

Capitolul 13. Tabele, restricții și actualizări prin interogări SQL

Schema mininală a unei baze de date a fost prezentată în capitolul 2, iar despre crearea tabelor, definirea atributelor și restricțiilor, precum și operațiunile de inserare, modificare și ștergere de linii am discutat în capitolul 3. Ceea ce aduce ca bonus acest capitol ține de folosirea consultărilor și subconsultărilor (mai mult sau mai puțin corelate) atât pentru definirea de restricții, cât și pentru actualizarea tabelor.

13.1. Crearea tabelor folosind interogări

Rezultatul unei interogări, adică al unei fraze SELECT, poate fi privit ca o tabelă anonimă, temporară, ad-hoc și non-actualizabilă. În aplicații, rezultatul trebuie să fie un set de înregistrări care constituie materia primă a unui raport, dar poate fi folosit și pentru a "popula" un combo-box sau o listă dintr-un formular s.a.m.d.

Despre crearea tabelor am discutat pe îndelete în capitolul 3. Din rațiuni didactice și de neimplementare în cele patru servere BD, nu am prezentat câteva opțiuni din SQL:2003 ale comenzii CREATE TABLE, cum ar fi:

- attribute generate de sistem (SYSTEM GENERATED), de utilizator (USER GENERATED) sau derivate (DERIVED);
- subtabele (clauza UNDER);
- crearea unei tabele pe baza definiției (plus/minus conținutul) unei alte tabele (LIKE).

După ce am deprins elementele esențiale ale interogărilor SQL, suntem în măsură să atacăm și câteva asemenea probleme. Astfel, dacă am vrea să creăm o tabelă cu numele FACTURI_2006 cu aceleași attribute ca ale tablei FACTURI, putem folosi comanda:

```
CREATE TABLE facturi_2006 LIKE facturi
```

Sintaxa este implementată în DB2, PostgreSQL-ul reclamând plasarea clauzei LIKE între paranteze:

```
CREATE TABLE facturi_2006 (LIKE facturi)
```

Noua tabelă va „moșteni” din sursă doar attributele, nu și restricțiile; în această privință standardul SQL:2003 și DB2 permit doar preluarea valorilor implicite (și a atributelor identitate):

```
DROP TABLE facturi_2006 ;
```

```
CREATE TABLE facturi_2006 LIKE facturi INCLUDING DEFAULTS ;
```

PostgreSQL este mai generos, întrucât formatul general permite preluarea atât a valorilor implicite, cât și a o serie de restricții:

```
DROP TABLE facturi_2006 ;  
CREATE TABLE facturi_2006 (LIKE facturi INCLUDING DEFAULTS  
INCLUDING CONSTRAINTS) ;
```

Clauza INCLUDING CONSTRAINTS poate să inducă în eroare: în realitate, numai restricțiile de tip CHECK sunt preluate în noua tabelă.

O altă modalitate de creare a unei tabele face apel la clauza AS. Iată sintaxa DB2:

```
DROP TABLE facturi_2006 ;  
CREATE TABLE facturi_2006 AS  
(SELECT *  
FROM facturi) DEFINITION ONLY ;
```

În Oracle, pentru a crea tabela cu atributele din sursă, sintaxa trebuie să includă clauza WHERE în care predicatul are valoarea logică *false*:

```
DROP TABLE facturi_2006 ;  
CREATE TABLE facturi_2006 AS  
(SELECT * FROM facturi WHERE 1=2 ) ;
```

În SQL Server comanda CREATE TABLE nu permite folosirea clauzei AS și a unei consultări; în schimb comanda SELECT prezintă clauza INTO:

```
SELECT *  
INTO facturi_2006  
FROM facturi WHERE 1=2
```

După cum îi spune și numele, FACTURI_2006 a fost gândită ca o tabelă-archivă în care să păstrăm toate facturile (înregistrările tabeli FACTURI) emise pe parcursul anului 2006, așa încât avem nevoie de conținut. Întrucât am creat structura (fără restricții), nu ne rămâne decât să adăugăm conținutul. Paragraful 13.3.1 este special dedicat folosirii subconsultărilor în comanda INSERT, așa că acum doar amintim comanda. Iată o sintaxă DB2/PostgreSQL/SQL Server:

```
INSERT INTO facturi_2006  
SELECT *  
FROM facturi  
WHERE DataFact BETWEEN '2006-01-01' AND '2006-12-31'
```

și o alta Oracle/PostgreSQL:

```
INSERT INTO facturi_2006  
SELECT *  
FROM facturi  
WHERE DataFact BETWEEN DATE'2006-01-01' AND DATE'2006-12-31'
```

Comanda CREATE TABLE permite în Oracle și PostgreSQL crearea tabelului FACTURI_2006 și popularea sa pot fi realizate dintr-o singură mișcare:

```
DROP TABLE facturi_2006 ;
CREATE TABLE facturi_2006 AS
  (SELECT * FROM facturi WHERE EXTRACT (YEAR FROM DataFact) = 2006
  );
```

În MS SQL Server comanda CREATE TABLE nu permite folosirea clauzei AS și a unei consultări; în schimb comanda SELECT prezintă clauza INTO:

```
SELECT *
INTO facturi_2006
FROM facturi
WHERE YEAR(DataFact) = 2006
```

Vă întrebați, firește, cât de frecvente sunt situațiile în care trebuie să creăm tabele pe baza altor tabele, prin subconsultări ? Prefer să ofer un răspuns evaziv, mai ales că pentru o parte dintre aceste cazuri (ex. furnizarea unui set de înregistrări pentru un raport, substituirea variabilelor publice (inexistente în unele SGBD-uri) etc.) pot fi găsite soluții mai bune bazate pe tabele temporare și/sau virtuale (vezi capitolul următor) sau proceduri stocate (vezi capitolul 16).

Noi începem cu o problemă suficient de realistă – arhivarea datelor de pe anii anteriori pentru tabele “trazaționale”, cum ar fi: FACTURI, LINIIFACT, INCASĂRI și ÎNCASFACT. Unii practicieni vor prefera ca arhivarea să fie făcută pe o structură denormalizată. De exemplu, în tabela FACTURI_2007 să includem în înregistrările tabelului FACTURI corespunzătoare anului calendaristic 2007 și trei atributele calculate care, ulterior, ne-ar scuti de o serie de joctiuni – *NrLinii*, *ValTotală* și *TVAFact* -, iar în LINIIFACT_2007 să adăugăm, *DenPr* și *TVALinie*. Zis și făcut.

În DB2 soluția ar putea arăta ca un script compus din trei comezi SQL, CREATE TABLE, INSERT și COMMIT:

```
CREATE TABLE facturi_2007 AS
  (SELECT f.*, COUNT(*) AS NrLinii,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValTotala,
        SUM(Cantitate * PretUnit * ProcTVA) AS TVAFact
  FROM facturi f
    INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
    INNER JOIN produse p ON lf.CodPr=p.CodPr
  WHERE YEAR(DataFact) = 2007
  GROUP BY f.NrFact, DataFact, CodCl, Obs
  ) DEFINITION ONLY ;
```

```

CREATE TABLE liniifact_2007 AS
  (SELECT lf.*, DenPr, UM, Cantitate * PretUnit * ProcTVA AS TVALinie
   FROM liniifact lf
        INNER JOIN produse p ON lf.CodPr=p.CodPr
   WHERE NrFact IN
        (SELECT NrFact FROM facturi
         WHERE YEAR(DataFact) = 2007)
   ) DEFINITION ONLY ;

INSERT INTO facturi_2007
  (SELECT f.*, COUNT(*) AS NrLinii,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValTotala,
        SUM(Cantitate * PretUnit * ProcTVA) AS TVAFact
   FROM facturi f
        INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
   WHERE YEAR(DataFact) = 2007
   GROUP BY f.NrFact, DataFact, CodCl, Obs
   ) ;

INSERT INTO liniifact_2007
  (SELECT lf.*, DenPr, UM, Cantitate * PretUnit * ProcTVA AS TVALinie
   FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
   WHERE NrFact IN
        (SELECT NrFact FROM facturi
         WHERE YEAR(DataFact) = 2007)
   ) ;

COMMIT ;

```

Soluția Oracle este comună cu cea PostgreSQL, creând și populând tabela dintr-o mișcare:

```

CREATE TABLE facturi_2007 AS
  (SELECT f.*, COUNT(*) AS NrLinii,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValTotala,
        SUM(Cantitate * PretUnit * ProcTVA) AS TVAFact
   FROM facturi f
        INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
   WHERE EXTRACT (YEAR FROM DataFact) = 2007
   GROUP BY f.NrFact, DataFact, CodCl, Obs
   );

CREATE TABLE liniifact_2007 AS

```

```

        (SELECT lf.*, DenPr, UM, Cantitate * PretUnit * ProcTVA AS TVALinie
        FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
        WHERE NrFact IN
            (SELECT NrFact FROM facturi
            WHERE EXTRACT (YEAR FROM DataFact) = 2007
            )
        )

```

Echivalenta acestor soluții care este funcționabilă în SQL Server se prezintă astfel:

```

SELECT f.*, COUNT(*) AS NrLinii,
        SUM(Cantitate * PretUnit * (1+ProcTVA)) AS ValTotala,
        SUM(Cantitate * PretUnit * ProcTVA) AS TVAFact
INTO facturi_2007
FROM facturi f
        INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE YEAR(DataFact) = 2007
GROUP BY f.NrFact, DataFact, CodCl, Obs ;

SELECT lf.*, DenPr, UM, Cantitate * PretUnit * ProcTVA AS TVALinie
INTO liniifact_2007
FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE NrFact IN
        (SELECT NrFact
        FROM facturi
        WHERE YEAR(DataFact) = 2007
        ) ;

```

Pentru a face lucrurile un pic mai interesante, în problema traseelor introducăm un orar al curselor. Modificăm tabela DISTANTE introducând un atribut Durata de tip INTERVAL DAY TO SECOND (în Oracle și PostgreSQL, singurele care acceptă tipuri INTERVAL) care va păstra minutele și secunde necesare parcurgerii distanțelor dintre două localități. Această durată depinde nu numai de distanța efectivă, ci și de limitele de viteză și, nu în ultimul rând, de numărul de gropi (număr rotunjit la ordinul sutelor). Apoi ștergem și repopulăm tabele – vezi listing 13.1.

Listing 13.1. Modificarea structurii și re-popularea tabelii DISTANTE

```

ALTER TABLE distante ADD (Durata INTERVAL DAY TO SECOND )1;
DELETE FROM distante ;
INSERT INTO distante VALUES ('Iasi', 'Vaslui', 71, INTERVAL '1:25' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Iasi', 'Tg.Frumos', 53, INTERVAL '1:05' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Tg.Frumos', 'Roman', 40, INTERVAL '0:50' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Roman', 'Bacau', 41, INTERVAL '0:47' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Roman', 'Vaslui', 80, INTERVAL '1:50' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Vaslui', 'Birlad', 54, INTERVAL '0:53' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Birlad', 'Tecuci', 48, INTERVAL '0:47' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Tecuci', 'Tisita', 20, INTERVAL '0:18' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Tisita', 'Focsani', 13, INTERVAL '0:15' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Bacau', 'Adjud', 60, INTERVAL '0:50' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Bacau', 'Vaslui', 85, INTERVAL '2:05' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Adjud', 'Tisita', 32, INTERVAL '0:40' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Focsani', 'Galati', 80, INTERVAL '1:50' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Focsani', 'Braila', 86, INTERVAL '1:57' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Focsani', 'Rm.Sarat', 42, INTERVAL '0:50' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Rm.Sarat', 'Braila', 84, INTERVAL '1:55' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Braila', 'Galati', 11, INTERVAL '0:25' HOUR TO MINUTE) ;
INSERT INTO distante VALUES ('Tecuci', 'Galati', 67, INTERVAL '1:09' HOUR TO MINUTE) ;

```

Aflarea numărului de minute necesare pentru a ajunge la fiecare dintre localitățile (stațiile) de pe ruta Iași-Bârlad-Tecuci-Focșani ar presupune în Oracle execuția unei interogări de genul:

```

WITH tab AS (
    SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
           LEVEL AS Nivel, d.*, ROWNUM AS Ord FROM distante d
    START WITH Loc1='Iasi'
    CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 20
    ORDER BY Ord)
SELECT DISTINCT t2.traseu, t2.Nivel AS LocNr, t2.Loc2 ,
       (SELECT SUM(Durata)
        FROM tab
        WHERE t2.Traseu LIKE Traseu || '%' ) AS Km
FROM tab t1 INNER JOIN tab t2 ON t1.Traseu LIKE '**Iasi**Vaslui%Focsani'
AND t1.traseu LIKE t2.Traseu || '%'
ORDER BY 1,2

```

¹ În DB2 și MS SQL Server, comanda ALTER TABLE va avea forma: *ALTER TABLE distante ADD Durata NUMERIC(5)*, iar în comenzile INSERT durata va fi exprimată printr-un număr întreg ce reprezintă minutele; sintaxa din PostgreSQL este: *ALTER TABLE distante ADD Durata INTERVAL DAY TO SECOND* (INSERT-urile sunt identice celor din Oracle/PostgreSQL)

