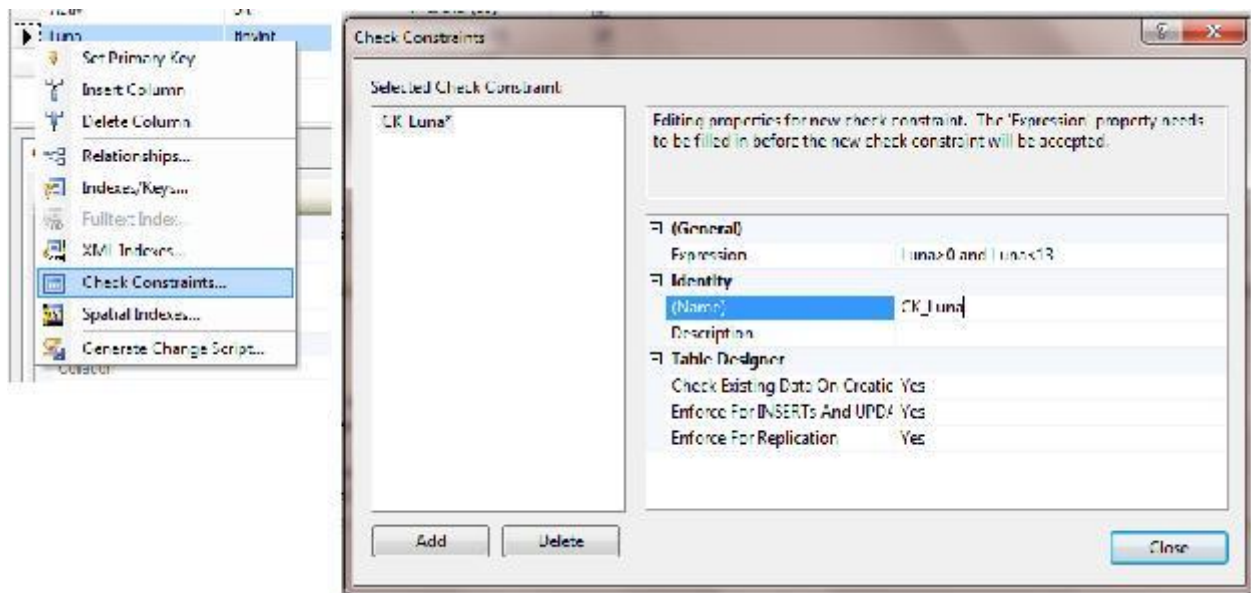


Fig. 10.

Pentru salvarea tabelii închidem fereastra tabelă de pe x-ul din dreapta sus (nu închidem tot SSMS-ul), dăm clic pe Yes, iar în fereastră scriem numele tabelii, apoi OK:



Fig. 11.

O să observăm, în partea stângă, o nouă intrare care e tabela nou creată. Obiectele sale sunt:

- **Coloane (Columns)** – dacă expandăm observăm denumirea coloanelor, tipul și dacă permite NULL sau nu. Deasemnea vedem câmpul cheie primară.
- **Chei (Keys)** – aici vedem cheile tabeli. În cazul de față e cheia primară
- **Constrângeri (Constraints)** – vedem constrângerea de tip CHECK stabilită asupra câmpului Luna și constrângerea de tip DEFAULT stabilită asupra câmpului Localitate
- **Declanșatori (Triggers)** – nedefinit nici un trigger
- **Indecși (Indexes)**- în momentul definirii cheii primare a fost automat definit un index de tip cluster. Orice alt index care va fi definti în această tabelă trebuie obligatoriu să fie de tip non-cluster.

