

Capitolul 18. SQL dinamic

Tema dedicată SQL dinamic (Dynamic SQL) este vecină cu cea a SQL-ului împachetat (Embedded SQL). Cea din urmă privește modul în care comenzile SQL pot fi incluse în programe scrise în Java, C, Basic, Cobol etc. Noi însă ne vom ocupa de prima care privește, în principal, modalitățile în care programele (blocurile) PL/pgSQL, PL/SQL, T-SQL și SQL PL conțin comenzi DDL și DML în care numele obiectelor (tabele, view-uri, attribute, proceduri etc.) sunt cunoscute abia în momentul execuției, nu la scrierea (compilarea) programului.

Cum deja v-ați obișnuit, nu voi epuiza subiectul, ci doar limita la câteva aspecte și avantaje, trecând sub tăcere dezavantajele/pericolele SQL-ului dinamic, cum ar fi viteza redusă de execuție și injecțiile SQL (*SQL injections*).

18.1. SQL dinamic în PostgreSQL PL/pgSQL

PostgreSQL este destul de tolerant în materie de creare de tabele prin programe. Astfel, funcția-procedură *p_test* (listing 18.1) conține o comandă CREATE TABLE. Blocul funcționează fără probleme la apelare:

```
SELECT p_test()
```

Listing 18.1. Funcție simplă PL/pgSQL ce crează o tabelă (la fel de simplă)

```
CREATE OR REPLACE FUNCTION p_test() RETURNS VOID
AS $$
BEGIN
    CREATE TABLE test (x NUMERIC(5), y VARCHAR(50)) ;
END ;
$$ LANGUAGE plpgsql ;
```

Al doilea exemplu prefațează problema arhivării facturilor, în jurul căreia vor gravita exemplele din acest capitol. Pentru a păstra tabela FACTURI cât mai suplă, ne propunem să creăm copii „fragmentate” ale acesteia, cu numele FACTURI_2007_8, FACTURI_2007_9... . Numele tabelelor arhivă indică anul și luna în care au fost emise facturile respective. Până să le creăm, redactăm o funcție care să ștergă toate eventualele arhive – vezi listing 18.2.

Listing 18.2. Funcție-procedură PL/pgSQL ce șterge toate „arhivele” tabelii FACTURI

```
CREATE OR REPLACE FUNCTION p_sterge_facturi1()
RETURNS void AS
$$
DECLARE
    rec_num VARCHAR(30) ;
BEGIN
    FOR rec_num IN (SELECT tablename FROM pg_tables
                    WHERE schemaname='public' AND tablename LIKE 'facturi_%') LOOP
        EXECUTE 'DROP TABLE ' || rec_num ;
    END LOOP;
END;
```

```

END LOOP ;
END ;
$$ LANGUAGE 'plpgsql'

```

Funcția-procedură șterge toate tabelele ale căror nume încep cu *FACTURI_*. Printr-un concurs de împrejurări, existau două arhive anuale, *FACTURI_2006*, și *FACTURI_2007*. Prin lansarea procedurii:

```
SELECT p_sterge_facturi1()
```

scăpăm de ele. Decoperim, însă, și un mod elegant de a bloca PostgreSQL-ul. Dacă după lansare, uităm să reîmprospătăm meniul arborescent (cu ajutorul butonului *Refresh* sau opțiunii cu același nume din meniul contextual), și „scăpăm” un click pe numele uneia dintre cele două tabele proaspăt șterse, avem șanse mari să provocăm înghețarea PostgreSQL-ului – vezi figura 18.1.

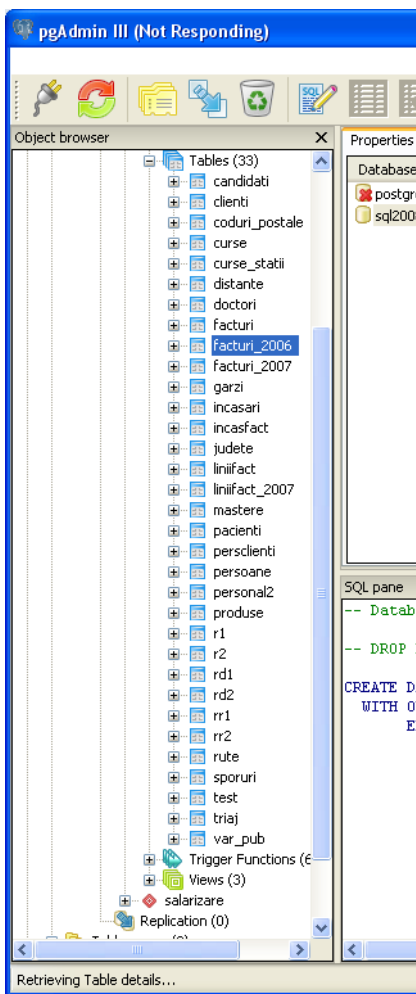


Figura 18.1. O sugestie (dezinteresată) pentru blocarea PostgreSQL-ului

Trecem la probleme ceva mai serioase. Acum, dacă tot funcția-procedură de ștergere merge strună, ar trebui să redactăm și procedura care să creeze arhivele. De data aceasta vom lucra mai îngrijit, adică mai parametrizat. Mai întâi, vom redacta o procedură de ștergere care va elimina din bază o arhivă a tabelului FACTURI pentru un an și o lună date. Eliminarea arhivelor este însă mai bine pusă la punct (prin comparație cu funcția din listing 18.2), pentru că datele legate de facturi se găsesc și în tabelele LINIIFACT, INCASARI (e un pic forțată includerea acestei tabele) și INCASFACT. Așadar, se vor șterge toate cele patru eventuale arhive – vezi listing 18.3.

Listing 18.3. Funcție-procedură PL/pgSQL ce șterge toate „arhivele” tabelului FACTURI

```
CREATE OR REPLACE FUNCTION p_sterge_arhive_fact_luna
(an_ SMALLINT, luna_ SMALLINT) RETURNS void AS
$$
DECLARE
    rec_nume VARCHAR(30);
    v_sir VARCHAR(15);
BEGIN
    v_sir := '_' || an_ || '_' || luna_;
    FOR rec_nume IN (SELECT tablename FROM pg_tables WHERE schemaname='public'
                     AND tablename IN ('incasfact' || v_sir, 'liniifact' || v_sir, 'incasari' || v_sir,
                                      'facturi' || v_sir) ) LOOP
        EXECUTE 'DROP TABLE ' || rec_nume;
    END LOOP;
END;
$$ LANGUAGE 'plpgsql'
```

Ordinea ștergerii nu este importantă, întrucât între cele patru tabele nu sunt definite restricții referențiale, după cum vom vedea în cele ce urmează. Procedura-funcție din listing 18.4 crează, pentru un an și o lună (*an_*, *luna_*) date, tabelele: *FACTURI_an_luna_*, *LINIIFACT_an_luna_*, *INCASARI_an_luna_* și *INCASFACT_an_luna_*. Crearea este precedată de ștergerea celor patru tabele arhivă, folosind procedura anterioară. Se evită, astfel, eroarea declanșată la tentativa de creare unei tabele cu un nume existent în schema curentă.

Listing 18.4. Funcție-procedură PL/pgSQL ce șterge toate „arhivele” tabelului FACTURI

```
CREATE OR REPLACE FUNCTION p_creatoare_arhive_fact_luna(an_ SMALLINT, luna_ SMALLINT)
RETURNS void AS
$$
DECLARE
    v_sir VARCHAR(15);
BEGIN
    EXECUTE p_sterge_arhive_fact_luna(an_, luna_);
    v_sir := '_' || an_ || '_' || luna_;
    EXECUTE 'CREATE TABLE facturi' || v_sir || ' AS SELECT * FROM facturi
            WHERE EXTRACT (YEAR FROM DataFact)= ' || an_ ||
            ' AND EXTRACT(MONTH FROM DataFact) = ' || luna_;
    EXECUTE 'CREATE TABLE liniifact' || v_sir || ' AS SELECT * FROM liniifact
            WHERE NrFact IN (SELECT NrFact FROM facturi' || v_sir || ')';
    EXECUTE 'CREATE TABLE incasari' || v_sir || ' AS SELECT * FROM incasari WHERE
            EXTRACT (YEAR FROM DataInc)= ' || an_ ||
            ' AND EXTRACT(MONTH FROM DataInc) = ' || luna_ ||
            ' UNION SELECT * FROM incasari WHERE CodInc IN ('
```

```

SELECT CodInc FROM incasfact WHERE NrFact IN (
    SELECT NrFact FROM facturi WHERE
    EXTRACT (YEAR FROM DataFact)= ' || an_ ||
    ' AND EXTRACT(MONTH FROM DataFact) = ' || luna_ || ' )';
EXECUTE 'CREATE TABLE incasfact' || v_sir || ' AS SELECT * FROM incasfact
WHERE CodInc IN (SELECT CodInc FROM incasari' || v_sir || ' )';
END ;
$$ LANGUAGE 'plpgsql'

```

În mod normal, ar trebui ca, după arhivare, să ștergem înregistrările plasate în tabele-arhivă din cele patru tabele „principale” – FACTURI, LINIIFACT, INCASARI și INCASFACT. Deocamdată, nu avem curajul acesta, câștigul din folosirea tabelelor arhivă fiind unul limitat. Testăm funcționarea procedurii-funcție:

```

SELECT p_creare_arhive_fact_luna(CAST (2007 AS SMALLINT),
                                CAST (1 AS SMALLINT))

```

Figura 18.2 surprinde ceea ce afișează meniul arborescent PostgreSQL după lansarea funcție (și reîmprospătarea meniului). Faptul că cele patru tabele arhivă sunt cam goale în cazul nostru are o importanță secundară.

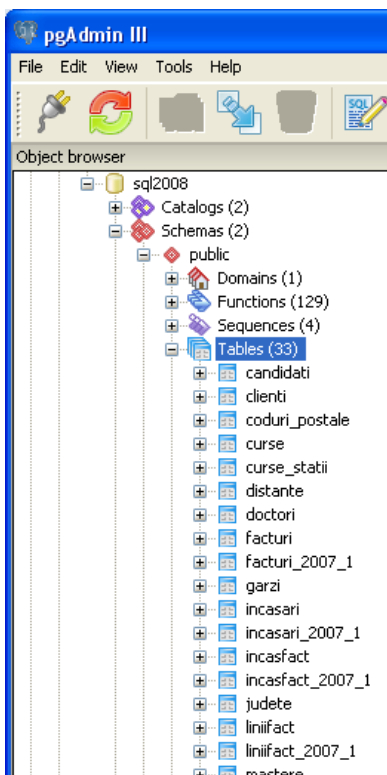


Figura 18.2. Verificarea procedurii de arhivare a facturilor (pentru anul 2007, luna 1)

Mai creăm tabele-arhivă și pentru lunile august, septembrie și octombrie 2007:

```

SELECT p_creare_arhive_fact_luna(CAST (2007 AS SMALLINT),

```

```

CAST (8 AS SMALLINT)) ;
SELECT p_creare_arhive_fact_luna(CAST (2007 AS SMALLINT),
CAST (9 AS SMALLINT)) ;
SELECT p_creare_arhive_fact_luna(CAST (2007 AS SMALLINT),
CAST (10 AS SMALLINT)) ;

```

Înainte de a redacta cea mai interesantă funcție, cea care obține setul de facturi pentru orice interval calendaristic pe baza tabelor-arhivă, revenim la o problemă legată de declanșatoare. În paragraful 17.2 formulăm problema protejării atributelor calculate ale tabelii FACTURI de modificarea lor directă, și acceptarea doar a modificărilor produse de declanșatoarele tabelii LINIIFACT. Neavând la dispoziție variabile publice (precum cele din Oracle), am creat o tabelă în care adăugăm, pentru fiecare utilizator conectat la baza de date, câte o înregistrare pentru fiecare variabilă publică (cum ar fi *v_trg_liniiifact*).