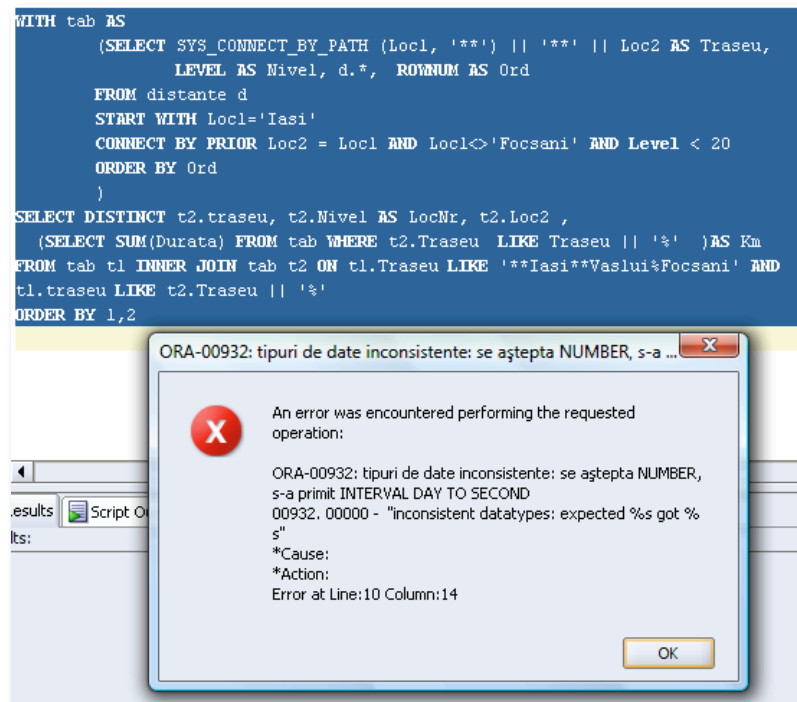


Figura 13.1. Funcția SUM nu poate fi aplicată (în Oracle) unui atribut de tip INTERVAL

Din păcate, Oracle nu știe să adune intervale cu funcția SUM din dotare – vezi mesajul din figura 13.1 care ne spune că SUM-ul se aștepta la un argument numeric, nu la surpriza-supriza noastră. De aceea suntem nevoiți să improvizăm un pic, convertind distanța într-un număr întreg de minute, după celebra formulă $NrTotalMinute = NrOre * 60 + NrMinute$:

```
WITH tab AS
  (SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
        LEVEL AS Nivel, d.*, ROWNUM AS Ord
   FROM distante d
   START WITH Loc1='Iasi'
   CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 20
   ORDER BY Ord
  )
SELECT DISTINCT t2.traseu, t2.Nivel AS LocNr, t2.Loc2 ,
  (SELECT SUM( Durata) FROM tab WHERE t2.Traseu LIKE Traseu || '%' )AS Km
FROM tab t1 INNER JOIN tab t2 ON t1.Traseu LIKE '**Iasi**Vaslui%Focsani' AND
t1.traseu LIKE t2.Traseu || '%'
ORDER BY 1,2
```

TRASEU	LOCNR	LOC2	MINUTE
IasiVaslui	1	Vaslui	85
IasiVaslui**Birlad	2	Birlad	138
IasiVaslui**Birlad**Tecuci	3	Tecuci	185
IasiVaslui**Birlad**Tecuci**Tisita	4	Tisita	203
IasiVaslui**Birlad**Tecuci**Tisita**Focsani	5	Focsani	218

Figura 13.2. Numărul de minute de la plecare până la fiecare stație de pe ruta Iași-Bârlad-Focșani

Rezultatul (vezi figura 13.2) ne pregătește pentru ceea ce urmează – crearea unei tabele în care, pentru început, vom stoca orarul unei curse Iași-Focșani care pleacă la ora 9:05 din Iași, pe data de 1 aprilie 2008. Întrucât clauza INTERVAL poate folosi doar literalii, iar noi trebuie să ținem cont de valorile nou-introdusului atribut Durata, pentru a calcula orele de sosire/plecare în fiecare stație suntem nevoiți să recurgem la funcția TO_DSINTERVAL:

WITH tab AS

```
(SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
      LEVEL AS Nivel, d.*, ROWNUM AS Ord
FROM distante d
START WITH Loc1='Iasi'
CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 20
ORDER BY Ord
)
```

```
SELECT 1 AS "StatieNr", 'Iasi' AS "Statie",
      TIMESTAMP'2008-04-01 09:05:00' AS "Data/Ora" FROM dual
```

UNION

```
SELECT t2.Nivel + 1, t2.Loc2, TIMESTAMP'2008-04-01 09:05:00' +
      TO_DSINTERVAL ('0 ' || FLOOR(
      (SELECT SUM( EXTRACT (HOUR FROM Durata) * 60 + EXTRACT (MINUTE
      FROM durata) ) FROM tab WHERE t2.Traseu LIKE Traseu || '%' )
      / 60)
      || ':' ||
      MOD ( (SELECT SUM( EXTRACT (HOUR FROM Durata) * 60 +
      EXTRACT (MINUTE FROM durata) ) FROM tab WHERE t2.Traseu
      LIKE Traseu || '%' ), 60)
      || ':00')
FROM tab t1 INNER JOIN tab t2 ON t1.Traseu LIKE '**Iasi**Vaslui%Focsani'
AND t1.traseu LIKE t2.Traseu || '%'
ORDER BY 1,2
```


Din păcate, folosind CREATE TABLE AS (SELECT...) nu mai putem recurge la expresia tabelă, ceea ce ne complică sintaxa comenzii. Vom începe, totuși prin a crea o tabelă-nomenclator al curselor în care introducem o linie:

```
CREATE TABLE curse AS (
    SELECT 10001 AS IdCursa, Traseu,
           TIMESTAMP'2008-04-01 09:05:00' AS DataOra_Plecare
    FROM
        (SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' ||
          Loc2 AS Traseu, LEVEL AS Nivel, d.*, ROWNUM AS Ord
         FROM distante d
         START WITH Loc1='Iasi'
         CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani'
         AND Level < 20
         ORDER BY Ord ) tab
    WHERE Traseu LIKE '**Iasi**Vaslui%Focsani'
) ;
```

Pentru a nu fi nevoiți să mă credeți (doar) pe cuvânt, figura 13.3 conține linia cu pricina. În continuare, trecem la miezul problemei, propunându-ne să creăm tabela CURSE_STATII în care vor fi introduse toate localitățile și orele corespunzătoare de oprire ale cursei 10001. Faptul de dispunem de tabela CURSE ne simplifică, în oarecare măsură, interogarea:

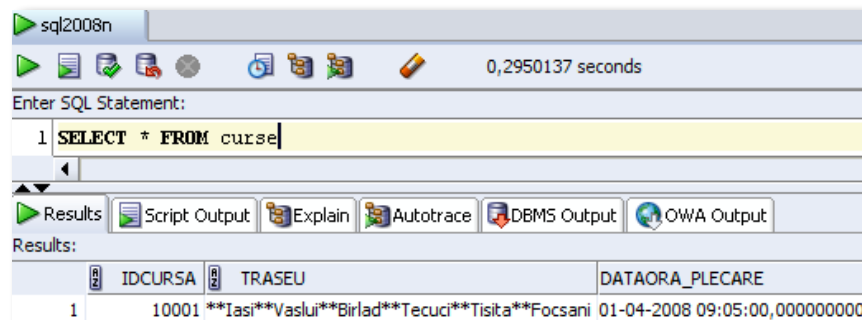


Figura 13.3. Prima (și singura) linie din proaspăt creată tabelă CURSE

```
CREATE TABLE curse_statii AS (
    SELECT IdCursa, 1 AS "StatieNr", 'Iasi' AS Statie,
           DataOra_Plecare AS Data_Ora
    FROM curse
    WHERE IdCursa=10001
    UNION
    SELECT IdCursa, Nivel + 1, Loc2, DataOra_Plecare + TO_DSINTERVAL ('0 ' ||
      FLOOR( (SELECT SUM( EXTRACT (HOUR FROM Durata) * 60 +
```

```

        EXTRACT (MINUTE FROM durata) ) FROM
        (
        SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
            LEVEL AS Nivel, d.*, ROWNUM AS Ord
        FROM distante d
        START WITH Loc1='Iasi'
        CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 20
        ORDER BY Ord
        ) tab1
        WHERE tab.Traseu LIKE tab1.Traseu || '%' ) / 60)
    || ':' ||
MOD ( (SELECT SUM( EXTRACT (HOUR FROM Durata) * 60 +
        EXTRACT (MINUTE FROM durata) ) FROM
        (
        SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2
            AS Traseu, LEVEL AS Nivel, d.*, ROWNUM AS Ord
        FROM distante d
        START WITH Loc1='Iasi'
        CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND
            Level < 20
        ORDER BY Ord
        ) tab2
        WHERE tab.Traseu LIKE tab2.Traseu || '%' ), 60)
    || ':00')
FROM
    (SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' || Loc2 AS Traseu,
        LEVEL AS Nivel, d.*, ROWNUM AS Ord
    FROM distante d
    START WITH Loc1='Iasi'
    CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani' AND Level < 20
    ORDER BY Ord
    ) tab INNER JOIN curse ON curse.Traseu LIKE tab.Traseu || '%'
)

```

Figura 13.4 ne convinge că efortul fizic și intelectual depus pentru redactarea acestei comenzi a meritat.

IDCURSA	StatieNr	STATIE	DATA_ORA
10001	1	Iasi	01-04-2008 09:05:00
10001	2	Vaslui	01-04-2008 10:30:00
10001	3	Birlad	01-04-2008 11:23:00
10001	4	Tecuci	01-04-2008 12:10:00
10001	5	Tisita	01-04-2008 12:28:00
10001	6	Focsani	01-04-2008 12:43:00

Figura 13.4. Stațiile și ora de sosire/plecare pentru cursa Iași-Focșani din 1 aprilie 2008

În SQL Server, rezultatul din figura 13.2 se obține prin interogarea:

```

WITH ierarhie (Cale, Nivel, Durata, Loc2) AS (
    SELECT CAST(Loc1 + ' -> ' + Loc2 AS VARCHAR(500)), 0 AS Nivel,
           CAST (Durata AS NUMERIC), Loc2
    FROM distante
    WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.Cale + ' -> ' + d.Loc2 AS VARCHAR(500)),
           Nivel + 1, CAST (i.Durata+d.Durata AS NUMERIC), d.Loc2
    FROM ierarhie i INNER JOIN distante d ON i.Loc2=d.Loc1
    WHERE Nivel < 20)
SELECT i2.Cale, i2.Nivel + 1 AS LocNr, i2.Loc2, i2.Durata AS Minute
FROM ierarhie i1 INNER JOIN ierarhie i2 ON i1.Cale LIKE 'Iasi -> Vaslui%Focsani' AND
      i1.Cale LIKE i2.Cale + '%'
ORDER BY 1,2

```

Cele două tabele, CURSE și CURSE_STAȚII se obțin printr-o interogare ceva mai puțin laborioasă decât cea din Oracle:

```

WITH ierarhie (Cale, Nivel, Durata, Loc2) AS (
    SELECT CAST(Loc1 + '**' + Loc2 AS VARCHAR(500)), 0 AS Nivel,
           CAST (Durata AS NUMERIC), Loc2 FROM distante WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.Cale + '**' + d.Loc2 AS VARCHAR(500)),
           Nivel + 1, CAST (i.Durata+d.Durata AS NUMERIC), d.Loc2
    FROM ierarhie i INNER JOIN distante d ON i.Loc2=d.Loc1
    WHERE Nivel < 20)
SELECT 1001 AS IdCursa, Cale AS Traseu,
       CAST ('2008-04-01 09:05:00' AS DATETIME) AS DataOra_Plecare
INTO curse
FROM ierarhie
WHERE Cale LIKE 'Iasi**Vaslui%Focsani' ;

```

```

WITH ierarhie (Cale, Nivel, Durata, Loc2) AS (
    SELECT CAST(Loc1 + '**' + Loc2 AS VARCHAR(500)), 0 AS Nivel,
           CAST (Durata AS NUMERIC), Loc2
    FROM distante WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.Cale + '**' + d.Loc2 AS VARCHAR(500)),
           Nivel + 1, CAST (i.Durata+d.Durata AS NUMERIC), d.Loc2
    FROM ierarhie i INNER JOIN distante d ON i.Loc2=d.Loc1
    WHERE Nivel < 20)
SELECT IdCursa, 1 AS StatieNr, 'Iasi' AS Statie, DataOra_Plecare AS DataOra
INTO curse_statii
FROM curse WHERE IdCursa=1001
UNION
SELECT IdCursa, Nivel + 1 AS StatieNr, Loc2,
       DATEADD(MINUTE, Durata, DataOra_Plecare)
FROM ierarhie INNER JOIN curse ON curse.Traseu LIKE ierarhie.Cale + '%'
ORDER BY 1,2

