

**Universitatea din Bucureşti
Facultatea de Matematică și Informatică**

***Aplicații profesionale în baze de date
orientate obiect***

**Master „Baze de date și tehnologii web”
Anul I, Semestrul I**

Cuprins

1. INTRODUCERE	4
2. ORACLE9I FORMS DEVELOPER	6
2.1. INTRODUCERE	6
2.2. ARHITECTURA SERVICIILOR FORMS	7
2.3. PUBLICAREA FORMELOR	8
2.4. PROIECTAREA FORMELOR	10
2.4.1. <i>Diagrama conceptuală exemplu</i>	10
2.4.2. <i>Componentele Forms Builder</i>	11
Object Navigator	11
Paleta de proprietăți (Property Palette).....	13
Editorul vizual (Layout Editor)	15
Editorul PL/SQL.....	16
2.4.3. <i>Instrumente de tip wizard</i>	17
2.4.4. <i>Testarea formelor</i>	27
Modul insert	31
Modul query	35
2.4.5. <i>Rafinarea formelor</i>	40
2.4.6. <i>Palete de proprietăți</i>	43
2.4.7. <i>Atribute vizuale</i>	47
2.4.8. <i>Proprietățile ferestrelor</i>	48
2.4.9. <i>Proprietățile unui bloc de date</i>	49
2.5. BLOCURI MASTER-DETAIL.....	58
2.6. GESTIONAREA ELEMENTELOR TEXT	71
2.7. BUTOANE (PUSH BUTTONS).....	83
2.8. CANVAS-URI DE TIP SUPRAPUS (STACKED CANVAS).....	87
2.9. LISTE DE VALORI (LOVS).....	92
2.10. ALTE TIPURI DE ELEMENTE INTR-UN FORMULAR	99
2.10.1. <i>Casete de validare</i>	99
2.10.2. <i>Elemente lista</i>	100
2.10.3. <i>Grupuri de butoane radio</i>	101
2.11. DECLANȘATORI (TRIGGER-Î).....	103
2.11.1. <i>Trigger-i de interogare (query triggers)</i>	109
2.11.2. <i>Trigger-i de validare</i>	111
2.11.3. <i>Trigger-ii navigationali</i>	114
2.11.4. <i>Trigger-ii tranzactionali</i>	115
3. ORACLE9I REPORTS DEVELOPER	120
3.1. INTRODUCERE	120

3.2.	PROIECTAREA RAPOARTELOR	123
3.2.1.	<i>Componentele Reports Builder</i>	123
Object Navigator	124	
Report Editor	125	
Property Inspector	127	
Editorul PL/SQL.....	127	
3.2.2.	<i>Instrumente de tip wizard</i>	127
3.3.	REPORT EDITOR – 5 MODURI DE VIZUALIZARE A RAPORTULUI	136
3.4.	RAPOARTE CU GRUPURI MASTER-DETAIL	145
3.5.	MODIFICAREA RAPOARTELOR FOLOSIND DATA MODEL	152
3.5.1.	<i>Crearea manuala a grupurilor</i>	153
3.5.2.	<i>Adaugarea manuala a coloanelor</i>	156
3.6.	MODIFICAREA RAPORTULUI FOLOSIND PAPER LAYOUT	159
3.6.1.	<i>Cele 3 sectiuni ale raportului: header, main, trailer</i>	159
3.6.2.	<i>Margini</i>	162

1. Introducere

Corporația *Oracle* oferă un set complet de instrumente pentru gestionarea datelor și dezvoltarea de aplicații performante. Produsele incluse în versiunea 9i sunt: ***Oracle9i Developer Suite*** (*Oracle9iDS*), care permite crearea de aplicații ce vor fi stocate și rulate pe ***Oracle9i Application Server*** (*Oracle9iAS*) și care accesează datele din ***Oracle9i Database***.

Versiunea 9i a sistemului *Oracle* prezintă o arhitectură pe 3 niveluri (3-tier): clientul este pe primul nivel, *Oracle9i Application Server* pe nivelul din mijloc și *server-ul* de baze de date este pe cel de-al treilea nivel.

Oracle9iDS (Developer Suite) este un set complet de instrumente pentru crearea de aplicații și servicii *web*, asigurând întregul ciclu de viață al unei aplicații, de la modelare, scriere de cod de orice tip (inclusiv *Java Server Pages*, *servlet-uri*, forme), până la testare și depanare.

Produsele *Oracle9i* folosesc un set comun de tehnologii, care permit utilizarea aceleiași aplicații în medii cu configurații arhitecturale variate. *Java*, *XML* și *PL/SQL* sunt limbaje complementare utilizate pentru dezvoltarea de aplicații, iar *Oracle9iDS* permite folosirea în paralel a celor trei.

Oracle9i Developer Suite oferă instrumente vizuale, *wizard-uri*, utilitare pentru verificarea automată a sintaxei, pentru editarea elementelor grafice și de interfață, toate acestea permitând dezvoltarea rapidă de aplicații și ușurarea muncii programatorului. Aplicații de tipuri diferite cer instrumente de dezvoltare diferite. Această idee este implementată cu succes în *Oracle9i Developer Suite* unde găsim, în principal, următoarele componente:

1. *Oracle Designer* – conține instrumente ce permit specificarea cerințelor și proiectarea schemei aplicației într-un mediu vizual, oferind utilitare de *reverse enginnering* și generare automată de cod;
2. *Oracle Forms Developer* – se ocupă de prezentarea logică a aplicației, generând formulare sofisticate ce permit operații *DML* pe tabelele bazei de date; formularele pot conține câmpuri calculate, sume totale sau parțiale, liste diverse, alerte – toate create cu un efort de programare minim;
3. *JDeveloper* – este un mediu integrat pentru crearea, depanarea și gestionarea aplicațiilor *XML* pe mai multe niveluri și a componentelor bazate pe *J2EE*;

4. *Oracle Portal* – organizează toată informația (conținut web, aplicații) sub forma unui portal. Poate fi permis accesul de la distanță al dispozitivelor *wireless*, opțiune utilă pentru persoanele care călătoresc des;

5. *Oracle Reports Developer*, *Oracle Discoverer*, *Oracle Express* – permit analizarea informației din surse diverse, crearea de rapoarte complexe și distribuirea lor în formate diverse și în medii diverse (*email*, imprimantă, fișier, *portlet*). Un raport poate fi împărțit logic în mai multe secțiuni și, printr-o singură rulare, secțiuni diferite vor fi trimise către destinatarii diferenți.

Oracle9iDS face saltul de la arhitectura *client-server* la cea *web-enabled*. În versiunile *6i* și anterioare acesteia era necesară instalarea de componente *Oracle* pe fiecare *client*. Pentru rularea formularelor și a rapoartelor trebuia instalată pe *client* componentele *Forms RunTime* și *Reports RunTime*. În versiunea *9i* aceste *runtime*-uri se instalează pe *Oracle9iAS*. Deci, pe *client* nu este necesară instalarea de aplicații *Oracle*, fiind încarcata, la rulare, doar interfața grafică a aplicației.

Algoritmul parcurs la rularea unui formular este următorul: clientul (reprezentat de un *browser web*) trimite o cerere către *server-ul* de aplicații (*Application Server - AS*). Această cerere este preluată de *server-ul* *HTTP Oracle (Apache)*, care recunoaște tipul ei și o redirecționează prin *server-ul* *web* către instrumentele corespunzătoare (de exemplu, dacă este un formular, îl trimit către *Oracle Forms*). În urma prelucrării cererii înaintate de către *browser*, *Oracle9iAS* îi răspunde acestuia printr-un document *HTML* care poate fi construit în unul din mai multe moduri: *PL/SQL*, *Java*, printr-un program *CGI* etc.

Oracle9iAS prezintă o arhitectură pe niveluri, cu următoarele tipuri de servicii:

- de comunicare – pentru recunoașterea unui număr mare de protocoale;
- un *container J2EE* – creează un mediu *runtime* pentru aplicații;
- servicii sistem;
- servicii de conectivitate pentru sisteme diverse.

Dintre componentele *Oracle9i Developer Suite* ne vom opri, în capitolele următoare, asupra următoarelor două: *Forms Developer* și *Reports Developer*.

2. Oracle9i Forms Developer

2.1. Introducere

Oracle Forms are două componente: *Oracle Forms Developer*, un mediu de dezvoltare de aplicații, care este inclus în *Oracle9i Developer Suite* și *Oracle9i Forms Services*, un mediu pentru publicarea aplicațiilor, aflat pe *Oracle9i Application Server*.

Oracle Forms Developer oferă un mediu pentru dezvoltarea de formulare complexe ce permit accesul la datele din baza de date. Integrarea sa cu *Oracle Designer* permite folosirea unui model creat în acest mediu de programare și generarea automată de formulare conform cerințelor proiectate în *Oracle Designer*. Mecanismul de inginerie inversă (*reverse engineering*) permite ca modificările făcute apoi în *Oracle Forms Developer* să fie implementate automat în model, păstrându-se, astfel, integritatea dintre model și aplicație.

Limbajul de programare folosit în *Forms* este *PL/SQL*, codul aplicațiilor fiind interpretat în timpul rulării. Interfața *web* a unei aplicații *Forms* este un *applet Java*, obținut fără ca utilizatorul să scrie un singur rând de cod *Java*.

Oracle9i Forms folosește un interpretor *runtime*, dar înlocuiește motorul *client/server* din versiunile anterioare cu unul *web*, care încarcă pe stația *client* interfața aplicației folosind un set generic de clase *Java*. Principalul avantaj este că se pot publica formele pe *web* fără a modifica nici o linie de cod. Formele *web* sunt accesibile printr-un *URL* scris în bara de adrese a *browser*-ului.

Iată, în continuare, câteva din caracteristicile reprezentative ale produsului:

- *Oracle Forms Developer* conține *wizard*-uri pentru crearea rapidă de aplicații în care nu este nevoie de nici o linie de cod. Funcționalitatea formularelor este asigurată de proceduri și funcții predefinite (numite *built-ins*).
- *Unitățile* de program *PL/SQL* pot fi plasate pe *server*-ul bazei de date sau în interiorul aplicației, în funcție de cerințe. Se poate face *drag-and-drop* asupra obiectelor, deplasându-le între module și *server*-ul bazei de date.
- *Aplicațiile Oracle9iDS* pot fi folosite de un singur utilizator sau de mai mulți utilizatori simultan, fără nici o modificare în cadrul sursei.
- *Oracle Forms Developer* folosește modelul de moștenire obiectuală, care permite moștenirea atributelor și a codului între obiecte și module.

2.2. Arhitectura serviciilor *Forms*

Componenta *Forms Services* reprezintă un cadru de lucru pe trei niveluri folosit pentru a publica formularele dumneavoastră pe *Internet*, folosind interfețe *Java* deosebite. Cele trei niveluri sunt: clientul, *server-ul* de aplicații și baza de date *Oracle9i*. Modul de lucru cu o aplicație *Oracle9i Forms* respectă următorul algoritm:

- 1) utilizatorul (reprezentat printr-un *browser web*) încearcă o cerere sub forma unei aplicații, ce va fi accesată printr-o adresă *URL*. Acest *URL* este trimis *server-ului HTTP (Apache)*, care sesizează că este vorba despre o aplicație *Developer Forms* și trimite cererea către *Forms Servlet*.
- 2) *Forms Servlet-ul* detectează ce *browser* are utilizatorul, apoi creează dinamic o pagină *HTML* conținând toată informația necesară pentru inițierea unei sesiuni *Forms* (numită și motor *runtime*). După ce pagina este încarcată în *browser*, acesta inițiază descărcarea unui *applet Java client*, constând într-un set de clase *Java* (doar dacă acest *applet* nu a fost deja descărcat în cadrul unei alte sesiuni, caz în care este păstrat în memoria *cache*). Clientul *Forms Java* rulează pe o mașină virtuală *Java* (care, implicit, este *JInitiator*) și accesează *servlet-ul Forms Listener* (aflat pe nivelul din mijloc unde este *server-ul* de aplicații). *Forms Listener* deschide un motor *runtime web*, creând pentru această sesiune un *cookie*, depozitat în memorie sau atașat *URL-ului* la fiecare cerere și răspuns. Acest *cookie* nu este niciodată stocat pe calculatorul utilizatorului.
- 3) Pentru fiecare client *Forms listener* demarează câte un proces *runtime web* nou (*ifweb90.exe*). Acest proces este cel care se conectează la baza de date și comunică cu clientul *Forms*, gestionând logica aplicației și procesarea. Astfel, datele nu sunt încărcate în *browser-ul client*, ci interschimbate între *server-ul* de aplicații (*Application Server*) și *server-ul* de bază de date (*Database Server*), reducându-se traficul în rețea și îmbunătățindu-se performanțele. Procesul *runtime* comunică cu clientul *Java* folosind protocolele *HTTP* sau *HTTPS*.
- 4) *Servlet-ul Forms Listener* stabilește o conexiune cu motorul *runtime*, care se conectează la baza de date. El este cel care, prin intermediul *server-ului HTTP* sau al *container-ului OC4J* gestionează comunicarea dintre *applet-ul Forms* și motorul *runtime*. Motorul *runtime web* comunică intelligent cu clientul *Java*, comparând mesajul actual cu cel transmis anterior și transmitând doar informația ce a fost actualizată. Astfel, traficul în rețea este redus substanțial.
- 5) Interfața utilizator *Forms* este un *applet Java* care folosește un set de clase *Java* prestabilite. Aceasta afișează interfața utilizator a aplicației în fereastra

browser a utilizatorului. Odată descărcat acest *applet* în *browser-ul* clientului, pot fi rulate oricât de multe și de complexe aplicații *Forms*, fără descărcări ulterioare. Setul de clase *Java* este păstrat în memoria *cache* până la detectarea unei versiuni noi a *Forms-ului*, caz în care va fi actualizat. Interfețele *Java* utilizate în *Oracle9i Developer Forms* oferă același *design* și calitate ca cele utilizate în aplicațiile *client/server*, fără a cere din partea utilizatorului final cunoștințe superioare. În plus, logica aplicației este mutată pe nivelul din mijloc, al *server-ului* pe aplicații, asigurând performanțe sporite aplicației.

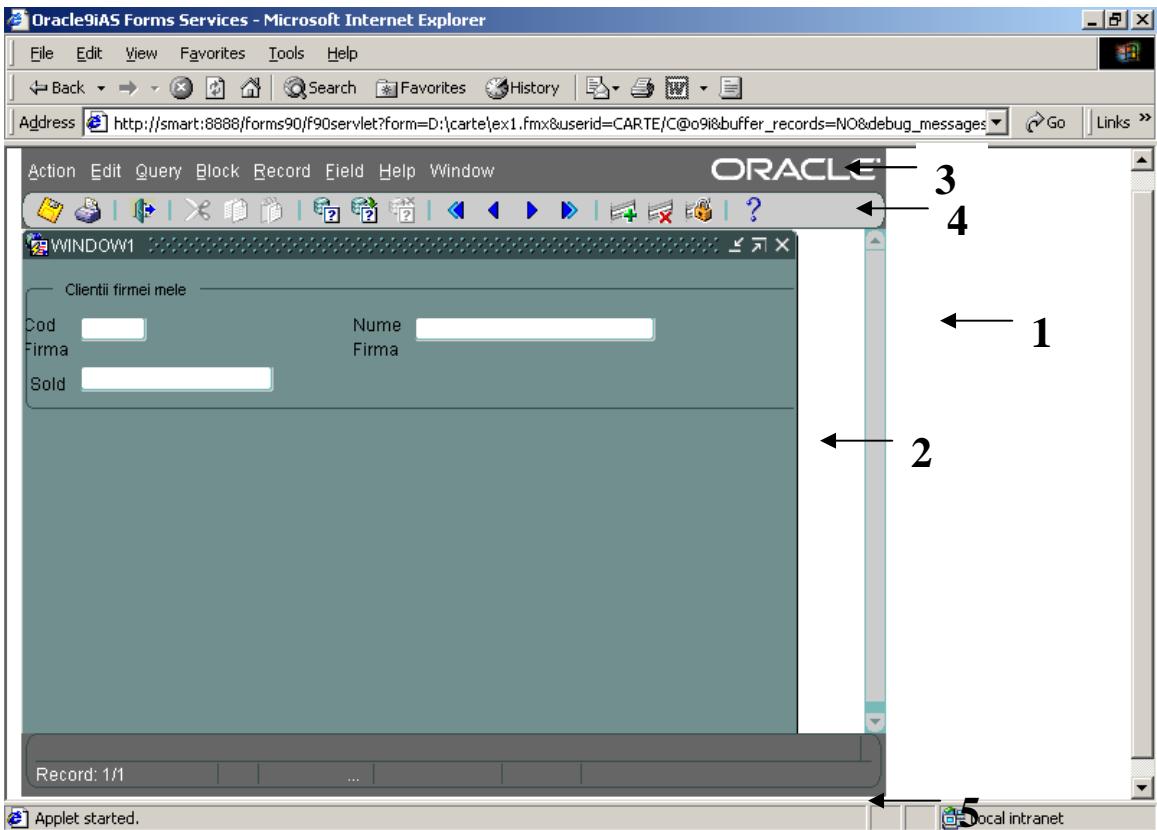
2.3. Publicarea formelor

Procesul de publicare pe *web* a formelor este implementat, deci, conform ideii că logica aplicației și motorul *runtime* al serviciilor *Forms* se află pe *server-ul* de aplicații. Toate procesele ce acționează asupra datelor se execută la nivelul bazei de date și al *server-ului* de aplicații, în timp ce interfața aplicației se află pe clientul *Forms*. Utilizatorul final rulează o aplicație *Forms Developer* într-un *browser web*, invocând-o printr-o adresă *URL* care pointează către aplicație. Componenta *Forms Services* generează apoi un fișier *HTML* care descarcă un *applet Java* pe mașina *client*. Acest mic *applet* este cel care permite afișarea interfeței formei, în timp ce logica aplicației se execută pe *server-ul* de aplicații.

La rularea formei sunt vizibile următoarele:

1. fereastra *browser-ului*;
2. *applet-ul Java* (conținut în fereastra *browser-ului*);
3. meniul implicit (conținut în *applet*);
4. meniul *smartbar* (conținut în *applet*);
5. bara de stare (conținută în *applet*).

O cerere înaintată de utilizator va avea următorul rezultat:



Meniul implicit este parte a tuturor aplicațiilor *Forms* și conține intrări pentru operațiile de bază ce se pot efectua asupra datelor din formă (salvare în baza de date, încărcarea în formular a unei înregistrări noi, ștergerea din baza de date a înregistrării curente, operații de *commit* și *rollback* etc.) Acest meniu poate fi eliminat sau personalizat, aşa cum vom vedea într-un capitol ulterior.

Meniul *smartbar* conține butoane a căror apăsare are aceeași acțiune cu unele dintre opțiunile meniului implicit și este utilizat pentru accesarea directă a acestora. În *runtime*, acest meniu este afișat deasupra oricărui eventual meniu definit de utilizator. Ca și cel precedent, și meniul *smartbar* poate fi eliminat sau personalizat.

Bara de stare este afișată în partea inferioară a ferestrei și conține informații dependente de context, cum ar fi:

- *Record:n/m* - sunt afișate înregistrări începând de la a *n*-a din grupul de *m* înregistrări încărcate de pe *server*;
- *Enter-Query* – blocul curent este în modul *Enter-Query*;

- *LOV* – elementul curent (cel pe care este concentrat *focus-ul*) are asociată o listă de valori;

În bara de adrese a *browser-ului* se poate observa *URL-ul* prin care este invocată aplicația. În cazul nostru, acesta este:

http://smart.com:8888/forms90/f90servlet?form=D:\carte\rulare.fmx&userid=CARTE/C@o9i&buffer_records=NO&debug_messages=NO&array=YES&query_only=NO&quiet=NO&RENDER=YES

Componentele *URL-ului* sunt:

Protocolul	<i>http</i>
Domeniul	<i>smart.com</i>
Portul pentru <i>server-ul HTTP</i> sau pentru <i>OC4J</i>	Xxxx – implicit pentru <i>server-ul http</i> 8888 – implicit pentru <i>OC4J</i>
Alias-ul <i>servlet-ului Forms</i>	<i>/forms90/f90servlet</i>
Parametri: secțiunea începe cu caracterul „?”; parametrii sunt separați prin „&” și pot fi specificați în <i>URL</i> sau în fișierul de configurare <i>Forms</i>	<i>form=D:\carte\rulare.fmx</i> <i>userid=CARTE/C@o9i</i> <i>buffer_records=NO</i> <i>debug_messages=NO</i> <i>array=YES</i> <i>query_only=NO</i> <i>quiet=NO</i> <i>RENDER=YES</i>

URL-ul pointează, deci, către *servlet-ul Forms* (*forms90/f90servlet*). Acesta este un *servlet Java* care creează o pagină *HTML* dinamică în conformitate cu informația luată din următoarele surse: fișierul de configurare *web* (*formsweb.cfg*), fișierul *HTML* de bază al *Forms-ului* (*basejini.html*) și parametrii din adresa *URL* a aplicației.

2.4. Proiectarea formelor

2.4.1. Diagrama conceptuală exemplu

Exemplile din lucrare se referă la un model de gestiune a componentelor de calculatoare dintr-un depozit. Modelul ales este simplu, astfel încât înțelegerea să să necesite efort minim, scopul nostru constând în focalizarea atenției cititorului către însușirea modului de lucru cu produsul *Oracle Forms*. Schemele relaționale din exemplele folosite în lucrare sunt:

- stoc (cod_prod#, tip_prod, den_prod, pret_unit, cantit);
- clienti (cod_firma#, nume_firma, sold);
- facturi (cod_fact#, cod_firma, data, io);
- produse (cod_fact#, cod_prod#, cantit).

Să facem următoarele precizări: clienții emit facturi pe baza cărora pot cumpăra sau vinde componente de calculatoare. Datele fiecărei facturi sunt reținute în tabela *facturi*; fiecare factură are un *cod_fact* care o individualizează. Câmpul *io* va reține tipul facturii (1 dacă factura este de intrare, 0 dacă este de ieșire). Pe o factură pot fi cumpărate (sau vândute) mai multe produse. Acestea sunt gestionate de tabela *produse*. Tabela *stocuri* conține informații despre componentele aflate în depozit.

Dorim să creăm o aplicație care să permită operarea facturilor emise de clienți. La inserarea unei facturi noi, stocul și soldul firmei vor fi automat reactualizate.

2.4.2. Componentele *Forms Builder*

Object Navigator

Object Navigator (figura 2.3.1) reprezintă o interfață ce oferă o prezentare ierarhică a obiectelor (aflate atât pe *client*, cât și pe *server*) din modulele deschise. Obiectele sunt grupate după nodurile corespunzătoare. De exemplu, toate ferestrele create pentru modulul curent sunt grupate în nodul *windows*.

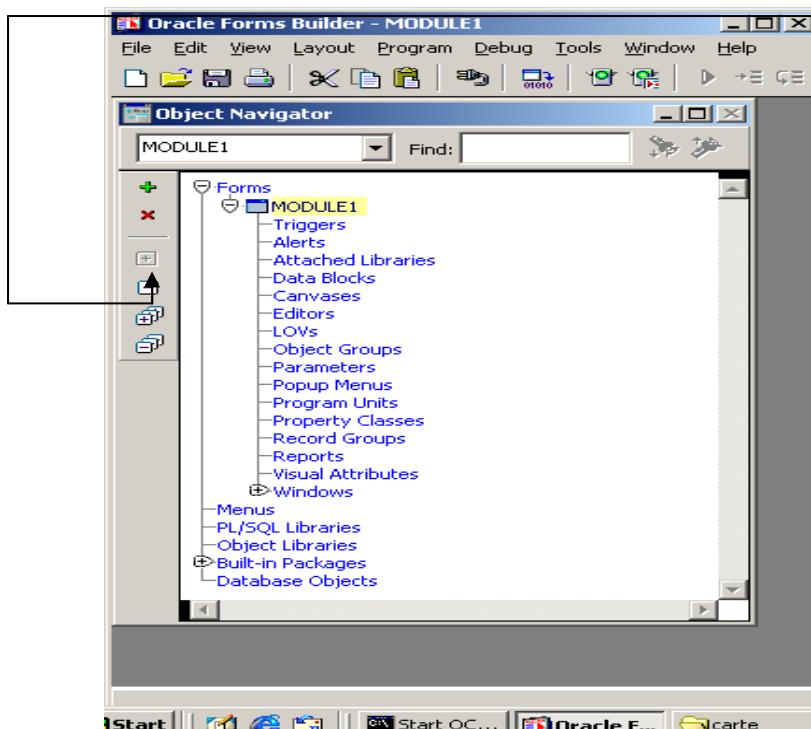


Figura 3.2.1. Interfața Object Navigator.

Nodurile de nivel cel mai înalt în ierarhia *Object Navigator* sunt: forme, meniuri, biblioteci *PL/SQL*, biblioteci de obiecte, pachete predefinite și obiecte ale bazei de date. Toate celelalte noduri sunt indentate pentru a arăta faptul că ele aparțin acestor noduri de nivel superior.

Obiectele și nodurile sunt afișate cu un „+” sau „–” în față, semnificând faptul că sunt sau nu expandate. Obiectele care nu pot conține obiecte de nivel inferior sunt afișate cu un disc în față. Fiecare obiect din *Object Navigator* are atașat un *icon* care îi sugerează tipul. În general, dublu *click* pe *icon-ul* atașat invocă un editor corespunzător obiectului (pentru stil sau conținut).

Prin intermediul interfeței *Object Navigator* se pot efectua operații complexe și diverse asupra obiectelor din formă:

- expandarea și restrângerea nodurilor pentru vizualizarea optimă a obiectelor;
- selectarea, crearea și stergerea de obiecte;
- copierea și mutarea obiectelor între noduri;

- accesarea altor moduri de vizualizare a obiectelor și proprietăților lor;
- localizarea rapidă a obiectelor prin folosirea opțiunii *FastSearch*.

Paleta de proprietăți (*Property Palette*)

Fiecare obiect dintr-un modul (inclusiv modulul însuși) are proprietăți specifice tipului său. Ele sunt accesibile prin intermediul paletelor de proprietăți, care se accesează din meniul de context al obiectului selectat (*click dreapta pe obiect*) → opțiunea *Property Palette* sau apăsând tasta *F4* când obiectul (sau grupul de obiecte) pentru care se vrea afișată paleta de proprietăți este selectat. Paletele de proprietăți ale obiectelor de același tip au întotdeauna aceleași capitulo și aceleași detalii; putem numai seta aceste proprietăți, nu le putem șterge sau adăuga unele noi. Fiind poziționați pe o proprietate, apăsarea tastei *F1* are ca efect afișarea unei ferestre-*help* cu descrierea proprietății respective.

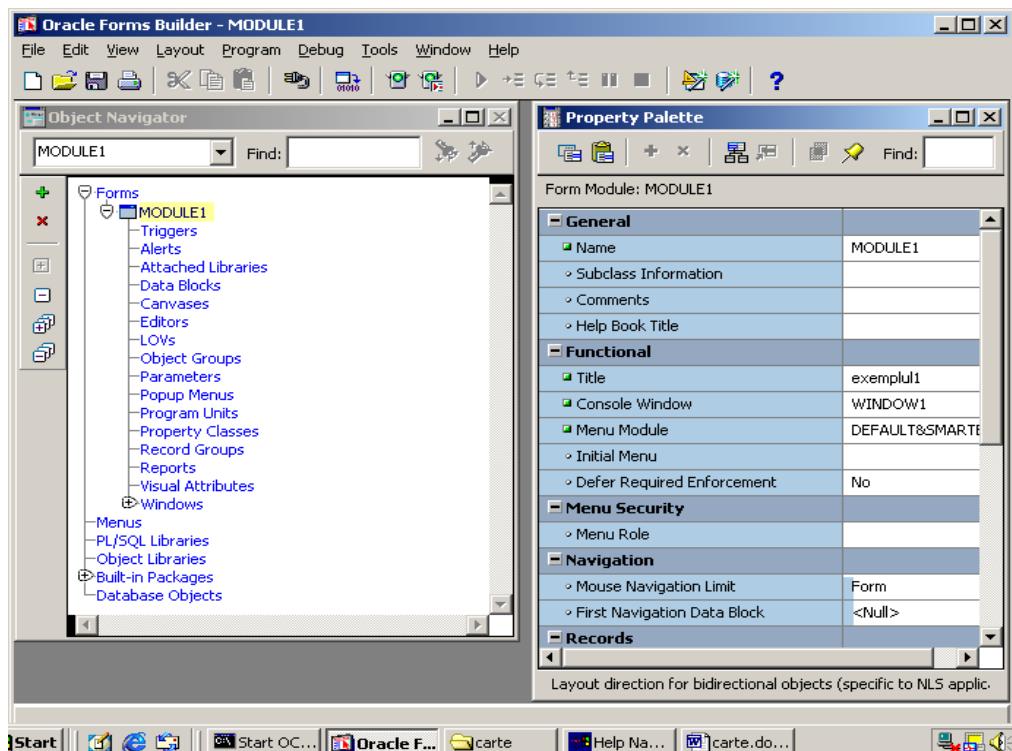


Figura 3.2.2. Paleta de proprietăți a unui obiect modul

În figura 3.2.2 este afișată paleta de proprietăți a unui modul. Observați faptul că proprietățile sunt grupate logic într-o ierarhie, în raport cu funcționalitatea lor. Intrările care au fost lăsate la valoarea implicită sunt prefixate de un disc gri, pe când celor ce au fost modificate le este atașat un dreptunghi verde. Există proprietăți moștenite, care sunt marcate printr-o sageată, precum și proprietăți care inițial au fost moștenite, apoi au fost suprascrisse, caz în care sunt prefixate de o sageată urmată de un „x” roșu.

Lista proprietăților unui obiect conține două coloane: prima indică numele proprietății, iar a doua valoarea proprietății (atributului). Se pot modifica atributele selectând proprietatea corespunzătoare; în cea de-a doua coloană, dedicată atributelor, poate apărea, în funcție de proprietatea selectată, un câmp text (pentru editarea atributului), o listă de tip *pop-up* (pentru atributele care au un set specificat de valori valide) sau un buton *More* (când proprietatea curentă cere valori mai complexe).

Paleta de proprietăți este dependentă de context. Totuși, există câteva intrări comune majorității obiectelor utilizate în formulare:

- zona *General* – dă obiectului un nume și-i definește tipul;
- zona *Functional* – stabilește modul de funcționare a cursorului, atunci când se află în câmpul asociat obiectului;
- zona *Navigation* – definește ordinea și stilul de navigare între obiecte;
- zona *Data* – controlează tipul obiectului și eventuala legătură dintre acesta și baza de date.

Implicit, paleta de proprietăți se actualizează automat la selectarea unui alt obiect. Când este necesară compararea proprietăților a două obiecte, se poate deschide o paletă adițională acționând în prealabil butonul *Pin/Unpin* . Selectarea mai multor obiecte (realizată prin ținerea apăsată a tastei *Shift* și *click* pe fiecare dintre obiecte) afișează o paletă de proprietăți comună obiectelor selectate, ca în figura 3.2.3:

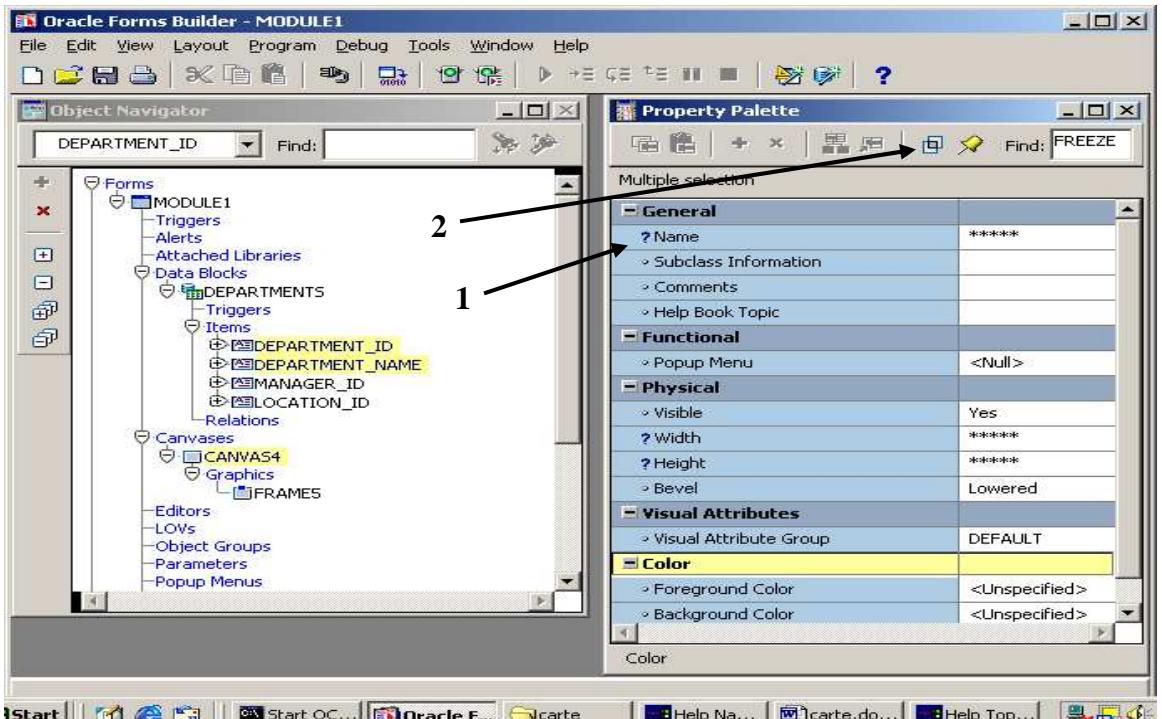
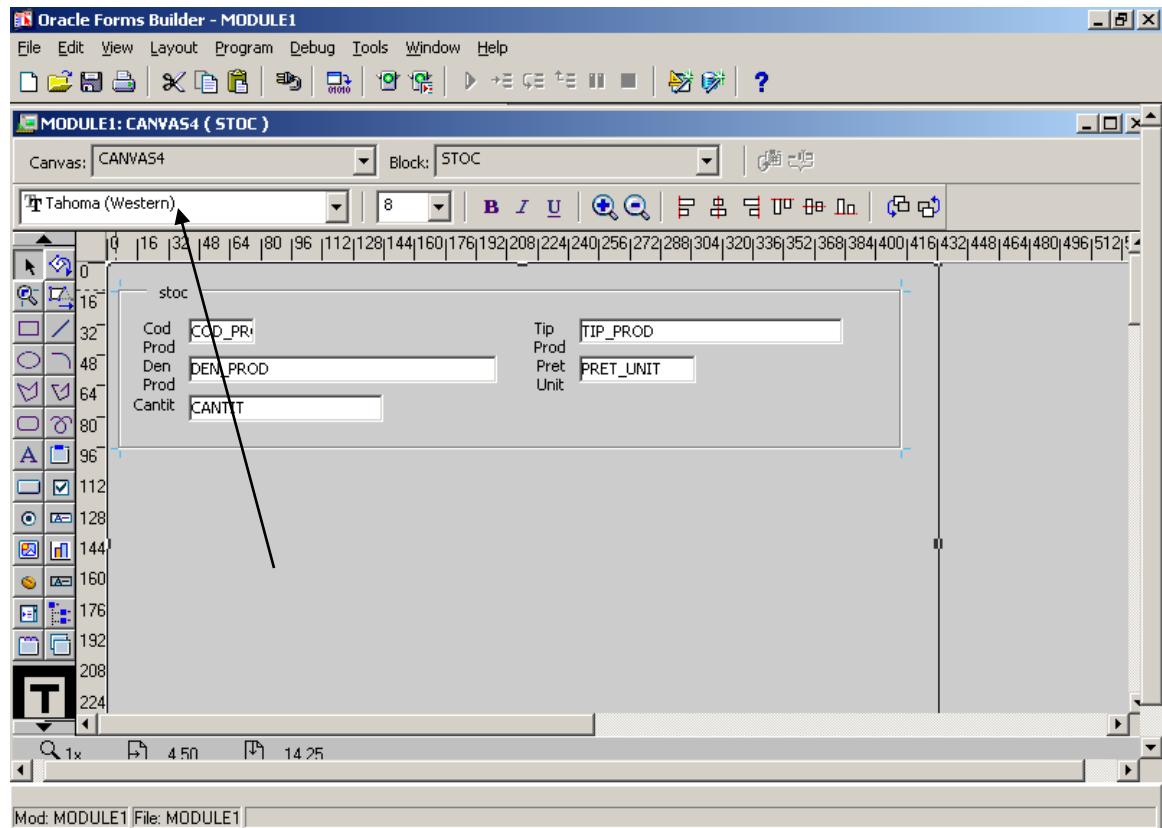


Figura 3.2.3. Paleta de proprietăți comună unui set de obiecte

Implicit, este afișată intersecția proprietăților obiectelor. Acest mod se poate schimba prin apăsarea butonului marcat cu 2 în figura 3.2.3 (butonul *Intersection/Union*). Atributele (valori ale proprietăților) diferite ale aceleiași proprietăți sunt marcate prin ***** (vezi marcajul 1 din figura 3.2.3).

Editorul vizual (*Layout Editor*)

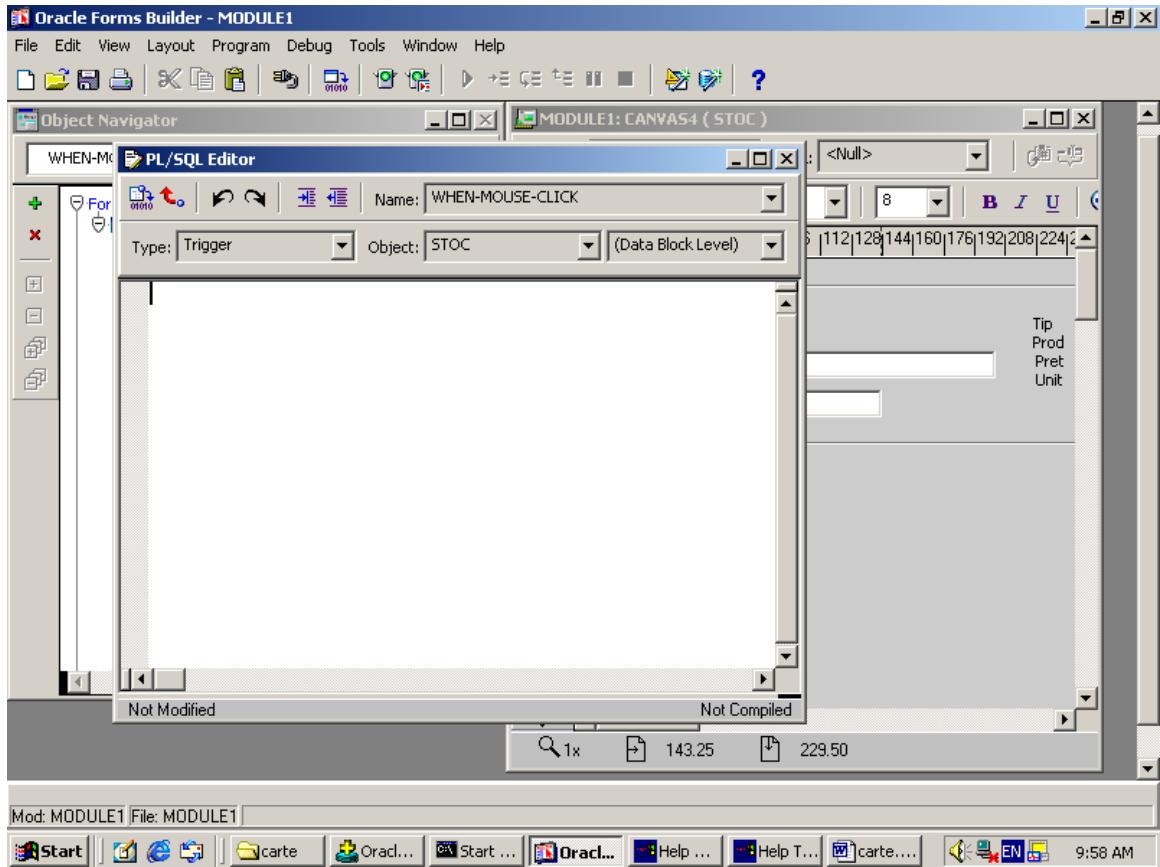
Editorul vizual este un instrument folosit pentru crearea și modificarea interfeței grafice a formei. De aici se pot seta culori, setul de caractere, mărimea și poziția elementelor incluse în formă. Acestea se așeză într-un *canvas*, care este afișat în *runtime* într-o fereastră (*window*). Un *canvas* este o suprafață aflată într-o fereastră-*container*, pe care se poziționează obiectele cu care interacționează utilizatorul la rularea formularului. Setând proprietatea *window* a *canvas-ului*, specificăm în ce fereastră va fi afișat acesta. Utilitarul *Layout Editor* permite lucrul, la un moment dat, cu un singur *canvas*. *Canvas-ul* conține doar elemente cu care utilizatorul nu poate interacționa: texte statice (care pot fi formataate într-o diversitate de moduri de către programator), elemente grafice etc. Făcând analogie cu munca unui pictor, fereastra este cadrul pe care acesta își fixează pânza, reprezentată de *canvas*.