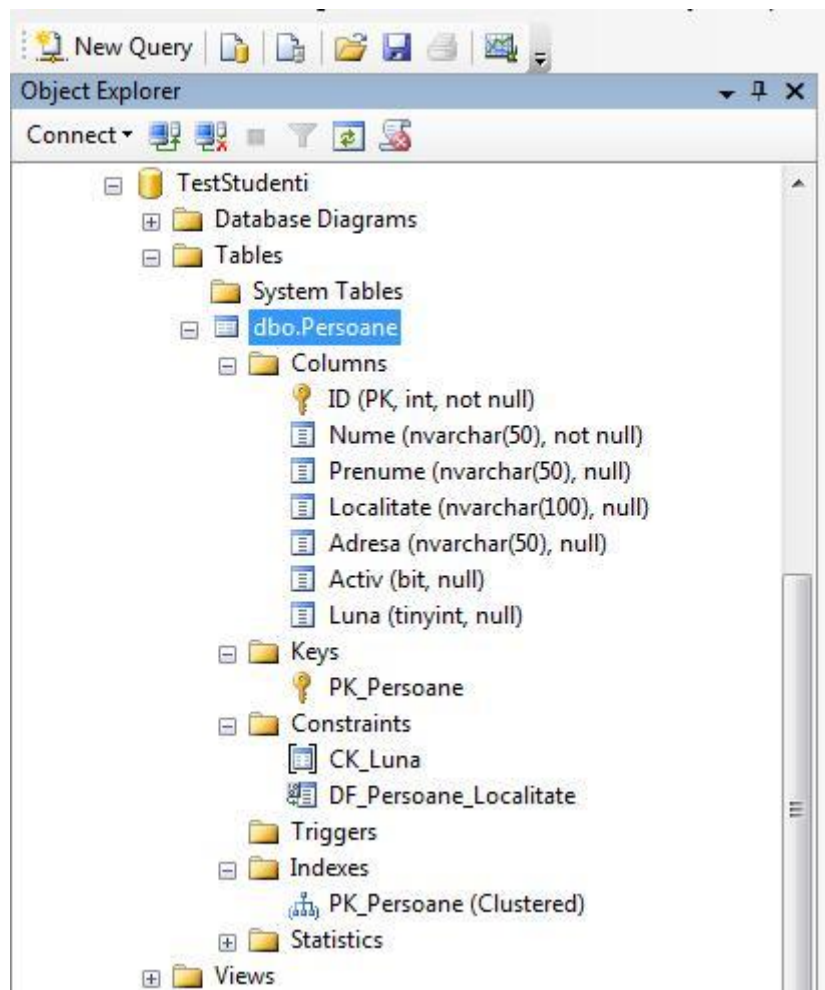


Fig. 12.

c. Modificarea structurii tabelor

Clic dreapta pe denumirea tabeli, apoi clic pe *Design*. Pentru a vedea modificările: clic dreapta pe numele tabeli, apoi clic pe *Refresh*.

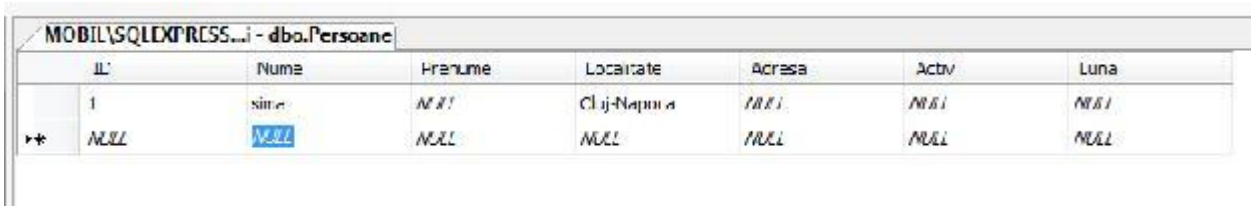
d. Ștergerea tabelelor

Clic dreapta pe denumirea tabelului, apoi clic pe *Delete*.

e. Introducerea datelor în tabelă

Clic dreapta pe denumirea tabelului, apoi clic pe *Edit Top 200 Rows*

De reținut. În bazele de date salvarea datelor introduse într-un rând se face automat atunci când se trece pe alt rând (cu Enter, cu tastele săgeți sau cu clic de mouse).



ID	Nume	Prenume	Localitate	Adresa	Activ	Luna
1	sime	NULL	Cluj-Napoca	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig. 13.

Vom **introduce un rând** ca în fig. 13. În câmpul ID scriem 1 (doar dacă nu e de tip Identity), apoi dăm ENTER (trecere la rândul următor). Dacă ID este de tip Identity introducem un prenume, apoi ENTER. Vom primi un mesaj în care ni se spune că Numele nu permite valori NULL. În consecință, vom introduce un nume, apoi ENTER. De această dată are loc salvarea datelor.

Observăm că, deși nu am introdus nimic în câmpul Localitate, a apărut *Cluj-Napoca*, ceea ce e firesc atâta timp cât am definit constrângerea DEFAULT.

Pe rândul 2 introducem un nume și o localitate. De această dată se reține în tabelă numele localității introduse, ceea ce este normal.

Modificare date

Revenim pe oricare rând și introducem în câmpul Luna valoarea 14. O să primim un mesaj în care ni se spune că avem o constrângere CHECK pe coloana Luna. Clic pe OK, apoi scriem o valoare validă (ex: 5). În cazul în care dorim să nu continuăm editarea, renunțăm cu tasta ESC.

Pentru a **șterge** unul sau mai multe rânduri se selectează cu mouse-ul pe antet rând (în stânga unde apare săgeata și steluța), clic dreapta, apoi clic pe *Delete*.

f. Vizualizarea datelor din tabelă

Clic dreapta pe denumirea tabelului, apoi clic pe *Select Top 1000 Rows*.

h. Crearea diagramelor

Legarea celor două tabele se poate face în mai multe modeuri. Aici vom construi o diagramă în care vom face legătura.

Pentru aceasta:

- clic dreapta pe *Database diagrams* din baza noastră de date
- Clic pe *New Database Diagrams*
- Clic pe OK
- Selectăm cele două tabele din fereastră, clic pe *Add*, apoi clic pe *Close*.
- Prindem cu mouse-ul ID-ul din Persoane și îl tragem peste id_persoana din telefoane
- Vor apărea 2 ferestre: verificăm să fie numele tabelor și a câmpurilor corecte apoi le închidem cu clic pe OK

Ar trebui să vedem legătura ca în fig. 15 (sau similar).