```

Beneficiind de experiența DB2 în materie de interogări ierarhice dobândită în paragraful 12.4 suntem în stare să formulăm interogarea pentru obținerea rezultatului din figura 13.2:

```

WITH ierarhie (Cale, Nivel, Durata, Loc2) AS(
    SELECT CAST(Loc1 || '-'>' || Loc2 AS VARCHAR(500)), 0 AS Nivel,
           CAST (Durata AS NUMERIC), Loc2
    FROM distante
    WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.cale || '-'>' || d.Loc2 AS VARCHAR(500)),
           Nivel + 1, CAST (i.durata+d.durata AS NUMERIC), d.loc2
    FROM ierarhie i, distante d
    WHERE i.Loc2=d.Loc1 AND Nivel < 20)
SELECT i2.cale, i2.nivel + 1 AS LocNr, i2.Loc2, i2.Durata AS Minute
FROM ierarhie i1 INNER JOIN ierarhie i2
    ON i1.Cale LIKE 'Iasi -> Vaslui%Focsani' AND
       SUBSTR(i1.Cale,1,LENGTH(i2.Cale)) = i2.Cale
ORDER BY 1,2

```

Ne împotmolim, însă, atunci când dorim crearea celor două tabele – CURSE și CURSE_STAȚII – creare ce am vrea să coincidă cu popularea. Deocamdată nu

putem decât defini structura celor două tabele, urmând să încercăm popularea în paragraful 13.3.1.

```
CREATE TABLE curse AS (
    SELECT 1001 AS IdCursa, CAST (' ' AS VARCHAR(200)) AS Traseu,
           CAST ('2008-04-01 09:05:00' AS TIMESTAMP) AS DataOra_Plecare
    FROM sysibm.sysdummy1
)      DEFINITION ONLY ;

CREATE TABLE curse_statii AS (
    SELECT 1001 AS IdCursa, 1 AS StatieNr, CAST (' ' AS VARCHAR(30)) AS Statie,
           CAST ('2008-04-01 09:05:00' AS TIMESTAMP) AS DataOra
    FROM sysibm.sysdummy1
)      DEFINITION ONLY ;
```

În PostgreSQL, funcția CONNECTBY() are sintaxa ceva mai rigidă, așa că, pentru a ne mai ușura munca, creăm o tabelă RUTE ce conține traseele posibile dintre Iași și orice altă localitate:

```
CREATE TABLE rute AS
    SELECT ierarhie.Loc2 AS Loc1, ierarhie.Loc1 AS Loc2, Distanta,
           Durata, ierarhie.Cale, LEVEL AS Nivel
    FROM ( SELECT *
            FROM CONNECTBY('distanta', 'Loc2', 'Loc1', 'Iasi', 0, '**')
            AS t2 (Loc1 TEXT, Loc2 TEXT, LEVEL INT, cale text)
          ) ierarhie, distante d
WHERE ierarhie.Loc2 =d.Loc1 AND ierarhie.Loc1=d.Loc2 ;
```

Pe baza acestei tabele, ne simplificăm crearea următoarelor două tabele, CURSE și CURSE_STAȚII.

```
CREATE TABLE curse AS (
    SELECT 10001 AS IdCursa, Cale AS Traseu, TIMESTAMP'2008-04-01 09:05:00'
           AS DataOra_Plecare
    FROM rute
    WHERE Cale LIKE 'Iasi**Vaslui%Focsani'
) ;
```

Înainte de treceri la crearea tablei CURSE_STAȚII trebuie să vă spun că gestiunea valorilor de tip INTERVAL este, în PostgreSQL, superioară Oracle ! Putem să însumăm fără nicio grijă duratele:

```
CREATE TABLE curse_statii AS (
    SELECT IdCursa, 1 AS "StatieNr", 'Iasi' AS Statie,
           DataOra_Plecare AS Data_Ora
    FROM curse
    WHERE IdCursa=10001
    UNION
```

```

SELECT IdCursa, Nivel + 1, Loc2, DataOra_Plecare + (
    SELECT SUM(Durata)
    FROM rute r2
    WHERE curse.Traseu LIKE r2.Cale || '%' AND Nivel <= rute.Nivel)
FROM curse INNER JOIN rute ON curse.Traseu LIKE rute.Cale || '%'
WHERE IdCursa=10001
ORDER BY 2
);

```

13.2. Restricții și aserțiuni pe bază de interogări

Posibilitatea de a crea restricții la nivel de înregistrare folosind fraze SELECT, care să returneze valoarea logică *adevărat* (TRUE) atunci când regula este respectată, și *fals* (FALSE) când regula este încălcată, a apărut în SQL din versiunea SQL-92. Joe Celko a cheltuit cu acest subiect destule pagini în mai toate cărțile sale dedicate SQL², iar Melton și Simon pomenesc, și ei, de această facilități³. După cum lăsam să se înțeleagă din capitolele 2 și 3, restricțiile trebuie gândite, pe cât posibil, în faza de proiectare a aplicației, și nu după ce avem date reale preluate. Aceasta deoarece adăugarea ulterioară a unor restricții poate să intre în contradicție cu informațiile existente în bază la momentul declarării restricției. Noi însă vom trece sub tăcere acest neajuns.

De exemplu, înstituiim regula ca *nu facem nicio vânzare vreunui client dacă nu avem în baza de date măcar o persoană de contact pentru acel client*. Vânzare înseamnă facturare (FACTURI), iar persoanele de contact sunt în tabela PERSCLIENTI. Regula poate fi declarată pentru atributul CodCl din FACTURI după cum urmează:

```

ALTER TABLE facturi ADD CONSTRAINT ck_facturi_codcl CHECK (
    1 <= ALL (
        SELECT COUNT(persclienti.CodCl)
        FROM
            (SELECT CodCl
             FROM facturi
             GROUP BY CodCl) clienti_fact
        LEFT OUTER JOIN

```

² În primul rând în [Celko 2000]

³ [Melton & Simon 2002], pp.370-375

```

        perscienti ON clienti_fact.CodCl=perscienti.CodCl
    GROUP BY clienti_fact.CodCl
)
)

```

Ar fi fost minunat să existe un mecanism prin care să luăm în calcul numai clientul din factura curentă (inserată sau modificată), numai că standardul (sau, cel puțin, Celko, Melton & Simon, și alții) nu face trimitere la o asemenea posibilitate, întrucât SQL-ul este un limbaj orientat pe seturi, noțiunea de *linie curentă* fiind în disonanță cu preceptele modelului relațional.

Continuăm cu un exemplu care pare mai ciudat: *în tabela PERSONAL2 musai trebuie să existe nici mai mult, nici mai puțin, de un director general*. Directorul general este cel care are NULL ca valoare a atributului MarcaSef. Iată care ar putea fi restricția:

```

ALTER TABLE personal2 ADD CONSTRAINT il_capo_di_tutti_capi CHECK (
    1 = ALL ( SELECT COUNT(*) FROM personal2 WHERE MarcaSef IS NULL )
)

```

Ne întrebăm ce se întâmplă dacă, urmând sfatul înțelepților, declarăm această restricție la crearea tabelului PERSONAL2 când, fatalitate!, nu există nicio linie (deci, nici un șef) în tabelă. Ei bine, nu-i bai ! Restricția de mai sus trebuie respectată de conținutul tabelului, conținut care, la momentul creării tabelului, nu există. Pentru a nu avea, totuși, necazuri, prima linie care va fi inserată în PERSONAL2 va fi cea a lui *el lider maximo*.

Uneori declararea unei reguli sub forma restricției de tip CHECK este destul de anevoioasă, motiv pentru care standardul SQL a introdus, însă din versiunea SQL-92 noțiunea de *asertiune*. La modul simplist, o asertiune este o regulă definită în baza de date, dar care nu este legată de o tabelă anume. De exemplu, firma pentru care implementăm baza de date destinată vânzărilor, instituie regula ca *niciunui client să nu îi trimitem mai mult de 10 facturi pe an, iar dacă soldul vreunui client (diferența dintre valoarea facturilor destinate acelui client și valoarea încasărilor de la acel client) este mai mare de, să zicem, 500 000 RON, atunci toate vânzările (facturile) se blochează*. Regula este cam dură, ca să nu spunem aiurită, dar, ca IT-iști, suntem doar servanți (dacă eram mai răsăriți, ne „făceam” *manageri*!). Iată care ar fi asertiunea ce ar implementa această regulă:

```

CREATE ASSERTION blocaj_facturare CHECK ( 500000 <= ALL (
    SELECT facturat - COALESCE(Incasat,0)
    FROM (
        SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat
        FROM facturi f INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
        GROUP BY CodCl
    ) fact LEFT OUTER JOIN
    (SELECT CodCl, SUM(Transa) AS Incasat
    FROM facturi f

```

```

INNER JOIN incasfact i ON f.NrFact=i.NrFact
GROUP BY CodCl) incas ON fact.CodCl=incas.CodCl
)

```

Chiar dacă problemele de implementare nu sunt o sursă de anxietate pentru cei care redactează specificațiile standardelor SQL, vă imaginați că pentru respectarea, în orice moment, a unei reguli precum aceasta, consumul de resurse va fi unui destul de mare, mai ales după câțiva ani și câteva milioane de înregistrări în LINIIFACT.

Unele restricțiile ridică temeri de alt gen. Să presupunem, de exemplu, că în tabela SPORURI numărul de linii din orice lună calendaristică trebuie să corespundă exact numărului de angajați (din tabela PERSONAL2) ai firmei în luna respectivă. Este un gen de problemă la care structura celor două tabele nu ne ajută prea mult, întrucât nu dispunem de atribute care să indice data angajării și data ieșirii din firmă ale fiecărui angajat. Unde mai punem că sunt persoane care se reîntorc în firmă după vreun „stagiu” în Italia, Spania sau orice țară aflată mai la vest. Putem însă reformula problema astfel: *numărul de înregistrări din tabela SPORURI pentru cea mai recentă lună calendaristică trebuie să fie egal cu numărul angajaților din tabela PERSONAL2*. Comanda de declarare a aserțiunii ar fi:

```

CREATE ASSERTION verificare_nr_angajati CHECK
(
    (SELECT COUNT(*)
     FROM sporuri
     WHERE (An || Luna) = (SELECT MAX(An || Luna) FROM sporuri)
    ) =
    (SELECT COUNT(*) FROM personal2)
)

```

Chiar și așa, știm că liniile se introduc pe rând în tabela SPORURI, deci aserțiunea pare condamnată din start. Ei bine, ca orice tip de restricție, și aserțiunea poate fi amânată (DEFERRED) până la finalul unei tranzacții. Prin urmare, dacă inserările s-ar face într-o singură tranzacție, am fi „asigurați”.

Din păcate, deși generoase în intenții, nici regulile de validare bazate pe consultări, nici aserțiunile nu sunt implementare în vreunul dintre cele patru produse discutate în această carte. Acum vă întrebați, de ce am cheltuit timpul cu ele. Ei bine, nu numai ca să mai treacă timpul, ci și pentru a discuta modalitățile de implementare ale acestui gen de reguli în capitolul dedicat declanșatoarelor (cap. 17). Vom putea, atunci (deși nu o s-o facem) implementa restricții dintre cele mai variate, cum ar fi:

- nu putem vinde mai mult de 15 buc din produsul 2 într-un an;
- o factură nu poate avea mai mult de zece linii;
- într-o gardă trebuie să fie, întotdeauna, un singur medic;
- cursă nu poate ajunge într-o stație dacă nu a trecut prin stația precedentă (de pe traseul respectiv).

PostgreSQL pune la dispoziție un mecanism destul de interesant prin care pot fi definite reguli (RULES) pentru tabele și tabele virtuale. Vom discuta acest mecanism în capitolul viitor (îl vom aplica asupra unor tabele virtuale).

13.3. Actualizarea tabelelor prin subconsultări

În materie de actualizări (INSERT-uri, UPDATE-uri, DELETE-uri) pe bază de interogări SQL-ul ne răsfătă, chiar și în condițiile în care tabelele pe care le-am folosit până acum în această carte nu au o structură din cale-afară de spectaculoasă, iar exemplele nu pot fi prea impresonante.

13.3.1. INSERT-uri ce folosesc comenzi SELECT

Pentru a verifica „comportamentul” interogărilor SQL, inclusiv resursele consumate, este recomandabil să populăm tabelele bazei cu cât mai multe înregistrări. Aici apare un obstacol insurmontabil – lenea – deghizat onorabil în lipsă de timp. În cartea dedicată Oracle 9i⁴, am întors pe câte fețe am putut tabelele legate de salarizare (PERSONAL, PONTAJE, SPORURI și RETINERI). Pe același calapod putem să inserăm în mult mai modesta tabelă SPORURI înregistrări pentru luna august 2007, câte o înregistrare „din oficiu” pentru fiecare angajat. Sintaxa următoare este comună celor patru dialecte SQL:

```
INSERT INTO sporuri (marca, an, luna)
SELECT marca, 2007, 8
FROM personal2
```

Complicându-ne un pic situația, ne propunem să inserăm pentru luna noiembrie 2007 facturile din august 2007, cu unele amendamente. Mai întâi, numărul trebuie să fie unic, așa ca vom incrementa valorile NrFact cu 4000. Apoi, vom elimina din discuție facturile care, în noiembrie 2007, ar fi întocmite sâmbăta și duminica. Soluția în dialectul Oracle ar fi:

```
INSERT INTO facturi (
SELECT 4000+NrFact, DataFact+ INTERVAL '3' MONTH, CodCl, Obs
FROM facturi
WHERE EXTRACT(YEAR FROM DataFact)=2007 AND
```

⁴ [Fotache s.a. 2003]

```

EXTRACT(MONTH FROM DataFact)=8 AND
RTRIM(TO_CHAR(DataFact+ INTERVAL '3' MONTH, 'DAY'))
NOT IN ('SATURDAY', 'SUNDAY')
)

```

Firește, înregistrările de mai sus nu ne folosesc prea mult dacă nu introducem și liniile acestor facturi. Cu acest prilej, exemplificăm folosirea unei interogări ca argument al clauzei INTO:

```

INSERT INTO (SELECT NrFact, Linie, CodPr, Cantitate, PretUnit
FROM liniifact)
(SELECT 4000+NrFact, Linie, CodPr, Cantitate, PretUnit
FROM liniifact
WHERE NrFact IN (SELECT NrFact-4000 FROM facturi
WHERE NrFact > 5000)
)

```

Folosirea unei interogări-argument în clauza INTO este utilă cu precădere atunci când, prin SELECT-ul care desemnează liniile inserate, se preiau numai valorile câtorva dintre atribute.