În bara de titlu a ferestrei editorului vizual avem informații despre contextul de lucru, inclusiv *canvas*-ul și blocul în care lucrăm. Putem schimba *canvas*-ul curent selectând un altul din lista indicată de săgeata din figura anterioară. Folosind bara verticală de instrumente din stânga ferestrei editorului vizual putem crea obiecte noi, care vor fi automat asignate *canvas*-ului și blocului curent.

Editorul PL/SQL

Editorul *PL/SQL* permite inserarea și compilarea de cod *PL/SQL*, precum și asocierea acestuia obiectelor din formular. Invocarea editorului *PL/SQL* se face prin *click dreapta* pe obiectul care permite atașarea de cod *PL/SQL*. Obiectele care suportă cod în *Forms Developer* sunt *trigger*-ii, subprogramele (funcții și proceduri), pachetele.

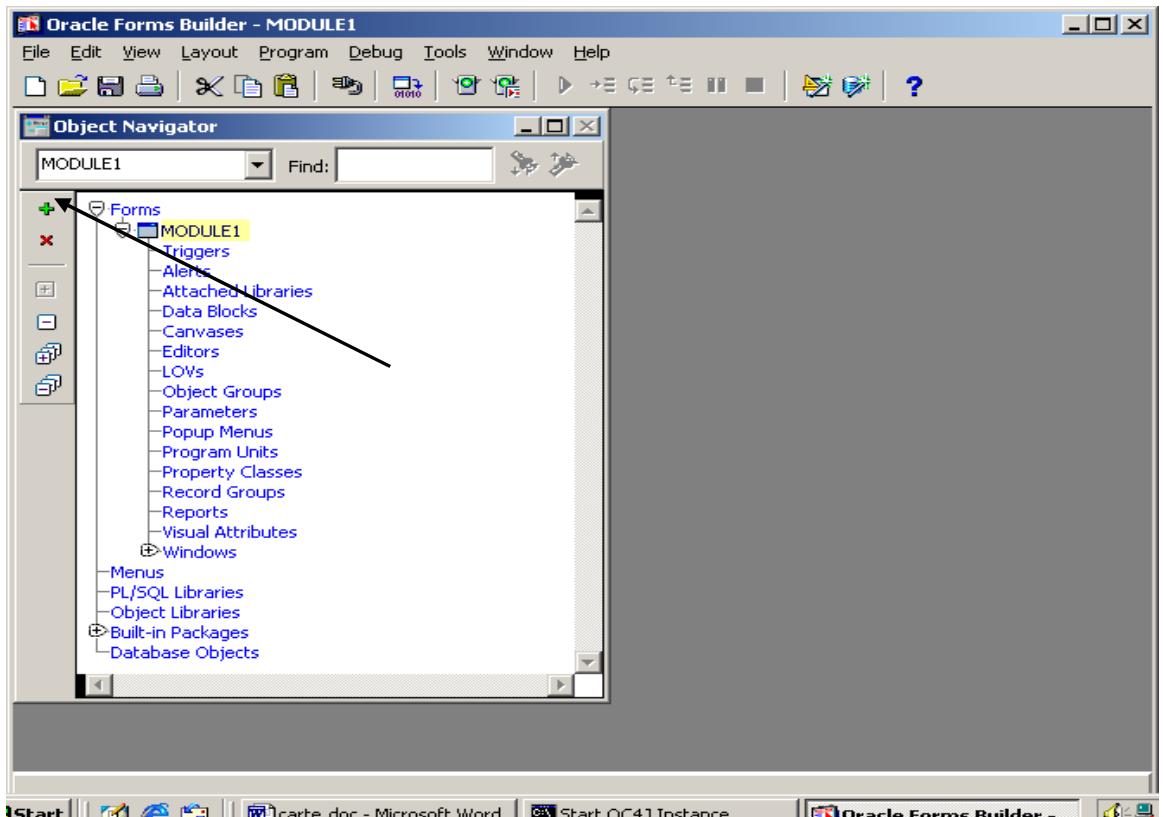


Opțiunea *Type* din partea superioară a ferestrei editorului *PL/SQL* definește tipul obiectului ce va fi creat (în cazul nostru – *trigger*), iar opțiunea *Object* atașează codul unui obiect specificat. În colțul stânga-sus al ferestrei se află butonul *Compile PL/SQL Code*, la apăsarea căruia compilatorul detectează erorile de sintaxă, semantică și referințele către obiecte inexistente.

2.4.3. Instrumente de tip *wizard*

Orice formă este creată pentru a interacționa cu un set de date, care se pot găsi în tabele sau pot fi generate de o procedură stocată. Pașii care trebuie parcursi pentru crearea unei forme noi sunt, în general, următorii:

- crearea unui modul nou – s-a efectuat prin *click* pe semnul  verde (marcat în figura de mai jos cu săgeată) din bara de instrumente a ferestrei *Object Navigator* (sau, echivalent, *File* → *New* → *Form*) și are rezultatul următor:



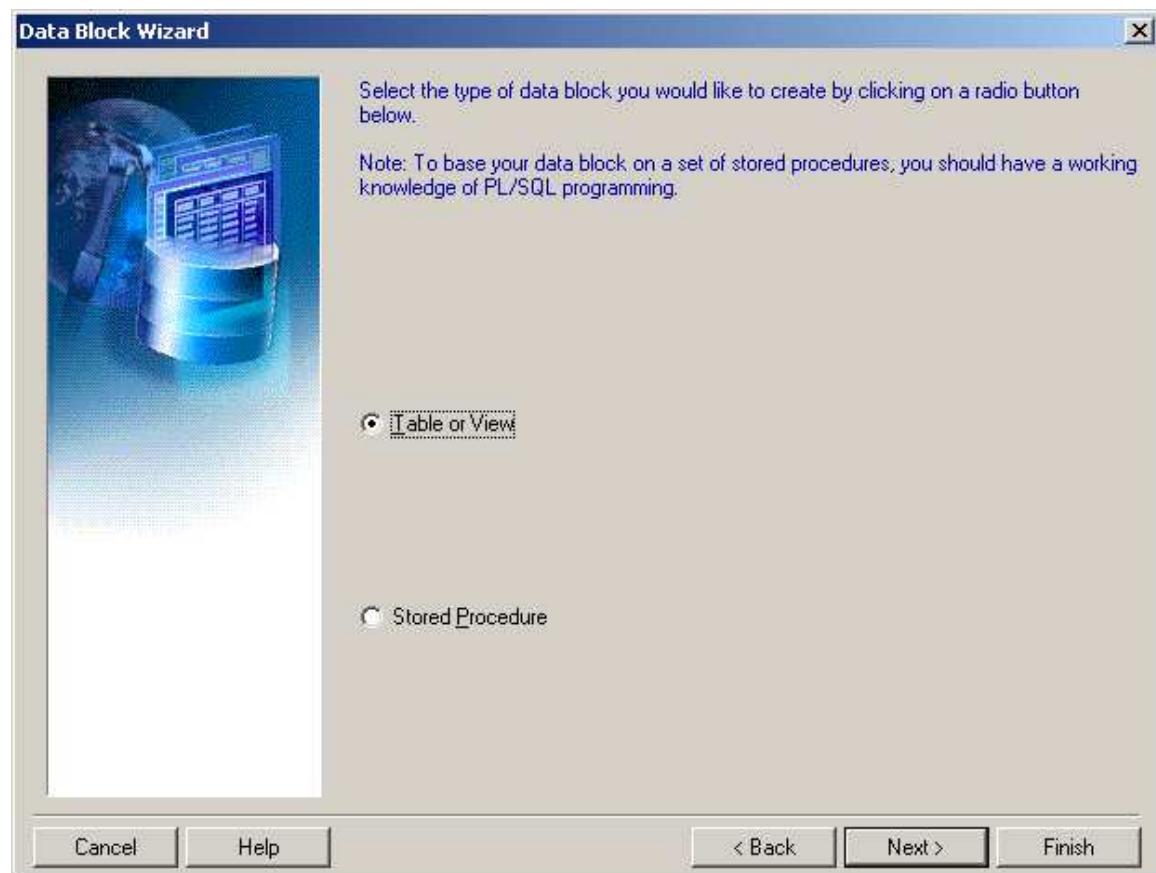
- crearea blocurilor de date și a elementelor sale (în general, se face cu ajutorul instrumentului *Data Block Wizard*);
- definirea proprietăților vizuale ale formei (cu ajutorul lui *Layout Wizard* sau *Layout Editor*);
- setarea proprietăților obiectelor (din paleta de proprietăți a fiecărui obiect);
- adăugarea de cod *PL/SQL*, acolo unde este cazul (folosim editorul *PL/SQL*);
- testarea formei (dând *click* pe butonul *Run Form* , reprezentat în bara de instrumente sub forma unui semafor verde).

Să parcurgem împreună acești pași și să creăm un formular simplu, care să permită gestionarea clientilor firmei. Primul pas constă, după cum am precizat, în stabilirea blocului de date pe care se bazează formularul.

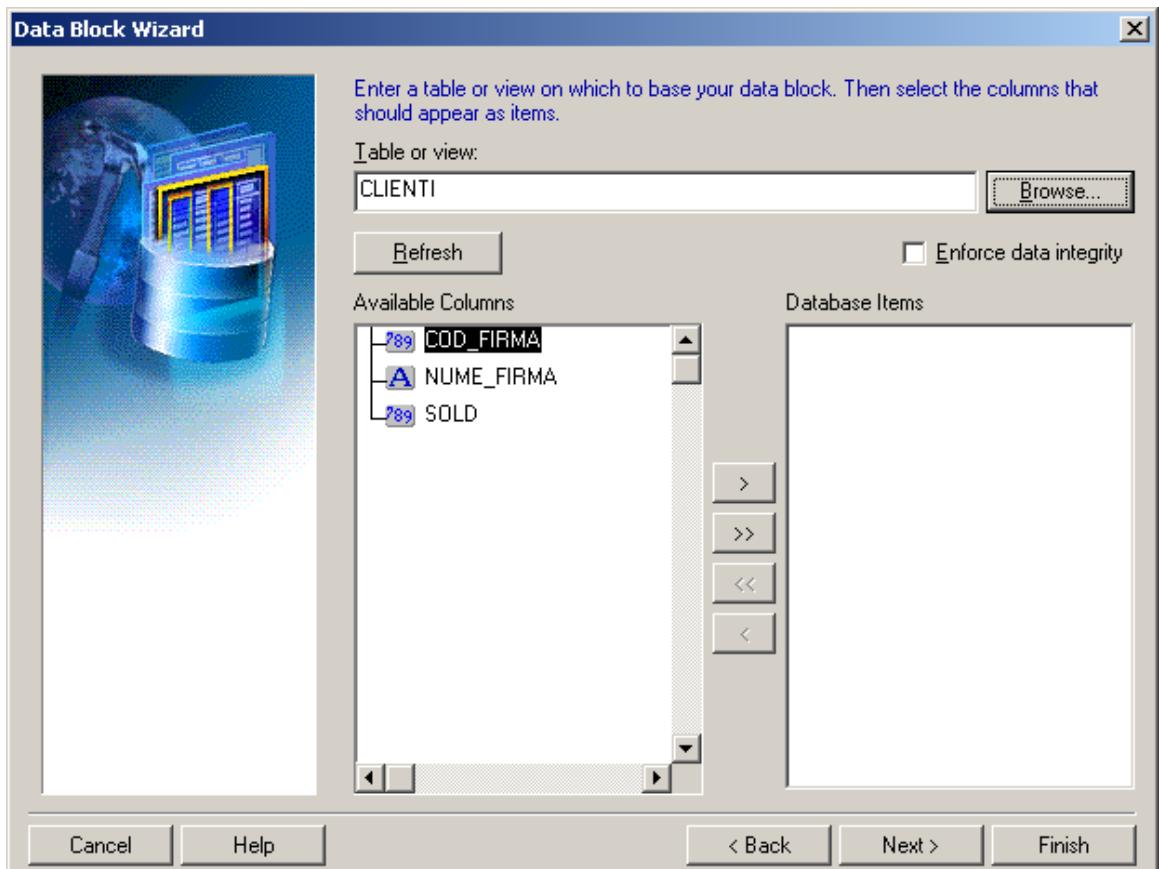
Există două tipuri de blocuri: de date și de control. Blocurile de ultimul tip conțin, în general, butoane care permit efectuarea unor acțiuni (definite prin trigger-i asociati unor evenimente cum ar fi *click* pe buton), câmpuri totalizatoare, câmpuri ascunse (folosite pentru a memora variabile) și orice obiect ce nu trebuie asociat cu un bloc de date. Blocurile de date sunt cele prin intermediu cărora manevrăm datele din baza de date. Într-un bloc de date se stochează liniile ce provin dintr-o tabelă sau cele ce vor fi introduse în tabelă; un bloc de date este, de fapt, un *container* pentru duplicarea informației din baza de date. Fiecare bloc de date gestionează o singură tabelă (folosirea în formă a datelor dintr-o altă tabelă implică definirea unui nou bloc de date) și are o funcționalitate implicită pentru modurile *insert* și *query* (cele două moduri de lucru cu o formă vor fi prezentate în mod detaliat în continuare). Modul în care funcționează un bloc de control trebuie definit explicit, programatic.

Începem, deci, cu definirea blocului de date pe care se bazează forma. Dublu *click* pe intrarea *Data Block* din *Object Navigator* determină apariția unei casete de dialog prin care definim modul în care vom crea blocul de date: folosind *Data Block Wizard* sau manual. Pentru început, să dăm *click* pe *OK*, lăsând bifată opțiunea implicită - *Use the Data Block Wizard*. Acesta este un utilitar care permite crearea sau modificarea rapidă a blocurilor de date folosite în aplicație, precum și generarea automată de cod pentru respectarea constrângerilor de integritate definite în baza de date.

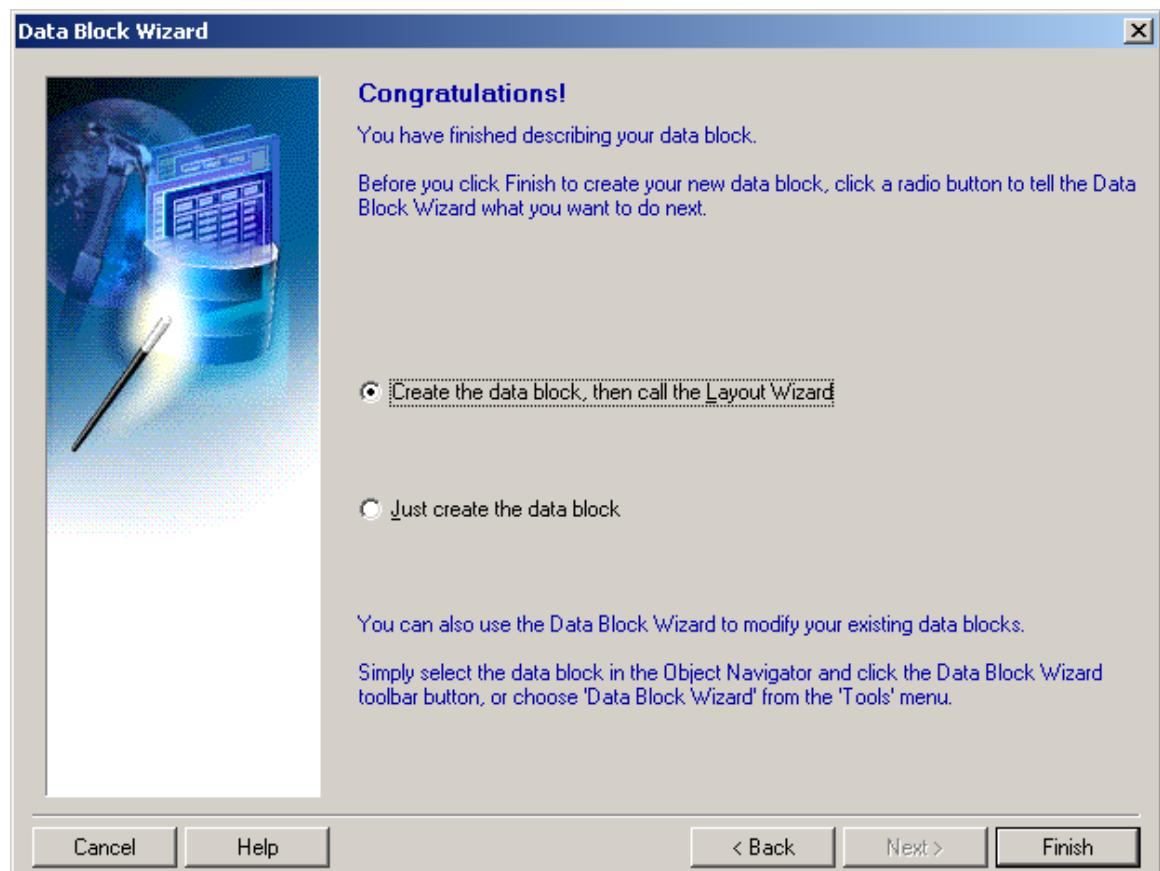
Primul pas al *wizard*-ului constă în afișarea unei ferestre de întâmpinare, a cărei apariție poate fi inhibată prin deselectarea casetei “*Display this page next time*”. Click pe butonul *Next* determină apariția unei ferestre în care suntem invitați să precizăm sursa blocului de date: dintr-o tabelă sau generate de o procedură. (vezi figura de mai jos)



Pentru început, vom crea o formă ce va permite gestionarea clienților firmei care lucrează cu depozitul de componente pentru calculatoare, deci vom lăsa bifată opțiunea “*Table or View*” și vom da *click* pe butonul *Next*, fapt ce va determina trecerea la următorul pas al *wizard*-ului:

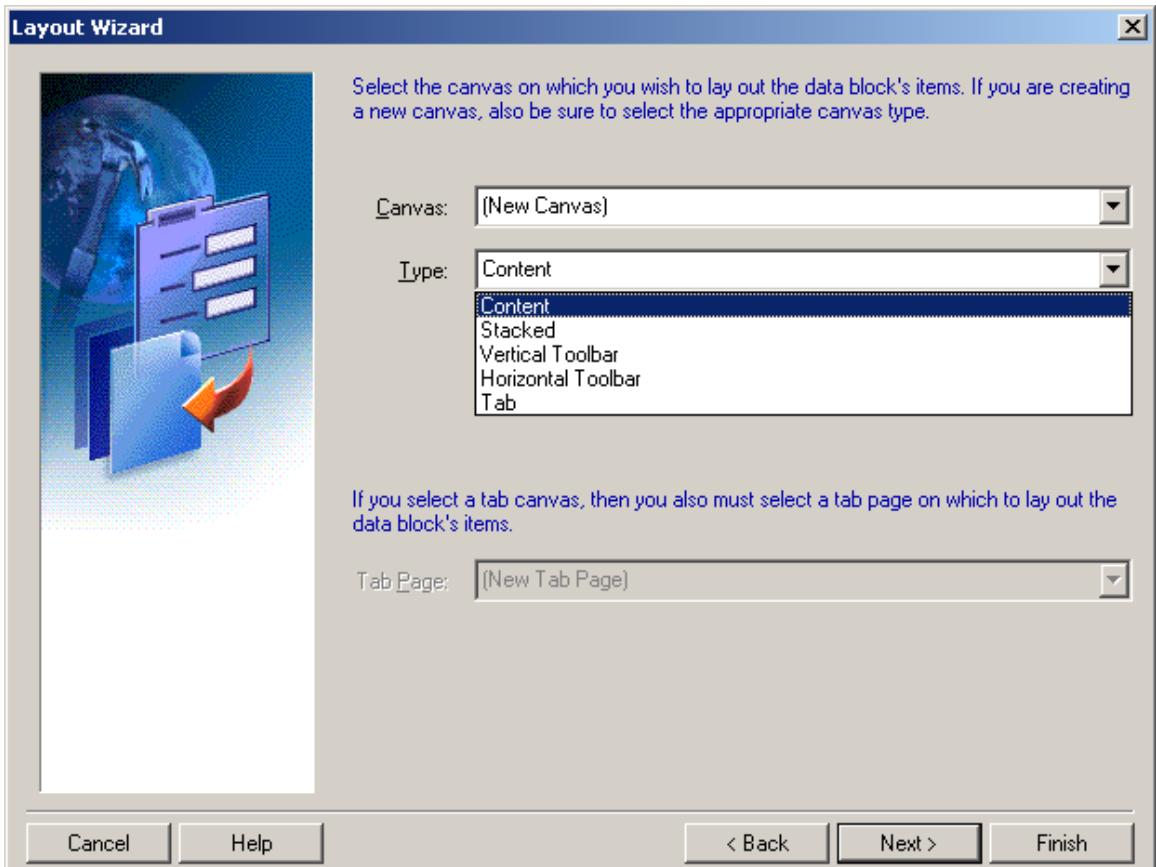


În câmpul “Table or View” va trebui să selectăm tabela sau vizualizarea pe baza căreia construim forma. Click pe butonul *Browse* atașat acestui câmp determină apariția unei casete de dialog *Connect*, prin care facem conectarea la baza de date, apoi afișarea tuturor tabelelor și vizualizărilor din schema utilizatorului curent. Am ales, din lista ce a apărut, tabela *clienti* și zona *Available Columns* a fost încărcată cu coloanele din această tabelă. Folosind butoanele etichetate *>*, *>>*, *<*, *<<* populăm zona *DataBase Items* cu coloanele ce vor fi utilizate în formular. La acest pas va fi creat, pentru fiecare dintre coloanele selectate de noi, câte un obiect de tip *text item*. Selectarea butonului *Enforce Data Integrity* determină impunerea automată a constrângerilor de integritate definite la nivelul bazei de date. Un click pe butonul *Next* ne va conduce la pasul următor al *wizard*-ului, unde denumim blocul de date pe care îl creăm. Implicit, numele blocului de date este același cu cel al tabelii sau vizualizării pe baza căreia a fost creat. Din motive didactice, schimbăm numele blocului de date în *BL_CLIENTI*. Click pe butonul *Next* determină afișarea unei noi ferestre:



În care suntem anunțați că am parcurs cu succes toți pașii *wizard*-ului și suntem întrebați dacă vrem doar să creăm blocul de date sau să invocăm următorul utilitar, *Layout Wizard*. Crearea blocului de date nu a implicat și crearea reprezentării vizuale a obiectelor incluse în el, așa că în continuare ne vom ocupa de modul cum vom afișa conținutul blocului de date în scopul interacțiunii cu utilizatorul. Putem face acest lucru manual sau dând *click* pe butonul *Finish* și continuând, deci, cu instrumentul *Layout Wizard*.

După o fereastră de întâmpinare a cărei apariție poate fi inhibată ca și în *wizard*-ul precedent, urmează pasul al doilea al *wizard*-ului, în care stabilim tipul *canvas*-ului în care vor fi plasate obiectele din blocul de date, precum și alte obiecte noi, pe care le vom defini pe parcurs.

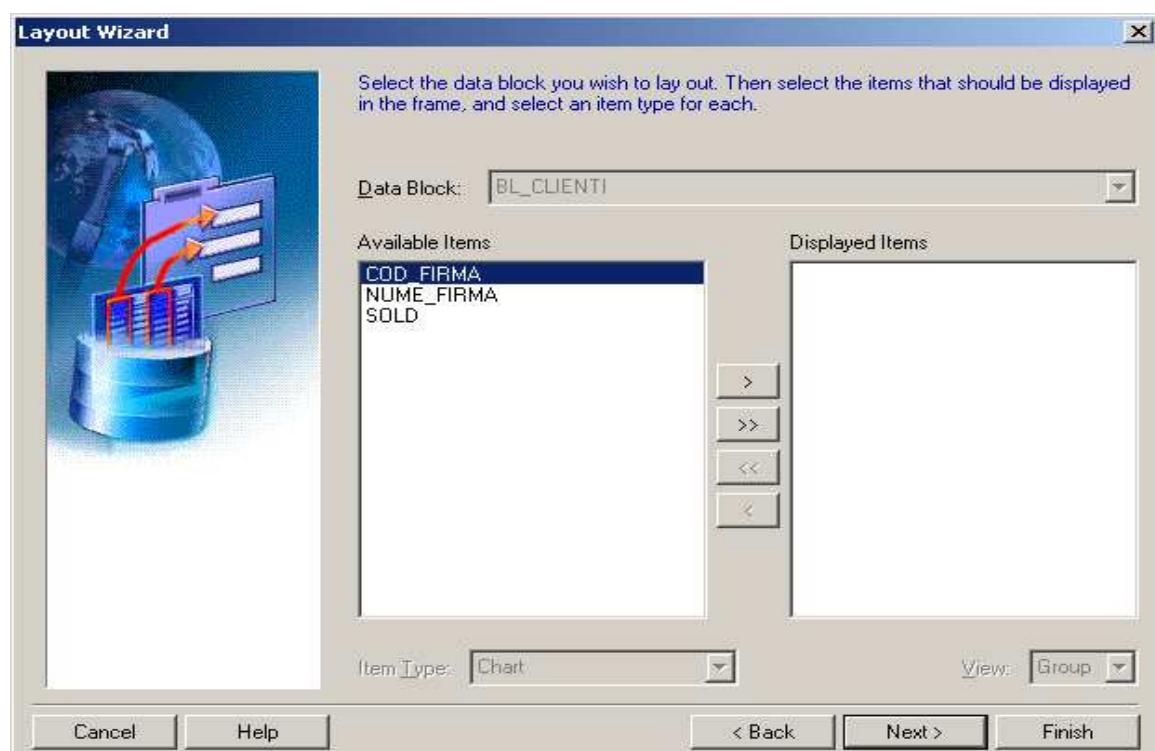


În mod implicit, o formă se bazează pe un *canvas* de tip *content* (conținut), însă o fereastră poate conține mai multe tipuri de *canvas*:

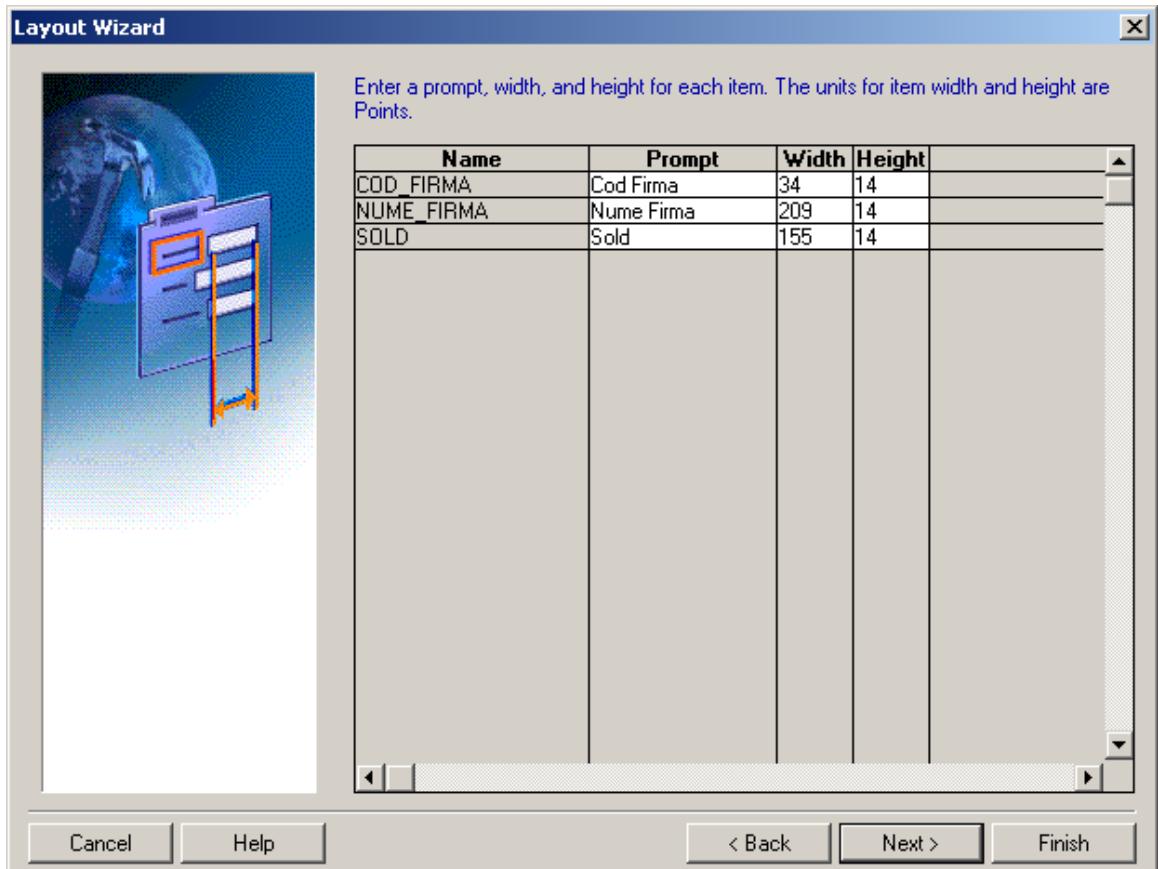
- *stacked* (stivă), folosit în special pentru afișarea de informații la declanșarea unui eveniment (de exemplu, atunci când *focus*-ul intră pe câmpul clienti vrem să apară în fereastra un *canvas* - din punct de vedere vizual va fi un dreptunghi care are definite elemente vizuale - care să ofere informații despre clienti; la click pe suprafața *canvas*-ului, acesta va disparea);
- *vertical toolbar* (bară verticală de instrumente), folosit, în general, pentru definirea unei bare verticale de instrumente. Poate conține butoane la apăsarea cărora se vor declanșa acțiuni specifice, de exemplu, prin secvențe de cod *PL/SQL* atașate butoanelor;
- *horizontal toolbar* (bară orizontală de instrumente) – identic, ca funcționalitate, cu bara verticală. Va fi poziționată automat în partea superioară a ferestrei, pe toată lățimea ei;

- *tab* creează aplicații *tab-type* în care datele înrudite sunt grupate pe aceeași pagină de *tab*. Sunt folosite, în general, când avem tabele cu un număr mare de coloane și aşezarea tuturor în același *canvas* ar produce confuzie.

Alegem ca datele să fie afișate într-un *canvas* de tip *content* și trecem la următorul pas al *wizard*-ului, în care va trebui să specificăm ce câmpuri din blocul de date vor fi afișate în formă:



Dintre câmpurile disponibile (*Available Items*) va trebui să selectăm, folosind grupul de butoane etichetate <, <<, >, >> pe cele care dorim să fie afișate. Câmpurile care nu vrem să fie afișate ca atare și care vor fi folosite (de exemplu, în formule de calcul) sau vor fi completate automat cu ajutorul unor formule trebuie lăsate în zona din partea stângă. Să selectăm toate câmpurile, dând *click* pe butonul >>. În colțul din stânga-jos al ferestrei observăm o listă etichetată *Item Type*. Implicit, tipul obiectelor create astfel este de tip *text item* și la acest moment le lăsăm și noi de acest tip. Dând *click* pe butonul *Next* ajungem la următorul pas al *wizard*-ului:

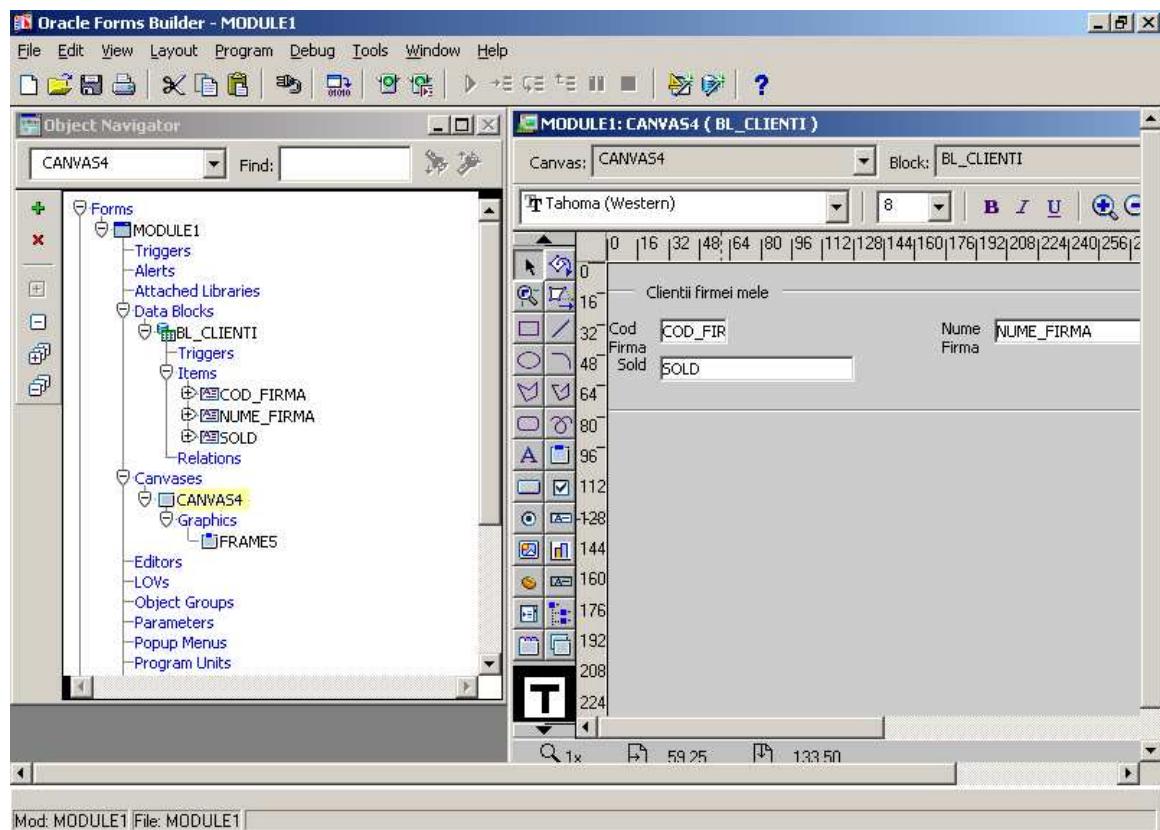


Wizard-ul deduce tipul, lungimea și alte informații referitoare la câmp din tabela pe baza căreia este construit blocul. Putem schimba aici eticheta (*Prompt*), lungimea și lățimea pe care va fi afișat conținutul obiectului din coloana *Name*, aceste modificări influențând *layout*-ul formei, nicidecum structura bazei de date. Dând *click* pe *Next* ajungem la următorul pas al *wizard*-ului, unde suntem întrebați ce stil de prezentare alegem pentru formular: *Form* sau *Tabular*. La selectarea uneia sau alteia dintre cele două opțiuni, în stânga ferestrei va fi afișată o imagine sugestivă a modului în care va arăta formularul. Menționând că stilul *Tabular* este folosit tradițional pentru forme de tip *master-detail*, lăsăm bifată opțiunea *Form* și trecem mai departe dând *click* pe familiarul de acum buton *Next*.

Următorul pas al utilitarului *Layout Wizard* ne cere să dăm un nume cadrului în care va fi afișat blocul de date (să-l completăm: “*Clientii firmei mele*”), numărul de înregistrări ce vor fi afișate la un moment dat (lăsăm valoarea implicită, 1), distanța dintre înregistrări, în cazul în care sunt afișate mai multe) precum și apariția sau nu a unei bare

verticale de derulare. Toate aceste lucruri pot fi modificate ulterior, cu ajutorul paletelor de proprietăți a obiectelor, aşa că la acest moment le lăsăm neschimbate și trecem la următorul pas, apăsând butonul *Next*.

La ultimul pas al *wizard*-ului dăm *click* pe butonul *Finish* și este afișat rezultatul:



În partea stângă a ferestrei *Forms Builder* avem *Object Navigator*, în care au apărut toate obiectele conținute în forma noastră: nodul *Data Blocks* are ca subnod pe *BL_CLIENTI*, care conține cele trei elemente de tip *text*, folosite pentru afișarea unei înregistrări din tabela *CLIENTI*. Aceste elemente sunt așezate în *canvas*-ul cu numele implicit *Canvas4*. Putem schimba numele *canvas*-ului dând două *click*-uri (dar nu dublu *click*) pe nume, apoi tastând numele dorit. Automat, numele *canvas*-ului va fi actualizat pentru obiectele aflate în orice fel de relație cu el.

După cum puteți observa în captura anterioară de ecran, odată cu crearea unui bloc de date cu ajutorul instrumentului *Data Block Wizard*, *Forms Builder*-ul creează automat următoarele:

- un modul care are o funcționalitate moștenită (sunt implementate automat operațiile de *insert*, *delete*, *update* și *query* pentru coloanele din tabela pe care este bazat blocul de date);
- un obiect *frame* (suportul vizual pe care sunt așezate *item*-urile);
- câte un *item* pentru fiecare coloană din tabela inclusă în blocul de date (un *item* este reprezentarea grafică a unui obiect; cele mai multe dintre *item*-uri sunt asociate coloanelor din tabelă);
- o etichetă pentru fiecare *item* (implicit, eticheta este numele coloanei);
- declanșatori (sau *trigger*-i) pentru impunerea constrângerilor de la nivelul bazei de date, în cazul în care caseta de validare “*Enforce data integrity*” a fost bifată.

În partea din dreapta a ferestrei *Builder*-ului s-a deschis automat utilitarul *Layout Editor*, care permite rafinarea aspectului formei. Să observăm că cele două ferestre se sincronizează, în sensul că selectarea unui obiect în una din ele duce la selectarea automată a aceluiași obiect în cealaltă fereastră.

Dacă nu avem pretenții prea mari de la aplicația noastră, putem considera că am încheiat faza de proiectare a formularului. Probabil că scopul muncii noastre nu este crearea unei forme de unică folosință, aşa că este indicat să salvăm rezultatul impresionantului nostru efort intelectual. Facem acest lucru dând click pe *File* → *Save*



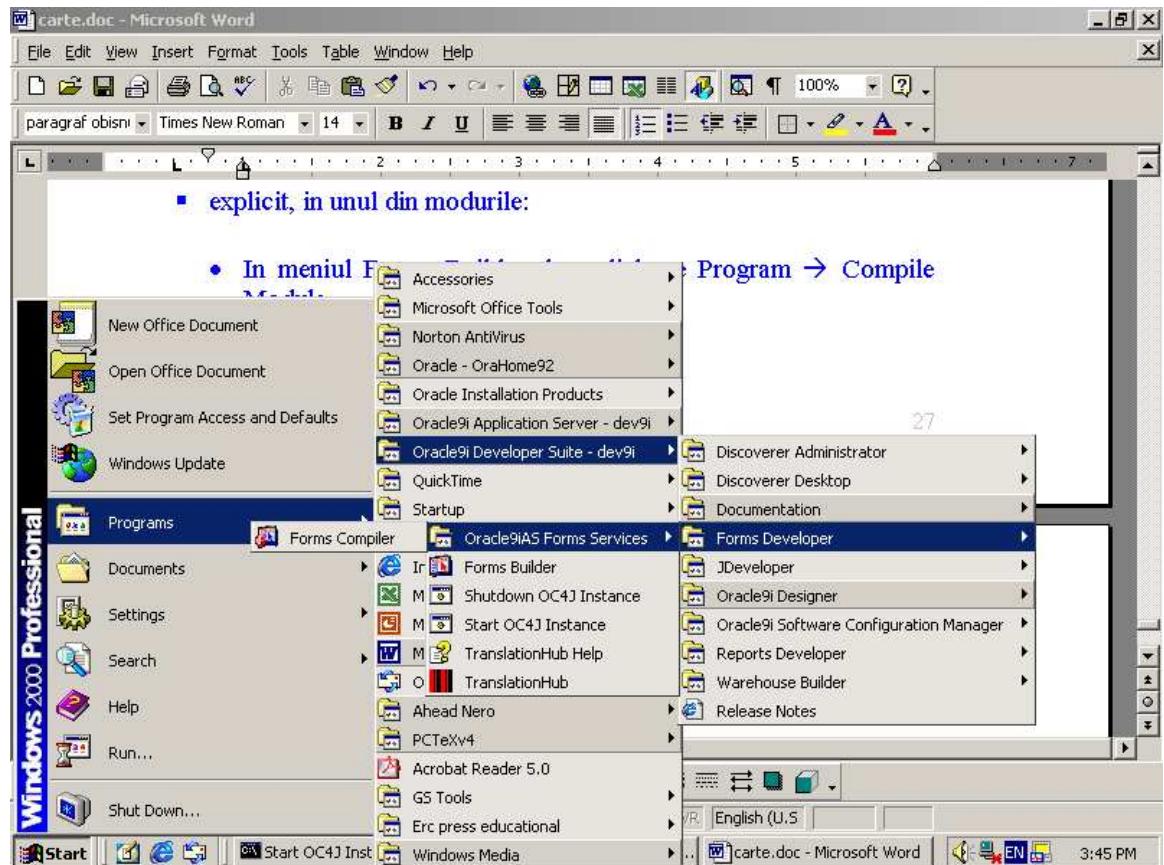
sau direct, pe butonul *Save* din meniul *Forms Builder*. O fereastră de dialog cunoscută deja din mediul *windows* ne invită să alegem un nume și o locație pentru fișierul pe care suntem pe cale să îl creăm. Dând click pe butonul *OK* obținem un fișier cu extensia *.fmb*. Acesta nu este un executabil și poate fi deschis numai cu *Forms Builder*-ul.