Dacă am creat o legătură greșită, pentru ștergerea sa (implicit și a constrângerii de cheie străină) dăm clic dreapta pe legătură apoi clic pe *Delete Relationships from Database*.

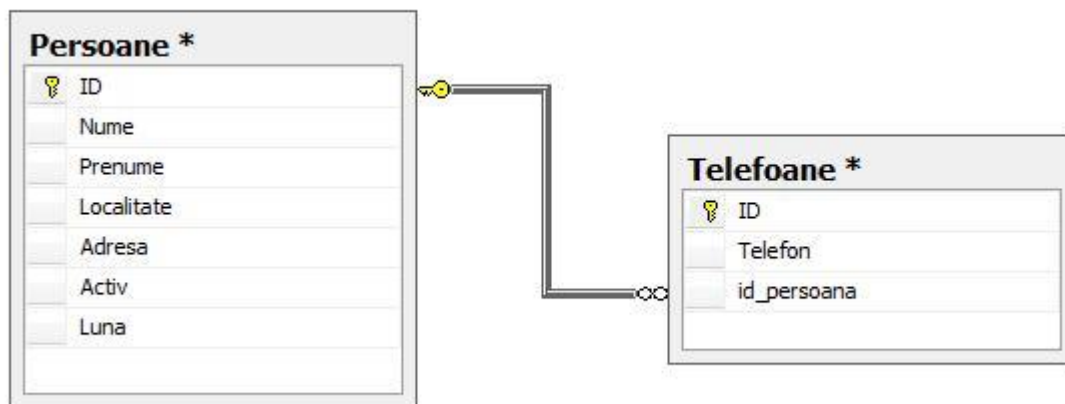


Fig. 15.

Închidem diagrama, îi dăm un nume și o salvăm.

Dacă dăm clic dreapta pe numele tabelii și Refresh o să vedem că avem o cheie suplimentară (care e și constrângerea de tip FOREIGN KEY).

Varianta 2 – Creare BD în linia comandă

a. Crearea bazei de date

O bază de date SQL Server este compusă din trei tipuri de fișiere: un fișier cu extensia **.mdf**,

Pentru crearea unei baze de date se folosește comanda CREATE DATABASE care în formă simplificată se prezintă conform următoarelor exemple:

1)

```
CREATE DATABASE dbStudenti
```

În acest exemplu baza de date dbStudenti va fi creată în folderul implicit “C:\Program Files\Microsoft SQL Server\MSSQL\Data”, precizat în timpul instalării sistemului SQL Server. Vor fi create fișierele

```
dbStudenti.mdf
```

```
dbStudenti_log.ldf
```

2)

```
CREATE DATABASE dbStudenti
```

```
ON
```

```
( NAME = dbStd,
```

```
  FILENAME = 'L:\DB\Studenti.mdf'
```

```
)
```

3)

```
CREATE DATABASE dbStudenti
```

```
ON
```

```
( NAME = dbStd,
```

```
  FILENAME = 'L:\DB\Studenti.mdf'
```

```
)
```

```
LOG ON
```

```
( NAME = dbStd_log,
```

```
  FILENAME = 'L:\DB\Studenti.ldf'
```

```
)
```

4)

```
CREATE DATABASE dbMultiFisier
```

```
ON PRIMARY
```

```
( NAME = F1,
```

```
  FILENAME = 'L:\db\Fisier1.mdf',
```

```
  SIZE = 3MB,
```

```
  MAXSIZE = 10MB,
```

```
  FILEGROWTH = 10%),
```

```
( NAME = F2,
```



```

        FILENAME = 'L:\db\Fisier2.ndf',
        SIZE = 1MB,
        MAXSIZE = 10MB,
        FILEGROWTH = 10%),

        ( NAME = F3,
        FILENAME = 'L:\db\Fisier3.ndf',
        SIZE = 1MB,
        MAXSIZE = 10MB,
        FILEGROWTH = 10%)
LOG ON

        ( NAME = F_Log1,
        FILENAME = 'L:\DB\Fisier_Log1.ldf',
        SIZE = 512KB,
        MAXSIZE = 10MB,
        FILEGROWTH = 10%),

        ( NAME = F_Log2,
        FILENAME = 'L:\db\Fisier_Log2.ldf',
        SIZE = 512KB,
        MAXSIZE = 10MB,
        FILEGROWTH = 10%)

```

In exemplele 2), 3) și 4) baza de date va fi creată în folderul DB al unității L (care poate fi, de exemplu, un stick usb).

Pentru exemplul 2) fișierul dbStudenti_log.ldf va fi creat tot în folderul DB al unității L.