Pentru luna decembrie (2007), ne propunem ceva mai ambițios. Tocmai pentru a nu avea aceleași vânzări precum una dintre lunile anterioare, vom „cumula” facturile din lunile august, septembrie și octombrie. Din data “originală” a fiecărei facturi vom prelua numai ziua; astfel 4 august, 4 septembrie și 4 octombrie vor deveni, fiecare, 4 decembrie. Numărătoarea va începe cu 7001, iar numerele vor fi consecutive – vezi figura 13.5 (soluție Oracle):

```

INSERT INTO facturi
(SELECT 7000 + Poz AS "Noul NrFact",
CASE EXTRACT (MONTH FROM DataFact)
WHEN 8 THEN DataFact+INTERVAL '4' MONTH
WHEN 9 THEN DataFact+INTERVAL '3' MONTH
WHEN 10 THEN DataFact+INTERVAL '2' MONTH
END AS "Noua DataFact", CodCl, Obs
FROM
(SELECT NrFact, RANK() OVER
(ORDER BY EXTRACT(DAY FROM DataFact), NrFact) AS Poz
FROM facturi
WHERE EXTRACT(YEAR FROM DataFact)=2007 AND
EXTRACT(MONTH FROM DataFact) IN (8,9,10)
) coresp INNER JOIN facturi f ON coresp.NrFact=f.NrFact AND
EXTRACT(YEAR FROM DataFact)=2007
AND EXTRACT(MONTH FROM DataFact) IN (8,9,10)
)

```

NrFact vechi	DataFact veche	Ordine	Noul NrFact	Noua DataFact
1111	01-08-2007	1	7001	01-12-2007
1112	01-08-2007	2	7002	01-12-2007
1113	01-08-2007	3	7003	01-12-2007
1114	01-08-2007	4	7004	01-12-2007
3111	01-09-2007	5	7005	01-12-2007
3112	01-09-2007	6	7006	01-12-2007
1115	02-08-2007	7	7007	02-12-2007
1116	02-08-2007	8	7008	02-12-2007
3113	02-09-2007	9	7009	02-12-2007
3115	02-09-2007	10	7010	02-12-2007
1117	03-08-2007	11	7011	03-12-2007
1118	04-08-2007	12	7012	04-12-2007
1119	07-08-2007	13	7013	07-12-2007
1120	07-08-2007	14	7014	07-12-2007
1121	07-08-2007	15	7015	07-12-2007
1122	07-08-2007	16	7016	07-12-2007
3119	07-09-2007	17	7017	07-12-2007
3116	10-09-2007	18	7018	10-12-2007

Figura 13.5. Renumerotarea facturilor din lunile august, septembrie și octombrie 2007

Inserarea rândurilor (liniilor) fiecăreia dintre facturi se realizează astfel:

```
INSERT INTO liniifact (
    SELECT 7000 + Poz AS "Noul NrFact", Linie, CodPr, Cantitate, PretUnit
    FROM
        (SELECT NrFact, RANK() OVER (ORDER BY
            EXTRACT(DAY FROM DataFact), NrFact) AS Poz
        FROM facturi
        WHERE EXTRACT(YEAR FROM DataFact)=2007
            AND EXTRACT(MONTH FROM DataFact) IN (8,9,10)
        ) coresp INNER JOIN facturi f ON coresp.NrFact=f.NrFact AND
            EXTRACT(YEAR FROM DataFact)=2007 AND
            EXTRACT(MONTH FROM DataFact) IN (8,9,10)
        INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
    )
```

Am văzut în primul paragraf al acestui capitol ca DB2 este destul de necooperant în materie de creare a tabelelor prin interogări. Este motivul pentru care CREATE TABLE AS poate fi folosită numai împreună cu *DEFINITION ONLY*, prin urmare tabela poate fi creată printr-un SELECT, dar fără înregistrări. Așa că amânam popularea pentru acest paragraf, bănuind că am putea folosi în comanda INSERT un SELECT care să folosească clauza WITH. Ei, bine, interogările următoare punctează serios la capitolul imagine:

```
INSERT INTO curse
```

```

WITH ierarhie (Cale, Nivel, Durata, Loc2) AS (
    SELECT CAST(Loc1 || '***' || Loc2 AS VARCHAR(500)), 0 AS Nivel,
           CAST (Durata AS NUMERIC), Loc2
    FROM distante WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.cale || '***' || d.Loc2 AS VARCHAR(500)),
           Nivel + 1, CAST (i.durata+d.durata AS NUMERIC), d.loc2
    FROM ierarhie i, distante d
    WHERE i.Loc2=d.Loc1 AND Nivel < 20)
SELECT 1001 AS IdCursa, Cale AS Traseu,
       CAST ('2008-04-01 09:05:00' AS TIMESTAMP) AS DataOra_Plecare
FROM ierarhie
WHERE Cale LIKE 'Iasi**Vaslui%Focsani' ;

INSERT INTO curse_statii
WITH ierarhie (Cale, Nivel, Durata, Loc2) AS (
    SELECT CAST(Loc1 || '***' || Loc2 AS VARCHAR(500)), 0 AS Nivel,
           CAST (Durata AS NUMERIC), Loc2
    FROM distante WHERE Loc1='Iasi'
    UNION ALL
    SELECT CAST (i.cale || '***' || d.Loc2 AS VARCHAR(500)),
           Nivel + 1, CAST (i.durata+d.durata AS NUMERIC), d.loc2
    FROM ierarhie i, distante d
    WHERE i.Loc2=d.Loc1 AND Nivel < 20)
SELECT IdCursa, 1 AS StatieNr, 'Iasi' AS Statie, DataOra_Plecare AS DataOra
FROM curse
WHERE IdCursa=1001
    UNION
SELECT IdCursa, nivel + 1, Loc2,
       DataOra_Plecare + Durata MINUTE
FROM ierarhie INNER JOIN curse
    ON SUBSTR(curse.Traseu,1,LENGTH(ierarhie.Cale)) = ierarhie.Cale
ORDER BY 1,2

```

Dacă am reușit operațiunea aceasta laborioasă, popularea tabelor FACTURI și LINIIFACT pentru luna noiembrie 2007 este floare la ureche:

– DB2: preluarea, pentru luna nov.2007 a conținutului (aprox.) facturilor din aug.2007

```

INSERT INTO facturi (NrFact, DataFact, CodCl, Obs) (
    SELECT 4000+NrFact, DataFact+ 3 MONTHS, CodCl, Obs
    FROM facturi
    WHERE YEAR(DataFact)=2007 AND MONTH(DataFact)=8

```

```

        AND DAYNAME(DataFact+ 3 MONTHS) NOT IN
        ('SATURDAY', 'SUNDAY')
    );
-- DB2: adaugarea liniilor facturilor "proaspete"
INSERT INTO (SELECT NrFact, Linie, CodPr, Cantitate, PretUnit FROM liniifact)
    (SELECT 4000+NrFact, Linie, CodPr, Cantitate, PretUnit
    FROM liniifact
    WHERE NrFact IN (
        SELECT NrFact-4000 FROM facturi WHERE NrFact > 5000)
    );

```

Nici cele două inserturi dedicate facturilor din luna decembrie 2007 nu ridică probleme deosebite:

```

-- DB2: adunarea facturilor din aug./sept./oct. si generarea facturilor pt. dec.2007
INSERT INTO facturi (NrFact, DataFact, CodCl, Obs)
    (SELECT 7000 + Poz AS "Noul NrFact",
        CASE MONTH(DataFact)
            WHEN 8 THEN DataFact+ 4 MONTHS
            WHEN 9 THEN DataFact+ 3 MONTHS
            WHEN 10 THEN DataFact + 2 MONTHS
        END AS "Noua DataFact", CodCl, Obs
    FROM
        (SELECT NrFact, RANK() OVER
            (ORDER BY DAY(DataFact), NrFact) AS Poz
        FROM facturi
        WHERE YEAR(DataFact)=2007 AND MONTH(DataFact) IN (8,9,10)
        ) coresp
    INNER JOIN facturi f ON coresp.NrFact=f.NrFact AND
        YEAR(DataFact)=2007 AND MONTH(DataFact) IN (8,9,10)
    );
-- DB2: liniile facturilor din decembrie 2007
INSERT INTO liniifact (
    SELECT 7000 + Poz AS "Noul NrFact", Linie, CodPr, Cantitate, PretUnit
    FROM
        (SELECT NrFact, RANK() OVER
            (ORDER BY DAY(DataFact), NrFact) AS Poz
        FROM facturi
        WHERE YEAR(DataFact)=2007 AND MONTH(DataFact) IN (8,9,10)
        ) coresp
    INNER JOIN facturi f ON coresp.NrFact=f.NrFact AND
        YEAR(DataFact)=2007 AND MONTH(DataFact) IN (8,9,10)
    );

```

```

INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
);

```

SQL Server nu permite specificarea după clauza INTO a unei interogări care să returneze atributele pentru care vor fi inserate valori, așa că sintaxa pentru popularea cu date pentru noiembrie 2007 are forma:

– SQL Server: preluarea, pentru luna nov.2007 a datelor din facturile lunii aug.2007

```

INSERT INTO facturi
SELECT 4000+NrFact, DATEADD(MONTH, 3, DataFact), CodCl, Obs
FROM facturi
WHERE YEAR(DataFact)=2007 AND MONTH(DataFact)=8
AND DATENAME(WEEKDAY, DATEADD(MONTH, 3, DataFact))
NOT IN ('SATURDAY', 'SUNDAY');

```

– SQL Server: adaugarea liniilor facturilor "proaspete"

```

INSERT INTO liniifact
SELECT 4000+NrFact, Linie, CodPr, Cantitate, PretUnit
FROM liniifact WHERE NrFact IN (
SELECT NrFact-4000 FROM facturi WHERE NrFact > 5000)

```

Iată și cele două INSERT-uri dedicate facturilor artificiale din luna decembrie 2007:

– SQL Server: adunarea facturilor din aug./sept./oct. și generarea facturilor pt. dec.2007

```

INSERT INTO facturi
SELECT 7000 + Poz,
CASE MONTH(DataFact)
WHEN 8 THEN DATEADD(MONTH, 4, DataFact)
WHEN 9 THEN DATEADD(MONTH, 3, DataFact)
WHEN 10 THEN DATEADD(MONTH, 2, DataFact)
END AS "Noua DataFact", CodCl, Obs
FROM
(SELECT NrFact, RANK() OVER (ORDER BY DAY(DataFact), NrFact)
AS Poz
FROM facturi
WHERE YEAR(DataFact)=2007 AND MONTH(DataFact) IN (8,9,10)
) coresp
INNER JOIN facturi f ON coresp.NrFact=f.NrFact AND
YEAR(DataFact)=2007 AND MONTH(DataFact) IN (8,9,10) ;

```

– SQL Server: liniile facturilor din decembrie 2007

```

INSERT INTO liniifact

```

```

SELECT 7000 + Poz, Linie, CodPr, Cantitate, PretUnit
FROM
    (SELECT NrFact, RANK() OVER (ORDER BY DAY(DataFact), NrFact)
     AS Poz
    FROM facturi
    WHERE YEAR(DataFact)=2007 AND MONTH(DataFact) IN (8,9,10)
    ) coresp
    INNER JOIN facturi f ON coresp.NrFact=f.NrFact AND
    YEAR(DataFact)=2007 AND MONTH(DataFact) IN (8,9,10)
    INNER JOIN liniifact lf ON f.NrFact=lf.NrFact ;

```

Sintaxa primelor două comenzi INSERT dedicate adăugării artificiale de înregistrări pentru luna noiembrie 2007 are în PostgreSQL două diferențe față de cea Oracle. Mai întâi intervalul se exprimă de maniera *INTERVAL '3 MONTH'*, și nu *INTERVAL '3' MONTH*. În al doilea rând, ca și în SQL Server, după INTO nu poate urma un SELECT:

– PostgreSQL: preluarea, pentru luna nov.2007 a datelor din facturile lunii aug.2007

```

INSERT INTO facturi (
    SELECT 4000+NrFact, DataFact+ INTERVAL '3 MONTH', CodCl, Obs
    FROM facturi
    WHERE EXTRACT(YEAR FROM DataFact)=2007 AND
    EXTRACT(MONTH FROM DataFact)=8 AND
    RTRIM(TO_CHAR(DataFact+ INTERVAL '3 MONTH', 'DAY')) NOT IN
    ('SATURDAY', 'SUNDAY')
);