Soluția are câteva limite. Poate cea mai importantă este legată de faptul că pot exista două conexiuni la baza de date ale aceluiași utilizator. Este de evitat ca persoane diferite să folosească același cont la folosirea bazei de date. Se mai întâmplă, însă. Plus că, uneori, pentru a face diferite teste, un utilizator poate să deschidă două sau mai multe sesiuni pe același calculator. Dacă mai multe sesiuni operează editări ale tabelor FACTURI și LINIIFACT, declanșatoarele din paragraful 17.2 nu sunt în stare să recunoască decât un singur utilizator, iar consecințele pot fi neplăcute.

Pentru a elimina acest neajuns, ne putem gândi la folosirea unei tabele temporare în locul celei „clasice”. În PostgreSQL tabelele temporare sunt locale, deci se vor șterge la finalizarea sesiunii. Este exact ce ne trebuie pentru a gestiona modificarea conținutului tabelor FACTURI și LINIIFACT. Funcția *f_variabile_publice* din listing-ul 18.5 verifică existența tabelii temporare VARIABILE_PUBLICE și, dacă este nevoie, o creează. De asemenea, se verifică dacă în tabela temporară se găsește o înregistrare pentru variabila publică parametru (*variabila_*). În caz că se găsește, se returnează valoarea curentă a acesteia.

Listing 18.5. Funcție-procedură ce crează o tabelă temporară locală pe post de variabilă publică

```

-- functia F_VARIABILE_PUBLICE creaza, la prima apelare, tabela VARIABILE_PUBLICE
-- si, daca e nevoie, insereaza cite o linie pentru fiecare variabila publica: V_TRG_LINIIFACT ....
-- pseudo-variabilele de tip BOOLEAN sunt pe setate, la creare, pe FALSE
CREATE OR REPLACE FUNCTION f_variabile_publice (variabila_ VARCHAR)
    RETURNS BOOLEAN AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM pg_tables WHERE tablename='variabile_publice') THEN
        EXECUTE 'CREATE LOCAL TEMPORARY TABLE variabile_publice (
            variabila VARCHAR(50),
            valoare BOOLEAN)' ;
        INSERT INTO variabile_publice VALUES (variabila_, FALSE) ;
        RETURN FALSE ;
    ELSE
        IF NOT EXISTS (SELECT 1 FROM variabile_publice WHERE variabila=variabila_) THEN
            INSERT INTO variabile_publice VALUES (variabila_, FALSE) ;

```

```

        RETURN FALSE ;
    ELSE
        RETURN (SELECT valoare FROM variabile_publice WHERE variabila=variabila_) ;
    END IF ;
END IF ;
END ;
$$LANGUAGE plpgsql ;

```

Acum trebuie să refacem și declanșatoarele. Să începem cu cel de modificare pentru FACTURI – vezi listing 18.6.

Listing 18.6. Noua variantă a declanșatorului de modificare pentru FACTURI

```

-- se verifică modul de modificare a atributelor TVA si VALTOTALA
CREATE OR REPLACE FUNCTION trg_facturi_upd()
RETURNS TRIGGER AS $trg_facturi_upd$
BEGIN
    IF NEW.tva <> OLD.tva OR NEW.valtotala <> OLD.valtotala THEN
        IF f_variabile_publice ('v_trg liniifact') = FALSE THEN
            RAISE EXCEPTION
                'Nu puteti modifica interactiv nici valoarea facturii, nici TVA-ul acesteia !' ;
        END IF ;
    END IF ;
    RETURN NEW ;
END ;
$trg_facturi_upd$ LANGUAGE plpgsql ;

```

```

DROP TRIGGER IF EXISTS trg_facturi ON facturi ;
DROP TRIGGER IF EXISTS trg_facturi_upd ON facturi ;
CREATE TRIGGER trg_facturi_upd AFTER UPDATE ON facturi
    FOR EACH ROW EXECUTE PROCEDURE trg_facturi_upd() ;

```

Continuăm cu declanșatoarele tabeli LINIIFACT: cel de inserare (vezi listing 18.7), cel de modificare (listing 18.8) și cel de ștergere (listing 18.9).

Listing 18.7. Noua variantă a declanșatorului de inserare pentru LINIIFACT

```

CREATE OR REPLACE FUNCTION trg liniifact_ins() RETURNS TRIGGER AS $trg liniifact_ins$
DECLARE
    v BOOLEAN ;
BEGIN
    v := p_variabile_publice('liniifact') ;
    UPDATE variabile_publice SET valoare = TRUE WHERE variabila = 'v_trg liniifact' ;

    NEW.tvalinie := NEW.cantitate * NEW.pretunit * f_proctva(NEW.codpr) ;
    UPDATE facturi SET tva = tva + NEW.tvalinie,
        valtotala = valtotala + NEW.cantitate * NEW.pretunit + NEW.tvalinie
        WHERE nrfact = NEW.nrfact ;

    -- refacem valoarea pseudo-variabilei
    UPDATE variabile_publice SET valoare = FALSE WHERE variabila = 'v_trg liniifact' ;
    RETURN NEW ;
END ;
$trg liniifact_ins$ LANGUAGE plpgsql ;

```

```
DROP TRIGGER IF EXISTS trg_liniiifact_ins ON liniiifact ;
CREATE TRIGGER trg_liniiifact_ins BEFORE INSERT ON liniiifact
FOR EACH ROW EXECUTE PROCEDURE trg_liniiifact_ins() ;
```

Listing 18.8. Noul declanșator de modificare pentru LINIIFACT

```
CREATE OR REPLACE FUNCTION trg_liniiifact_upd_br()
  RETURNS TRIGGER AS $trg_liniiifact_upd_br$
DECLARE
  v BOOLEAN ;
BEGIN
  v := f_variabile_publice('v_trg_liniiifact') ;
  UPDATE variabile_publice SET valoare = TRUE WHERE variabila = 'v_trg_liniiifact' ;

  IF NEW.nrfact = OLD.nrfact THEN      -- nu s-a schimbat nrfact !!!!
    IF NEW.codpr <> OLD.codpr OR NEW.cantitate <> OLD.cantitate OR
       NEW.pretunit <> OLD.pretunit OR NEW.tvalinie <> OLD.tvalinie THEN
      NEW.tvalinie := NEW.cantitate * NEW.pretunit * f_proctva(NEW.codpr) ;
      UPDATE facturi SET tva = tva - OLD.tvalinie + NEW.tvalinie,
        valtotala = valtotala - (OLD.cantitate * OLD.pretunit + OLD.tvalinie)
        + (NEW.cantitate * NEW.pretunit + NEW.tvalinie)
        WHERE nrfact = NEW.nrfact ;
    END IF ;
  ELSE -- vechile valori trebuie scazute de la "vechea" factura
    -- iar noile valori adaugate "noi" facturi
    UPDATE facturi SET tva = tva - OLD.tvalinie,
      valtotala = valtotala - (OLD.cantitate * OLD.pretunit + OLD.tvalinie)
    WHERE nrfact = OLD.nrfact ;

    NEW.tvalinie := NEW.cantitate * NEW.pretunit * f_proctva(NEW.codpr) ;
    UPDATE facturi SET tva = tva + NEW.tvalinie,
      valtotala = valtotala + (NEW.cantitate * NEW.pretunit + NEW.tvalinie)
    WHERE nrfact = NEW.nrfact ;
  END IF ;
  -- refacem valoarea pseudo-variabilei
  UPDATE variabile_publice SET valoare = FALSE WHERE variabila = 'v_trg_liniiifact' ;
  RETURN NEW ;
END ;
$trg_liniiifact_upd_br$ LANGUAGE plpgsql ;
```

```
DROP TRIGGER IF EXISTS trg_liniiifact_upd_br ON liniiifact ;
CREATE TRIGGER trg_liniiifact_upd_br BEFORE INSERT ON liniiifact
FOR EACH ROW EXECUTE PROCEDURE trg_liniiifact_upd_br() ;
```

Listing 18.9. Declanșatorul de ștergere pentru tabela LINIIFACT

```
CREATE OR REPLACE FUNCTION trg_liniiifact_del2()
  RETURNS TRIGGER AS $trg_liniiifact_del2$
DECLARE
  v BOOLEAN ;
BEGIN
  v := f_variabile_publice('v_trg_liniiifact') ;
  UPDATE variabile_publice SET valoare = TRUE WHERE variabila = 'v_trg_liniiifact' ;
  UPDATE facturi
    SET tva = tva - OLD.tvalinie,
```

```

        valtotala = valtotala - OLD.cantitate * OLD.pretunit - OLD.tvalinie
        WHERE nrfact = OLD.nrfact ;
        UPDATE variabile_publice SET valoare = FALSE WHERE variabila = 'v_trg_liniiifact' ;
        RETURN OLD ;
    END ;
    $trg_liniiifact_del2$ LANGUAGE plpgsql ;

```

```

DROP TRIGGER IF EXISTS trg_liniiifact_del2 ON liniiifact ;
CREATE TRIGGER trg_liniiifact_del2 BEFORE DELETE ON liniiifact
FOR EACH ROW EXECUTE PROCEDURE trg_liniiifact_del2() ;

```

Ajungem, astfel, și la funcția care fuzionează arhivele pentru a furniza setul de facturi pentru orice interval dorit de către utilizator. Pentru încălzire, redactăm o funcție care verifică existența unei tabele parametru în schema PUBLIC – vezi listing 18.10.

Listing 18.10. Funcție ce semnalizează existența (sau inexistența) unei tabele

```

CREATE OR REPLACE FUNCTION f_exista_tabela (nume_ VARCHAR)
    RETURNS BOOLEAN AS
$$
BEGIN
    RETURN EXISTS (SELECT tablename FROM pg_tables WHERE schemaname='public'
                    AND tablename = nume_) ;
END ;
$$ LANGUAGE 'plpgsql'

```

Listing-ul 18.11 conține funcția în care punem în valoare adevărata forță a SQL-ului dinamic în PL/pgSQL. Bucla principală se execută pentru fiecare an din intervalul *data_inceput* – *data_final*. Bucla inclusă în aceasta parcurge toate lunile de pe anul în curs (*v_an*). Pentru primul an din interval, se începe cu luna din *data_inceput* și se termină cu 12 (luna decembrie). În cazul anului final din interval, luna de început este 1 (ianuarie), iar cea finală este extrasă din *data_final*. Pentru lunile de început și sfârșit se face o filtrare suplimentară, prin variabila *clauza_where*. Pentru fiecare combinație (*an_*, *luna_*) se verifică dacă există tabela arhivă corespunzătoare și, dacă da, se folosește SQL dinamic și o variabilă cursor care va returna toate înregistrările (eventual, filtrate) din tabela-arhivă pentru luna/anul curente care nu se găsesc în tabelele principale.