Dacă lucrăm simultan cu mai multe module, acestea vor fi stocate în fișiere separate, deci operația de salvare trebuie făcută pentru fiecare în parte.

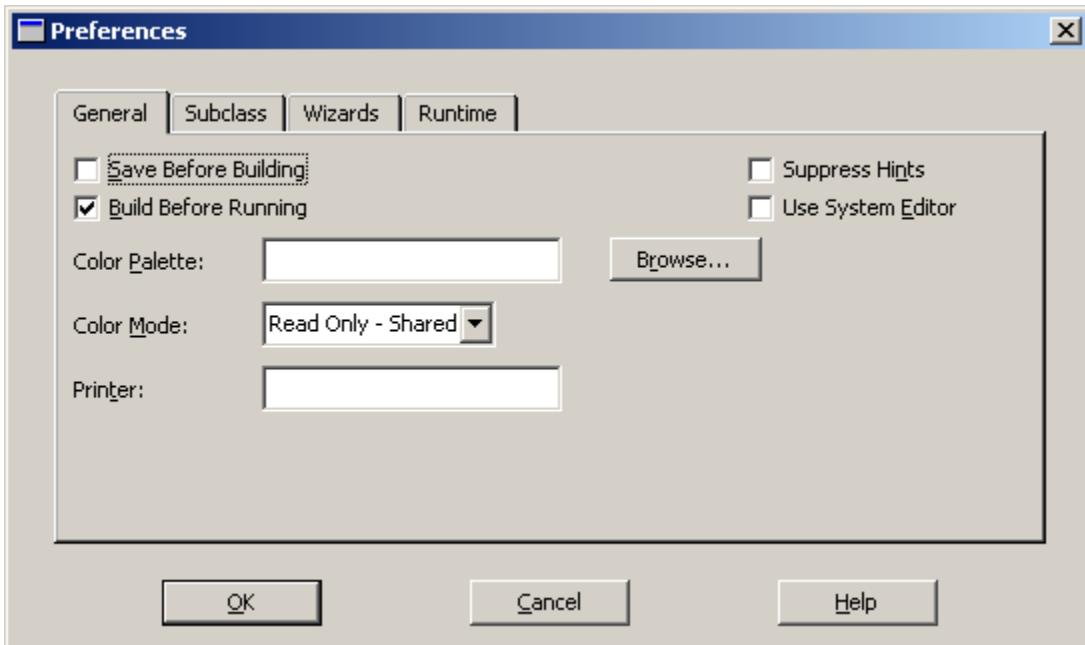
2.4.4. Testarea formelor

Înainte de rularea formei trebuie compilat fișierul *.fmb* creat. Automat, va fi creat un fișier executabil cu același nume și extensia *.fmx*. Operația de compilare poate fi efectuată:

- explicit, în unul din modurile:
 - în meniul *Forms Builder* dăm click pe *Program* → *Compile Module*
 - lansând *Form Compiler* din meniul *Start* al *Windows*-ului:



- lansând executabilul *Form Compiler* (*ifcmp90.exe*) din linie de comandă
- implicit, setând opțiunea *Build Before Running* aflată în meniul *Forms Builder: Edit → Preferences*



În concluzie, modulele *Forms* pe care le creăm sunt stocate în fișiere binare *.fmb* care sunt portabile între platforme. Fișierele executabile *.fmx* sunt stocate, de asemenea, în format binar, dar nu sunt portabile între platforme. Pentru fiecare modul poate fi generat un fișier text (portabil) cu extensia *.fmt*: în meniul *Form Builder* alegem *File→Convert*, iar în caseta de dialog ce apare selectăm tipul modulului, fișierul pe care trebuie să îl convertim și direcția (*Binary-to-Text* sau invers).

Putem genera automat documentație pentru modulul nostru, selectându-l în *Object Navigator* și dând click pe *File → Administration → Object List Report*. Această operație va produce un fișier *ASCII* cu numele identic cu cel al modulului și extensia *.txt*.

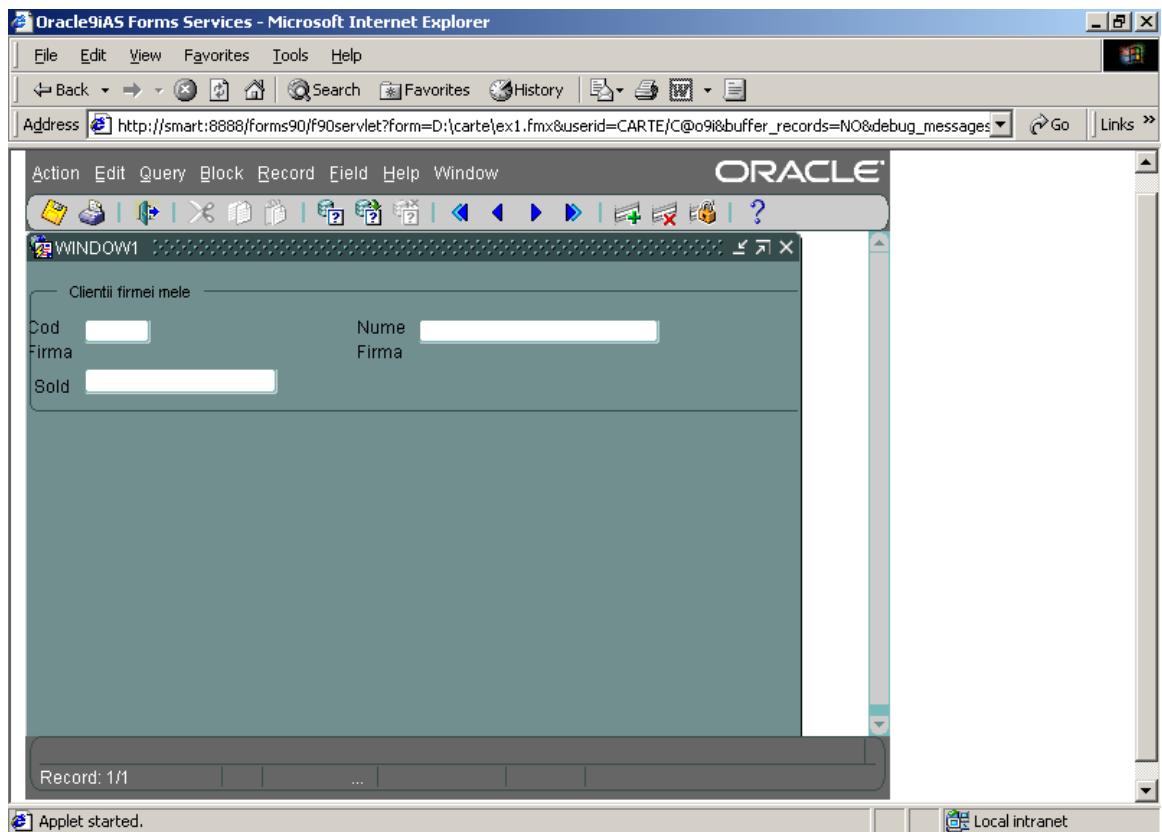
Cu toate că modulele create pot fi testate pe mașina *client*, pentru dezvoltarea unor aplicații profesionale aceste module trebuie transferate pe *server-ul* de aplicații. În cazul în care acesta rulează pe o platformă diferită, se transferă (cu ajutorul utilitarului *ftp*, de exemplu) fișierul *.fmb* pe mașina pe care se află *server-ul* de aplicații, iar acolo se invocă *Form Compiler*, pentru recompilare. În cazul în care *server-ul* de aplicații rulează pe aceeași platformă ca cea pe care a fost dezvoltată aplicația, poate fi transferat direct fișierul *.fmx*.

După ce fișierele executabile au fost transferate pe *Oracle9iAS*, aplicația poate fi încărcată în fereastra unui *browser* prin intermediul *URL-ului* său, care pointează către *servlet-ul Forms*.

Deocamdată ne aflăm, însă, în faza de testare și depanare a aplicației, iar acestea se pot face și pe mașina pe care rulăm *Form Builder*. În acest scop, în *Oracle9i Developer Suite* a fost inclus *Oracle9iAS Containers for J2EE (OC4J)*, astfel nefiind necesară instalarea produsului *Oracle9i Application Server* pentru testarea aplicațiilor *Developer Forms*.

Instanța *OC4J* se pornește dând dublu *click* pe fișierul *startinst.bat* și oferă un mediu complet pentru rularea aplicațiilor *Java 2 Enterprise Edition (J2EE)* conținând, printre altele, și *servlet-ul Forms Listener*. *OC4J* rulează într-o fereastră *DOS*, care poate fi minimizată, dar nu închisă atâtă timp cât vrem să rulăm modulele create. Pentru oprirea instanței *OC4J* se rulează fișierul de comenzi *stopinst.bat*.

Ne întoarcem acum la formularul creat pentru gestionarea clienților firmei. Să startăm, deci, instanța *OC4J* și să apăsăm butonul *Run Form*, pe care îl căutam în bara de instrumente a *Form Builder*-ului și îl recunoaștem după semaforul verde cu care este etichetat. Rezultatul obținut este afișat în fereastra unui *browser*:



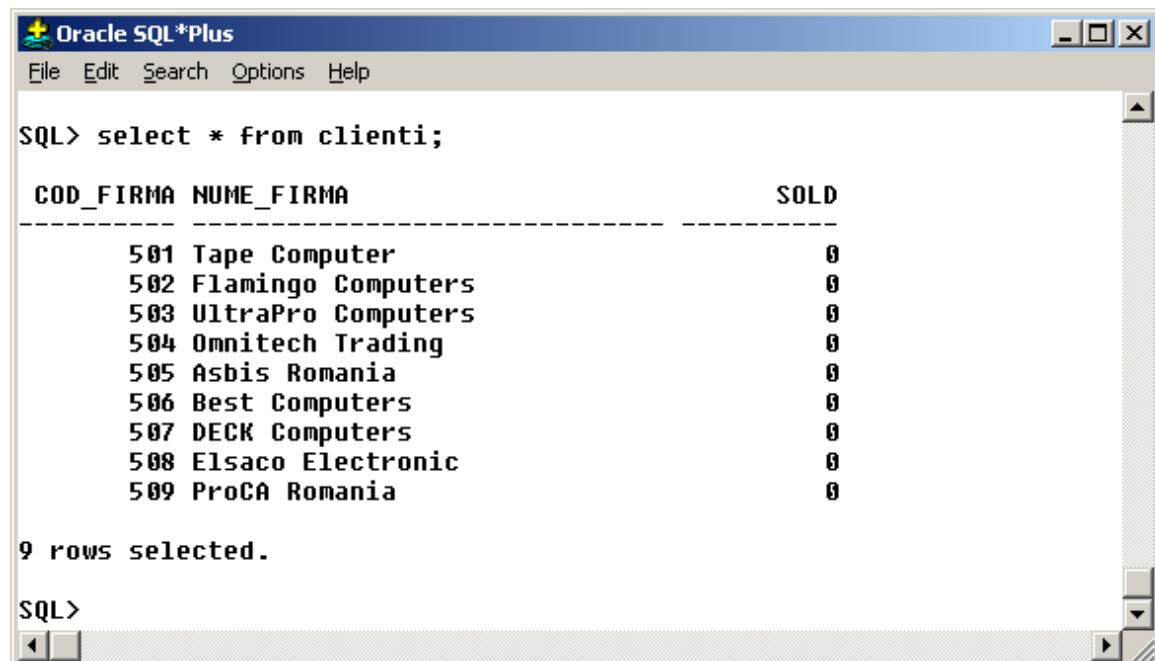
Comportamentul implicit al lui *Oracle Forms* constă în două moduri de funcționare:

- modul *insert* (inserare);
- modul *query* (interrogare).

Modul *insert*

La deschiderea unei forme, aceasta se găsește în modul de lucru *insert*, care permite operații *DML* pe tabelă. În bara de stare a *applet*-ului *Java* este semnalat modul în care lucrăm. În cazul modului *insert*, mesajul *Record: 1/1* semnifică faptul că suntem poziționați pe prima înregistrare, din grupul de 1 care sunt încărcate în *applet*.

Pentru exemplificare, verificăm ce clienți are, deocamdată, firma. Să lucrăm într-un mediu presupus cunoscut: deschidem o sesiune *SQL*Plus* și afișăm toate înregistrările tabelei *clienti*:



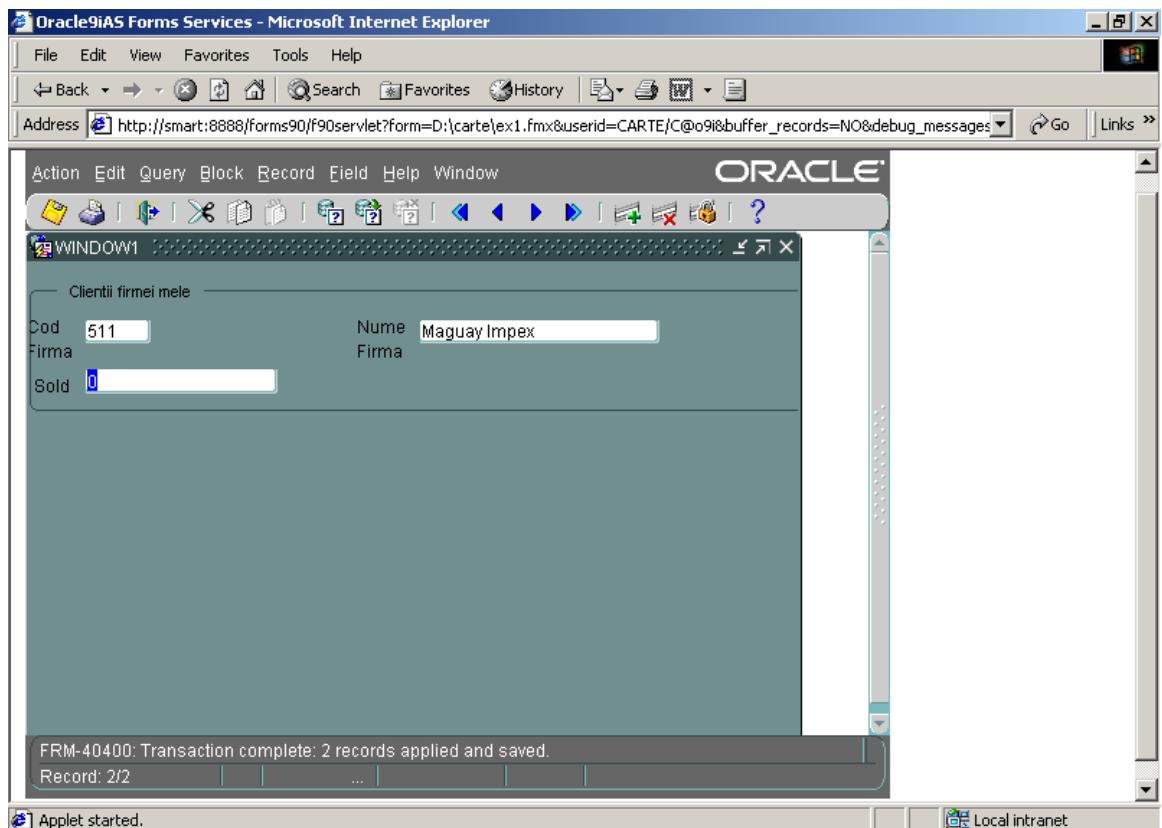
The screenshot shows the Oracle SQL*Plus interface. The title bar reads "Oracle SQL*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The command line displays the SQL command: "SQL> select * from clienti;". The output window shows the following data:

COD_FIRMA	NUME_FIRMA	SOLD
501	Tape Computer	0
502	Flamingo Computers	0
503	UltraPro Computers	0
504	Omnitech Trading	0
505	Asbis Romania	0
506	Best Computers	0
507	DECK Computers	0
508	Elsaco Electronic	0
509	ProCA Romania	0

Below the table, the message "9 rows selected." is displayed. The bottom of the window shows the prompt "SQL>".

Să presupunem că dorim adăugarea unui nou client al firmei, al cărui cod este 510, iar nume - RDC Computers. Soldul inițial este 0. Completăm aceste date în câmpurile corespunzătoare din formă.

Simpla completare a câmpurilor nu va duce la salvarea datelor în tabelă (și ne putem convinge de acest lucru interogând, în *SQL*Plus*, tabela *clienti*). În cazul în care dorim editarea la acest moment a datelor mai multor clienți, apăsăm tasta săgeată în jos și observăm, în bara de stare, că s-a trecut la o nouă înregistrare, vidă, care se așteaptă a fi completată. Practic, a fost creată o înregistrare nouă, vidă. O completăm și salvăm rezultatul muncii noastre (prin *click* pe *Action* → *Save* din meniul implicit al produsului *Developer Forms*), acțiune care este echivalentă cu binecunoscutul *commit*. Există și un echivalent al instrucțiunii *rollback*, invocat prin *Action* → *Clear All* (sunt anulate toate modificările facute de la ultima instrucțiune *commit*).



Observați bara de stare, în care avem confirmarea salvării celor două înregistrări. Dacă nu avem, deocamdată, destulă încredere în abilitățile noastre, putem verifica printr-o comandă *SQL* faptul că cei doi clienți se află în tabela *clienti*:

The screenshot shows the Oracle SQL*Plus interface. The title bar reads "Oracle SQL*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays the following SQL command and its results:

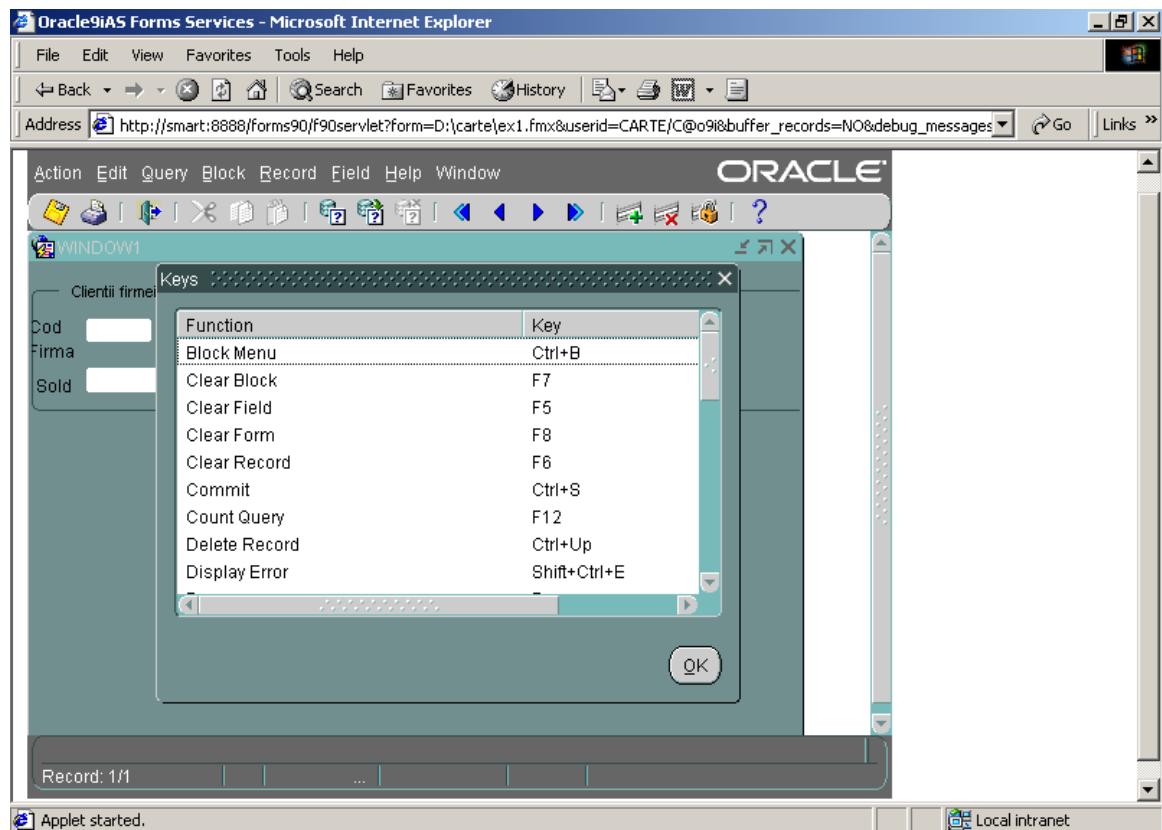
```
SQL> select * from clienti;
```

COD_FIRMA	NUME_FIRMA	SOLD
501	Tape Computer	0
502	Flamingo Computers	0
503	UltraPro Computers	0
504	Omnitech Trading	0
505	Asbis Romania	0
506	Best Computers	0
507	DECK Computers	0
508	Elsaco Electronic	0
509	ProCA Romania	0
510	RDC Computers	0
511	Maguay Impex	0

```
11 rows selected.
```

```
SQL>
```

După cum ați sesizat deja, în formă putem lucra cu *mouse*-ul sau cu tastatura. Fiecare buton are asociată câte o cheie funcțională. La crearea unei aplicații profesionale, trebuie găsit modul optim în care toate aceste facilități să fie comunicate *end-user*-ului. Corespondențele dintre butoane și taste se gasesc la *Help → Keys* (sau Ctrl+K):



Navigarea între obiectele din formă se poate face în mai multe moduri:

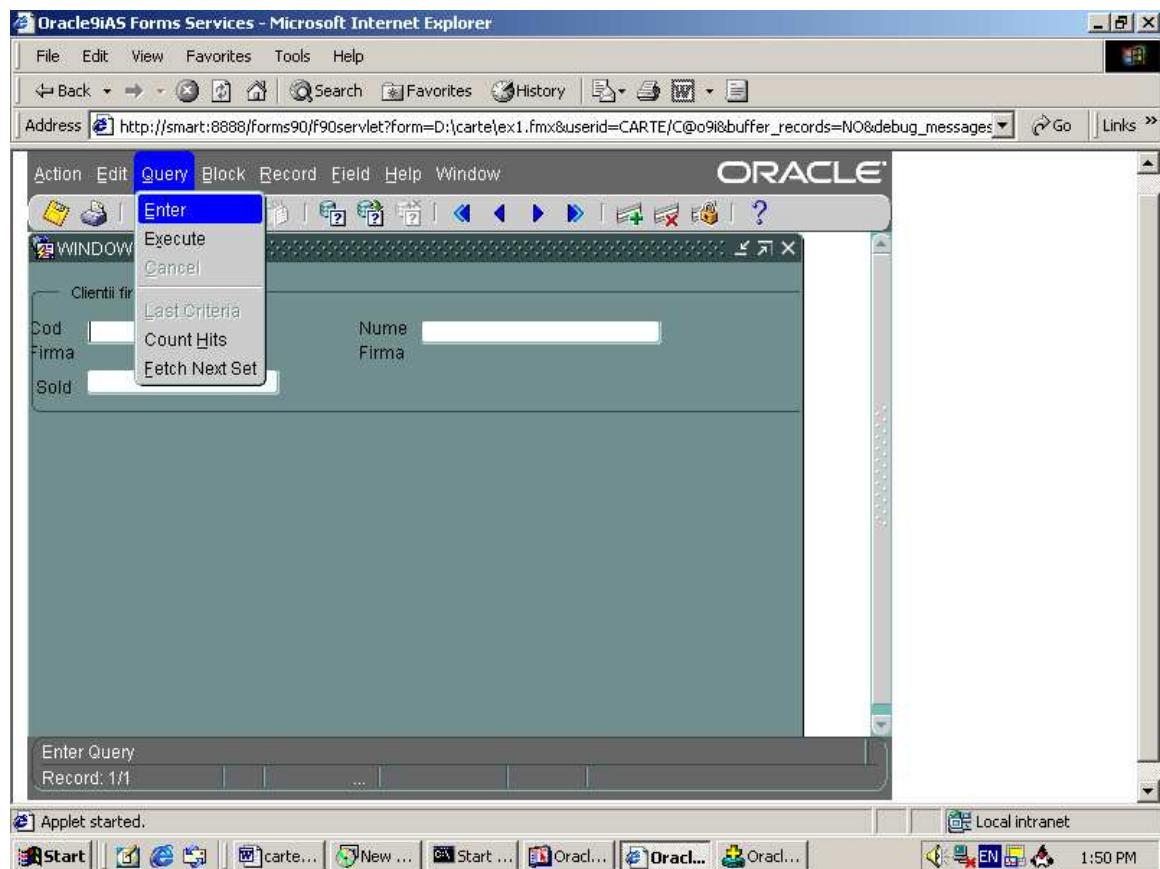
- utilizând tasta *Tab*;
- folosind opțiunile *Field → Previous* sau *Next* din meniul implicit *Forms*;
- dând *click* pe câmpul pe care dorim să îl edităm;

Încercarea de a părasi câmpul *cod_firma* fără ca acesta să aibă un conținut va eșua, fapt care intră în comportamentul implicit al formei; acest câmp mapează coloana *cod_firma* a tabelei, care este cheie primară și, implicit, se presupune că nu trebuie lăsată vidă. Cum se poate修改 acest mod de lucru vom vedea ceva mai târziu.

Ordinea câmpurilor în *Object Navigator* este foarte importantă; în această ordine vor fi accesate în formă la navigarea cu tasta *Tab* (sau cu *Field → Next*).

Modul *query*

Trecerea din modul *insert* în modul *query* se face prin opțiunea din meniu implicit *Query* → *Enter*:



Utilizatorul aduce în formular o copie a înregistrărilor asupra cărora vrea să lucreze. Modul *Query* permite definirea unui criteriu (filtru) pe baza căruia sistemul va crea fraza *select* executată în momentul populării blocului de date. Eventualele modificări ale înregistrărilor nu vor fi operate în baza de date decât la comanda *Action* → *Save*.

Există două feluri de cereri:

- nerestricționate (sau globale), prin care utilizatorul aduce în formă copii ale tuturor rândurilor tabelei pe care se bazează blocul de date asupra căruia se efectuează cererea;

- restricționate, acestea fiind echivalentul selectării unei submulțimi a mulțimii rândurilor tabelei, conform unui filtru indicat de utilizator.

Pentru realizarea unei cereri nerestricționate, vom efectua una din acțiunile:

- selectăm *Query* → *Execute*;
- click pe butonul *Execute Query* din meniul *smartbar*;
- apăsăm combinația de taste Ctrl+F11.

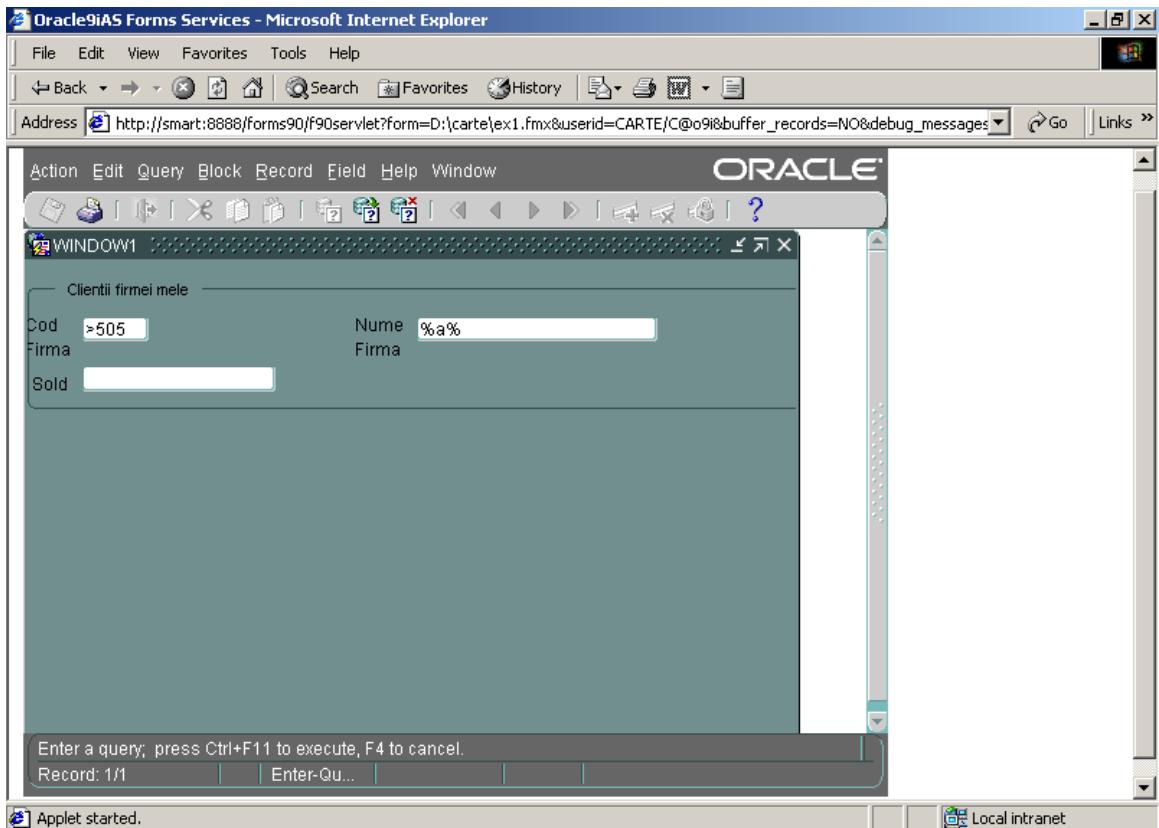
Oricare din aceste acțiuni are ca efect popularea blocului de date cu toate înregistrările din tabela de bază.

Să încercăm acum efectuarea unei cereri restricționate: comutăm în modul *query* (*Query* → *Enter*) și introducem, în câmpul *cod_firma*, criteriul: >505. După cum v-ați dat seama, click pe *Query* → *Execute* va avea ca rezultat generarea unei cereri *SQL* cu o clauză *where* creată conform filtrului introdus de utilizator, în cazul nostru încărcarea în formular a copiilor înregistrărilor pentru care *cod_firma* este mai mare decât 505.

Observați cum, în bara de stare, este semnalat faptul că formularul se află în modul *Enter-Query*. În acest moment, blocul pe care am fost surprinși nu poate fi părăsit, singurele acțiuni posibile fiind:

- introducerea unui filtru și executarea cererii (cu Ctrl+F11 sau *Query* → *Execute*);
- abandonarea modului *query* prin apăsarea tastei F4 sau *Query* → *Cancel*.

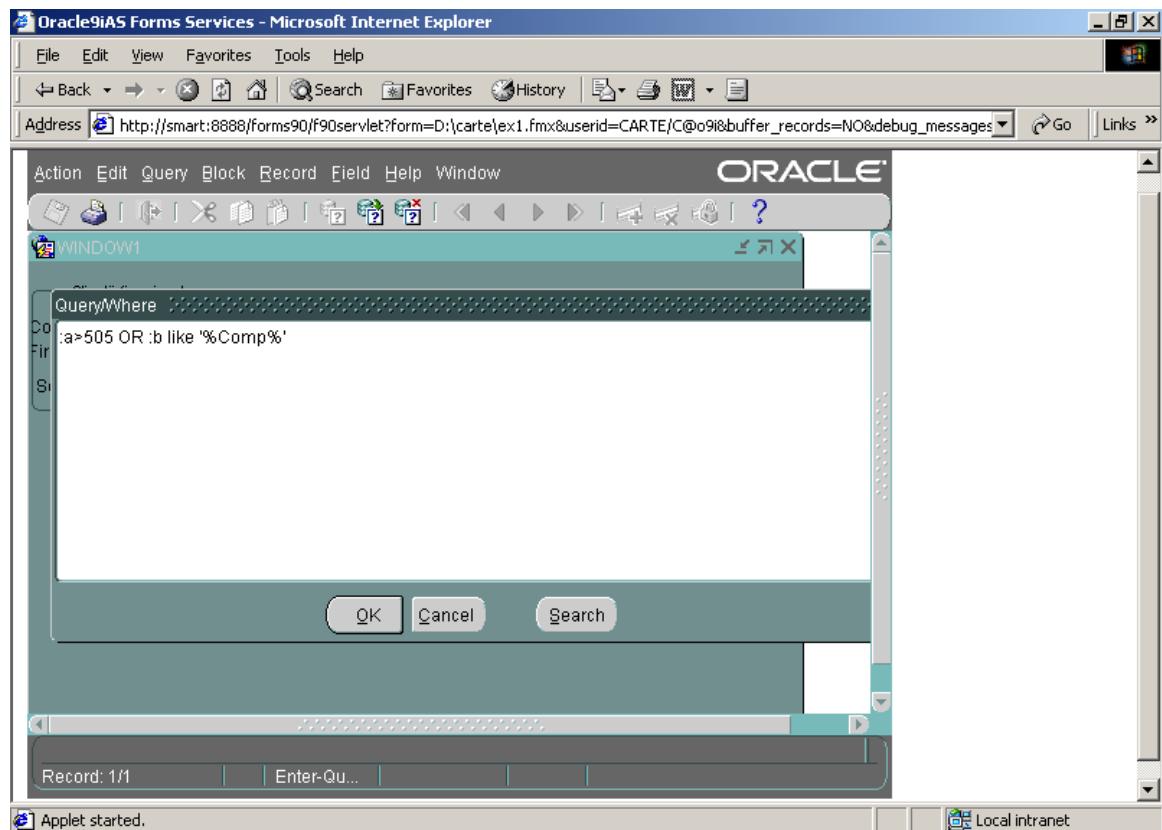
Se pot introduce filtre complexe, pe mai multe câmpuri simultan. Următorul filtru:



va afișa toate firmele al căror cod este mai mare decât 505 și care au litera "a" în numele firmei.

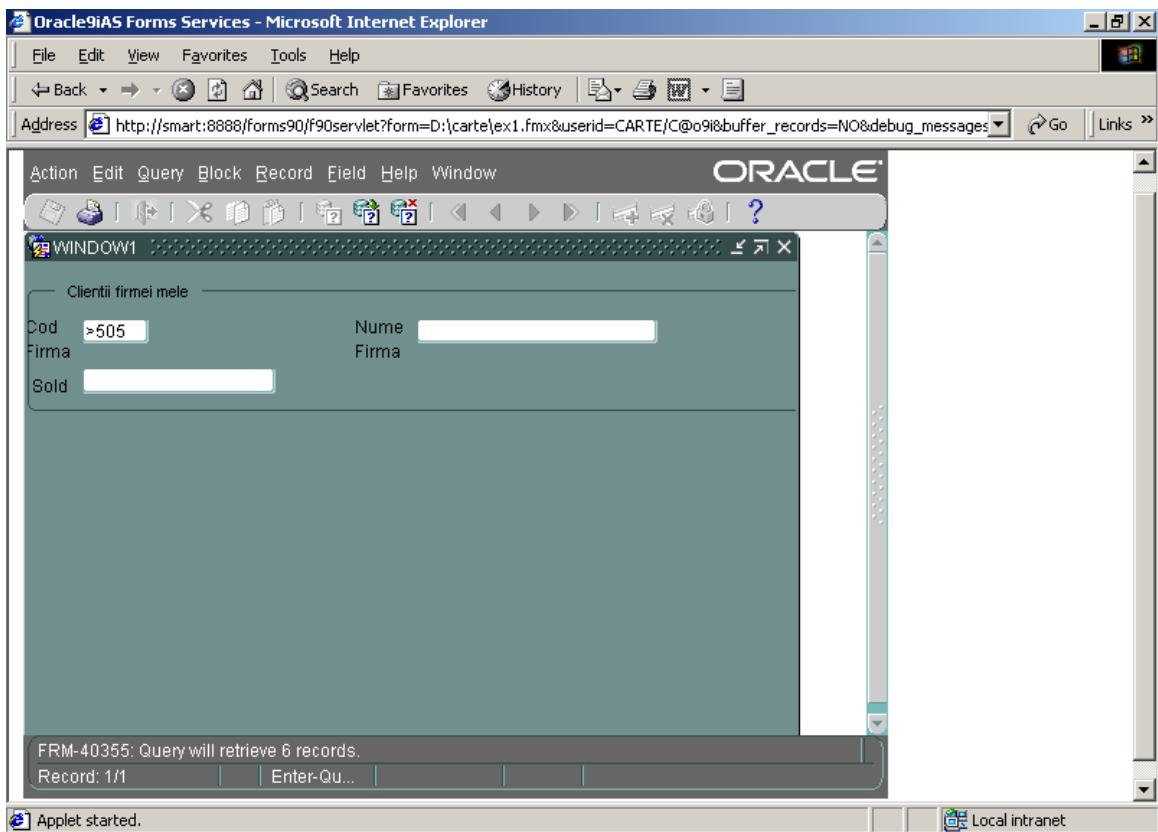
Să subliniem, deci, că introducerea de filtre pe mai multe *item-uri* se transpune într-o frază *select* cu o clauză *where* în care între aceste condiții este pus operatorul *AND*. Dacă vrem o clauză *where* cu disjuncție între condiții, procedăm în felul următor: să presupunem că vrem să aducem în formă copii ale înregistrărilor în care *cod_firma* este mai mare decât 505 SAU ale celor care conțin sirul 'Comp' în denumire. Pentru aceasta invocăm modul *Enter Query* și completăm în câmpul *cod_firma* valoarea :*a*, iar în câmpul *nume_firma* valoarea :*b*. Întrebarea imediată este: cine sunt *a* și *b*, prefixati de :? Acestea reprezintă variabile de legătură (*bind variable*), semnul : semnificând faptul că acestea nu sunt interne mediului curent.

Selectarea opțiunii *Query* → *Execute* va determina afișarea unei ferestre numite *Query/Where*, unde se așteaptă introducerea clauzei *where*, după toate regulile din *SQL*:



După stabilirea clauzei *where* ca în figura anterioară, click pe *OK* va determina aducerea în formă a tuturor firmelor al căror cod este mai mare decât 505 SAU al căror nume conține sirul 'Comp'. Comutăm între înregistrări cu ajutorul tastelor-săgeți sus și jos sau din meniul implicit *Forms, Record* → *Next* sau *Previous*. În meniul *smartbar* există, de asemenea, butoane corespunzătoare acestor funcții. Le puteți descoperi ușor, deoarece au atașate imagini sugestive și, mai mult decât atât, trecerea cu *mouse*-ul peste fiecare determină apariția unui text informativ relativ la butonul curent.

După introducerea filtrelor există posibilitatea aflării (înaintea aducerii de copii în formă) numărului de înregistrări ce verifică criteriul, folosind opțiunea din meniul implicit *Query* → *Count Hints*:



Observați mesajul din bara de stare: cererea va găsi 6 înregistrări (ce îndeplinesc criteriul cod_firma>505).

Sintetizând, obținem următoarele asemănări și diferențe între cele două moduri implicite de lucru în formă:

Acțiuni	Modul <i>Insert</i>	Modul <i>Query</i>
cereri restricționate (cu clauza <i>where</i>)	<input checked="" type="checkbox"/>	✓
cereri nerestricționate (fără clauza <i>where</i>)	✓	✓
casetă de dialog <i>Query/Where</i>	<input checked="" type="checkbox"/>	✓
<i>insert/update/delete</i>	✓	<input checked="" type="checkbox"/>
navigarea în afara blocului curent	✓	<input checked="" type="checkbox"/>
<i>shutdown-ul</i> procesului <i>runtime</i>	✓	<input checked="" type="checkbox"/>

Criteriile după care vor fi formulate cererile restricționate sunt următoarele:

- completarea (în modul *Query*) a unui *item* cu o valoare și executarea cererii va implica aducerea în formă a înregistrărilor pentru care câmpul corespunzător *item*-ului are valoarea specificată;
- completarea unui *item* cu un sir conținând % sau _ va duce la formularea unei cereri cu o clauză *where* conținând operatorul *like*;
- folosiți semnul # în fața operatorilor *SQL*; exemplu: #between 501 and 505;
- utilizați formatul implicit al datei: DD-MON-YY;
- în caseta de dialog *Query/Where* sirurile de caractere și datele calendaristice vor fi incluse între apostrofuri.

