SQL - 5

CREAREA TABELELOR

STUD

MATR	NUME	AN	GRUPA	DATAN	LOC	TUTOR	PUNCTAJ	CODS
1456	GEORGE	4	1141A	12-MAR-82	BUCURESTI		2890	11
1325	VASILE	2	1122A	05-OCT-84	PITESTI	1456	390	11
1645	MARIA	3	1131B	17-JUN-83	PLOIESTI		1400	11
3145	ION	1	2112B	24-JAN-85	PLOIESTI	3251	1670	21
2146	STANCA	4	2141A	15-MAY-82	BUCURESTI		620	21
3251	ALEX	5	2153B	07-NOV-81	BRASOV		1570	21
2215	ELENA	2	2122A	29-AUG-84	BUCURESTI	2146	890	21
4311	ADRIAN	3	2431A	31-JUL-83	BUCURESTI		450	24
3514	FLOREA	5	2452B	03-FEB-81	BRASOV		3230	24
1925	OANA	2	2421A	20-DEC-84	BUCURESTI	4311	760	24
2101	MARIUS	1	2412B	02-SEP-85	PITESTI	3514	310	24
4705	VOICU	2	2421B	19-APR-84	BRASOV	4311	1290	24

SPEC si BURSA

CODS	NUME	DOMENIU
11	MATEMATICA	STIINTE EXACTE
21	GEOGRAFIE	UMANIST
24	ISTORIE	UMANIST

TIP	PMIN	PMAX	SUMA
FARA BURSA	0	399	
BURSA SOCIALA	400	899	100
BURSA DE STUDIU	900	1799	150
BURSA DE MERIT	1800	2499	200
BURSA DE EXCEPTIE	2500	9999	300

OBIECTIV

Scopul acestui capitol este de a prezenta elementele limbajului pentru descrierea datelor (DDL) referitoare la:

- Tipurile de date permise pentru coloanele tabelelor,
- Crearea de noi tabele,
- Constrângeri de integritate,
- Modificarea structurii unei tabele,
- Modificarea şi activarea constrângerilor de integritate,
- Dicţionarul de date al sistemului.

TIPURI DE DATE

Sistemul Oracle pune la dispoziţie un set optim de tipuri de date care pot fi asociate coloanelor unei tabele, grupate în mai multe categorii:

- Tipuri numerice scalare
- Tipuri scalare şir de caractere
- Tipuri scalare binare
- Tipuri pentru date calendaristice, timp şi interval de timp
- Tipuri LOB (large object)
- Tipuri compuse: TABLE şi VARRAY

TIPURI DE DATE (2)

- Aceste tipuri, cu unele diferenţe în ceea ce priveşte dimensiunea maximă admisă, pot fi folosite şi în limbajul procedural PL/SQL.
- În continuare sunt prezentate tipurile din primele cinci categorii.
- Crearea unei tabele cu coloane de tip TABLE şi VARRAY şi manipularea datelor de acest fel nu este prezentată în acest capitolul.

TIPURI NUMERICE SCALARE

Tip	Descriere
NUMBER	Număr real de dimensiune variabilă, cu 38 de cifre semnificative, având valori între 1E-130 şi 10E125.
NUMBER(n)	Număr întreg cu maxim *n* cifre
NUMBER(n, z)	Număr real cu *n* cifre dintre care *z* zecimale
DEC, DECIMAL, NUMERIC	Subtipuri pentru NUMBER. Numere în virgulă fixă cu 38 de cifre semnificative.
DOUBLE PRECISION, FLOAT	Subtipuri pentru NUMBER. Numere în virgulă mobilă cu 38 de cifre semnificative.
REAL	Subtip pentru NUMBER. Numere în virgulă mobilă cu 18 cifre semnificative.

TIPURI NUMERICE SCALARE (2)

Tip	Descriere
INTEGER, INT, SMALLINT	**Subtipuri pentru NUMBER. Numere întregi cu maxim 38 de cifre**
BINARY_INTEGER	**Întregi între -2^{31} şi 2^{31}.**
NATURAL, POSITIVE	**Subtipuri pentru BINARY_INTEGER. Numere întregi non-negative, respectiv pozitive.**
NATURALN, POSITIVEN	**Subtipuri pentru BINARY_INTEGER. Numere întregi nenule non-negative, respectiv pozitive.**
SIGNTYPE	**Subtip al BINARY_INTEGER. Poate lua doar valorile -1, 0 şi 1.**
PLS_INTEGER	**Întregi între -2^{31} şi 2^{31}. Similar cu BINARY_INTEGER dar operaţiile cu astfel de numere sunt mai rapide şi în caz de depăşire se ridică o excepţie.**

SIRURI

Tip	Descriere
CHAR(n), CHAR	**Şir de caractere de lungime fixă, egală cu *n*. Valoarea maximă pentru *n* este 2000. Dacă n lipseşte, şir de caractere de lungime 1.**
CHARACTER, CHARACTER(n)	**Identice cu cele anterioare. Introduse pentru compatibilitatea cu alte sisteme.**
NCHAR(n)	**Analog cu CHAR dar poate stoca şiruri scrise în seturi de caractere naţionale (multioctet)**
VARCHAR2(n)	**Şir de caractere de lungime variabilă egală cu *n*. Valoarea maximă pentru *n* este 4000.**
STRING(n), VARCHAR(n)	**Identice cu VARCHAR2, introduse pentru compatibilitatea cu alte sisteme**
NVARCHAR(n)	**Analog cu VARCHAR. Poate stoca şiruri scrise în seturi de caractere naţionale (multioctet)**

SIRURI (2)

Tip	Descriere
LONG	**Şir de caractere de maxim 2^{31} octeţi. Este permisă doar o singură coloană de acest tip pentru o tabelă.**
ROWID	**Poate stoca un identificator pentru o linie dintr-o tabelă. Pentru conversia la/de la şir de caractere (18 caractere) se pot folosi funcţiile SQL ROWIDTOCHAR respectiv CHARTOROWID.**
UROWID	**Universal ROWID. Poate stoca un identificator logic şi fizic de linie într-o tabelă, indexată sau nu precum şi un identificator de linie extern (non-Oracle). Nu este necesară folosirea funcţiilor de conversie la/de la şir de caractere (conversie automată).**

TIPURI BINARE

Tip	Descriere
RAW(n)	Similar cu VARCHAR2 dar conține date binare. Valoarea maximă pentru *n* este de 2000.
LONG RAW	Similar cu LONG dar conține date binare.