Listing 18.11. Funcție de reconstituire a facturilor pentru un interval calendaristic oarecare

```

CREATE OR REPLACE FUNCTION f_facturi (data_inceput DATE, data_final date)
    RETURNS SETOF facturi AS
$$
DECLARE
    v_an SMALLINT ;
    v_luna SMALLINT ;
    v_luna_inceput SMALLINT ;
    v_luna_final SMALLINT ;
    v_nume VARCHAR(30) ;
    rec1 RECORD ;
    v_clauza_where VARCHAR(1000) := '' ;

```



```

BEGIN
  FOR v_an IN EXTRACT(YEAR FROM data_inceput)..EXTRACT(YEAR FROM data_final)
  LOOP

    v_luna_inceput = CASE WHEN v_an = EXTRACT(YEAR FROM data_inceput)
                        THEN EXTRACT(MONTH FROM data_inceput)
                        ELSE 1
                    END ;
    v_luna_final = CASE WHEN v_an = EXTRACT(YEAR FROM data_final)
                      THEN EXTRACT(MONTH FROM data_final)
                      ELSE 12
                    END ;

    FOR v_luna IN v_luna_inceput..v_luna_final LOOP
      v_num := 'facturi_' || v_an || '-' || v_luna ;

      IF f_exista_tabela(v_num) THEN
        v_clauza_where := ' WHERE NrFact NOT IN (SELECT NrFact FROM facturi) ' ;
        IF v_an = EXTRACT(YEAR FROM data_inceput) AND v_luna =
          EXTRACT(MONTH FROM data_inceput) THEN
          v_clauza_where := v_clauza_where ||
            ' AND DataFact >= DATE''' || data_inceput || '''' ;
        END IF ;
        IF v_an = EXTRACT(YEAR FROM data_final) AND v_luna =
          EXTRACT(MONTH FROM data_final) THEN
          v_clauza_where := v_clauza_where ||
            ' AND DataFact <= DATE''' || data_final || '''' ;
        END IF ;
        FOR rec1 IN EXECUTE 'SELECT * FROM ' || v_num ||
          v_clauza_where LOOP
          RETURN NEXT rec1 ;
        END LOOP ;
      END IF ;

    END LOOP ;
  END LOOP ;
  FOR rec1 IN EXECUTE ' SELECT * FROM facturi WHERE DataFact BETWEEN DATE''' ||
    data_inceput || '''' || ' AND DATE''' || data_final || '''' LOOP
    RETURN NEXT rec1 ;
  END LOOP ;
END ;
$$ LANGUAGE 'plpgsql'

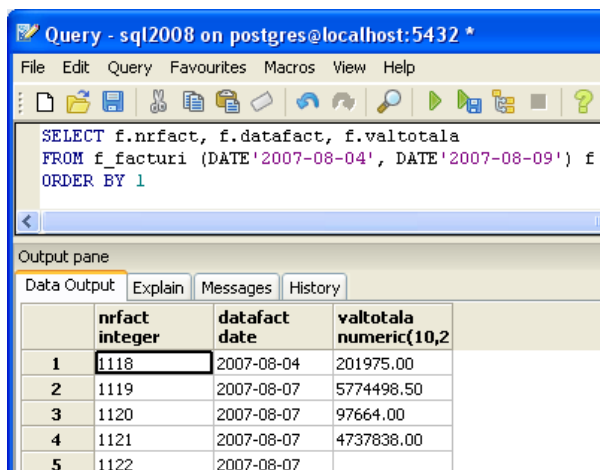
```

După ieșirea din bucla principală, se parcurg și liniile „rămase” în tabela **FAC-TURI** care se încadrează în intervalul dat. Această operațiune este necesară deoarece, la momentul execuției funcției *f_facturi*, este posibil ca arhivarea să se fi făcut numai pentru o parte dintre lunile intervalului, sau poate chiar pentru niciuna. La testarea funcției descoperim că lucrurile chiar sunt în regulă (vezi figura 18.3):

```

SELECT f.nrfact, f.datafact, f.valtotala
FROM f_facturi (DATE'2007-08-04', DATE'2007-08-09') f
ORDER BY 1

```



Query - sql2008 on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

```

SELECT f.nrfact, f.datafact, f.valtotala
FROM f_facturi (DATE'2007-08-04', DATE'2007-08-09') f
ORDER BY 1

```

Output pane

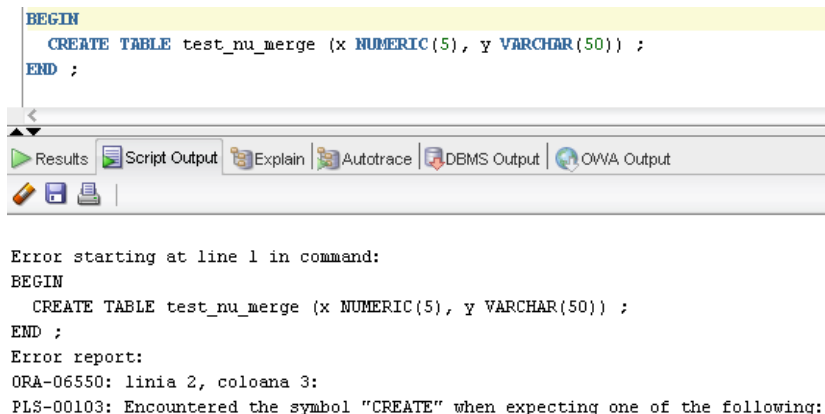
Data Output Explain Messages History

	nrfact integer	datafact date	valtotala numeric(10,2)
1	1118	2007-08-04	201975.00
2	1119	2007-08-07	5774498.50
3	1120	2007-08-07	97664.00
4	1121	2007-08-07	4737838.00
5	1122	2007-08-07	

Figura 18.3. Verificarea procedurii de reconstituire a facturilor

18.2. SQL Dinamic în PL/SQL

Oracle PL/SQL pune la dispoziție două tipuri de SQL dinamic, prin pachetul standard DBMS_SQL (furnizat la instalare) și prin comanda EXECUTE IMMEDIATE. Noi vom folosi exclusiv a doua variantă, referită în documentații drept NDS (Native Dynamic SQL). Din capul locului trebuie precizat că orice modul PL/SQL este necooperant în materie de comenzi DDL - vezi, pentru ilustrare, blocul anonim din figura 18.4.



```

BEGIN
  CREATE TABLE test_nu_merge (x NUMERIC(5), y VARCHAR(50)) ;
END ;

```

Results Script Output Explain Autotrace DBMS Output OWA Output

Error starting at line 1 in command:

```

BEGIN
  CREATE TABLE test_nu_merge (x NUMERIC(5), y VARCHAR(50)) ;
END ;

```

Error report:

ORA-06550: linia 2, coloana 3:

PLS-00103: Encountered the symbol "CREATE" when expecting one of the following:

Figura 18.4. Imposibilitatea lansării unei comenzi CREATE TABLE dintr-un bloc PL/SQL

În paragraful 16.3 spuneam că puteam salva un set de înregistrări într-o tabelă temporară și, totodată, amânam discuția pentru acum. Beneficiem de avantajul

pachetelor PL/SQL, așa că grupăm toate funcțiile și procedurile de arhivare în *pac_arhivare*. După cum se poate vedea în listing 18.12, pachetul conține două „istanțe” ale funcției *f_există_tabelă*, o procedură care șterge arhivele pentru o lună și un an date, *p_sterge_arhive_fact*, o procedură care face arhivarea pentru un an/lună date, *p_creare_arhive_fact_luna*, o funcție care, pentru intervalul (*data_inceput*, *data_final*) dat furnizează șirul de caractere (SELECT * ... UNION SELECT *....) care, lansat în execuție, va reconstitui toate facturile emise în perioada respectivă.

Listing 18.12. Specificațiile pachetului *pac_arhivare*

```
=====
-- pachetul pac_arhivare este cel care exemplifica o serie de optiuni de tip SQL dinamic
=====
CREATE OR REPLACE PACKAGE pac_arhivare AUTHID CURRENT_USER AS

-- o functie "supraincarcata" ce verifica existenta unei tabele
FUNCTION f_exista_tabela (tabela_ VARCHAR2, an_ NUMBER, luna_ NUMBER)
    RETURN BOOLEAN ;
FUNCTION f_exista_tabela (tabela_ VARCHAR2) RETURN BOOLEAN ;

-- procedura ce sterge o arhiva lunara
PROCEDURE p_sterge_arhive_fact_luna (an_ NUMBER, luna_ NUMBER) ;

-- procedura P_CREATE_ARHIVE_FACT_LUNA creaza o arhiva pt. fiecare tabela "tranzactionala"
-- cu inregistrările pe un an si luna, apoi sterge din tabela principala inregistrările
-- arhivate (de ex. FACTURI pentru luna 8 din 2007 - se creeaza FACTURI_2007_8
-- se muta acolo toate facturile emise (DataFact) in luna august 2007
PROCEDURE p_creare_arhive_fact_luna (an_ NUMBER, luna_ NUMBER) ;

-----
-- urmeaza o procedura si o functie pentru "recompunerea" dinamica a inregistrărilor arhivate

-- sirul care reuneste toate facturile pentru un interval dat
FUNCTION f_sir_facturi (data_inceput DATE, data_final DATE) RETURN VARCHAR2 ;

-- varianta 1 de reconstituire facturi : tabela temporara globala
PROCEDURE p_fuziune_facturi1 (datai DATE, dataf DATE) ;

END pac_arhivare ;
=====
```

Există câteva comentarii importante și despre corpul pachetului (listing 18.13). Procedura de ștergere a arhivelor (*p_sterge_arhive_fact_luna*) este ceva mai precaută. Pentru a nu se pierde iremediabil înregistrările care se găsesc în tabelele-arhivă, acestea (înregistrările) se re-adăugă în tabelele din care au fost extrase la momentul arhivării. Adăugarea trebuie făcută selectiv pentru a nu se încălca restricțiile de cheie primară. În plus, pentru ca triggerul *trg_liniiifact_ins* să nu genereze eroarea de mutanță a tabeli LINIIFACT, în loc de *INSERT INTO liniiifact SELECT...*, folosim o variabilă cursor pentru a re-introduce eventualele înregistrări din arhivă în LINIIFACT una câte una.

Întrucât înregistrările din arhive trebuie adăugate în tabelele principale fără a se mai executa declanșatoarele, am inclus în procedura de arhivare și comenzi EXECUTE IMMEDIATE de dezactivare și, ulterior, reactivare a declanșatoarelor

trg_facturi_ins și *trg_liniiifact_ins*. Exemplu este mai mult didactic, întrucât activarea și dezactivarea declanșatoarelor este nerecomandată în aplicații complexe, cu mulți utilizatori. Mai nimerită era modificarea declanșatoarelor pentru a nu executa anumite comenzi în cazul arhivării/dezarhivării (putem folosi una sau mai multe variabile publice în acest scop). Observația este valabilă și pentru dezactivarea/activarea declanșatoarelor de ștergere din procedura de arhivare.

Arhivarea facturilor și încasărilor pentru un an și lună date (*p_creatoare_arhive_fact_luna*) poate fi făcută acum fără grijă, și începe cu apelul procedurii de ștergere de care vorbeam acum câteva rânduri. De notat și, eventual, corectat (dacă nu sunt în regulă) predicatele aplicate la arhivare celor patru tabele tranzacționale.

Listing 18.13. Corpul pachetului *pac_arhivare*

```

=====
CREATE OR REPLACE PACKAGE BODY "PAC_ARHIVARE" AS
=====
-----
FUNCTION f_exista_tabela(tabela_ VARCHAR2, an_ NUMBER, luna_ NUMBER)
RETURN boolean
IS
  v_tabela VARCHAR2(100);
  v_unu NUMBER(1);
BEGIN
  v_tabela := UPPER(tabela_) || '_' || LTRIM(to_char(an_, '9999')) || '_' || LTRIM(to_char(luna_, '99'));
  SELECT 1 INTO v_unu FROM user_tables
  WHERE TABLE_NAME = v_tabela;
  RETURN TRUE;
EXCEPTION
WHEN no_data_found THEN
  RETURN FALSE;
END f_exista_tabela;
-----
FUNCTION f_exista_tabela(tabela_ VARCHAR2) RETURN boolean IS v_tabela VARCHAR2(100);
v_unu NUMBER(1);
BEGIN
  v_tabela := UPPER(tabela_);
  SELECT 1 INTO v_unu FROM user_tables WHERE TABLE_NAME = v_tabela;
  RETURN TRUE;
EXCEPTION
WHEN no_data_found THEN
  RETURN FALSE;
END f_exista_tabela;
-----
PROCEDURE p_sterge_arhive_fact_luna (an_ NUMBER, luna_ NUMBER)
IS
  v_sir VARCHAR2(15) ;
  v_sir2 VARCHAR2(2000) ;
  TYPE tRefCursor IS REF CURSOR;
  vRefCursor tRefCursor;
  rec_lf liniifact%ROWTYPE ;
BEGIN
  v_sir := '_' || an_ || '_' || luna_ ;
  -- pentru a nu pierde inregistrările arhivate le readaugăm la tabelele "curente"
  EXECUTE IMMEDIATE 'ALTER TRIGGER trg_facturi_ins DISABLE' ;
  EXECUTE IMMEDIATE 'ALTER TRIGGER trg_liniiifact_ins DISABLE' ;