```

– PostgreSQL: adaugarea liniilor facturilor "proaspete" pt. nov.2007

```

INSERT INTO liniifact
    (SELECT 4000+NrFact, Linie, CodPr, Cantitate, PretUnit
    FROM liniifact
    WHERE NrFact IN (
        SELECT NrFact-4000
        FROM facturi
        WHERE NrFact > 5000)
    );

```

Partea cea mai interesantă, cea cu agregarea facturilor din august, septembrie și octombrie și, astfel, generarea (artificială) a facturilor lunii decembrie 2007 se lovește de un obstacol destul de neplăcut în PostgreSQL – inexistența funcțiilor OLAP, în cazul nostru a funcției RANK. Avem însă o soluție bazată pe corelare, prezentată în paragraful 10.1 în exemplele cu afișarea numărului curent al fiecărei facturi (figurile 10.7 și 10.8). Iată-o:

```

INSERT INTO facturi
    (SELECT 7000 + Poz AS "Noul NrFact",

```

```

CASE EXTRACT (MONTH FROM DataFact)
    WHEN 8 THEN DataFact+INTERVAL '4 MONTH'
    WHEN 9 THEN DataFact+INTERVAL '3 MONTH'
    WHEN 10 THEN DataFact+INTERVAL '2 MONTH'
END AS "Noua DataFact", CodCl, Obs
FROM
(SELECT NrFact, (SELECT COUNT(*) + 1
    FROM facturi
    WHERE EXTRACT(YEAR FROM DataFact)=2007 AND
        EXTRACT(MONTH FROM DataFact) IN (8,9,10) AND
        (EXTRACT(DAY FROM DataFact) <
            EXTRACT(DAY FROM f1.DataFact)
        OR EXTRACT(DAY FROM DataFact) = EXTRACT(DAY FROM f1.DataFact) AND
        NrFact < f1.NrFact)
    ) AS Poz
FROM facturi f1
WHERE EXTRACT(YEAR FROM DataFact)=2007 AND
    EXTRACT(MONTH FROM DataFact) IN (8,9,10)
) coresp
INNER JOIN facturi f ON coresp.NrFact=f.NrFact AND
    EXTRACT(YEAR FROM DataFact)=2007
    AND EXTRACT(MONTH FROM DataFact) IN (8,9,10)
);

INSERT INTO liniifact (
    SELECT 7000 + Poz , Linie, CodPr, Cantitate, PretUnit
    FROM
    (SELECT NrFact, (SELECT COUNT(*) + 1
        FROM facturi
        WHERE EXTRACT(YEAR FROM DataFact)=2007 AND EXTRACT(MONTH FROM
            DataFact) IN (8,9,10) AND (EXTRACT(DAY FROM
            DataFact) < EXTRACT(DAY FROM f1.DataFact) OR
            EXTRACT(DAY FROM DataFact) = EXTRACT(DAY FROM f1.DataFact)
            AND NrFact < f1.NrFact)
        ) AS Poz
    FROM facturi f1
    WHERE EXTRACT(YEAR FROM DataFact)=2007 AND
        EXTRACT(MONTH FROM DataFact) IN (8,9,10)
    ) coresp
    INNER JOIN facturi f ON coresp.NrFact=f.NrFact AND

```



```

EXTRACT(YEAR FROM DataFact)=2007
AND EXTRACT(MONTH FROM DataFact) IN (8,9,10)
INNER JOIN liniifact lf ON f.NrFact=lf.NrFact
);

```

13.3.2. Inserări în tabele multiple

Comanda INSERT are, numai în Oracle, și un format prin care, simultan, pot fi introduse linii în două sau mai multe tabele, pe baza unor condiții aplicate rezultatului unei fraze SELECT. Revenim la o problemă introdusă în primul paragraf din acest capitol, dar pe care o complicăm exagerat. Ne interesează ca datele despre facturi (celebrul „tandem” FACTURI-LINIIFACT) să fie arhivate lunar. Chiar dacă noi pe anul 2007 avem introduse datele pe lunile august-decembrie (pentru ultimele luni am „semănat” câteva înregistrări chiar în paragraful precedent), imaginăm o soluție aplicabilă pentru un an întreg. Pentru 2007 vom crea 12 de tabele-fragment pe baza tabelii FACTURI: FACTURI_2007_1, FACTURI_2007_2, ..., FACTURI_2007_12 și 12 din LINIIFACT: LINIIFACT_2007_2, ... – vezi listingul 13.2.

Listing 13.2. Crearea „clasică” a tabelilor de arhivare

```

CREATE TABLE facturi_2007_1 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=1 );
CREATE TABLE facturi_2007_2 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=2 );
CREATE TABLE facturi_2007_3 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=3 );
CREATE TABLE facturi_2007_4 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=4 );
CREATE TABLE facturi_2007_5 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=5 );
CREATE TABLE facturi_2007_6 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=6 );
CREATE TABLE facturi_2007_7 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=7 );
CREATE TABLE facturi_2007_8 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=8 );
CREATE TABLE facturi_2007_9 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=9 );
CREATE TABLE facturi_2007_10 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=10 );
CREATE TABLE facturi_2007_11 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=11 );
CREATE TABLE facturi_2007_12 AS (SELECT * FROM facturi WHERE EXTRACT
(YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=12 );

CREATE TABLE liniifact_2007_1 AS (SELECT * FROM liniifact
WHERE NrFact IN (SELECT NrFact FROM facturi_2007_1) );
CREATE TABLE liniifact_2007_2 AS (SELECT * FROM liniifact
WHERE NrFact IN (SELECT NrFact FROM facturi_2007_2) );
CREATE TABLE liniifact_2007_3 AS (SELECT * FROM liniifact
WHERE NrFact IN (SELECT NrFact FROM facturi_2007_3) );
CREATE TABLE liniifact_2007_4 AS (SELECT * FROM liniifact
WHERE NrFact IN (SELECT NrFact FROM facturi_2007_4) );

```

```

CREATE TABLE liniifact_2007_5 AS (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_5));
CREATE TABLE liniifact_2007_6 AS (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_6));
CREATE TABLE liniifact_2007_7 AS (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_7));
CREATE TABLE liniifact_2007_8 AS (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_8));
CREATE TABLE liniifact_2007_9 AS (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_9));
CREATE TABLE liniifact_2007_10 AS (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_10));
CREATE TABLE liniifact_2007_11 AS (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_10));
CREATE TABLE liniifact_2007_12 AS (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_12));

```

Să presupunem că, la arhivare, câteva facturi au fost omise (din cauze neelucidate) și, pentru siguranță, reexecutăm arhivarea pentru întreg anul 2007. În locul clasicei variante DELETE/INSERT din listingul 13.3, putem încerca și ceva mai exotic, chiar dacă câștigul de productivitate nu este chiar entuziasmant.

Listing 13.3. Re-popularea tabelor de arhivare

```

DELETE FROM facturi_2007_1 ;
DELETE FROM facturi_2007_2 ;
...

INSERT INTO facturi_2007_1 (SELECT * FROM facturi WHERE EXTRACT
    (YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=1 );
INSERT INTO facturi_2007_2 (SELECT * FROM facturi WHERE EXTRACT
    (YEAR FROM DataFact)=2007 AND EXTRACT (MONTH FROM DataFact)=2 );
....

DELETE FROM liniifact_2007_1 ;
DELETE FROM liniifact_2007_2 ;
...

INSERT INTO liniifact_2007_1 (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_1));
INSERT INTO liniifact_2007_2 (SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi_2007_2));
...

```

Numai pentru arhivele tabeli FACTURI cele 12 comenzi INSERT pot fi înlocuite cu unul singur cu următoarea formă:

INSERT ALL

```

WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-01' THEN INTO facturi_2007_1
WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-02' THEN INTO facturi_2007_2
WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-03' THEN INTO facturi_2007_3
WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-04' THEN INTO facturi_2007_4
WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-05' THEN INTO facturi_2007_5
WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-06' THEN INTO facturi_2007_6

```

```

        WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-07' THEN INTO facturi_2007_7
        WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-08' THEN INTO facturi_2007_8
        WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-09' THEN INTO facturi_2007_9
        WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-10' THEN INTO facturi_2007_10
        WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-11' THEN INTO facturi_2007_11
        WHEN TO_CHAR(DataFact, 'YYYY-MM')='2007-12' THEN INTO facturi_2007_12
SELECT * FROM facturi

```

O aplicare spectaculoasă a comenzii INSERT ALL este legată de „verticalizarea orizontalității”. Să luăm un exemplu relativ apropiat de cazul înscrierii la master la Facultatea de Economie și Administrarea Afacerilor. Există șase specializări de master, fiecare cu un anumit număr de locuri disponibile:

```

CREATE TABLE mastere (
    Spec VARCHAR2(4) PRIMARY KEY,
    DenSpec VARCHAR2(40) NOT NULL,
    NrLocuri NUMBER(3)
);

```

La înscriere candidații completează o fișă în care, pe lângă nume și media anilor de studiu (*Media_AS*) și media de la licență (*Media_Lic*), precizează, în ordinea preferenței, programele pe care ar dori să le urmeze. Repartizarea se face în funcție de ordinea mediei de admitere care se calculează prin formula $Med_AS * 0.6 + Med_Lic * 0.4$. Dacă un student, nu „încapă” la prima specializare – *Spec1* – pe care a solicitat-o (locurile de la specializarea de master respectivă s-au completat pentru că au fost studenți cu medii mai mari care au avut prioritate), atunci va fi repartizat la *Spec2*; dacă și aici locurile sunt ocupate, se va încerca repartizarea sa la *Spec3* s.a.m.d. Dacă, însă, candidatul nu e interesat decât de două specializări, va completa *Spec1* și *Spec2*, restul atributelor (*Spec3*, ..., *Spec6*) fiind NULL:

```

CREATE TABLE candidati (
    IdCandidat NUMBER(6) NOT NULL PRIMARY KEY,
    NumePren VARCHAR2(50) NOT NULL,
    Media_AS NUMBER(4,2) NOT NULL CONSTRAINT ck_media_as
        CHECK (media_as BETWEEN 5 AND 10),
    Media_Lic NUMBER(4,2) NOT NULL CONSTRAINT ck_media_lic
        CHECK (media_lic BETWEEN 6 AND 10),
    Spec1 VARCHAR2(4) NOT NULL REFERENCES mastere(Spec),
    Spec2 VARCHAR2(4) REFERENCES mastere(Spec),
    Spec3 VARCHAR2(4) REFERENCES mastere(Spec),
    Spec4 VARCHAR2(4) REFERENCES mastere(Spec),
    Spec5 VARCHAR2(4) REFERENCES mastere(Spec),
    Spec6 VARCHAR2(4) REFERENCES mastere(Spec),
    CONSTRAINT ck_spec CHECK (

```

```

COALESCE(spec2, '') = COALESCE(spec2, spec3, '') AND
COALESCE(spec3, '') = COALESCE(spec3, spec4, '') AND
COALESCE(spec4, '') = COALESCE(spec4, spec5, '') AND
COALESCE(spec5, '') = COALESCE(spec5, spec6, '') )
);

```

Probabil cel mai interesant punct al comenzii este regula de validare declarată la finalul comenzii. Cei mai răbdători dintre dvs. și-au dat seama că acest CHECK urmărește ca, odată ce un candidat nu a completat una din opțiunile Spec2, ..., Spec5, niciuna dintre următoarele nu mai poate fi nenulă. Cu ocazia aceasta am ilustrat și sintaxa cu trei argumente a comenzii COALESCE. Listingul 13.4 poluează cu câteva înregistrări cele două tabele.