2.4.5. Rafinarea formelor

După crearea modulului (folosind cele două *wizard*-uri prezentate anterior), utilizatorul va dori să îl personalizeze sau să îl modifice, lucru ce poate fi făcut în unul sau mai multe moduri:

1) rafinarea blocului de date se poate face:

- invocând *Data Block Wizard* în modul *reentrant*;
- efectuând modificări manual, de exemplu adăugarea sau ștergerea de *item*-uri;
- modificând proprietățile obiectelor, folosind paleta de proprietăți;

2) rafinarea machetei modulului se poate face:

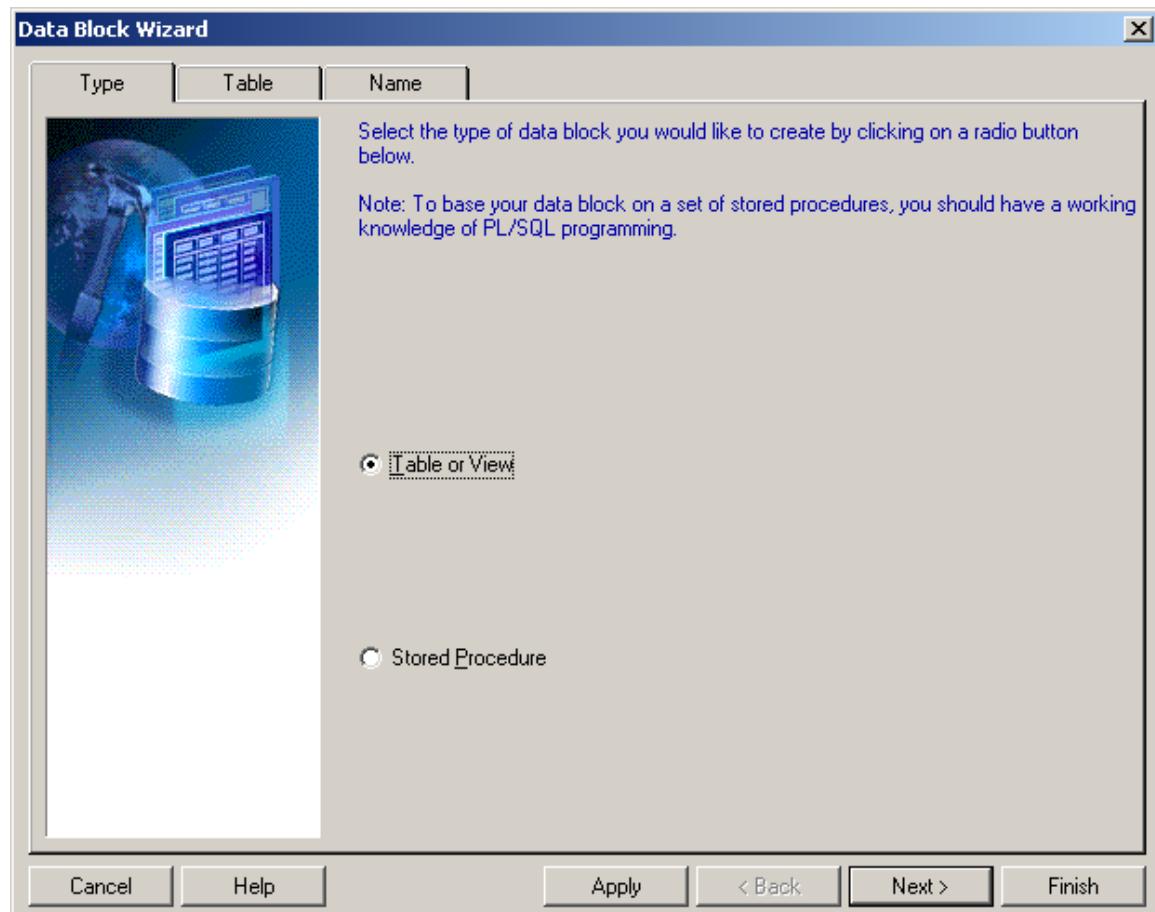
- invocând *Layout Wizard* în modul *reentrant*;
- invocând *Layout Editor* (prin selectarea *Tools* → *Layout Editor* sau click drept pe un obiect în *Object Navigator* → *Layout Editor*) și efectuând modificări manual;
- modificând proprietățile cadrului în paleta sa de proprietăți.

O caracteristică foarte utilă a celor două *wizard*-uri prezentate anterior este capacitatea de a putea fi apelate în modul *reentrant* (chiar și pentru blocurile sau *frame*-urile care nu au fost create cu *Data Block Wizard*, respectiv *Layout Wizard*), mod în care toți pașii parcursi sunt prezenți într-o singură fereastră de tip *tab*.

Pentru invocarea utilitarului *Data Block Wizard* în modul *reentrant*:

- selectăm un obiect din blocul de date sau *frame*-ul în *Object Navigator* sau în *Layout Editor* (observați că cele două ferestre sunt sincronizate – la selectarea unui obiect în oricare dintre ele, același obiect va fi automat selectat în cealaltă fereastră);
- apelăm instrumentul *Data Block Wizard* în oricare dintre modurile învățate până acum:
 - selectând *Tools* → *Data Block Wizard* din meniul *Form Builder*;
 - *click* dreapta și selectând *Data Block Wizard* din meniul de context ce apare;
 - *click* pe butonul *Data Block Wizard* din meniul numit *main toolbar* al *Builder*-ului;

Rezultatul oricărei dintre aceste acțiuni va fi:

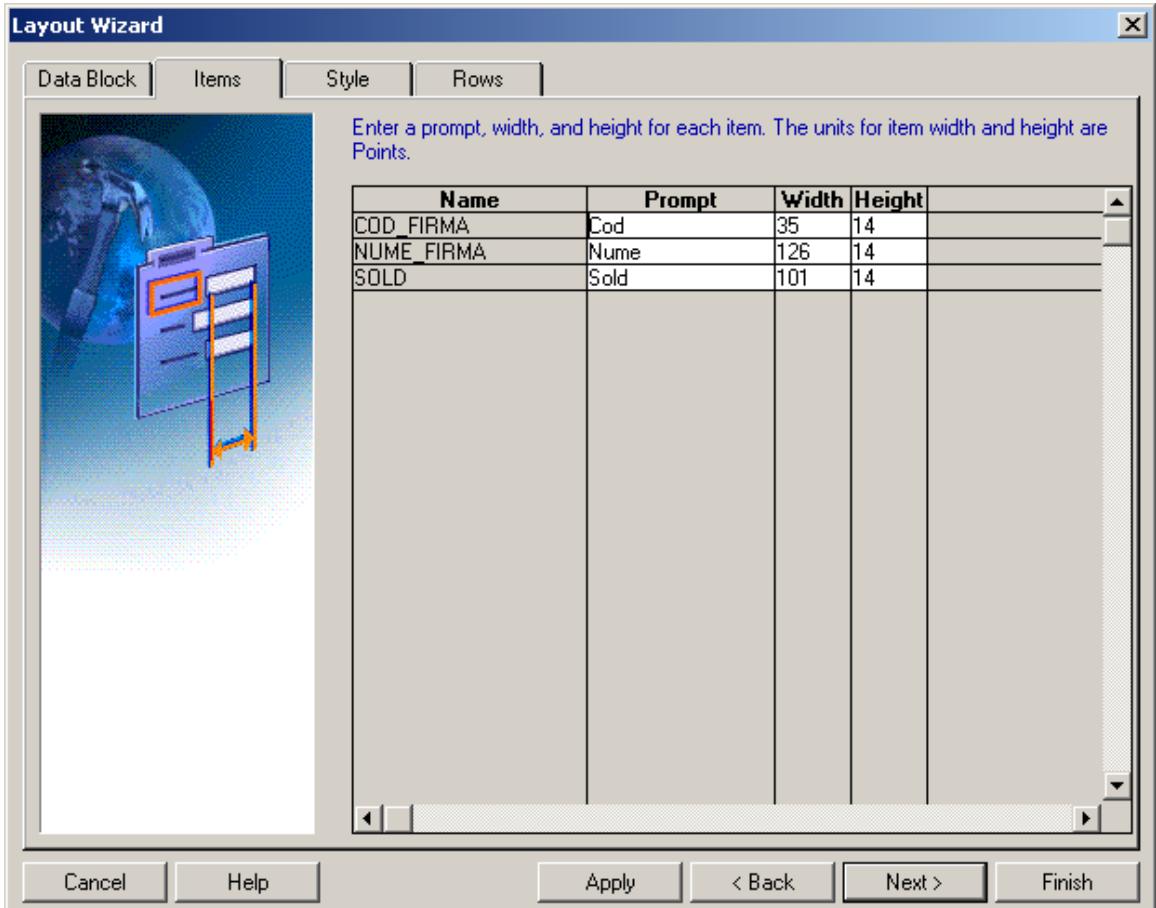


Observați în fereastra care a apărut cele trei pagini *Tab* corespunzătoare celor trei pași parcurși de utilitarul *Data Block Wizard*.

Invocarea instrumentului *Layout Wizard* în modul *reentrant* se face:

- selectând cadrul (*frame*) în *Object Navigator* sau *Layout Editor*;
- lansând *Layout Wizard*, în oricare dintre modurile:
 - *click* pe *Tools* → *Layout Wizard*;
 - *click* dreapta pe cadru și selectând *Layout Wizard*;
 - *click* pe butonul *Layout Wizard* din meniul *Builder*-ului.

Rezultatul va fi, ca și în cazul precedent, o fereastră *tab-type* având pe fiecare pagină *tab* câte unul din pașii parcurși de *wizard*:

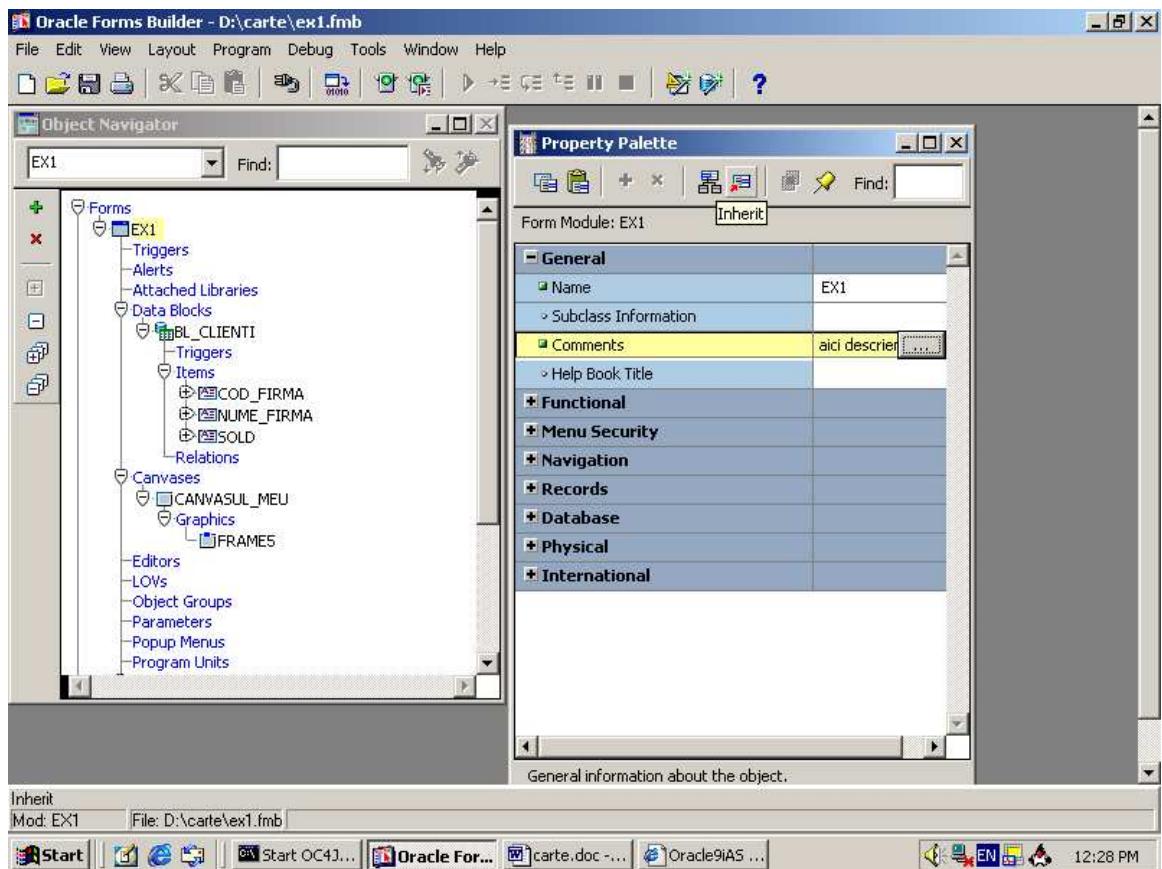


Modificăm, pe pagina a doua, eticheta *item-urilor*, apoi acționăm butonul *Finish* (care determină o acțiune prealabilă implicită *Apply*). Rulați aplicația și observați modificările făcute.

2.4.6. Palete de proprietăți

O altă modalitate de a rafina modulele create este modificarea comportamentului sau a modului de afișare a obiectelor folosind paletele lor de proprietăți.

Primul pas în dezvoltarea unei aplicații *Forms* constă în crearea unui modul. Acesta are proprietăți predefinite, care pot fi, însă, modificate folosind paleta de proprietăți a modulului. Selectăm modulul în *Object Navigator* și din meniul de context ce apare la *click* dreptă selectăm *Property Palette* (același lucru se poate obține apăsând tasta *F4* atunci când modulul este selectat în *Object Navigator*):



Se pot obține informații despre fiecare din proprietățile ce se pot seta la acest nivel selectând proprietatea și apăsând tasta *F1*. Proprietățile care au încă valoarea implicită sunt prefixate, în paleta de proprietăți, de un disc, în timp ce acele care au fost modificate sunt marcate cu un pătrat verde. O proprietate ce a fost modificată poate fi adusă la valoarea implicită prin apăsarea butonului *Inherit* din bara de meniu a paletelor de proprietăți. Toate modulele vor avea aceleași intrări în paleta de proprietăți. Capitolul **General** oferă informații generale legate de modul. Intrarea *Name* permite modificarea numelui modulului (lucru ce poate fi realizat și în *Object Navigator*, dând de două ori *click* pe numele modulului și editând noul nume), iar *Comments* este folosit pentru adăugarea de comentarii utile pentru dezvoltatorii de aplicații, ușurând astfel munca de întreținere și depanare.

La capitolul **Functional** se poate seta fereastra în care va apărea modulul (*Console Window*), precum și meniul care va fi asociat acestuia. Implicit, modulul va fi afișat cu cele două meniuri precizate anterior, aşa că la intrarea *Menu Module* vom găsi

DEFAULT&SMARTBAR. Ștergerea oricărui dintre cele două nume va determina dispariția meniu respectiv la rularea aplicației. Putem defini, de asemenea, meniuri-utilizator, iar aici este locul în care le asociem modulului curent. Capitolul **Menu Security** conține intrarea *Menu Role*. Aici se specifică rolul care îi este acordat utilizatorului la rularea meniului.

Capitolul **Navigation** conține două proprietăți:

- *Mouse Navigation Limit* – în care se precizează cât de departe poate naviga *end-user*-ul cu *mouse*-ul; implicit, proprietatea este setată la *Form*, însemnând că este permisă navigarea în orice *item* din formă; alte valori posibile sunt:
 - *block* – permite navigarea cu *mouse*-ul doar în interiorul *item*-urilor ce aparțin blocului curent;
 - *record* – utilizatorul se poate deplasa cu *mouse*-ul numai în *item*-urile din înregistrarea curentă;
 - *item* – previne navigarea cu *mouse*-ul în afara *item*-ului curent (valoarea este aplicată în cazul în care se vrea interzicerea folosirii *mouse*-ului în forma curentă).
- *First Navigation Data Block* – specifică numele blocului cu care *Forms Developer*-ul începe navigarea la încărcarea formularului sau după o operație *CLEAR_FORM* (de aducere la valoarea implicită a tuturor obiectelor din formă).

Capitolul **Database** definește modul de interacțiune a modulului cu baza de date, prin definirea proprietăților:

- *Validation Unit* – determină domeniul de validare al formularului în *runtime*; pentru cele mai multe din aplicații, valoarea *default* este setată la *item*, semnificând faptul că validarea datelor introduse de utilizator se face la nivel de *item*, adică odată cu încercarea de părăsire a respectivului obiect;
- *Interaction Mode* – determină modul de interacțiune a utilizatorului cu formularul, în cazul modului *Enter-Query*; implicit, această proprietate este setată la *Blocking*, ceea ce previne orice posibilitate de interacțiune a utilizatorului cu formularul atât timp cât nu au fost încărcate toate înregistrările din baza de date. Dacă valoarea este setată la *Non-blocking*, *end-user*-ul poate interacționa cu forma în timpul în care înregistrările sunt încărcate. Această valoare este folosită în special când se emit cereri consumatoare de timp și se vrea ca utilizatorul să poată întrerupe executarea cererii.

- *Maximum Query Time* – permite specificarea timpului după care orice cerere va fi întreruptă, dacă execuția ei nu a fost terminată. Valoarea implicită este 0 care semnifică, de fapt, faptul că cererea nu va fi întreruptă automat (vom vedea că în *Oracle Forms Developer* valoarea 0 are, în multe din cazuri, altă semnificație față de cea atribuită ei de matematicieni cu mai bine de două milenii în urmă). Orice valoare nenulă este luată în considerare doar dacă proprietățile *Query Allowed* și *Query All Records* sunt setate la Yes (aveți răbdare, despre ele veți afla ceva mai tarziu!).
- *Maximum Records Fetched* – specifică numărul de înregistrări ce vor fi încărcate în formular înainte de abandonarea execuției cererii. Comentariile referitoare la valoarea implicită 0 sunt valabile și în acest caz.
- *Isolation Mode* – permite definirea modului de rezolvare a concurenței tranzacțiilor; proprietatea se setează pe *Serializable* în cazul în care câțiva utilizatori lansează un număr mic de tranzacții pe o bază de date de dimensiuni mari, deci atunci când șansa ca două tranzacții concurente să încearcă modificarea aceleiași înregistrări este mică. Dacă acest lucru se întâmplă, unul din *user-i* va primi eroarea cu mesajul *Cannot serialize access*. În cazul în care se lansează tranzacții intensive, este recomandată lăsarea acestei proprietăți la valoarea implicită, *Read Committed*.

Capitolul **Physical** conține informații referitoare la atributele fizice ale obiectelor modulului:

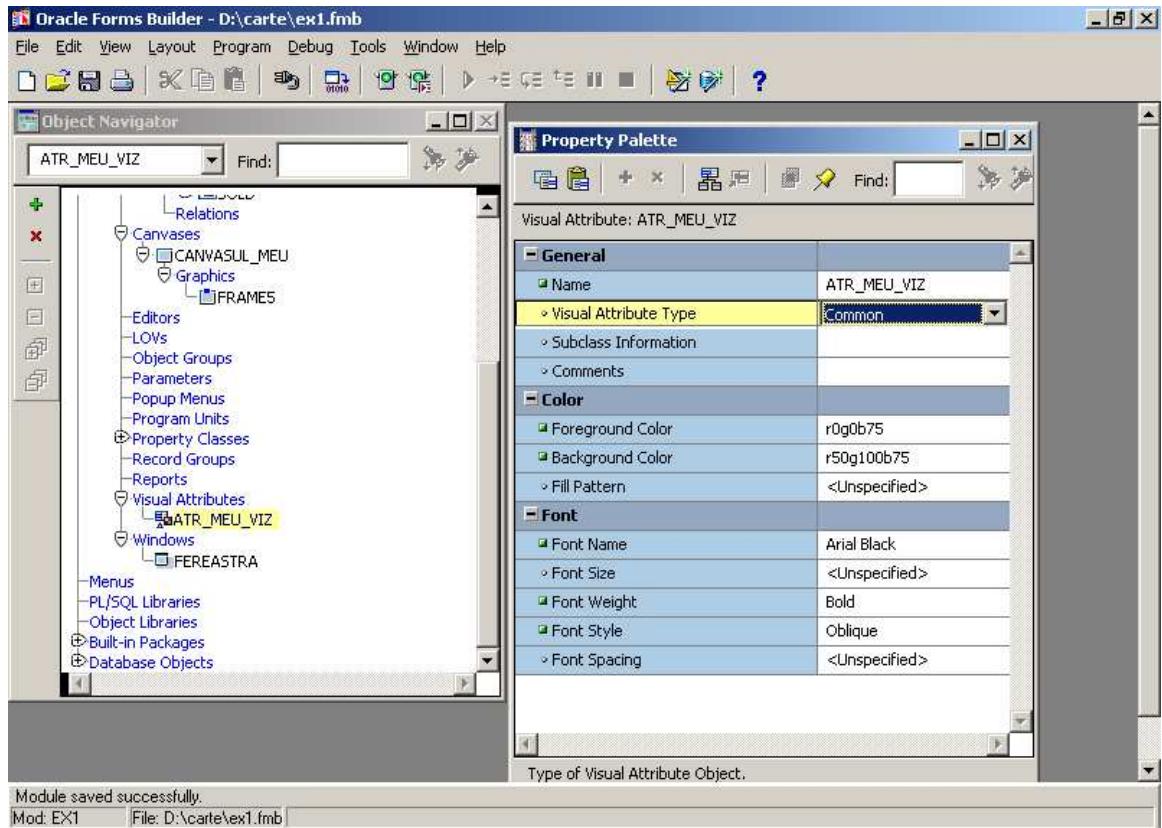
- *Coordinate System* – permite definirea unității de măsură folosite pentru stabilirea dimensiunii de afișare și a poziției obiectelor. Această unitate de măsură se poate exprima în:
 - caractere, mărimea unui caracter fiind dată de setul implicit de caractere sau putând fi specificată de programator;
 - unități de măsură reale, având posibilitatea de a alege între: pixeli, centimetri, puncte etc..
- *Use 3D Controls* – proprietate valabilă doar pe sistemele ce rulează *Microsoft Windows* și a cărei setare la valoarea Yes permite afișarea *item-urilor* cu efecte tri-dimensionale;
- *Form Horizontal Toolbar Canvas* – permite specificarea (prin numele său), pentru *Microsoft Windows*, a unui *canvas* de tip *horizontal toolbar* care să fie afișat ca bară orizontală de meniu;
- *Form Vertical Toolbar Canvas* – proprietate echivalentă cu cea precedentă, folosită pentru cazul meniurilor verticale.

Capitolul ***International*** conține proprietatea *Direction*, a cărei valoare ar trebui lăsată neschimbată deoarece *end-user*-ul nu va fi încântat, probabil, de posibilitatea scrierii de la dreapta la stânga.

Am trecut, în descrierea capitoilelor de proprietăți ale unui modul, peste capitolul ***Records***, care conține o singură intrare, *Current Record Visual Attribute Group*. Această proprietate este specifică atât modulelor, cât și blocurilor de date și *item*-urilor și permite specificarea unui atribut vizual pentru obiectul căruia îi este aplicată.

2.4.7. Atribute vizuale

Un atribut vizual este un obiect pe care îl creăm în *Object Navigator* (selectând nodul *Visual Attributes* și dând *click* pe butonul *New*) și care are proprietăți ce definesc: setul de caractere, culoarea și modul de hașurare. Un atribut vizual aplicat unui obiect va determina moștenirea de către obiect doar a celor proprietăți care sunt setate în atributul vizual, restul rămânând neschimbate.

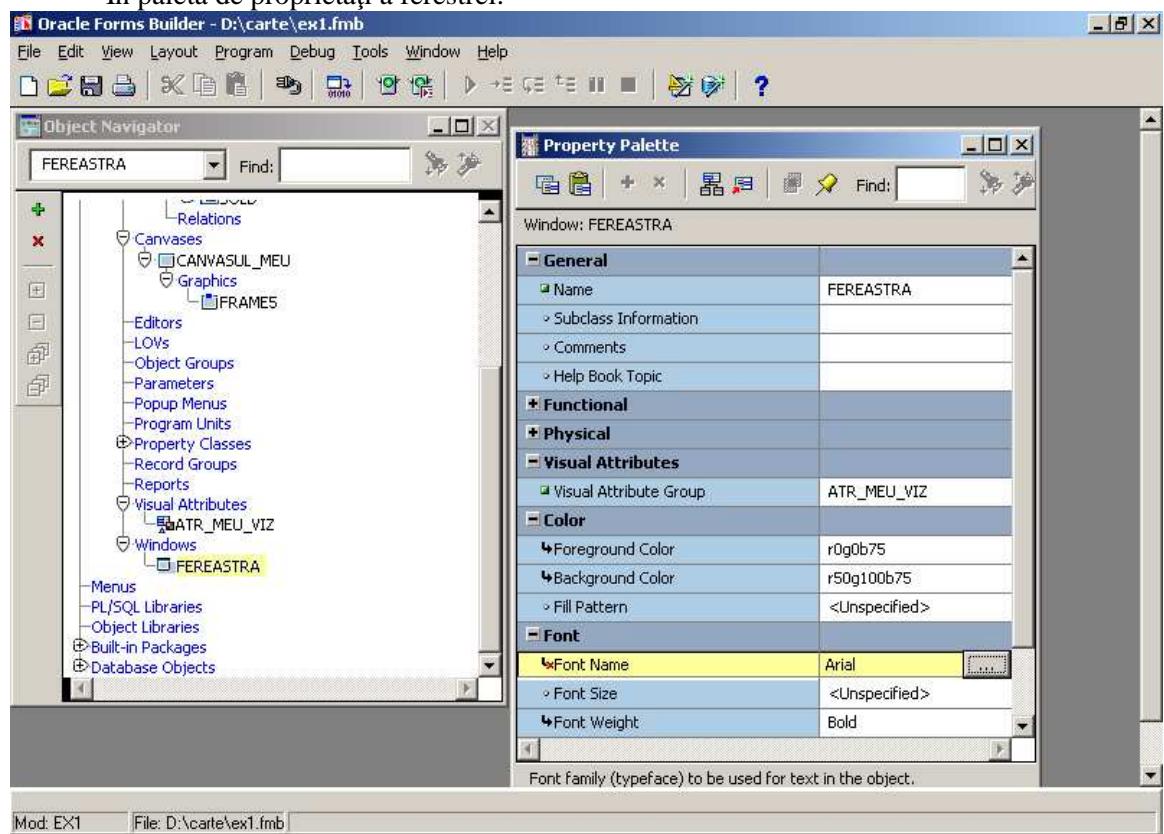


Observați în fereastra *Object Navigator* crearea, la nodul *Visual Attributes*, a obiectului numit *ATR_MEU_VIZ*. Invocarea paletei de proprietăți a acestui obiect are efectul prezentat în fereastra din dreapta. Numele proprietăților legate de font și culoare sunt sugestive și pot fi setate cu usurință. Proprietatea *Visual Attribute Type* stabilește domeniul de aplicare al atributului: *common*, *prompt* sau *title*.

2.4.8. Proprietățile ferestrelor

Scopul creării unui obiect este folosirea (și refolosirea) lui, aşa că vom utiliza atributul vizual creat anterior pentru setarea proprietăților ferestrei în care este încărcat modulul. Apare următoarea întrebare: pentru ce ne trebuie un atribut vizual, prin care se definesc proprietăți cum ar fi culoarea fundalului și a setului de caractere, când pot face acest lucru direct, la nivelul capitolului **Color** al paletei de proprietăți a ferestrei? Unul din avantajele (re)utilizării unui obiect este acela că orice modificare a unei proprietăți a obiectului se va moșteni automat de toate instanțele respectivului obiect.

În paleta de proprietăți a ferestrei:



modificăm proprietatea *Visual Attribute Group*, selectând atributul vizual creat anterior din lista *pop-up* corespunzătoare. Remarcăm modificarea imediată a valorilor proprietăților legate de culoare și font ale ferestrei conform celor moștenite de la atributul vizual și faptul că proprietățile moștenite sunt marcate cu o sageată. Să presupunem că dorim ca fontul cu care apare textul în *item-uri* să fie altul decât cel definit în atributul vizual și moștenit, implicit, de către fereastră. Modificând valoarea fontului observăm că

săgeții care prefixează proprietatea respectivă îi este adăugat un *x* roșu, semnificând faptul că, inițial, proprietatea a fost moștenită, dar am renunțat la moștenire

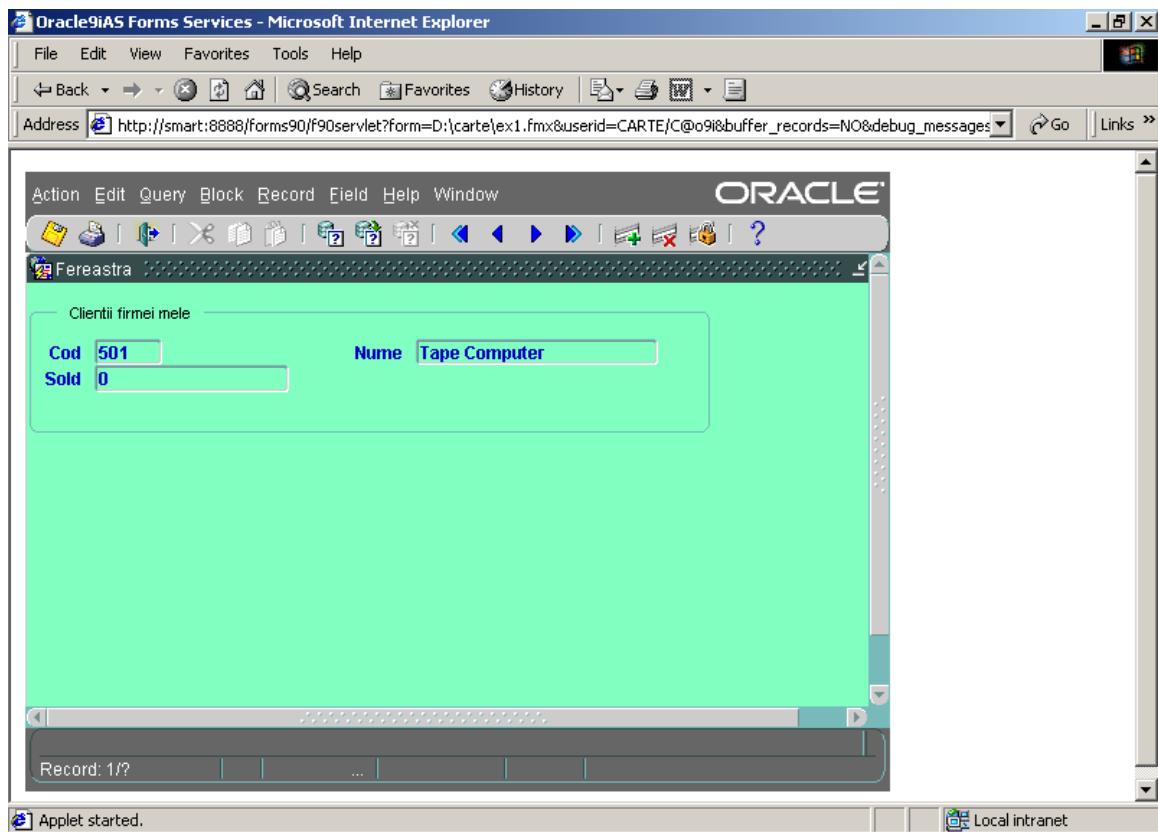
și am suprascris-o. Acționarea în această situație a butonului *Inherit* va determina aducerea valorii la cea moștenită de la obiectul atribut vizual ATR_MEU_VIZ.

Două capituloare importante în paleta de proprietăți a ferestrei merită detaliate: ***Functional*** și ***Physical***. Capitolul *Physical* permite specificarea locului în care va apărea obiectul (în cazul nostru, fereastra) pe ecran, prin stabilirea coordonatelor colțului din stânga-sus (proprietățile *X Position* și *Y Position*) și a dimensiunilor sale (proprietățile *Width* și *Height*). Proprietatea *Bevel* setează tipul chenarului obiectului, iar ultimele două proprietăți comută între afișarea/inhibarea barelor de *scroll* verticale și orizontale pentru obiectul respectiv.

Capitolul ***Functional*** permite definirea modului de funcționare a ferestrei, setând proprietăți cum ar fi: *Close Allowed* (este sau nu permisă comanda *Close window*), *Move Allowed*, *Resize Allowed* etc. Stilul ferestrei este definit de proprietatea *Window Style*, care poate fi setată la una din valorile *Document* sau *Dialog*. Proprietatea *Modal* stabilește tipul ferestrei. O fereastră de tip modal nu poate fi părăsită liber (dând *click* în afara ei), ci numai programatic. Implicit, o fereastră *Modalless*, adică nerestricționată, este o fereastră pe care utilizatorul o poate părăsi liber. Ferestrele de acest tip nu sunt în mod necesar active atunci când sunt afișate.

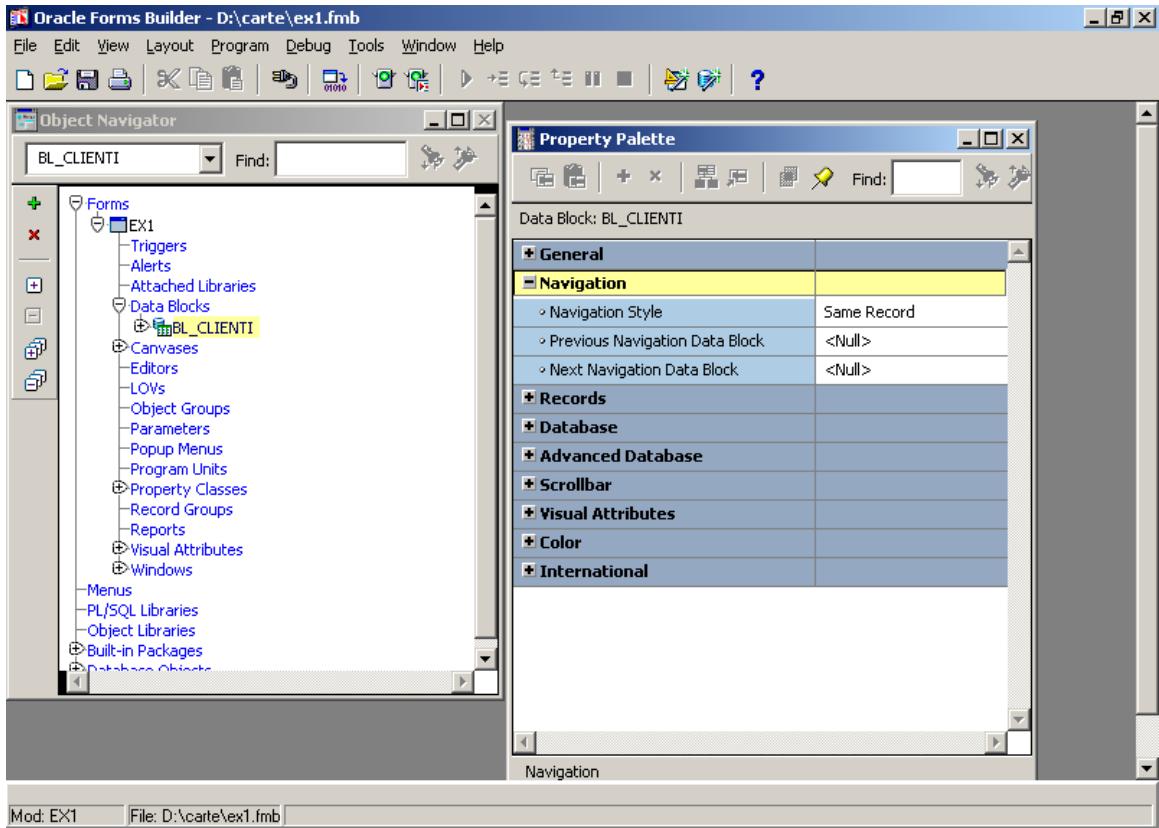
2.4.9. Proprietățile unui bloc de date

Întorcându-ne la aplicația pe care vrem să o creăm (nu am avansat extraordinar de mult, aşa e?), haideți să ne reamintim că ne aflăm la etapa în care creasem, folosind două utilitare (*DataBlock Wizard* și *Layout Wizard*), un modul prin care vrem să gestionăm clienții firmei. Am creat un atribut vizual pe care l-am aplicat ferestrei în care va apărea modulul, iar rezultatul este următorul:



Modulul conține un singur bloc de date, numit *BL_CLIENTI*. Acesta are 3 *itemuri* (elemente), corespunzătoare celor 3 câmpuri din tabela *clienti*. Utilizatorul poate modifica interfața și comportamentul unui bloc setându-i proprietățile.

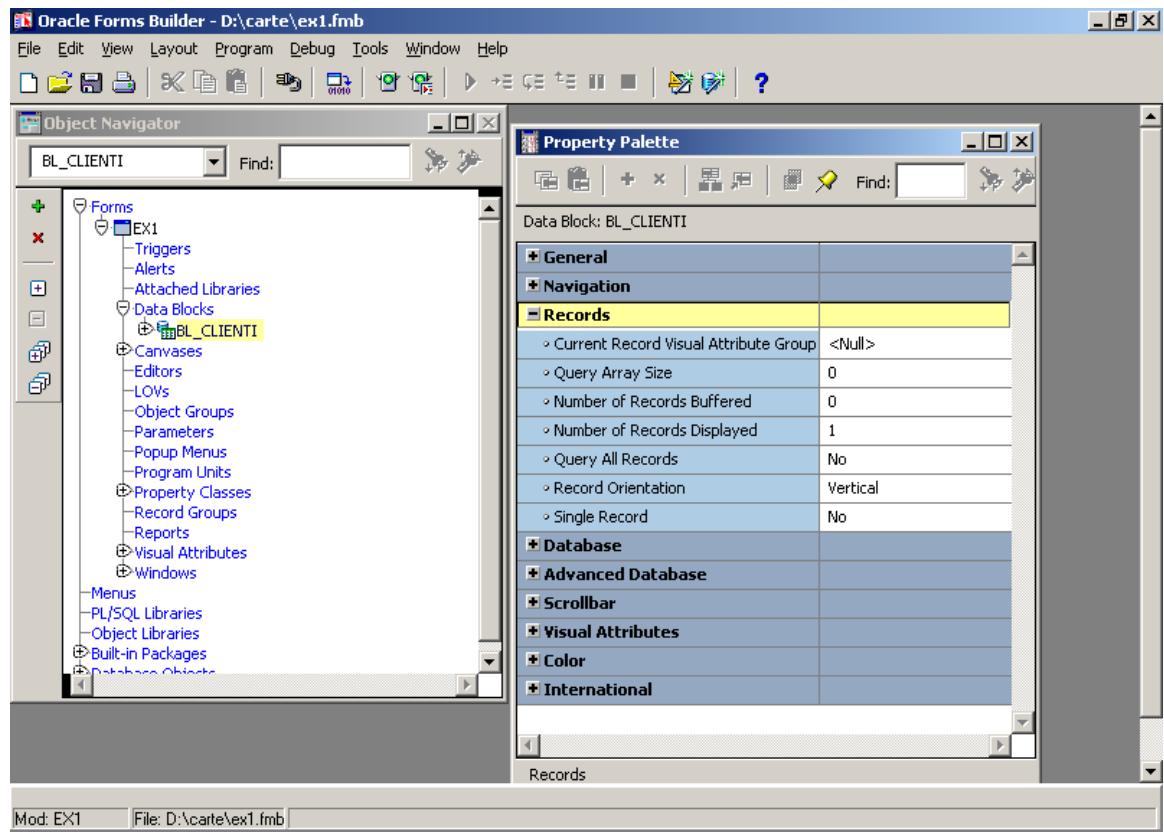
Să analizăm paleta de proprietăți a blocului de date (pentru accesarea acestea folosim una din metodele prezentate într-un capitol anterior, de exemplu: ne poziționăm pe blocul de date în *Object Navigator* și apăsăm tasta *F4*). Primul grup de proprietăți este *General*, cu intrările deja cunoscute. Urmează grupul *Navigation*, care gestionează modul de navigare cu tastatura în interiorul blocului. Implicit, pentru efectuarea unui salt circular între *item-urile* din modul se folosește tasta *Tab* pentru navigarea înainte și combinația *Shift+Tab* pentru navigare inapoi. Proprietățile ce se pot seta din grupul *Navigation* sunt:



- *Navigation Style* – implicit, la navigarea din ultimul *item* al blocului, *Forms* returnează focusul pe primul *item* din bloc. Acest mod de lucru corespunde valorii *Same Record* al acestei proprietăți; alte valori posibile (ce se pot selecta din meniul *pop-up* ce apare la *click* pe această proprietate) sunt:
 - *Change Record* – caz în care cursorul se mută pe primul *item* al următoarei înregistrări la părăsirea ultimului *item* din înregistrarea curentă
 - *Change Data Block* – această valoare determină saltul la blocul următor odată cu părăsirea ultimului *item* din înregistrarea curentă
- *Previous (Next) Navigation Data Block* – valoare setată implicit la *Null*, semnificând faptul că blocul anterior (respectiv următor) blocului curent este cel ce se află în *Object Navigator* înaintea blocului curent, respectiv după acesta. Aici este permisă definirea unui alt bloc la care se va face

saltul cursorului în cazul unei navigări cu tastatura spre blocul anterior (următor).