```

```

IF f_exista_tabela ('FACTURI', an_, luna_) THEN
    EXECUTE IMMEDIATE 'INSERT INTO facturi SELECT * FROM facturi' || v_sir ||
    ' WHERE NrFact NOT IN (SELECT NrFact FROM facturi)';
END IF;
IF f_exista_tabela ('INCASARI', an_, luna_) THEN
    EXECUTE IMMEDIATE 'INSERT INTO incasari SELECT * FROM incasari' || v_sir ||
    ' WHERE CodInc NOT IN (SELECT CodInc FROM incasari)';
END IF;
IF f_exista_tabela ('LINIIFACT', an_, luna_) THEN
    v_sir2 := 'SELECT * FROM liniifact' || v_sir || ' WHERE (NrFact, Linie) NOT IN ('
    || '(SELECT NrFact, Linie FROM liniifact ) ORDER BY NrFact, Linie ' ;
    OPEN vrefcursor FOR v_sir2 ;
    LOOP
        FETCH vRefCursor INTO rec_If ;
        EXIT WHEN vRefCursor%NOTFOUND ;
        INSERT INTO liniifact VALUES rec_If ;
    END LOOP;
    CLOSE vrefcursor;
END IF;
IF f_exista_tabela ('INCASFACT', an_, luna_) THEN
    EXECUTE IMMEDIATE 'INSERT INTO incasfact SELECT * FROM incasfact' || v_sir ||
    ' WHERE (CodInc, NrFact) NOT IN (SELECT CodInc, NrFact FROM incasfact)';
END IF;

EXECUTE IMMEDIATE 'ALTER TRIGGER trg_facturi_ins ENABLE';
EXECUTE IMMEDIATE 'ALTER TRIGGER trg liniifact_ins ENABLE';

FOR rec_num IN (SELECT table_name FROM user_tables
    WHERE table_name IN ( 'INCASFACT' || v_sir, 'LINIIFACT' || v_sir,
    'INCASARI' || v_sir, 'FACTURI' || v_sir ) ) LOOP
    EXECUTE IMMEDIATE 'DROP TABLE ' || rec_num.table_name ;
END LOOP;
END p_sterge_arhive_fact_luna ;

-----
PROCEDURE p_creatoare_arhive_fact_luna (an_ NUMBER, luna_ NUMBER)
IS
    v_sir VARCHAR2(15);
    v_tabela VARCHAR2(100);
BEGIN
    p_sterge_arhive_fact_luna (an_, luna_);

    v_tabela := 'FACTURI' || '_' || LTRIM(TO_CHAR(an_, '9999')) || '_' ||
    LTRIM(TO_CHAR(luna_, '99'));
    IF f_exista_tabela ('FACTURI', an_, luna_) THEN
        raise_application_error(-20120, 'A fost facuta deja arhivarea pe luna '
        || luna_ || ', anul ' || an_);
    END IF;

    v_sir := '_' || an_ || '_' || luna_ ;
    EXECUTE IMMEDIATE 'CREATE TABLE facturi' || v_sir || ' AS SELECT * FROM facturi
    WHERE EXTRACT (YEAR FROM DataFact)= ' || an_ ||
    ' AND EXTRACT(MONTH FROM DataFact) = ' || luna_ ;
    EXECUTE IMMEDIATE 'CREATE TABLE liniifact' || v_sir ||
    ' AS SELECT * FROM liniifact
    WHERE NrFact IN (SELECT NrFact FROM facturi' || v_sir || ' )';

    EXECUTE IMMEDIATE 'CREATE TABLE incasari' || v_sir ||

```

```

' AS SELECT * FROM incasari WHERE
    EXTRACT (YEAR FROM DataInc)= ' || an_ ||
    ' AND EXTRACT(MONTH FROM DataInc) = ' || luna_ ||
' UNION SELECT * FROM incasari WHERE CodInc IN (
    SELECT CodInc FROM incasfact WHERE NrFact IN (
        SELECT NrFact FROM facturi WHERE
        EXTRACT (YEAR FROM DataFact)= ' || an_ ||
        ' AND EXTRACT(MONTH FROM DataFact) = ' || luna_ || ' ) )';

EXECUTE IMMEDIATE 'CREATE TABLE incasfact' || v_sir ||
' AS SELECT * FROM incasfact
    WHERE NrFact IN (SELECT NrFact FROM facturi' || v_sir || ')';

EXECUTE IMMEDIATE 'ALTER TRIGGER trg_liniifact_del1 DISABLE';
EXECUTE IMMEDIATE 'ALTER TRIGGER trg_liniifact_del2 DISABLE';
EXECUTE IMMEDIATE 'ALTER TRIGGER trg_liniifact_del3 DISABLE';

EXECUTE IMMEDIATE 'DELETE FROM incasfact WHERE (CodInc, NrFact) IN '
    || '(SELECT CodInc, NrFact FROM incasfact' || v_sir || ')';
EXECUTE IMMEDIATE 'DELETE FROM liniifact WHERE (NrFact, Linie) IN '
    || '(SELECT NrFact, Linie FROM liniifact' || v_sir || ')';
EXECUTE IMMEDIATE 'DELETE FROM facturi WHERE NrFact IN '
    || '(SELECT NrFact FROM facturi' || v_sir ||
    ') AND ValTotala = ValIncasata AND NrFact NOT IN (SELECT NrFact FROM incasfact)';
EXECUTE IMMEDIATE 'DELETE FROM incasari WHERE CodInc IN (SELECT CodInc FROM ' ||
    'incasari' || v_sir || ') AND CodInc NOT IN (SELECT CodInc FROM incasfact)';

EXECUTE IMMEDIATE 'ALTER TRIGGER trg_liniifact_del1 ENABLE';
EXECUTE IMMEDIATE 'ALTER TRIGGER trg_liniifact_del2 ENABLE';
EXECUTE IMMEDIATE 'ALTER TRIGGER trg_liniifact_del3 ENABLE';
END p_creatoare_arhive_fact_luna;

-----
FUNCTION f_sir_facturi (data_inceput DATE, data_final DATE) RETURN VARCHAR2
IS
    v_luna_inceput NUMBER(2);
    v_luna_final NUMBER(2);
    v_nume VARCHAR2(30);
    v_clauza_where VARCHAR2(500) := '';
    v_sir VARCHAR2(5000) := '';
BEGIN
    FOR v_an IN EXTRACT(YEAR FROM data_inceput)..EXTRACT(YEAR FROM data_final)
    LOOP
        IF v_an = EXTRACT(YEAR FROM data_inceput) THEN
            v_luna_inceput := EXTRACT(MONTH FROM data_inceput);
        ELSE
            v_luna_inceput := 1;
        END IF;
        v_luna_final := CASE WHEN v_an = EXTRACT(YEAR FROM data_final)
            THEN EXTRACT(MONTH FROM data_final) ELSE 12 END;
        FOR v_luna IN v_luna_inceput..v_luna_final LOOP
            v_nume := 'facturi_' || v_an || '_' || v_luna;
            IF f_exista_tabela(v_nume) THEN
                v_clauza_where := ' WHERE NrFact NOT IN (SELECT NrFact FROM facturi)';
                IF v_an = EXTRACT(YEAR FROM data_inceput) AND v_luna =
                    EXTRACT(MONTH FROM data_inceput) THEN
                    v_clauza_where := v_clauza_where || ' AND DataFact >= DATE'''
                        || data_inceput || '''';
                END IF;
            END IF;
        END LOOP;
    END LOOP;

```

```

        IF v_an = EXTRACT(YEAR FROM data_final) AND v_luna =
            EXTRACT(MONTH FROM data_final) THEN
            v_clauza_where := v_clauza_where ||
                ' AND DataFact <= DATE' || data_final || '' ;
        END IF ;
        v_sir := CASE WHEN v_sir = '' THEN v_sir ELSE v_sir || ' UNION ' END
            || 'SELECT * FROM ' || v_nume || v_clauza_where ;
    END IF ;
END LOOP ;
END LOOP ;
v_sir := CASE WHEN v_sir = '' THEN v_sir ELSE v_sir || ' UNION ' END
    || 'SELECT * FROM facturi WHERE DataFact BETWEEN DATE' ||
        data_inceput || '' || ' AND DATE' || data_final || '' ;
RETURN v_sir ;
END f_sir_facturi ;

-----
PROCEDURE p_fuziune_facturi1(datai DATE, dataf DATE)
IS
    v_sir VARCHAR2(5000) := ' ' ;
    v_luna_initiala NUMBER(2);
    v_luna_finala NUMBER(2);
    v_an_initial NUMBER(4) := EXTRACT(YEAR FROM datai);
    v_an_final NUMBER(4) := EXTRACT(YEAR FROM dataf);
BEGIN
    v_sir := f_sir_facturi (datai, dataf) ;
    --DBMS_OUTPUT.PUT_LINE(v_sir) ;
    IF pac_arhivare.f_exista_tabela('TEMP_FACTURI') THEN
        EXECUTE IMMEDIATE 'TRUNCATE TABLE TEMP_FACTURI' ;
        EXECUTE IMMEDIATE 'DROP TABLE TEMP_FACTURI' ;
    END IF ;
    EXECUTE IMMEDIATE
        'CREATE GLOBAL TEMPORARY TABLE temp_facturi ON COMMIT '
        || 'PRESERVE ROWS AS SELECT * FROM (' || v_sir || ') ' ;

    -- dupa lansarea acestei proceduri, se face un SELECT pe tabela virtuala TEMP_facturi
    -- SELECT * FROM temp_facturi
END p_fuziune_facturi1;

END pac_arhivare;

=====
/

```

Lansăm arhivarea pe lunile august, septembrie și octombrie 2007:

```

BEGIN
    pac_arhivare.p_creaire_arhive_fact_luna (2007, 8) ;
    pac_arhivare.p_creaire_arhive_fact_luna (2007, 9) ;
    pac_arhivare.p_creaire_arhive_fact_luna (2007, 10) ;
END ;

```

După cum se poate vedea și în figura 18.5, chiar și după arhivare, o parte din facturi rămân în continuare în FACTURI (pentru luna septembrie 2007 toate cele șapte facturi). Explicația este că aceste facturi comune nu sunt, la momentul arhivării, și încasate total, deci o să apară în înregistrări ulterioare în INCASFACT.

Enter SQL Statement:

```

SELECT MrFact, DataFact, CodCl, ValTotala, TVA, ValIncasata
FROM facturi_2007_9
INTERSECT
SELECT MrFact, DataFact, CodCl, ValTotala, TVA, ValIncasata
FROM facturi

```

Results: Script Output Explain Autotrace DBMS Output OWA Output

	NRFACT	DATAFACT	CODCL	VALTOTALA	TVA	VALINCASATA
1	3111	01-09-2007	1001	4442959,5	702409,5	0
2	3112	01-09-2007	1005	153442	17142	0
3	3113	02-09-2007	1002	127530	10530	0
4	3115	02-09-2007	1001	110907,5	9157,5	0
5	3116	10-09-2007	1007	136849,5	11299,5	0
6	3117	10-09-2007	1001	287855	33355	0
7	3118	17-09-2007	1001	179565	25065	0

Figura 18.5. Facturi arhivate care se păstrează (încă) și în tabela principală

Este un motiv cât se poate de serios pentru ca arhivarea pentru o lună să se facă după un timp suficient – 6 luni, 1 an.

sql2008

0,23726228 seconds

Enter SQL Statement:

```

ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD' ;

SELECT pac_arhivare.f_sir_facturi (DATE'2007-09-05', CURRENT_DATE) FROM dual ;

```

Results: Script Output Explain Autotrace DBMS Output OWA Output

Results:

	PAC_ARHIVARE.F_SIR_FACTURI(DATE'2007-09-05',CURRENT_DATE)
1	SELECT * FROM facturi_2007_9 WHERE NrFact NOT IN (SELECT NrFact FROM facturi) AND DataFact >= DATE'2007-09-05' UNION SELECT * FROM facturi_2007_10 WHERE NrFact NOT IN (SELECT NrFact FROM facturi) UNION SELECT * FROM facturi WHERE DataFact BETWEEN DATE'2007-09-05' AND DATE'2009-01-21'

Figura 18.6. Verificarea funcției *pac_arhivare.f_sir_facturi*

În continuare, testăm funcția *f_sir_facturi* pentru a vedea dacă SELECT-ul construit dinamic este corect. Intervalul care interesează este 5 sept. 2007 – data curentă. Figura 18.6 conține o parte din șir a cărui valoare este, de fapt: *SELECT * FROM facturi_2007_9 WHERE NrFact NOT IN (SELECT NrFact FROM facturi) AND DataFact >= DATE'2007-09-05' UNION SELECT * FROM facturi_2007_10 WHERE NrFact NOT IN (SELECT NrFact FROM facturi) UNION SELECT * FROM facturi WHERE DataFact BETWEEN DATE'2007-09-05' AND DATE'2009-01-21'*

WHERE DataFact BETWEEN DATE'2007-09-05' AND DATE'2009-01-21'. Apelul funcției este precedat de comanda prin care ne asigurăm că formatul datei în sesiunea curentă este YYYY-MM-DD.