DATE, TIMP SI INTERVAL

Tip	Descriere
DATE	**Dată calendaristică (secol, an, lună, zi, oră, minut, secundă).**
TIMESTAMP[(n)]	**Extensie a tipului DATE. Conţine şi fracţiuni de secundă. Dacă este prezent, *n* specifică numărul de zecimale pentru acestea. Implicit n = 6**
TIMESTAMP [(n)] WITH TIME ZONE	**Extinde tipul TIMESTAMP conţinând şi o diferenţă între ora locală şi ora universală (GMT)**
TIMESTAMP [(n)] WITH LOCAL TIME ZONE	**Similar cu tipul anterior dar la stocarea în baza de date valorile sunt convertite la ora zonei bazei de date iar la regăsire la ora zonei aplicaţiei client.**
INTERVAL YEAR [(n)] TO MONTH	**Se stochează intervale de ani şi luni. Precizia *n* specifică numărul de cifre pentru an (între 0 şi 4, implicit 2).**
INTERVAL DAY [(z)] TO SECOND [(s)]	**Similar cu tipul anterior, dar pentru intervale de zile şi secunde. valorile *z* şi *s* sunt preciziile pentru zile, respectiv secunde (0-9, implicit 2 pentru *z* şi 6 pentru *s*).**

TIPURI LARGE OBJECT

- Aceste tipuri, introduse în ultimele versiuni ale sistemului, permit stocarea unor cantităţi mari de date pe coloanele unei tabele sau a unei referinţe (numită şi locator) către un fişier extern bazei de date.
- Manipularea valorilor de acest tip se face în PL/SQL cu ajutorul pachetelor de proceduri şi funcţii puse la dispoziţie de sistem.
- Caracteristicile lor sunt următoarele:

TIPURI LOB – cont.

Tip	Descriere
CLOB	**Şir de caractere de până la 4 GB. Se recomandă ca în aplicaţiile noi să fie folosit în locul lui LONG.**
NCLOB	**Similar cu CLOB dar se pot stoca şiruri utilizând seturi naţionale de caractere. Dimensiunea maximă este de asemenea de 4 GB**
BLOB	**Date binare de dimensiune până la 4 GB**
BFILE	**Date binare de dimensiune până la 4 GB stocate în fişiere externe. Nu participă la tranzacţii, replicare şi pot fi doar citite nu şi modificate.**

CREATE TABLE

- Cea mai simplă formă a cererii SQL de creare a unei noi tabele are următoarea sintaxă:

```
CREATE TABLE [schema.]nume_tabela
   (nume_coloana_1  tip_coloana_1 [DEFAULT
   expresie_1],
    nume_coloana_2  tip_coloana_2 [DEFAULT
   expresie_2],
        .   .   .
    nume_coloana_n  tip_coloana_n [DEFAULT
   expresie_n]);
```

unde:

- **nume_tabela** este numele tabelei care se crează,
- **nume_coloana_1, nume_coloana_2, ...** sunt numele coloanelor acesteia

CREATE TABLE – cont.

- **tip_coloana_1, tip_coloana_2, ...** reprezintă tipurile de date pentru coloanele respective, alese dintre cele prezentate în paragraful anterior, cu specificarea, dacă este cazul, a dimensiunii maxime sau preciziei,
- clauza opţională **DEFAULT expresie_i** specifică o valoare implicită care este introdusă automat în acea coloană în cazul în care la adăugarea unei noi linii nu se specifică o valoare pe coloana respectivă,
- schema este numele de utilizator Oracle al proprietarului noii tabele. Valoarea implicită pentru acesta este numele utilizatorului care execută cererea de creare.

TREBUIE CA:

- Utilizatorul are drepturile necesare (dreptul sau privilegiul - în terminologia uzuală - de CREATE TABLE).
- Există spaţiu de stocare pentru noua tabelă.
- Numele tabelei şi al coloanelor respectă restricţiile uzuale Oracle (maxim 30 de caractere, începe cu o literă, conţine litere, cifre şi caracterele _, $, #, nu este cuvânt rezervat Oracle). Sunt permise şi alte caractere dacă numele este inclus între ghilimele.
- Nu există deja un alt obiect cu acelaşi nume în aceeaşi schemă (pentru acelaşi utilizator Oracle).

LITERE MARI SI MICI

- Ca şi în cazul cuvintelor cheie, literele mici şi cele mari sunt considerate egale în numele de tabele şi coloane.
- Exemplu: tabela de studenţi din exemple este aceeaşi dacă la creare se foloseşte oricare din numele STUD, Stud sau StuD.

OBSERVATII

- Expresia din clauza opţională DEFAULT trebuie să se evalueze la o valoare de tip compatibil cu al coloanei respective. Ea poate conţine:
  - Constante numerice sau şir de caractere,
  - Funcţii SQL, inclusiv SYSDATE sau USER.
  - dar nu poate conţine:
  - Numele unei coloane,
  - Numele unei pseudocoloane. (de exemplu pseudocoloanele definite de o secvenţă: NEXTVAL sau CURRVAL).

EXEMPLU

```
CREATE TABLE STUD(
MATR NUMBER(4),
NUME VARCHAR2(10),
AN NUMBER(1) DEFAULT 1,
GRUPA VARCHAR2(6),
DATAN DATE,
LOC VARCHAR2(10) DEFAULT 'BUCURESTI',
TUTOR NUMBER(4),
PUNCTAJ NUMBER(4) DEFAULT 0,
CODS NUMBER(2) );
```

OBSERVAM CA:

- MATR, AN, TUTOR, PUNCTAJ şi CODS care conţin valori de tip număr întreg au fost definite ca NUMBER(n) unde n este 1, 2 sau 4,
- NUME, GRUPA şi LOC sunt şiruri de caractere de lungime variabilă definite ca VARCHAR2,
- DATAN care conţine data naşterii studentului este de tipul DATE,
- Pentru unele dintre coloane au fost asociate valori implicite care vor fi stocate automat în acestea dacă la inserarea unei noi linii nu se specifică o valoare, nulă sau nenulă.
- Valorile implicite sunt compatibile cu tipul coloanelor respective (întreg sau şir de caractere, după caz).

