T_SQL. Utilizarea comenzilor Create DataBase, Alter Database, Drop Database, Create Table, Alter Table, Drop Table

1. Obiectivul lucrării:

   Formarea și dezvoltarea abilităților de creare a bazelor de date.

2. Breviar teoretic cu exerciții și probleme rezolvate

CREAREA BAZELOR DE DATE SQL SERVER

Microsoft SQL Server este un sistem de gestine a bazelor de date relaționale.

Microsoft SQL Sever foloseşte o variantă de SQL numită T-SQL, sau Transact-SQL, o implementare a standardului SQL-92 (fundament al multor SGBD-uri comerciale), cu extensii pentru proceduri stocate, tranzactii etc.

O bază de date Microsoft SQL Server este compusă din trei tipuri de fişiere: un fişier cu extensia **.mdf,** zero sau mai multe fişiere cu extensia **.ndf** şi unul sau mai multe fişiere cu extensia **.ldf**

În fişierul cu extensia *.mdf* sunt stocate obiectele bazei de date precum tabele, indecși, vederi, proceduri stocate etc. împreună cu definiţiile lor. Fişierele cu extensia *.ndf* sunt fişiere secundare ce conţin numai date, iar fişierul cu extensia *.ldf* conţine jurnalul de tranzacţii. Orice bază de date are asociat un jurnal de tranzacţii. Actualizarea unei înregistrări a bazei de date presupune memorarea în jurnalul de tranzacţii a conţinutului înregistrării dinainte şi după actualizare. Jurnalul de tranzacţii este folosit pentru restaurarea bazei de date în situaţia în care apare o eroare ce necesită anularea sau reluarea unor operaţii înregistrate.

Incepând cu versiunea SQL 7.0, a fost introdus conceptul de **nume logic**. Există câte un nume logic pentru fiecare fisier *.mdf, .ndf, .ldf*. Implicit, pentru fisierul *.mdf*, numele logic coincide cu numele fișierului fizic(fără extensie) iar pentru fisierul .ldf numele logic coincide cu numele fișierului fizic(fără extensie) completat cu *_log*.

Numele logice sunt utilizate de funcțiile de administrare ale sistemului SQL Server pentru identificarea fișierelor și nu trebuie neapărat schimbate când se schimbă denumirea bazei de date. Deși există rareori nevoia de a schimba numele de fișier logic, începând cu SQL Server 2000 este introdusă această funcționalitate utilizând comanda *Alter database.*

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

```
1)
CREATE DATABASE dbStudenti
2)
CREATE DATABASE dbStudenti
ON
( NAME = Std,
  FILENAME = 'L:\DB\Studenti.mdf'
)

3)
CREATE DATABASE dbStudenti
ON
( NAME = Std,
  FILENAME = 'L:\DB\Studenti.mdf'
```

```
)
LOG ON
( NAME = Std_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 exemplul 1) 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 fişierele

`dbStudenti.mdf`
`dbStudenti_log.ldf`

In exemplele 2), 3) şi 4) baza de date va fi creată în folderul DB al unităţii L (care poate fi, de exemplu, un hard disk extern).

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

ON – semnifică utilizarea unui grup de fişiere

NAME -furnizează numele logic al fişierului datelor, respectiv al jurnalului. Numele logice ale fișierelor sunt utilizate de functiile administrative ale SQL Server. Ele sunt unice la nivelul fişierelor.

FILENAME -furnizează numele fizic al fişierului datelor, respectiv al jurnalului

SIZE –parametru opţional, specifică dimensiunea iniţială a fişierului măsurată în KB, MB sau GB, valoare implicită 1MB pentru fişierul de date şi 512KB pentru fişierul jurnal. Unitatea de măsură implicită este MB.

MAXSIZE -parametru opţional, specifică dimensiunea maximă la care poate ajunge fişierul. Dacă se specifică MAXSIZE sau i se atribuie UNLIMITED atunci fişierul creşte cât îi permite spaţiul liber de pe disc

FILEGROWTH – parametru opţional, precizează pasul cu care creşte dimensiunea fişierului, în valoare absolută sau în procente raportat la fişierul asociat. Valoarea implicită este de 256KB cu valoarea minimă 64KB. Valoarea 0 impiedică creşterea fişierului.

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.
}
```

```
< filespec > ::=
  (  NAME = logical_file_name
    [ , NEWNAME = new_logical_name ]
    [ , SIZE = size ]
    [ , MAXSIZE = { max_size | UNLIMITED } ]
    [ , FILEGROWTH = growth_increment ]
  )
```

```
Use master
Go
Alter database dbTest
 Add file
  (name=fd4,
   Filename='d:\db\fisier4.ndf',
   Size=5mb,
   Maxsize=unlimited,
   Filegrowth=5mb
  )
Go
Alter database dbtest
  Remove file fd4
```

