

Capitolul 4. Crearea tabelelor și definirea restricțiilor

Crearea structurilor fizice ale unei baze de date, asupra căreia am zăbovit în capitolul 2, constituie etapa premergătoare implementării tabelelor, tabele încadrabile în categoria structurilor logice. Această operațiune presupune nu numai specificarea numelor tabelelor și atributelor care le compun, ci și declararea restricțiilor menite a asigura integritatea și coerența informațiilor din bază. Este ceea ce propunem spre discuție în acest capitol.

4.1. Prezentarea schemei bazei de date folosite în lucrare

Baza de date în jurul căreia vor gravita majoritatea exemplelor din acest capitol și din cele următoare are o structură nu supărător de complexă. Obiectivul principal îl constituie gestionarea informațiilor legate de salarizare pentru o firmă care își plătește angajații doar pe baza orelor lucrate (în sensul că alte forme de salarizare, precum acordul direct, acordul global etc. nu sunt luate în considerare).

Cele șase tabele inițiale ale bazei sunt: PERSONAL, PONTAJE, SPORURI, RETINERI, SALARII și TRANȘE_SV, fiind reprezentate în figura 4.1. Pe parcurs, local, vom folosi și alte tabele sau vom modifica ușor structura existentă.

Tabela PERSONAL reprezintă nomenclatorul angajaților firmei. Atribute:

- `Marca` este un număr unic, secvențial, atribuit la angajare fiecărui om al muncii;
- `NumePren` – nume și prenumele;
- `Compart` – compartimentul căruia îi este afiliat angajatul curent;
- `DataSV` – data de la care se vor calcula anii de vechime, ani necesari determinării sporului de vechime; este, de multe ori, data primei angajări, dar există suficiente situații în care o persoană a plecat pentru câteva luni/ani la lucru la negru prin țările UE sau alte spețe în care `DataSV` (data pentru spor de vechime) nu coincide cu data primei angajări;
- `SalOrar` – salariul orar (curent);
- `SalOrarCO` – salariul orar pentru calculul drepturilor pentru perioada de concediu de odihnă; poate fi la același nivel, dar și diferit de cel al salariului orar;
- `Colaborator` – indică poziția de angajat curent sau temporar a persoanei respective.

Fiecare linie corespunde unui angajat, așa că atributul *Marca* este cheia primară a tabelului.

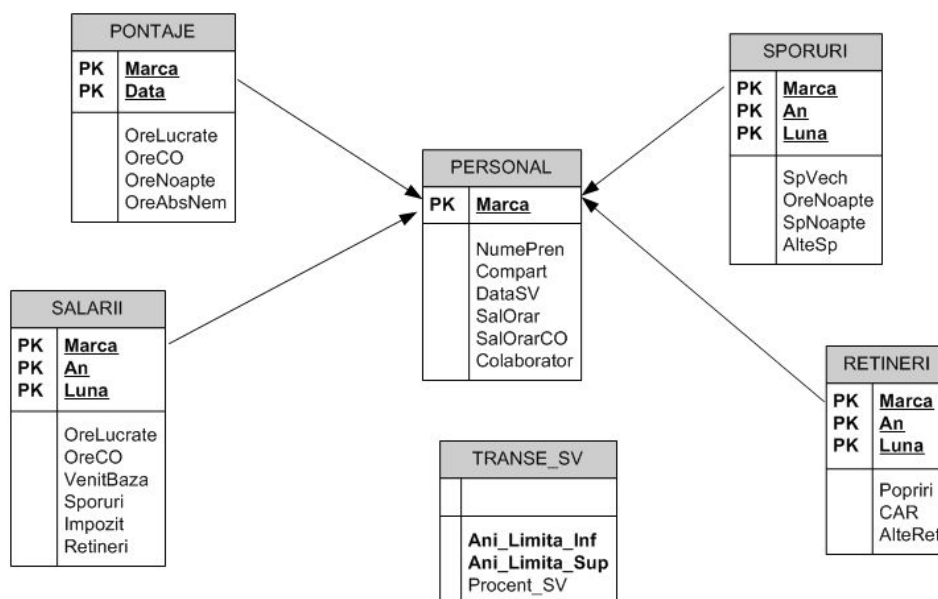


Figura 4.1. Cele șase tabele inițiale ale bazei

Tabela PONTAJE conține datele importante despre orele lucrate în fiecare zi de către un angajat. Atribute:

- *Marca* are aceeași semnificație precum atributul *PERSONAL.Marca*;
- *Data* este ziua pentru care se introduce pontajul;
- *OreLucrate* - numărul de ore lucrate în ziua curentă de angajatul cu marca curentă; valoarea implicită a acestui atribut este 8;
- *OreCO* - are valoarea 0 dacă persoana nu este în concediu în ziua curentă și 8 în caz contrar (și fericit);
- *OreNoapte* - conține câte ore, dintre cele lucrate, sunt asimilate activităților de noapte și plătite corespunzător;
- *OreAbsNem* - ore absențe nemotivate, adică numărul orelor de chiul fără avizul șefului pentru ziua respectivă.

Fiecare linie corespunde unui angajat și unei zile lucrate, prin urmare cheia primară este compusă: (*Marca*, *Data*).

Tabela SPORURI centralizează sporurile lunare ale fiecărui angajat. Atribute:

- *Marca* are aceeași semnificație precum atributul *PERSONAL.Marca*;
- *An* este anul la care se referă sporurile de pe linia curentă;
- *Luna* este luna la care se referă sporurile de pe linia curentă;

- `SpVech` – sporul de vechime corespunzător angajatului pentru anul și luna de pe linia curentă; se calculează înmulțind venitul de bază procentul de spor de vechime, procent obținut din tabela `TRANȘE_SV` pe baza numărului de ani de vechime ai angajatului curent; venitul de bază este descris la tabela `SALARII`;
- `OreNoapte` – centralizează orele lucrate în regim de noapte de un angajat pentru luna curentă;
- `SpNoapte` – sporul de noapte, calculat prin înmulțirea salariului orar cu numărul orelor de noapte, produs ponderat cu 15% (atât acordă firma în plus pentru orele lucrate noaptea).
- `AlteSp` – alte sporuri (stres, toxicitate, rușine, condiții deosebite etc.)

Fiecare linie corespunde unui angajat și unei luni (și an), cheia primară fiind combinația: (Marca, An, Luna).

Tabela `REȚINERI` centralizează reținerile lunare ale fiecărui angajat. Atribute:

- `Marca` are aceeași semnificație precum atributul `PERSONAL.Marca`;
- `An` este anul la care se referă reținerile de pe linia curentă;
- `Luna` este luna la care se referă reținerile de pe linia curentă;
- `Propriiri`, `CAR`, `AlteRet` (alte rețineri) – diverse forme de rețineri de pe statul de plată în contul unor debite, obligații etc;

Fiecare linie corespunde unui angajat și unei luni (și an), cheia primară fiind combinația: (Marca, An, Luna).

Tabela `SALARII` este cea care “culege” informații din `PONTAJE`, `SPORURI` și `REȚINERI`, fiecare linie fiind o sinteză a drepturilor și reținerilor unui angajat și lună-an date. Atributele sale sunt:

- `Marca` are aceeași semnificație precum atributul `PERSONAL.Marca`;
- `An` și `Luna` reprezintă luna pentru care se centralizează datele privind salarizarea;
- `OreLucrate` se determină prin însumarea orelor lucrate de angajatul curent în luna de referință (pe baza `OreLucrate` din `PONTAJE`);
- `OreCO` – suma orelor în care, pe luna curentă, angajatul a fost în concediu de odihnă;
- `VenitBază` – venitul de bază de determină prin însumarea produsului `SALARII.OreLucrate * PERSONAL.SalOrar` cu `SALARII.OreCO * PERSONAL.SalOrarCO`;
- `Sporuri` se determină prin expresia: `SPORURI.SpVech + SPORURI.SpNoapte + SPORURI.AlteSp`;
- `Impozit` este suma reținută și vărsată minunatului stat român, pre-amilei administrații și bravei clase politice pentru destinații pe care n-are rost să le detaliez aici.
- `Rețineri` se calculează astfel: `REȚINERI.Propriiri + REȚINERI.CAR + REȚINERI.AlteRet`;

Fiecare linie corespunde unui angajat și unei luni (și an), cheia primară fiind combinația: (Marca, An, Luna).

Tabela TRANSE_SV este izolată, rolul său fiind decisiv în calcularea procentului sporului de vechime acordat fiecărui angajat. Procentul depinde de tranșa de ani de vechime în care se încadrează angajatul în prima zi a lunii de referință. De exemplu, între 0 și trei ani de vechime, procentul este zero, între trei și șase ani este 5% (probabil). Atributele Ani_Limita_Inf, Ani_Limita_Sup și Procent_SV conțin cele trei elemente ale fiecărei tranșe.

4.2. Principalele tipuri de date Oracle

Ca o condiție prealabilă a creării unei baze de date, este mai mult decât necesar să se cunoască ce tipuri de date pot fi gestionate prin SGBD-ul folosit. În orice server de baze de date, pentru a crea o tabelă trebuie specificate tipurile de date folosite la stocarea fiecărui câmp. De obicei, în Oracle tipurile de date sunt împărțite astfel:

- Tipuri predefinite:
 - scalare;
 - colecții;
 - referințe;
- Tipuri definite de utilizator.

4.2.1. Tipurile scalare predefinite

Tabelul 4.1 redă într-o formă schematizată principalele tipuri scalare Oracle, alături de o descriere sumară a caracteristicilor fiecăruia.

Tabelul 4.1 Principalele tipuri scalare de date în Oracle

Tip	Descriere	Comentarii
CHAR (n)	Șir de caractere cu lungime fixă egală cu n octeți (bytes)	Lungimea fixă presupune completarea automată cu spații a valorilor cu lungime mai mică; lungimea maximă admisă este de 2000 bytes. Înainte de specificarea lungimii trebuie avut în vedere setul de caractere utilizat (cu reprezentare pe un byte sau multibyte)
VARCHAR2 (n)	Șir de caractere cu lungime variabilă (maxim n bytes)	O valoare de lungime mai mică a atributului la un moment dat nu va fi completată cu spații adiționale. Lungimea maximă este 4000 bytes

NUMBER (p, s)	Date numerice de lungime variabilă	p=precizia (<i>lungimea maximă a numărului</i>) iar s=scala (numărul de poziții zecimale rezervate, incluse în lungimea maximă). Spre exemplu, pentru coloana Salariu Number(5,2) valoarea 100.55 este validă în timp ce 1000.5 sau 1.222 este invalidă. Precizia poate fi de maxim 38 cifre
DATE	Date calendaristice și temporale	Lungime fixă de 7 bytes (vezi și comentariile ce urmează acestui tabel)
CLOB	Character Large Object	Poate stoca șiruri de caractere până la 4 gigabytes. Un atribut de acest tip nu poate fi folosit în subinterogări, funcții sau clauza WHERE a unei fraze SELECT SQL
LONG	Date de tip șir de caractere de lungime variabilă	Poate stoca șiruri de caractere până la 2 gigabytes. Aceleași restricții ca și în cazul tipului CLOB. Documentația Oracle recomandă destul de insistent renunțarea la acest tip în favoarea CLOB
BLOB	Date binare nestructurate	Maxim 4 gigabytes.
BFILE	Date binare stocate într-un fișer extern	Pointer spre un fișer de pe disc cu dimensiunea maximă 4 gigabytes.
ROWID	Date în format binar reprezentând adresa fizică pe disc a înregistrării	Acest tip este specific pseudocoloanei ROWID ce este asociată în mod implicit fiecărei table a bazei de date. Valorile identifică în mod unic fiecare înregistrare în baza de date Oracle.

Ar mai trebuie adăugate câteva chestii legate de tipurile scalare, și anume:

- specificatorul pentru o constantă de tip șir de caractere este apostroful și nu ghilimele;
- pentru datele calendaristice formatul implicit Oracle este DD-MON-RR.

Spre exemplu, pornind de la o definiție a unei table de maniera următoare:

```
CREATE TABLE test (tdata DATE);
```

o instrucțiune INSERT SQL validă ar fi (observați utilizarea apostrofurilor chiar dacă tipul este DATE):

```
INSERT INTO test VALUES ( '12-FEB-98' );
```

De cele mai multe ori, însă, se recurge însă la funcții de conversie:

```
INSERT INTO test VALUES (TO_DATE('12/02/1998','DD/MM/YYYY')
```

Pentru detalii legate de funcțiile de conversie și alte funcții de lucru cu date numerice, șiruri de caractere și calendaristice – vezi capitolul 6.