Listing 13.4. Popularea tabelelor MASTERE și CANDIDAȚI

```

DELETE FROM candidati ;
DELETE FROM mastere ;

INSERT INTO mastere VALUES ('SIA', 'Sisteme informationale pentru afaceri', 4) ;
INSERT INTO mastere VALUES ('FAB', 'Finante-Asigurari-Banci', 6) ;
INSERT INTO mastere VALUES ('CEA', 'Contabilitate, expertiza si audit', 6) ;
INSERT INTO mastere VALUES ('MARK', 'Marketing', 5) ;
INSERT INTO mastere VALUES ('EAI', 'Economie si afaceri internationale', 5) ;
INSERT INTO mastere VALUES ('MRU', 'Managementul resurselor umane', 4) ;

INSERT INTO candidati VALUES ( 1, 'Popescu I. Irina', 7.5, 9.50, 'FAB', 'MARK', 'EAI', 'MRU',
    NULL, NULL);
INSERT INTO candidati VALUES ( 2, 'Babias D. Ecaterina', 8.5, 9.20, 'SIA', 'EAI', NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES ( 3, 'Strat P. Iulian', 7.35, 9.50, 'CEA', 'EAI', 'MRU', 'SIA',
    'FAB', 'MARK');
INSERT INTO candidati VALUES ( 4, 'Georgescu M. Honda', 8.5, 9.00, 'MRU', 'MARK', 'EAI',
    'FAB', 'CEA', 'SIA');
INSERT INTO candidati VALUES ( 5, 'Munteanu A. Optimista', 9.5, 9.50, 'SIA', 'CEA', 'FAB',
    NULL, NULL, NULL);
INSERT INTO candidati VALUES ( 6, 'Dumitriu F. Dura', 9.5, 10, 'EAI', NULL, NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES ( 7, 'Mesnita G. Plinutul', 10, 10, 'EAI', 'MARK', 'MRU', NULL,
    NULL, NULL);
INSERT INTO candidati VALUES ( 8, 'Greavu V. Doru', 9.25, 8.50, 'SIA', 'CEA', 'FAB', NULL,
    NULL, NULL);
INSERT INTO candidati VALUES ( 9, 'Baboi P. Iustina', 8.5, 8.50, 'SIA', NULL, NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (10, 'Postelnicu I. Irina', 9.5, 8.25, 'MARK', NULL, NULL,
    NULL, NULL, NULL);
INSERT INTO candidati VALUES (11, 'Fotache H. Fanel', 9.75, 9.50, 'MRU', NULL, NULL,
    NULL, NULL, NULL);
INSERT INTO candidati VALUES (12, 'Moscovici J. Cristina', 9.80, 9.30, 'CEA', 'SIA', 'FAB',
    NULL, NULL, NULL);
INSERT INTO candidati VALUES (13, 'Rusu I. Vanda', 7.80, 9.10, 'FAB', 'CEA', 'MARK', 'MRU',
    NULL, NULL);
INSERT INTO candidati VALUES (14, 'Spinu M. Algebra', 7.25, 9.00, 'SIA', 'CEA', 'FAB', NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (15, 'Sandovici I. Irina', 7.05, 7.50, 'MARK', 'MRU', 'EAI', 'CEA',
    NULL, NULL);
INSERT INTO candidati VALUES (16, 'Plai V. Picior', 7.5, 7.90, 'EAI', 'SIA', 'CEA', 'FAB',
    NULL, NULL);

```

```

INSERT INTO candidati VALUES (17, 'Ambuscada B. Cristina', 8.25, 9.50, 'SIA', 'CEA', 'FAB', 'EAI',
    NULL, NULL);
INSERT INTO candidati VALUES (18, 'Pinda A. Axinia', 8.75, 9.00, 'SIA', 'FAB', 'CEA', NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (19, 'Planton V. Grigore', 9.25, 9.50, 'FAB', NULL, NULL,
    NULL, NULL, NULL);
INSERT INTO candidati VALUES (20, 'Sergentu I. Zece', 7.5, 9.50, 'FAB', 'CEA', NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (21, 'Ababei T. Marian-Vasile', 7.5, 9.50, 'CEA', 'MRU', NULL,
    NULL, NULL, NULL);
INSERT INTO candidati VALUES (22, 'Popistasu J. Maria', 7.5, 9.50, 'FAB', 'EAI', 'CEA', 'MRU',
    NULL, NULL);
INSERT INTO candidati VALUES (23, 'Plop R. Robert', 7.5, 9.50, 'FAB', 'MARK', 'MRU', NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (24, 'Aflorei H. Crina', 7.5, 9.50, 'EAI', 'SIA', NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (25, 'Afaunei P. Gina', 7.5, 9.50, 'SIA', NULL, NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (26, 'Vatamanu I. Alexandrina', 7.5, 9.50, 'MRU', 'MARK', 'EAI',
    NULL, NULL, NULL);
INSERT INTO candidati VALUES (27, 'Lovin P. Marian', 7.5, 9.50, 'MARK', 'MRU', NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (28, 'Antiteza W. Florin', 7.5, 9.50, 'EAI', 'MARK', 'MRU',
    NULL, NULL, NULL);
INSERT INTO candidati VALUES (29, 'Prepelita V. Ion', 7.5, 9.50, 'EAI', NULL, NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (30, 'Cioara X. Sanda', 7.5, 9.50, 'CEA', 'FAB', 'EAI', NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (31, 'Metafora Y. Vasile', 7.5, 9.50, 'SIA', 'CEA', NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (32, 'Strasina R. Elvis', 7.5, 9.50, 'CEA', NULL, NULL, NULL,
    NULL, NULL);
INSERT INTO candidati VALUES (33, 'Durere V. Vasile', 7.5, 9.50, 'FAB', 'CEA', 'EAI', 'MARK',
    'MRU', NULL);
INSERT INTO candidati VALUES (34, 'Sedentaru L. Marius-Daniel', 7.5, 9.50, 'MARK', 'MRU',
    'EAI', NULL, NULL, NULL);
INSERT INTO candidati VALUES (35, 'Zgircitu I. Daniel', 7.5, 9.50, 'MRU', NULL, NULL, NULL,
    NULL, NULL);

```

Pentru a putea prelucra opțiunile fiecărui candidat (vezi capitolul 16) și a determina specializarea de master la care a fost admis un candidat (vor exista, însă, și candidați respinși), vom crea o tabelă specială – **OPȚIUNI** – cu atributele *IdCandidat*, *Ordine* și *Spec*. Tot din comoditate, în locul clasicei variante **CREATE TABLE ...** listă atribute și restricții... vom apela la o interogare:

```

CREATE TABLE optiuni AS
    SELECT IdCandidat, 99 AS Ordine, Spec1 AS Spec
FROM candidati
WHERE 1=2

```

Condiția din clauza **WHERE** ne va asigura că, la creare, tabela **OPTIUNI** nu va avea nicio linie. Popularea cu înregistrări a acestei noi tabele poate fi realizată dintr-o suflare astfel:

```

INSERT ALL

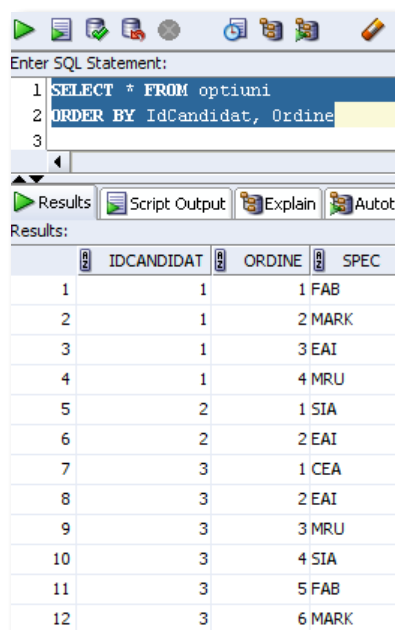
```

```

WHEN Spec1 IS NOT NULL THEN INTO optiuni
VALUES (IdCandidat, 1, Spec1)
WHEN Spec2 IS NOT NULL THEN INTO optiuni
VALUES (IdCandidat, 2, Spec2)
WHEN Spec3 IS NOT NULL THEN INTO optiuni
VALUES (IdCandidat, 3, Spec3)
WHEN Spec4 IS NOT NULL THEN INTO optiuni
VALUES (IdCandidat, 4, Spec4)
WHEN Spec5 IS NOT NULL THEN INTO optiuni
VALUES (IdCandidat, 5, Spec5)
WHEN Spec6 IS NOT NULL THEN INTO optiuni
VALUES (IdCandidat, 6, Spec6)
SELECT * FROM candidati

```

Pentru confirmare, figura 13.6 prezintă câteva dintre liniile tabelului.



	IDCANDIDAT	ORDINE	SPEC
1	1	1	FAB
2	1	2	MARK
3	1	3	EAI
4	1	4	MRU
5	2	1	SIA
6	2	2	EAI
7	3	1	CEA
8	3	2	EAI
9	3	3	MRU
10	3	4	SIA
11	3	5	FAB
12	3	6	MARK

Figura 13.6. Câteva linii din tabela OPTIUNI

13.3.3. Ștergeri ceva mai sofisticate

După cum am văzut în capitolul 3, comenzile de modificare (UPDATE) și de ștergere (DELETE) prezintă clauzele FROM și WHERE similare frazei SELECT. După discuțiile din capitolele precedente suntem în măsură să formulăm condiții mai sofisticate de ștergere a unor linii. Astfel, dacă dorim să eliminăm din baza de

date toate facturile care nu au nicio linie, putem folosi cel puțin trei variante. Prima este și cea mai „cuminte”:

```
DELETE FROM facturi
WHERE NrFact NOT IN (SELECT NrFact FROM liniifact)
```

A doua „corelează” (un pic):

```
DELETE FROM facturi
WHERE NOT EXISTS
(SELECT 1 FROM liniifact
WHERE liniifact.NrFact = facturi.NrFact)
```

Mai spectaculoasă este însă varianta în care, în locul numelui unei tabele, se folosește o subconsultare în clauza FROM, sintaxă care este, însă, acceptată numai de DB2 și Oracle:

```
DELETE FROM (
SELECT *
FROM facturi
WHERE NrFact NOT IN (SELECT NrFact FROM liniifact)
)
```

Interesant este că interogarea aceasta funcționează și dacă în locul calificativului pentru toate atributele tabeli (*) am fi folosit NrFact sau oricare alt atribut (chiar și Obs care prezintă valori nule):

```
DELETE FROM (SELECT NrFact FROM facturi WHERE NrFact NOT IN
(SELECT NrFact FROM liniifact) )
```

Întrucât am făcut arhivarea facturilor pentru lunile anului 2007, dorim să ștergem liniile arhivate din tabele FACTURI și LINIIFACT. Să luăm exemplul lunii iulie, sintaxa fiind una comună Oracle/DB2:

```
DELETE FROM (SELECT * FROM liniifact ) lf
WHERE EXISTS
(SELECT 1 FROM facturi_2007_7 f07 WHERE f07.NrFact=lf.NrFact) ;
```

```
DELETE FROM (SELECT * FROM facturi ) f
WHERE EXISTS
(SELECT 1 FROM facturi_2007_7 f07 WHERE f07.NrFact=f.NrFact) ;
```

În cartea dedicată Oracle 9i am prezentat o găselniță interesantă – includerea în clauza FROM a unei comenzi DELETE a joncțiunii unei tabele. Astfel, comanda:

```
DELETE FROM
(SELECT * FROM liniifact lf INNER JOIN facturi f
ON lf.NrFact=f.NrFact AND DataFact=DATE'2007-07-21')
```

va șterge din tabela LINIIFACT toate liniile facturilor emise pe 21 iulie 2007, păstrându-se însă înregistrările din tabela părinte (FACTURI). Sintaxa este valabilă doar în Oracle.

Dacă, însă, folosim o joncțiune externă, lucrurile se schimbă și în Oracle. Încercăm o altă variantă de ștergere a facturilor fără linii, de data aceasta pe bază de joncțiune externă:

```
DELETE FROM
    (SELECT * FROM facturi f LEFT OUTER JOIN liniifact lf
     ON f.NrFact=lf.NrFact AND lf.NrFact IS NULL)
```

Ceea ce obținem este un mesaj de eroare – vezi figura 13.7. De notat că eroarea semnalizată face trimitere la o „vedere” (view), subiect abordat abia în capitolul următor.

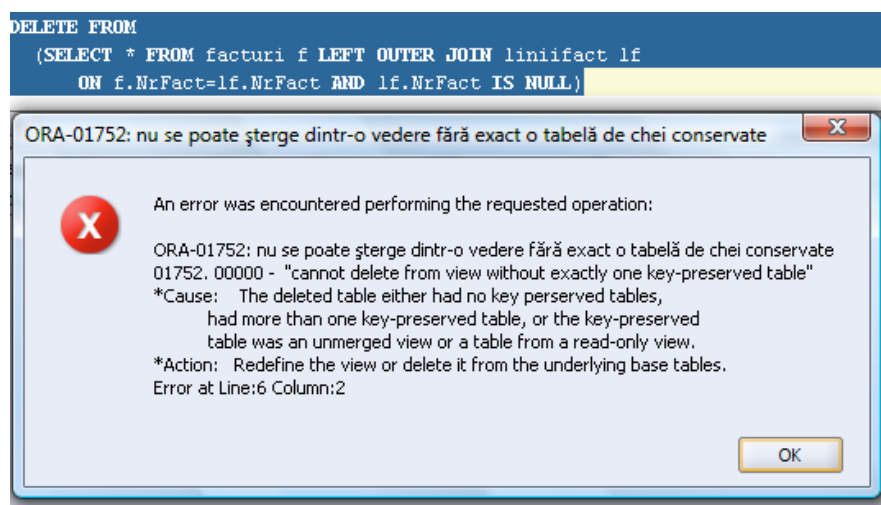


Figura 13.7. Eroare la folosirea joncțiunii externe în clauza FROM a unei comenzi DELETE

Schimbăm un pic registrul, referindu-ne la traseele firmei de transport. Știm din capitolul anterior cum să generăm traseele (rutele) dintre două localități. Să generăm, astfel, o tabelă care va conține, pentru început, traseele Iași-Focșani.

```
CREATE TABLE trasee AS
    SELECT Ord,Traseu,
        (SELECT SUM(Distanta) FROM (
            SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**'
              || Loc2 AS Traseu, LEVEL AS Nivel, d.*,
                ROWNUM AS Ord
            FROM distante d
            START WITH Loc1='Iasi'
            CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani'
```



```

        AND Level < 50
    ORDER BY Ord          ) t2
    WHERE t1.Traseu LIKE Traseu || '%' AS Km
FROM (
    SELECT SYS_CONNECT_BY_PATH (Loc1, '**') || '**' ||
        Loc2 AS Traseu, LEVEL AS Nivel, d.*, ROWNUM AS Ord
    FROM distante d
    START WITH Loc1='Iasi'
    CONNECT BY PRIOR Loc2 = Loc1 AND Loc1<>'Focsani'
        AND Level < 50
    ORDER BY Ord) t1
WHERE Traseu LIKE '**Iasi%Focsani'
ORDER BY Km

```

Acum, că tot le-am creat, a sosit momentul să le ștergem. Dar nu oricum. Ne propunem să eliminăm toate rutele cu distanța mai mare decât media distanțelor pentru toate rutele Iași-Focșani. Nimic mai simplu:

```

DELETE FROM trasee WHERE Traseu LIKE '**Iasi%Focsani' AND Km <
    (SELECT AVG(Km) FROM trasee WHERE Traseu LIKE '**Iasi%Focsani')

```

Redactarea perechii de comenzi CREATE TABLE / DELETE în celelalte dialecte SQL rămâne ca o încercare pe cont propriu.