- Crearea unei tabele în care se stochează date despre evenimente: momentul de început, durata şi o descriere a acestora.

```
CREATE TABLE EVENIMENT(
COD NUMBER(10),
"MOMENT INCEPUT" TIMESTAMP(3) WITH LOCAL
  TIME ZONE,
DURATA INTERVAL DAY(2) TO SECOND(3),
"DESCRIERE (PE LARG)" LONG);
```

OBSERVAM CA:

- COD: un număr întreg de maxim 10 cifre,
- MOMENT ÎNCEPUT: conţine momentul începutului unui eveniment în forma: data, ora, minutul, secunda şi miimile de secundă: TIMESTAMP(3) arată că fracţiunile de secundă sunt memorate cu 3 zecimale. Numele coloanei conţine un spaţiu şi a trebuit pus între ghilimele.
- DURATA: durata evenimentului în zile, ore, minute, secunde şi fracţiuni de secundă. Numărul de zile poate avea maxim 2 cifre iar numărul de secunde maxim 3 zecimale.
- DESCRIERE (PE LARG): conţine un text de descriere a evenimentului care poate avea până la 2 GB caractere. Se folosesc ghilimelele pentru că numele coloanei conţine spaţii şi paranteze.

CREATE din SELECT

```
CREATE TABLE [schema.]nume_tabela
   [(descriere_coloana_1, ...,
   descriere_coloana_n)]
AS
   cerere_SELECT;
```

- Se crează o tabelă având numele specificat şi aceeaşi structură cu a rezultatului returnat de SELECT.

CREATE din SELECT (2)

- **Descrierea coloanelor nu este prezentă în cerere:**
  - Numele coloanelor tabelei create precum şi tipul acestora este identic cu al celor din rezultatul cererii SELECT.
  - Noua tabelă nu moşteneşte nici una dintre constrângerile de integritate ale tabelei/tabelelor din care provine rezultatul.
  - Dacă în lista de expresii din clauza SELECT există unele care nu returnează pentru capul de tabel al rezultatului un nume valid de coloană este obligatorie folosirea unor aliasuri de coloană.

CREATE din SELECT (3)

```
CREATE TABLE STUD11
AS
  SELECT MATR, NUME, PUNCTAJ*1.1 "PUNCTAJ MARIT"
  FROM STUD WHERE CODS = 11;
```

- Conţinutul tabelei STUD11 va fi următorul:

```
 MATR NUME       PUNCTAJ MARIT
----- ---------- -------------
 1456 GEORGE               3179
 1325 VASILE                429
 1645 MARIA                1540
```

- iar descrierea structurii sale este:

```
Nume coloana    Tip
------------    ------------
MATR            NUMBER(4)
NUME            VARCHAR2(10)
PUNCTAJ MARIT   NUMBER
```

CREATE din SELECT (4)

- Crearea tabelei se face inclusiv în cazul în care cererea SELECT nu returnează nici o linie. De exemplu:

```
CREATE TABLE STUD100
AS
   SELECT MATR, NUME, PUNCTAJ*1.1 "PUNCTAJ
   MARIT"
   FROM STUD
   WHERE CODS = 100;
```

- va avea ca efect crearea tabelei STUD100 având aceeaşi structură cu STUD11 dar aceasta nu va conţine nici o linie.

CREATE din SELECT (5)

- **Cererea conţine descrierea coloanelor:**
- Descrierea unei coloane are următoarea formă:

nume_coloana [DEFAULT expresie] [constrângeri_integritate]

- Numărul de descrieri de coloană trebuie să fie egal cu numărul de coloane din rezultatul cererii SELECT.
- Numele coloanelor tabelei create este cel din descriere iar tipul lor este identic cu al celor din rezultatul cererii SELECT.
- Dacă descrierea unei coloane conţine clauza DEFAULT expresie, se asociază acestei coloane valoarea implicită respectivă.
- Noua tabelă nu moşteneşte nici una dintre constrângerile de integritate ale tabelei/tabelelor din care provine rezultatul dar primeşte constrângerile de integritate din descriere, dacă acestea există

CREATE din SELECT (6)

```
CREATE TABLE STUD11
(NUMAR DEFAULT 0 NOT NULL, NUME, PUNCTE NOT NULL)
AS
   SELECT MATR, NUME, PUNCTAJ*1.1 "PUNCTAJ MARIT"
   FROM STUD
   WHERE CODS = 11;
```

- Conţinutul tabelei este acelaşi, ultima coloană are alt nume, prima are o valoare implicită iar două dintre ele au asociată o constrângere de tip NOT NULL (nu se pot stoca valori nule pe acele coloane):

```
Nume coloana   Null?    Tip             Implicit
-------------- -------- --------------  ---------
NUMAR          NOT NULL NUMBER(4)            0
NUME                    VARCHAR2(10)
PUNCTE         NOT NULL NUMBER
```

CONSTRANGERI

- Constrângerile de integritate reprezintă reguli pe care valorile conţinute într-o tabelă trebuie să le respecte.
- Ele previn introducerea de date eronate în baza de date şi definesc forma corectă a valorilor respective dar nu iau în consideraţie semnificaţia acestora.
- Constrângerile de integritate sunt verificate automat de sistemul de gestiune atunci când au loc operaţii de modificare a conţinutului tabelelor (adăugare, ştergere şi modificare linii).
- În cazul în care noile valori nu sunt valide operaţia de modificare este rejectată de sistem şi se generează o eroare.

TIPURI DE CONSTRANGERI

- NOT NULL: valorile nu pot fi nule
- PRIMARY KEY: defineşte cheia primară a unei tabele
- UNIQUE: defineşte o altă cheie a tabelei
- FOREIGN KEY: defineşte o cheie străină (externă)
- CHECK: introduce o condiţie (expresie logică).