Următorul grup de proprietăți al unui bloc de date poartă numele *Records* și arată cam așa:



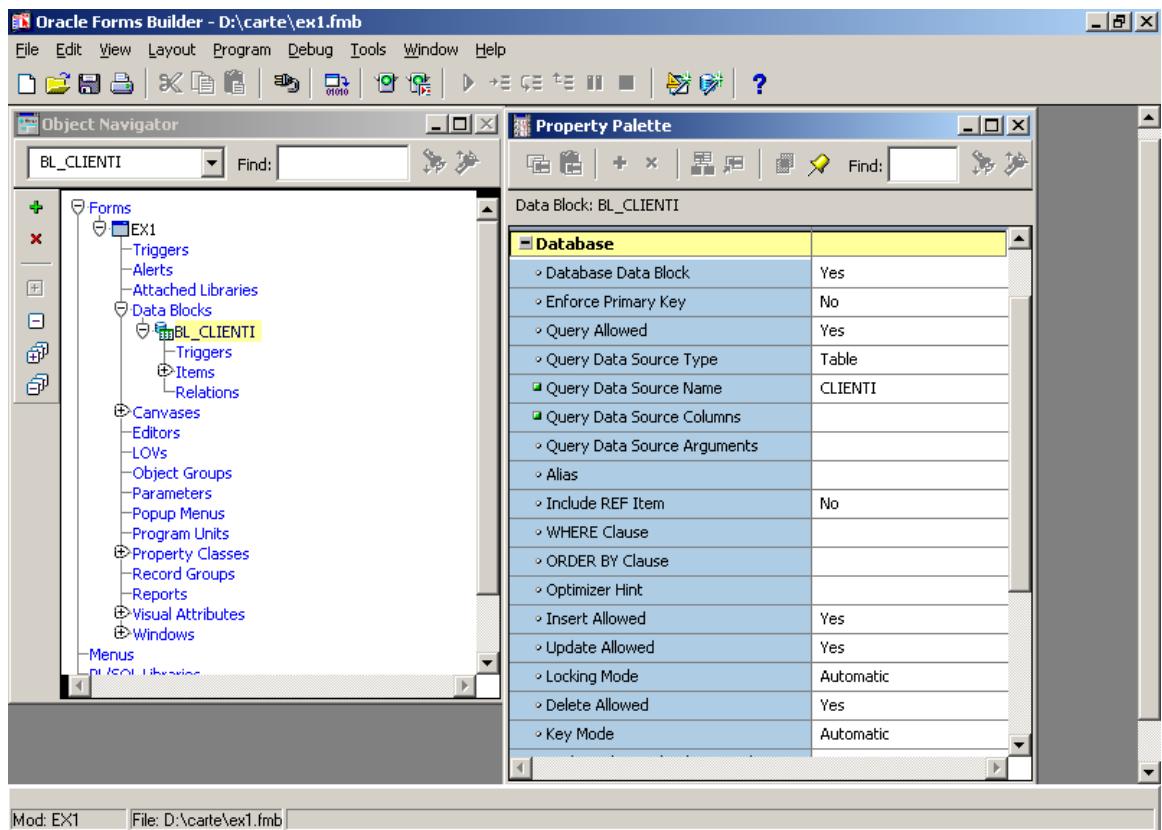
Grupul *Records* controlează modul în care înregistrările din tabelă sunt încărcate de *Forms* în modul și conține următoarele proprietăți:

- *Current Records Visual Attribute Group* – permite specificarea unui atribut vizual care va fi folosit ca mască pentru afișarea înregistrărilor
- *Query Array Size* – specifică numărul maxim de înregistrări pe care *Forms*-ul le încarcă la o accesare a bazei de date. O valoare mică a acestei proprietăți va implica un timp de răspuns per accesare a bazei de date mic; o valoare mare implică un număr mai mic de accesări a bazei date și, prin urmare, un trafic în rețea redus și un timp total de procesare mai mic. Valoarea implicită

a acestei proprietăți este 0, semnificând faptul că la o accesare a bazei de date numărul de înregistrări încărcate în formă este egal cu valoarea proprietății *Number of Records Displayed*

- *Number of Records Buffered* – este o proprietate ce se folosește pentru a mări viteza de lucru. *Buffer-ul* este o zonă de memorie în care *Forms Builder-ul* își aduce înregistrările care sunt în aşteptare. Valoarea implicită 0 înseamnă, de fapt, numărul de înregistrări afișate (setat prin proprietatea *Number of Records Displayed*) plus 3. Înregistrările adiționale sunt salvate de *Forms* pe *hard-disk*, într-un fișier temporar. O valoare mare a acestei proprietăți îmbunătățește viteza de lucru, dar folosește mai mult spațiu de stocare.
- *Number of Records Displayed* – această proprietate setează numărul de înregistrări pe care un bloc le afișează în *canvas* la un moment dat. Setați această proprietate la o valoare mai mare, apoi rulați forma și observați rezultatul! (Asigurați-vă de faptul că în *canvas* este loc suficient pentru afișarea simultană a mai multor înregistrări. În caz contrar, poziționați-vă pe un obiect din *Object Navigator* și apăsați tasta *F2* pentru accesarea instrumentului *Layout Editor* (aceasta este doar una din multiplele posibilități în care acesta poate fi invocat); redimensionați *frame-ul*, eventual și *canvas-ul*, astfel încât să poată fi vizualizate toate înregistrările.
- *Query All Records* – dacă această proprietate este setată la valoarea *Yes* vor fi aduse în formă toate înregistrările odată. Implicit, aceasta valoare este setată la *No* pentru îmbunătățirea performanțelor de lucru, dar este esențială modificarea ei la valoarea *Yes* în cazul când în formă se fac calcule statistice (de exemplu: sume, medii etc.) Este logic faptul că, pentru un calcul corect, trebuie luate în considerare toate valorile din baza de date, deci acestea trebuie aduse în formă de la prima accesare a ei.
- *Record Orientation* – determină orientarea înregistrărilor în bloc (vertical sau orizontal)
- *Single Record* – se setează la *Yes* în cazul unui bloc de control care conține un câmp de tip *item* calculat conform unei formule statistice și specifică faptul că acest bloc trebuie să conțină, la un moment dat, numai o înregistrare.

Următorul grup din paleta de proprietăți al unui bloc determină modul de interacțiune dintre elementele blocului și tabelul pe baza căruia este construit acesta. Sub denumirea *Database* găsim următoarele proprietăți:

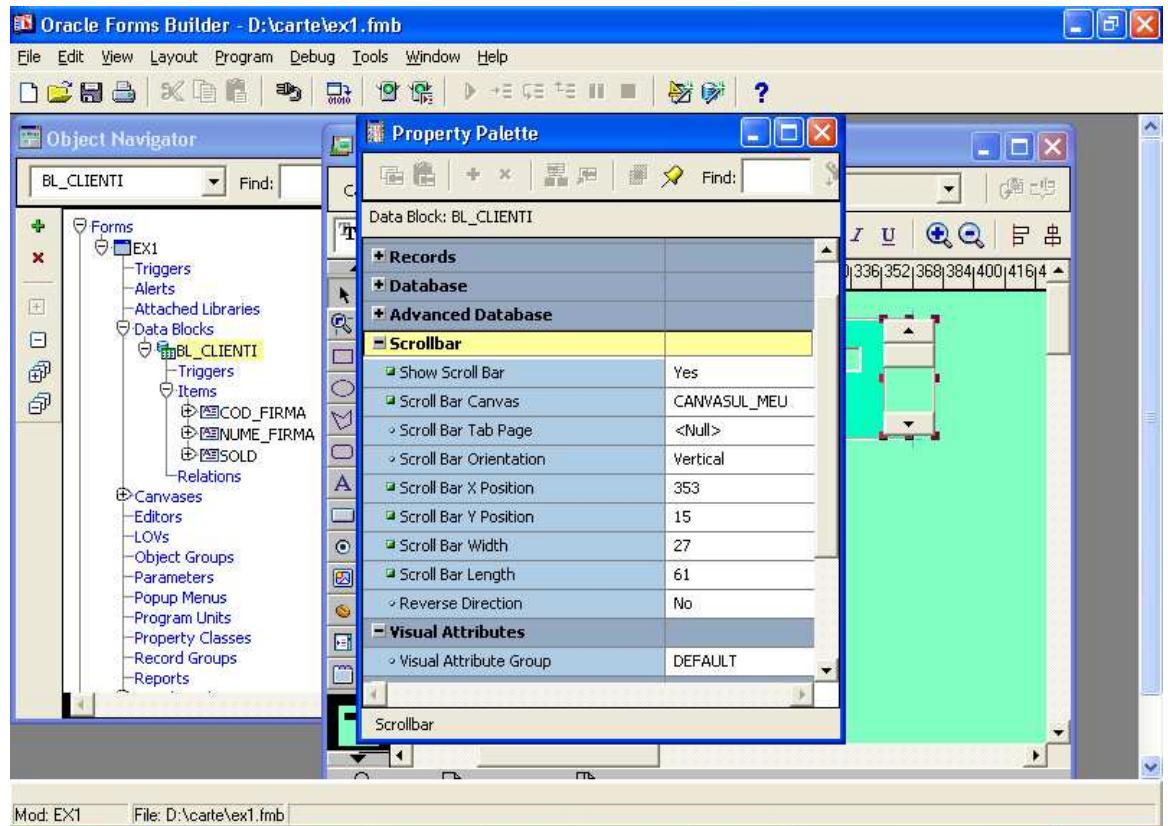


- *Database Data Block* – valoarea Yes înseamnă că blocul va fi populat cu informație din baza de date (există și blocuri populate prin proceduri PL/SQL)
- *Enforce Primary Key* – precizează dacă *Forms*-ul va verifica sau nu constrângerea de cheie primară de la nivelul tabelei înainte de inserarea sau modificarea înregistrărilor în baza de date
- *Query Allowed* – sunt sau nu permise interogări pe acest bloc
- următoarele proprietăți permit specificarea tabelei și a câmpurilor pe baza cărora este construit blocul curent. Există, de asemenea, proprietăți care controlează acțiunile care se pot efectua de către utilizator asupra înregistrărilor din blocul de date: *Query/Insert/Update/Delete Allowed*.
- *Where Clause* – dacă vrem să impunem un filtru la nivelul blocului de date, aici este locul lui; orice cerere restricționată efectuată asupra blocului de date va fi automat conjuncționată cu condiția impusă la

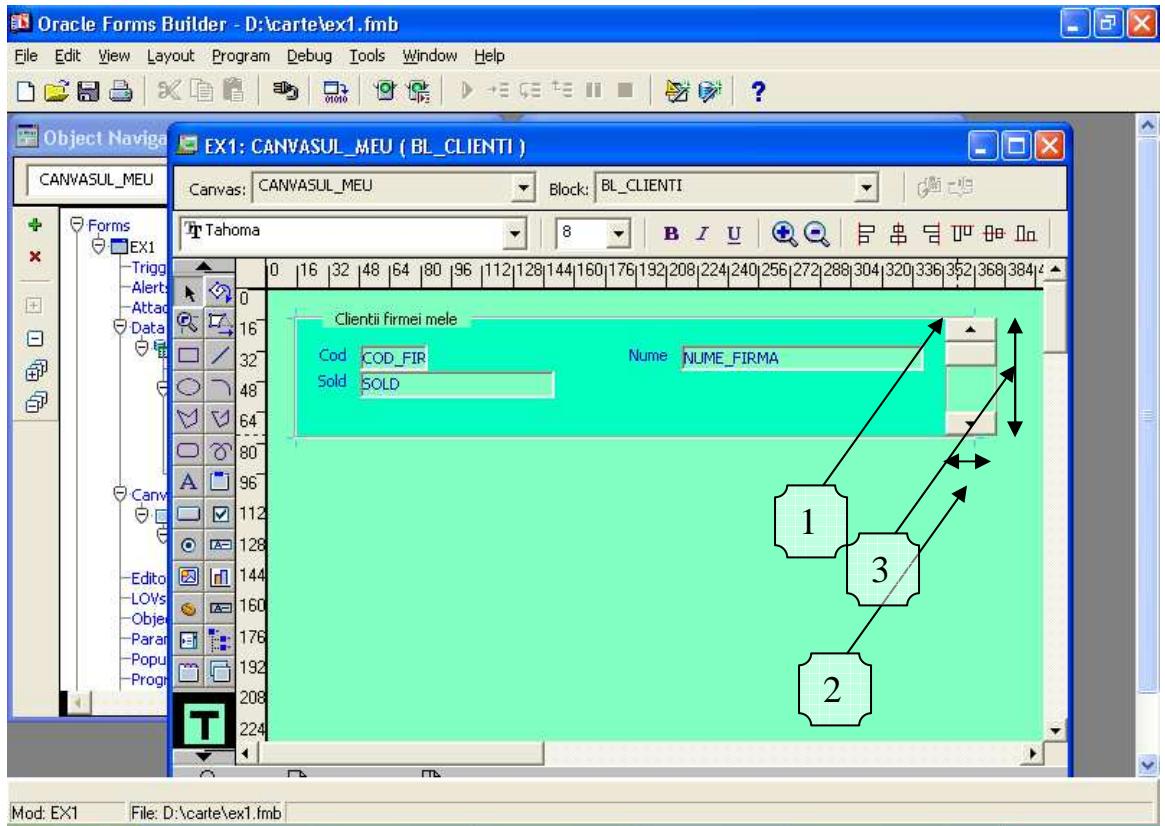
clauza *where*. De exemplu, condiția *cod_firma>505* va avea ca efect încărcarea tuturor înregistrărilor care verifică atât condiția impusă de utilizator, cât și condiția: codul firmei mai mare decât 505.

- *Order by Clause* – determină un mod implicit de afișare a înregistrărilor, sortate după *item*-ul precizat aici. Această ordine poate fi stabilită și în *runtime*, folosind caseta *Query Where*, procedură care a fost discutată anterior în acest capitol.
- *Locking Mode* – precizează modul de blocaj al unei înregistrări. Implicit, valoarea este setată la *Automatic*, care în cazul bazelor de date *Oracle* este echivalent cu *Immediate*, iar în cazul altor baze de date încearcă un comportament cât mai apropiat: atunci când utilizatorul inițiază editarea unui *item*, înregistrarea corespunzătoare din tabelă este blocată și nimeni nu o poate modifica (din *Forms* sau din alt orice alt mediu, de exemplu *SQL*), ci o poate doar accesa. La salvarea datelor din formular (și efectuarea unui *commit* asupra bazei de date), înregistrarea este deblocată. Modul de lucru *Delayed* realizează blocarea înregistrării doar în timpul operației de salvare.
- *Update Changed Columns Only* – când proprietatea este setată la *Yes* doar acele *item*-uri modificate de utilizator vor fi salvate în coloanele din baza de date corespunzătoare lor.
- *Enforce Column Security* – când această proprietate are valoarea *Yes*, elementele blocului de date pot fi modificate doar dacă utilizatorul curent are acest drept asupra coloanelor din tabela cărora le corespund elementele respective
- *Maximum Query Time* – oferă posibilitatea precizării unui interval de timp la expirarea căruia o cerere nesoluționată de către sistem este abandonată; este utilă atunci când proprietatea *Query All Records* este setată la valoarea *Yes*
- *Maximum Records Fetched* – permite precizarea unui număr maxim de înregistrări ce vor fi aduse în formă; după aducerea acestui număr de înregistrări, execuția cererii este abandonată.

Capitolul *Scrollbar* conține proprietăți legate de apariția și funcționalitatea unei bare de derulare la nivelul blocului de date. În cazul când este afișată o singură înregistrare din blocul de date nu își are sensul crearea unei bare de derulare, așa că vom studia ulterior, în detaliu, modul de definire al unui asemenea obiect. Totuși, să menționăm proprietățile unei bare de derulare:



Prima dintre acestea, *Show Scroll Bar*, determină/inhibă apariția barei de derulare. În cazul setării acestei proprietăți la valoarea *Yes*, elementele ce urmează în paleta de proprietăți permit definirea *canvas-ului* în care bara va apărea, a orientării sale (*Vertical/Horizontal*), precum și a poziției (colțul din stânga-sus al barei – vezi săgeata 1 din figura urmatoare) și a dimensiunilor ei (lățimea barei de derulare, indicată de sageata 2 și lungimea – săgeata 3 din figura de mai jos):



Renunțăm deocamdată la afișarea unei bare de derulare (setând proprietatea *Show Scroll Bar* a blocului de date la valoarea *No*) și analizăm următoarele elemente din paleta sa de proprietăți. Acestea sunt legate de eventualele atrbute vizuale folosite pentru afișarea sa (ce se setează în capitolul **Visual Attributes**), precum și de precizarea șirului de caractere, a culorii și a modului de hașurare a spațiului pe care se afișează elementele blocului de date (capitolul **Color**). De fapt, stabilirea unor valori pentru proprietăți legate de culoare/hașurare la nivel de bloc nu are nici un efect, ele aplicându-se doar în cazul: *item-urilor*, *paginilor tab*, *canvas-urilor*, *ferestrelor* și *butoanelor radio*.

Sintetizând ceea ce facem de câteva pagini bune încoaace (atributul *bune* se referă, evident, la număr și nu la calitatea paginilor, ultima caracteristică fiind lăsată la latitudinea cititorului), concluzionăm că am învățat să creăm un formular simplu, pe care apoi îl putem rafina folosind paletele de proprietăți ale obiectelor pe care le conține. Transferul de informație dintre formular și baza de date se efectuează prin intermediul meniului implicit.

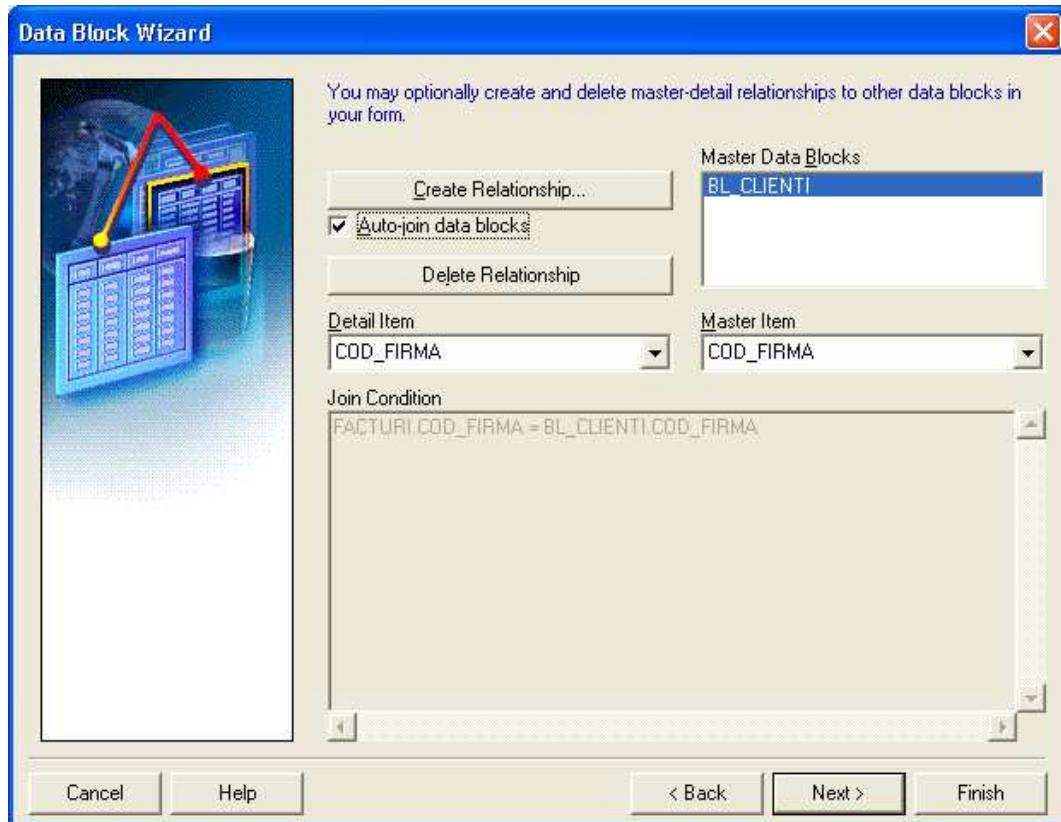
2.5. Blocuri *master-detail*

Formularul creat de noi gestionează datele referitoare la clienții firmei (inserare de noi clienți, modificarea informației legate de cei existenți, ștergerea celor care ne-au supărat, precum și interogarea tabelei clientilor). Ar fi mult mai plăcut dacă afișarea datelor clientului *x* ar fi însotită de afișarea, în același formular, a facturilor clientului respectiv. Acest lucru este posibil datorită existenței, la nivelul bazei de date, a unei relații *one-to-many* (unu-la-mai-multe) între cele două tabele pe care se bazează blocurile de date, fapt ce se implementează în *Forms Developer* prin crearea unui formular de tip *master-detail*. Într-un astfel de formular va exista o sincronizare între înregistrările din blocul *detail* și cele din blocul *master*.

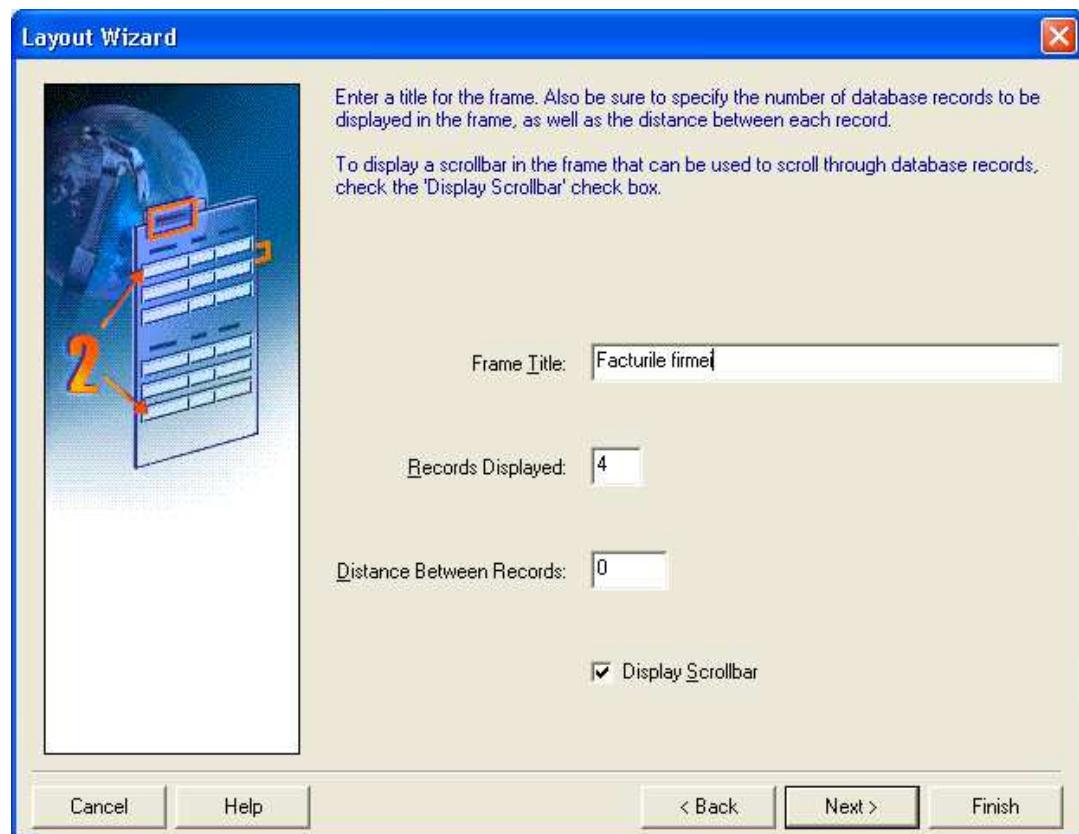
Un formular poate, deci, conține unul sau mai multe blocuri de date. Fiecare bloc de date poate gestiona o singură tabelă. Blocurile de date pot fi independente sau corelate printr-o relație de tip *master-detail*; efectul unei astfel de relații constă în sincronizarea datelor între blocul de date *master*, care este bazat pe tabela care conține cheia primară și cel *detail*, ce se bazează pe tabela care conține cheia externă. Pot fi create relații complexe, în care blocul *detail* al uneia este *master* pentru o alta, sau în care un bloc *master* are mai multe blocuri *detail*. De asemenea, se pot defini formulare de tip *master-detail* fără ca între tabelele pe care se bazează blocurile de date să fie explicit definită o relație unu-la-mai-mulți, caz mai rar întâlnit însă.

Să revenim la exemplul nostru, în care, până acum, am construit un formular pentru gestionarea clientilor firmei noastre. Unicul bloc de date definit, *BL_CLIENTI*, este bazat pe tabela *clienti*. Vom construi încă un bloc de date, care va gestiona înregistrările din tabela *facturi*. Procedeul de construire, cu ajutorul instrumentului *Data Block Wizard*, este deja cunoscut. La un moment dat, însă, să observăm că apar elemente specifice unei aplicații *master-detail*; ne poziționăm, deci, pe blocul *BL_CLIENTI*, apăsăm butonul *New*, reprezentat, în partea din stânga a ferestrei *Object Navigator*, prin

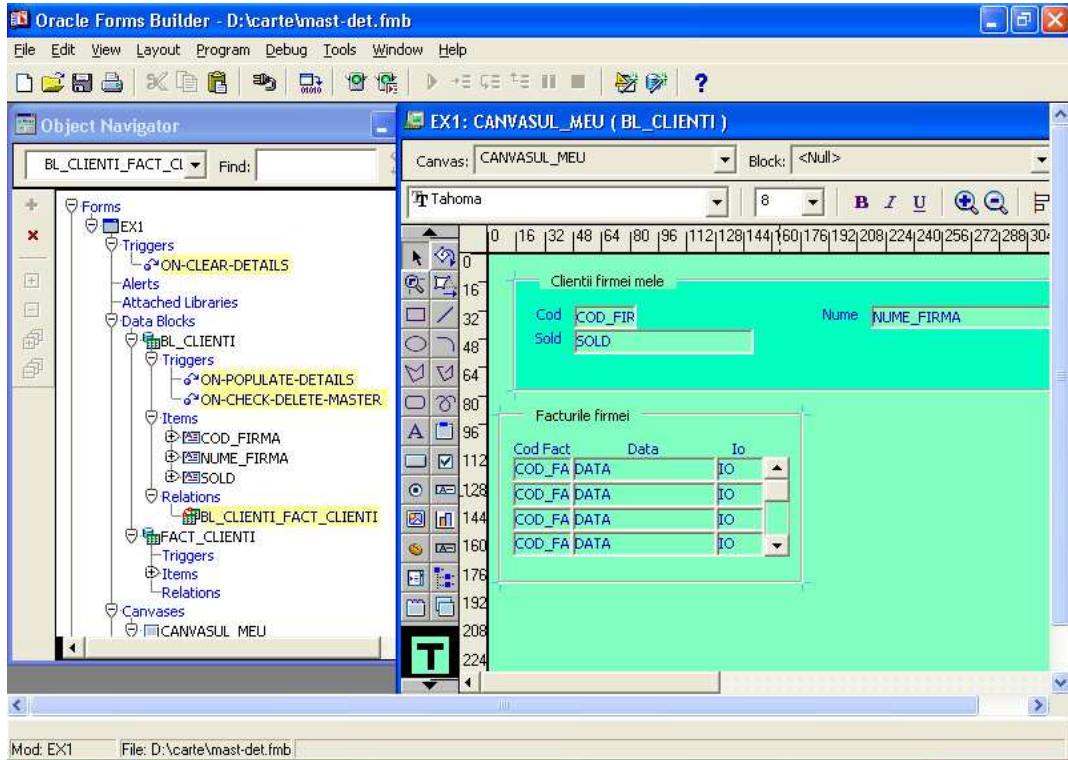
 icon-ul. Primii pași ai *wizard-ului* sunt cei prezenți la crearea blocului anterior. Definim blocul ca fiind bazat pe tabela *facturi*, selectăm toate coloanele pentru a fi aduse din tabelă în aplicația *Forms*, iar la pasul următor este sesizată existența unui alt bloc în formular și suntem chestionați în legătură cu crearea unei relații între cele două blocuri. În pagina *Master-detail* care apare se selectează butonul *Create Relationship...*, iar din lista de valori care apare se alege blocul *BL_CLIENTI*. Astfel, dacă la nivelul celor două tabele este definită o cheie externă, atunci instrumentul *Data Block Wizard* va genera automat o condiție de *join* între cele două blocuri; în caz contrar, va trebui să o definim explicit, alegând din cele două liste de valori *Detail Item* și *Master Item* elementele după care se va face corelarea între blocurile de date:



Următoarele pagini sunt cele cunoscute. Să continuăm, apelând *Layout Wizard*. Vom proceda la afișarea tuturor elementelor blocului asupra căruia lucrăm (numit de noi *FACT_CLIENTI*), cu excepția *item*-ului *cod_firma*. Acesta va fi afișat în blocul *BL_CLIENTI* și, datorită faptului că cele două blocuri vor fi corelate, afișarea acestui *item* va produce informație redundanta. Să alegem, de data aceasta, modul *tabular* de afișare (în care înregistrările vor fi prezentate sub formă de tabel), precum și afișarea simultană a 4 instanțe din blocul *detail* la un moment dat:

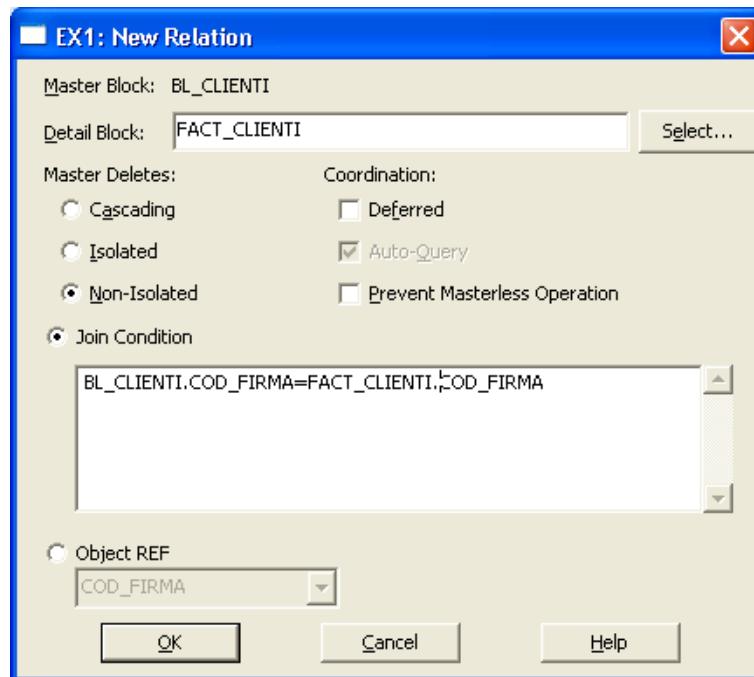


Rezultatul procesului de creare a acestui al doilea bloc este prezentat în figura următoare:



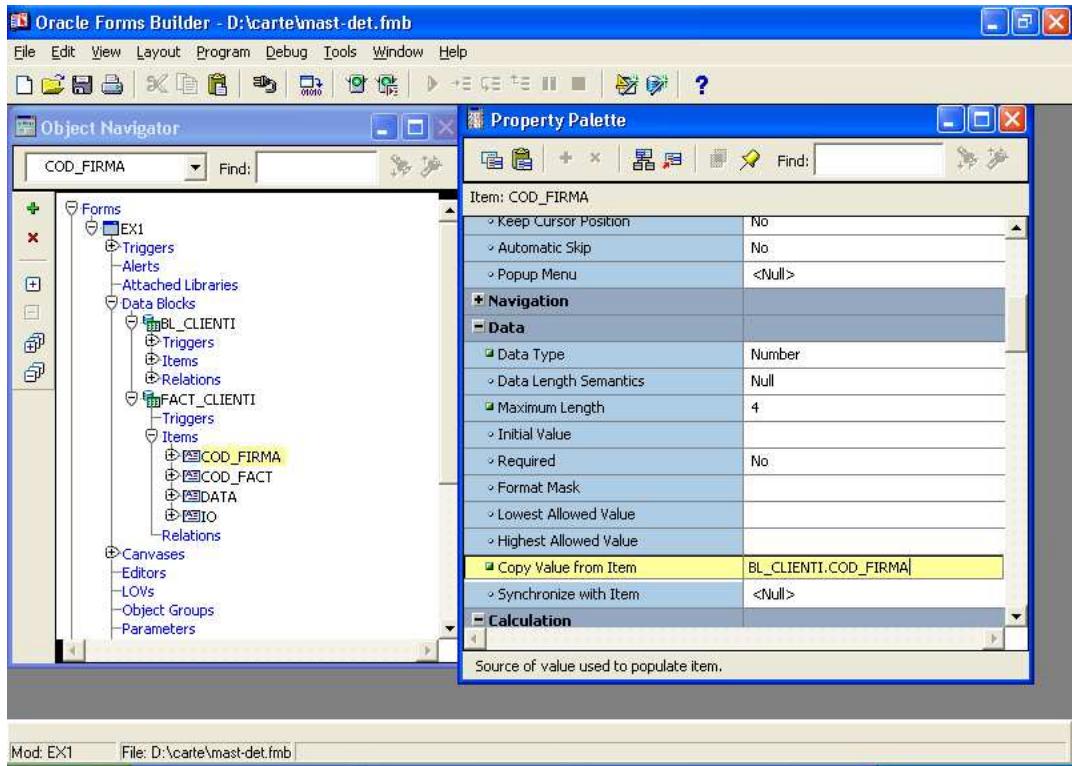
Corelarea între blocul *master* și cel *detail* este realizată de un obiect nou, de tip *Relations*, aparținând nodului blocului de date *master*. Implicit, numele relației se formează după următoarea regulă: *NumeBlocMaster_NumeBlocDetail*. Sunt generate automat unități de program (*trigger-i*) care mențin coordonarea între cele două blocuri. Priviți figura de mai sus și veți remarcă imediat acesti *trigger-i*, la creare sunt automat selectați în *Object Navigator*.

Există cazuri (de exemplu cel amintit anterior) în care între cele două tabele nu este definită o relație *one-to-many*. Dacă vrem, totuși, o coordonare între blocurile de date, trebuie să creăm obiectul *Relations* manual. Dând dublu *click* pe capitolul *Relations* al blocului de date *master*, obținem următoarea fereastră:

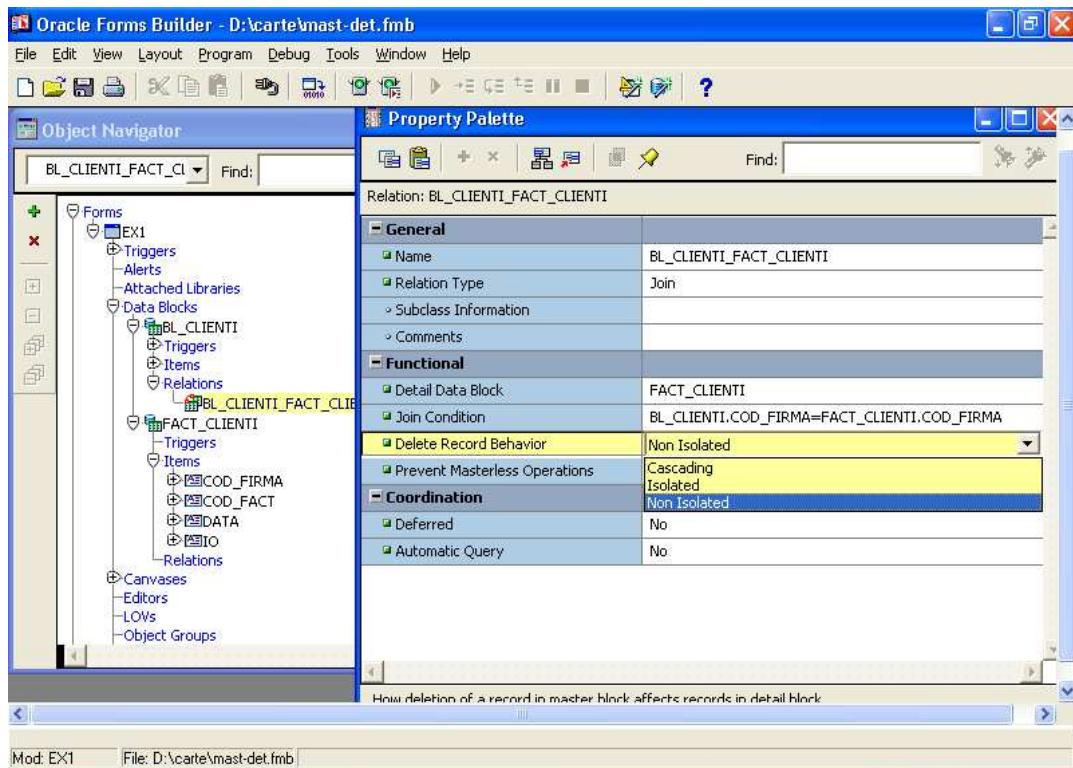


în care specificăm blocul de date *detail*, condiția de *join*, precum și modul în care va lucra corelarea dintre cele două blocuri. Atenție: definiți condiția de *join* folosind sintaxa *SQL* de definire a unei astfel de condiții, dar utilizați numele blocurilor în locul numelor tabelelor, precum și numele *item*-urilor în locul coloanelor tabelelor.

Acum sincronizarea dintre cele două blocuri se efectuează automat la schimbarea *focus*-ului în cadrul blocului *master*, în două etape: se golește blocul *detail*, apoi se repopulează conform frazei *select-SQL* definită de condiția de *join*. Mecanismul de repopulare are la bază folosirea proprietății *Copy Value from Item* din paleta de proprietăți a elementului *cod_firma* din blocul *detail*. Astfel, când se actualizează blocul *detail* (în urma apariției unui eveniment care necesită sincronizare), *item*-ul care are rol de cheie externă din blocul *detail* ia valoarea *item*-ului cheie primară din blocul *master* și sunt afișate numai înregistrările care verifică această condiție:

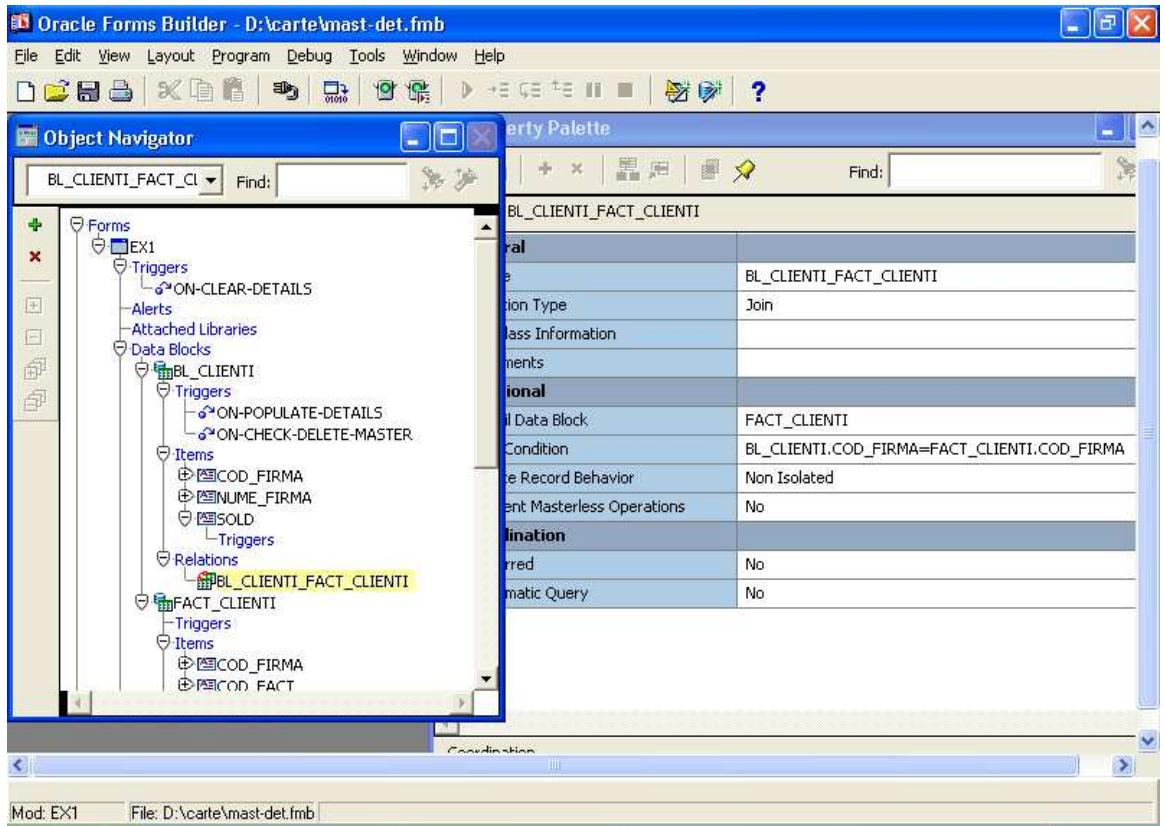


Modul în care se efectuează ștergerile și coordonarea dintre cele două blocuri se poate seta din fereastra în care am definit relația de *join* sau poate fi modificat ulterior, folosind paleta de proprietăți a relației:



Una din cele mai importante proprietăți este *Delete Record Behaviour*, care determină modul în care se fac ștergerile în blocul *master*. Implicit, valoarea acestei proprietăți este setată la *Non Isolated*, nepermittând ștergerea unei înregistrări din blocul *master* atât timp cât există detalii care se referă la ea. Dacă nu renunțăți la ideea ștergerii înregistrării *master*, ar trebui să se șterse, întâi, înregistrările corespondente ei din blocul de detalii, ... sau puteți să setați proprietatea despre care tot discutăm la valoarea *cascading*, fapt ce va determina ștergerea automată a înregistrărilor *detail* corespunzătoare înregistrării *master* pe care încercăm să o ștergem. O altă variantă este selectarea valorii *Isolated* pentru proprietatea *Delete Record Behaviour*, fapt ce permite ștergerea înregistrării *master*, chiar dacă există înregistrări *detail* care se referă la ea.

Cu siguranță ați observat deja faptul că modificarea modului de ștergere a înregistrărilor din formular a produs schimbări ale *trigger-ilor* generați de relația dintre cele două blocuri: să ne reamintim că, în cazul modului de ștergere implicit, *Non Isolated*, *trigger-ii* (sau, pe română, declanșatorii) care au apărut automat au fost:



- *on-clear-details* (definit la nivel de modul),
- *on-populate-details*,
- *on-check-delete-master* (definit la nivelul blocului *master*).