- ON – semnifică utilizarea unui grup de fișiere
- NAME -furnizează numele logic al fișierului datelor, respectiv al jurnalului. Numele logice ale fișierelor sunt utilizate de funcțiile administrative ale SQL Server. Ele sunt unice la nivelul fișierelor.⁴
- FILENAME -furnizează numele fizic al fișierului datelor, respectiv al jurnalului

- **SIZE** –parametru opțional, specifică dimensiunea inițială a fișierului măsurată în KB, MB sau GB, valoare implicită 1MB pentru fișierul de date și 512KB pentru fișierul jurnal. Unitatea de măsură implicită este MB.
- **MAXSIZE** -parametru opțional, specifică dimensiunea maximă la care poate ajunge fișierul. Dacă se specifică **MAXSIZE** sau i se atribuie **UNLIMITED** atunci fișierul crește cât îi permite spațiul liber de pe disc
- **FILEGROWTH** – parametru opțional, precizează pasul cu care crește dimensiunea fișierului, în valoare absolută sau în procente raportat la fișierul asociat. Valoarea implicită este de 256KB cu valoarea minimă 64KB. Valoarea 0 împiedică creșterea fișierului.

Modificarea bazelor de date

Comanda **ALTER DATABASE** permite modificarea unei baze de date prin adăugarea sau eliminarea fișierelor asociate precum și modificarea atributelor fișierelor asociate.

Clauzele uzuale ale comenzii **ALTER DATABASE** sunt

```
ALTER DATABASE database
{ ADD FILE < filespec > [ ,...n ]
  --Specifică faptul că un fișier este adăugat.
| ADD LOG FILE < filespec > [ ,...n ]
  -- Specifică faptul că se adaugă la baza de date un fișier de log
| REMOVE FILE logical_file_name
  -- se folosește pentru eliminarea fișierelor asociate.
  -- Nu pot fi eliminate fișierele nevide.
| MODIFY FILE < filespec >
  -- se folosește pentru modificarea atributelor fișierelor asociate.
| MODIFY NAME = new_dbname
  -- se folosește pentru modificarea denumirii bazei de date.
}
```

Clauza **REMOVE FILE** se folosește pentru eliminarea fișierelor asociate. Nu pot fi eliminate fișierele nevide.

```
< filespec > ::=
( NAME = logical_file_name
  [ , NEWNAME = new_logical_name ]
```

```
[ , SIZE = size ]
[ , MAXSIZE = { max_size | UNLIMITED } ]
[ , FILEGROWTH = growth_increment ]
)
```

Pentru a modifica numele logic al unui fișier de date sau un fișier jurnal, se specifică în NAME numele fișierului logic care urmează a fi redenumit, și se precizează în NEWNAME noul nume logic pentru fișier.

```
Use master
Go
Alter database dbTest
Add file
(name=fd3,
Filename='d:\db\fișier3.ndf',
Size=5mb,
Maxsize=unlimited,
Filegrowth=5mb
)
Go
Alter database dbtest
Remove file fd3
```

Ștergerea bazelor de date

Sintaxa:

```
DROP DATABASE denumire_bază_date
```

b. Crearea tabelelor

Comanda CREATE TABLE permite crearea unui nou tabel. Sintaxa sa este următoarea:

```
CREATE TABLE denumire_tabel
( { < definiție_coloană >
  | denumire_coloană AS expresie_coloană_calculată
  | < constrângere_tabel >
} [ ,... ]
```

)

< definiție_coloană > ::= *denumire_coloană tip_dată*

[[DEFAULT *expresie_constantă*] | [IDENTITY [(*valoare_inițială, pas*)]]]
[< constrângere_coloană >] [...]

< constrângere_coloană > ::= [CONSTRAINT *denumire_constrângere*]

{ [NULL | NOT NULL]
| [{ PRIMARY KEY | UNIQUE } [CLUSTERED | NONCLUSTERED]]
]
| [[FOREIGN KEY]
REFERENCES *tabel_referit* [(*coloana_referită*)]
[ON DELETE { CASCADE | NO ACTION }]
[ON UPDATE { CASCADE | NO ACTION }]
]
| CHECK (*expresie_logică*)
}

< constrângere_tabel > ::= [CONSTRAINT *denumire_constrângere*]

{ [{ PRIMARY KEY | UNIQUE }
[CLUSTERED | NONCLUSTERED]
{ (*coloană* [ASC | DESC] [, ...]) }
]
| FOREIGN KEY
[(*coloană* [, ...])]
REFERENCES *tabel_referit* [(*coloană_referită* [, ...])]
[ON DELETE { CASCADE | NO ACTION }]
[ON UPDATE { CASCADE | NO ACTION }]
[NOT FOR REPLICATION]
| CHECK (*expresie_logică*)
}

Înainte de crearea unui nou tabel, trebuie să activăm baza de date în care tabelul va fi creat:

USE *denumire_bază_de_date*

Într-o comandă CREATE TABLE cuvântul cheie AS permite crearea unei coloane calculate pe baza valorilor din celelalte coloane.

Clauza CONSTRAINT permite implementarea constrângerilor de integritate. O constrângere este un mecanism care ne asigură că valorile unei coloane sau ale unei mulțimi de coloane satisfac o condiție dată. Dacă nu se specifică un nume explicit pentru constrângere atunci sistemul îi atribuie unul nume. Constrângerile se pot defini la nivel de coloană sau la nivel de tabel, după cum ele se referă la datele unei singure coloane sau la datele mai multor coloane. Există mai multe categorii de integritate a datelor:

- Integritatea domeniului: definește un interval sau o listă de valori posibile pentru o coloană;
- Integritatea entității: în cadrul unei entități nu pot exista două realizări (instanțe) identice;
- Integritatea referențială: valorile unei chei străine pot fi ori *null*, ori printre valorile cheii primare referite.