4.2.2. Tipuri abstracte definite de utilizator.

Oracle oferă și suport pentru definirea unor tipuri abstracte de date, ca extensie spre modelul obiectual. Astfel de tipuri se declară de maniera următoare:

```
CREATE TYPE dentip AS
  OBJECT (atrib_1 Tip, atrib_2 Tip ...)
```

Tipurile abstracte pot fi apoi utilizate ca tipuri ale atributelor din tabele. Astfel, vom crea tipul Adresa_TY pentru a defini adresa completă a persoanelor ce locuiesc într-o anumită localitate. Definim apoi o tabelă, LOCUITORI, ce va conține persoanele ce domiciliază într-o localitate, adresa lor fiind specificată prin intermediul tipului nou creat.

```
CREATE TYPE Adresa_TY AS OBJECT (
  Strada VARCHAR2(30),
  Numar VARCHAR2(10),
  Bloc VARCHAR2(10),
  Ap INTEGER
);

CREATE TABLE Locuitori (
  Cnp INTEGER,
  Nume VARCHAR2(30),
  Adresa ADRESA_TY);
```

La inserarea înregistrărilor în această tabelă, pentru atributul Adresa vom utiliza comanda INSERT în următorul format (valorile fiecărui atribut component al tipului se scriu între paranteze):

```
INSERT INTO locuitori VALUES (11111, 'Fane C.',
  ADRESA_TY('CUG','22 BIS','A5', 13));
INSERT INTO locuitori VALUES (11112, 'Mandache V.' ,
  ADRESA_TY('CUG','22 BIS','A5', 13));
INSERT INTO locuitori VALUES (11113, 'Croitoru P.' ,
  ADRESA_TY('AL. T. Neculai','112','B1', 13));
```

Extragerea de informații din tabele ce folosesc un tip, precum și modificarea valorilor presupun furnizarea adresei complete, conform schemei/ierarhiei (vezi figura 4.2): AliasTabelă.AtributTabelă.AtributTip

```
SQL> SELECT nume, L.Adresa.Strada FROM locuitori L WHERE cnp=11111;
```

NUME	ADRESA.STRADA
Fane C.	CUG

Figura 4.2. Interogarea atributelor definite pe baza unui tip-utilizator

Atributele definite în tipul abstract pot fi incluse și în clauza WHERE a unei fraze SELECT SQL (figura 4.3).

```
SQL> SELECT nume FROM locuitori L WHERE L.Adresa.Strada='CUG' ;
```

NUME
Fane C.
Mandache U.

```
SQL> UPDATE locuitori L SET L.Adresa.Strada='G.Enescu' WHERE L.Adresa.Strada='CUG' ;
```

2 rows updated.

```
SQL> SELECT * FROM locuitori;
```

CNP	NUME
11111	Fane C.
11112	Mandache U.
11113	Croitoru P.

```
ADRESA(STRADA, NUMAR, BLOC, AP)
```

CNP	NUME	ADRESA(STRADA, NUMAR, BLOC, AP)
11111	Fane C.	ADRESA_TV('G. Enescu', '22 BIS', 'A5', 13)
11112	Mandache U.	ADRESA_TV('G. Enescu', '22 BIS', 'A5', 13)
11113	Croitoru P.	ADRESA_TV('AL. T. Neculai', '112', 'B1', 13)

Figura 4.3. Modificarea valorilor stocate în atributele tipurilor abstracte și interogări pe baza acestora.

Comanda dedicată ștergerii unui tip este DROP TYPE DenumireTip, iar cea de modificare este: ALTER TYPE DenumireTip .

Atenție! Este obligatorie utilizarea unui alias (sinonim) pentru reușita interogării și actualizării atributelor definite în tipul abstract de date; în caz contrar vom obține un mesaj de eroare ORA-00904: invalid column name. *Un tip utilizat de o tabelă nu poate fi șters!*

4.2.3. Colecții

Colecțiile sunt seturi de date ce pot fi tratate ca parte a unei singure înregistrări dintr-o tabelă. Există două astfel de tipuri de date: vectori cu marime variabilă

(varying arrays) și tabele încapsulate (nested tables). Un vector cu marime variabilă se declară astfel:

```
CREATE OR REPLACE TYPE <Tip> AS VARRAY(12) of <TipElemente>
```

unde `TipElemente` poate fi scalar sau tip abstract. Ulterior, tipul astfel declarat poate fi utilizat la declararea atributelor unei tabele. Vom considera o tabelă `SALARIAȚI` al cărei atribut `ZileLucrate` va stoca în fiecare înregistrare un vector de 12 elemente pentru numărul zilelor lucrate de salariatul respectiv în fiecare lună a unui an.

Listing 4.1 Crearea unui vector cu mărime variabilă și utilizarea lui într-o tabelă

```
CREATE OR REPLACE TYPE Zile_TY AS VARRAY(12) OF INTEGER
/
CREATE TABLE salariati (
    Nume VARCHAR2(50),
    zilelucr ZILE_TY);

INSERT INTO salariati VALUES ('Atomei Maria', Zile_TY(22,21,20,22,23));
INSERT INTO salariati VALUES ('Agheorgiesei V', Zile_TY(null, null, 18,21,20,22,23));
```

Extragerea datelor într-o formă lizibilă necesită utilizarea funcției `TABLE()` ce preia ca argument atributul de tip vector și generează, ca rezultat al interogării, câte o nouă linie pentru fiecare element căruia i s-a atribuit o valoare (chiar și NULL). Elementele neinițializate sunt ignorate. Figura 4.4 prezintă interogarea tabelului, mai întâi după formatul standard, apoi utilizând funcția de mai sus.

```
SQL> COLUMN zilelucr FORMAT A40
SQL> SELECT * FROM salariati;
```

NUME	ZILELUCR
Atomei Maria	ZILE_TY(22, 21, 20, 22, 23)
Agheorgiesei V	ZILE_TY(NULL, NULL, 18, 21, 20, 22, 23)

```
SQL> SELECT nume, A.* FROM salariati, TABLE(salariati.zilelucr) A;
```

NUME	COLUMN_VALUE
Atomei Maria	22
Atomei Maria	21
Atomei Maria	20
Atomei Maria	22
Atomei Maria	23
Agheorgiesei V	
Agheorgiesei V	
Agheorgiesei V	18
Agheorgiesei V	21
Agheorgiesei V	20
Agheorgiesei V	22
Agheorgiesei V	23

```
12 rows selected.
SQL>
```

Figura 4.4 Utilizarea funcției `TABLE()` pentru interogarea colecțiilor

Deși un pic mai complicat, un vector poate fi interogată și numai după anumite valori (apelăm la o subinterogare în clauza `FROM`, după cum urmează (vezi figura 4.5):

```
SQL> SELECT nume, nrzile FROM
2      (SELECT nume, A.COLUMN_VALUE as NrZile FROM salariati, TABLE (salariati.zilelucr) A) B
3      WHERE B.nrzile=18;
```

NUME	NRZILE
Agheorgiesei U	18

```
SQL> |
```

Figura 4.5 Interogare după un anumit element al vectorului

O limită a acestui tip de colecție ține de imposibilitatea adăugării sau actualizării valorilor din vector, altfel decât prin înlocuirea întregului vector cu altul nou:

```
UPDATE salariati
SET zilelucr = Zile_TY(20,22,18,22,20,22,20,22,20,21,22,22)
WHERE nume = 'Atomei Maria';
```

Tablourile (tabelele) încapsulate reprezintă un alt tip de colecție care, spre deosebire de vectori, poate stoca un număr nelimitat de elemente și oferă posibilitatea actualizării directe a unui anumit element. Așa cum sugerează și numele, un tablou încapsulat presupune existența unei tabele sub forma unui atribut al altei tabele. Colecția se declară astfel:

```
CREATE OR REPLACE TYPE <tip> AS TABLE OF <TipElemente>
```

Exemplificăm utilizând un tip abstract de date și anume `Salariat_TY` care păstrează informații despre un angajat. De asemenea vom defini în tabela `COMPARTIMENT` un atribut de tip tablou încapsulat care va stoca elemente de tip angajat (`Salariat_TY`) pentru fiecare compartiment în parte.

Listing 4.2 Tabelă încapsulată definită pe baza unui tip abstract și utilizată ca atribut al unei tabele clasice

```
CREATE TYPE Salariat_TY AS OBJECT (
    Nume VARCHAR2(30),
    Salariu NUMBER(16,2)
)
/
CREATE TYPE Salariati_NT AS TABLE OF Salariat_TY
/
CREATE TABLE compartiment (
    DenComp VARCHAR2(30),
    Salariati SALARIATI_NT
) NESTED TABLE Salariati STORE AS salariati_nt_tab;
```

În listingul de mai sus, la definirea tabelului COMPARTIMENT vom observa o extensie a instrucțiunii CREATE TABLE și anume :

```
NESTED TABLE <NumeAtribut> STORE AS <NumeTabelaDeStocare>
```

Această extensie se datorează faptului că elementele atributului Salariati sunt stocate sub forma unei tabeli clasice (vezi figura 4.6), complet separată, cu numele precizat în clauza STORE AS. Această tabelă nu poate fi accesată direct.

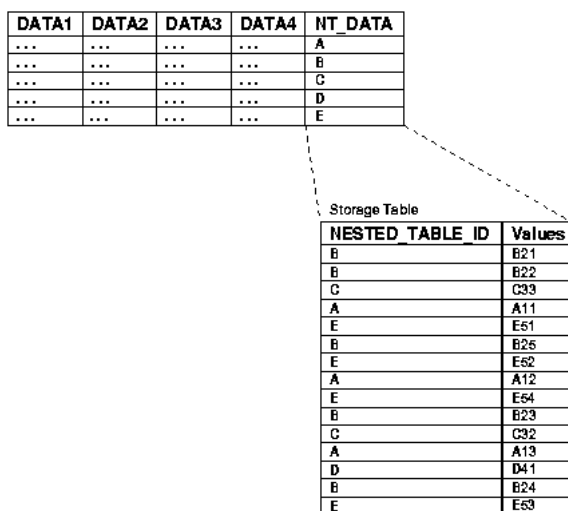


Figura 4.6. Modul de stocare al valorilor unei tabeli încapsulate¹

Adăugarea unor înregistrări noi în tabela Compartiment se realizează folosind notația:

```
NumeTipTabelaIncapsulata(NumeTipElement(valAtrib1, ...))
```

Ca urmare, adăugarea compartimentului Financiar cu doi salariați se va realiza astfel:

```
INSERT INTO compartiment VALUES ( 'Financiar',
    SALARIATI_NT (
        SALARIAT_TY('Preda T.',2000000) ,
        SALARIAT_TY('Nistor A.',2300000)
    )
);
```

¹ Preluat din documentația Oracle

Extragerea datelor presupune utilizarea funcției THE cu formatul:

```
THE(Select <atributTipTabela> from <tabelaParinte>)
```

ce are rolul de a "aplatiza" înregistrările din tabela încapsulată și a le prezenta sub formă de linii ca rezultat al interogării tablei părinte (vezi figura 4.7).

```
SQL> SELECT NT.* FROM THE(SELECT Salariati FROM compartiment WHERE dencomp='Financiar') NT;
```

NUME	SALARIU
Preda T.	2000000
Nistor A.	2300000

```
SQL> |
```

Figura 4.7 Interogarea tabelor ce conțin atribute de tip tabela încapsulată

În final, figura 4.8 prezintă un exemplu de interogare după valorile atributelor tipului abstract (Salariat_TY) stocat în tabela încapsulată și un exemplu de actualizare a liniilor din tabela încapsulată.

```
SQL> SELECT NT.Salariu FROM THE(SELECT Salariati FROM compartiment WHERE dencomp='Financiar') NT
2 WHERE NT.Nume='Preda T.';
```

SALARIU
2000000

```
SQL> ed
Wrote file afiedt.buf

1 UPDATE THE(SELECT Salariati FROM compartiment WHERE dencomp='Financiar') NT SET NT.Salariu=0
2* WHERE NT.Nume='Preda T.'
3 /

1 row updated.

SQL> SELECT NT.* FROM THE(SELECT Salariati FROM compartiment WHERE dencomp='Financiar') NT;
```

NUME	SALARIU
Preda T.	0
Nistor A.	2300000

```
SQL> |
```

Figura 4.8. Interogarea și actualizarea tabelor încapsulate

Pentru a adăuga o nouă înregistrare în tabela încapsulată utilizăm aceeași funcție THE (), astfel:

```
INSERT INTO
  THE(SELECT salariati
    FROM compartiment WHERE dencomp='Financiar') NT
VALUES (SALARIAT_TY('Amariei V.',2000000));
```

4.3. Formatul simplu al comenzii SQL CREATE TABLE. Valori implicite

Pentru crearea unei tabele și declararea atributelor acesteia, fără nici o referire la restricții și valori implicite, comanda `CREATE TABLE` are, în cazul tablei `PERSONAL`, următoarea formă:

```
CREATE TABLE personal (  
    marca INTEGER  
    , numepren VARCHAR2(40)  
    , compart VARCHAR2(5)  
    , dataSV DATE  
    , salorar NUMBER(16,2)  
    , salorarco NUMBER(16,2)  
    , colaborator CHAR(1)  
    ) ;
```

Dat fiind că `marca` este un număr atribuit secvențial la angajarea fiecărui om al muncii, pentru atributul `Marca` s-a ales tipul `INTEGER`. Salariile orare curente și pentru calculul indemnizației pentru concediul de odihnă ar fi trebuit să fie tot întregi. Cum, însă, discuțiile despre leul greu revin periodic în actualitate, ne-am gândit că, prin tăierea celor patru de zero, o să fie nevoie și de zecimale, iar expresia “De doi bani” nu o să mai aibă conotații atât de derizorii. Este motivul principal pentru care tipul celor două atribute, `SalOrar` și `SalOrarCO`, a fost ales `NUMBER`.