CONSTRANGERI (2)

- Fiecare constrângere de integritate poate avea asociat un nume specificat la crearea tabelei care permite activarea sau dezactivarea constrângerii şi alte operaţii cu aceasta.
- În cazul în care nu se asociază un astfel de nume sistemul generează automat unul.
- Locul definirii unei constrângeri de integritate în cererea de creare a unei tabele poate fi:

CONSTRANGERI (3)

- În descrierea unei coloane, dacă acea constrângere se referă doar la aceasta (uzual se spune despre o astfel de constrângere că este definită *la nivel de coloană*),
- În continuarea listei de descrieri de coloane (*la nivel de tabelă*).
- În funcţie de locul unde se găseşte, sintaxa definirii unei constrângeri poate fi diferită.
- Pentru fiecare tip de constrângere sunt prezentate ambele sintaxe şi exemple de folosire a lor.

DETALIERE CONSTRANGERI

- NOT NULL: valorile nu pot fi nule
- PRIMARY KEY: defineşte cheia primară a unei tabele
- UNIQUE: defineşte o altă cheie a tabelei
- FOREIGN KEY: defineşte o cheie străină (externă)
- CHECK: introduce o condiţie (expresie logică).

NOT NULL

- Acest tip de constrângere se aplică unei coloane a noii tabele şi specifică faptul că aceasta nu poate conţine valori nule. El poate fi definit doar la nivel de coloană şi are următoarea sintaxă:

```
coloana [CONSTRAINT nume_constr] NOT NULL
```

EXEMPLU

```
CREATE TABLE SPEC
(CODS NUMBER(2),
 NUME VARCHAR2(10) CONSTRAINT
  NUMENENUL NOT NULL,
 DOMENIU VARCHAR2(15) NOT
  NULL);
```

DETALIERE CONSTRANGERI

- NOT NULL: valorile nu pot fi nule
- PRIMARY KEY: defineşte cheia primară a unei tabele
- UNIQUE: defineşte o altă cheie a tabelei
- FOREIGN KEY: defineşte o cheie străină (externă)
- CHECK: introduce o condiţie (expresie logică).

PRIMARY KEY

- Sintaxa la nivel de coloană:

```
coloana [CONSTRAINT nume_constrangere]
 PRIMARY KEY
```

- Sintaxa la nivel de tabelă:

```
[,CONSTRAINT nume_constrangere] PRIMARY
KEY(lista_coloane)
```

EXEMPLE

```
CREATE TABLE SPEC(
CODS NUMBER(2) CONSTRAINT SPEC_PK PRIMARY
  KEY,
NUME VARCHAR2(10),
DOMENIU VARCHAR2(15));
```

Si la nivel de tabela:

```
CREATE TABLE SPEC(
CODS NUMBER(2),
NUME VARCHAR2(10),
DOMENIU VARCHAR2(15),
CONSTRAINT SPEC_PK PRIMARY KEY(CODS));
```

PK DIN >1 ATRIBUT

```
CREATE TABLE BURSA(
PMIN NUMBER(4),
PMAX NUMBER(4),
TIP VARCHAR2(20),
SUMA NUMBER(4),
CONSTRAINT BURSA_PK PRIMARY KEY(PMIN,PMAX));
```

DETALIERE CONSTRANGERI

- NOT NULL: valorile nu pot fi nule
- PRIMARY KEY: defineşte cheia primară a unei tabele
- UNIQUE: defineşte o altă cheie a tabelei
- FOREIGN KEY: defineşte o cheie străină (externă)
- CHECK: introduce o condiţie (expresie logică).

UNIQUE

- Sintaxa la nivel de coloană:

```
coloana [CONSTRAINT nume_constrangere]
 UNIQUE
```

- Sintaxa la nivel de tabelă:
-

```
 [,CONSTRAINT nume_constrangere]
UNIQUE(lista_coloane)
```

EXEMPLUL 1

- Dacă în tabela de SPEC nu pot exista niciodată două specializări cu acelaşi nume, definiţia anterioară se completează cu o constrângere de tip UNIQUE:

```
CREATE TABLE SPEC(
CODS NUMBER(2) CONSTRAINT SPEC_PK
  PRIMARY KEY,
NUME VARCHAR2(10) CONSTRAINT NUMES_UNIC
  UNIQUE,
DOMENIU VARCHAR2(15));
```

EXEMPLUL 1 – cont.

- Aceeasi constrangere, la nivel de tabela:

```
CREATE TABLE SPEC(
CODS NUMBER(2) CONSTRAINT SPEC_PK PRIMARY
  KEY,
NUME VARCHAR2(10),
DOMENIU VARCHAR2(15),
CONSTRAINT NUMES_UNIC UNIQUE(NUME));
```

EXEMPLUL 2

- În tabela BURSA nu pot exista două burse cu aceeaşi sumă. Cererea de creare anterioară devine:

```
CREATE TABLE BURSA(
PMIN NUMBER(4),
PMAX NUMBER(4),
TIP VARCHAR2(20),
SUMA NUMBER(4) CONSTRAINT SUMA_UNICA
  UNIQUE,
CONSTRAINT BURSA_PK PRIMARY KEY(PMIN,
  PMAX));
```

- În acest caz în tabelă nu pot exista două linii care conţin aceeaşi valoare nenulă pe coloana SUMA dar pot exista oricâte linii cu valori nule pe această coloană.

CE E UNIC?

- În cazul în care o cheie definită cu UNIQUE conţine mai multe coloane, verificarea se face astfel:
  - nu există două linii care au aceleaşi valori nenule pentru toate coloanele cheii.
  - nu există două linii care au aceleaşi valori nenule pentru unele coloane ale cheii şi valori nule în rest.
- Exemplu: În cazul unei tabele NUMERE creată cu cererea:

```
CREATE TABLE NUMERE(
NUMAR1 NUMBER(4),
NUMAR2 NUMBER(4),
UNIQUE(NUMAR1, NUMAR2));
```

CONTINUT VALID

NUMAR1	NUMAR2
NULL	NULL
NULL	NULL
1000	2000
1000	3000
NULL	2000
1000	NULL

CONTINUT INVALID

NUMAR1	NUMAR2
NULL	NULL
NULL	NULL
1000	2000
1000	3000
NULL	2000
1000	NULL
1000	2000
NULL	2000
1000	NULL

DETALIERE CONSTRANGERI

- NOT NULL: valorile nu pot fi nule
- PRIMARY KEY: defineşte cheia primară a unei tabele
- UNIQUE: defineşte o altă cheie a tabelei
- FOREIGN KEY: defineşte o cheie străină (externă)
- CHECK: introduce o condiţie (expresie logică).