În fine, procedura *p_fuziune_facturi1* crează, folosind șirul de returnat de funcția *f_sir_facturi* și SQL dinamic, o tabelă temporară – TEMP_FACTURI - al cărui conținut este privat (accesibil doar în și din sesiunea curentă). Deranjant este că, de câte ori dorim să afișăm facturile din intervalul specificat, trebuie să lansăm procedura (printr-un bloc anonim) și apoi să interogăm tabela temporară:

```
begin
  pac_arhivare.p_fuziune_facturi1 (DATE'2007-09-05', CURRENT_DATE) ;
end ;
```

SELECT * FROM temp_facturi ORDER BY 1;

Din fericire, soluția pare să funcționeze – vezi figura 18.7.

Enter SQL Statement:

```
begin
  pac_arhivare.p_fuziune_facturi1 (DATE'2007-09-05', CURRENT_DATE) ;
end ;

SELECT * FROM temp_facturi ORDER BY 1;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	NRFAC	DATAFACT	CODCL	OBS	VALTOTALA	TVA	VALINC
1	3116	2007-09-10	1007	Pretul propus initial a fost modificat	136849,5	11299,5	
2	3117	2007-09-10	1001	(null)	287855	33355	
3	3118	2007-09-17	1001	(null)	179565	25065	
4	3119	2007-10-07	1003	(null)	5819668	919468	
5	3500	2008-12-05	1002	(null)	33700	3700	
6	5111	2007-11-01	1001	(null)	4346037,5	687287,5	
7	5112	2007-11-01	1005	Probleme cu transportul	125516	13116	
8	5113	2007-11-01	1002	(null)	106275	8775	

Figura 18.7. Verificarea procedurii *pac_arhivare.p_fuziune_facturi1*

18.3. SQL dinamic în Transact-SQL

Față de Oracle PL/SQL, T-SQL este destul de îngăduitor cu multe comenzi DDL care pot fi incluse, fără prea mari probleme, în proceduri și funcții. Un argument în acest sens este hiper-simpla procedură *p_test* din listing 18.14 ce crează tabela TEST și care, la execuție, nu întâmpină deloc rezistență.

Listing 18.14. Procedură T-SQL prin care se crează o tabelă

```
CREATE PROCEDURE p_test
AS
```

```
BEGIN
  CREATE TABLE test (x INTEGER, y VARCHAR(50)) ;
END
```

Înscriindu-ne pe linia comparabilității cu cele două paragrafe precedente, redactăm, mai întâi funcția *f_există_tabela*, funcție altminteri ajutătoare întrucât nu conține elemente de SQL dinamic (listing 18.15). Reamintim că SQL Server este alergic la tipul BOOLEAN, nu numai în tabele, ci și în proceduri/funcții, tipul returnat de această funcție fiind BIT.

Listing 18.15. Funcție T-SQL prin care se verifică existența unei tabele

```
CREATE FUNCTION f_exista_tabela
  (@tabela_ VARCHAR(40), @an_ SMALLINT, @luna_ TINYINT)
  RETURNS BIT
BEGIN
  DECLARE @v_tabela VARCHAR(40)
  DECLARE @v BIT
  SET @v_tabela = RTRIM(UPPER(@tabela_)) + '_' + CAST (@an_ AS CHAR(4))
    + '_' + RTRIM(CAST (@luna_ AS CHAR(2)))
  SET @v = (SELECT 1 FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_SCHEMA = 'dbo' AND TABLE_TYPE = 'BASE TABLE'
    AND UPPER(TABLE_NAME) = @v_tabela )
  RETURN COALESCE(@v,0)
END
```

Ajungem, astfel, la prima procedură cu urme vizibile de SQL dinamic – *p_sterge_arhive_fact_luna* (listing 18.16). Ștergerea arhivelor este precedată de re-adăugarea înregistrărilor care, la ștergerea din arhive, s-ar pierde iremediabil, în tabelele „tranzacționale”. Pentru aceasta sunt necesare două operațiuni. Una dintre ele este comună PostgreSQL și Oracle, și anume dezactivarea declanșatoarelor, care se poate face direct în procedură prin comenzile DISABLE/ENABLE TRIGGER. Comanda T-SQL esențială în materie de SQL dinamic este EXECUTE cu o sintaxă ușor diferită de echivalentele sale din PL/pgSQL și PL/SQL.

Listing 18.16. Procedură T-SQL de ștergere a arhivelor lunare legate de facturi

```
CREATE PROCEDURE p_sterge_arhive_fact_luna
  (@an SMALLINT, @luna TINYINT)
AS
BEGIN
  DECLARE @v_sir VARCHAR(15)
  SET @v_sir = '_' + CAST (@an AS CHAR(4)) + '_' + RTRIM(CAST (@luna AS CHAR(2)))

  DECLARE @v_tabela VARCHAR(40)
  DECLARE c_tab CURSOR FOR SELECT table_name
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_SCHEMA = 'dbo' AND TABLE_TYPE = 'BASE TABLE'
    AND UPPER(TABLE_NAME) IN ( 'INCASFACT' + @v_sir, 'LINIIFACT' + @v_sir,
      'INCASARI' + @v_sir, 'FACTURI' + @v_sir ) ;

  -- pentru a nu pierde inregistrările arhivate le readaugăm la tabelele "curente"
  DISABLE TRIGGER dbo.trg_facturi_ins ON dbo.facturi ;
  DISABLE TRIGGER dbo.trg liniifact_ins ON dbo.liniifact ;
```

```

IF dbo.f_exista_tabela ('FACTURI', @an, @luna) = 1
    EXECUTE ('INSERT INTO facturi SELECT * FROM facturi' + @v_sir +
    ' WHERE NrFact NOT IN (SELECT NrFact FROM facturi)' );

IF dbo.f_exista_tabela ('INCASARI', @an, @luna) = 1
    EXECUTE ('SET IDENTITY_INSERT dbo.incasari ON ;' +
    'INSERT INTO incasari (CodInc, DataInc, CodDoc, NrDoc, DataDoc)' +
    ' SELECT * FROM incasari' + @v_sir + ' WHERE CodInc NOT IN ' +
    ' (SELECT CodInc FROM incasari) ; SET IDENTITY_INSERT dbo.incasari OFF ;' );

IF dbo.f_exista_tabela ('LINIIFACT', @an, @luna) = 1
    EXECUTE ('INSERT INTO liniifact SELECT * FROM liniifact' + @v_sir +
    ' WHERE CAST (NrFact AS CHAR(8)) + CAST (Linie AS CHAR(2)) NOT IN ' +
    ' (SELECT CAST (NrFact AS CHAR(8)) + CAST (Linie AS CHAR(2)) FROM liniifact )
    ORDER BY NrFact, Linie ' )

IF dbo.f_exista_tabela ('INCASFACT', @an, @luna) = 1
    EXECUTE ('INSERT INTO incasfact SELECT * FROM incasfact' + @v_sir +
    ' WHERE CAST (CodInc AS CHAR(10)) + CAST (NrFact AS CHAR(6)) NOT IN ' +
    ' (SELECT CAST (CodInc AS CHAR(10)) + CAST (NrFact AS CHAR(6)) FROM incasfact)' );

ENABLE TRIGGER dbo.trg_facturi_ins ON dbo.facturi ;
ENABLE TRIGGER dbo.trg liniifact_ins ON dbo.liniifact ;

OPEN c_tab
FETCH NEXT FROM c_tab INTO @v_tabela
WHILE @@FETCH_STATUS = 0 BEGIN
    EXECUTE ('DROP TABLE dbo.' + @v_tabela)
    FETCH NEXT FROM c_tab INTO @v_tabela
END
CLOSE c_tab
DEALLOCATE c_tab
END

```

A doua operațiune este nouă, întrucât în tabela INCASARI valorile atributului CodInc sunt generate automat, ceea ce are urmări serioase la reinserarea înregistrărilor din tabela arhivă. Pentru a evita acest gen de probleme, se folosește comanda *SET IDENTITY_INSERT dbo.incasari ON*. După posibila re-adăugare a înregistrărilor, se re-activează declanșatoarele și se trece la regimul de generare automată a valorilor CodInc din INCASARI.

Crearea tabelor arhivă pentru o lună dată este subiectul procedurii prezentate în listing 18.17. Ținând seama că, după crearea arhivelor, o parte importantă dintre înregistrările arhivate trebuie șterse din tabelele tranzacționale, este necesară dezactivarea declanșatoarelor de ștergere ale tablei LINIIFACT (cele ale tablei INCASFACT nu fost încă redactate).

Listing 18.17. Procedură T-SQL de creare a arhivelor lunare legate de facturi

```

CREATE PROCEDURE p_creare_arhive_fact_luna
    (@an SMALLINT, @luna TINYINT)
AS
BEGIN
    DECLARE @v_sir VARCHAR(15)
    SET @v_sir = '_' + CAST (@an AS CHAR(4)) + '_' + RTRIM(CAST (@luna AS CHAR(2)))

```

```

EXECUTE dbo.p_sterge_arhive_fact_luna @an, @luna

IF dbo.f_exista_tabela ('FACTURI', @an, @luna) = 1 BEGIN
    RAISERROR(N'Luna arhivata deja !', 18, 1)
    ROLLBACK TRANSACTION
END;

EXECUTE ('SELECT * INTO facturi' + @v_sir + ' FROM facturi ' +
' WHERE YEAR(DataFact)= ' + @an + ' AND MONTH(DataFact)= ' + @luna )
EXECUTE ('SELECT * INTO liniifact' + @v_sir +
' FROM liniifact WHERE NrFact IN (SELECT NrFact FROM facturi' + @v_sir + ') ' )
EXECUTE ('SELECT * INTO incasari' + @v_sir + ' FROM incasari WHERE ' +
' YEAR(DataInc)= ' + @an + ' AND MONTH(DataInc)= ' + @luna +
' UNION SELECT * FROM incasari WHERE CodInc IN ( ' +
' SELECT CodInc FROM incasfact WHERE NrFact IN ( ' +
' SELECT NrFact FROM facturi WHERE ' +
' YEAR(DataFact)= ' + @an + ' AND MONTH(DataFact)= ' + @luna + ' ) ) ' )
EXECUTE ('SELECT * INTO incasfact' + @v_sir + ' FROM incasfact ' +
' WHERE NrFact IN (SELECT NrFact FROM facturi' + @v_sir + ') ' );

DISABLE TRIGGER dbo.trg_liniifact_del ON dbo.liniifact ;

EXECUTE ('DELETE FROM incasfact WHERE CAST(CodInc AS CHAR(10)) + '
+ 'CAST (NrFact AS CHAR(6)) IN '
+ '(SELECT CAST(CodInc AS CHAR(10))+CAST(NrFact AS CHAR(6)) FROM incasfact'
+ @v_sir + ') ' )
EXECUTE ('DELETE FROM liniifact WHERE CAST (NrFact AS CHAR(6)) + '
+ 'CAST (Linie AS CHAR(2)) IN (SELECT CAST(NrFact AS CHAR(6)) + '
+ ' CAST (Linie AS CHAR(2)) FROM liniifact' + @v_sir + ') ' )
EXECUTE ('DELETE FROM facturi WHERE NrFact IN '
+ '(SELECT NrFact FROM facturi' + @v_sir +
' ) AND ValTotala = ValIncasata AND NrFact NOT IN '
+ '(SELECT NrFact FROM incasfact) ' )
EXECUTE ('DELETE FROM incasari WHERE CodInc IN (SELECT CodInc FROM ' +
' incasari' + @v_sir
+ ') AND CodInc NOT IN (SELECT CodInc FROM incasfact) ' );

ENABLE TRIGGER dbo.trg_liniifact_del ON dbo.liniifact ;
END

```

Similar paragrafului anterior, testăm funcționarea procedurii de arhivare prin execuția sa pentru lunile august, septembrie și octombrie 2007:

```
EXECUTE dbo.p_creare_arhive_fact_luna 2007, 8
```

```
EXECUTE dbo.p_creare_arhive_fact_luna 2007, 9
```

```
EXECUTE dbo.p_creare_arhive_fact_luna 2007, 10
```

După arhivarea celor trei luni se observă că în FACTURI au mai rămas facturi (1111, 1114, ...) care nu mai au înregistrări corespondente în LINIIFACT – vezi figura 18.8. Explicația este aceeași cu cea furnizată pentru Oracle: facturile „rămase” nu sunt încasate în întregime, deci o să apară, în luni următoare celor arhivate, în tabela INCASFACT, deci nu trebuie șterse din tabela părinte (FACTURI).