Pentru a modifica numele logic al unui fişier de date sau al uni fişier jurnal, în comanda `Alter database` se specifică în NAME numele fişierului logic care urmează a fi redenumit şi în NEWNAME noul nume logic pentru fişier.

ŞTERGEREA BAZELOR DE DATE

Sintaxa:

```
DROP DATABASE denumire_bază_date
```

SALVAREA BAZELOR DE DATE

```
BACKUP DATABASE dbStudenti
       TO DISK='g:\dbSalvari\dbStudenti.bak'
```

Incepând cu SQL Server 2008 a fost introdus backup-ul compresat. Acesta se realizează cu comanda BACKUP la care se adaugă clauza WITH COMPRESSION.

Exemplu:

```
BACKUP DATABASE dbStudenti
       TO DISK='g:\dbSalvari\dbStudenti.bak' WITH COMPRESSION
```

RESTAURAREA BAZELOR DE DATE

```
RESTORE DATABASE dbStudenti
        FROM DISK='g:\salvari\dbStudeti.bak'
```

INSTALAREA BAZELOR DE DATE

Procedura de sistem **sp_attach_db** este folosită pentru a face o bază de date portabilă, accesibilă de pe un server nou. Astfel, dacă o bază de date nevidă este stocată pe un suport amovibil(nu neapărat), ea poate fi instalată printr-un apel de forma

```
execute sp_attach_db dbStd, 'L:\DB\Studenti.mdf'
Numele bazei de date nu este necesar să coincidă cu cel
iniţial
```

Instalarea unei baze de date se poate face şi cu ajutorul comenzii *create database* cu folosirea clauzei *for attach* ca mai jos:

```
create database dbStd
on
 (filename='L:\db\Studenti.mdf')
for attach
```

DEZINSTALAREA BAZELOR DE DATE

Procedura de sistem **sp_detach_db** este folosită pentru dezinstalarea bazelor de date prin apeluri de forma:

```
Execute sp_detach_db dbStd
```

Observaţie

SQL Server Management Studio oferă facilităţi grafice pentru instalarea şi dezinstalarea bazelor de date

TIPURI DE DATE MICROSOFT SQL SERVER

Microsoft SQL Server conţine următoarele tipuri de date predefinite, împărţite în grupe:

Tipuri de date pentru numere întregi

bit	pentru numere întregi care pot lua una din valorile: 0 , 1 sau NULL.
tinyint	pentru numere întregi fără semn reprezentate pe 1 octet cu valori admisibile de la 0 la 255 sau NULL.
smallint	pentru numere întregi cu semn pe 2 octeţi, valori admisibile de la -2^{15} (-32768) până la 2^{15} - 1 (32767) sau NULL.
int sau ***integer***	pentru numere întregi cu semn pe 4 octeţi, valori admisibile de la -2^{31} (-2147483648) până 2^{31} - 1 (2147483647) sau NULL.
Bigint	pentru numere întregi cu semn pe 8 octeţi, valori admisibile de la -2^{63} până 2^{63} – 1 sau NULL.