FOREIGN KEY

- Prin această constrângere valorile unei coloane/unor coloane sunt forţate să fie doar dintre cele ale cheii unei tabele (cheie primară sau UNIQUE).
- Coloanele constrânse în acest fel formează ceea ce se numeşte în terminologia de specialitate o *cheie straina* sau *cheie externa* iar constrângerea mai este denumită şi *de integritate referentiala*.

SINTAXA

- La nivel de coloană:

```
coloana [CONSTRAINT nume_constrangere]
  REFERENCES tabela(coloana)
  [ON DELETE CASCADE |
   ON DELETE SET NULL]
```

- La nivel de tabelă:

```
[,CONSTRAINT nume_constrangere] FOREIGN
 KEY(lista_coloane)
 REFERENCES tabela(lista_coloane)
 [ON DELETE CASCADE |
  ON DELETE SET NULL]
```

- În cazul tabelei STUD avem două coloane care pot avea asociată o astfel de constrângere:
  - Coloana CODS poate fi constrânsă să conţină doar valori ale cheii primare a tabelei SPEC (formată dintr-o coloană cu acelaşi nume - CODS).
  - Coloana TUTOR poate fi constrânsă să conţină doar valori nenule ale cheii primare a lui STUD - MATR.
- În acest caz cererea anterioara de creare pentru STUD devine:

STUD:

```
CREATE TABLE STUD(
MATR NUMBER(4) PRIMARY KEY,
NUME VARCHAR2(10),
AN NUMBER(1) DEFAULT 1,
GRUPA VARCHAR2(6),
DATAN DATE,
LOC VARCHAR2(10) DEFAULT 'BUCURESTI',
TUTOR NUMBER(4) REFERENCES STUD(MATR),
PUNCTAJ NUMBER(4) DEFAULT 0,
CODS NUMBER(2),
CONSTRAINT CODS_FK FOREIGN KEY(CODS)
   REFERENCES SPEC(CODS));
```

OBSERVATII

- În cazul în care tabela SPEC nu există încă sistemul va semnala eroarea: *ORA-00942: table or view does not exist*
- În cazul în care tabela SPEC există dar nu are cheia CODS se semnalează eroarea: *ORA-02270: no matching unique or primary key for this column-list*
- După crearea lui STUD, dacă se încearcă ştergerea tabelei SPEC se semnalează eroarea: *ORA-02449: unique/primary keys în table referenced by foreign keys*

OBSERVATII – cont.

- La inserarea de noi linii în tabela STUD, acestea trebuie să conţină pe coloanele CODS şi TUTOR fie valori care există deja în SPEC.CODS şi STUD.MATR, fie valori nule.
- La încercarea de a şterge linii din tabelele SPEC şi STUD, dacă acestea au valoari ale chei referite în STUD prin constrângerile de mai sus se va semnala de asemenea o eroare.
- La încercarea de a modifica valorile cheii în linii din tabelele SPEC şi STUD, dacă acestea sunt referite în STUD se va semnala de asemenea o eroare.

CLAUZA ON DELETE

- În cazul în care se doreşte ca o linie dintr-o tabelă conţinând o valoare de cheie referită într-o altă tabelă (sau în aceeaşi tabelă) să poată fi ştearsă fără a se obţine un mesaj de eroare, la definirea constrângerii se poate specifica modul în care se tratează această ştergere din perspectiva tabelei care referă valoarea:
  - Daca se doreşte ca la ştergerea liniei conţinând o valoare de cheie să fie şterse suplimentar şi toate liniile care referă această valoare, se specifică ON DELETE CASCADE
  - Dacă se doreşte ca liniile care referă valoarea cheii respective să nu fie şterse, se specifică ON DELETE SET NULL care are următorul efect: în toate liniile care referă acea valoare ea este înlocuită cu valori nule.

EXEMPLU

```
CREATE TABLE STUD(
MATR NUMBER(4) PRIMARY KEY,
. . . . . .
TUTOR NUMBER(4) REFERENCES STUD(MATR)
   ON DELETE SET NULL,
. . . . . .
CONSTRAINT CODS_FK FOREIGN KEY(CODS)
   REFERENCES SPEC(CODS)
   ON DELETE CASCADE);
```

EXEMPLU – cont.

- La ştergerea unei specializări din tabela SPEC vor fi şterşi automat şi toţi studenţii din tabela STUD având pe coloana CODS codul acesteia.
- La ştergerea unui student care este tutorul altor studenţi, automat acestora li se va modifica valoarea pe coloana TUTOR la una nulă.
- Tabela SPEC poate fi golită - ceea ce va antrena şi golirea tabelei STUD - dar nu ştearsă.

FK PENTRU UNIQUE

- În cazul în care constrângerea se defineşte pentru o cheie de tip UNIQUE (care poate conţine şi valori nule) ea este verificată pentru cheile străine formate doar din valori nenule.

DETALIERE CONSTRANGERI

- NOT NULL: valorile nu pot fi nule
- PRIMARY KEY: defineşte cheia primară a unei tabele
- UNIQUE: defineşte o altă cheie a tabelei
- FOREIGN KEY: defineşte o cheie străină (externă)
- CHECK: introduce o condiţie (expresie logică).

CHECK

- Prin acest tip de constrângeri valorile aflate pe o linie a tabelei sunt forţate să verifice o condiţie (expresie logică).
- Aceasta poate conţine toate elementele prezentate anterior pentru conditii, inclusiv apeluri de funcţii, cu câteva excepţii:
  - apelurile unor funcţii ca SYSDATE, USER, UID
  - referinţa la pseudocoloanele NEXTVAL, CURRVAL (specifice secvenţelor), LEVEL şi ROWNUM)
  - subcereri.
- O aceeaşi coloană poate participa la mai multe constrângeri de acest tip, având fiecare o altă condiţie asociată.
- Condiţia se evaluează doar la nivelul liniei respective, fără a se putea lua în considerare valori aflate pe alte linii ale tabelei.