SQL> describe user_tables		
Name	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
IOT_NAME		VARCHAR2(30)
PCT_FREE		NUMBER
PCT_USED		NUMBER
INIT_TRANS		NUMBER
MAX_TRANS		NUMBER
INITIAL_EXTENT		NUMBER
NEXT_EXTENT		NUMBER
MIN_EXTENTS		NUMBER
MAX_EXTENTS		NUMBER
PCT_INCREASE		NUMBER
FREELISTS		NUMBER
FREELIST_GROUPS		NUMBER
LOGGING		VARCHAR2(3)
BACKED_UP		VARCHAR2(1)
NUM_ROWS		NUMBER
BLOCKS		NUMBER
EMPTY_BLOCKS		NUMBER
AVG_SPACE		NUMBER
CHAIN_CNT		NUMBER
AVG_ROW_LEN		NUMBER
AVG_SPACE_FREELIST_BLOCKS		NUMBER
NUM_FREELIST_BLOCKS		NUMBER
DEGREE		VARCHAR2(10)
INSTANCES		VARCHAR2(10)
CACHE		VARCHAR2(5)
TABLE_LOCK		VARCHAR2(8)
SAMPLE_SIZE		NUMBER
LAST_ANALYZED		DATE
PARTITIONED		VARCHAR2(3)
IOT_TYPE		VARCHAR2(12)
TEMPORARY		VARCHAR2(1)
SECONDARY		VARCHAR2(1)
NESTED		VARCHAR2(3)
BUFFER_POOL		VARCHAR2(7)
ROW_MOVEMENT		VARCHAR2(8)
GLOBAL_STATS		VARCHAR2(3)
USER_STATS		VARCHAR2(3)
DURATION		VARCHAR2(15)
SKIP_CORRUPT		VARCHAR2(8)
MONITORING		VARCHAR2(3)
CLUSTER_OWNER		VARCHAR2(30)
DEPENDENCIES		VARCHAR2(8)

Figura 4.9 Informații (prea) generoase despre tabelele bazei

Tipul `DATE` este inevitabil în cazul atributului folosit pentru a determina anii de vechime (în muncă) ai angajatului, însă pentru colaborator ar fi fost mai normal să alegem un tip `LOGICAL` sau `BOOLEAN`. Curios, Oracle nu poate gestiona în tabele acest gen de date, așa că am recurs la tipul caracter – `CHAR(1)`. `NumePren` și `Compart` prezintă variații mari de lungime, de la angajat la angajat, așa că au fost declarate `VARCHAR2`.

În dicționarul de date Oracle, o parte dintre informații se găsesc în tabela sistem `USER_TABLES`, cu structura din figura 4.9, obținută în SQL*Plus prin comanda `DESCRIBE`. Pentru moment, singura informație care ne va interesa este numele tablei. Restul după ce vom ști ceva mai multe despre administrarea bazelor de date. Putem afla numele tuturor tabelor din bază prin interogarea:

```
SELECT table_name
FROM user_tables ;
```

O altă tabelă virtuală a dicționarului din care pot fi extrase numele tabelelor este USER_OBJECTS:

```
SELECT object_name
FROM user_objects
WHERE object_type = 'TABLE' ;
```

Pentru a afla detalii despre atributele unei tabeli, este necesară consultarea unei alte tabeli a dicționarului de date - USER_TAB_COLUMNS – vezi figura 4.10.

SQL> desc user_tab_columns		
Name	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
DATA_TYPE		VARCHAR2(106)
DATA_TYPE_MOD		VARCHAR2(3)
DATA_TYPE_OWNER		VARCHAR2(30)
DATA_LENGTH	NOT NULL	NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
NULLABLE		VARCHAR2(1)
COLUMN_ID		NUMBER
DEFAULT_LENGTH		NUMBER
DATA_DEFAULT		LONG
NUM_DISTINCT		NUMBER
LOW_VALUE		RAW(32)
HIGH_VALUE		RAW(32)
DENSITY		NUMBER
NUM_NULLS		NUMBER
NUM_BUCKETS		NUMBER
LAST_ANALYZED		DATE
SAMPLE_SIZE		NUMBER
CHARACTER_SET_NAME		VARCHAR2(44)
CHAR_COL_DECL_LENGTH		NUMBER
GLOBAL_STATS		VARCHAR2(3)
USER_STATS		VARCHAR2(3)
AVG_COL_LEN		NUMBER
CHAR_LENGTH		NUMBER
CHAR_USED		VARCHAR2(1)
US00_FMT_IMAGE		VARCHAR2(3)
DATA_UPGRADED		VARCHAR2(3)

Figura 4.10. Detalii despre atributele tabelilor

Și în acest caz, datele furnizate sunt cu mult peste ceea ce putem gestiona în acest moment. Cel mai frecvent, ne interesează numele, tipul, lungimea, precizia (pentru atribute numerice) și numărul de poziții fracționare (pentru atributele numerice) ale câmpurilor, ceea ce, în cazul tabelii PERSONAL, presupune execuția unei frazei `SELECT` și afișarea unui rezultat ca în figura 4.11.

```

SQL> SELECT column_name, data_type, data_length, data_precision, data_scale
2   FROM user_tab_columns
3   WHERE table_name = 'PERSONAL'
4   /

```

COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE
MARCA	NUMBER	22		0
NUMEPREN	VARCHAR2	40		
COMPART	VARCHAR2	5		
DATASV	DATE	7		
SALORAR	NUMBER	22	16	2
SALORARCO	NUMBER	22	16	2
COLABORATOR	CHAR	1		

Figura 4.11. Obținerea de informații despre atributele tabeli PERSONAL

În conformitate cu standardele SQL, declararea valorilor implicite pe care le poate lua un atribut atunci când, la inserarea unei linii, nu se specifică valoarea atributului respectiv, presupune utilizarea clauzei `DEFAULT`.

```

CREATE TABLE personal (
marca NUMBER(5)
, numepren VARCHAR2(40)
, compart VARCHAR2(5) DEFAULT 'PROD'
, dataSV DATE DEFAULT SYSDATE
, salorar NUMBER(16,2) DEFAULT 45000
, salorarco NUMBER(16,2) DEFAULT 40000
, colaborator CHAR(1) DEFAULT 'N'
) ;

```

Executată după comanda de mai sus, comanda `INSERT` următoare:

```
INSERT INTO personal (marca, numepren) VALUES (101, 'A1') ;
```

va adăuga în tabelă înregistrarea din figura 4.12.

MARCA	NUMEPREN	COMPART	DATASV	SALORAR	SALORARCO	COLABORATOR
101	A1	PROD	1/24/2003 12:35:57 PM	45000	40000	N

Figura 4.12. Înregistrare cu (multe) valori implicite

Spre deosebire de versiunile precedente, Oracle 9i permite declararea datei și orei curente drept valoare implicită. Nu pot fi, însă, declarate în clauza `DEFAULT` nume de funcții stocate (facilitate existentă, spre exemplu, în Visual FoxPro) sau fraze `SELECT`. Atunci când valorile implicite trebuie calculate prin expresii mai complexe, se folosesc declanșatoarele (de tip `AFTER INSERT ROW`), după cum vom vedea într-un alt capitol.

Se mai cuvine de amintit, în treacăt, o altă comandă, cel puțin la fel de îndrăgită precum `CREATE TABLE`, și anume `DROP TABLE`. După cum bine îi zice numele,

`DROP TABLE` șterge o tabelă din baza de date. Paragrafele următoare vor fi presărate cu multe comenzi de ștergere, în vederea re-creării tabelelor.

De asemenea, o tabelă Oracle poate fi creată și pe baza unei interogări SQL. Pentru exemplificare, creăm o copie a tabelului `personal` cu un nume destul de sugestiv – `COPIE_PERSONAL`:

```
CREATE TABLE copie_personal AS
SELECT * FROM personal
```

Noua tabelă va prelua toate atributele din `PERSONAL`, nu însă și restricțiile (cu excepția valorilor nenule).

Deși nu se leagă neapărat cu subiectul creării de tabele, merită amintit că în Oracle 9i o tabelă fi redenumită. Astfel, dacă vrem să schimbăm numele proaspetei tabele din `COPIE_PERSONAL` în `PERSONAL_IAN2003`, putem lansa comanda `RENAME` astfel:

```
RENAME copie_personal TO personal_ian2003
```

Tot în categoria comezilor DDL (Data Definition Language) se încadrează și `TRUNCATE TABLE`. La prima vedere, este curioasă încadrarea acestor comenzi în categoria DDL, câtă vreme sarcina principală a sa este ștergerea tuturor liniilor (din acest punct de vedere, `DELETE FROM tabelă` are același efect). Spre deosebire de `DELETE FROM`, `TRUNCATE` eliberează și spațiul ocupat de înregistrările șterse. Sporul de productivitate are și un preț: `TRUNCATE` nu poate fi folosită într-o tranzacție, așa încât este cu neputință de recuperat înregistrările tabelei. În plus, nici eventualele declanșatoare de ștergere nu sunt lansate.

4.4. Valori nenule: clauza NOT NULL

Începem detalierea restricțiilor definibile în Oracle cu una dintre cele mai simple clauze, `NOT NULL`. Pentru unele atribute se poate institui obligativitatea precizării unei valori, atât la inserare, cât și la modificare. Conform teoriei relaționale, câmpurile ce alcătuiesc cheia primară nu trebuie să prezinte valori nule. Pentru aceasta, la crearea tabelului sau la modificarea structurii sale, se folosește opțiunea `NOT NULL` pentru atributul respectiv.

Rescriem comanda de creare a tabelului `PERSONAL`, interzicând apariția de valori nule pentru atributele `Marca`, `Numepren`, `Compart` și `Colaborator`:

```
CREATE TABLE personal (
    marca NUMBER(5) NOT NULL
    , numepren VARCHAR2(40) NOT NULL
    , compart VARCHAR2(5) DEFAULT 'PROD' NOT NULL
    , dataSV DATE DEFAULT SYSDATE
    , salorar NUMBER(16,2) DEFAULT 45000
    , salorarco NUMBER(16,2) DEFAULT 40000
```



```
, colaborator CHAR(1) DEFAULT 'N' NOT NULL
) ;
```

Inserarea unei linii în care nu se precizează valorilor atributelor nenule se soldează cu un mesaj de eroare de genul celui din figura 4.13.

```
SQL> /
INSERT INTO personal (numepren) VALUES ('A2')
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("FOTACHEM"."PERSONAL"."MARCA")
```

Figura 4.13. Încălcarea restricției de nenulitate

Aflarea posibilității sau imposibilității primirii de valori nule pentru fiecare atribut al tabelului PERSONAL presupune execuția interogării:

```
SELECT column_name, nullable
FROM user_tab_columns
WHERE table_name = 'PERSONAL' ;
```

4.5. Chei primare și alternative: clauzele PRIMARY KEY și UNIQUE

Cheia primară a unei relații, cea care identifică fără ambiguitate oricare linie a tabelului, este declarată prin clauza `PRIMARY KEY`, plasată fie imediat, după atributul cheie, fie după descrierea ultimului atribut al tabelului. A doua variantă este întrebuințată cu precădere atunci când cheia primară a tabelului este compusă. Pentru attributele de tip cheie candidat se poate folosi clauza `UNIQUE` care va asigura respectarea unicității valorilor. Scriptul din listing 4.3 re-crează tablele PERSONAL și PONTAJE. Tabela PERSONAL are drept cheie primară atributul Marca, în timp ce cheia primară a tabelului PONTAJE este una compusă: (Marca, Data).

Listing 4.3. Declarația cheilor primare și alternative

```
DROP TABLE pontaje ;
DROP TABLE personal ;

CREATE TABLE personal (
    marca INTEGER NOT NULL PRIMARY KEY
    , numepren VARCHAR2(40) NOT NULL UNIQUE
    , compart VARCHAR2(5) DEFAULT 'PROD' NOT NULL
    , dataSV DATE DEFAULT SYSDATE
    , salorar NUMBER(16,2) DEFAULT 45000
    , salorarco NUMBER(16,2) DEFAULT 40000
    , colaborator CHAR(1) DEFAULT 'N' NOT NULL
);

CREATE TABLE pontaje (
    marca INTEGER NOT NULL
```

```

, data DATE DEFAULT SYSDATE NOT NULL
, orelucrate NUMBER(2) DEFAULT 8
, oreco NUMBER(2) DEFAULT 0
, orenoapte NUMBER(2) DEFAULT 0
, oreabsnem NUMBER(2) DEFAULT 0
, PRIMARY KEY (marca,data)
);

```

Ca și cheia primară, o cheie alternativă poate identifica o linie a tabelului din care face parte, altfel spus, nu există două înregistrări în tabelă în care valoarea atributului (atributelor) care o alcătuiește (alcătuiesc) să se repete. De multe ori, o tabelă poate avea mai multe chei candidate. Administratorul bazei alege numai una drept cheie primară, iar celelalte devin chei alternative. Alegerea se bazează pe criterii, precum: compoziție minimală, ușurință în folosire, lungime, stabilitate în timp, în timp ce altele, precum pile, relații, trafic de influență sunt mult mai puțin frecvente decât în viața de zi cu zi. Rolul de cheie alternativă îl are, cel puțin deocamdată, atributul `NumePren` în `PERSONAL`.