Observţii:

- SQL Server întoarce un mesaj de eroare dacă se încearcă inserarea unei valori care nu se încadrează în domeniul de valori corespunzător tipului de date.

-Tipul bit are o reprezentare optimizată, astfel dacă într-un tabel există mai multe atribute de tip bit, acestea vor fi împachetate la nivel de octet: 1 octet – pană la 8 atribute, 2 octeţi 9-16 atribute etc.

Tipuri de date pentru numere zecimale cu virgula fixă

Decimal[(p[,s])]	pentru numere zecimale cu virgula fixă, valori posibile între -10^{38} -1 şi 10^{38} -1 sau NULL.
Numeric[(p[,s])]	echivalent cu tipul *Decimal* sau NULL.

p (precizia) – numărul total de cifre care pot fi stocate, inclusiv partea întreagă şi 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 octeţi alocaţi tipului *decimal/numeric* depinde de precizie după cum urmează:

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

Tipuri de date pentru unităţi monetare

money	pentru numere zecimale cu 4 cifre după virgulă reprezentate pe 8 octeţi, cu valori posibile în intervalul -922,337,203,685,477.5808 la +922,337,203,685,477.5807 sau NULL.

smallmoney	numere zecimal cu 4 cifre după virgulă reprezentate pe 4 octeţi, ia valori în intervalul [-214,748.3648 , +214,748.3647] sau NULL.

Obs. Toate tipurile numerice de date cu excepţia *money* şi *smallmoney* sunt convertite implicit din şiruri de caractere în timpul executării instrucţiunilor Insert şi Update.

De exemplu

```
insert into tabelTest(c1) values('45')
```

va genera un mesaj de eroare dacă c1 este de tip *money* şi va funcţiona corect pentru orice alt tip numeric

Pentru rezolvarea problemei, putem face o transformare explicită astfel:

```
insert into tabelTest(c1) values(cast('45' as money))
```

Tipuri de date pentru numere zecimale cu virgula mobilă

real – pentru numere reprezentate în virgula flotantă, din intervalul de la 3.4^{-38} până la 3.4^{+38} cu o precizie de 7 cifre. Este reprezentat pe 4 octeţi.

float[(n)] – Dacă se specifică o valoare între 1 şi 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 şi 15, numerele stocate se pot afla în intervalul de la - 2.23^{-308} până 1a 1.79^{+308}(pozitive şi negative).

Observaţie: Valorile în virgulă mobilă sunt supuse erorilor de rotunjire. Ele asigură acurateţe 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.

De exemplu

```
 insert into tabelTest (c1) values(123456789123)
```

va stoca o valoare aproximativă daca c1 este de tip `real`, după cum se observa şi din rezultatul furnizat de următoarea frază select:

```
select c, cast(c as bigint) from t1
```

```
 1.234568E+11  123456790528
```

Tipuri de date pentru date calendaristice şi timp

datetime – Este reprezentat pe 8 octeţi şi păstrează data şi ora. Data poate fi o valoarea din intervalul de la 1 ianuarie anul 1753 până la 31 decembrie anul 9999. Timpul se defineşte cu exactitate de sutimi de secunde(1/300 dintr-o secundă).

smalldatetime – Este reprezentat pe 4 octeţi şi păstrează data şi 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 acurateţe de 1 minut.

Tipuri de date pentru şiruri de caractere

char[(n)] – pentru şiruri de caractere ASCII de lungime fixată de n caractere, dacă n lipseşte lungimea este de 1 caracter. Parametrul n poate lua valori intre 1 şi 8000. Şirul de caractere se va completa cu caractere spaţiu dacă mărimea curentă a şirului este mai mică decât n.

varchar(n) - şiruri de caractere ASCII de lungime variabilă (maximum 8000 caractere), se foloseşte când datele au lungimi ce variază in plaje largi. Spaţiul de stocare folosit se adaptează la numărul curent de caractere al şirului.