SINTAXA

- La nivel de coloană:

```
coloana [CONSTRAINT nume_constrangere]
CHECK (expresie_logica)
```

- La nivel de tabelă:

```
[,CONSTRAINT nume_constrangere]
  CHECK (expresie_logica)
```

EXEMPLU

- Tabela BURSA poate avea asociate mai multe constrângeri care verifică următoarele condiţii:
  - PMIN şi PMAX sunt pozitive
  - PMIN < PMAX
  - SUMA este între 0 şi 500
- Cererea de creare va fi:

```
CREATE TABLE BURSA(
PMIN NUMBER(4) CHECK (PMIN >= 0),
PMAX NUMBER(4) CHECK (PMAX >= 0),
TIP VARCHAR2(20),
SUMA NUMBER(4) CHECK (SUMA BETWEEN 0 AND 500),
CONSTRAINT PMINPMAX CHECK (PMIN < PMAX));
```

EXEMPLU – cont.

- În cazul în care pe coloanele respective se găsesc valori nule, linia nu este rejectată ci se consideră că verifică expresia logică.
- Pentru tabela BURSA definită ca mai sus se pot introduce linii având valori nule pentru unele din coloanele PMIN, PMAX şi SUMA fără a se semnala violarea constrângerilor de integritate de tip CHECK.
- Expresia logică asociată unei constrângeri CHECK poate fi compusă: cererea de creare de mai sus poate fi rescrisă prin combinarea tuturor condiţiilor în una singură:

EXEMPLU – cont.

```
CREATE TABLE BURSA(
PMIN NUMBER(4), PMAX NUMBER(4),
TIP VARCHAR2(20),
SUMA NUMBER(4),
CONSTRAINT BURSA_CK CHECK (PMIN < PMAX
  AND PMIN >= 0 AND PMAX >= 0 AND SUMA
  BETWEEN 0 AND 500));
```

MODIFICAREA STRUCTURII

- Vizualizarea structurii tabelei
- Adaugarea unei coloane
- Stergerea unei coloane
- Modificarea unei coloane
- Adaugarea unei constrangeri
- Stergerea unei constrangeri
- Activare/dezactivare constrangeri
- Golirea unei tabele
- Stergerea unei tabele
- Redenumirea unei tabele
- Adaugarea de comentarii

1. VIZUALIZARE STRUCTURA

- Clientul SQL*Plus pune la dispoziţie comanda DESCRIBE prin care se pot afişa numele coloanelor, tipul acestora şi eventualele constrângeri de tip NOT NULL.
- Sintaxa comenzii SQL*Plus este:

```
DESCRIBE nume_tabela
```

- Observaţie: DESCRIBE nu este o cerere SQL ci o comandă SQL*Plus. Ea nu trebuie terminată cu punct şi virgulă (;).
- Exemplu. Pentru a obţine descrierea tabelei SPEC folosim comanda:

```
DESCRIBE SPEC
```

- Rezultat obţinut:

```
Name                 Null?    Type
-------------------- -------- ----------------
CODS                          NUMBER(2)
NUME                          VARCHAR2(10)
DOMENIU                       VARCHAR2(15)
```

2. ADAUGARE COLOANA

- Sintaxa:

```
ALTER TABLE nume_tabela ADD
  (nume_coloana tip_date
   [DEFAULT expresie] [constrangere]);
```

- Exemplu: Adăugarea unei noi coloane la tabela SPEC în care se memorează anul înfiinţării fiecărei specializări:

```
ALTER TABLE SPEC ADD(AN_INFIINTARE
  NUMBER(4)
   CONSTRAINT ANS_NENUL NOT NULL);
```

ADAUGARE COLOANA – cont.

- Dacă tabela conţine deja cel puţin o linie, pe noua coloană vor fi prezente doar valori nule.
- Adăugarea unei coloane care are asociată o constrângere de tip NOT NULL (ca în exemplul anterior) nu se poate face decât în cazul în care tabela nu conţine nici o linie.
- Coloana se va adăuga întotdeauna după celelalte, devenind ultima coloană din tabelă.

3. STERGERE COLOANA

- Sintaxa:

```
ALTER TABLE nume_tabela DROP COLUMN
  nume_coloana [CASCADE CONSTRAINTS];
```

- sau

```
ALTER TABLE nume_tabela DROP (lista
  coloane) [CASCADE CONSTRAINTS];
```

Exemplu: Ştergerea coloanei adăugată anterior:

```
ALTER TABLE SPEC
DROP COLUMN AN_INFIINTARE;
```

STERGERE COLOANA – cont.

- Cererea se poate executa atât în cazul în care tabela este goală cât şi dacă ea conţine deja linii,
- Dacă tabela are doar o singură coloană, aceasta nu se poate şterge,
- Opţiunea CASCADE CONSTRAINTS şterge suplimentar toate constrângerile de integritate în care sunt implicate coloanele şterse, inclusiv cele de tip FOREIGN KEY care referă valorile coloanei sau coloanelor respective.

UNUSED

- Ştergerea unei coloane este o operaţie costisitoare din punct de vedere al resurselor sistemului.
- De aceea, în cazul în care la un moment dat acesta este foarte încărcat ştergerea se poate face în două etape:
- Etapa 1: Marcarea ca neutilizate a unei coloane sau a unei liste de coloane prin opţiunea SET UNUSED. Sintaxa este:

```
ALTER TABLE nume_tabela SET
  UNUSED(lista_coloane);
```

sau

```
ALTER TABLE nume_tabela SET UNUSED COLUMN
  nume_coloana;
```

UNUSED – cont.

- Etapa 2: Ştergerea efectivă a coloanelor marcate ca neutilizate la un moment ulterior de timp, când sistemul nu mai este foarte încărcat. Sintaxa cererii este:

```
ALTER TABLE nume_tabela DROP
  UNUSED COLUMNS;
```

EXEMPLE

- Pentru marcarea coloanelor DATAN, LOC şi TUTOR din tabela STUD se pot executa cererile:

```
ALTER TABLE STUD SET UNUSED(DATAN,
  LOC);
ALTER TABLE STUD SET UNUSED COLUMN
  TUTOR;
```

- Ştergerea coloanelor marcate se face prin cererea:

```
ALTER TABLE SPEC DROP UNUSED COLUMNS;
```

UNUSED - Observatii

- Coloanele marcate ca neutilizate:
  - nu mai apar în structura afişată de DESCRIBE,
  - nu mai pot fi folosite în cererile SQL asupra tabelei respective
  - se pot adăuga noi coloane în tabelă având acelaşi nume.