Dicționarul bazei păstrează informații despre restricții într-o tabelă virtuală specială numită `USER_CONSTRAINTS` a cărei structură vizualizată în `SQL*Plus` se prezintă ca în figura 4.14.

SQL> desc user_constraints		
Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
CONSTRAINT_TYPE		VARCHAR2(1)
TABLE_NAME	NOT NULL	VARCHAR2(30)
SEARCH_CONDITION		LONG
R_OWNER		VARCHAR2(30)
R_CONSTRAINT_NAME		VARCHAR2(30)
DELETE_RULE		VARCHAR2(9)
STATUS		VARCHAR2(8)
DEFERRABLE		VARCHAR2(14)
DEFERRED		VARCHAR2(9)
VALIDATED		VARCHAR2(13)
GENERATED		VARCHAR2(14)
BAD		VARCHAR2(3)
RELY		VARCHAR2(4)
LAST_CHANGE		DATE
INDEX_OWNER		VARCHAR2(30)
INDEX_NAME		VARCHAR2(30)
INVALID		VARCHAR2(7)
VIEW_RELATED		VARCHAR2(14)

Figura 4.14. Detalii despre restricții

Pentru a afla dacă o tabelă are declarată cheia primară și/sau chei alternative, se poate recurge la atributul `constraint_type` care, pentru chei primare, are valoarea 'P', iar pentru chei alternative (unice) 'U'. Pentru a ne referi la o restricție anume, ar trebui să-i știm numele (`constraint_name`). Folosind scriptul de mai sus, fiecare restricție are un nume-sistem, generat automat, cum ar fi `SYS_C003200`. La un număr mai mare de tabele, atribute și restricții, este foarte dificil de lucrat cu aceste nume criptice, așa că mulți profesioniști folosesc clauza `CONSTRAINT`

pentru a atribui un nume explicit fiecărei restricții. Pentru exemplificare, listingul 4.4 este o nouă prezentare a precedentului, dar cu nume sugestive pentru fiecare restricție.

Listing 4.4. “Botezarea” restricțiilor

```

DROP TABLE pontaje ;
DROP TABLE personal ;

CREATE TABLE personal (
    marca INTEGER
        CONSTRAINT pk_personal PRIMARY KEY
        CONSTRAINT nn_personal_marca NOT NULL
    , numepren VARCHAR2(40)
        CONSTRAINT nn_personal_numepren NOT NULL
        CONSTRAINT un_personal_nuempren UNIQUE
    , compart VARCHAR2(5) DEFAULT 'PROD'
    , dataSV DATE DEFAULT SYSDATE
        CONSTRAINT nn_personal_dataSV NOT NULL
    , salorar NUMBER(16,2) DEFAULT 45000
    , salorarco NUMBER(16,2) DEFAULT 40000
    , colaborator CHAR(1) DEFAULT 'N'
        CONSTRAINT nn_personal_colaborator NOT NULL
);

CREATE TABLE pontaje (
    marca INTEGER
        CONSTRAINT nn_pontaje_marca NOT NULL
    , data DATE DEFAULT SYSDATE
        CONSTRAINT nn_pontaje_data NOT NULL
    , orelucrate NUMBER(2) DEFAULT 8
    , oreco NUMBER(2) DEFAULT 0
    , orenoapte NUMBER(2) DEFAULT 0
    , oreabsnem NUMBER(2) DEFAULT 0
    , CONSTRAINT pk_pontaje PRIMARY KEY (marca,data)
);

```

Ne ușurăm lucrul folosind o notație “pseudo-ungurească” (după modelul programării vizuale), adică prefixând numele fiecărei restricții cu:

- pk_ (PRIMARY KEY) pentru cheile primare;
- un_ (UNIQUE) pentru cheile alternative;
- nn_ (NOT NULL) pentru attributele obligatorii (ce nu pot avea valori nule);
- ck_ (CHECK) pentru reguli de validare la nivel de atribut sau înregistrare ;
- fk_ (FOREIGN KEY) pentru cheile străine.

Revenind la dicționarul de date, tabela USER_CONSTRAINTS oferă informații cuprinzătoare cu privire la restricțiile definite pe fiecare tabelă din bază. Pentru a afla attributele ce fac parte din cheile primare sau cele alternative, avem nevoie de o altă tabelă a dicționarului, USER_CONS_COLUMNS, a cărei structură afișată în SQL*Plus este cea din figura 4.15.

```
SQL> desc user_cons_columns
```

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
POSITION		NUMBER

Figura 4.15. Atributele incluse în restricții

Oprindu-ne la tabela PERSONAL, aflarea atributelor declarate în toate restricțiile este posibilă prin fraza SELECT:

```
SELECT constraint_name, column_name, position
FROM user_cons_columns
WHERE table_name = 'PERSONAL'
```

rezultatul (în SQL*Plus) fiind cel din figura 4.16. Coloana Position prezintă interes la restricțiile de unicitate care au cheia compusă, și indică poziția fiecărui atribut în cadrul cheii (indexului).

CONSTRAINT_NAME	COLUMN_NAME	POSITION
NN_PERSONAL_COLABORATOR	COLABORATOR	
NN_PERSONAL_DATASU	COMPART	
NN_PERSONAL_MARCA	MARCA	
NN_PERSONAL_NUMEPREN	NUMEPREN	
PK_PERSONAL	MARCA	1
UN_PERSONAL_NUEMPREN	NUMEPREN	1

Figura 4.16. Restricții și atribute pentru tabela PERSONAL

Astfel, dacă dorim să aflăm componența cheii primare a tabeli PONTAJE, al cărei nume îl cunoaștem - PK_PONTAJE - putem recurge la interogarea următoare, al cărei rezultat este cel din figura 4.17:

```
SELECT constraint_name, column_name, position
FROM user_cons_columns
WHERE constraint_name = 'PK_PONTAJE'
```

CONSTRAINT_NAME	COLUMN_NAME	POSITION
PK_PONTAJE	MARCA	1
PK_PONTAJE	DATA	2

Figura 4.17. Componența cheii primare a tabeli PONTAJE

La încălcarea restricțiilor de cheie primară și de unicitate, se declanșează o eroare de genul celei din figura 4.18.

```
SQL> INSERT INTO personal (marca, numepren) VALUES (101, 'A1') ;

1 row created.

SQL> INSERT INTO personal (marca, numepren) VALUES (101, 'A2') ;
INSERT INTO personal (marca, numepren) VALUES (101, 'A2')
*
ERROR at line 1:
ORA-00001: unique constraint (FOTACHEM.PK_PERSONAL) violated
```

Figura 4.18. Încălcarea restricției (cheie primară) PK_PERSONAL

De remarcat și faptul că, la declararea restricțiilor de tip cheie primară/unicitate, Oracle crează automat indecși corespunzători.

4.6. Reguli de validare la nivel de atribut și înregistrare: clauza CHECK

Restricțiile-utilizator, denumite și restricții de comportament, sunt implementate sub forma regulilor de validare la nivel de câmp (*field validation rule*), la nivel de înregistrare (*record validation rule*) sau, dacă sunt mai complexe, sunt incluse în declanșatoare (*triggere*).

Pentru tabela PERSONAL instituim următoarele restricții la nivel de atribut:

- atributul *Marca* nu poate avea valori mai mici de 100;
- pentru nume și prenume, prima literă din fiecare cuvânt (nume, inițiala tatălui, prenume) este majusculă; restul, obligatoriu, litere mici;
- firma are numai șase compartimente: CONTA (contabilitate), FIN (financiar), PERS (personal-salarizare), MARK (marketing), PROD (producție), IT (sau sisteme informaționale).
- atributul *Colaborator* poate avea doar două valori, 'N' (Nu, adică nu e colaborator, ci angajat cu carte de muncă) și 'D'.

Pentru tabela PONTAJE regulile de validate la nivel de câmp sunt:

- atributul *Data* să fie cuprins între 1 ianuarie 2003 și 31 decembrie 2010;
- oricât de harnic ar fi, un angajat nu poate lucra într-o zi mai mult de 12 ore;
- un angajat în concediu are maximum 8 ore (este vorba de atributul *OreCO*) în fiecare zi de relache;

Spre deosebire de restricțiile la nivel de câmp, cele la nivel de înregistrare privesc simultan două sau mai multe atribute. În plus, momentul verificării lor succede verificării regulilor la nivel de atribut. Tabela PONTAJE prezintă trei asemenea restricții:

- CK_PONTAJE1, potrivit căreia un angajat este ori în concediu (orelucrate = 0 AND oreco >=0) ori la lucru (orelucrate >=0 AND oreco = 0);
- CK_PONTAJE2, după care un angajat nu poate avea un numărul de ore de noapte superior numărului orelor totale lucrate în ziua respectivă (orelucrate >= orenoapte);
- CK_PONTAJE3, prin care se evită eroarea de a avea mai multe ore de absențe nemotivate decât totalul orelor de lucru (posibile).

Listingul 4.5 conține clauzele definirii acestor noi restricții.

Listing 4.5. Reguli de validare la nivel de atribut și înregistrare

```

DROP TABLE pontaje ;
DROP TABLE personal ;

CREATE TABLE personal (
    marca INTEGER
        CONSTRAINT pk_personal PRIMARY KEY
        CONSTRAINT nn_personal_marca NOT NULL
    CONSTRAINT ck_personal_marca CHECK ( marca >= 100 )
    , numepren VARCHAR2(40)
        CONSTRAINT nn_personal_numepren NOT NULL
    CONSTRAINT un_personal_nuempren UNIQUE
    CONSTRAINT ck_personal_numepren
        CHECK (numepren=LTRIM(INITCAP(numepren)))
    , compart VARCHAR2(5) DEFAULT 'PROD'
        CONSTRAINT ck_personal_compart
        CHECK (compart in ( 'CONTA', 'FIN', 'PERS', 'MARK','PROD', 'IT' ) )
    , dataSV DATE DEFAULT SYSDATE
        CONSTRAINT nn_personal_datasv NOT NULL
    , salorar NUMBER(16,2) DEFAULT 45000
    , salorarco NUMBER(16,2) DEFAULT 40000
    , colaborator CHAR(1) DEFAULT 'N'
    CONSTRAINT nn_personal_colaborator NOT NULL
        CONSTRAINT ck_personal_colaborator CHECK (colaborator IN ('D','N'))
);

CREATE TABLE pontaje (
    marca INTEGER
        CONSTRAINT nn_pontaje_marca NOT NULL
    , data DATE DEFAULT SYSDATE
        CONSTRAINT nn_pontaje_data NOT NULL
    CONSTRAINT ck_pontaje_data CHECK
    (TO_CHAR(data,'YYYY') BETWEEN '2003' AND '2010')
    , orelucrate NUMBER(2) DEFAULT 8
    CONSTRAINT ck_pontaje_orelucrate CHECK (orelucrate BETWEEN 0 AND 12)
    , oreco NUMBER(2) DEFAULT 0
    CONSTRAINT ck_pontaje_oreco CHECK (oreco BETWEEN 0 AND 8)
    , orenoapte NUMBER(2) DEFAULT 0
    , oreabsnem NUMBER(2) DEFAULT 0
    , CONSTRAINT pk_pontaje PRIMARY KEY (marca,data)
    , CONSTRAINT ck_pontaje1 CHECK ((orelucrate = 0 AND oreco >=0) OR
    (orelucrate >=0 AND oreco = 0))
    , CONSTRAINT ck_pontaje2 CHECK (orelucrate >= orenoapte)
    , CONSTRAINT ck_pontaje3 CHECK (orelucrate >= oreabsnem)
);

```

```
);
```

Aflarea regulilor de validare la nivel de atribut și înregistrare din tabelele PERSONAL și PONTAJE presupune folosirea consultării:

```
SELECT constraint_name, table_name, search_condition
FROM user_constraints
WHERE table_name IN ('PERSONAL', 'PONTAJE')
AND constraint_type = 'C'
```

Rezultatul în SQL*Plus este cel din figura 4.19.