text - şiruri de caractere ASCII de lungime variabilă(lungimea maximală 2^{31}-1 =2,147,483,647 caractere). Şirul de caractere este memorat în pagini de 8ko fiecare

nchar[(n)] - şiruri de caractere UNICODE de lungime fixată (maxium 4000 caractere)

nvarchar[(n)] - şiruri de caractere UNICODE de lungime variabilă (maxium 4000 caractere)

ntext - şiruri de caractere UNICODE de lungime variabilă (lungimea maximală 2^{30}–1=1,073,741,823 caractere)

Tipuri de date pentru şiruri binare

binary [(n)] – şir binar de lungime fixată (maximum 8000 octeţi). Se foloseşte pentru stocarea unor secvenţe de biţi. Valorile de tip binar sunt reprezentate în sistem hexazecimal şi se introduc uzual tot în hexazecimal(precedate de 0x).

varbinary [(n)] – şir binar de lungime variabilă (maximum 8000 octeţi).

image – şir binar de lungime fixată (maximum 2^{31}-1 = 2,147,483,647 octeţi).

timestamp –O valoare de tip *timestamp*(marcă temporală) 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 ficare modificare a tuplei. Ea ne arată ordinea operaţiilor efectuate de SQL Server. Marcile de timp(*timestamp*) se pot folosi pentru a impiedica doi utilizatori să modifice aceeaşi tuplă. Tipul *timestamp* nu reprezintă data şi ora. 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 octeţi şi asigură unicitatea valorilor la nivelul bazei de date. Generarea în Transact SQL a unui nou *uniqueidentifier* se face cu **NEWID()**

Loturi și scripturi

Un *lot* este o succesiune de comenzi care urmează să fie executate ca un întreg, orice eroare de sintaxă din cadrul lotului conducând la neexecutarea întregului lot. Un lot se încheie cu comanda GO(scrisă singură pe rând).
Loturile au unele limitări ca de exemplu:
comenzile *create view, create procedure, create trigger* trebuie să fie scrise în loturi distincte ca prime comenzi, spre deosebire de comenda c*reate table* care pote fi folosită în același lot de mai multe ori.

Un *script* este un set de loturi, care pot fi salvate într-un fișier cu extensia .SQL.

Comentarii

Comentariile sunt texte care se introduc în seturile de comenzi pentu a explica rolul lor. Comentariile nu sunt executate. Dacă un comentariu începe intr-o singură linie atunci va fi prefixat cu --, altfel va fi scris între /* și */.

Crearea schemelor

Obiectele unei baze de date sunt organizate în scheme, altfel spus, o schemă este un set logic de obiecte (tabele, vederi, indexuri etc.) ale bazei de date care sunt gestionate împreună. *Schema* este deținută de un *user*(*owner*).

SQL Server vine cu mai multe scheme predefinite, printre care:

sys
information_schema
dbo.

Schemele *sys* și *information_schema* sunt rezervate pentru obiectele de system. Nu putem crea și nici șterge obiecte în/din aceste scheme.

```
select * from sys.schemas
```

name	*schema_id*	*principal_id*
dbo	1	1
guest	2	2
INFORMATION_SCHEMA	3	3
sys	4	4
db_owner	16384	16384
db_accessadmin	16385	16385
db_securityadmin	16386	16386
db_ddladmin	16387	16387
db_backupoperator	16389	16389
db_datareader	16390	16390
db_datawriter	16391	16391
db_denydatareader	16392	16392
db_denydatawriter	16393	16393

```
select name from sys.databases
```

name
Master

Tempdb
Model
Msdb
ReportServer
ReportServerTempDB
dbFacturare

```
use dbFacturare
select name from sys.tables
```

Name
tAngajati
tProduse
tClienti
tFacturi
tDetaliiFact

Schema *dbo* este schema implicită pentru o bază de date nou creată și este deținută de contul utilizator *dbo*.