4. MODIFICARE COLOANA

- Sintaxa:

```
ALTER TABLE nume_tabela
  MODIFY(nume_coloana [tip_date]
  [DEFAULT expresie] [constrangere])
```

- Prin această cerere se pot modifica următoarele caracteristici ale unei coloane:
  - Se poate schimba tipul de date al coloanei
  - Se poate asocia o nouă valoare implicită
  - Se poate adăuga o constrângere de tip NOT NULL pentru acea coloană.
- Printr-o singură cerere se pot efectua toate operaţiile de mai sus sau doar o parte a lor.

EXEMPLU

- modificarea caracteristicilor coloanelor NUME şi DOMENIU din tabela SPEC este efectuată prin următoarele trei cereri de tip ALTER TABLE:

```
ALTER TABLE SPEC MODIFY (DOMENIU
  VARCHAR2(20) CONSTRAINT DOM_MENUL NOT
  NULL);
ALTER TABLE SPEC MODIFY (DOMENIU
  DEFAULT 'NECOMPLETAT');
ALTER TABLE SPEC MODIFY(NUME
     CONSTRAINT NUME_NENUL NOT NULL);
```

MODIFICARE COLOANA – cont.

- În ceea ce priveşte modificările tipului de date asociat unei coloane, acestea pot fi:
  - Creşterea dimensiunii coloanelor de tip şir de caractere,
  - Creşterea dimensiunii şi preciziei datelor numerice,
  - Schimbarea tipurilor CHAR în VARCHAR2 şi reciproc doar dacă au aceeaşi dimensiune.

MODIFICARE COLOANA – cont.

- Doar în cazul în care tabela este goală sau coloana conţine doar valori nule se mai pot efectua şi operaţiile:
  - Scăderea dimensiunii coloanelor de tip şir de caractere
  - Scăderea dimensiunii şi preciziei datelor numerice,
  - Orice altă schimbare a tipului coloanei respective.
- În cazul schimbării valorii implicite pentru o coloană, noua valoare va fi folosită doar pentru inserările de linii ulterioare modificării.

5. ADAUGARE CONSTRANGERE

- Sintaxa:

```
ALTER TABLE nume_tabela
  ADD [CONSTRAINT nume] tip(coloana);
```

- Tipul constrângerii nu poate fi NOT NULL. În schimb se pot defini constrângeri de tip CHECK conţinând condiţii de acest tip.
- Exemplu: Se doreşte ca pe coloana NUME a tabelei SPEC să nu existe valori nule sau identice şi lungimea minimă să fie de 6 caractere:

EXEMPLU

```
ALTER TABLE SPEC
  ADD CONSTRAINT NUME_NENUL
  CHECK(NUME IS NOT NULL);
ALTER TABLE SPEC
  ADD CONSTRAINT NUME_UNIC
  UNIQUE(NUME);
ALTER TABLE SPEC
  ADD CONSTRAINT NUME_5
  CHECK(LENGTH(NUME)>5);
```

6. STERGERE CONSTRANGERE

- Sintaxa:

```
ALTER TABLE nume_tabela DROP PRIMARY KEY
  [CASCADE];
ALTER TABLE nume_tabela DROP
  UNIQUE(lista_coloane)
  [CASCADE];
ALTER TABLE nume_tabela DROP CONSTRAINT nume
  [CASCADE];
```

- Efectul acestor cereri este:
  - DROP PRIMARY KEY şi DROP UNIQUE specifică ştergerea constrângerii de tip cheie primară/cheie unică pentru tabela respectivă. Constrângerea poate să nu aibă un nume asociat la definire.
  - DROP CONSTRAINT specifică ştergerea unei constrângeri având asociat un nume.
  - Opţiunea CASCADE se aplică în cazul în care există constrângeri dependente şi specifică ştergerea suplimentară a acestora.

EXEMPLU

- Stergerea cheii primare a tabelei SPEC şi a constrângerilor dependente de acestea (de exemplu cea de cheie străină din STUD) se face astfel:

```
ALTER TABLE SPEC DROP PRIMARY KEY
  CASCADE;
```

7. ACTIVARE/DEZACTIVARE CONSTRANGERE

- Sintaxa: Dezactivare:

```
ALTER TABLE nume_tabela DISABLE PRIMARY KEY
  [CASCADE]
ALTER TABLE nume_tabela
  DISABLE UNIQUE(lista_coloane) [CASCADE]
ALTER TABLE nume_tabela DISABLE CONSTRAINT
  nume
    [CASCADE]
```

- Reactivare:

```
ALTER TABLE nume_tabela ENABLE PRIMARY KEY;
ALTER TABLE nume_tabela ENABLE
  UNIQUE(lista_coloane);
ALTER TABLE nume_tabela ENABLE CONSTRAINT
  nume;
```

EFECT

- DISABLE specifică dezactivarea constrângerii de integritate respective.
- Opţiunea CASCADE duce la dezactivarea suplimentară a tuturor constrângerilor dependente.

- ENABLE CONSTRAINT specifică reactivarea constrângerii respective.
- În momentul reactivării sistemul verifică dacă datele la care se referă constrângerea sunt conforme cu aceasta.
- În cazul în care nu se constată respectarea constrângerii, ea nu este activată şi se generează un mesaj de eroare. În caz de activare cu succes, constrângerile dependente nu sunt reactivate.

EXEMPLU

- Dezactivarea şi reactivarea constrângerii de cheie primară SPEC_PK pentru tabela SPEC:

```
ALTER TABLE SPEC DISABLE PRIMARY
  KEY CASCADE;
ALTER TABLE SPEC ENABLE
  CONSTRAINT SPEC_PK;
```

8. GOLIRE TABELA

- Sintaxa:

```
TRUNCATE TABLE nume_tabela [REUSE
  STORAGE];
```

- Exemplu: golirea tabelei SPEC se poate face prin cererea SQL

```
TRUNCATE TABLE SPEC;
```

- Opţiunea REUSE STORAGE este folosită pentru a specifica faptul că spaţiul ocupat de liniile şterse rămâne alocat tabelei şi poate fi folosit la inserările ulterioare în aceasta.
- În lipsa acestei opţiuni spaţiul respectiv devine disponibil, putând fi utilizat şi pentru alte tabele.