Constrângeri de domeniu

- *Constrângerea NULL* Specifică faptul că sunt permise valori Null pentru coloana respectivă. Constrângerea NULL este implicită.

Exemplu:

```
Telefon char(10) NULL
```

- *Constrângerea NOT NULL*. Specifică faptul că sunt interzise valorile Null pentru coloana respectivă

Exemplu:

```
Nume char(30) NOT NULL
```

- *Constrângerea DEFAULT*. Specifică o valoare implicită pentru coloană. Această valoare este utilizată de fiecare dată când se adaugă o nouă înregistrare și nu se specifică nicio valoare pentru coloana în cauză.

Exemple:

```
Data SmallDateTime default getdate(),  
CodJudet char(2) default 'AG',  
Cantitate numeric(8,3) default 0
```

- *Constrângerea CHECK*. Defițește o restricție de domeniu pe care trebuie să o satisfacă datele din coloana respectivă. Expresia logică atașată unei constrângeri *Check* la nivel de tabel poate face referire la orice coloană a tabelului curent.

Sintaxa:

```
CHECK (expresie_logică)
```

Exemple:

```
Salariu int CHECK (Salariu >= 600 AND Salariu <= 8000)
Nota INT CHECK (Nota BETWEEN 1 AND 10)
DenumireZi char(10) check(DenumireZi in ('luni', 'marti', 'miercuri'))
Constraint ck_pret check(pretVanzare>=pretAprovizionare)
```

Observatie: Coloanele supuse unor reguli *check* pot primi valoarea *null* dacă nu se impune constrângerea *not null*.

Constrângeri pentru integritatea entităţii

- **Constrângerea PRIMARY KEY.** Impune valori unice şi nenule pentru coloana (grupul de coloane) în cauză, coloana (grupul de coloane) va reprezenta cheia primară a tabelului.

Exemplu:

```
CodFurnizor char(10) primary key
```

- **Constrângerea UNIQUE.** Impune valori unice pentru coloana (grupul de coloane) în cauză, coloana (grupul de coloane) reprezintă o cheie candidat a tabelului.

Exemple:

```
marca int unique
CNP char(13) constraint ix_cnp unique
```

- Constrângerile PRIMARY KEY şi UNIQUE generează implicit chei de indexare. Un index poate fi de tip CLUSTERED, caz în care ordinea fizică a rândurilor tabelului coincide cu ordinea logică(tabelul este sortat după cheia indexului clustered), sau NONCLUSTERED. Evident că o singură cheie poate fi de tip clustered. Opţiunea implicită pentru PRIMARY KEY este CLUSTERED, iar pentru UNIQUE este NONCLUSTERED, dar pot fi schimbate ca în exemplul următor:

```
Create table tCandidati
( CodCandidat int primary key nonclustered,
  NumeCandidat varchar(30) not null,
  CNP char(13)constraint ix_cnp unique clustered
)
```

- **Constrângerea IDENTITY.** Indică o coloană asociată unui tip de dată întreg, pentru care SQL Server generează în mod automat valori incrementale, unice la nivel de tabel.

Exemplu

```
id_detaliu int identity(101,1)
id_bigint identity
```

Primul parametru reprezintă valoarea atribuită primei tuple, iar al doilea parametru reprezintă pasul de incrementare. Parametrii pot lipsi, ei au valoarea implicită 1.

Constrângerea integrității referențiale

Constrângerea Foreign key se folosește, uzual împreună cu Primary key, pentru a rezolva problema integrității referențiale.

Exemplu:

```
Create table tRezultateExamene
( Id_Nota int identity primary key,
  CodStudent char(10) not null
    FOREIGN KEY REFERENCES tStudenti(CodStudent)
    ON DELETE CASCADE ON UPDATE CASCADE ,
  CodCurs char(13) not null
    FOREIGN KEY REFERENCES tCursuri(CodCurs)
    ON DELETE CASCADE ON UPDATE CASCADE,
  DataExaminare datetime,
  Nota tinyint chech (Nota between(1 and 10))
)
```

Coloana_referită (grupul de coloane) trebuie să fie definită ca *Primary key*, *Unique* sau *index unic* în tabelul referit.