Schemele permit gestionarea drepturilor utilizatorilor de obiecte. Un utilizator este mapat obligatoriu pe o schema încă de la crearea sa. În cazul în care nici un nume de schemă nu este specificat, atunci utilizatorul va fi mapat în mod implicit pe schema *dbo*.

Pentru a interoga obiectele bazei de date, un utilizator poate scrie doar numele obiectului în cazul în care acesta este inclus în schema pe care este mapat. În caz contrar, utilizatorul trebuie să precizeze schema obiectului, numele obiectului și în acel moment, SQL Server va căuta dacă utilizatorul are dreptul de a utiliza obiectul pe care încearcă să-l acceseze. Reținem faptul că schemele sunt folosite în principal pentru a facilita schimbul de informații între utilizatorii mapați pe aceste scheme.

Putem crea o nouă schemă utilizând comanda simplificată:

```
CREATE SCHEMA Denumire_schema
```

Exemple:

```
use dbFacturare
go
Create schema Aprovizionare
go
Create schema Desfacere
```

```
select * from sys.schemas
```

name	*schema_id*	*principal_id*
dbo	1	1
guest	2	2
INFORMATION_SCHEMA	3	3
sys	4	4
Aprovizionare	5	1
Desfacere	6	1

db_owner	16384	16384
db_accessadmin	16385	16385
db_securityadmin	16386	16386
db_ddladmin	16387	16387
db_backupoperator	16389	16389
db_datareader	16390	16390
db_datawriter	16391	16391
db_denydatareader	16392	16392
db_denydatawriter	16393	16393

Transferul obiectelor dintr-o schemă în alta se face cu comanda ALTER SCHEMA utilizând sintaxa:

```
ALTER SCHEMA denumire_schemă TRANSFER denumire_obiect
```

Ştergerea unei scheme se face cu comanda

```
DROP SCHEMA denumire_schema
```

Exemplificare:

```
create schema Schema1
go
create schema Schema2
go
create table Schema1.t1( x int);

alter schema Schema2 transfer Schema1.t1

drop table Schema2.t1

drop schema Schema2
drop schema Schema1

select * from sys.schemas
```

CREAREA TABELELOR

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

```
CREATE TABLE
[denumire_baza_date.][denumire_schema.]denudenumire_tabel
    ( { < definiţie_coloană >
        | denumire_coloană AS expresie_coloană_calculată
        | < constrângere_tabel >
      } [ ,... ]
    )
```

```
< definiţie_coloană > ::= denumire_coloană  tip_dată
    [ [DEFAULT expresie_constantă] |
      [IDENTITY[(valoare_iniţială, 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. Coloana calculată nu poate participa la crearea constrângerilor.

Următorul exemplu conține definiții de coloane calculate:

```
create table tStudenti
( codStd int primary key,
  nume varchar(20) not null,
  prenume char(20) not null,
  numeComplet as nume+' '+prenume,
  cnp char(13) constraint ix_cnp unique,
  dataNasterii as convert(smalldatetime,substring(cnp,2,6),12),
 --12 indica faptul ca stringul este de forma: yymmdd(an,luna,zi)
-- ...
)
```

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

- Integritatea domeniului: defineşte un interval sau o listă de valori posibile pentru o coloană;
- Integritatea entităţii: în cadrul unei entităţi nu pot exista două realizări (instanţe) identice;
- Integritatea referenţială: 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 şi 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

Defineşte o restricţie de domeniu pe care trebuie să o satisfacă datele din coloana respectivă. Expresia logică ataşată 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 constrîngerea *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.

Exemple:

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

```
Constraint pk_DetaliiFact primary key(NrFact,CodPrododus)
```

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
```

```
constraint uk1 unique(Nume,Prenume,DataNasterii)
```

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 integrităţii referenţiale

Constrângerea Foreign key se foloseşte, uzual împreună cu Primary key dar poate fi folosită și cu o cheie de tip unique, pentru a rezolva problema integrităţii referenţiale.