CONSTRAINT_NAME	TABLE_NAME	SEARCH_CONDITION
NN_PERSONAL_MARCA	PERSONAL	"MARCA" IS NOT NULL
NN_PERSONAL_NUMEPREN	PERSONAL	"NUMEPREN" IS NOT NULL
NN_PERSONAL_DATASU	PERSONAL	"DATASU" IS NOT NULL
NN_PERSONAL_COLABORATOR	PERSONAL	"COLABORATOR" IS NOT NULL
CK_PERSONAL_MARCA	PERSONAL	marca > 100
CK_PERSONAL_NUMEPREN	PERSONAL	numepren=LTRIM(INITCAP(numepren))
CK_PERSONAL_COMPART	PERSONAL	compart in ('CONTA', 'FIN', 'PERS', 'MARK', 'PROD', 'IT')
CK_PERSONAL_COLABORATOR	PERSONAL	colaborator IN ('D','N')
NN_PONTAJE_MARCA	PONTAJE	"MARCA" IS NOT NULL
NN_PONTAJE_DATA	PONTAJE	"DATA" IS NOT NULL
CK_PONTAJE_DATA	PONTAJE	TO_CHAR(data,'YYYY') BETWEEN '2003' AND '2010'
CK_PONTAJE_ORELUCRATE	PONTAJE	orelucrate BETWEEN 0 AND 12
CK_PONTAJE_ORECO	PONTAJE	oreco BETWEEN 0 AND 8
CK_PONTAJE1	PONTAJE	(orelucrate = 0 AND oreco >=0) OR (orelucrate >=0 AND oreco = 0)
CK_PONTAJE2	PONTAJE	orelucrate >= orenoapte
CK_PONTAJE3	PONTAJE	orelucrate >= oreabsnem

Figura 4.19. Restricții de tip CHECK pentru tabele PERSONAL și PONTAJE

4.7. Restricții referențiale

Restricțiile referențiale constituie, probabil, categoria care dă cele mai mari bătaii de cap SGBD-urilor, din zorii bazelor de date relaționale, până în prezent. Oracle, însă, se comportă onorabil, deoarece folosind clauzele REFERENCES / FOREIGN KEY se rezolvă majoritatea problemelor referențiale. Astfel, dacă formatul comenzii CREATE TABLE pentru PONTAJE este:

```
CREATE TABLE pontaje (
    marca INTEGER
        CONSTRAINT nn_pontaje_marca NOT NULL
        CONSTRAINT fk_pontaje_personal
            REFERENCES personal (marca)
    , data DATE DEFAULT SYSDATE
    ...
```

se va institui o restricție referențială între PONTAJE (tabelă copil) și PERSONAL (tabelă părinte), în virtutea căreia Oracle va “veghea neabătut” ca:

- să nu se șteargă vreo înregistrare în PERSONAL (părinte), dacă acel angajat are minim un pontaj (o înregistrare copil în PONTAJE);
- în PONTAJE să nu apară vreo marcă (fie printr-o inserare, fie printr-o modificare) ce nu se regăsește în PERSONAL.

Mai mult, se poate institui și regula ștergerii în cascadă (ON DELETE CASCADE): la ștergerea unui părinte, se șterg automat toți copiii săi (fără sentimentalisme, vă rugăm !).

```
CREATE TABLE pontaje (
    marca INTEGER
        CONSTRAINT nn_pontaje_marca NOT NULL
        CONSTRAINT fk_pontaje_personal
            REFERENCES personal (marca)
            ON DELETE CASCADE
    , data DATE DEFAULT SYSDATE
    ...
```

Sau, tot pentru a nu încalca restricția referențială, la ștergerea unei înregistrări părinte toate valorile copil pot fi "trecute" pe NULL (ON DELETE SET NULL):

```
CREATE TABLE pontaje (
    marca INTEGER
        CONSTRAINT nn_pontaje_marca NOT NULL
        CONSTRAINT fk_pontaje_personal
            REFERENCES personal (marca)
            ON DELETE SET NULL
    , data DATE DEFAULT SYSDATE
    ...
```

Cea mai rezonabilă opțiune este cea implicită, care interzice ștergerea unei înregistrări părinte, atâta timp cât există măcar un copil (un soi de ON DELETE RESTRICT).

Din păcate, nici în Oracle 9i nu apare clauza ON UPDATE CASCADE. Astfel încât, la modificarea valorii unei chei primare, aceasta trebuie modificată prin declanșatoare în toate înregistrările copil. Implicit, după instituirea restricției referențiale, în tabela părinte regula de modificare a cheii primare este ON UPDATE RESTRICT (modificarea este interzisă dacă există măcar o înregistrare copil pentru vechea valoare).

În dicționarul de date, informațiile despre restricțiile referențiale sunt păstrate tot în tabelele USER_CONSTRAINTS și USER_CONS_COLUMNS. Fraza SELECT următoare afișează conținutul din figura 4.20.

```
SELECT constraint_name, table_name, r_constraint_name
FROM user_constraints
```



```
WHERE constraint_type = 'R'
```

CONSTRAINT_NAME	TABLE_NAME	R_CONSTRAINT_NAME
FK_PONTAJE_PERSONAL	PONTAJE	PK_PERSONAL

Figura 4.20. Date despre restricția referențială

Extragerea corespundențelor dintre atributele copil și cele părinte e o sarcină mai dificilă și presupune execuția unei fraze `SELECT` mult mai elaborate:

```
SELECT uc.constraint_name AS restrictie_copil,
       uc1.table_name AS tabela_copil,
       uc1.column_name AS atribut_copil,
       uc.r_constraint_name AS restrictie_parinte,
       uc2.table_name AS tabela_parinte,
       uc2.column_name AS atribut_parinte
FROM user_constraints uc, user_cons_columns uc1,
     user_cons_columns uc2
WHERE uc.constraint_name = uc1.constraint_name AND
      uc.r_constraint_name = uc2.constraint_name
      AND uc1.position = uc2.position AND
      uc.constraint_type = 'R'
ORDER BY uc.constraint_name, uc1.position
```

Vă invităm să o verificați, atât acum, cât și după lansarea scriptului următor.

4.8. Scriptul de creare a tabelelor și declararea restricțiilor

Listingul 4.6 conține comenzile și clauzele necesare creării tabelelor și declarării restricțiilor pentru baza de date pe care o vom utiliza în continuare. În afara observației că tabelele `SPORURI`, `REȚINERI` și `SALARII` au chei primare compuse din atributele `Marca`, `An` și `Luna`, credem că orice alt comentariu este de prisos.

Listing 4.6. Definirea tabelelor/restricțiilor

```
DROP TABLE transe_sv ;
DROP TABLE salarii ;
DROP TABLE retineri ;
DROP TABLE sporuri ;
DROP TABLE pontaje ;
DROP TABLE personal ;

CREATE TABLE personal (
  marca INTEGER
    CONSTRAINT pk_personal PRIMARY KEY
    CONSTRAINT nn_personal_marca NOT NULL
    CONSTRAINT ck_personal_marca CHECK ( marca > 100 )
  , numepren VARCHAR2(40)
```

```

        CONSTRAINT nn_personal_numepren NOT NULL
        CONSTRAINT un_personal_numepren UNIQUE
        CONSTRAINT ck_personal_numepren CHECK
            (numepren=LTRIM(INITCAP(numepren)))
    , compart VARCHAR2(5) DEFAULT 'PROD'
        CONSTRAINT ck_personal_compart CHECK
            (compart in ( 'CONTA', 'FIN', 'PERS', 'MARK','PROD', 'IT' ) )
    , dataSV DATE DEFAULT SYSDATE
        CONSTRAINT nn_personal_datasv NOT NULL
    , salorar NUMBER(16,2) DEFAULT 45000
    , salorarco NUMBER(16,2) DEFAULT 40000
    , colaborator CHAR(1) DEFAULT 'N'
        CONSTRAINT nn_personal_colaborator NOT NULL
        CONSTRAINT ck_personal_colaborator CHECK (colaborator IN ('D','N'))
    );

CREATE TABLE pontaje (
    marca INTEGER
        CONSTRAINT nn_pontaje_marca NOT NULL
        CONSTRAINT fk_pontaje_personal REFERENCES personal (marca)
    , data DATE DEFAULT SYSDATE
        CONSTRAINT nn_pontaje_data NOT NULL
        CONSTRAINT ck_pontaje_data CHECK
            (TO_CHAR(data,'YYYY') BETWEEN '2003' AND '2010')
    , orelucrate NUMBER(2) DEFAULT 8
        CONSTRAINT ck_pontaje_orelucrate CHECK (orelucrate BETWEEN 0 AND 12)
    , oreco NUMBER(2) DEFAULT 0
        CONSTRAINT ck_pontaje_oreco CHECK (oreco BETWEEN 0 AND 8)
    , oreoapte NUMBER(2) DEFAULT 0
    , oreabsnem NUMBER(2) DEFAULT 0
    , CONSTRAINT pk_pontaje PRIMARY KEY (marca,data)
    , CONSTRAINT ck_pontaje1 CHECK ((orelucrate = 0 AND oreco >=0) OR (orelucrate >=0
        AND oreco = 0))
    , CONSTRAINT ck_pontaje2 CHECK (orelucrate >= oreoapte)
    , CONSTRAINT ck_pontaje3 CHECK (orelucrate >= oreabsnem)
    );

CREATE TABLE sporuri (
    marca INTEGER
        CONSTRAINT nn_sporuri_marca NOT NULL
        CONSTRAINT fk_sporuri_personal REFERENCES personal (marca)
    , an NUMBER(4)
        CONSTRAINT nn_sporuri_an NOT NULL
        CONSTRAINT ck_sporuri_an CHECK ( an BETWEEN 2003 AND 2010 )
    , luna NUMBER(2)
        CONSTRAINT nn_sporuri_luna NOT NULL
        CONSTRAINT ck_sporuri_luna CHECK (luna BETWEEN 1 AND 12)
    , spvech NUMBER(16,2)
    , oreoapte NUMBER (3)
    , spnoapte NUMBER (16,2)
    , altesp NUMBER (16,2)
    , CONSTRAINT pk_sporuri PRIMARY KEY (marca, an, luna)
    );

CREATE TABLE retineri (
    marca INTEGER
        CONSTRAINT nn_retineri_marca NOT NULL
        CONSTRAINT fk_retineri_marca REFERENCES personal (marca)
    ,an NUMBER (4)
        CONSTRAINT nn_retineri_an NOT NULL

```

```

        CONSTRAINT ck_retineri_an CHECK ( an BETWEEN 2001 AND 2012 )
    , luna NUMBER (2)
        CONSTRAINT nn_retineri_luna NOT NULL
        CONSTRAINT ck_retineri_luna CHECK (luna BETWEEN 1 AND 12)
    , popririi NUMBER (16,2)
    , CAR NUMBER (16,2)
    , alteret NUMBER (16,2)
    , CONSTRAINT pk_retineri PRIMARY KEY (marca, an, luna)
    );

CREATE TABLE salarii (
    marca INTEGER
        CONSTRAINT nn_salarii_marca NOT NULL
        CONSTRAINT fk_salarii_personal REFERENCES personal (marca)
    , an NUMBER (4)
        CONSTRAINT nn_salarii_an NOT NULL
        CONSTRAINT ck_salarii_an CHECK ( an BETWEEN 2001 AND 2012 )
    , luna NUMBER (2)
        CONSTRAINT nn_salarii_luna NOT NULL
        CONSTRAINT ck_salarii_luna CHECK (luna BETWEEN 1 AND 12)
    , orelucrate NUMBER (3)
    , oreco NUMBER (3)
    , venitbaza NUMBER (16,2)
    , sporuri    NUMBER (16,2)
    , impozit    NUMBER (16,2)
    , retineri   NUMBER (16,2)
    , CONSTRAINT pk_salarii PRIMARY KEY (marca, an, luna)
    );

CREATE TABLE transe_sv (
    ani_limita_inf INTEGER
    , ani_limita_sup INTEGER
    , procent_sv NUMBER (4,2)
    );

```

4.9. Modificarea structurii tabelelor: comanda ALTER TABLE

Schema reprezintă aspectul constant, invariabil sau, mai bine zis, puțin variabil în timp al unei baze de date. Cu toate acestea, apariția unui atribut nou, modificarea lungimii unui atribut sunt probleme de care orice administrator/dezvoltator de aplicații se ciocnește destul de frecvent. Comanda SQL dedicată modificării structurii bazei de date este `ALTER TABLE`.

4.9.1. Adăugări/modificări/ștergeri de attribute

Adăugarea atributului `DataNast` (data nașterii) în tabela `COPIE_PERSONAL` se realizează astfel:

```
ALTER TABLE copie_personal ADD datanast DATE
```

La prima vedere, ștergerea unui câmp (posibilă începând cu versiunea 8i) e la fel de simplă ca și adăugarea. Aceasta în Oracle 9i, deoarece în versiuni mai vechi ștergerea nu era permisă. Atributul pe care tocmai l-am adăugat poate fi șters astfel:

```
ALTER TABLE copie_personal DROP COLUMN datanast
```

Ștergerea atributului `Compart` atrage după sine, cum este și normal ștergerea restricțiilor definite pentru acesta:

```
ALTER TABLE copie_personal DROP COLUMN compart
```

Dacă, însă, vom dori să eliminăm atributul `OreLucrate` din `PONTAJE`, avem toate șansele să primim mesajul de eroare din figura 4.21.

```
SQL> ALTER TABLE pontaje DROP COLUMN orelucrate ;
ALTER TABLE pontaje DROP COLUMN orelucrate
*
ERROR at line 1:
ORA-12991: column is referenced in a multi-column constraint
```

Figura 4.21. Tentativă de ștergere a unui câmp ce apare într-o regulă la nivel de înregistrare

Mesajul este limpede: nu putem șterge în acest fel atributul, deoarece apare într-o regulă de validate la nivel de înregistrare. Aflarea tuturor restricțiilor în care este implicat atributul `OreLucrate` este o operațiune destul de simplă:

```
SELECT r1.constraint_name, r1.table_name, search_condition
FROM USER_CONS_COLUMNS r1, USER_CONSTRAINTS r2
WHERE r1.constraint_name = r2.constraint_name AND
      column_name = 'ORELUCRATE'
```

În SQL*Plus se afișează rezultatul din figura 4.22.