- **ON DELETE CASCADE** – menține integritatea referențială în cazul ștergerii unui rând din tabel_referit (care conține cheia primară sau unică), prin ștergerea tuturor rândurilor ce conțin cheii străine dependente. Valoarea implicită este ON DELETE NO ACTION.
- **ON UPDATE CASCADE** -menține integritatea referențială în cazul modificării valorii coloanei referite din tabelul asociat (care conține cheia primară sau unică), prin propagarea modificării tuturor rândurilor ce conțin cheia străină dependentă. Valoarea implicită este ON UPDATE NO ACTION.

c. Modificarea structurii tabelelor

Comanda ALTER TABLE permite modificarea structurii unui tabel. Se pot efectua următoarele tipuri de modificări:

- *Modificare tipului de dată al unei coloane:*

```
ALTER TABLE denumire_tabel  
    ALTER COLUMN denumire_coloană tip_nou_de_dată [ ( precizia [ , scala ] ) ]  
    [ NULL | NOT NULL ]
```

Exemplu:

```
Alter table tCandidati  
    alter column LocalitateDomiciliu varchar(30) not null
```

- *Adăugarea unor noi coloane (împreună cu eventuale constrângeri pentru aceste coloane):*

```
ALTER TABLE denumire_tabel  
ADD < definiție_coloană > [...]  
< definiție_coloană > are aceeași semnificație ca la CREATE TABLE.
```

Exemplu

```
Alter table tCandidati  
    add codJudetDomiciliu char(2) default 'AG'
```

- *Adăugarea unor coloane calculate:*

```
ALTER TABLE denumire_tabel  
ADD denumire_coloană AS expresie_coloană_calculată [...]
```

Exemplu:

```
Alter table tCandidati  
    ADD media as (nota1+nota2)/2
```

- *Ștergerea unor coloane:*

```
ALTER TABLE denumire_tabel  
    DROP COLUMN denumire_coloană [ ,... ]
```

Exemplu:

```
Alter table tCandidati  
    Drop column media
```


- *Adăugarea unor noi constrângeri :*

```
ALTER TABLE denumire_tabel
```

```
[WITH CHECK | WITH NOCHECK ] ADD < constrângere_tabel > [ ,... ]
```

WITH CHECK | WITH NOCHECK specifică dacă datele existente sunt sau nu validate în raport cu constrângerea nou adăugată. Opțiunea implicită este WITH CHECK.

< constrângere_tabel > are aceeași semnificație ca la CREATE TABLE

Exemplu:

```
Alter table tSalarii
```

```
with nocheck add constraint ck_cifra_unit_este_zero check(salariu%10=0)
```

- *Adăugarea unei valori implicite pentru o coloană:*

```
ALTER TABLE denumire_tabel
```

```
ADD [constraint denumire_constrângere] DEFAULT expresie_constantă FOR coloană
```

Exemplu:

```
Alter table tChitante
```

```
add constraint df_dataChitanta default getdate() for dataChitanta
```

```
Alter table tChitante
```

```
add default 0 for suma
```

- *Ștergerea unor constrângeri :*

```
ALTER TABLE denumire_tabel
```

```
DROP CONSTRAINT denumire_constrângere [ ,... ]
```

Exemplu:

```
Alter table tSalarii
```

```
drop constraint constraint ck_cifra_unit_este_zero
```

- *Activarea(CHECK) sau dezactivarea(NOCHECK) constrângerilor de tip CHECK și FOREIGN KEY:*

```
ALTER TABLE denumire_tabel
```

```
{CHECK | NOCHECK } CONSTRAINT { ALL | denumire_constrângere [ ,... ] }
```

ALL specifică faptul că toate constrângerile sunt activate sau dezactivate.

denumire_constrângere denumirile constrângerilor care sunt activate sau dezactivate

d. Ștergerea tabelelor

Comanda DROP TABLE șterge din baza de date definiția unui tabel împreună cu indecșii, triggerii și constrângerile asociate. Sintaxa:

DROP TABLE denumire_tabel

Un tabel poate fi șters dacă nu este referit printr-o constrângere de tip FOREIGN KEY.

e. Introducerea datelor în tabelă

INSERT INTO tabel (camp1, camp2, camp3) VALUES (valoarea1, valoarea2, valoarea3);	# introduce în tabelul cu numele 'tabel', în 'campul1' 'valoarea1', în 'campul2' 'valoarea2' și în 'campul3' 'valoarea3'. Iata cum ar arăta în format tabelar:						
	<table><tr><td>campul1</td><td>campul2</td><td>campul3</td></tr><tr><td>valoarea1</td><td>valoarea2</td><td>valoarea3</td></tr></table>	campul1	campul2	campul3	valoarea1	valoarea2	valoarea3
campul1	campul2	campul3					
valoarea1	valoarea2	valoarea3					
INSERT INTO tabel (camp1, camp2) VALUES (valoarea1, valoarea2);	# Se poate omite una din coloane, dacă avem 5 coloane, dar vrem să introducem numai în 3, specificăm câmpul și valoarea doar pentru cele pe care le vrem, restul le ignorăm.						
	<table><tr><td>campul1</td><td>campul2</td><td>campul3</td></tr><tr><td>valoarea1</td><td>valoarea2</td><td></td></tr></table>	campul1	campul2	campul3	valoarea1	valoarea2	
campul1	campul2	campul3					
valoarea1	valoarea2						
INSERT INTO tabel VALUES (valoarea1, valoarea2, valoarea3);	# o variantă simplificată care se poate aplica doar când introducem valori în toate câmpurile tabelului (nu se poate omite)						
INSERT INTO tabel VALUES (valoarea1, valoarea2, "");	# identică ca cea dinainte, doar că în lipsa unei valori se pun ghilimele.						