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 referite) trebuie să fie definită ca *Primary key*, *Unique* sau *index unic* în tabelul referit.

ON DELETE CASCADE – menţine integritatea referenţială în cazul ştergerii unui rând din `tabel_referit` (care conţine cheia primară sau unică), prin ştergerea tuturor rândurilor ce conţin cheia străină dependentă.

Valoarea implicită este ON DELETE NO ACTION.

ON UPDATE CASCADE -menţine integritatea referenţială în cazul modificării valorii coloanei referite din tabelul asociat (care conţine cheia primară sau unică), prin propagarea modificării tuturor rândurilor ce conţin cheia străină dependentă.

Valoarea implicită este ON UPDATE NO ACTION.

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
              [ 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   < definiţie_coloană > [,...]
```

< definiţie_coloană > are aceeaşi semnificaţie 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
```

- Ştergerea 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 stabileşte dacă datele existente sunt sau nu validate în raport cu constrângerea nou adăugată. Opţiunea implicită este WITH CHECK.

< constrângere_tabel > are aceeaşi semnificaţie 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
```

```
Alter table tChitante
 add constraint df_dataChitanta default getdate() for dataChitanta
```

```
Alter table tChitante
  add default 0 for sumaAchitata
```

- Ştergerea unor constrângeri :

```
ALTER TABLE  denumire_tabel
       DROP CONSTRAINT  denumire_constrângere  [ ,... ]
```

Exemplu:

```
 Alter table tSalarii
     drop constraint  ck_cifra_unit_este_zero
```

– Activarea(CHECK) sau dezactivarea(NOCHECK) constrângerilor de tip CHECK şi 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

Constrângerile ne permit să definim o modalitate de a pune în aplicare în mod automat integritatea unei baze de date. Uneori poate fi necesar să dezactivăm una sau toate constrângerile tabelului, în scopul de a importa date, trunchia tabele, etc
Dacă dezactivăm o constrângere pentru anumite motive, trebuie să ne asigurăm că o reactivăm mai târziu.
Constrângerile *Foreign key* și *Check* sunt constrângerile care pot fi dezactivate/activate. Constrângerile *primary key* și constrângerile *unique* nu pot fi dezactivate.
Fie constrângerea:

```
ALTER table tProduse  WITH NOCHECK  ADD constraint CK__tProduse__UM
       CHECK  ([UM] in ('Kg','L','Buc','Bax'))
```

Pentru a introduce]n tabelul tProduse un produs care nu verifică această constrângere, putem proceda astfel:

```
alter table tProduse  nocheck constraint CK__tProduse__UM

insert into tProduse values('50001','ciment','sac',25)

alter table tProduse  check constraint CK__tProduse__UM
```

Observaţie legată de ştergerea coloanelor. Dacă există constrângeri legate de o coloană pe care dorim să o ştergem, atunci, mai întâi trebuie să ştergem constrângerile impuse ei(*drop constraint*) după care putem şterge coloana (*drop column*):

```
alter table tCandidati
 add mediu char
     constraint ck_rural_urban  check(mediu in ('R','U'))
     constraint df_rural_urban  default 'U'

 -- …

alter table tCandidati
drop constraint df_rural_urban, ck_rural_urban

alter table tCandidati
drop column mediu
```

In următorul exemplu adăugăm cheii străine clauzele *on delete cascade* și *on update cascade*:

```
alter table tDetaliiFact
   drop constraint FK__tDetaliiF__NrFac__2F10007B
```