Un declanșator este un fragment de cod care se execută la declanșarea unui eveniment. Să încercăm împreună să înțelegem ce este cu acești *trigger*-i: primul dintre ei, *on-clear-details*, se declanșează când are loc un eveniment de coordonare între două blocuri. Dacă veți da *dublu-click* în *Object Navigator* pe *icon-ul* atașat acestui declanșator, veți vedea codul sursă, în *PL/SQL*. Poate deocamdată pare mai complicat, dar sperăm ca, treptat, să lămurim lucrurile. La declanșarea lui, acest *trigger* va goli blocul *detail*, urmând ca repopularea să fie efectuată de alți *trigger*-i, definiți la nivel de bloc *master*. Acolo vedem declanșatorul *on-populate-details*, generat implicit de *Forms* la crearea relației între cele două blocuri. Acest *trigger* se ocupă de popularea blocului *detail* cu înregistrări corelate cu înregistrarea curentă din blocul *master*, conform strategiei definite de *Forms*. Putem schimba, însă, modul de populare a blocului *detail*, modificând codul *PL/SQL* al acestui *trigger*. Atenție, acest *trigger* nu se va declanșa în

absența unui *trigger* de tip *on-clear-details*. Ultimul dintre *trigger-i*, *on-check-delete-master*, se declanșează la încercarea de stergere a unei înregistrări dintr-un bloc *master* aparținând unei relații *master-detail*. Să privim împreună codul acestui *trigger*:

```
--  
-- Begin default relation declare section  
--  
DECLARE  
    Dummy_Define CHAR(1);  
    --  
    -- Begin FACT_CLIENTI detail declare section  
    --  
    CURSOR FACT_CLIENTI_curs IS  
        SELECT 1 FROM FACTURI F  
        WHERE F.COD_FIRMA = :BL_CLIENTI.COD_FIRMA;  
    --  
    -- End FACT_CLIENTI detail declare section  
    --  
    --  
    -- End default relation declare section  
    --  
    --  
    -- Begin default relation program section  
    --  
BEGIN  
    --  
    -- Begin FACT_CLIENTI detail program section  
    --  
    OPEN FACT_CLIENTI_curs;  
    FETCH FACT_CLIENTI_curs INTO Dummy_Define;  
    IF ( FACT_CLIENTI_curs%found ) THEN  
        Message('Cannot delete master record when matching  
detail records exist.');
```

CLOSE FACT_CLIENTI_curs;
RAISE Form_Trigger_Failure;

```
END IF;  
CLOSE FACT_CLIENTI_curs;  
--  
-- End FACT_CLIENTI detail program section  
--  
END;  
--  
-- End default relation program section  
--
```

După câte vă amintiți, un program *PL/SQL* are 3 secțiuni: cea declarativă (optională), corpul programului și o secțiune de tratare a excepțiilor (tot optională). În cazul nostru, în prima parte este declarat un cursor. Acesta reprezintă o zonă de memorie în care *Oracle* stochează înregistrările returnate de o comandă *SQL*. Orice instrucțiune *select* generează un cursor, care este gestionat automat de sistem, însă utilizatorul poate crea cursoare proprii, apoi poate prelucra, înregistrare cu înregistrare, conținutul cursorului. Lucrul cu un cursor explicit implică următorii pași:

- crearea cursorului, în secțiunea de declarații (adică specificarea comenzi *SQL* care va popula cursorul); la acest moment cursorul este doar declarat, nu și populat;
- deschiderea cursorului (cu comanda *open nume_cursor*), fapt ce induce popularea sa;
- extragerea, înregistrare cu înregistrare, a conținutului cursorului în scopul prelucrării (cu comanda *fetch nume_cursor into nume-variabla*);
- închiderea cursorului (cu *close nume_cursor*).

Observați, în *trigger-ul on-check-delete-master*, definirea și deschiderea unui cursor care conține toate codurile de facturi ale clientului al cărui cod este valoarea item-ului *cod_client* din blocul *master BL_CLIENTI*:

```
CURSOR FACT_CLIENTI_curs IS
    SELECT 1 FROM FACTURI F
    WHERE F.COD_FIRMA = :BL_CLIENTI.COD_FIRMA;
```

Deci, în cazul în care înregistrarea *master* are detalii corelate cu ea, instrucțiunea *FACT_CLIENTI_curs%found* va întoarce valoarea *true*, în bara de stare va fi afișat mesajul

Message('Cannot delete master record when matching detail records exist.');

cursorul va fi închis, iar instrucțiunea *RAISE Form_Trigger_Failure* va produce eșuarea *trigger-ului* și întreruperea acțiunii curente (de ștergere a înregistrării *master* când are înregistrări *detail* corelate). În cazul în care cursorul nu conține înregistrări, acesta este închis, *trigger-ul* se termină cu succes și ștergerea este permisă.

Aceștia au fost, deci, *trigger-ii* creați automat în cazul unei relații *master-detail* cu condiție de ștergere *Non Isolated*. Să schimbăm, în paleta de proprietăți a relației, această valoare cu *Isolated*. Observați dispariția *trigger-ului on-clear-details*, lucru care era de așteptat deoarece acum știm că acest mod de lucru permite ștergerea înregistrărilor *master* chiar în cazul existenței unor înregistrări *detail* corelate cu aceasta.

Ultimul mod de ștergere este *Cascading*. Reamintim că în acest caz, la ștergerea unei înregistrări *master*, vor fi șterse automat înregistrările *detail* corespunzătoare acesteia. Ce înseamnă de fapt "automat"? Cine efectuează această ștergere? Bineînteles

că trigger-ul *pre-delete* care a fost generat la nivel de bloc. Trigger-ii *pre-* se declanșează înainte de eveniment, în cazul nostru acesta se va declanșa înainte de încercarea de a șterge o înregistrare *master* și va șterge toate înregistrările *detail* corelate cu aceasta. Secvența de cod *PL/SQL* este, de aceasta dată, mult mai simplă (șterge din tabela *facturi* toate înregistrările corespunzătoare clientului al cărui cod este în blocul *BL_CLIENTI*, item-ul *cod_firma*):

```
-- 
-- Begin default relation program section
--
BEGIN
  --
  -- Begin FACT_CLIENTI detail program section
  --
  DELETE FROM FACTURI F
  WHERE F.COD_FIRMA = :BL_CLIENTI.COD_FIRMA;
  --
  -- End FACT_CLIENTI detail program section
  --
END;
--
-- End default relation program section
--
```

În urma execuției acestui *trigger* nu vor mai exista în tabela înregistrări copil ale înregistrării ce se vrea ștearsă din tabela părinte și operația se termină cu succes.

Cred că este timpul să vă luați o clipă de răgaz și să rulați aplicația, încercând să vedeați cum funcționează în fiecare din cele trei cazuri.

Nu vă bucurați, nu am terminat cu paleta de proprietăți a relației! Mai sunt câteva lucruri interesante, cum ar fi *Prevent Masterless Operation*. Valoarea implicită este *No*, însemnând că se poate face *query* pe blocul *detail*, indiferent dacă blocul *master* este sau nu populat cu vreo înregistrare. În cazul contrar, când valoarea acestei proprietăți este *Yes*, încercarea de a efectua o cerere pe blocul *detail* eşuează, eroarea obținută fiind:

FRM-41105: You cannot query records without a saved parent record

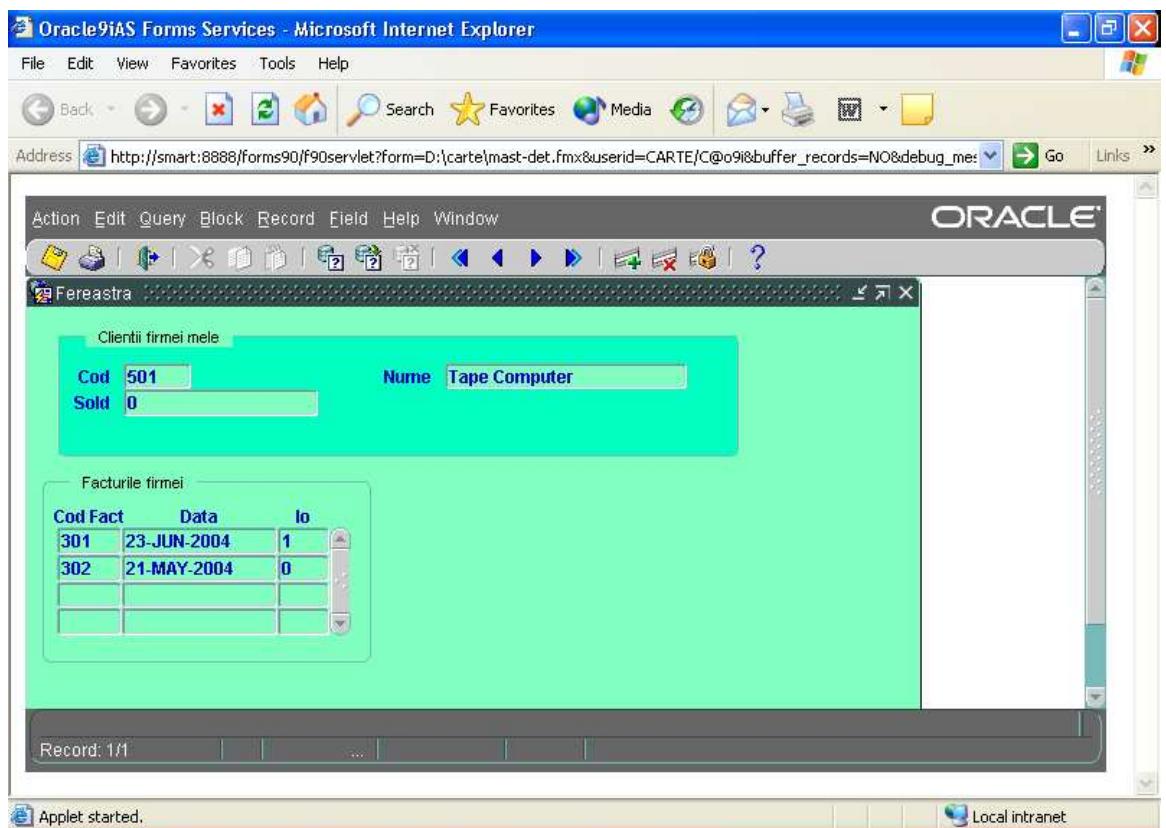
Ultimul nod în paleta de proprietăți a relației, *Coordination*, se referă la modul în care se face coordonarea între blocuri. Proprietatea *Deferred* este setată implicit la *No*, valoarea *Yes* determinând popularea blocului detaliu doar atunci când *focus-ul* intră pe el. Acest mod de lucru poartă numele de populare cu întârziere, iar ea poate fi de două feluri:

- în cazul când proprietatea *Automatic Query* are valoarea *No*, popularea se numește a fi cu întârziere, dar cu subcerere, pentru popularea blocului *detail* fiind necesară încă o dată comanda *Execute Query*;

- în cazul când proprietatea *Automatic Query* are valoarea *Yes*, popularea se numește a fi cu întârziere și cu cerere implicită, pentru popularea blocului *detail* fiind necesară doar mutarea *focus*-ului pe acesta.

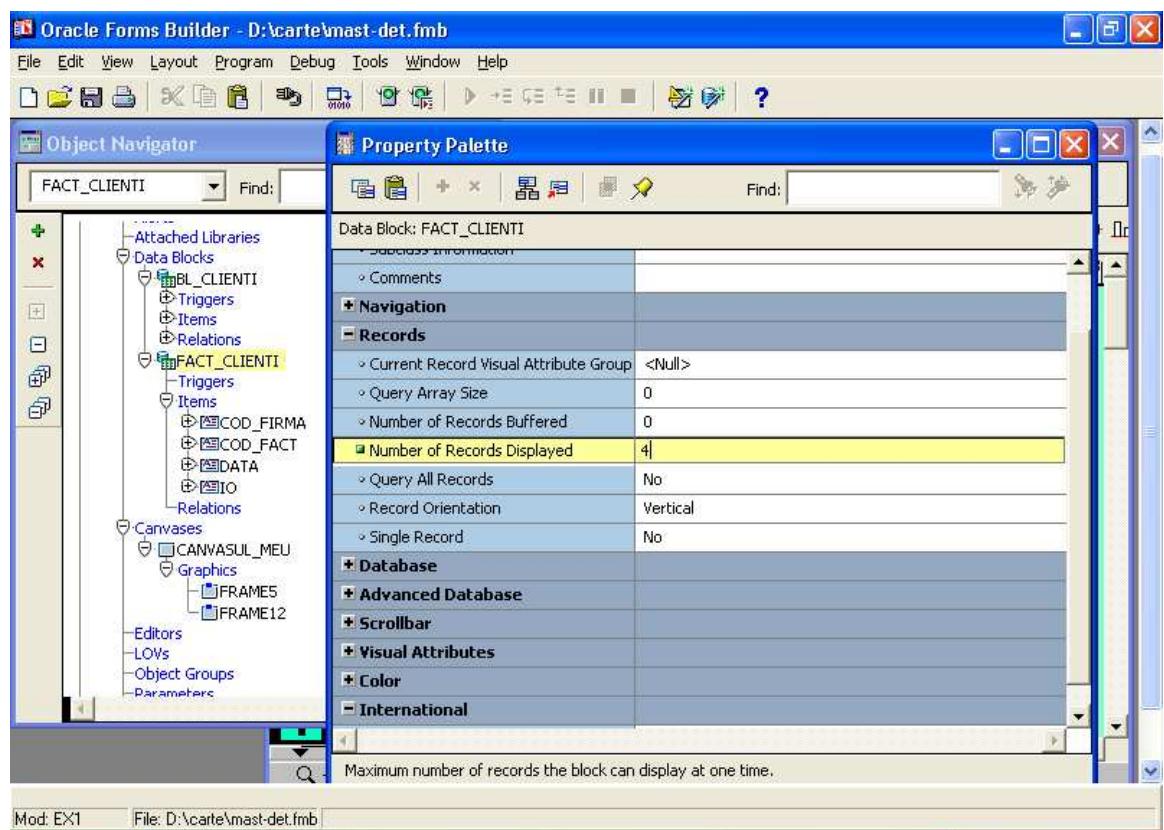
Primul mod de coordonare (populare cu întârziere și cu subcerere) este folosit atunci când se dorește obținerea înregistrărilor ce îndeplinesc un filtru cu o condiție dublă, atât din primul cât și din al doilea bloc. În acest caz se scrie condiția în primul bloc, se actionează butonul *Execute Query*, se dă *click* pe al doilea (nu va fi încă populat), se introduce și a doua condiție, apoi se actionează din nou butonul *Execute Query*.

Am creat, deci, un formular de tip *master-detail*, i-am asigurat funcționalitatea necesară, iar acum ar trebui să fim mulțumiți de rezultatele efortului nostru:



Să reamintim că între cele două blocuri corelarea se face automat (în cazul interogărilor blocului *master*, blocul *detail* va fi populat automat; inserând o înregistrare *detail*, ea va fi automat asociată cu înregistrarea afișată în blocul *master*), iar regula implicită de ștergere este următoarea: nu se poate șterge o înregistrare *master* atât timp cât ea are înregistrări *detail* corespondente (aceste moduri de lucru pot fi modificate, așa cum am aratat anterior).

Remarcați afișarea simultană, în blocul *detail*, a 4 înregistrări. Acest fapt a fost setat la crearea interfeței blocului de date (făcută, va amintiți?, cu *Layout Wizard*) și se regăsește, bineînțeles, în paleta de proprietăți a blocului de date, de unde poate fi și modificată:

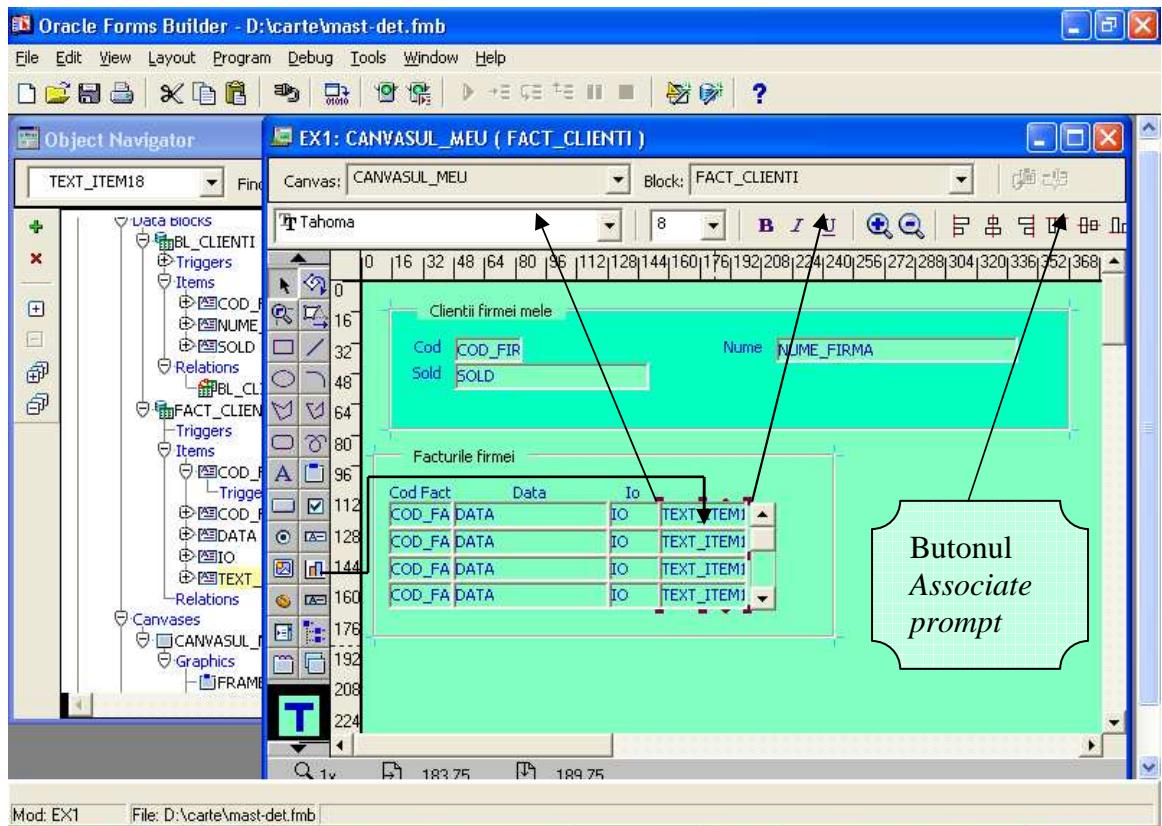


2.6. Gestionarea elementelor *text*

Un element de tip *text* este o interfață prin care puteți interoga și actualiza datele. În general, un obiect *text* corespunde unei coloane dintr-o tabelă. La crearea unui element nou, tipul său implicit este *text*. Proprietățile elementului sunt, apoi, definite în paleta sa de proprietăți. Crearea unui element *text* nou se poate face în unul din mai multe moduri:

- folosind utilitarele de tip *wizard*;
- convertind un element deja existent într-unul de tip *text* (în paleta sa de proprietăți, la *Item Type*, alegem din lista de valori atașata intrarea *text*);
- acționând instrumentul *Text Item* din *Layout Editor*;
- apăsând butonul *Create icon* din *Object Navigator*;

Pentru exemplificarea celei de a treia metode de creare a unui element *text*, să invocăm *Layout Editor*, să largim cadrul ce conține blocul de date *FAC_CLIENTI*, să mutăm bara de *scroll* spre dreapta, pentru a face loc unui element de tip *text*, apoi, selectând butonul pentru crearea unui astfel de obiect, să îl poziționăm pe locul în care vrem să apară. Atenție, în partea superioară a acestui editor avem două liste *pop-up* de valori din care trebuie să selectăm blocul de date și *canvas*-ul cărora vrem să le aparțină elementul nou creat.

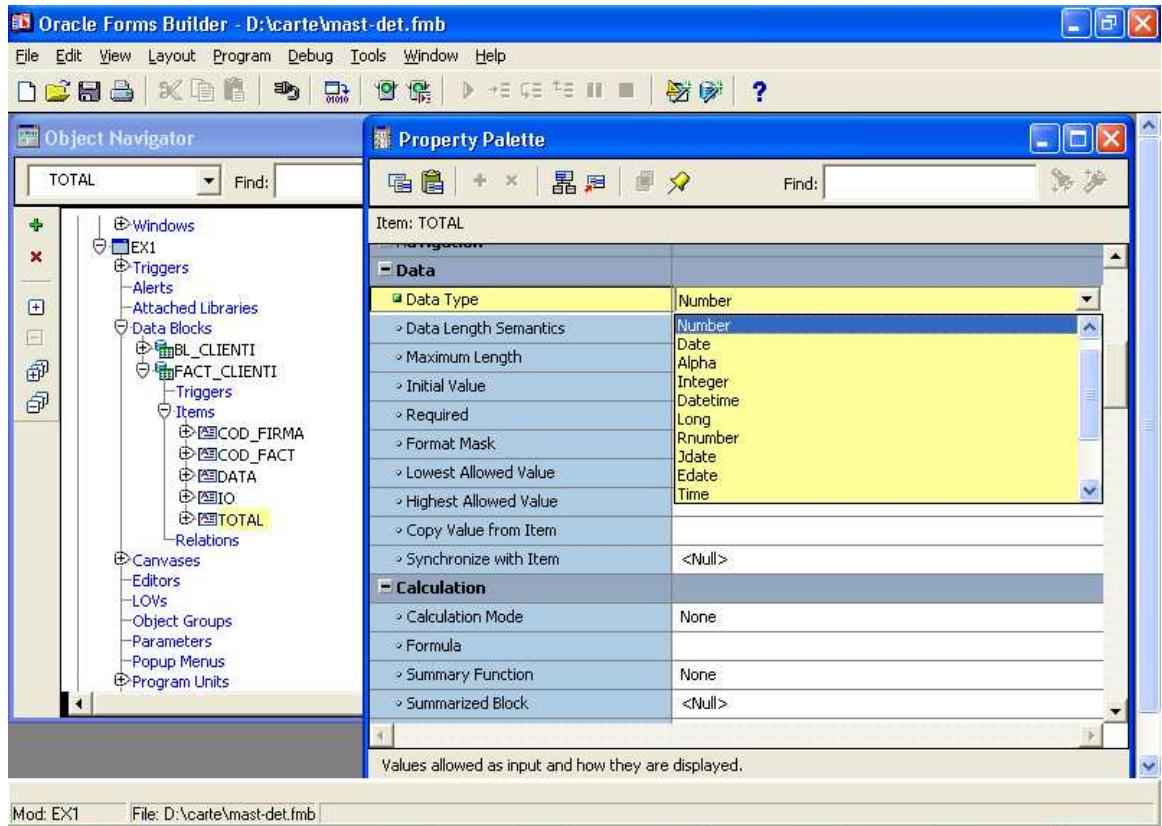


Ar fi foarte bine dacă am reușit să asociem elementului creat o etichetă. Pentru aceasta acționăm butonul etichetat cu litera A din bara de instrumente aflată în partea stângă și creăm un element nou, deasupra celui de tip *text* creat anterior. După ce adăugăm textul dorit, selectarea simultană (ținând tasta *Ctrl* apăsată) a celor două obiecte și acționarea butonului *Associate prompt* (vezi figura anterioară) va determina asocierea etichetei cu elementul *text* și tratarea lor unitar (orice încercare de deplasare a unuia dintre elemente va determina deplasarea celuilalt, ștergerea se face în grup etc).

Remarcați faptul că, deoarece noul element *text* aparține blocului *FACT_CLIENTI*, care afișează simultan câte 4 înregistrări, acesta va avea afișate tot câte 4 instanțe ale sale. Noul *item* nu ne va fi, însă, de prea mare folos dacă nu îi setăm proprietățile. Trebuie să specificăm ce vrem să afișeze noul element; este sau nu bazat pe o coloană dintr-o tabelă, se calculează după vreo formula etc.

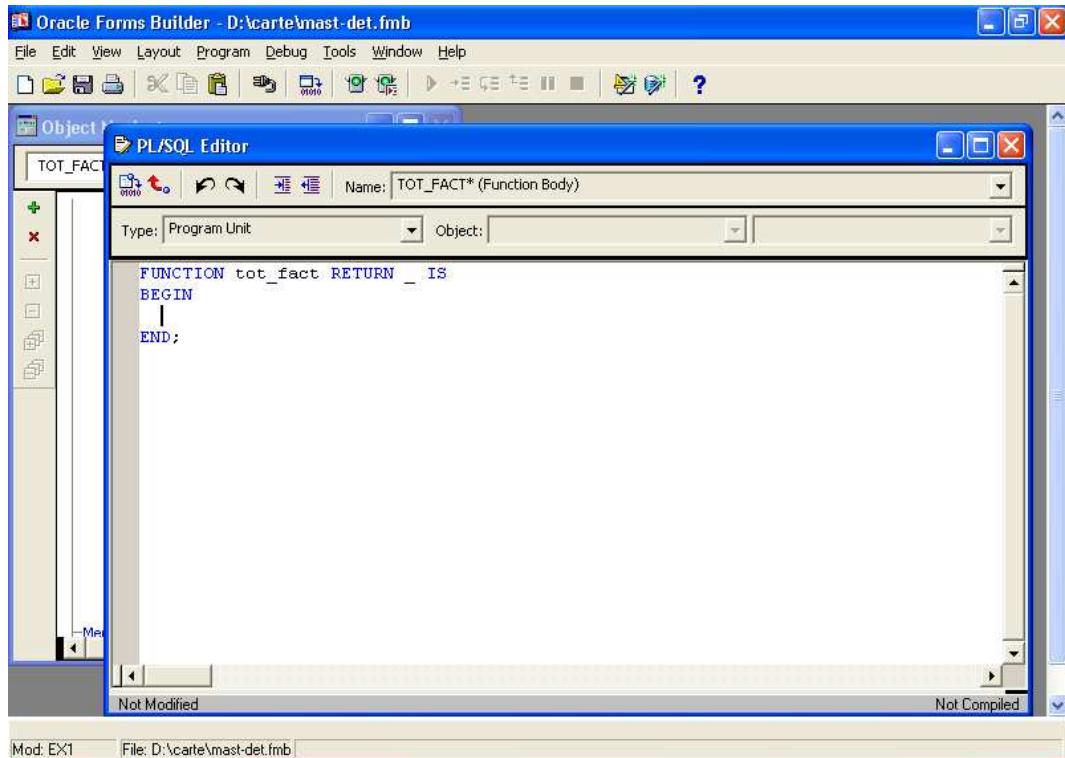
Haideți să facem un pas important (sperăm că acesta va fi înainte) și să încercăm să convingem acest element să ne arate, pentru fiecare factură, valoarea totală a

produselor pe care aceasta le conține. Pentru început să acționăm tasta *F4* invocând, astfel, paleta de proprietăți a acestui *item*:

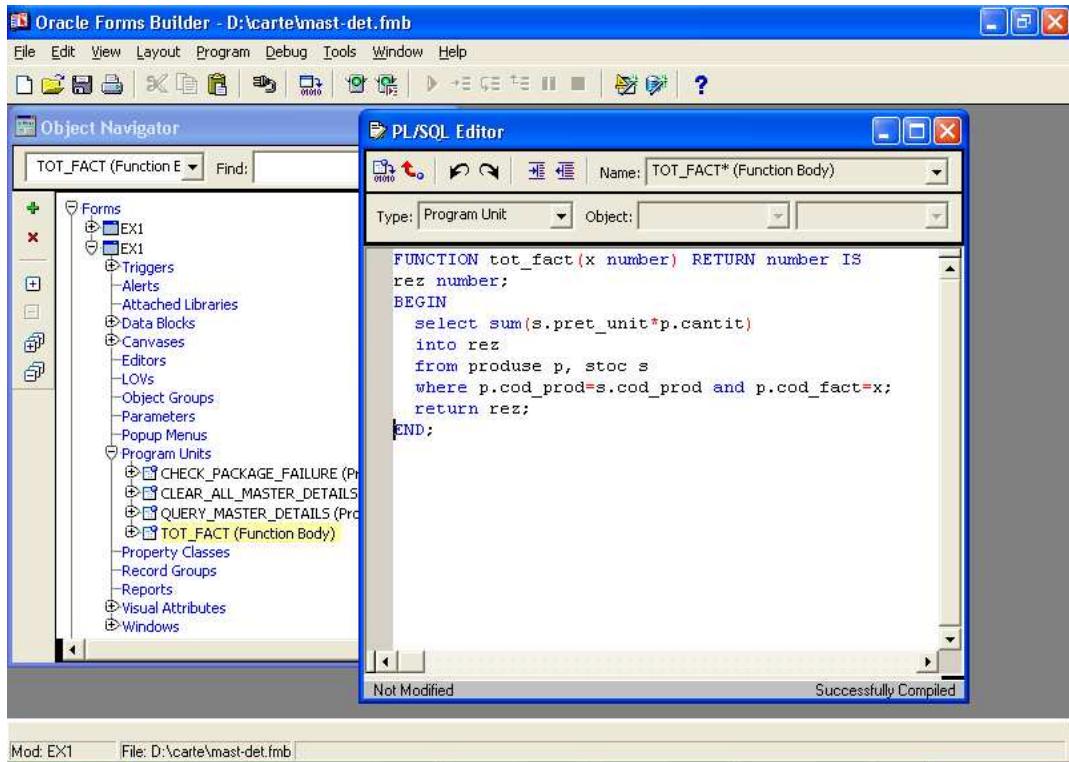


După ce, la intrarea *Name*, am numit acest element într-un mod mai “uman” (“total”, de exemplu), îi setăm tipul ca fiind *Number*, lucru foarte important. Implicit, acest element ar fi fost tratat ca fiind de tip sir de caractere. Capitolul *Calculation* este locul în care vom defini modul de calcul pentru noul element. Nu vom folosi funcțiile oferite la intrarea *Summary Function* (sunt cele disponibile și în SQL), ci vom seta *Calculation Mode* la valoarea *Formula*, iar la intrarea *Formula* va trebui să specificam o expresie *PL/SQL* mai complexă. Putem să trecem ca formulă o expresie aritmetică ce are ca operanți *item*-urile din formularul curent (folosind sintaxa *NumeBloc.NumeItem*) sau, dacă avem nevoie de informație care nu a fost încă încărcată în formular (cazul nostru – trebuie să interogăm tabela *stoc*), vom crea o funcție ce va întoarce valoarea ce se dorește a fi atribuită nouui element. Funcția va fi creată cu ajutorul cunoștințelor pe care le avem (sau pe care ar trebui să le avem) relativ la limbajul *PL/SQL*. Putem crea această funcție la nivelul bazei de date, caz în care poate fi reutilizată în orice mediu de programare

Oracle sau la nivel de modul, caz în care poate fi utilizată doar în modulul în care a fost creată. Optăm pentru a doua varianta și, în *Object Navigator*, selectăm nodul *Program Units*, apoi acționăm butonul *New*. În caseta de dialog care apare selectăm tipul obiectului ce va fi creat la valoarea *Function*, iar numele: *tot_fact*. Click pe *OK* determină invocarea editorului *PL/SQL*:

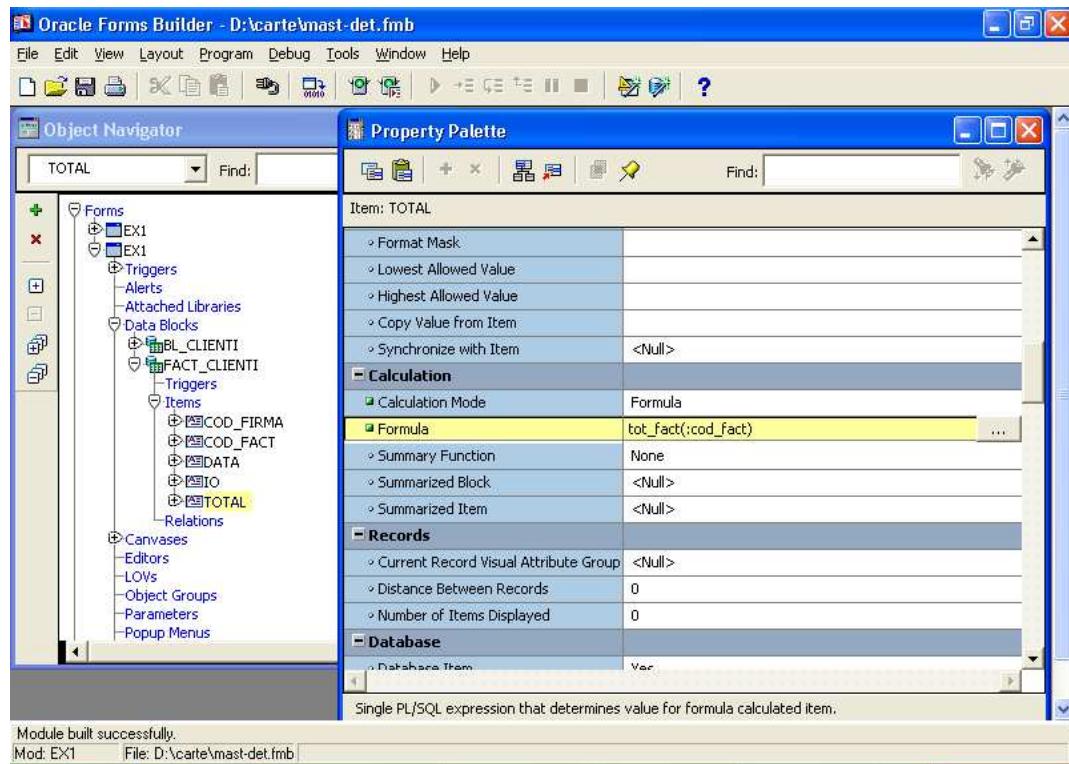


Să ne reamintim că vrem ca această funcție să calculeze, pentru o factură dată, suma produselor existente pe acea factură. Ca să funcționeze în caz mai general, codul facturii ar trebui dat ca parametru al funcției. Ne reamintim că în specificația unei funcții *PL/SQL* trebuie să precizăm tipul de date returnat de funcție (în cazul nostru *number*). În variabila *rez* calculăm suma produselor dintre valoarea unitară a produselor de pe factură și cantitate. Prețul fiecărui produs se află în tabela *stoc*, în timp ce cantitatea se află în tabela *facturi*, aşa ca am facut un *join* între aceste tabele. Este returnată valoarea variabilei *rez*:



Acționarea butonului din stânga-sus al editorului *PL/SQL* (etichetat *Compile*) va determina compilarea funcției și, în cazul fericit, afișarea în dreapta jos a mesajului *Successfully Compiled*. Dacă există erori, acestea vor fi afișate în partea de jos a ferestrei editorului *PL/SQL* și vor trebui corectate.

Odată creată, funcția poate fi invocată oriunde în aplicația curentă. În paleta de proprietăți a *item-ului total* setăm modul de calculare la valoarea *Formula*, iar valoarea pe care o ia acest element va fi cea returnată de funcția *tot_fact*, calculată pentru parametrul :*cod_prod*. Să precizăm doar faptul că un *item* poate să nu fie prefixat de numele blocului din care face parte doar în cazul în care el este invocat în același bloc. Rezultatul este:



După ce veți rula aplicația și vă veți convinge că totalul de pe fiecare factură este afișat corect, vă propun să revenim la elementele de tip *text*, acestea fiind cele mai frecvente în aplicațiile *Forms*, și să analizăm, în continuare, proprietățile lor.

Am creat un *item* de tip *text* folosind instrumentul *Layout Editor*. Precizam, cu câteva pagini în urmă, că acesta putea fi creat și din *Object Navigator*, dând *click* pe butonul *New* în timp ce este selectat un alt element *text* din același bloc cu cel ce se dorește a fi creat. Un element *text* obținut în acest mod, nu va fi regăsit în *Layout Editor* și aceasta deoarece nu î se asociază implicit nici un *canvas* în care să fie afișat. Să analizăm paleta de proprietăți a unui element *text*: capitolul *General* permite modificarea numelui elementului și selectarea, din lista *pop-up* a proprietății *Item Type*, a tipului său. Capitolul *Functional* permite modificarea funcționalității implicate a elementului și conține următoarele proprietăți:

- *Enabled* este corelată cu proprietatea *Keyboard Navigable* a capitolului *Navigation*, astfel:

<i>Enabled</i>	<i>Keyboard navigable</i>	<i>Comportament</i>
----------------	-------------------------------	---------------------

<i>Yes</i>	<i>Yes</i>	<i>Item-ul este inclus în navigare; poate fi accesat și cu ajutorul mouse-ului</i>
<i>Yes</i>	<i>No</i>	<i>Item-ul nu poate fi accesat prin navigarea implicită, cu tastatura; poate fi accesat doar cu mouse-ul</i>
<i>No</i>	<i>No</i>	<i>Item-ul este practic exclus din procedeul de navigare; nu poate fi manipulat nici cu mouse-ul, nici cu tastatura</i>
<i>No</i>	<i>Yes</i>	<i>Item-ul nu este activ, el nu poate fi manipulat. Valoarea celei de a doua proprietăți este, practic, tot No.</i>

- *Justification* – permite specificarea modului de aliniere a valorii *item-ului*;
- *Multi-line* – este sau nu permisă spațierea textului pe mai multe linii;
- *Wrap Style* - modul în care se efectuează trecerea la linie nouă în cazul în care textul depășește suprafața de editare;
- *Case Restriction* – textul este convertit la majuscule, litere mici sau este afișat așa cum este introdus de utilizator;
- *Conceal Data* – este o proprietate specifică doar *item-urilor text* pe un singur rând și este folosită pentru introducerea parolelor: la fiecare tastă apăsată de utilizator, în câmpul respectiv este afișat caracterul *.
- *Keep Cursor Position* – cursorul nu se mișcă în timp ce utilizatorul tastează valoarea câmpului;
- *Automatic Skip* – salt la *item-ul următor* când cel curent este completat pe toată lungimea sa.

Capitolul *Navigation* stabilește modul de navigare spre și de la *item* la apăsarea tastei *Tab*, permitând specificarea *item-urilor anterior și următor* celui curent, în cazul în care acesta este *Keyboard Navigable*. Valoarea implicită *No* a acestor două proprietăți determină navigarea în ordinea în care *item-urile* apar în *Object Navigator*.

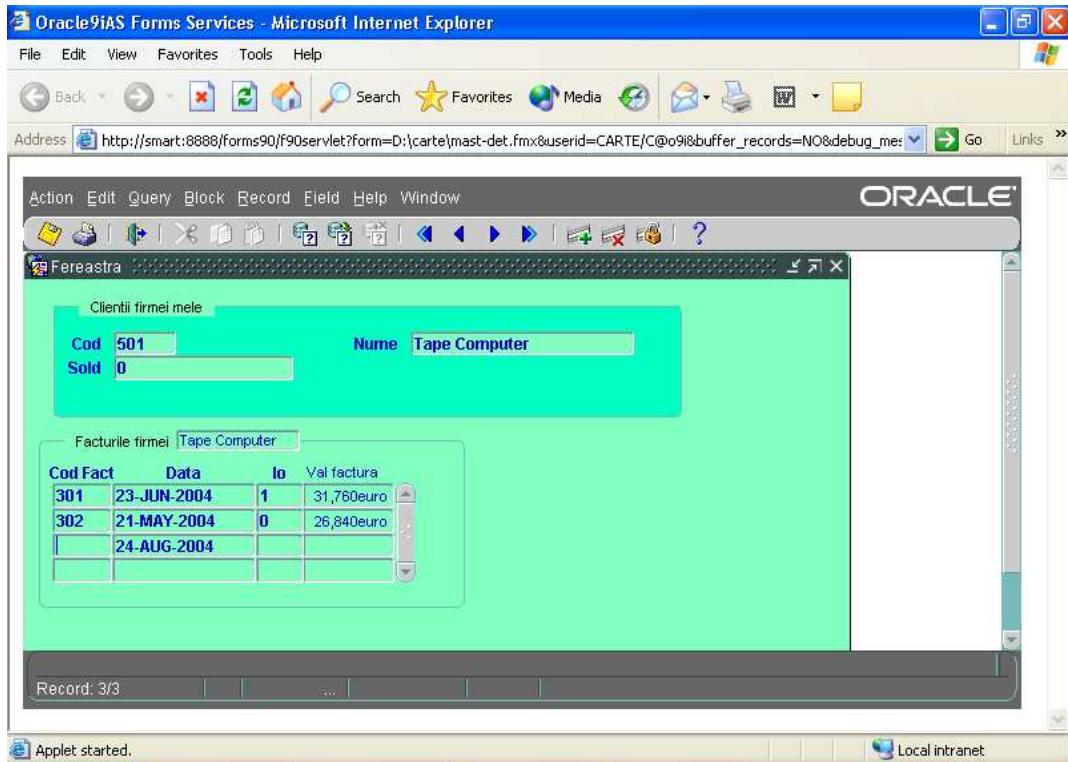
Capitolul *Data* este utilizat pentru controlul modului în care datele sunt introduse și afișate. Cel mai des folosite proprietăți sunt:

- *Data Type* – specificarea tipului de valori pe care *Forms Developer* le permite a fi introduse;
- *Maximum Length* – lungimea maximă a valorii introduse de utilizator;

- *Initial Value* – valoarea implicită pe care *Forms Developer* o asociază respectivului *item* de fiecare dată când e creată o înregistrare nouă; poate fi, de exemplu, o valoare fixă, sau poate fi selectată dintr-o secvență, folosindu-se sintaxa *:sequence.NumeSecventa.nextval*; pentru o variabilă de tip *date* pot fi folosite data și ora sistemului de operare, în forma \$\$DATE\$\$, \$\$DATETIME\$\$, \$\$TIME\$\$.
- *Required* – dacă această proprietate este setată la valoarea *Yes*, înregistrarea curentă este considerată invalidă în cazul când acest *item* are valoarea *Null*;
- *Format Mask* – permite definirea unui prototip (conform standardelor *SQL*) pentru introducerea datelor. Se pot defini prototipuri-utilizator, prin inserarea caracterelor adiționale între ghilimele (de ex., pentru elementul *total*, care afișează valoarea totală a unei facturi, un prototip ar putea arata așa: 999.999”euro”);
- *Lowest/Higher Value* – specifică domeniul maxim de valori acceptate;
- *Copy Value from Item* – indică sursa valorii pe care *Forms*-ul o folosește pentru a popula *item*-ul. Este utilă în cazul unei relații *master-detail*, fapt discutat într-un capitol anterior;
- *Synchronize with Item* – specifică numele unui *item* (care poate fi ales dintr-o listă de valori conținând *item*-urile din blocul curent) cu care *item*-ul curent se va sincroniza. Orice modificare a valorii aceluiași *item* va produce actualizarea celui curent.