```
EXECUTE dbo.p_creare_arhive_fact_luna 2007, 9
EXECUTE dbo.p_creare_arhive_fact_luna 2007, 10

SELECT * FROM facturi ORDER BY 1
SELECT * FROM liniifact ORDER BY 1,2
```

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA	ValIncasata
1	1111	2007-08-01 ...	1001	NULL	4346037.50	687287.50	53996.00
2	1114	2007-08-01 ...	1006	NULL	6021706.50	950856.50	0.00
3	1115	2007-08-02 ...	1001	NULL	151237.50	12487.50	0.00
4	1116	2007-08-02 ...	1007	Pretul pr...	126712.50	10462.50	0.00
5	1117	2007-08-03 ...	1001	NULL	222050.00	27050.00	23204.00
6	1119	2007-08-07 ...	1003	NULL	5774498.50	912848.50	0.00
7	1120	2007-08-07 ...	1001	NULL	97664.00	8064.00	7315.00

	NrFact	Linie	CodPr	Canitate	PretUnit	TVAInlie
1	3500	1	2	200	100.00	1800.00
2	3500	2	3	100	100.00	1900.00
3	5111	1	1	50	1000.00	9500.00
4	5111	2	2	75	1050.00	7087.50
5	5111	3	5	500	7060.00	67070.00

Figura 18.8. Conținutul (parțial) al tabelor FACTURI și LINIIFACT după arhivare

Întrucât tablele INCASARI și INCASFACT conțin doar câteva operațiuni din august 2007, acum conținutul acestora este gol – vezi figura 18.9, toate înregistrările acestora fiind „mutate” în INCASARI_2007_8 și INCASFACT_2007_8.

```
SELECT * FROM incasari ORDER BY 1
SELECT * FROM incasfact ORDER BY 1,2
SELECT * FROM incasari_2007_8 ORDER BY 1
SELECT * FROM incasfact_2007_8 ORDER BY 1,2
```

	CodInc	DataInc	CodDoc	NrD...	DataDoc
1	1234	2007-08-15 ...	OP	111	2007-08-10 ...
2	1235	2007-08-15 ...	CHIT	222	2007-08-15 ...
3	1236	2007-08-16 ...	OP	333	2007-08-09 ...
4	1237	2007-08-17 ...	CEC	444	2007-08-10 ...
5	1238	2007-08-17 ...	OP	555	2007-08-10 ...
6	1239	2007-08-18 ...	OP	666	2007-08-11 ...

	CodInc	NrFact	Transa
1	1234	1111	53996.00
2	1234	1118	101975.00
3	1235	1112	125516.00
4	1236	1117	9754.00
5	1236	1118	100000.00
6	1236	1120	7315.00
7	1237	1117	9754.00
8	1238	1113	106275.00
9	1239	1117	3696.00

Figura 18.9. Conținutul tabelor INCASARI și INCASFACT, precum și ale arhivelor lor pe luna aug. 2007

Funcția ce construiește, prin eventuale UNION-uri, întreaga interogare prin care, din arhive și tabela FACTURI, sunt extrase toate facturile dintr-un interval calendaristic specificat este afișată în listing 18.18. Cum deja ne-am obișnuit, în T-SQL nu există structuri repetitive de tipul *FOR i = 1 TO n...*, așa că se folosește WHILE. O altă problemă a fost conversia constantelor de tip dată calendaristică în șiruri de caractere, operațiune pentru care am folosit funcția CONVERT.

Listing 18.18. Funcție ce returnează interogarea SQL pentru reconstituirea facturilor

```
CREATE FUNCTION f_sir_facturi
  (@data_inceput SMALLDATETIME, @data_final SMALLDATETIME)
RETURNS VARCHAR(5000)
AS
BEGIN
  DECLARE @v_luna TINYINT
  DECLARE @v_luna_inceput TINYINT
  DECLARE @v_luna_final TINYINT
  DECLARE @v_clauza_where VARCHAR(500)
  DECLARE @v_sir VARCHAR(5000)
  DECLARE @v_numa VARCHAR(50)
  SET @v_sir = ''
  DECLARE @v_an SMALLINT
  SET @v_an = YEAR(@data_inceput)
  SET @v_clauza_where = ''

  WHILE @v_an <= YEAR(@data_final) BEGIN

    IF @v_an = YEAR(@data_inceput)
      SET @v_luna_inceput = MONTH(@data_inceput)
    ELSE
      SET @v_luna_inceput = 1

    IF @v_an = YEAR(@data_final)
      SET @v_luna_final = MONTH(@data_final)
    ELSE
      SET @v_luna_final = 12

    SET @v_luna = @v_luna_inceput

    WHILE @v_luna <= @v_luna_final BEGIN
      SET @v_numa = 'facturi_' + CAST( @v_an AS CHAR(4)) + '_' +
        RTRIM(CAST (@v_luna AS CHAR(2)))

      IF dbo.f_exista_tabela ('facturi', @v_an, @v_luna ) = 1 BEGIN
        SET @v_clauza_where =
          ' WHERE NrFact NOT IN (SELECT NrFact FROM facturi ) '

        IF @v_an = YEAR(@data_inceput) AND @v_luna = MONTH(@data_inceput)
          SET @v_clauza_where = @v_clauza_where + ' AND DataFact >= ''' +
            CONVERT (CHAR(10), @data_inceput, 102) + ''''

        IF @v_an = YEAR(@data_final) AND @v_luna=MONTH(@data_final)
          SET @v_clauza_where = @v_clauza_where + ' AND DataFact <= ''' +
            CONVERT (CHAR(10), @data_final, 102) + ''''

        SET @v_sir = CASE WHEN @v_sir = '' THEN @v_sir ELSE @v_sir +
          ' UNION ' END + 'SELECT * FROM ' + @v_numa + @v_clauza_where
      END
      SET @v_luna = @v_luna + 1
    END
    SET @v_an = @v_an + 1
  END

  SET @v_sir = @v_sir + ' '
  RETURN @v_sir
END
```

```

        END
        SET @v_luna = @v_luna + 1
    END

    SET @v_an = @v_an + 1
    END

    SET @v_sir = CASE WHEN @v_sir = '' THEN @v_sir ELSE @v_sir + ' UNION ' END
    + ' SELECT * FROM facturi WHERE DataFact BETWEEN "' +
        CONVERT (CHAR(10), @data_inceput, 102) + '" + ' AND "' +
        CONVERT (CHAR(10), @data_final, 102) + '" '

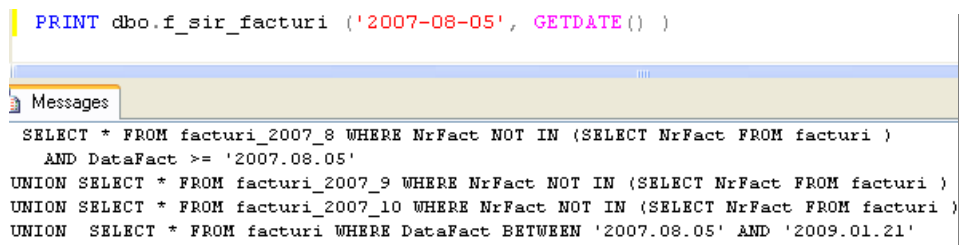
    RETURN @v_sir
END

```

Testarea funcției poate fi făcută în SQL Server și cu PRINT (am avut, de fapt, nevoie de o opțiune pentru afișarea unui șir lung de caractere pe mai multe linii):

```
PRINT dbo.f_sir_facturi ('2007-08-05', GETDATE())
```

șirul fiind conform cu așteptările (vezi figura 18.10).



```

PRINT dbo.f_sir_facturi ('2007-08-05', GETDATE() )

SELECT * FROM facturi_2007_8 WHERE NrFact NOT IN (SELECT NrFact FROM facturi )
AND DataFact >= '2007.08.05'
UNION SELECT * FROM facturi_2007_9 WHERE NrFact NOT IN (SELECT NrFact FROM facturi )
UNION SELECT * FROM facturi_2007_10 WHERE NrFact NOT IN (SELECT NrFact FROM facturi )
UNION SELECT * FROM facturi WHERE DataFact BETWEEN '2007.08.05' AND '2009.01.21'

```

Figura 18.10. Conținutul tabelor INCASARI și INCASFACT, precum și ale arhivelor lor pe luna aug. 2007

Procedura care crează tabela temporară globală este una destul de simplă – vezi listing 18.19. Numele tabelii temporare este specificat la invocare de către utilizator (vezi figura 18.11):

```
EXECUTE dbo.p_fuziune_facturi '2007-09-05', '2007-11-10', 'fact_temp2'
```

```
SELECT * FROM dbo.##fact_temp2 ORDER BY 1
```

Listing 18.19. Procedura T-SQL care crează o tabelă temporară ce reconstituie facturile

```

CREATE PROCEDURE p_fuziune_facturi
    (@datai SMALLDATETIME, @dataf SMALLDATETIME, @nume_tabela VARCHAR(30))
AS
BEGIN
    DECLARE @v_sir VARCHAR(5000)
    SET @v_sir = dbo.f_sir_facturi (@datai, @dataf)
    EXECUTE ('SELECT * INTO ##' + @nume_tabela + ' FROM (' + @v_sir + ') t')
END

```

```
EXECUTE dbo.p_fuziune_facturi '2007-09-05', '2007-11-10', 'fact_temp2'
SELECT * FROM dbo.##fact_temp2 ORDER BY 1
```

	NrFact	DataFact	CodCl	Obs	ValTotala	TVA
1	3116	2007-09-10 00:00:00	1007	Pretul propus initial a fost modificat	136849.50	11299.50
2	3117	2007-09-10 00:00:00	1001	NULL	287855.00	33355.00
3	3118	2007-09-17 00:00:00	1001	NULL	179565.00	25065.00
4	3119	2007-10-07 00:00:00	1003	NULL	5819668.00	919468.00
5	3120	2007-10-01 00:00:00	1001	NULL	571200.00	91200.00
6	5111	2007-11-01 00:00:00	1001	NULL	4346037.50	687287.50
7	5112	2007-11-01 00:00:00	1005	Probleme cu transportul	125516.00	13116.00
8	5113	2007-11-01 00:00:00	1002	NULL	106275.00	8775.00
9	5114	2007-11-01 00:00:00	1006	NULL	6021706.50	950856.50
10	5115	2007-11-02 00:00:00	1001	NULL	151237.50	12487.50
11	5116	2007-11-02 00:00:00	1007	Pretul propus initial a fost modificat	126712.50	10462.50
12	5117	2007-11-03 00:00:00	1001	NULL	222050.00	27050.00
13	5118	2007-11-04 00:00:00	1001	NULL	201975.00	29475.00
14	5119	2007-11-07 00:00:00	1003	NULL	5774498.50	912848.50
15	5120	2007-11-07 00:00:00	1001	NULL	97664.00	8064.00
16	5121	2007-11-07 00:00:00	1004	NULL	4737838.00	747638.00

Figura 18.11. Crearea și apelul tabelii temporare ##TEMP_FACT

18.4. SQL dinamic în IBM DB2 SQL PL

Oracle PL/SQL rămâne singurul limbaj dintre cele patru discutate în această lucrare ce nu permite crearea unei tabeli prin program, întrucât procedura simplă redactată în IBM DB2 SQL PL din listing 18.20 nu întâmpină rezistență la prima sa lansare în execuție.

Listing 18.20. O (banală) procedură SQL PL prin care se crează o tabelă