```
alter table tDetaliiFact
  add constraint FK__tDetaliiF__NrFact
    foreign key (nrfact)  references tFacturi(nrfact)
      on delete cascade
      on update cascade
```

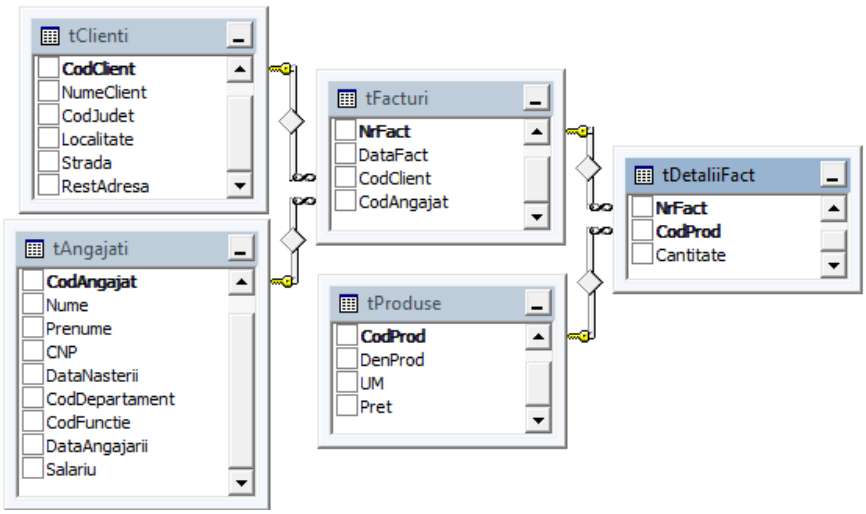
ŞTERGEREA TABELELOR

Comanda DROP TABLE şterge din baza de date definiţia unui tabel împreună cu indecşii, triggerii şi constrângerile asociate. Sintaxa:

```
DROP TABLE denumire_tabel
```

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

Aplicaţia 1: Comenzile de mai jos crează baza de date dbFacturare care implementează următorul model relațional:

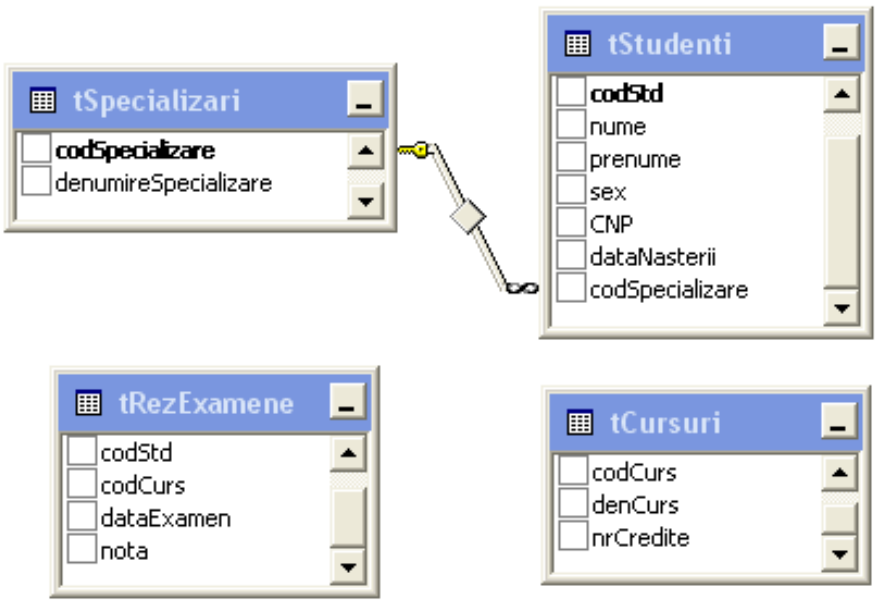



```
use dbFacturare

if OBJECT_ID('tDetaliiFact') is not null
 drop table tDetaliiFact
if OBJECT_ID('tFacturi') is not null
 drop table tFacturi
 if OBJECT_ID('tProduse') is not null
 drop table tProduse
 if OBJECT_ID('tClienti') is not null
 drop table tClienti
if OBJECT_ID('tAngajati') is not null
 drop table tAngajati

create table tAngajati
  ( CodAngajat char(10) primary key,
    Nume varchar(25) not null,
    Prenume varchar(30) not null,
    CNP char(13) not null,
    DataNasterii as convert (smalldatetime,substring(cnp,2,6),12),
```