Încercati să convingeți aplicația noastră să facă următoarele lucruri:

- Lângă numele cadrului al doilea să fie un *item* care să arate numele firmei ale cărei facturi sunt afișate;
- Valoarea totală a facturii să fie afișată conform capturii de ecran următoare;
- La crearea unei înregistrări pentru o factură nouă, câmpul *data* să fie completat automat cu data sistemului;



Indicații: observați că lângă numele celui de al doilea cadru apare și denumirea firmei care trebuie să se sincronizeze cu item-ul *nume-firma* din blocul *BL_CLIENTI*. Creați, deci, în blocul *master* un element nou, de tip *text*. Poziționați-l ca în captura anterioară de ecran și setați-i proprietatea *Synchronize with Item* la valoarea *nume_firma*. Setați, pentru item-ul *total*, proprietatea *Format Mask* la "999.999"euro", iar proprietatea *Initial Value* a câmpului *data* la data sistemului, în forma \$\$DATE\$\$.

Capitolul *Calculation* a fost prezentat odată cu crearea item-ului numit *total*, aşa că trecem la capitolul *Records*. Una din cele mai importante proprietăți se află aici: *Number of Items Displayed*. Valoarea implicită 0 înseamnă, de fapt, că numărul de instanțe afișate este același cu numărul de înregistrări afișate simultan, proprietate setată la nivelul blocului de date (vă mai amintiți? Se numea *Number of Records Displayed*.) Modificarea acestei valori determină afișarea unui număr de instanțe corespunzător noii valori.

Capitolul *Database* conține proprietăți ce controlează sursa informației din item-ul curent (este bazat pe o tabelă sau este item de control), precum și modul de comunicare între acesta și tabelă. La nodul *Database* găsim:

- *Database Item* – setați această proprietate la valoarea *Yes* dacă elementul curent ia informația dintr-o tabelă;
- *Column Name* – numele coloanei (din tabela pe care este construit blocul curent) din care *item-ul* ia informația;
- *Primary Key* – coloana pe care se bazează *item-ul* conține sau nu cheie primară; în caz afirmativ, *Forms Developer* verifică unicitatea valorii acordate acestui *item*;
- *Query Only* – implicit are valoarea *No*; schimbarea ei la valoarea *Yes* permite includerea *item-ului* numai în executarea de cereri, nu și în operații *insert* sau *update*;
- *Query/Insert/Update Allowed* - controlează ce operații *DML* sunt permise;
- *Query Length* – specifică lungimea maximă a unui sir ce poate fi stocat în acest *item* în cazul efectuării unei cereri. Valoarea implicită 0 înseamnă, de fapt, lungimea maximă a câmpului, setată prin proprietatea *Maximum Length*, de la capitolul *Data*;
- *Case Insensitive Query* – permite sau nu utilizatorului să efectueze pentru *item-ul* respectiv cereri *case-insensitive*;
- *Update Only if Null* – setată la valoarea *Yes*, această proprietate nu permite utilizatorului modificarea *item-ului* curent decât dacă acesta are valoarea *Null*;
- *Lock Record* – blochează (sau nu) rândul din tabelă corespunzător înregistrării curente în cazul când aceasta este modificată.

Următoarele două capitole din paleta de proprietăți a unui element *text*, *List of Values* și *Editor* vor fi studiate în detaliu ulterior; acum ne oprim asupra proprietăților grupate sub numele *Physical*, care se referă la poziția și dimensiunile *item-ului*. Găsim aici:

- *Visible* – care setată la valoarea *Yes* produce afișarea elementului; în caz contrar, elementul există, poate fi folosit, de exemplu, pentru calculul unor valori, dar nu va fi vizibil;
- *Canvas* – în ce *canvas* va fi afișat elementul;
- *Tab Page* – în cazul *canvas-urilor* de tip *Tab* permite specificarea paginii *canvas-ului* pe care va fi afișat elementul;
- *X/Y Position* – permite specificarea coordonatelor colțului din stânga-sus al elementului relativ la același colț al *canvas-ului*;
- *Width și Height* - setează lățimea și înălțimea *item-ului* curent;

- *Bevel* – permite setarea chenarului *item*-ului curent;
- *Rendered* – la ieșirea *focus*-ului de pe element, acesta își va modifica sau nu aspectul;
- *Show Vertical Scrollbar* – este evident la ce se referă, nu?

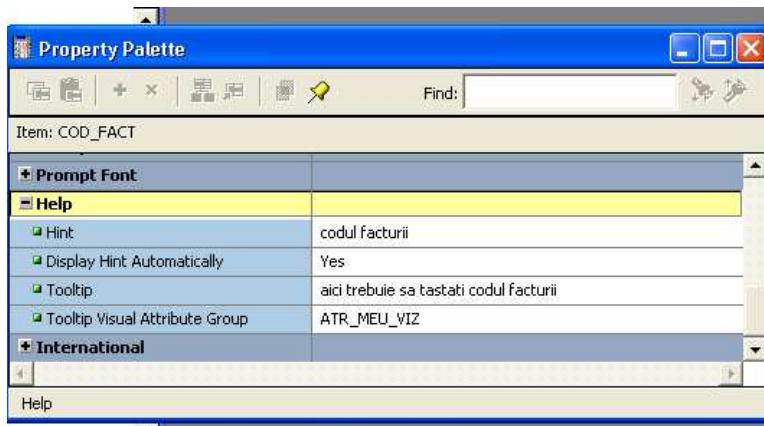
Urmatoarele grupe de proprietăți, *Visual Attributes*, *Color* și *Font* contin proprietăți deja familiare nouă, aşa ca trecem la capitolul *Prompt*, care personalizează apariția etichetei elementului *text* curent:

- *Prompt* – textul afișat ca etichetă;
- *Prompt Display Style* – cum să fie afișată eticheta: doar o dată, lângă prima instanță a elementului, câte o dată pentru fiecare instanță sau să fie ascuns;
- *Prompt Justification* – modul de aliniere al textului;
- *Prompt Attachment Edge* – lângă ce latură a elementului va fi afișată eticheta;
- *Prompt Alignment* – setează modul în care este aliniata eticheta de-a lungul laturii stabilită prin proprietatea anterioară;
- *Prompt Attachment Offset* – distanța dintre *item* și eticheta sa;

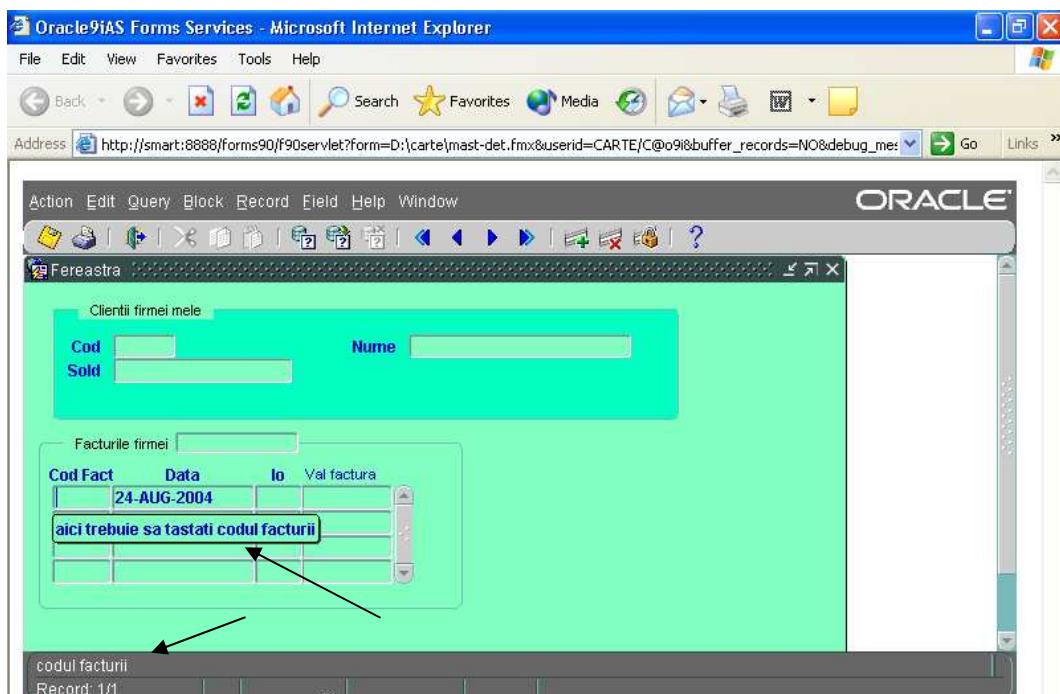
Capitolele *Prompt Color* și *Prompt Font* stabilesc culoarea fundalului pe care va fi afișata eticheta, precum și caracteristicile *font*-ului folosit la afișarea acestuia.

Capitolul *Help* permite programatorului afișarea de mesaje ajutătoare, *case-sensitive*, pentru utilizatorul final al aplicației. Textul care este atribuit proprietății *Help* va fi afișat în bara de stare a aplicației, la rulare, în modul următor: dacă proprietatea *Display Hint Automatically* este lăsată la valoarea implicită, *No*, mesajul ajutător va fi afișat doar când *focus*-ul se află pe element și utilizatorul apasă tasta F1. În cazul în care această proprietate are valoarea *Yes*, mesajul va fi afișat la intrarea *focus*-ului pe elementul curent. Proprietatea *Tooltip* permite definirea unui text care va fi afișat într-o casetă alăturată elementului, când utilizatorul mișcă *mouse*-ul deasupra acestuia. Atributul vizual folosit pentru afișarea *tooltip*-ului se setează la proprietatea *Tooltip Visual Attribute Group*.

Încercați să setați grupul de proprietăți *Help* pentru elementul *cod_prod* la valorile următoare:



Veți obține, în *runtime*, următorul efect:

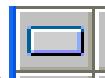


2.7. Butoane (*push buttons*)

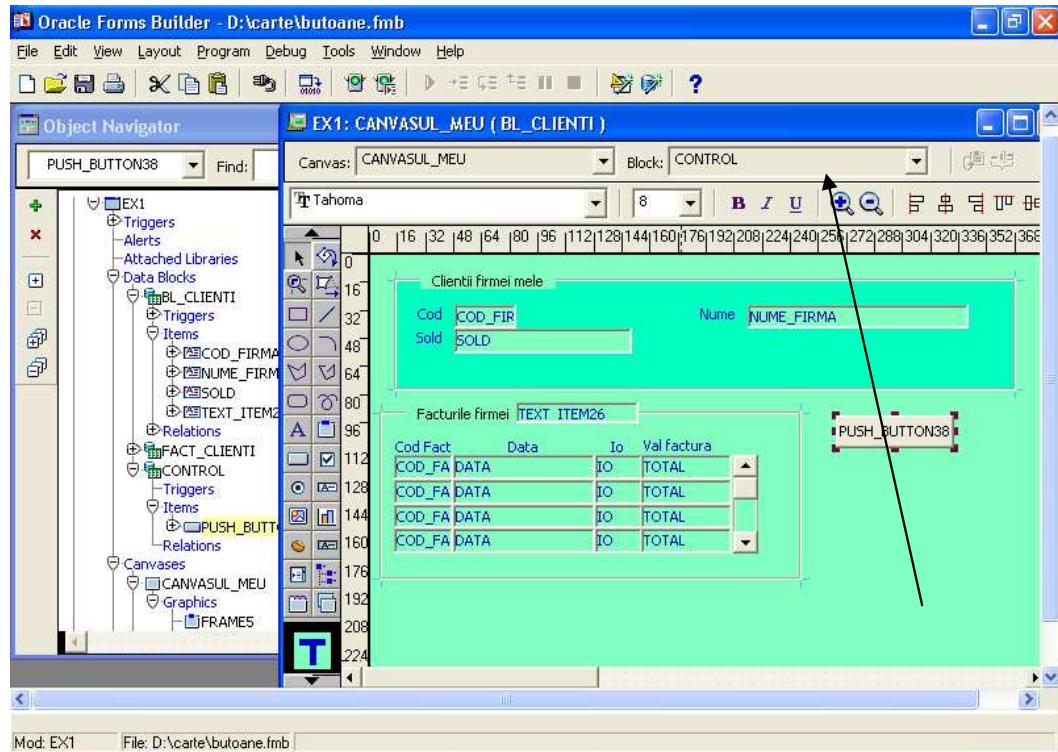
Un buton este o interfață grafică pe care utilizatorul dă *click* pentru inițierea unei acțiuni. Acesta este, în general, reprezentat sub forma unui dreptunghi ce include o etichetă conținând text și/sau imagine; nu poate reține sau afișa valori (cu toate că eticheta poate fi modificată programatic în timpul rulării). În general, butoanele sunt adăugate formularelor pentru obținerea unei funcționalități sporite și a unui mod rapid de acces la cel mai des utilizată operații.

Este recomandată poziționarea butoanelor în blocuri speciale, numite blocuri de control. Să creăm, în aplicația noastră, un bloc nou, pe care să îl denumim *CONTROL*. Există mai multe modalități prin care se poate crea un buton:

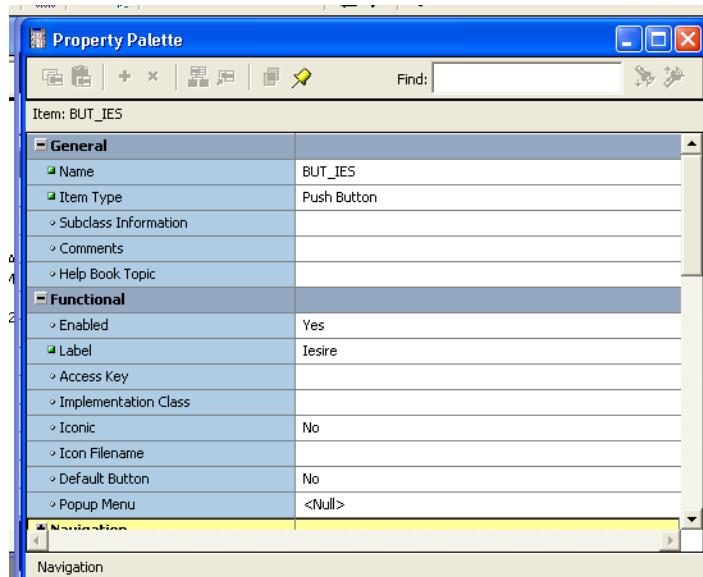
1. convertind un element deja existent la unul de tip buton (în paleta de proprietăți setăm *Item Type* la valoarea *Push Button*);
2. creând, în *Object Navigator*, un element nou, de tipul buton;



3. în *Layout Editor* acționăm butonul din bara de instrumente, apoi prin *drag&drop*, poziționăm obiectul pe locul dorit, având grijă să fie selectat blocul de control:

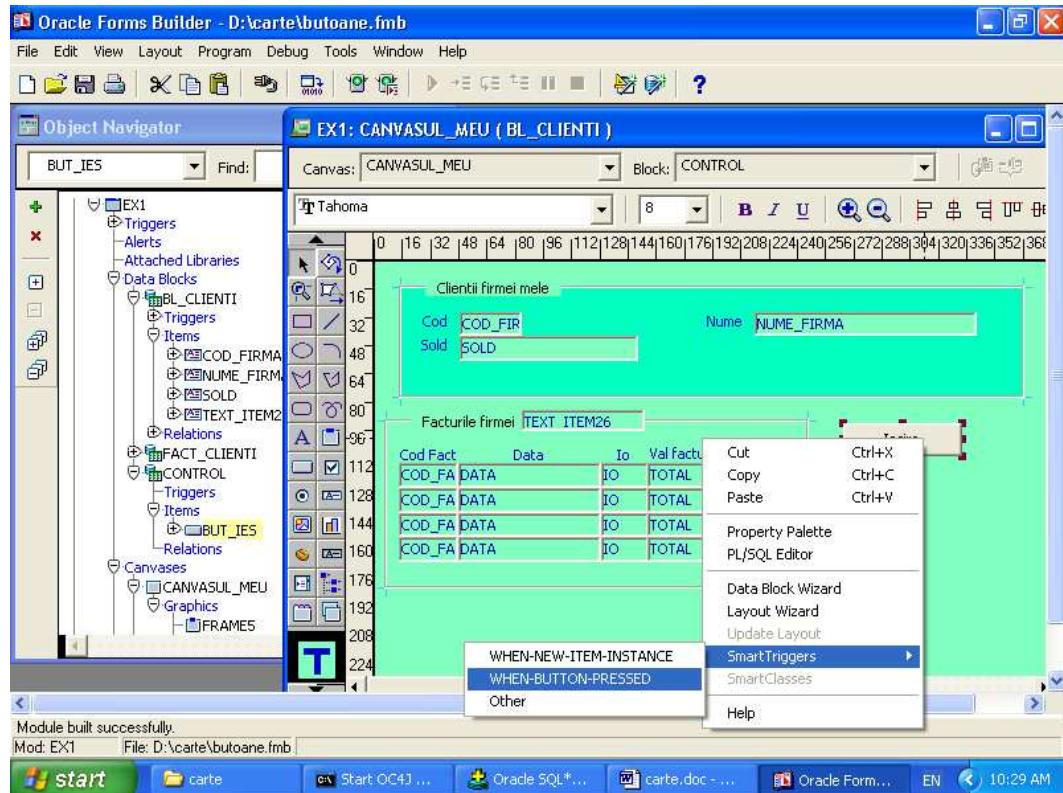


Obiectul nou creat îi modificăm proprietățile în modul deja cunoscut:

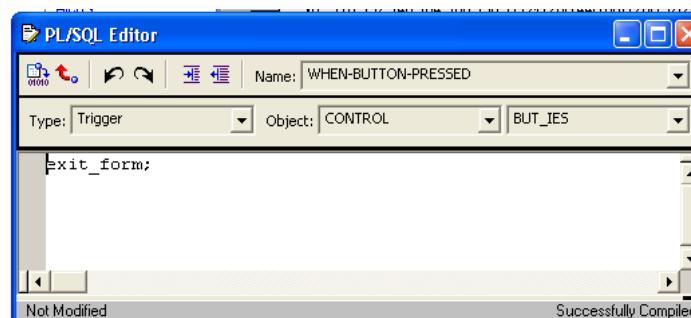


Setând proprietatea *Name*, numim butonul *but_ies* (acesta va fi un buton la a cărui apăsare se va închide formularul), iar eticheta (*Label*) va fi setată la valoarea *Iesire*. Proprietatea *Iconic* permite specificarea modului de afișare a butonului – imagine sau text? În primul caz, la intrarea *Icon Filename* precizăm numele fișierului (fără extensie!) care va fi folosit ca etichetă pentru butonul curent. Putem să stabilim, cu ajutorul proprietății *Access Key* o combinație de taste (de exemplu Alt-J) la a cărei apăsare să fie actionat butonul. Celelalte proprietăți sunt cunoscute, ele fiind aceleași cu cele ale elementelor *text* create într-un capitol precedent.

După crearea părții grafice a butonului, ar trebui să îi acordăm funcționalitate. Pentru aceasta avem la dispozitie *trigger-ii*, secvențe de cod care se declanșează la anumite evenimente. Efectuând *click* dreapta pe buton, din meniul de context care apare alegem intrarea *smart triggers*, reprezentând cel mai des utilizat *trigger-i* pentru obiectul respectiv. La *click* pe buton se va declanșa, după cum ați bănuit deja, *trigger-ul when-button-pressed*. Îl selectăm:



și va fi invocat editorul *PL/SQL*, unde suntem invitați să scriem codul *trigger*-ului:



Am folosit procedura predefinită *exit_form*, care închide formularul curent. Rulăm forma și testăm butonul creat. S-ar putea să deranjeze faptul că, atunci când *focusul* se află pe un câmp care are proprietatea *Data Required* setată la valoarea *Yes*, nu putem acționa butonul până când nu completăm câmpul curent. Acest context va fi neplăcut

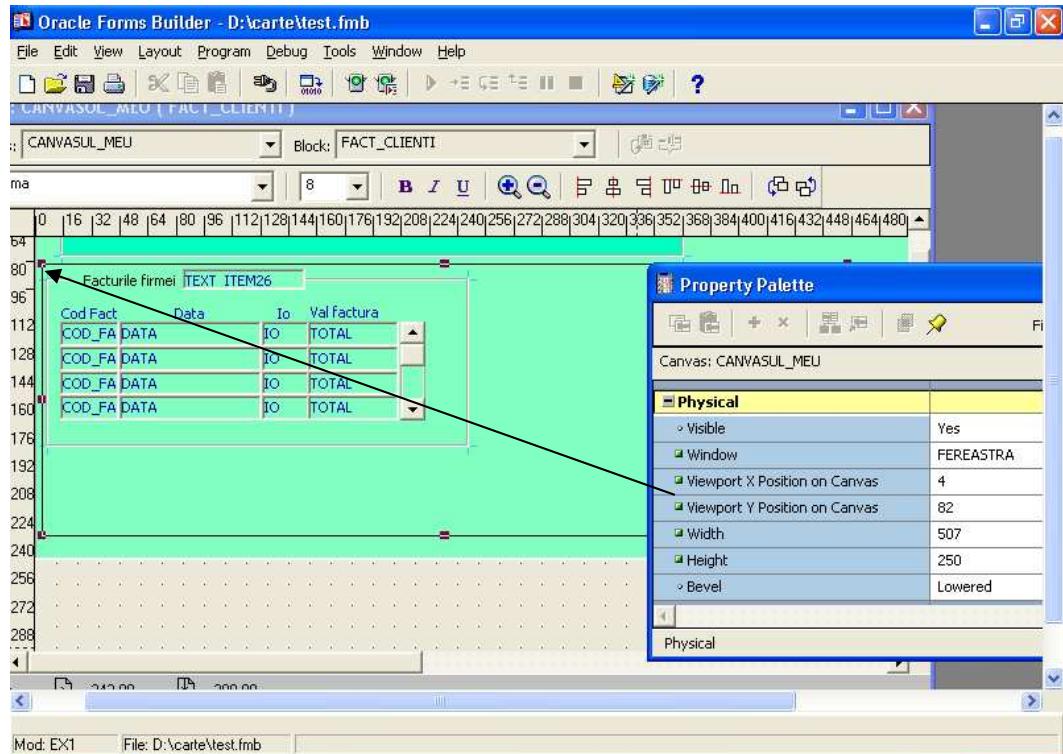
pentru utilizatorul care nu va înțelege de ce trebuie să scrie în câmpul respectiv o valoare arbitrară ca să poată închide forma. Putem salva situația printr-un truc: setăm proprietatea *Mouse Navigate* a butonului la valoarea *No*, acest lucru neînsemnând că butonul nu poate fi accesat cu *mouse*-ul, ci că *focus*-ul rămâne pe câmpul anterior.

2.8. *Canvas-uri de tip suprapus (stacked canvas)*

Clarificasem, într-un capitol anterior, faptul că o fereastră este un *container* pentru obiecte, iar *canvas*-ul este pânza pe care le aşezăm. Pentru a fi afişat în *runtime*, un *canvas* trebuie să fie asociat unei ferestre. Există 4 tipuri de *canvas*-uri. Până acum am folosit doar *canvas*-uri de tipul implicit, numit conținut (*content canvas*). Trebuie respectate câteva reguli în lucrul cu *canvas*-uri de tip conținut:

- acestea ocupă toată suprafața ferestrei;
- este obligatoriu ca fiecare fereastră să conțină cel puțin un *canvas*, iar acesta să fie de tip *content*;
- nu pot fi afişate simultan mai multe *canvas*-uri conținut, comutarea între acestea făcându-se programatic (putem să aducem simultan două *canvas*-uri pe ecran doar dacă le asignăm la ferestre diferite).

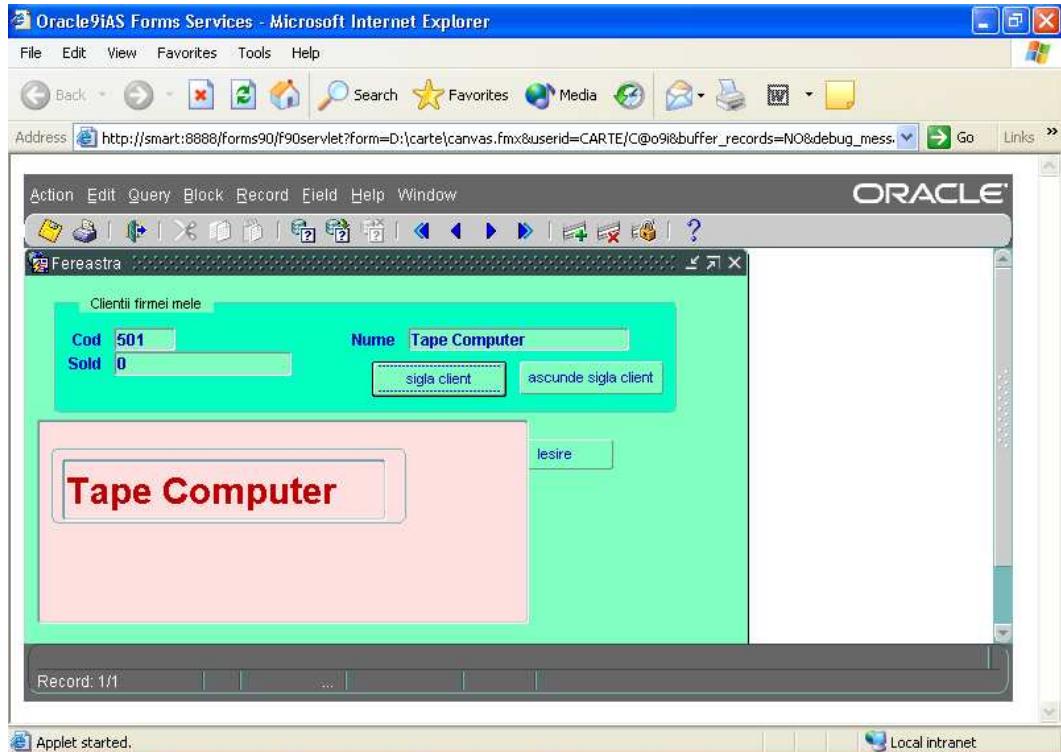
Partea vizibilă în *runtime* este determinată de proprietățile *Viewport X/Y Position on Canvas* (care determină coordonatele, relativ la *canvas*, ale colțului stânga-sus al suprafeței vizibile). La nodul *Functional* al paletei de proprietăți a unui *canvas* găsim proprietățile *Height* și *Width*, care se referă, însă, la dimensiunile *canvas*-ului, nu la cele ale *viewport*-ului.



Adițional *canvas-urilor* conținut, *Forms Buider* oferă alte 3 tipuri de asemenea elemente, care au un comportament diferit, în sensul că pot fi afișate peste *canvas-urile content*, în același timp cu acestea.

- *Stacked canvas* (suprapus) – se afișează în aceeași fereastră cu un *canvas* de tip conținut, peste acesta; dimensiunile lui sunt, în general, mai mici; într-o fereastră pot fi definite mai multe *canvas-uri* suprapuse; este utilizat, în general, pentru afișarea mesajelor de *help*, a informației aditionale, a antetelor cu informație statică;
- *Toolbar canvas* – reprezintă o alternativă pentru meniu; pe un asemenea *canvas* putem așeza mai multe butoane, fiecare cu o funcționalitate specifică;
- *Tab canvas* – permite organizarea și afișarea informației pe pagini separate, care se pot accesa prin *click* pe eticheta acestora; sunt afișate peste un *canvas* de tip conținut.

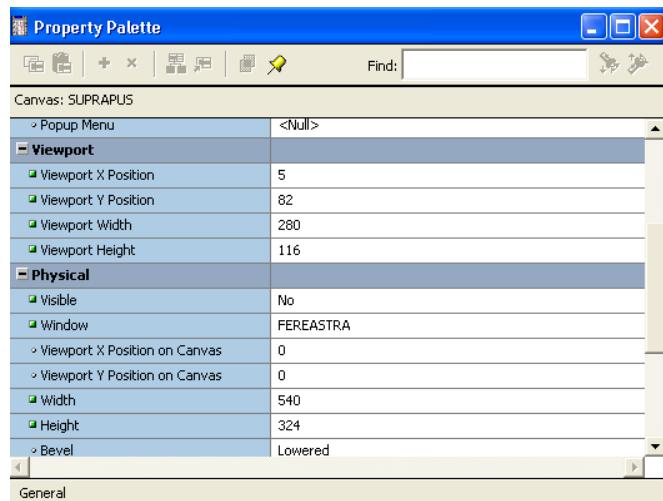
Fiind înarmați cu noțiunile de bază referitoare la *canvas*-uri, să purcedem la crearea unuia de tip *stacked*. Dorim ca, la acționarea unui buton, să fie afișat într-un mod atractiv numele firmei (eventual o siglă, în cazul în care aceasta este disponibilă):



Pași care trebuie urmați pentru crearea noului *canvas* sunt:

1. În *Object Navigator* efectuăm dublu-click pe icon-ul *canvas*-ului *content* în care vrem să creăm *canvas*-ul suprapus;
2. Click pe butonul *Stacked Canvas* din bara de instrumente, pe care îl recunoașteți după eticheta sa:
3. Efectuăm *click* și *drag* cu *mouse*-ul în *canvas*-ul de tip *content*, acolo unde vrem să poziționăm *canvas*-ul suprapus. Atenție: la afișarea unui *canvas* nu trebuie să opturăm cu el câmpul current (pe care se află *focus*-ul). În caz contrar, afișarea nu se face.

4. Invocăm paleta de proprietăți a noului element, pentru a-i seta funcționalitatea:



În primul rând, schimbăm numele acordat implicit *canvas*-ului la valoarea *suprapus*. Proprietățile *Viewport X/Y Position* stabilesc coordonatele colțului din stângă-sus al *canvas*-ului relativ la același colț al ferestrei în care acesta este afișat, în timp ce proprietățile *Viewport Width/Height* permit setarea dimensiunilor parții vizibile la rulare a *canvas*-ului. Proprietatea *Visible* aflată la nodul *Physical* indică dacă acest *canvas* este afișat sau nu la rulare. Setăm această proprietate la valoarea *No*, urmând să afișăm acest *canvas* programatic: creăm, în blocul de control, două butoane, unul pentru afișarea siglei clientului, celălalt pentru ascunderea siglei. Funcționalitatea lor este dată de *trigger*-ii *when-button-pressed*, în care vom folosi două proceduri *built-in* (predefinite):

- *show_view('NumeCanvas')* determină afișarea *canvas*-ului
- *hide_view('NumeCanvas')* produce inhibarea afișării *canvas*-ului.

Ce afișăm în *canvas*? Hotărâsem să afișăm, într-un mod mai spectaculos, numele firmei. Avem în formular această informație în blocul *BL_CLIENTI*, însă un *item* nu poate fi folosit decât într-un singur *canvas*, așa că definim un nou bloc de date, pe care îl numim *STACKED_CLIENTI* și îl asociem tabeliei *CLIENTI*. Conținutul acestui *canvas* va fi afișat în *canvas*-ul numit *suprapus*. Creăm o relație *master-detail* între blocul *BL_CLIENTI* și noul bloc, astfel încât să existe sincronizare între *item*-urile *nume_firma*.

O mică problemă apare: în cazul în care nici o înregistrare nu este adusă în formular, acționarea butonului etichetat *sigla client* va produce un *canvas* vid, a cărui afișare poate fi neplăcută. Putem, atunci, rezolva aceasta problemă în unul din două moduri: afișăm, în acest caz, un mesaj predefinit sau, pur și simplu, "învățăm" butonul să

nu afișeze *canvas*-ul, dacă nu conține informație. În primul din cele două cazuri, procedura *PL/SQL* arată aşa:

```

-- Screenshot 1 (Left):
if :stacked_clienti.nume_firma is null
then :stacked_clienti.nume_firma:='nu e nici o firma!';
end if;
show_view('suprapus');

-- Screenshot 2 (Right):
if :stacked_clienti.nume_firma is null
then null;
else show_view('suprapus');
end if;

```

Bineînțeles, butoanele (*push buttons*) nu reprezintă singura modalitate de a afișa/ascunde un *canvas* de tip suprapus. Codul *PL/SQL* anterior poate fi "convins" să se declanșeze și în alte situații, de exemplu, când *focus*-ul este transferat pe elementul *cod_firma*. Dar pentru a obține un asemenea comportament va trebui să învățăm mai multe despre tipologia *trigger*-ilor.

Putem, totuși, să înlătărim cele două butoane folosite pentru afișarea, respectiv inhibarea siglei firmei cu unul singur, care să își schimbe eticheta în funcție de context. Generalizând afirmația din capitolul anterior (conform căreia un buton își poate schimba programatic, în *runtime*, eticheta), precizăm că orice element își poate schimba programatic, în timpul rulării, proprietățile. Motorul *PL/SQL* din *Forms* are câteva extensii legate de particularitățile elementelor specifice formularelor. Astfel, pentru a accesa proprietățile acestora nu se folosește sintaxa clasica *object.proprietate=valoare*, ci au fost create

- funcția predefinită *GET_TipObject_PROPERTY* (având doi parametri, numele obiectului și proprietatea) și
- procedura predefinită *SET_TipObject_PROPERTY* (cu trei parametri: numele obiectului, proprietatea, noua valoare).

Folosind aceste *built-ins*-uri, să creăm în blocul *CONTROL* un nou buton (în locul celor două) pentru afișarea *canvas*-ului, având numele *af* și eticheta inițială *afișare*. *Trigger*-ul *when-button-pressed* asociat lui va arăta aşa:

```

PL/SQL Editor
Name: WHEN-BUTTON-PRESSED
Type: Trigger Object: CONTROL AF
DECLARE
    eticheta varchar2(20) := get_item_property('control.af',LABEL);
BEGIN
    if eticheta='sigla'
    then
        if :stacked_clienti.nume_firma is null
        then null;
        else
            set_item_property('control.af',LABEL,'ascundere');
            show_view('suprapus');
        end if;
    else
        set_item_property('control.af',LABEL,'sigla');
        hide_view('suprapus');
    end if;
END;

```

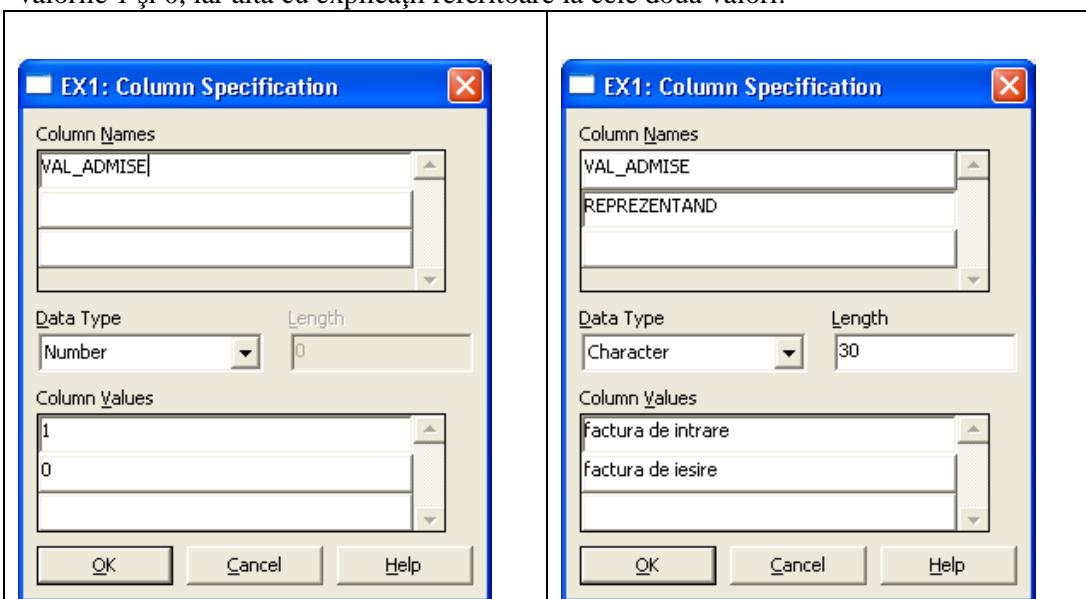
Not Modified Successfully Compiled

2.9. Liste de valori (LOVs)

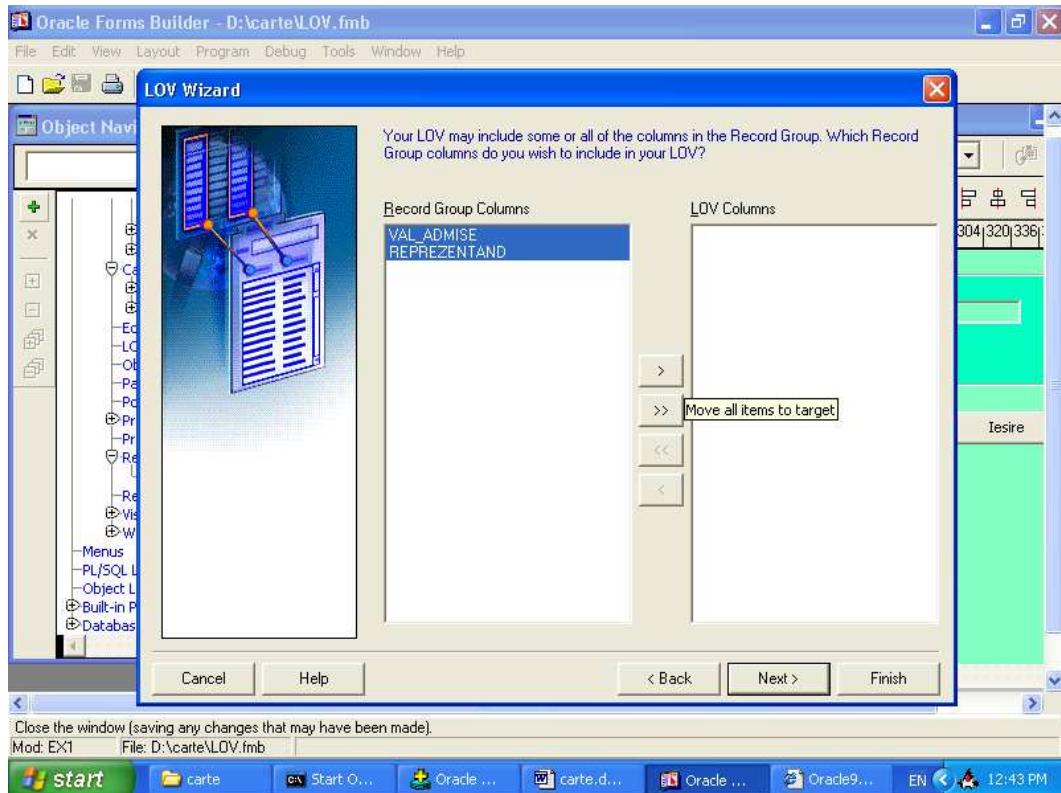
Listele de valori sunt obiecte ale modulului care își deschid propria fereastră la apelarea în timpul rulării aplicației. Ele oferă utilizatorului posibilitatea de a alege un *item* dintr-o listă cu mai multe coloane, listă care poate fi definită static (are aceleași valori indiferent de context) sau dinamic, creată pe baza unei instrucțiuni *select*. Obiectele *LOV* sunt definite la nivel de modul, astfel încât pot fi utilizate oriunde în modulul curent. Tipologia unei liste de valori implică existența a trei elemente:

1. un grup de înregistrări (*record group*) - un obiect *Forms Buider* folosit pentru stocarea unei mulțimi de valori; acesta poate fi static sau dinamic; poate fi creat independent sau odată cu crearea *LOV*-ului; poate fi utilizat pentru definirea uneia sau mai multor liste de valori, fiecare dintre acestea afișând o mulțime specificată de coloane ale grupului;
2. *LOV* – lista însăși, formată din una sau mai multe coloane ale grupului de valori pe care se bazează; permite utilizatorului selectarea unei valori din listă, care va fi automat asignată unui *item*;
3. Un element text – beneficiarul listei de valori, cel căruia *LOV*-ul îi returnează o valoare; un singur *LOV* poate returna valori mai multor *item*-uri simultan; un *LOV* poate fi afișat automat, la intrarea *focus*-ului pe respectivul element sau manual, la apăsarea de către utilizator a combinației de taste *Ctrl-L* sau a opțiunii *Edit → Display List* din meniul implicit *Forms*.