13.3.4. Actualizări și subconsultări

UPDATE are formatul general apropiat de cel al comenzii DELETE. Față de ștergeri, la modificări lucrurile pot fi mai complicate (implicit, mai interesante), deoarece putem folosi subconsultări atât în predicatul clauzei WHERE, cât și pentru a preciza expresiile pentru atribuirea valorii unor atribute.

Să începem, însă, cu o problemă mai simplă. Pentru facturile care nu au nici o linie am vrea să adăugăm un „semnal”, lipit la eventuala valorare nenulă a atributului Obs:

```

UPDATE facturi
SET Obs = COALESCE(Obs, '') || 'fara linii!!!'
WHERE NrFact NOT IN (SELECT NrFact FROM liniifact)

```

Similar comenzii DELETE, tabela-argument a comenzii UPDATE poate fi înlocuită cu o subconsultare, astfel încât în DB2 și Oracle este corectă și versiunea:

```

UPDATE      (SELECT Obs FROM facturi WHERE NrFact NOT IN
              (SELECT NrFact FROM liniifact)
            )
SET Obs = COALESCE(Obs, '') || 'fara linii!!!'

```

Pentru a avea o mai mare libertate de mișcare, adăugăm trei atribute pe care le vom lega de denormalizarea bazei de date⁵. Astfel, în tabela FACTURI adăugăm nu mai puțin de cinci atribute:

- *ValTotală* reprezintă valoarea totală (inclusiv TVA) a facturii;
- *TVA* – taxa pe valoarea adăugată pe care ar trebui să o plătim statului (pentru că ne-a dat voie să facem vânzări, sau cel puțin asta e părerea lui);
- *ValÎncasată* reprezintă suma plătită de client din contravaloarea facturii respective;
- *Reduceri*: dacă un client plătește o factură înainte de termenul obișnuit (să zicem zece zile), îi putem acorda o reducere de 5% pentru că ne-am procopsit rapid cu lichidități; procentul poate fi negociat diferit cu fiecare client;
- *Penalități*: este opusul atributului anterior. Prin contract sau prin lege, dacă un client este mai leneș cu plata la timp a facturilor, i se pot aplica penalități, fără a avea însă certitudinea că le-am putea încasa vreodată.

Practic, prin aceste noi atribute, urmărirea încasării facturilor se schimbă sensibil. *Valoarea de încasat* dintr-o factură este *valoarea totală* plus *penalități* minus *reduceri*. Asta e vestea bună. Vestea proastă ține de faptul că atributul *ValTotală* n-ar trebui să se modifice decât la actualizarea tabelii LINIIFACT, nu ? Altminteri, dacă utilizatorul ar modifica printr-un UPDATE acest atribut, ar apărea un decalaj supărător între liniile din facturi și valorile acestora. Actualizarea automată a unor atribute calculate este unul din scopurile declarate ale declanșatoarelor (trigger-elor) pe care le vom discuta în capitolul 17. Adăugăm cele cinci atribute în tabela FACTURI, sintaxa următoare fiind funcțională în toate cele patru dialecte SQL:

```
ALTER TABLE facturi ADD ValTotala NUMERIC(10,2) DEFAULT 0;
ALTER TABLE facturi ADD TVA NUMERIC(10,2) DEFAULT 0;
ALTER TABLE facturi ADD ValIncasata NUMERIC(10,2) DEFAULT 0;
ALTER TABLE facturi ADD Reduceri NUMERIC(9,2) DEFAULT 0;
ALTER TABLE facturi ADD Penalizari NUMERIC(9,2) DEFAULT 0;
```

Acum, dacă tot le-am creat, trebuie să le “umplem” cu valorile curente din tabelele LINIIFACT, PRODUSE și INCASFACT. Pentru calculul valorii totale a unei facturi avem nevoie de o interogare corelată după cum urmează:

```
UPDATE facturi
```

⁵ Dacă la prima ediție făceam trimiterea la denormalizare păstrând o oarecare discreție asupra ei, între timp am tratat subiectul ceva mai detaliat în capitolul 9 al cărții publicate în 2005 [Fotache 2005]

```

SET ValTotala = (
    SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
    FROM liniifact lf INNER JOIN produse p
        ON lf.CodPr=p.CodPr
    WHERE NrFact = facturi.NrFact
)

```

Subconsultarea ia în calcul cantitatea, prețul unitar și procentul TVA pentru fiecare produs vândut. Prin corelare se vor lua în considerare numai liniile din LINIIFACT (și PRODUSE) corespunzătoare facturii curente (linia curentă din FACTURI). Și această comandă funcționează fără modificări în toate cele patru servere BD.

Cât privește atributul Reduceri, instituim următoarea regulă: se acordă o reducere de 5% pentru toate tranșele unei facturi încasate în termen de 15 zile de la data vânzării. Iată sintaxa Oracle:

```

UPDATE facturi
SET Reduceri = (
    SELECT SUM(
        CASE WHEN DataInc <= DataFact + INTERVAL '15' DAY
            THEN Transa * 0.05 ELSE 0 END
    )
    FROM incasfact incf INNER JOIN incasari i ON incf.CodInc=i.CodInc
        INNER JOIN facturi f2 ON incf.NrFact=f2.NrFact
    WHERE f2.NrFact = facturi.NrFact
)6

```

sau, mai simplu, folosind corelarea cu liniile din tabela-argument FACTURI (sintaxa valabilă doar în Oracle și DB2, însă aici trebuie să înlocuim INTERVAL...):

```

UPDATE facturi
SET Reduceri = (
    SELECT SUM( CASE WHEN DataInc <= DataFact + INTERVAL '15' DAY
        THEN Transa * 0.05 ELSE 0 END)
    FROM incasfact incf INNER JOIN incasari i ON incf.CodInc=i.CodInc
    WHERE incf.NrFact = facturi.NrFact
)

```

⁶ *DataFact + INTERVAL '15' DAY* trebuie înlocuit cu *DataFact + 15 DAY* în DB2, cu *DATEADD(DAY, 15, DataFact)* în SQL Server și cu *DataFact + INTERVAL '15 DAYS'* în PostgreSQL.

Atenție, însă, dacă plasăm condiția din clauza WHERE în continuarea clauzei ON (știm din capitolele anterioare că, de obicei, în interogări putem face lucrul acesta fără probleme):

UPDATE facturi

SET Reduceri = (

SELECT SUM(CASE WHEN DataInc <= DataFact + INTERVAL '15' DAY
THEN Transa * 0.05 ELSE 0 END)

FROM incasfact incf INNER JOIN incasari i ON incf.CodInc=i.CodInc

AND incf.NrFact = facturi.NrFact

)

chiar în Oracle obținem un mic mesaj de eroare – vezi figura 13.8, în timp ce DB2-ul nu ridică probleme (dacă înlocuim INTERVAL...).

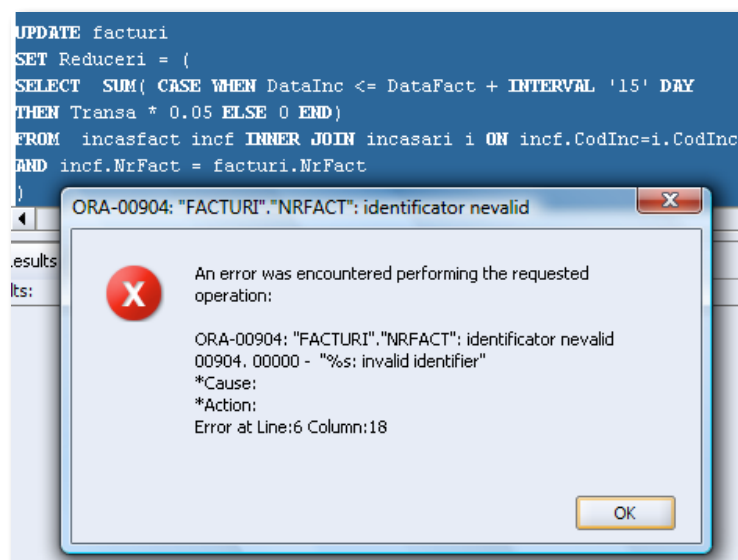


Figura 13.8. Restricție (Oracle) la joncțiune “corelată” cu tabela “centrală” din UPDATE

Complicăm un pic cazul reducerilor. Acordăm 10% pentru tranșele încasate în mai puțin de 15 zile de la data vânzării, 9% pentru 16 zile și 8% pentru 17 zile. Soluția Oracle este⁷:

UPDATE facturi

⁷ Pentru modificările din celelalte dialecte (INTERVAL...), vezi indicațiile de pe pagina anterioară.

```

SET Reduceri = (
    SELECT SUM ( CASE
        WHEN DataInc <= DataFact + INTERVAL '15' DAY THEN Transa * 0.1
        WHEN DataInc <= DataFact + INTERVAL '16' DAY THEN Transa * 0.09
        WHEN DataInc <= DataFact + INTERVAL '17' DAY THEN Transa * 0.08
        ELSE 0
    END)
    FROM incasfact incf INNER JOIN incasari i ON incf.CodInc=i.CodInc
    WHERE incf.NrFact = facturi.NrFact
)

```

Deși pentru varianta simplificată a reducerilor (interogarea precedentă) nu a ridicat nicio obiecție, la execuția acestui SELECT SQL Server se împotrivese – vezi figura 13.9.

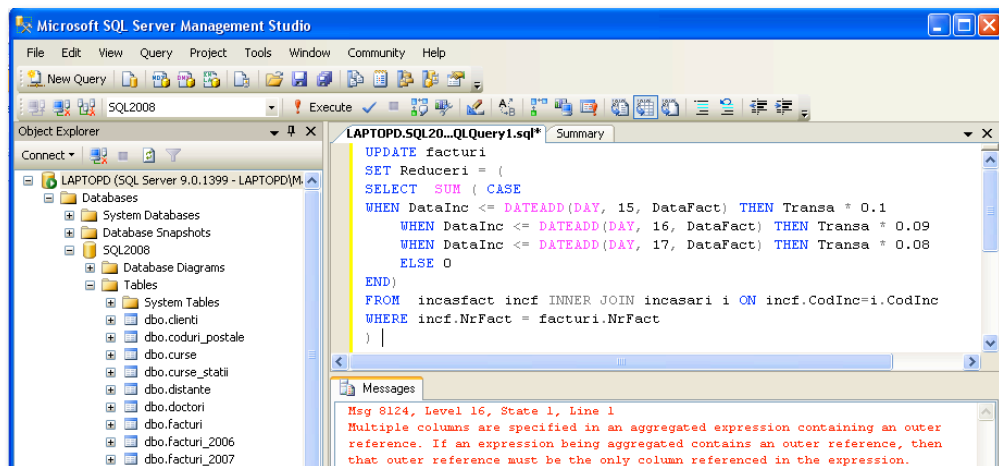


Figura 13.9. Eroare SQL Server datorată modificării structurii CASE

Varianta funcțională este:

```

UPDATE facturi
SET Reduceri = (
    SELECT SUM(
        CASE WHEN DataInc <= DATEADD(DAY, 15, DataFact)
        THEN Transa * 0.1
        ELSE
            CASE WHEN DataInc <= DATEADD(DAY, 16, DataFact)
            THEN Transa * 0.09
            ELSE
                CASE WHEN DataInc <=
                    DATEADD(DAY, 17, DataFact)
                THEN Transa * 0.08

```

```

ELSE 0
END
END
END
)
FROM incasfact incf
INNER JOIN incasari i ON incf.CodInc=i.CodInc
INNER JOIN facturi f2 ON incf.NrFact=f2.NrFact
WHERE f2.NrFact = facturi.NrFact
)

```

Întrucât suntem într-o formă samariteană maximă, nu calculăm penalități, chiar dacă sunt clienți care le-ar merita cu vârf și îndesat. Plasăm cele discutate despre cele cinci atributele calculate într-o interogare „unificatoare”:

```

UPDATE facturi
SET
ValTotala = ( SELECT SUM(Cantitate * PretUnit * (1+ProcTVA))
FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE lf.NrFact=facturi.NrFact
),
TVA = ( SELECT SUM(Cantitate * PretUnit * ProcTVA)
FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr
WHERE lf.NrFact=facturi.NrFact),
ValIncasata = COALESCE( (SELECT SUM(Transa)
FROM incasfact
WHERE NrFact=facturi.NrFact),0),
Reduceri = COALESCE((SELECT SUM ( CASE
WHEN DataInc <= DataFact + INTERVAL '15' DAY THEN Transa * 0.1
WHEN DataInc <= DataFact + INTERVAL '16' DAY THEN Transa * 0.09
WHEN DataInc <= DataFact + INTERVAL '17' DAY THEN Transa * 0.08
ELSE 0
END)
FROM incasfact incf INNER JOIN incasari i
ON incf.CodInc=i.CodInc
WHERE incf.NrFact = facturi.NrFact
),0),
Penalizari = 0

```

Sintaxa este Oracle, dar cu modificările de rigoare funcționează și în celelalte servere BD. Pentru a fi convingător, în figura 13.10 am listat primele nouă înregistrări ale tabelului FACTURI.