9. STERGERE TABELA

- Sintaxa:

```
DROP TABLE nume_tabela;
```

- Exemplu: ştergerea tabelei SPEC se poate face prin cererea SQL

```
DROP TABLE SPEC;
```

- Observaţie: ştergerea unei tabele este definitivă.
- O dată ştearsă ea poate fi restaurată doar din salvările bazei de date efectuate de administrator.

10. REDENUMIRE TABELA

- Sintaxa:

```
RENAME nume_vechi TO nume_nou
```

- Exemplu: redenumirea tabelei SPEC se poate face prin cererea SQL

```
RENAME SPEC TO SPECIALIZARI;
```

- Observaţie: comanda RENAME nu este specifică tabelelor ci tuturor obiectelor din baza de date.
- Cu ajutorul ei se pot redenumi de asemenea vederi, secvenţe şi sinonime.

11. COMENTARII

- Tabelele şi coloanele acestora pot avea asociat un comentariu. Acesta este un text şi se asociază astfel:

```
COMMENT ON TABLE nume_tabela IS 'text`;
COMMENT ON COLUMN nume_tabela.cume_coloana IS
  'text`;
```

- Acestea sunt stocate în dicţionarul de date în tabelele:
- USER_COL_COMMENTS şi ALL_COL_COMMENTS:
- comentariile asociate coloanelor
- USER_TAB_COMMENTS şi USER_TAB_COMMENTS
- comentariile asociate tabelelor

DICTIONARUL DE DATE

- Una dintre cerinţele fundamentale pentru ca un sistem de gestiune a bazelor de date să fie considerat cu adevărat relaţional este aceea ca datele interne ale acestuia să fie organizate şi stocate în acelaşi mod cu datele utilizatorilor şi aplicaţiilor care îl folosesc.
- În acest sens, şi sistemul Oracle îşi ţine aceste date, numite uzual *dictionarul de date al sistemului*, sub formă de tabele, iar utilizatorii care au drepturile necesare le pot accesa fie direct fie folosind o serie de vederi şi sinonime puse la dispoziţie de sistem.

DICTIONARUL DE DATE – cont.

- Există mai multe categorii de vederi prin care uzual aceste date sunt accesate, fiecare având ca şi caracteristică un prefix al numelui care defineşte categoria respectivă.
- Cele mai folosite vederi intră în categoriile definite de prefixele următoare:
  - Prefixul USER_ este comun vederilor prin care se pot accesa informaţii despre obiectele deţinute de utilizator.
  - Prefixul ALL_ este specific vederilor care conţin informaţii despre toate tabelele la care utilizatorul are acces.
  - Prefixul DBA_ este folosit pentru vederile accesibile doar utilizatorilor care au drepturi (privilegii) de administrator al bazei de date.

EXEMPLE

- Prezentarea tuturor acestor vederi nu face obiectul lucrării de faţă.
- În continuare sunt prezentate câteva dintre cele mai importante, legate mai ales de inventarul de tabele şi constrângeri de integritate existent la un moment dat în sistem:
  - USER_CATALOG
  - USER_OBJECTS
  - USER_CONSTRAINTS
  - USER_CONS_COLUMNS

USER_CATALOG

- utilizator. Are doar două coloane, prima conţinând numele şi al doilea tipul obiectului. O parte a conţinutului acestei tabele, obţinut cu cererea SQL:

```
SELECT * FROM USER_CATALOG;
```

- este următorul:

```
TABLE_NAME          TABLE_TYPE
------------------- -----------
BURSA               TABLE
EVENIMENT           TABLE
NUMERE              TABLE
SPEC                TABLE
STUD                TABLE
```

USER_OBJECTS

- Conţine numele, tipul şi alte informaţii despre obiectele deţinute de utilizator, inclusiv indecşi.
- Informaţii sumare despre conţinutul vederii se pot obţine cu cererea:

```
SELECT OBJECT_NAME, OBJECT_TYPE
FROM USER_OBJECTS;
```

- Rezultatul include rezultatul obţinut în urma cererii anterioare (USER_CATALOG).

USER_CONSTRAINTS

- Conţine date despre constrângerile definite de utilizator. Dintre coloanele sale menţionăm:
  - CONSTRAINT_NAME : Numele constrângerii
  - CONSTRAINT_TYPE: Tipul acesteia, codificat pe un caracter: P: cheie primară, U: cheie (UNIQUE), R pentru integritatea referenţială şi C pentru constrângerile CHECK (aici sunt incluse şi cele de tip NOT NULL)
  - SEARCH_CONDITION: Condiţia asociată constrângerii

EXEMPLU

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE,
       SEARCH_CONDITION
FROM USER_CONSTRAINTS;
```

Rezultat:

```
CONSTRAINT_NAME              C SEARCH_CONDITION
---------------------------- - ----------------------
BURSA_CK                     C PMIN < PMAX AND PMIN >= 0
                               AND PMAX >= 0 AND SUMA
                               BETWEEN 0 AND 500
DOM_NENUL                    U
DOM_5                        C LENGTH(NUME)>5
NUME_NENUL                   C NUME IS NOT NULL
```

USER_CONS_COLUMNS

- Conţine informaţii despre coloanele care sunt implicate în constrângerile de integritate. Cererea următoare afişează perechi (nume constrângere, nume coloană implicată) pentru constrângerea BURSA_CK:

```
SELECT CONSTRAINT_NAME, COLUMN_NAME
FROM USER_CONS_COLUMNS
WHERE CONSTRAINT_NAME = 'BURSA_CK';
```

- Rezultatul este:

```
CONSTRAINT_NAME            COLUMN_NAME
-------------------------- --------------------
BURSA_CK                   PMIN
BURSA_CK                   PMAX
BURSA_CK                   SUMA
```

CATE VEDERI SUNT?

- Există peste 2000 de vederi puse la dispoziţie de sistemul Oracle (V9) dintre care peste 200 în fiecare dintre categoriile USER_, ALL_ şi DBA_.
- O descriere completă a lor se găseşte în documentaţia aferentă sistemului.
- Lista tuturor vederilor poate fi obţinută cu cererea:

```
SELECT VIEW_NAME FROM ALL_VIEWS;
```

Sfarsitul capitolului

CREAREA TABELELOR