f. Modificarea datelor în tabelă

UPDATE tabel SET coloana1='noua valoare a coloanei 1', coloana2='noua valoare a coloanei 2' WHERE conditii;	# pentru actualizarea conținutului unei înregistrări din tabel. Sintaxa este la fel ca la comanda SELECT. (se șterge valoarea veche și se scrie cea nouă)
UPDATE tabel SET coloana1='noua	# pentru actualizarea conținutului datelor din coloana1 și

valoare a coloanei 1', coloana2='noua	coloana 2 din întregul tabel. Sintaxa este la fel ca la
valoare a coloanei 2' ;	comanda SELECT. (se șterge valoarea veche și se scrie cea nouă)

g. Eliminarea datelor din tabelă

DELETE FROM tabel WHERE conditii;	# șterge înregistrarea din tabel.
DELETE FROM tabel	# șterge toate înregistrările din tabel.

Sarcini de realizat:

Exercițiul 1. Să se creeze tabela firme și tabela agenți în care să fie precizate restricțiile de integritate. firme

Coloana	Tipul de date	Constrângerea
codfirma	NUMBER(2)	CONSTRAINT PKey_firme PRIMARY KEY
denfirma	VARCHAR(20)	NOT NULL
loc	VARCHAR(20)	
zona	VARCHAR(15)	CONSTRAINT FZONA_CK check (zona IN ('Moldova','Ardeal','Banat','Muntenia','Dobrogea','Transilvania'))

CREATE TABLE agenti

Coloana	Tipul de date	Constrângerea
codagent	VARCHAR(3)	CONSTRAINT pk_agent PRIMARY KEY
numeagent	VARCHAR(25)	NOT NULL
dataang	DATE	DEFAULT SYSDATE
datanast	DATE	
functia	VARCHAR(20)	
codfirma	NUMBER(2)	NSTRAINT FK_agenti FOREIGN KEY (codfirma) REFERENCES firme(codfirma)

Exercițiul 2. Se consideră activitatea de evidență a comenzilor încheiate de o societate comercială cu diverși clienți prin intermediul angajaților care dețin anumite funcții și își desfășoară activitatea în diverse departamente, aflate în mai multe țări/regiuni/locații. Comenzile pot fi plasate și online și conțin produse aflate în depozitul societății.

Să se creeze această bază de date.