```
CREATE PROCEDURE p_test ( )
P1: BEGIN
    CREATE TABLE test (x INTEGER, y VARCHAR(50)) ;
END P1
```

Nici în SQL PL funcția *f_există_tabela* nu are legătură cu SQL dinamic (listing 18.21). Similar SQL Server, SQL PL este destul de irascibil la tipul BOOLEAN, chiar și în proceduri și funcții (în acest caz, DB2 nu acceptă clauza *RETURNS BOOLEAN*). Așa că tipul returnat de funcție este SMALLINT.

Listing 18.21. Funcție SQL PL prin care se verifică existența unei tabeli

```
CREATE FUNCTION f_exista_tabela( tabela_ VARCHAR(40), an_ SMALLINT, luna_ SMALLINT )
RETURNS SMALLINT
F1: BEGIN ATOMIC
    DECLARE v_tabela VARCHAR(40) ;
    DECLARE v SMALLINT ;
```



```

SET v_tabela = RTRIM(UPPER(tabela_)) || '_' || CAST (an_ AS CHAR(4))
|| '_' || RTRIM(CAST (luna_ AS CHAR(2)));

SET v = (SELECT 1 FROM SYSCAT.tables
WHERE tabschema=CURRENT_USER AND type='T' AND UPPER(tabname) = v_tabela ) ;
RETURN COALESCE(v,0) ;
END

```

Noutatea majoră căreia trebuie să-i găsim o soluție în DB2 este faptul că declanșatoarele nu pot fi dezactivate și reactivate. În aceste condiții, orice inserare în LINIIFACT incrementează ValTotala și TVA în FACTURI, iar orice inserare în INCASFACT incrementează ValIncasata din FACTURI. La fel (dar cu semn schimbat) stau lucrurile și la ștergeri de înregistrări din LINIIFACT și INCASFACT. Renunțăm la procedura de ștergere a arhivelor și mutăm o parte din operațiunile sale în procedura *p_creare_arhive_fact_luna* – vezi listing 18.22. Prima porțiune a procedurii verifică dacă există arhivă pe anul și luna date. Dacă da, atunci se șterg din arhivă numai înregistrările care sunt în tabela principală corespondentă. Dacă nu există arhiva, se crează folosind sintaxa *CREATE TABLE tabela_an_luna LIKE tabela*. După verificarea existenței arhivei, și crearea sa, sau ștergerea unora dintre înregistrările sale, în arhivă se inserează înregistrările din tabela principală ce corespund anului și lunii curente.

Listing 18.22. Variata SQL PL a procedurii de arhivare lunară a facturilor/încasărilor

```

CREATE PROCEDURE p_creare_arhive_fact_luna
(an_ SMALLINT, luna_ SMALLINT)
P1: BEGIN
DECLARE v_exec VARCHAR(2000) ;
DECLARE v_sir VARCHAR(15) ;
SET v_sir = '_' || CAST (an_ AS CHAR(4)) || '_' || RTRIM(CAST (luna_ AS CHAR(2)));

--- creare arhive sau ștergerea unora dintre înregistrările acestora (ce exista în tabelele principale)
IF f_exista_tabela ('FACTURI', an_, luna_) = 1 THEN
SET v_exec = 'DELETE FROM facturi' || v_sir ||
' WHERE NrFact IN (SELECT NrFact FROM facturi)' ;
EXECUTE IMMEDIATE v_exec ;
ELSE
SET v_exec = 'CREATE TABLE facturi' || v_sir || ' LIKE facturi ' ;
EXECUTE IMMEDIATE v_exec ;
END IF ;
SET v_exec = 'INSERT INTO facturi' || v_sir || ' SELECT * FROM facturi ' ||
' WHERE YEAR(DataFact)= ' || CAST (an_ AS CHAR(4)) ||
' AND MONTH(DataFact)= ' || CAST (luna_ AS CHAR(2)) ;
EXECUTE IMMEDIATE v_exec ;

IF f_exista_tabela ('LINIIFACT', an_, luna_) = 1 THEN
SET v_exec = 'DELETE FROM liniifact' || v_sir ||
' WHERE (NrFact, Linie) IN (SELECT NrFact, Linie FROM liniifact)' ;
EXECUTE IMMEDIATE v_exec ;
ELSE
SET v_exec = 'CREATE TABLE liniifact' || v_sir || ' LIKE liniifact ' ;
EXECUTE IMMEDIATE v_exec ;
END IF ;
SET v_exec = 'INSERT INTO liniifact' || v_sir ||

```

```

' SELECT * FROM liniifact WHERE NrFact IN (SELECT NrFact FROM facturi' || v_sir || ' ');
EXECUTE IMMEDIATE v_exec ;

IF f_exista_tabela ('INCASARI', an_, luna_) = 1 THEN
    SET v_exec = 'DELETE FROM incasari' || v_sir ||
        ' WHERE CodInc IN (SELECT CodInc FROM incasari) ' ;
    EXECUTE IMMEDIATE v_exec ;
ELSE
    SET v_exec = 'CREATE TABLE incasari' || v_sir || ' LIKE incasari ' ;
    EXECUTE IMMEDIATE v_exec ;
END IF ;
SET v_exec = 'INSERT INTO incasari' || v_sir || ' SELECT * FROM incasari WHERE ' ||
    ' YEAR(DataInc)= ' || CAST (an_ AS CHAR(4)) ||
    ' AND MONTH(DataInc)= ' || CAST (luna_ AS CHAR(2)) ||
' UNION SELECT * FROM incasari WHERE CodInc IN ( ' ||
    ' SELECT CodInc FROM incasfact WHERE NrFact IN ( ' ||
    ' SELECT NrFact FROM facturi WHERE ' ||
    ' YEAR(DataFact)= ' || CAST (an_ AS CHAR(4)) || ' AND MONTH(DataFact)= ' ||
    CAST (luna_ AS CHAR(2)) || ' ) ) ' ;
EXECUTE IMMEDIATE v_exec ;

IF f_exista_tabela ('INCASFACT', an_, luna_) = 1 THEN
    SET v_exec = 'DELETE FROM incasfact' || v_sir ||
        ' WHERE (CodInc, NrFact) IN (SELECT CodInc, NrFact FROM incasfact) ' ;
    EXECUTE IMMEDIATE v_exec ;
ELSE
    SET v_exec = 'CREATE TABLE incasfact' || v_sir || ' LIKE incasfact ' ;
    EXECUTE IMMEDIATE v_exec ;
END IF ;
SET v_exec = 'INSERT INTO incasfact' || v_sir || ' SELECT * FROM incasfact ' ||
    ' WHERE NrFact IN (SELECT NrFact FROM facturi' || v_sir ||
        ' ) AND CodInc IN (SELECT CodInc FROM incasari' || v_sir || ' ) ' ;
EXECUTE IMMEDIATE v_exec ;

--- urmeaza arhivarea propriu-zisa; din tabele principale se sterg numai datele
--- facturilor incasate complet

SET v_exec = 'DELETE FROM incasfact WHERE (CodInc, NrFact) IN '
    || '(SELECT CodInc, NrFact FROM incasfact' || v_sir ||
        ' ) AND NrFact IN (SELECT NrFact FROM facturi WHERE ValTotala = ValIncasata ) ' ;
EXECUTE IMMEDIATE v_exec ;

SET v_exec = 'DELETE FROM liniifact WHERE (NrFact, Linie) IN ( ' ||
    ' SELECT NrFact, Linie FROM liniifact' || v_sir || ' ) AND NrFact IN ( ' ||
    ' SELECT NrFact FROM facturi WHERE ValTotala = ValIncasata ) ' ;
EXECUTE IMMEDIATE v_exec ;

SET v_exec = 'DELETE FROM facturi WHERE NrFact IN '
    || '(SELECT NrFact FROM facturi' || v_sir ||
        ' ) AND ValTotala = ValIncasata ' ;
EXECUTE IMMEDIATE v_exec ;

SET v_exec = 'DELETE FROM incasari WHERE CodInc IN (SELECT CodInc FROM ' ||
    ' incasari' || v_sir
    || ' ) AND CodInc NOT IN (SELECT CodInc FROM incasfact) ' ;
EXECUTE IMMEDIATE v_exec ;

```

END P1

După arhivare se șterg numai înregistrări care se referă la facturi încasate în totalitate. Se crează arhivele pe lunile august, septembrie și octombrie 2007:

CALL p_creare_arhive_fact_luna (2007,8)

CALL p_creare_arhive_fact_luna (2007,9)

CALL p_creare_arhive_fact_luna (2007,10)

Listing-ul 18.22 conține funcția SQL PL pentru recompunerea facturilor din tabelele curente și din arhive pentru un interval dat.

Listing 18.23. Variantă SQL PL a funcției de reconstituire a facturilor arhivate

```
CREATE FUNCTION f_sir_facturi (data_inceput DATE, data_final DATE)
RETURNS VARCHAR(5000)
NO EXTERNAL ACTION
F1: BEGIN ATOMIC
DECLARE v_luna SMALLINT ;
DECLARE v_luna_inceput SMALLINT ;
DECLARE v_luna_final SMALLINT ;
DECLARE v_clauza_where VARCHAR(500) ;
DECLARE v_sir VARCHAR(5000) ;
DECLARE v_ume VARCHAR(50) ;
DECLARE v_an SMALLINT ;
SET v_sir = '' ;
SET v_an = YEAR(data_inceput) ;
SET v_clauza_where = '' ;

WHILE (v_an <= YEAR(data_final)) DO

    IF v_an = YEAR(data_inceput) THEN
        SET v_luna_inceput = MONTH(data_inceput) ;
    ELSE
        SET v_luna_inceput = 1 ;
    END IF ;

    IF v_an = YEAR(data_final) THEN
        SET v_luna_final = MONTH(data_final) ;
    ELSE
        SET v_luna_final = 12 ;
    END IF ;

    SET v_luna = v_luna_inceput ;
    WHILE (v_luna <= v_luna_final) DO

        SET v_ume = 'facturi_' || CAST( v_an AS CHAR(4)) || '_' ||
            RTRIM(CAST (v_luna AS CHAR(2))) ;
        IF f_exista_tabela ('facturi', v_an, v_luna) = 1 THEN
            SET v_clauza_where = ' WHERE NrFact NOT IN (SELECT NrFact FROM facturi) ' ;
            IF v_an = YEAR(data_inceput) AND v_luna = MONTH(data_inceput) THEN
                SET v_clauza_where = v_clauza_where || ' AND DataFact >= ' ||
                    CAST (data_inceput AS CHAR(10)) || ' ' ;
            END IF ;
            IF v_an = YEAR(data_final) AND v_luna = MONTH(data_final) THEN
                SET v_clauza_where = v_clauza_where || ' AND DataFact <= ' ||
                    CAST (data_final AS CHAR(10)) || ' ' ;
            END IF ;
        END IF ;
    END WHILE ;
END WHILE ;
```

```

        END IF ;
        SET v_sir = CASE WHEN v_sir = '' THEN v_sir ELSE v_sir || ' UNION ' END
        || 'SELECT * FROM ' || v_nume || v_clauza_where ;
    END IF ;
    SET v_luna = v_luna + 1 ;
END WHILE ;
SET v_an = v_an + 1 ;
END WHILE ;

SET v_sir = CASE WHEN v_sir = '' THEN v_sir ELSE v_sir || ' UNION ' END
|| 'SELECT * FROM facturi WHERE DataFact BETWEEN "' ||
    CAST (data_inceput AS CHAR(10)) || "' AND '" ||
    CAST (data_final AS CHAR(10)) || "' ;

RETURN v_sir ;

END

```

În fine, ca și în T-SQL, procedura din listing 18.24 crează o tabelă temporară globală cu un nume indicat la apelare, tabelă temporară ce conține facturile din intervalul calendaristic specificat prin *data_initiala* și *data_finala*.

Listing 18.24. Procedură SQL PL ce crează o tabelă temporară globală

```

CREATE PROCEDURE p_fuziune_facturi1 (data_initiala DATE,
    data_finala DATE, tab_temporara VARCHAR(30) )
P1: BEGIN
    DECLARE v_sir VARCHAR(5000) ;

    SET v_sir = 'DECLARE GLOBAL TEMPORARY TABLE session.' || tab_temporara ||
        ' LIKE facturi ON COMMIT PRESERVE ROWS ' ;
    EXECUTE IMMEDIATE v_sir ;

    SET v_sir = ' INSERT INTO session.' || tab_temporara || ' '
        || f_sir_facturi (data_initiala, data_finala) ;
    EXECUTE IMMEDIATE v_sir ;

END P1

```

Tot similar T-SQL, apelăm procedura, numele indicat al tabelii temporare fiind TAB_TEMP3, și se obțin înregistrările din figura 18. 12.