CONSTRAINT_NAME	TABLE_NAME	SEARCH_CONDITION
CK_PONTAJE_ORELUCRATE	PONTAJE	orelucrate BETWEEN 0 AND 12
CK_PONTAJE1	PONTAJE	(orelucrate = 0 AND oreco >=0) OR (orelucrate >=0 AND oreco = 0)
CK_PONTAJE2	PONTAJE	orelucrate >= orenoapte
CK_PONTAJE3	PONTAJE	orelucrate >= oreabsnem

Figura 4.22. Restricțiile în care apare atributul `OreLucrate`

Așadar, o primă variantă ar fi să ștergem mai întâi toate restricțiile (ultimele trei cu deosebire), și apoi să reîncercăm să lansăm comanda `ALTER TABLE` în formatul de mai sus. Se poate, însă, și mai simplu, prin folosirea clauzei `CASCADE CONSTRAINTS`:

```
ALTER TABLE pontaje DROP COLUMN orelucrate
CASCADE CONSTRAINTS
```

Cu această nouă clauză se poate șterge orice atribut din orice tabelă. Efectul poate fi, pe alocuri, devastator, așa că nu e recomandată uitucilor cardiaci.

De la Oracle 8i apare și o facilitate ce pare, la prima vedere, mai exotică: declararea unei coloane (atribut) ca... nefolositoare. Cei de la firma producătoare s-au gândit, probabil, că, întrucât ștergerea unei coloane în condițiile existenței unui mare număr de utilizatori ce lucrează simultan cu tabela, generează serioase blocaje, ar fi de dorit ca, mai întâi, coloana/coloanele să fie marcate ca inutilizabile, iar apoi, într-un “moment de respiro” al bazei să se procedeze la ștergerea definitivă.

```
ALTER TABLE personal SET UNUSED (salorarco)
sau
ALTER TABLE personal SET UNUSED COLUMN salorarco
```

Din momentul declarației, atributul respectiv este ca și inexistent, nemaiputând fi invocat. Astfel, comanda:

```
INSERT INTO personal (marca, numepren, salorarco)
VALUES (102, 'A2', 45000)
```

va genera eroare. Eliminarea definitivă a tuturor atributelor “marcate” ca inutilizabile se realizează prin:

```
ALTER TABLE personal DROP UNUSED COLUMNS
```

Cum, după marcare, nu există nici un mod de a vizualiza/accesa atributul respectiv în tabelă, singura informație pe care o avem în dicționar este numărul total de attribute marcate în fiecare tabelă, afișabil prin interogarea:

```
SELECT * FROM USER_UNUSED_COL_TABS
```

Creșterea numărului de caractere pentru un câmp nu e nici o problemă:

```
ALTER TABLE personal MODIFY compart VARCHAR2(15)
```

Reducea lungimii este o altă poveste, după cum lesne se vede din figura 4.23.

```
SQL> ALTER TABLE personal MODIFY compart VARCHAR2(2)
2 /
ALTER TABLE personal MODIFY compart VARCHAR2(2)
*
```

ERROR at line 1:
ORA-01441: cannot decrease column length because some value is too big

Figura 4.23. Tentativă de reducere a lungimii atributului Compart

În Oracle modificarea tipului unui atribut se poate face astfel: din CHAR în VARCHAR2 sau VARCHAR; din VARCHAR2 sau VARCHAR în CHAR, dar numai dacă respectivul atribut conține valoarea NULL în toate liniile tabelului sau dacă nu se modifică și lungimea atributului. Pentru exemplificare, transformăm tipul atributului `Compart` din VARCHAR2(5) în CHAR(5):

```
ALTER TABLE personal MODIFY compart CHAR(5)
```

Pentru modificarea valorii implicite a unui atribut, în cazul nostru `Compart` (din 'PROD' în 'IT'), formatul comenzii este:

```
ALTER TABLE personal MODIFY compart DEFAULT 'IT'
```

Oracle 9i permite și redenumirea unui atribut:

```
ALTER TABLE personal RENAME COLUMN compart TO compartiment
```

4.9.2. Adăugări/ștergeri de restricții

Interzicerea valorilor nule pentru atributul `SalOrar` se realizează astfel:

```
ALTER TABLE personal MODIFY salorar NOT NULL
```

Invers, pentru a permite valori NULL pentru același atribut:

```
ALTER TABLE personal MODIFY salorar NULL
```

Toate restricțiile, adică valori nenule, cheie primară, unicitate, reguli de validare, restricții referențiale pot fi declarate și ulterior creării tabelului și, bineînțeles, șterse la un moment dat. În scriptul de creare a tabelului din bază, `TRANSE_SV` nu are nici o restricție. Ceea ce nu înseamnă că este prea târziu. Mai întâi, instituim o regulă de validare la nivel de înregistrare, `CK_TRANSE_SV1`, care să verifice că limita inferioară a tranșei este strict mai mică decât limita superioară:

```
ALTER TABLE transe_sv ADD CONSTRAINT ck_transe_sv1  
CHECK (ani_limita_inf < ani_limita_sup)
```

Iar dacă vrem să facem atributul `Ani_Limita_Inf` cheie primară, nu ne rămâne decât să lansăm în execuție:

```
ALTER TABLE transe_sv  
ADD CONSTRAINT pk_transe_sv  
PRIMARY KEY (ani_limita_inf)
```

Cunoscând numele restricției ce se dorește a fi ștearsă, cum ar fi: CK_PERSONAL_COLABORATOR, ștergerea sa se poate realiza astfel:

```
ALTER TABLE personal  
DROP CONSTRAINT ck_personal_colaborator
```

Ca și în cazul ștergerii atributelor, nu în toate cazurile lucrurile stau așa de simplu. Astfel, dacă se dorește ștergerea cheii primare a tablei PERSONAL apare o opoziție din partea serverului, deoarece atributul Marca apare în o serie de restricții referențiale. Este motivul pentru care se afișează mesajul din figura 4.24.

```
SQL> ALTER TABLE personal DROP PRIMARY KEY  
2 /  
ALTER TABLE personal DROP PRIMARY KEY  
*  
ERROR at line 1:  
ORA-02273: this unique/primary key is referenced by some foreign keys
```

Figura 4.24. Tentativă de ștergere a cheii primare din PERSONAL

Dacă am fi vrut să eliminăm restricția de cheie primară a tablei SALARII, prin lansarea comenzii:

```
ALTER TABLE salarii DROP PRIMARY KEY  
sau  
ALTER TABLE salarii DROP CONSTRAINT pk_salarii
```

nu am fi întâmpinat prea mare rezistență. Oracle nu se împotrivesc ștergerii restricției PK_PERSONAL, indiferent de câte alte restricții referențiale depind de aceasta, dacă folosim clauza CASCADE prin care se șterg și toate restricțiile care depind de cea ștearsă:

```
ALTER TABLE personal DROP PRIMARY KEY CASCADE
```

4.10. Restricții dezactivate, reactivate, amânate, imediate

O serie de situații, precum importul datelor din SGBD-uri mai simple, ce nu prezintă mecanisme de declarare a restricțiilor de cheie primară/unicitate, reguli de validare sau referențiale (ex. FoxPro 2.x), reclamă dezactivarea temporară a unor restricții, efectuarea importului (sau altor operațiuni), verificarea consistenței și integrității datelor, și numai după aceea reactivarea restricțiilor. Varianta ștergerii și re-creării restricțiilor înseamnă un efort mult mai mare din partea dezvoltatorilor de aplicații sau administratorilor bazei de date, și, în plus, consumă

serioase resurse ale SGBD-ului, lucru cu atât mai evident cu cât tabelele sunt mai stufoase.

Dezactivarea restricției CK_PONTAJE1 asociată tablei PONTAJE se realizează folosind tot comanda ALTER TABLE în formatul:

```
ALTER TABLE pontaje DISABLE CONSTRAINT ck_pontaje1;
```

Ulterior, reactivarea presupune execuția comenzii:

```
ALTER TABLE pontaje ENABLE CONSTRAINT ck_pontaje1;
```

Analog ștergerii de attribute și de restricții, dezactivarea “simplă” este întimpinată cu ostilitate de Oracle dacă de restricția dezactivată depind alte restricții. Firește, cel mai la îndemână exemplu este tot PK_PERSONAL:

```
ALTER TABLE personal DISABLE CONSTRAINT pk_personal
```

se soldează cu un mesaj de eroare (ORA-02297: cannot disable constraint (PERSONAL.PK_PERSONAL) - dependencies exist). Soluția ține de folosirea clauzei CASCADE.

Mai trebuie amintit faptul că activarea obișnuită a unei restricții implică verificarea tuturor înregistrărilor existente deja în tabela respectivă. Cazul luat drept exemplu este cel în care după crearea tablei și popularea sa cu numeroase înregistrări, intervine o modificare asupra definiției inițiale, cum ar fi adăugarea unei noi restricții sau modificarea uneia existente. Dacă administratorul bazei de date nu are răbdare să modifice datele existente pentru a corespunde noilor condiții, dar nu dorește nici ca să-și complice viața cu noi înregistrări inconsistente, atunci poate activa restricțiile cu pricina fără *validarea* înregistrărilor prezente. Acest mod de activare înseamnă că doar noile înregistrări vor fi supuse controlului conformității cu aceste restricții.

Să luăm ca exemplu cazul în care întreprinderea își modifica politica în ceea ce privește programul de lucru reducându-l la maximum 10 ore. În baza de date problema se reduce la modificarea restricției CK_PONTAJE_ORELUCRATE din tabela PONTAJE.

Zis și făcut:

```
ALTER TABLE pontaje DROP CONSTRAINT ck_pontaje_orelucrate;  
ALTER TABLE pontaje ADD CONSTRAINT ck_pontaje_orelucrate  
CHECK (orelucrate BETWEEN 0 AND 10);
```

Însă ... surpriză, serverul ar putea returna un mesaj de eroare bizar:

```
SQL> ALTER TABLE pontaje ADD CONSTRAINT ck_pontaje_orelucrate
2 CHECK (orelucrate BETWEEN 0 AND 10);
ALTER TABLE pontaje ADD CONSTRAINT ck_pontaje_orelucrate
*
```

```
ERROR at line 1:
ORA-02293: cannot validate (PERSONAL.CK_PONTAJE_ORELUCRATE) - check constraint violated
```

Figura 4.25. Imposibilitatea validării restricției ck_pontaje_orelucrate

Ținând cont de faptul că nu este implicată nici o frază INSERT, nenorocirea provine de la eventualele pontaje existente (ar putea exista un salariat atât de harnic încât să fi lucrat peste 10 ore):

```
SQL> SELECT * FROM pontaje;
```

MARCA	DATA	ORELUCRATE	ORECO	ORENOAPTE	OREABSNEM
2001	07-FEB-03	6	0	0	0
2002	07-FEB-03	6	0	0	0
2003	07-FEB-03	6	0	0	0
2004	07-FEB-03	6	0	0	0
2005	07-FEB-03	6	0	0	0
2002	10-FEB-03	10	0	2	0
2001	10-FEB-03	12	0	2	0
2004	11-FEB-03	4	0	4	0
2001	11-FEB-03	4	0	4	0

Figura 4.26. Există un salariat „prea” harnic care nu respectă restricția pe care am dorit să o impunem

Cum, în cazul de față, pontajul angajatului nu poate fi modificat, fiind implicat în calculul drepturilor pe luna februarie, singura soluție pentru a tăia elanul altor salariați vrednici (de acum înainte) este:

```
ALTER TABLE pontaje ADD CONSTRAINT ck_pontaje_orelucrate
CHECK (orelucrate BETWEEN 0 AND 10) ENABLE NOVALIDATE;
```

sau:

```
ALTER TABLE pontaje ADD CONSTRAINT ck_pontaje_orelucrate
CHECK (orelucrate BETWEEN 0 AND 10) DISABLE;
ALTER TABLE pontaje ENABLE NOVALIDATE
CONSTRAINT ck_pontaje_orelucrate;
```

Dacă opțiunile de dezactivare/reactivare a restricțiilor fac parte de o bună bucată de vreme din arsenalul Oracle, din versiunea 8 a fost introdusă și posibilitatea de a amâna momentul verificării unei restricții. Cele declarate amânabile și activate în acest fel sunt verificate abia la sfârșitul tranzacției (COMMIT), iar cele ne-amânabile, sau amânabile dar configurate ca *immediate*, sunt verificate la fiecare instrucțiune DML (INSERT/UPDATE/DELETE). Această caracteristică permite (evident, forțând un pic nota) chiar propagarea în cascadă a modificărilor dintr-o tabelă părinte în tabelele copil fără a crea declanșatoarele care

să execute actualizarea cheilor străine din tabelele copil la modificările cheilor primare din tabelele părinte (UPDATE CASCADE).