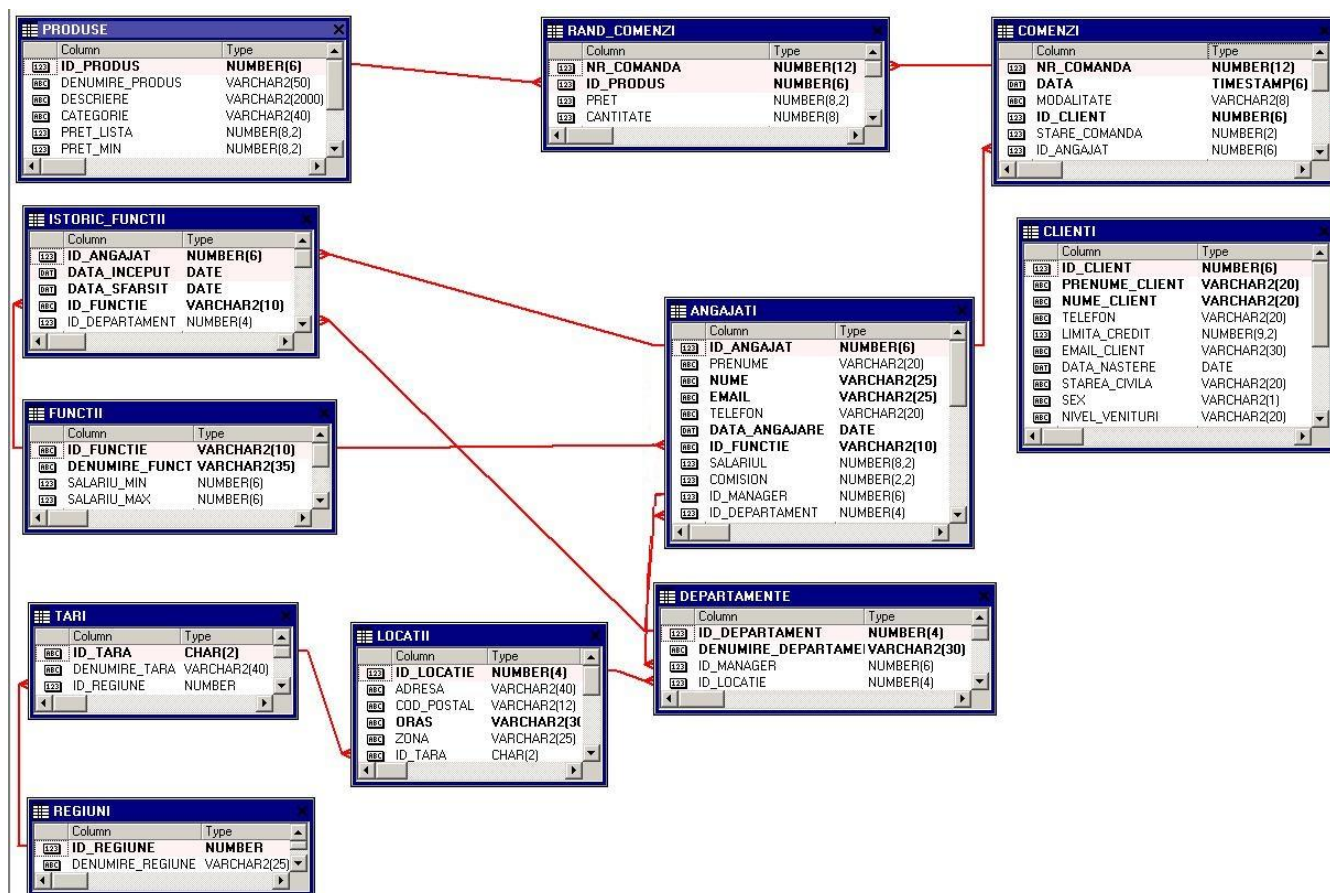


Figura 1. Schema BD pentru activitatea de evidenta a comenzilor si a resurselor umane

Exercițiul 3. Să se creeze tabela `fosti_agenti` pe baza tabelului `agenti` și care va conține doar o parte din coloanele tabelului inițial (`codagent`, `numeagent`, `functia`, `codfirma`).
Se va folosi structura:

Exercițiul 4.

- 4.1. Să se redenumescă tabela `agenti` cu `personal`.
- 4.2. Să se adauge coloanele `email` și `vârsta` în tabela `personal`.
- 4.3. Să se modifice tipul de date al coloanei `email`.
- 4.4. Să se ștergă coloana `email`.
- 4.5. Să se inactivezeze coloana `functia`.
- 4.6. Să se ștergă coloanele inactive.
- 4.7. Să se adauge o restricție pe coloana `vârsta`.
- 4.8. Să se dezactiveze restricția anterioară.
- 4.9. Să se ștergă restricția anterioară.

Exercițiul 5.

- 5.1. Să se ștergă tabela `fosti_agenti`.

Exercițiul 6.

- 6.1. Să se ștergă înregistrările tabelului `personal`.

Comanda `TRUNCATE TABLE` șterge înregistrările unei tabeli și eliberează spațiul alocat acestora

Exercițiul 7.

7.1. Să creeze tabela salariati pe baza tabelii angajati fără a prelua și înregistrările (doar structura) și să se adauge un nou angajat.

	❖ COLUMN_NAME	❖ DATA_TYPE	❖ NULLABLE	DATA_DEFAULT	❖ COLUMN_ID	❖ COMMENTS
1	ID_ANGAJAT	NUMBER(6,0)	No	(null)	1	(null)
2	PRENUME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)
3	NUME	VARCHAR2(25 BYTE)	Yes	(null)	3	(null)
4	EMAIL	VARCHAR2(25 BYTE)	Yes	(null)	4	(null)
5	TELEFON	VARCHAR2(20 BYTE)	Yes	(null)	5	(null)
6	DATA_ANGAJARE	DATE	Yes	(null)	6	(null)
7	ID_FUNCTIE	VARCHAR2(10 BYTE)	Yes	(null)	7	(null)
8	SALARIUL	NUMBER(8,2)	Yes	(null)	8	(null)
9	COMISION	NUMBER(2,2)	Yes	(null)	9	(null)
10	ID_MANAGER	NUMBER(6,0)	Yes	(null)	10	(null)
11	ID_DEPARTAMENT	NUMBER(4,0)	Yes	(null)	11	(null)

7.2. Să se adauge în tabela salariati toți angajații din tabela angajați care lucrează în departamentele 20, 30 și 50. Să se finalizeze tranzacția (salvarea modificării).

7.3. Să se adauge în tabela salariati un angajat ale cărui date sunt introduse de utilizator de la tastatură.

Exercițiul 8.

8.1. Să se majoreze cu 100 salariul angajaților din tabela salariati care au salariul mai mic decât 3000.

8.2. Să se actualizeze salariul angajaților al căror manager are id = 122 cu salariul angajatului cu id = 125.

8.3. Să se actualizeze salariul și comisionul angajaților din tabela salariati cu salariul și comisionul angajatului cu id_angajat = 167 din tabela angajati, doar pentru angajații care au salariul mai mic decât salariul angajatului cu id = 173 din tabela angajati și care lucrează în departamentul 50.

Exercițiul 9.

9.1. Să se șteargă angajații din tabela salariati care au id_manager egal cu 122 sau 123.

9.2. Să se șteargă angajații din tabela salariati angajați înainte de anul 2016.

9.3. Să se șteargă totii angajații din tabela salariati. Să se anuleze tranzacția.

Bibliografie

1. [http://msdn.microsoft.com/en-us/library/aa258256\(SQL.80\).aspx](http://msdn.microsoft.com/en-us/library/aa258256(SQL.80).aspx)
2. <http://www.dotnet-france.com>
3. Sima I. Crearea structurii bazei de date. Disponibil online la adresa:
<https://simaioan.files.wordpress.com/2016/10/tu1-2-create-bd-din-management-studio.pdf>