```

CALL p_fuziune_facturi1 (CAST ('2007-09-05' AS DATE),
    CAST ('2007-11-10' AS DATE), 'tab_temp3')

SELECT * FROM session.tab_temp3 ORDER BY 1

```

```
CALL p_fuziune_facturil (CAST ('2007-09-05' AS DATE), CAST ('2007-11-10' AS DATE), 'tab_temp3')
```

```
SELECT * FROM session.tab_temp3 ORDER BY 1
```

```
----- Commands Entered -----
SELECT * FROM session.tab_temp3 ORDER BY 1;
-----
SELECT * FROM session.tab_temp3 ORDER BY 1
```

NRFACT	DATAFACT	CODCL	OBS	VALTOTALA	TVA	VAL
3116	09/10/2007	1007	Pretul propus initial a fost modificat	136849.50	11299.50	
3117	09/10/2007	1001	-	287855.00	33355.00	
3118	09/17/2007	1001	-	179565.00	25065.00	
3119	10/07/2007	1003	-	5819668.00	919468.00	
5111	11/01/2007	1001	-	4346037.50	687287.50	
5112	11/01/2007	1005	Probleme cu transportul	125516.00	13116.00	
5113	11/01/2007	1002	-	106275.00	8775.00	
5114	11/01/2007	1006	-	6021706.50	950856.50	
5115	11/02/2007	1001	-	151237.50	12487.50	
5116	11/02/2007	1007	Pretul propus initial a fost modificat	126712.50	10462.50	
5117	11/03/2007	1001	-	222050.00	27050.00	
5118	11/04/2007	1001	-	201975.00	29475.00	
5119	11/07/2007	1003	-	5774498.50	912848.50	
5120	11/07/2007	1001	-	97664.00	8064.00	
5121	11/07/2007	1004	-	4737838.00	747638.00	
5122	11/07/2007	1005	-	-	-	

```
16 record(s) selected.
```

Figura 18.12. Crearea și apelul tabelii temporare TAB_TEMP3

Încheiem paragraful cu datoria din capitolul anterior legată de furnizarea unui mecanism minimal de protejare a atributelor calculate de modificări interactive. Spuneam atunci că există în DB2 variabile globale, însă acestea nu pot fi modificate din declanșatoare, așa că suntem nevoiți să găsim o altă soluție. Soluția pe care o discutăm acum se bazează pe o tabelă temporară (care se crează automat în schema SESSION) prin comanda DECLARE GLOBAL TEMPORARY TABLE. Tabela temporară își poate păstra schema și conținutul doar pe perioada unei sesiuni de lucru cu baza de date, conținutul său fiind privat. Necazul este că orice procedură, funcție și declanșator ce face referire la tabela temporară o să aibă necazuri mari la compilare, întrucât este vorba de un obiect nepersistent în schema bazei.

Tocmai aici își arată o parte dintre virtuți SQL-ul dinamic. Redactăm, pentru început o procedură (vezi listing 18.25) care verifică existența tabelii temporare în schema SESSION. Dacă nu există, o crează și îi inserează o linie. Tabela se numește V_TRG_LINIIFACT și conține un atribut X care va avea două valori, 0 și 1.

Listing 18.25. Procedură SQL PL ce crează o tabelă temporară globală

```
CREATE PROCEDURE p_creare_v_trg liniifact ( )
P1: BEGIN
    DECLARE v_sir VARCHAR(500) ;
    DECLARE v SMALLINT ;
    DECLARE v2 SMALLINT ;
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE SQLCODE INT DEFAULT 0;

    DECLARE c_notfound CONDITION FOR SQLSTATE '42704';
    DECLARE CONTINUE HANDLER FOR c_notfound SET v2 = 1 ;
```

```

SET v_sir = 'UPDATE SESSION.V_TRG_LINIIFACT SET x=x ' ;
EXECUTE IMMEDIATE v_sir ;
IF v2 = 1 THEN
  SET v_sir = 'DECLARE GLOBAL TEMPORARY TABLE v_trg_liniifact (x SMALLINT ) ON COMMIT
PRESERVE ROWS ' ;
  EXECUTE IMMEDIATE v_sir ;
  SET v_sir = 'INSERT INTO session.v_trg_liniifact VALUES (0)' ;
  EXECUTE IMMEDIATE v_sir ;
END IF;
END P1

```

O a doua procedură, cea din listing 18.26, modifică valoarea atributului X din tabela temporară. Pentru a evita eroarea ce s-ar declanșa la apelul acestei proceduri fără crearea prealabilă a tabelului, comanda UPDATE este precedată de un apel la procedura anterioară.

Listing 18.26. Procedură SQL PL ce setează valoarea atributului X în tabela temporară

```

CREATE PROCEDURE p_setare_v_trg_liniifact ( valoare_ SMALLINT )
P1: BEGIN
  DECLARE v_sir VARCHAR(500) ;
  CALL p_creat_v_trg_liniifact ;
  SET v_sir = 'UPDATE session.v_trg_liniifact SET x = ' || CAST (valoare_ AS CHAR(1)) ;
  EXECUTE IMMEDIATE v_sir ;
END P1

```

Ceva mai mult de lucru am avut cu o funcție care să returneze valoarea curentă a atributului X din singura linie a tabelului temporar. Cum n-am fost în stare să o redactez, am apelat la o procedură altminteri simplă (vezi listing 18.27) în care utilizăm un parametru de tip OUT (*v_*). Procedura folosește un cursor dinamic care încarcă singura înregistrare din tabela temporară în parametrul de ieșire.

Listing 18.27. Procedură SQL PL ce preia valoarea atributului X din tabela temporară

```

CREATE PROCEDURE p_get_v_trg_liniifact ( OUT v_ SMALLINT )
P1: BEGIN
  DECLARE v_sir VARCHAR(200);
  DECLARE c_x CURSOR FOR v_cur_stmt;
  SET v_sir = 'SELECT x FROM session.v_trg_liniifact ' ;
  PREPARE v_cur_stmt FROM v_sir ;
  OPEN c_x;
  FETCH c_x INTO v_ ;
  CLOSE c_x;
END P1

```

Nu ne mai rămâne decât să modificăm declanșatoarele tabelilor LINIIFACT și FACTURI. În DB2 modificarea declanșatoarelor nu este posibilă, așa că mai întâi le ștergem și apoi compilăm noile versiuni.

DROP TRIGGER trg_liniifacturi_ins2

Declanșatorul de tip BEFORE INSERT (trg_linii_fact_ins1) rămâne neschimbat. Cel de-al doilea declanșator – vezi listing 18.28 – apelează înainte și după actualizarea atributelor calculate din tabela FACTURI procedura prin care se setează

valoarea atributului X în tabela temporară. Setarea se face pe 1 înainte de UPDATE și pe 0 după UPDATE. Scopul este clar: valoarea atributului X din V_TRG_LINII-FACT va fi 1 doar pe perioada actualizării celor două atribute ale tabelii FACTURI.

Listing 18.28. Noua variantă a celui de-al doilea declanșator de inserare în LINIIFACT

```
-- NOUL declansatorul la nivel de linie de tipul AFTER
CREATE TRIGGER trg_liniiufacturi_ins2
AFTER INSERT ON liniiifact
REFERENCING NEW AS NEW FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    -- declansatorul TRG_LINIIFACT_INS1 ramine neschimbat

    CALL p_setare_v_trg_liniiifact (1) ;

    UPDATE facturi
    SET tva = COALESCE(tva,0) + NEW.tvalinie,
        valtotala = COALESCE(valtotala,0) + NEW.cantitate * NEW.pretunit + NEW.tvalinie
    WHERE NrFact = NEW.NrFact ;

    CALL p_setare_v_trg_liniiifact (0) ;
END
```

Și declanșatorul de modificare a unei înregistrări din FACTURI trebuie actualizat. Mai întâi scăpăm de versiunea curentă:

DROP TRIGGER trg_facturi_upd_a_r_nrfect

și apoi îl compilăm pe cel din listing 18.29 care are un nume ușor schimbat.

Listing 18.29. Noul declanșator de modificare pentru FACTURI

```
CREATE TRIGGER trg_facturi_upd_a_r
AFTER UPDATE OF NrFact, ValTotala, TVA, ValIncasata ON facturi
REFERENCING OLD AS OLD NEW AS NEW FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    DECLARE v_vp SMALLINT DEFAULT 0 ;
    IF OLD.NrFact <> NEW.NrFact THEN
        UPDATE liniiifact SET NrFact = NEW.NrFact WHERE NrFact = OLD.NrFact ;
        UPDATE incasfact SET NrFact = NEW.NrFact WHERE NrFact = OLD.NrFact ;
    END IF ;

    IF OLD.tva <> NEW.tva OR OLD.ValTotala <> NEW.ValTotala THEN
        CALL p_get_v_trg_liniiifact (v_vp) ;
        IF COALESCE(v_vp,0) <> 1 THEN
            SIGNAL SQLSTATE '80000'
            SET MESSAGE_TEXT='Nu puteti modifica interactiv valorilor TVA si ValTotala !' ;
        END IF ;
    END IF ;
END
```

Testarea o facem inserând două linii într-o factură nouă, cu numărul 4002:

**INSERT INTO facturi VALUES (4002, CURRENT_DATE, 1002, NULL,
123456, 234567, 345678, NULL, NULL)**

```
INSERT INTO liniifact VALUES (4002, 1, 2, 200, 100, 8989989) ;
```

```
INSERT INTO liniifact VALUES (4002, 2, 3, 100, 100, 0) ;
```

după care încercăm să modificăm direct valoarea atributului ValTotală din FAC-
TURI în linia corespunzătoare facturii 4002:

```
UPDATE facturi SET ValTotala = 1234567 WHERE NrFact = 4002
```

Declanșatoarele funcționează corect, după cum ne confirmă figura 18.13.

```
INSERT INTO facturi VALUES (4002, CURRENT_DATE, 1002, NULL, 123456, 234567, 345678, NULL, NULL)
INSERT INTO liniifact VALUES (4002, 1, 2, 200, 100, 8989989) ;
INSERT INTO liniifact VALUES (4002, 2, 3, 100, 100, 0) ;

UPDATE facturi SET ValTotala = 1234567 WHERE NrFact = 4002

----- Commands Entered -----
INSERT INTO liniifact VALUES (4002, 1, 2, 200, 100, 8989989) ;
-----
INSERT INTO liniifact VALUES (4002, 1, 2, 200, 100, 8989989)
DB20000I The SQL command completed successfully.

----- Commands Entered -----
INSERT INTO liniifact VALUES (4002, 2, 3, 100, 100, 0) ;
-----
INSERT INTO liniifact VALUES (4002, 2, 3, 100, 100, 0)
DB20000I The SQL command completed successfully.

----- Commands Entered -----
UPDATE facturi SET ValTotala = 1234567 WHERE NrFact = 4002;
-----
UPDATE facturi SET ValTotala = 1234567 WHERE NrFact = 4002
DB21034E The command was processed as an SQL statement because it was not a
valid Command Line Processor command. During SQL processing it returned:
SQL0438N Application raised error with diagnostic text: "Nu puteti modifica
interactiv valorilor TVA si ValTotala !". SQLSTATE=80000

SQL0438N Application raised error with diagnostic text: "Nu puteti modifica interactiv valorilor TVA si ValTotala !"
```

Figura 18.13. Verificarea procedurilor și declanșatoarelor ce protejează attributele calculate