De exemplu, conducerea firmei emite o notă internă prin care modul de generare a mărcilor angajaților nu se mai face secvențial, ci după un alt criteriu. Administratorul bazei de date va încerca să modifice mărcile consemnate în tabela PERSONAL, ceea ce, evident, va genera o eroare (vezi figura 4.27):

```
SQL> UPDATE personal SET marca = 1001 WHERE marca = 2001;
UPDATE personal SET marca = 1001 WHERE marca = 2001
*
ERROR at line 1:
ORA-02292: integrity constraint (PERSONAL.FK_PONTAJE_PERSONAL) violated - child record found
```

Figura 4.27. Modificarea valorilor atributelor din cheia primară (tabela părinte) este aparent imposibilă

Singura modalitate de a evita mesajul anterior constă în actualizarea prealabilă a tabelului copil și apoi a celui părinte. Să presupunem, însă, că avem de a face cu un administrator încăpățânat care dorește actualizarea în succesiunea PERSONAL - PONTAJE. Implicit, la creare, restricțiile sunt *neamânabile*. Pentru a declara restricția de integritate referențială a tabelului PONTAJE ca amânabilă, atunci se poate proceda în felul următor:

```
ALTER TABLE pontaje DROP CONSTRAINT fk_pontaje_personal;
ALTER TABLE pontaje ADD
    CONSTRAINT fk_pontaje_personal FOREIGN KEY (marca)
    REFERENCES personal (marca)
    DEFERRABLE INITIALLY IMMEDIATE;
```

Restricția FK_PONTAJE_PERSONAL va fi *amânabilă*, însă clauza INITIALLY IMMEDIATE determină deocamdată verificarea acesteia la fiecare comandă SQL (la fel ca în cazul restricțiilor neamânabile obișnuite). Trecerea din starea *amânabilă imediată* în *amânabilă amânată* presupune lansarea comenzii:

```
SET CONSTRAINT fk_pontaje_personal DEFERRED;
```

De acum înainte în sesiunea curentă vom putea executa fără teamă comenzi „violatoare”, în prima fază, a restricțiilor referențiale:

```
UPDATE personal SET marca = 1001 WHERE marca = 2001;
```

Dacă vom încerca să "comitem" modificările, atunci vom vedea că mecanismul de asigurare a restricțiilor nu poate fi păcălit atât de simplu:

```

SQL> SET CONSTRAINT fk_pontaje_personal DEFERRED;

Constraint set.

SQL> UPDATE personal SET marca = 1001 WHERE marca = 2001;

1 row updated.

SQL> COMMIT;
COMMIT
*
ERROR at line 1:
ORA-02091: transaction rolled back
ORA-02292: integrity constraint (PERSONAL.FK_PONTAJE_PERSONAL) violated - child record found

```

Figura 4.28. Vigilența serverului față de restricțiile amânate se face simțită în momentul comiterii

Serverul Oracle s-a simțit atât de ofensat, încât a anulat întreaga tranzacție doar pentru o încercare „nevinovată” de a păcăli o banală restricție. Pentru a avea succes trebuie procedat cuminte în felul următor (chiar dacă am amânat o restricție, Oracle n-are memorie (RAM) scurtă):

```

SET CONSTRAINT fk_pontaje_personal DEFERRED;
UPDATE personal SET marca = 1001 WHERE marca = 2001;
UPDATE pontaje SET marca = 1001 WHERE marca = 2001;
COMMIT;

```

4.11. Scripturi SQL*Plus de refacere a tabelelor și restricțiilor

Pe baza complexelor informații stocate în dicționarul bazei, pot fi generate o serie de scripturi pentru salvarea tabelelor și restricțiilor din schema curentă. Operațiunea este utilă atât pentru mutarea bazei de date pe alt server, fără a recurge la operațiunile de export/import, cât și pentru salvarea structurii bazei de date, și, astfel, diminuarea semnificativă a gradului de stres a administratorului BD și familiei acestuia.

4.11.1. Re-crearea tabelelor

Dacă numele tabelelor pot fi preluate din tabela virtuală a catalogului sistem USER_TABLES, numele, tipul, lungimea și valorile implicite ale atributelor se regăsesc în USER_TAB_COLUMNS. Scriptul se obține în SQL*Plus prin comanda SPOOL prin care rezultatele frazei SELECT următoare vor fi salvate în fișierul ASCII re_creare_tabele.sql din directorul f:\oracle_car-te\cap04_creare_tabele.

Listing 4.7. Re-crearea tabelelor din schema curentă

```

column "--Creare_tabele" FORMAT A60
column C2 FORMAT A40

```

```

spool f:\oracle carte\cap04 creare_tabele\re creare_tabele.sql
SELECT
  ' CREATE TABLE ' || table_name || ' (' AS "--Creare_tabele",
  '-' || RPAD(table_name,30) || '-' as c2
FROM user_tables
UNION
SELECT CASE column_id WHEN 1 THEN '' ELSE ',' END ||
  column_name || ' ' || data_type ||
  CASE WHEN data_type NOT IN ('NUMBER', 'CHAR', 'VARCHAR2') THEN ''
  ELSE '(' ||
    NVL(data_precision, data_length) ||
    CASE WHEN data_scale > 0 THEN ', ' || data_scale ELSE '' END
  || ')'
END ,
  '-' || RPAD(table_name,30) || TO_CHAR(column_id,'999')
FROM user_tab_columns
UNION
SELECT ')';' || CHR(13) , '-' || RPAD(table_name,30) || '-' || CHR(123)
FROM user_tables
UNION
SELECT '' || CHR(13) , '-' || RPAD(table_name,30) || '-' || CHR(124)
FROM user_tables
ORDER BY 2 ;

spool off

```

Cele trei părți ale comenzii CREATE TABLE:

- CREATE TABLE tabelă (
- atribut1 tip1 (lungime1 [, poz_fracționare1])
[, atribut2 tip2 (lungime2 [, poz_fracționare2])
...]
-) ;

sunt generate de trei fraze SELECT speciale, conectate prin operatorul UNION. La acestea se mai adaugă o linie pentru spațiere, linie care conține spațiu și caracterul ASCII al tastei RETURN (13). Pentru ca dispunerea liniilor să fie corectă, se folosește, drept artificio, o a doua coloană ce concatenează numele tablei cu:

- spațiu, pentru linia CREATE TABLE ...;
- identificatorul coloanei (column_id), pentru linia corespondentă fiecărui atribut;
- caracterul ce are codul ASCII 123, mai mare decât orice identificator de coloană;
- caracterul ce are codul ASCII 124, pentru ultima linie a fiecărei table.

În mod normal, SELECT-ul de creare ar fi trebuit precedat de unul de ștergere, de genul SELECT ' DROP TABLE ' || table_name || ' CASCADE ; ' AS " " FROM user_tables. Noi, însă, am aplicat un principiu preluat din contabilitate, și anume cel al prudenței. O porțiune din fișierul ASCII generat este prezentată în figura 4.29.

Figura 4.29. Fișierul generat de listingul 4.7.

4.11.2. Re-crearea clauzelor DEFAULT

La prima tentativă de recreare a tabelor, observând existența atributului `Data_Default` în `USER_TAB_COLUMNS` ne-am avântat să includem și clauza `DEFAULT`. Din păcate, cei de la Oracle s-au gândit că formatul ideal pentru păstrarea valorilor implicite ale atributelor este unul `LONG`. Păcatul decurge din faptul că acest tip de date este extrem de năzuros, chiar documentația Oracle recomandând degrabă conversia a acestor atribute în formate `LOB`. Recomandarea nu se aplică, însă, la propriile tabele din dicționarul de date, astfel încât pentru a salva situația am angajat temporar tabela `TEMP1`, în care atributului `Val_Implicită` este de tip `CLOB`, iar la inserarea liniilor din `USER_TAB_COLUMNS` s-a folosit funcția `TO_LOB` – vezi listing 4.8.

Listing 4.8. Extragerea valorilor implicite

```
DROP TABLE temp1;

CREATE TABLE temp1 (
  tabela VARCHAR2(30),
  atribut VARCHAR2(30),
  id_atribut NUMBER(2),
  tip_atribut VARCHAR2(20),
  val_implicita CLOB
);

INSERT INTO temp1
  SELECT table_name, column_name, column_id, data_type, TO_LOB(data_default)
  FROM user_tab_columns
  WHERE data_default IS NOT NULL ;
```

```

COMMIT ;

spool f:\oracle_carte\cap04_creare_tabele\re_creare_default.sql

SELECT ' ALTER TABLE ' || tabela || ' MODIFY ' || atribut || ' DEFAULT '
      || val_implicita || ';' AS ""
FROM temp1 ;

spool off

```

Fișierul ASCII generat prin comanda SPOOL este re_creare_default.sql – vezi figura 4.30.

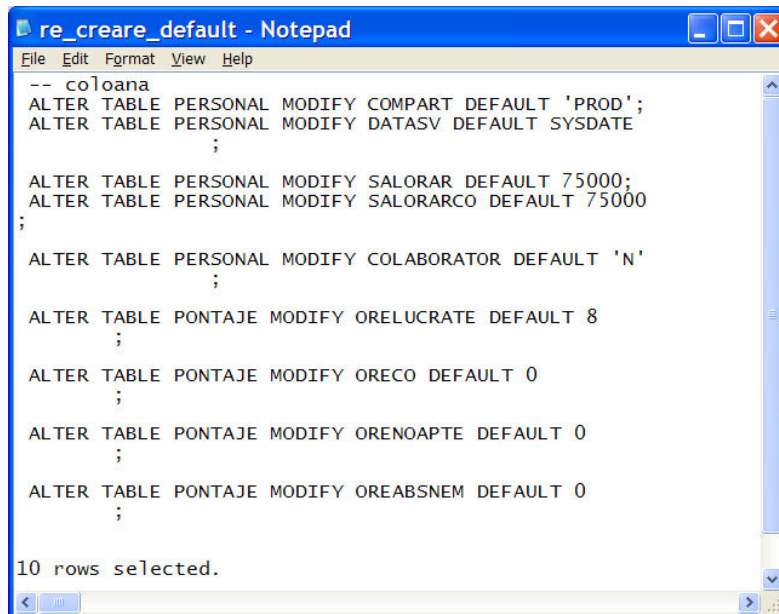


Figura 4.30. Scriptul de refacere a valorilor implicite

4.11.3. Restricții NOT NULL

Pentru fiecare atribut declarat NOT NULL creăm o restricție cu numele *nn_tabelă_atribut*, așa încât scriptul din listingul 4.9 funcționează fără prea multe emoții.

Listing 4.9. Re-declararea restricțiilor NOT NULL

```

spool f:\oracle_carte\cap04_creare_tabele\re_creare_notnull.sql

SELECT ' ALTER TABLE ' || table_name || ' MODIFY ( ' || column_name
      || ' CONSTRAINT NN_' || table_name || '_' || column_name || ' NOT NULL ); '
FROM user_tab_columns
WHERE nullable = 'N'

```

```
ORDER BY table name. column name :
```

```
spool off
```

Am avea oarecari probleme dacă restricțiile de tip `NOT NULL` pentru un atribut ar apărea, singure sau combinate, în clauze `CHECK` la nivel de atribut sau înregistrare. Noi, însă, lucrăm ordonat (din acest punct de vedere).

4.11.4. Chei primare și alternative

După cum am văzut câteva pagini mai sus, cheilor primare (`PRIMARY KEY`) și alternative le corespund în tabela virtuală din dicționar `USER_CONSTRAINTS` câte o restricție de tip 'P' și 'U'. Elementul de dificultate ține de faptul că o cheie primară sau candidată poate fi compusă din două sau mai multe atribute, fapt pentru care trebuie folosit atributul `Position` din tabela virtuală `USER_CONS_COLUMNS`.

Scriptul din listing 4.10 șterge mai întâi toate restricțiile de tip cheie primară și alternativă, apoi, prin două fraze `SELECT`, salvează în fișierul ASCII în care s-a făcut redirectarea două rânduri de comenzi `ALTER TABLE` corespunzătoare celor două tipuri de cheie candidat. Logic ar fi fost să procedăm analog (în sensul ștergerii) și pentru restricțiile de non-nulitate din scriptul discutat în paragraful anterior, însă niciodată nu e prea târziu.