NRFACT	DATAFACT	CODCL	OBS	VALTOTALA	TVA	VALINCASATA	REDUCERI	PENALIZARI
1111 01-08-2007		1001 (null)		4346037,5	687287,5	53996	5399,6	0
1112 01-08-2007		1005 Probleme cu transportul		125516	13116	125516	12551,6	0
1113 01-08-2007		1002 (null)		106275	8775	106275	9564,75	0
1114 01-08-2007		1006 (null)		6021706,5	950856,5	0	0	0
1115 02-08-2007		1001 (null)		151237,5	12487,5	0	0	0
1116 02-08-2007		1007 Pretul propus initial a fost modificat		126712,5	10462,5	0	0	0
1117 03-08-2007		1001 (null)		222050	27050	23204	2320,4	0
1118 04-08-2007		1001 (null)		201975	29475	201975	20197,5	0
1119 07-08-2007		1003 (null)		5774498,5	912848,5	0	0	0

Figura 13.10. Valorile atributelor nou introduse în FACTURI

Comanda poate fi simplificată, oarecum, prin calcularea simultană a două, respectiv trei atribute folosind câte o singură interogare, însă doar DB2 și Oracle permit acest lucru (noi ne oprim la sintaxa Oracle):

UPDATE facturi

SET (ValTotala, TVA, Penalizari) =

(SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)),

SUM(Cantitate * PretUnit * ProcTVA),

0

FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr

WHERE lf.NrFact=facturi.NrFact),

(ValIncasata, Reduceri) =

(SELECT COALESCE(SUM(Transa),0),

COALESCE((SUM (CASE

WHEN DataInc <= DataFact + INTERVAL '15' DAY THEN Transa * 0.1

WHEN DataInc <= DataFact + INTERVAL '16' DAY THEN Transa * 0.09

WHEN DataInc <= DataFact + INTERVAL '17' DAY THEN Transa * 0.08

ELSE 0

END)),0)

FROM incasfact incf INNER JOIN incasari i ON incf.CodInc=i.CodInc

WHERE incf.NrFact = facturi.NrFact

)

sau:

UPDATE (SELECT NrFact, DataFact, ValTotala, TVA, ValIncasata, Reduceri, Penalizari

FROM facturi) f

SET

(ValTotala, TVA, Penalizari) =

(SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)),

SUM(Cantitate * PretUnit * ProcTVA),

0

FROM liniifact lf INNER JOIN produse p ON lf.CodPr=p.CodPr

WHERE lf.NrFact=f.NrFact),

```

(ValIncasata, Reduceri) =
    (SELECT COALESCE(SUM(Transa),0),
        COALESCE((SUM ( CASE
            WHEN DataInc <= DataFact + INTERVAL '15' DAY THEN Transa * 0.1
            WHEN DataInc <= DataFact + INTERVAL '16' DAY THEN Transa * 0.09
            WHEN DataInc <= DataFact + INTERVAL '17' DAY THEN Transa * 0.08
            ELSE 0
                END)),0)
        FROM incasfact incf INNER JOIN incasari i ON incf.CodInc=i.CodInc
        WHERE incf.NrFact = f.NrFact
    )

```

Trecând prea-puțin-vătămați prin experiența actualizării atributelor calculate în tabela FACTURI, mai adăugăm discret atributul *TVALinie* în LINIIFACT:

```
ALTER TABLE liniifact ADD TVALinie NUMERIC(9,2) DEFAULT 0;
```

„Umplerea” sa s-ar face destul de simplu cu varianta universală:

```

UPDATE liniifact
SET TVALinie = Cantitate * PretUnit *
    (SELECT ProcTVA FROM produse WHERE CodPr=liniifact.CodPr)

```

sau una „particulară” Oracle:

```

UPDATE      (SELECT * FROM liniifact INNER JOIN produse
              ON liniifact.CodPr=produse.CodPr)
SET TVALinie = Cantitate * PretUnit * ProcTVA

```

Ca și în cazul comenzii DELETE, Oracle ne oferă o eroare dacă apelăm la o joncțiune externă (vezi figura 13.11):

```

UPDATE      (SELECT Obs FROM facturi LEFT OUTER JOIN liniifact
              ON facturi.NrFact=liniifact.NrFact
              WHERE liniifact.NrFact IS NULL)
SET Obs = COALESCE(Obs, ") || 'fara linii!!!"

```

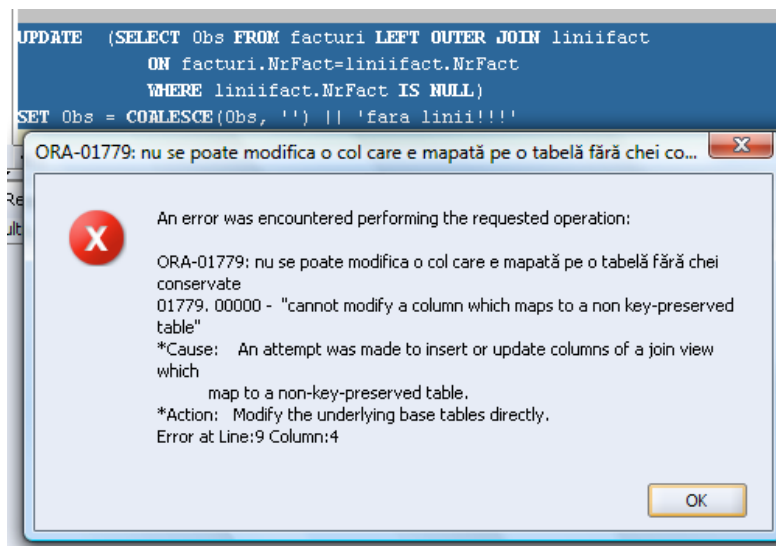



Figura 13.11. Eroare datorată folosirii joncțiunii în interogarea-argument al comenzii UPDATE

De data aceasta, nu este vorba numai de joncțiune externă, ci de orice tip de joncțiune în care condiția nu se exprimă printr-o cheie candidat (vezi și capitolul următor). Același gen de mesaj de eroare se obține și la execuția comenzii:

```
UPDATE      (SELECT Obs FROM facturi INNER JOIN liniifact
              ON facturi.NrFact=liniifact.NrFact)
SET Obs = COALESCE(Obs, '') || 'ARE linii!!!'
```

13.3.5. O comandă recentă: MERGE

Am prezentat comanda MERGE în cartea publicată în 2003 dedicată Oracle⁸, lăsând să se înțeleagă că este o găselniță ceva mai exotică. Comanda a fost inclusă în standard începând cu SQL:2003, iar dintre cele patru servere BD numai dialectele DB2 și Oracle au implementat-o. La drept vorbind, apare și în SQL Server 2008, dar noi am lucrat în această carte preponderent cu versiunea 2005. MERGE este, de fapt, o „combinată” INSERT-UPDATE.

Pentru a ilustra discuția cu un exemplu simplu, să presupunem că utilizatorilor din compartimentul Marketing li se pune la dispoziție o tabelă redundantă numită

⁸ [Fotache s.a.2003]

CLIENTI_PRODUSE_ANI cu atributele *DenCl*, *DenPr*, *UM*, *An* și *Vinzari*, creată astfel:

```
CREATE TABLE clienti_produce_ani AS
    (SELECT DenCl, DenPr, UM, 2007 AS An, 10000000.00 AS Vinzari
    FROM clienti c
        INNER JOIN facturi f ON c.CodCl=f.CodCl
        INNER JOIN liniifact lf ON f.NrFact=lf.Nrfact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
    WHERE 1=2
    )9;
ALTER TABLE clienti_produce_ani ADD CONSTRAINT pk_cpa
    PRIMARY KEY (DenCl, DenPr, UM, An) ;
```

Din rațiuni de securitate (și pentru că cei de la Marketing sunt certați cu responsabilitii IT), angajații compartimentului Marketing au drepturi depline pe această nouă tabelă, însă nu pot consulta tabelele FACTURI și LINIIFACT, ci numai arhivele lunare ale acestora (ex. FACTURI_2007_1, FACTURI_2007_2 s.a.m.d) care li se livrează după fiecare închidere de lună. Prin urmare, după fiecare lună *j* din anul *i*, datele din perechea de tabele FACTURI_*i-j* - LINIIFACT_*i-j* trebuie „vărsate” în CLIENTI_PRODUSE_ANI.

Dacă vă gândiți la INSERT-ul:

```
INSERT INTO clienti_produce_ani
    (SELECT DenCl AS DenumireCl, DenPr AS DenumirePr, UM,
        EXTRACT (YEAR FROM DataFact) AS Anul,
        SUM(Cantitate * PretUnit * ( 1 + ProcTVA)) AS Vinzari_Lunare
    FROM clienti c
        INNER JOIN facturi_2007_1 f ON c.CodCl=f.CodCl
        INNER JOIN liniifact_2007_1 lf ON f.NrFact=lf.Nrfact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
    GROUP BY DenCl, DenPr, UM, EXTRACT (YEAR FROM DataFact)
    )
```

să știți că nu merge decât pentru ianuarie. În februarie, mai mult ca sigur că vom avea vânzări pentru produse/clienti introduse deja din ianurie, iar INSERT-ul va încălca restricția de cheie primară. Nici UPDATE-ul nu e o idee grozavă, pentru

⁹ Sintaxă Oracle. În DB2 trebuie adăugat *DEFINITION ONLY*.

pot apărea valori noi ale „cuplului” client/produs. Așa că, cel mai sigur e combinația UPDATE/INSERT următoare:

– pt. luna august

```
INSERT INTO clienti_produce_ani
    (SELECT DenCl AS DenumireCl, DenPr AS DenumirePr, UM,
        EXTRACT (YEAR FROM DataFact) AS Anul,
        SUM(Cantitate * PretUnit * ( 1 + ProcTVA)) AS Vinzari_Lunare
    FROM clienti c
        INNER JOIN facturi_2007_8 f ON c.CodCl=f.CodCl
        INNER JOIN liniifact_2007_8 lf ON f.NrFact=lf.Nrfact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
    GROUP BY DenCl, DenPr, UM, EXTRACT (YEAR FROM DataFact)
    HAVING (DenCl, DenPr, UM, EXTRACT (YEAR FROM DataFact)) NOT IN
        (SELECT DenCl, DenPr, UM, An FROM clienti_produce_ani )
    );
```

```
UPDATE clienti_produce_ani SET Vinzari = Vinzari +
    (SELECT SUM(Cantitate * PretUnit * ( 1 + ProcTVA))
    FROM clienti c
        INNER JOIN facturi_2007_8 f ON c.CodCl=f.CodCl
        INNER JOIN liniifact_2007_8 lf ON f.NrFact=lf.Nrfact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
    WHERE DenCl= clienti_produce_ani.DenCl AND
        DenPr=clienti_produce_ani.DenPr
        AND EXTRACT (YEAR FROM DataFact) = clienti_produce_ani.An
    );
```

Ei bine, logica celor două comenzi poate fi înglobată într-o comandă MERGE, după cum urmează:

```
MERGE INTO clienti_produce_ani cpa USING
    (SELECT DenCl AS DenumireCl, DenPr AS DenumirePr, UM,
        EXTRACT (YEAR FROM DataFact) AS Anul,
        SUM(Cantitate * PretUnit * ( 1 + ProcTVA)) AS Vinzari_Lunare
    FROM clienti c
        INNER JOIN facturi_2007_1 f ON c.CodCl=f.CodCl
        INNER JOIN liniifact_2007_1 lf ON f.NrFact=lf.Nrfact
        INNER JOIN produse p ON lf.CodPr=p.CodPr
    GROUP BY DenCl, DenPr, UM, EXTRACT (YEAR FROM DataFact)
    ) vinzari_lunare
    ON (cpa.DenCl=vinzari_lunare.DenumireCl AND
        cpa.DenPr=vinzari_lunare.DenumirePr AND
```

```
                cpa.UM=vinzari_lunare.UM AND cpa.An=vinzari_lunare.Anul)
WHEN MATCHED THEN
    UPDATE
        SET Vinzari = Vinzari + vinzari_lunare.Vinzari_Lunare
WHEN NOT MATCHED THEN
    INSERT
        VALUES (vinzari_lunare.DenumireCl, vinzari_lunare.DenumirePr,
            vinzari_lunare.UM, vinzari_lunare.Anul, vinzari_lunare.Vinzari_Lunare) ;
```

Tabela specificată după clauza INTO este cea în care vor fi efectuate actualizările – în cazul nostru CLIENȚI_PRODUSE_ANI (sinonim local CPA). După clauza USING este plasată fraza SELECT care centralizează vânzările pentru fiecare combinație client-produs-unitate de măsură-an. Comanda UPDATE de pe ramura WHEN MATCHED se va executa numai pentru valorile combinației din fraza SELECT care există, la momentul lansării comenzii, în CLIENȚI_PRODUSE_ANI. Valorile combinației ce nu se găsesc în tabela destinație vor fi adăugate tabelii prin intermediul comenzii INSERT de pe ramura WHEN NOT MATCHED.