```
    -- DataNasterii este coloană calculată
    CodDepartament char(10) not null,
    CodFunctie char(6) not null,
    DataAngajarii smalldatetime not null,
    Salariu int not null
  )

create table tProduse
 ( CodProd char(10) primary key,
   DenProd varchar(30) not null,
   UM char(6) not null,
   Pret numeric(7,2) not null
 )

create table tClienti
  ( CodClient char(10) primary  key,
    NumeClient varchar(30) not null,
    CodJudet char(2) default 'AG',
    Localitate varchar(30) not null,
    Strada varchar(30),
    RestAdresa varchar(40)
  )

create table tFacturi
  ( NrFact char(10) primary key,
    DataFact smalldatetime default getdate(),
    CodClient char(10) foreign key references tClienti(codClient),
    CodAngajat char(10) foreign key references tAngajati(codAngajat)
  )

create table tDetaliiFact
 (
   NrFact char(10) foreign key references tFacturi(nrFact),
   CodProd char(10) foreign key references tProduse(codProd),
   Cantitate numeric(8,2) not null,
   Constraint pk_DetaliiFact primary key(NrFact,CodProd)
 )
```

Aplicaţia 2:

In exemplul de mai jos sunt create tabelele tSpecializari, tStudenti, tRezExamene, tCursuri fiind impuse diverse constrângeri. Ulterior creării, se vor folosi comenzi *alter table* pentru adăugarea de noi coloane şi pentru impunerea de noi constrângeri.




```
create database dbStudenti
go
```

```
use dbStudenti
```

```
Create table tSpecializari
( codSpecializare    char(10) primary key,
  denumireSpecializare     varchar(30) not null
)
```

```
Create table tStudenti(
  codStd             char(10) primary key,
  nume               varchar (20),
  prenume            varchar(25),
  sex                char check (sex in ('m' ,'f')),
  CNP                char(13),
  dataNasterii       as
substring(CNP,6,2)+'/'+substring(CNP,4,2)+'/'+substring(CNP,2,2),
  codSpecializare char(10)  foreign key references
tSpecializari(codSpecializare)
)
```

```
create table tCursuri(
  codCurs   char(10),
  denCurs   char(30),
  nrCredite       numeric(2,0),
)
```

```
create table tRezExamene(
  codStd    char(10) not null,
  codCurs   char(10) not null,
  dataExamen      datetime not null,
  nota            numeric(2,0) not null
)
```

MODIFICAREA TABELELOR

```
Alter table tStudenti
  add codJud char(2) default 'AG',
      localitate varchar(20)
```

```
Alter table tCursuri
  add  tipCurs char   default 'B' check (tipCurs in ('B','O','F'))
              /*
               B=din modulul de baza(obligatoriu)
               O=optional
               F=facultativ
              */
```

```
Alter table tRezExamene
  add constraint ck_nota check (nota between 1 and 10)
```

```
Alter table tRezExamene
  add constraint pk_ex primary key(codStd,codCurs)
```

```
Alter table tCursuri
  alter column codCurs char(10) not null
```

```
Alter table tStudenti
  alter column codStd char(10) not null
```

```
Alter table tCursuri
  add  constraint pk_curs primary key (codCurs);
```

```
Alter table tStudenti
  add constraint pk_CodStd primary key(codStd)
```

```
Alter table tRezExamene
  add constraint fk_std
  foreign key (codStd) references tStudenti(codStd)
  on delete cascade on update cascade
```

```
Alter table tRezExamene
  add constraint fk_curs
  foreign key (codCurs) references tCursuri(codCurs)
```

STERGEREA TABELELOR DIN BAZA DE DATE

```
 Drop table tRezExamene
 Drop table tStudenti
 Drop table tCursuri
 Drop table tSpecializari
```

Bibliografie

http://msdn.microsoft.com/en-us/library/aa258256(SQL.80).aspx
http://www.dotnet-france.com