Listing 4.10. Re-declararea cheilor primare și alternative

```
spool f:\oracle_carte\cap04_creare_tabele\re_creare_chei_primare_si_alternative.sql

SELECT 'ALTER TABLE ' || table_name || ' DROP CONSTRAINT ' ||
       constraint_name || ' CASCADE ;' AS "--Stergere restrictie"
FROM user_constraints
WHERE constraint_type IN ('P', 'U') ;

SELECT 'ALTER TABLE ' || table_name || ' ADD CONSTRAINT pk_' || table_name ||
       ' PRIMARY KEY ' || '(' || atr1 ||
       CASE WHEN atr2 IS NOT NULL THEN ', ' || atr2 END ||
       CASE WHEN atr3 IS NOT NULL THEN ', ' || atr3 END
       || ') ;' AS "--Cheile primare"
FROM user_constraints uc
LEFT OUTER JOIN
    (SELECT constraint_name, column_name AS atr1
     FROM user_cons_columns
     WHERE position = 1) a1 ON uc.constraint_name = a1.constraint_name
LEFT OUTER JOIN
    (SELECT constraint_name, column_name AS atr2
     FROM user_cons_columns
     WHERE position = 2) a2 ON uc.constraint_name = a2.constraint_name
LEFT OUTER JOIN
    (SELECT constraint_name, column_name AS atr3
     FROM user_cons_columns
     WHERE position = 3) a3 ON uc.constraint_name = a3.constraint_name
WHERE constraint_type = 'P' ;

SELECT 'ALTER TABLE ' || table_name || ' ADD CONSTRAINT un_' || table_name ||
```

```

' ' || atr1 ||
CASE WHEN atr2 IS NOT NULL THEN ' ' || atr2 END ||
CASE WHEN atr3 IS NOT NULL THEN ' ' || atr3 END
|| ' UNIQUE ' || '(' || atr1 ||
CASE WHEN atr2 IS NOT NULL THEN ' ' || atr2 END ||
CASE WHEN atr3 IS NOT NULL THEN ' ' || atr3 END
|| ')'; AS "--Cheile alternative"
FROM user_constraints uc
LEFT OUTER JOIN
(SELECT constraint_name, column_name AS atr1
FROM user_cons_columns
WHERE position = 1) a1 ON uc.constraint_name = a1.constraint_name
LEFT OUTER JOIN
(SELECT constraint_name, column_name AS atr2
FROM user_cons_columns
WHERE position = 2) a2 ON uc.constraint_name = a2.constraint_name
LEFT OUTER JOIN
(SELECT constraint_name, column_name AS atr3
FROM user_cons_columns
WHERE position = 3) a3 ON uc.constraint_name = a3.constraint_name
WHERE constraint_type = 'U';

spool off

```

Structurile de tip CASE sunt limitate la trei coloane (atr1, atr2 și atr3), deoarece știm că cele mai lungi chei candidat din schema noastră au maxim trei atribute. Figura 4.31 arată conținutul fișierului ASCII re_creare_chei_primare_si_alternative.

```

re_creare_chei_primare_si_alternative - Notepad
File Edit Format View Help
--Stergere restrictie
-----
ALTER TABLE PERSONAL DROP CONSTRAINT PK_PERSONAL CASCADE ;
ALTER TABLE PERSONAL DROP CONSTRAINT UN_PERSONAL_NUMEPREN CASCADE ;
ALTER TABLE PONTAJE DROP CONSTRAINT PK_PONTAJE CASCADE ;
ALTER TABLE RETINERI DROP CONSTRAINT PK_RETINERI CASCADE ;
ALTER TABLE SALARII DROP CONSTRAINT PK_SALARII CASCADE ;
ALTER TABLE SPORURI DROP CONSTRAINT PK_SPORURI CASCADE ;
ALTER TABLE TRANSE_SV DROP CONSTRAINT PK_TRANSE_SV CASCADE ;

7 rows selected.

--Cheile primare
-----
ALTER TABLE PERSONAL ADD CONSTRAINT pk_PERSONAL PRIMARY KEY (MARCA) ;
ALTER TABLE PONTAJE ADD CONSTRAINT pk_PONTAJE PRIMARY KEY (MARCA, DATA) ;
ALTER TABLE SPORURI ADD CONSTRAINT pk_SPORURI PRIMARY KEY (MARCA, AN, LUNA) ;
ALTER TABLE RETINERI ADD CONSTRAINT pk_RETINERI PRIMARY KEY (MARCA, AN, LUNA) ;
ALTER TABLE SALARII ADD CONSTRAINT pk_SALARII PRIMARY KEY (MARCA, AN, LUNA) ;
ALTER TABLE TRANSE_SV ADD CONSTRAINT pk_TRANSE_SV PRIMARY KEY (ANI_LIMITA_INF) ;

6 rows selected.

--Cheile alternative
-----
ALTER TABLE PERSONAL ADD CONSTRAINT un_PERSONAL_NUMEPREN UNIQUE (NUMEPREN) ;

```

Figura 4.31. Fișierul text pentru refacerea cheilor primare și alternative

Atenție ! Denumirile restricțiilor nu trebuie să depășească 30 de caractere, așa încât ar trebui luate măsuri de siguranță (trunchierea unor eventuale denumiri mai lungi).

4.11.5. Reguli de validare la nivel de atribut și înregistrări

Trei chestiuni trebuie să avem în vedere atunci când extragem informații despre regulile de validare:

- dicționarul de date le stochează în tabele USER_CONSTRAINTS cu valoarea 'C' pentru atributul Constraint_Type;
- expresia ce constituie restricția propriu-zisă reprezintă valoarea coloanei Search_Condition care este, ați ghicit, de tip LONG;
- Oracle include în aceste reguli și pe cele de tip NOT NULL.

Așa încât scriptul din listing 4.11 refolosește tabela TEMP1 pentru conversia atributului LONG în CLOB.

Listing 4.11. Preluarea regulilor de validare

```
spool f:\oracle_cartel\cap04_creare_tabele\re_creare_check.sql

DROP TABLE temp1;

CREATE TABLE temp1 (
    restrictie VARCHAR2(30),
    tip_restrictie varchar(20),
    tabela VARCHAR2(30),
    expresie CLOB
);

INSERT INTO temp1
    SELECT constraint_name, constraint_type, table_name,
           TO_LOB(search_condition)
    FROM user_constraints
    WHERE constraint_type = 'C' ;

COMMIT ;

SELECT 'ALTER TABLE ' || tabela || ' DROP CONSTRAINT ' || restrictie || ';'
    AS "--Stergere reguli de validare"
FROM temp1
WHERE expresie NOT LIKE '%NULL%' ;

SELECT 'ALTER TABLE ' || tabela || ' ADD CONSTRAINT ' || restrictie ||
    ' CHECK (' || CAST (expresie AS VARCHAR2(300)) || ')'; AS "-- check-uri"
FROM temp1
WHERE expresie NOT LIKE '%NULL%' ;

spool off
```

4.11.6. Restricțiile referențiale

Cu restricțiile referențiale am ajuns la partea cea mai solicitantă din ceea ce ne-am propus în acest paragraf. Avem de identificat nu numai componența cheilor străine (din tabelele copil), ci și pe aceea a cheilor corespondente din tabele copil, iar atributele pot avea denumiri diferite. Dicționarul de date Oracle, prin tabela virtuală USER_CONSTRAINTS, furnizează atât numele restricției din tabela copil (constraint_name) pe baza căreia pot fi identificate atributele cheii străine, cât și numele restricției (primare/unice) corespunzătoare din tabela părinte (r_constraint_name).

Impresionantul script din listing 4.12 folosește în SELECT-ul principal două rânduri de joncțiuni externe la stânga: mai întâi, pentru extragerea câmpurilor din cheia străină, iar al doilea pentru atributele corespunzătoare din tabela copil.

Listing 4.12. Preluarea restricțiilor referențiale

```
spool f:\oracle_cartel\cap04_creare_tabele\re_creare_ref.sql

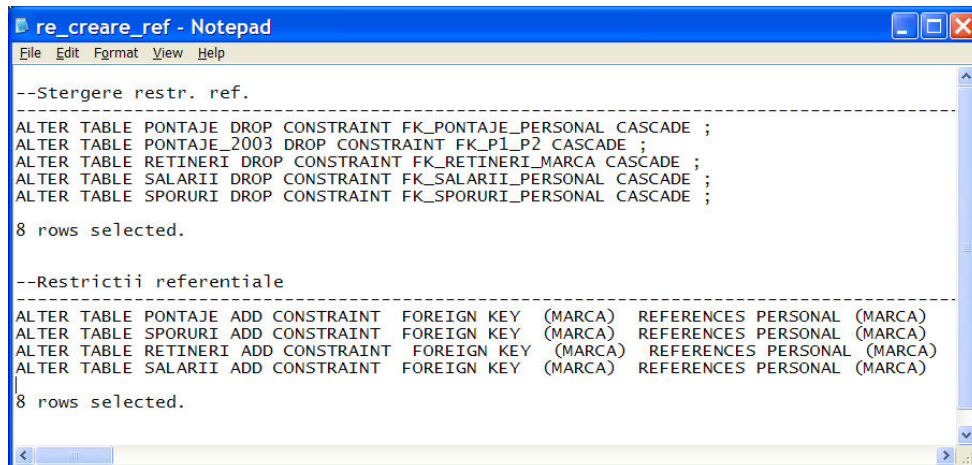
SELECT 'ALTER TABLE ' || table_name || ' DROP CONSTRAINT ' ||
       constraint_name || ' CASCADE ;' AS "--Stergere restr. ref."
FROM user_constraints
WHERE constraint_type = 'R' ;

SELECT 'ALTER TABLE ' || uc_copil.table_name || ' ADD CONSTRAINT ' || constraint_name ||
       ' FOREIGN KEY ' || '(' || atrc1 ||
       CASE WHEN atrc2 IS NOT NULL THEN ', ' || atrc2 END ||
       CASE WHEN atrc3 IS NOT NULL THEN ', ' || atrc3 END
       || ')' ||
       ' REFERENCES ' || uc_parinte.table_name || '(' || atrp1 ||
       CASE WHEN atrp2 IS NOT NULL THEN ', ' || atrp2 END ||
       CASE WHEN atrp3 IS NOT NULL THEN ', ' || atrp3 END
       || ')' AS "--Restricții referențiale"
FROM user_constraints uc_copil
     INNER JOIN user_constraints uc_parinte
           ON uc_copil.r_constraint_name = uc_parinte.constraint_name
     LEFT OUTER JOIN
       (SELECT constraint_name, column_name AS atrc1
        FROM user_cons_columns
        WHERE position = 1) c1 ON uc_copil.constraint_name = c1.constraint_name
     LEFT OUTER JOIN
       (SELECT constraint_name, column_name AS atrc2
        FROM user_cons_columns
        WHERE position = 2) c2 ON uc_copil.constraint_name = c2.constraint_name
     LEFT OUTER JOIN
       (SELECT constraint_name, column_name AS atrc3
        FROM user_cons_columns
        WHERE position = 3) c3 ON uc_copil.constraint_name = c3.constraint_name
     LEFT OUTER JOIN
       (SELECT constraint_name, column_name AS atrp1
        FROM user_cons_columns
        WHERE position = 1) p1 ON uc_parinte.constraint_name = p1.constraint_name
     LEFT OUTER JOIN
       (SELECT constraint_name, column_name AS atrp2
        FROM user_cons_columns
        WHERE position = 2) p2 ON uc_parinte.constraint_name = p2.constraint_name
     LEFT OUTER JOIN
```

```
(SELECT constraint_name, column_name AS atp3
FROM user_cons_columns
WHERE position = 3) p3 ON uc_parinte.constraint_name = p3.constraint_name
WHERE uc_copil.constraint_type = 'R';

spool off
```

Fișerul ASCII generat este prezentat în figura 4.32. Deși nu apare explicit în figură, scriptul funcționează și atunci când cheile străine sunt compuse, și atunci când restricțiile referențiale sunt recursive, adică tabela copil este simultan părinte (ex. ierarhiilor). După cum se observă, însă, am ținut cont că în schema noastră, o cheie primară (și, firește, străină) nu mai poate avea mai mult de trei atribute.



```
re_creare_ref - Notepad
File Edit Format View Help

--Stergere restr. ref.
-----
ALTER TABLE PONTAJE DROP CONSTRAINT FK_PONTAJE_PERSONAL CASCADE ;
ALTER TABLE PONTAJE_2003 DROP CONSTRAINT FK_P1_P2 CASCADE ;
ALTER TABLE RETINERI DROP CONSTRAINT FK_RETINERI_MARCA CASCADE ;
ALTER TABLE SALARII DROP CONSTRAINT FK_SALARII_PERSONAL CASCADE ;
ALTER TABLE SPORURI DROP CONSTRAINT FK_SPORURI_PERSONAL CASCADE ;

8 rows selected.

--Restricții referențiale
-----
ALTER TABLE PONTAJE ADD CONSTRAINT FOREIGN KEY (MARCA) REFERENCES PERSONAL (MARCA)
ALTER TABLE SPORURI ADD CONSTRAINT FOREIGN KEY (MARCA) REFERENCES PERSONAL (MARCA)
ALTER TABLE RETINERI ADD CONSTRAINT FOREIGN KEY (MARCA) REFERENCES PERSONAL (MARCA)
ALTER TABLE SALARII ADD CONSTRAINT FOREIGN KEY (MARCA) REFERENCES PERSONAL (MARCA)

8 rows selected.
```

Figura 4.32. Fișierul text pentru refacerea restricțiilor referențiale