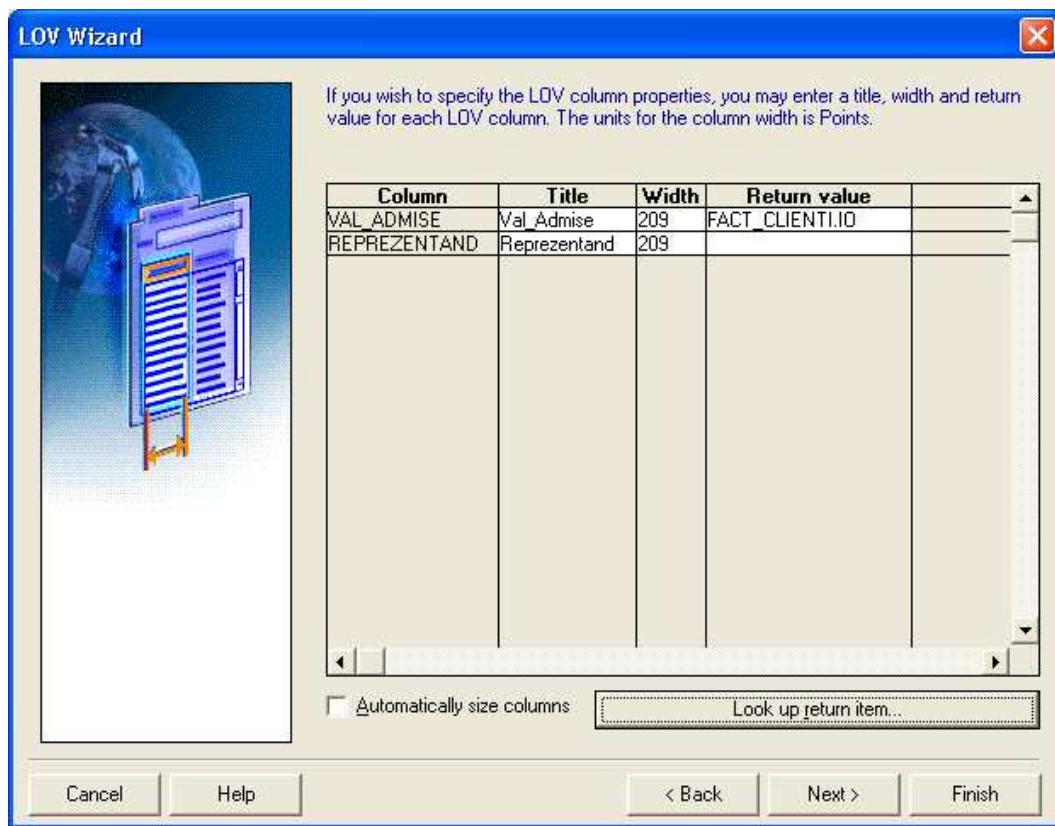
Să exemplificăm crearea și afișarea unei liste de valori statice asociate elementului FACT_CLIENTI.IO. Valoarea pe care o poate lua câmpul *IO* din tabela FACTURI este 1 sau 0, aşa încât dorim ca la completarea *item-ului* corespunzător acestui câmp, utilizatorul să aibă posibilitatea de a alege, dintr-o listă, una din cele două valori. Procedăm întâi la crearea unui grup de valori pe care se va baza *LOV-ul*. În *Object Navigator* selectăm nodul *Record Groups* și acționăm butonul *Create*. Din casetă de dialog care apare alegem opțiunea *Based on Static Values*, după care suntem invitați să completăm valorile ce vor apărea în grup. Să creăm două coloane, una ce va conține valorile 1 și 0, iar alta cu explicații referitoare la cele două valori:



Să numim acest grup *GRUP_IO* și să trecem la crearea listei de valori dorite. Procedăm în același mod, selectând în *Object Navigator* nodul *LOVs* și acționând butonul *Create*. Din casetă de dialog afișată să alegem crearea listei cu ajutorul unui instrument de tip *wizard*, iar în prima fereastră ce apare să specificăm faptul că noua listă va fi creată pe baza unui grup deja existent (cel creat anterior). Dorim ca, la invocarea *LOV-ului*, să fie afișate utilizatorului ambele coloane ale grupului de înregistrări, deci în casetă următoare selectăm ambele elemente:



Pasul următor este foarte important; precizăm căror elemente le vor fi asignate coloanele *LOV*-ului:

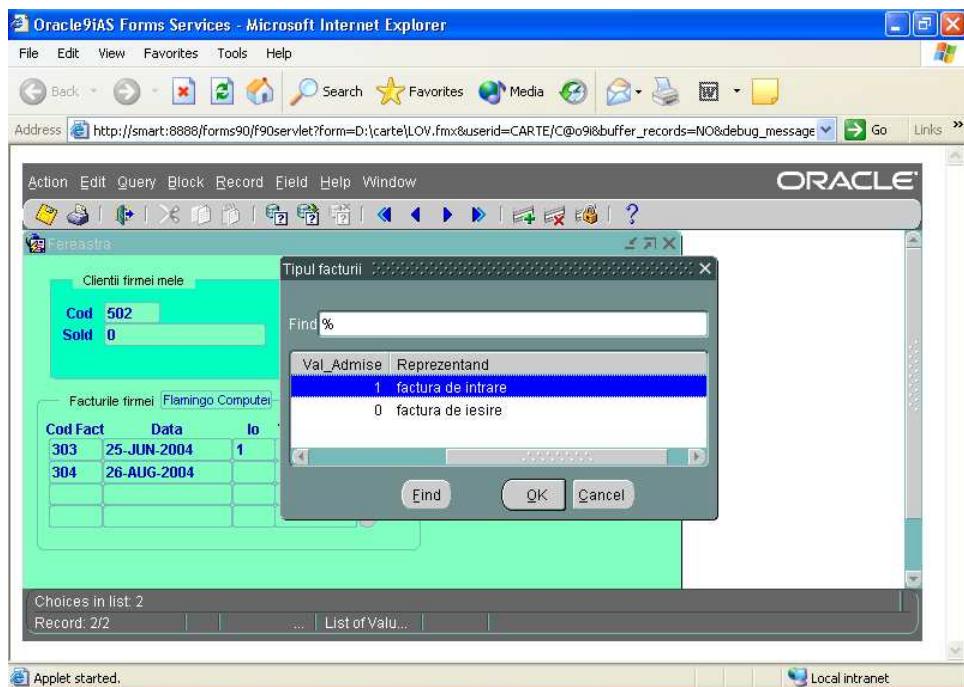


Am actionat butonul *Look up return item* și din lista de valori ce a apărut (cuprindând toate elementele din modul) am selectat *FACT_CLIENTI.IO*, element căruia vrem să îi fie atribuită valoarea coloanei *VAL ADMISE* selectate de utilizator. Cea de a doua coloană a listei este doar informativă, aşa că nu îi asociem nici un element. În caseta următoare dăm un nume ferestrei în care este afișată lista de valori (de exemplu *Tipul facturii*) și putem stabili poziția pe care aceasta va apărea pe ecran. Dând *click* pe butonul *Next* ajungem la pagina *Advanced*, unde putem să îi permitem sau nu utilizatorului să filtreze la afișare lista de valori creată. Pagina finală selectează dintre *item-urile* care primesc valori din listă pe cele care, având *focus-ul* pe ele, pot afișa (în modul specificat anterior) *LOV-ul*. Observăm în *Object Navigator* crearea unui obiect *LOV nou*, pe care să îl denumim *LOV_IO*. Analizând paleta sa de proprietăți, remarcăm faptul că toate elementele setate cu ajutorul instrumentului *LOV Wizard* se regăsesc și pot fi modificate aici.

Analizând paleta de proprietăți a elementului *FACT_CLIENTI.IO* observăm că sub nodul *LOV* a fost modificată automat proprietatea *List of Values*, fiindu-i atribuită valoarea *LOV_IO*. Proprietatea *Validate from List* a fost lasată la valoarea implicită *No*,

permisând astfel utilizatorului și inserarea de alte valori, în afara celor din *LOV*. Noi nu dorim acest lucru, aşa că îl obligăm pe utilizator să aleagă una dintre cele două valori oferite de noi setând această proprietate la valoarea *Yes*.

Rulând aplicația, observam că atunci când *focus-ul* se află pe câmpul *Io* în bara de stare este semnalată prezența unei *LOV* pentru acest element, iar la apăsarea combinației de taste *Ctrl-L* este afișată lista:

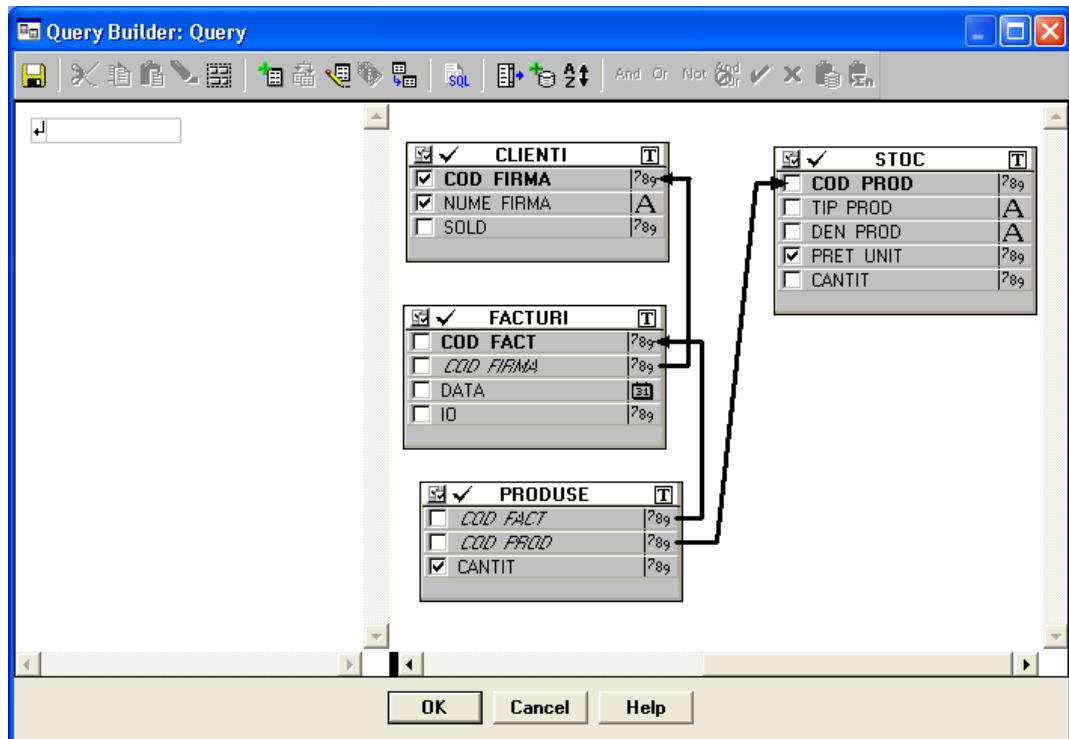


Dublu *click* pe înregistrarea dorită (sau *click* simplu urmat de acționarea butonului *OK*) va atribui *item-ului Io* valoarea selectată din listă. Curiozitatea vă va face să încercați completarea elementului cu o altă valoare, diferită de 1 sau 0, caz în care *Forms Developer* va afișa lista de valori și nu va permite trecerea la alt *item* înainte de completarea corectă a celui curent.

Lista de valori poate fi determinată să apară automat, la intrarea *focus-ului* pe câmpul *Io*, prin setarea proprietății *Automatic Display* aflată la nodul *Functional* al listei.

Am construit astfel un grup de înregistrări statice, pe baza căruia am creat un *LOV* pe care l-am asignat unui element *text*. Să exemplificăm obținerea unui *LOV* dinamic, creat pe baza unei comenzi *select*. Vom selecta, în *Object Navigator*, nodul *LOVs*, apoi vom acționa butonul *Create*. Alegem și de această dată crearea listei cu

ajutorul utilitarului *LOV Wizard*, iar la pasul următor precizăm că lista își va lua înregistrările dintr-un grup nou, bazat pe o cerere (selectăm opțiunea *New Record Group Based on a Query*). Pasul următor al *wizard-ului* cere definirea instrucției *select* care va popula grupul. Putem invoca utilitarul *Built SQL Query* (acționând butonul corespunzător), iar în caseta ce apare selectăm tabelele ce vor participa la instrucție *select*, precum și câmpurile ce vor fi utilizate. Să presupunem că vrem mai mult decât poate acest utilitar: dorim ca grupul de înregistrări să conțină 3 coloane: pe prima să avem codul firmei, pe a doua numele firmei, iar pe a treia suma valorilor tuturor facturilor operate de firma respectivă (indiferent dacă sunt de intrare sau ieșire) – astfel vom avea o imagine a intensității activității firmei respective. Ceea ce permite utilitarul amintit anterior este selectarea celor 4 tabele pentru includerea lor în cerere, depistarea corectă a relațiilor dintre ele și alegerea câmpurilor ce vor participa la rezultat:



Acționarea butonului *OK* va produce o cerere pe care o modificăm manual, conform cerințelor prezentate anterior (modificările operate de noi sunt scrise cu litere mici):



La pașii următori includem toate cele trei coloane ale grupului în *LOV* și returnăm valorile primelor două elemente *BL_CLIENTI.COD_FIRMA*, respectiv *BL_CLIENTI.NUME_FIRMA* (folosind opțiunea *Look up Return Item*). Pentru o afișare elegantă, este recomandată redimensionarea spațiului pe care sunt afișate coloanele grupului. La pasul următor putem da un titlu ferestrei în care va fi afișat *LOV*-ul; următoarea casetă permite specificarea numărului maxim de înregistrări care va fi afișat în fereastra *LOV*-ului și dacă instrucția *select* ce populează lista va fi executată la fiecare afișare sau o singură dată, la prima sa afișare (proprietatea *Refresh Record Group Data Before Displaying LOV*). Bifarea ultimei casete (*Let the user filter records before displaying them*) permite utilizatorului introducerea unui filtru la afișarea listei de valori. Click pe butonul *Next* ne conduce la pasul următor, unde specificăm căror elemente le asignăm lista de valori (de pe ce *item*-uri poate fi aceasta invocată). Să selectăm aici doar *BL_CLIENTI.COD_FIRMA*. Butonul *Finish* încheie procesul de creare a listei de valori, al cărei nume îl vom modifica din *Object Navigator* (să îl numim, de exemplu, *LOV_CLIENTI*). Încheind și această etapă, rămâne doar să rulăm aplicația și să admirăm rezultatele obținute.

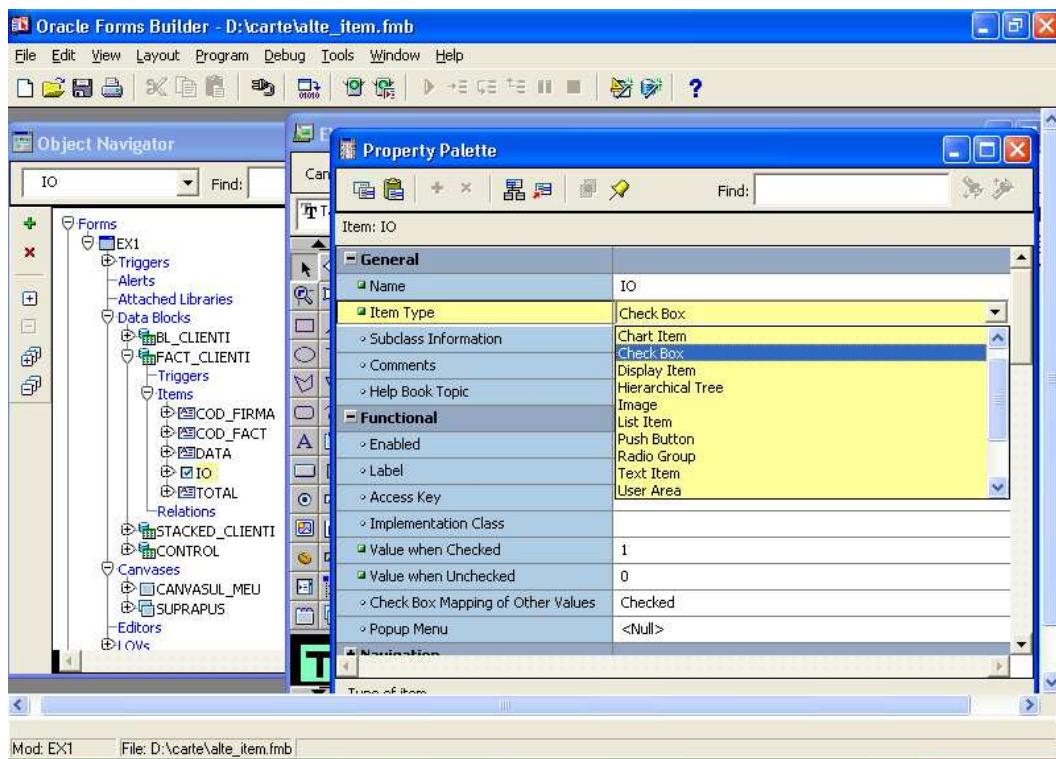
2.10. Alte tipuri de elemente într-un formular

În afara elementelor *text*, un formular poate conține și alte tipuri de elemente utilizate pentru introducerea datelor de către utilizator. Cel mai des folosite sunt casetele de validare (*check boxes*), grupurile de butoane radio (*radio groups*) și elementele listă.

2.10.1. Casete de validare

O casetă de validare este un obiect-interfață ce poate avea una din două stări: bifat sau nebifat. Un astfel de obiect poate fi obținut prin convertirea unui obiect deja existent sau prin crearea unuia nou, în *Layout Editor*, selectând butonul corespunzător:

Să exemplificăm crearea unui obiect casetă de validare folosind prima metodă. Elementul *FACT_CLIENTI.IO* poate lua una din două valori (1 sau 0) și se pretează, deci a fi transformat într-o casetă de validare. Acționând paleta sa de proprietăți, o modificăm pe cea care definește tipul elementului, alegând *check box*:



Unele dintre proprietățile acestui nou obiect sunt comune cu cele ale unuia de tip *text*. La nodul *Functional* se află, totuși, câteva proprietăți specifice:

- *Value when Checked* – permite specificarea unei valori corespunzătoare stării bifat. Atunci când o valoare egală cu cea specificată aici este încărcată din tabelă și atribuită casetei, aceasta va apărea bifată. Reciproc, atunci când utilizatorul va bifa caseta, valoarea asignată acestei proprietăți va fi transferată coloanei pe care o mapează caseta.
- *Value when Unchecked* – este echivalentul proprietății anterioare pentru cazul când caseta rămâne nebifată.
- *Check Box Mapping of Other Values* – se folosește în cazul în care pot fi încărcate sau asignate casetei alte valori, în afara celor specificate prin cele două proprietăți anterioare și permite definirea modului în care orice altă valoare va fi procesată (*Not Allowed*, *Checked* sau *Unchecked*)

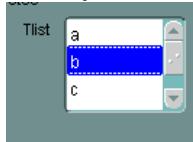
2.10.2. Elemente listă

Un element listă este un obiect care afișează o mulțime predefinită de elemente, fiecare corespunzând unei valori specificate de programator. Există trei tipuri de liste și în nici unul dintre cele trei nu se pot face selectări multiple:

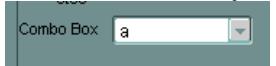
- *Poplist* – apare ca un câmp de tip text având atașat un buton reprezentat printr-o săgeată. Acționarea butonului va determina expandarea listei, sub forma:



- *Plist* – apare ca un dreptunghi care afișează elementele listei. Când suprafața de afișare nu este suficientă, îi va fi atașată automat o bară de derulare:

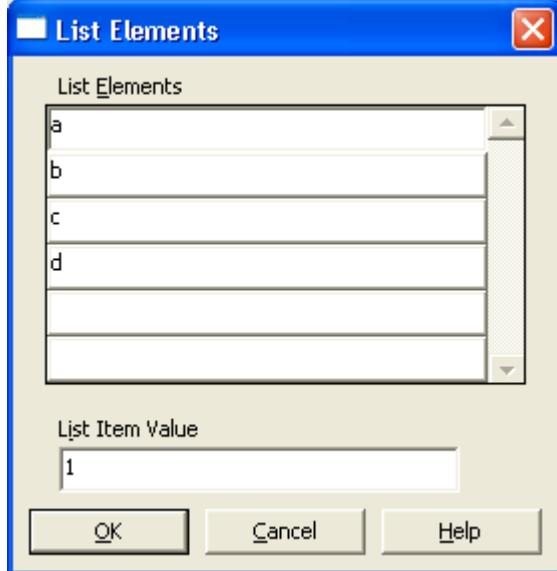


- *Combo Box* – apare ca un element *text* cu o săgeată în jos atașată, a cărei apăsare determină afișarea elementelor listei. Spre deosebire de listele de tip *poplist*, elementele *combo box* permit introducerea de către utilizator și a altor



valori, în afara celor din lista de valori predefinite:

O proprietate importantă, comună tuturor tipurilor de liste, este *Elements in List*, locul unde se definesc elementele listei și valorile corespunzătoare acestor elemente. Click pe această proprietate va afișa o casetă de editare a acestor valori:



În partea superioară a casetei vor fi editate elementele listei (valorile afișate în *runtime*), iar în partea inferioară va fi definită, pentru fiecare element al listei, o valoare asociată lui. Celelalte proprietăți ale listelor sunt cunoscute din capitolele anterioare.

2.10.3. Grupuri de butoane radio

Un grup radio este un obiect format (din punct de vedere vizual) dintr-o mulțime de butoane radio, reprezentând toate valorile posibile pe care le poate lua acest element. Dintre acestea, unul și numai unul poate fi bifat (utilizatorului i se oferă posibilitatea unei selectări unice din mulțimea de valori permise). Acest tip de element reprezintă o alternativă la casetele de validare (în cazul când mulțimea de valori permise are două elemente) sau la liste (în cazul când această mulțime are mai puțin de 4 elemente). Pentru mulțimi de valori cuprinzând mai mult de patru elemente, este recomandată folosirea listelor, afișarea unor grupuri de butoane radio atât de numeroase aglomerând inutil suprafața de afișare. Valoarea grupului de butoane radio poate fi specificată de utilizator (în timpul rulării aplicației), prin proprietatea *Initial value* sau programatic. Se pot executa, de asemenea, interogări folosind acest tip de element.

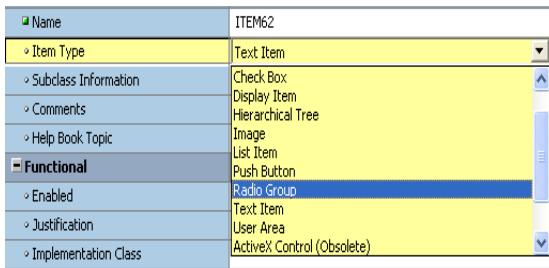
Un grup de butoane radio poate fi creat în oricare din modurile deja cunoscute de creare a elementelor *Forms*:

- Convertind un element deja existent la tipul *radio group*;

- Creând un element nou în *Layout Editor*, caz în care acționăm, în bara de instrumente, butonul corespunzător:



- Utilizând, în *Object Navigator*, icon-ul *Create* ; această acțiune va avea ca efect crearea unui element nou, de tipul (implicit) *text*, care va fi transformat în grup radio prin setarea corespunzătoare a proprietății *Item Type*. Creând astfel grupul de butoane radio, acesta nu va fi afișat în *Layout Editor* până în momentul când vom seta proprietatea *Canvas*, specificând astfel *canvas-ul* în care elementul va apărea. Butoanele vor fi poziționate în colțul din stânga sus al formularului, pasul următor fiind repoziționarea lor .



2.11. Declanșatori (*trigger-i*)

Declanșatorii reprezintă cea mai importantă modalitate de a adăuga funcționalitate unui formular sau de a-i redefini funcționalitatea implicită. Un declanșator are trei elemente definitorii:

- Tipul – definește evenimentul care va declanșa *trigger-ul*;
- Codul – definește acțiunea efectuată de *trigger* (este scris în *PL/SQL*);
- Domeniul – definește nivelul la care sunt ascultate evenimentele de către respectivul *trigger*.

Numele *trigger-ilor* este format din două componente, separate prin cratimă: prefixul, care determină când (în raport cu evenimentul generator) se declanșează *trigger-ul* și evenimentul, care determină tipul de acțiune ce va declanșa *trigger-ul*.

În funcție de prefix, *trigger-ii* se împart în următoarele categorii:

- *When-eveniment* – procesării asociate implicit evenimentului îi sunt adăugate acțiunile definite de *trigger* (de exemplu, *When-validate-item* se declanșează imediat ce *Forms* validează datele dintr-un câmp; *when-new-item-instance* – se declanșează când *focus-ul* părăsește câmpul curent)
- *On-eveniment* – funcționalitatea definită implicit în cazul declanșării acestui eveniment este înlocuită cu acțiunile definite în corpul *trigger-ului* (de exemplu, *On-logon* se declanșează în locul conectării la baza de date *Oracle* și este folosit, de exemplu, când se dorește conectarea programatică la altă bază de date);
- *Pre-eveniment* – se declanșează chiar înainte de eveniment; aceștia preced *trigger-ii when-eveniment* sau *on-eveniment* și sunt folosiți în scopul pregătirii obiectelor și a datelor pentru evenimentul ce se va declanșa;
- *Post-eveniment* – se declanșează imediat după *trigger-ii when-eveniment* sau *on-eveniment* și este folosit, de obicei, pentru a valida datele sau pentru a efectua anumite acțiuni bazate pe evenimentul ce doar a avut loc;
- *Key* – se declanșează la apăsarea de către utilizator a tastei sau a combinației de taste căreia îi sunt asignați (amintiți-vă că cele mai multe interfețe grafice oferă utilizatorului mai multe moduri pentru accesul la același obiect).

Există, în *Forms Developer*, peste 100 de declanșatori predefiniți, fiecare identificat unic prin numele său, format întotdeauna conform sintaxei: *prefix-eveniment*.

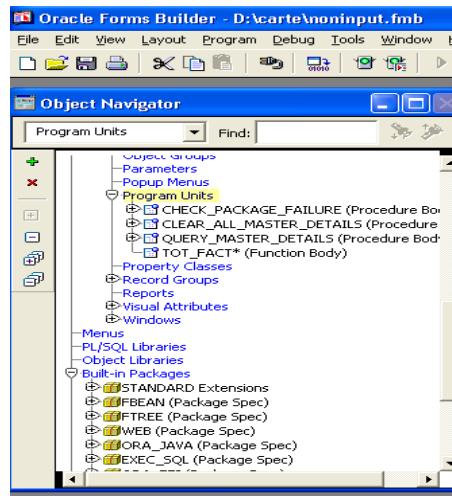
Evenimentele ce au loc la nivelul bazei de date pot declanșa, de asemenea, anumiți *trigger-i*, dar aceștia sunt diferiți de cei din *Forms*. Există, deci, *trigger-i* la

nivelul bazei de date și *trigger*-i la nivel de formular, aceștia tratând evenimente specifice formularului.

Forms Builder-ul suportă declanșatoare definite de programator, dar acestea se vor executa doar atunci când sunt apelate dintr-un alt *trigger* sau dintr-o unitate de program, folosind sintaxa *execute NumeTrigger*.

Codul unui *trigger* definește acțiunile care vor fi efectuate atunci când acesta se declanșează. Sintaxa este aceeași cu unui bloc *PL/SQL*; dacă lipsesc partea de declarații și cea de tratare a erorilor, nu este necesară includerea corpului *trigger*-ului între *begin* și *end*. Secvențele de cod *PL/SQL* scrise pentru aplicațiile *Forms* cuprind adesea apeluri de subprograme.

Subprogramele PL/SQL folosite în aplicațiile *Forms* sunt: funcțiile și procedurile predefinite (*Built-in*) sau cele definite de utilizator. Primele se găsesc în *Object Navigator* la nodul *Built-in Packages*, iar următoarele la nodul *Program Units*. Observați în figura de mai jos existența funcției *tot_fact*, creată anterior de noi. Funcțiile și procedurile *built-in* sunt grupate în pachete, conform funcționalității lor și se apelează folosind sintaxa: *NumePachet.NumeProcedura*. Pentru cele aflate în pachetul *standard* numele acestuia poate fi omis.



Theoretic, aceste *built-ins*-uri pot fi folosite în orice subprogram *PL/SQL* sau *trigger*. Practic, unele dintre acestea oferă funcționalități care nu sunt permise în anumiți *trigger*-i. Acesta este motivul pentru care *Built-ins*-urile au fost împărțite în două categorii:

- *nerestricționate* – nu afectează navigația logică sau fizică și pot fi apelate în orice *trigger* sau subprogram;
- *restricționate* – afectează navigația în formular și pot fi apelate doar în *trigger-ii* care nu generează navigație internă. De exemplu, apelul unui *built-in* restricționat (cum ar fi *go_block('numeBloc')*;) într-un *trigger* navigațional (ca *when-new-item-instance*, explicitat mai jos) se compilează cu succes, dar generează eroare la rulare.

Să ne amintim că am folosit până acum câteva dintre *built-ins*-urile definite în pachetul *standard*: *show_view*, *hide_view*, *exit_form*, *get_item_property*, *set_item_property*. *Forms Builder* permite un mod rapid de inserare a prototipului unui *built-in* în secvența de cod curentă, astfel:

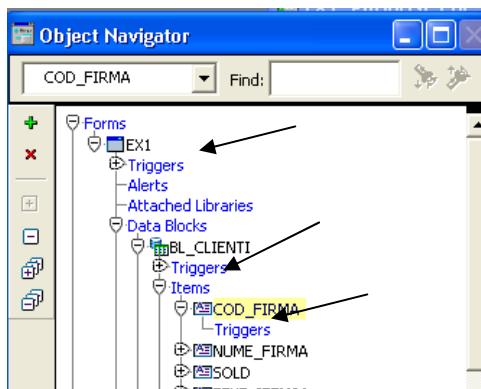
1. poziționăm cursorul în editorul *PL/SQL* exact în locul în care dorim să fie inserat apelul de subprogram;
2. expandăm nodul *Built-in Packages* din *Object Navigator* și selectăm procedura sau funcția pe care vrem să o utilizăm;
3. selectăm *Edit → Paste Name* sau *Edit → Paste Arguments* din meniul *Forms Builder* (prima opțiune va genera inserarea doar a numelui, pe când cea de a doua va include atât numele, cât și argumentele subprogramului);
4. prototipul *built-in*-ului va fi copiat la poziția curentă a cursorului, iar utilizatorul va continua, dacă este cazul, cu completarea numelor argumentelor.

Variabilele folosite pentru stocarea valorilor în aplicațiile *Forms* sunt de două tipuri:

- Variabile *PL/SQL* – se definesc în secțiunea declarativă și sunt disponibile până la încheierea execuției blocului curent; nu sunt prefixate de “`:`”; pot fi definite în pachete, caz în care se apelează prin *NumePachet.NumeVariabila* și sunt accesibile din toți *trigger-ii* care utilizează respectivul pachet;
- Variabile *Forms Builder* - sunt gestionate de *Forms Builder* și sunt văzute de *PL/SQL* ca variabile externe; pentru deosebirea lor de obiectele *PL/SQL* se impune prefixarea cu semnul `:` (exceptând cazul în care numele lor este transmis ca parametru unui subprogram). Există patru tipuri de variabile *Forms Builder*:

Tipul variabilei	Domeniul	Scopul utilizării	Sintaxa
<i>Item</i> (<i>text, list, radio group etc.</i>)	Modulul curent și meniul atașat	Interacțiunea cu utilizatorul	<i>:NumeBloc.NumeItem</i>
Variabilă globală	Toate modulele din sesiunea curentă	Stocarea valorilor pe parcursul unei sesiuni de lucru	<i>:GLOBAL.NumeVariabila</i>
Variabilă sistem	Formularul curent și meniul atașat	Controlul mediului de lucru	<i>:SYSTEM.NUME_VARIABILA</i> (numele acestor variabile se scriu cu majuscule)
Parametri	Modulul curent	Transferul valorilor spre și de la modulul curent	<i>:PARAMETER.Nume</i>

Domeniul de acțiune al unui *trigger* (numit în engleză *trigger scope*) este determinat de poziția sa în ierarhia obiectelor din *Object Navigator*:

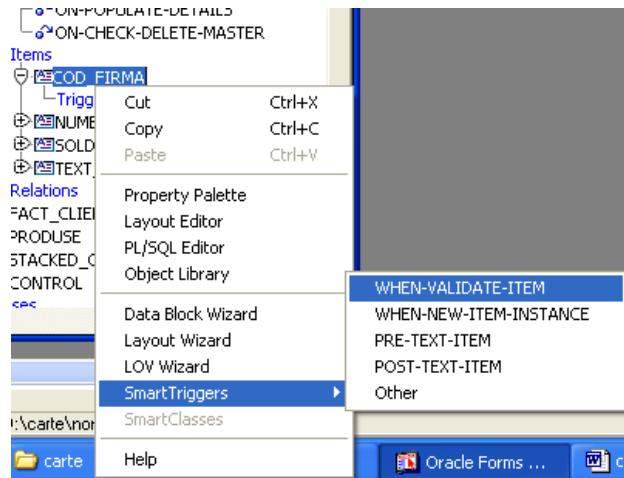


După cum vedeați în figura anterioară, există *trigger-i* la nivel de: modul, bloc și element. Unii *trigger-i* nu pot fi definiți sub un anumit nivel. Un *trigger* răspunde unui

eveniment asociat obiectului care deține acel *trigger* sau oricărui alt obiect care este deținut de primul. Putem avea, în formular, mai mulți *trigger*-i cu același nume pe niveluri diferite; în mod implicit, pe un obiect se declanșează un singur *trigger* de același tip: cel deținut de obiectul respectiv sau, în cazul în care nu există, cel aflat pe nivelul ierarhic superior cel mai apropiat. Acest comportament este definit de proprietatea *Execution Hierarchy*, setată la valoarea *Override*. Alte valori posibile ale acestei proprietăți sunt: *Before* (caz în care *trigger*-ul aparținând obiectului curent se va declanșa înaintea oricărui de același tip aflat pe nivelul superior cel mai apropiat) și *After* (caz în care *trigger*-ul curent se va declanșa după oricare altul de același tip aflat pe nivelul superior cel mai apropiat).

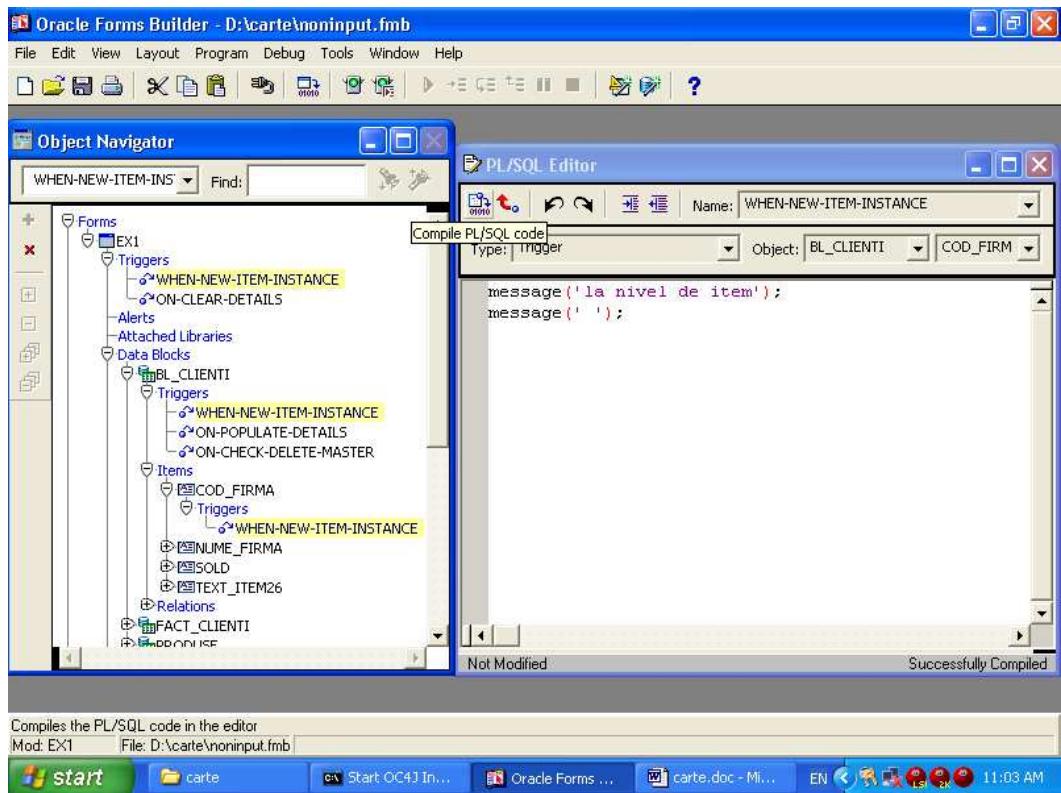
Cu cât un *trigger* este mai sus, cu atât el ascultă mai multe evenimente. Să luăm ca exemplu *trigger*-ul *when-new-item-instance*, care se declanșează la părăsirea de către *focus* a *item*-ului curent și navigarea către un alt *item*. Putem crea acest *trigger* (și oricare altul) în unul din mai multe moduri:

- Selectând din meniul de context al obiectului căruia vrem să îi asociem *trigger*-ul intrarea *Smart triggers*, care va afișa un *LOV* cu *trigger*-ii cel mai des declanșati de obiectul curent:



- Selectând, în *Object Navigator*, nodul *Trigger* al obiectului căruia vrem sa îi atașăm *trigger*-ul și acționând butonul *Create*.

Rezultatul oricărei dintre aceste acțiuni va fi afișarea instrumentului *PL/SQL Editor*, unde vom scrie secvența de cod ce va fi rulată la declanșarea *trigger*-ului:



Built-in-ul message('text') are ca efect afișarea mesajului în bara de stare a ferestrei *Forms*-ului. Apelarea de două ori consecutiv a acestei proceduri are ca efect aglomerarea mesageriei și apariția pe ecran a unei casete de alertă, rezultat mult mai vizibil utilizatorului. După ce ați scris codul asociat *trigger*-ului, nu uitați să acționați butonul care îl compilează și corectați eventualele erori. Putem crea *trigger*-ul *when-new-item-instance* la nivel de element, bloc sau modul. Dacă vom scrie un singur *trigger* de acest tip pentru item-ul *cod_firma*, de exemplu, el se va declanșa doar atunci când *focus*-ul părăsește acest *item*. Dacă, însă, *trigger*-ul va fi asociat blocului *bl_clienti*, el va fi declanșat de evenimentul de părăsire a oricărui element aflat în acest bloc, iar dacă acest *trigger* va fi asociat modulului, va fi declanșat de mutarea *focus*-ului de pe oricare dintre elementele formularului. Putem, de asemenea, defini trei *trigger*-i *when-new-item-instance*, la toate cele trei niveluri (vezi figura anterioară), caz în care ordinea de declanșare este stabilită de proprietatea *Execution Hierarchy* detaliată anterior. Încercați toate modalitățile de definire a acestor *trigger*-i, apoi rulați, în fiecare caz, aplicația și observați rezultatele!

Am clasificat *trigger*-ii după mai multe criterii, dar cel mai important este cel referitor la contextul în care se declanșează. Astfel, există:

- *trigger-i* de validare – se declanșează când *Forms Developer* validează date dintr-un câmp sau înregistrare. Există conceptul de *unitate de validare*, reprezentând volumul maxim de date pe care operatorul le poate introduce înainte de a se efectua validarea lor (verificarea corectitudinii). Implicit, unitatea de validare este *item-ul*, ceea ce înseamnă că *Forms Developer* inițiază procesul de validare imediat ce se încearcă navigarea (programatic sau de către operator) către alt *item*;
- *trigger-i* tranzacționali – apar ca răspuns a unor evenimente generate de interacțiunea dintre formular și baza de date;
- *trigger-i* de interogare – se declanșează înainte (*pre-query*) sau după (*post-query*) ce operatorul sau aplicația execută o cerere;
- *trigger-i* navigaționali – permit controlul asupra poziției *mouse-ului*, definind, de asemenea, acțiuni ce se vor declanșa atunci când *focus-ul* intră sau părăsește un obiect;
- *trigger-i de procesare a blocului de date* – se declanșează la operații de interacțiune a formularului cu baza de date, creare de înregistrări noi, eliberare a celor existente în bloc;
- *trigger-i master-detail* – au fost prezentați detaliat la crearea unui modul *master-detail*; aceștia sunt: *on-check-delete-master*, *on-clear-detail*, *on-populate-detail*;
- *trigger-i interface-event* – se declanșează la evenimente specifice interfeței utilizator: *when-mouse-click*; *when-mouse-down*, *when-button-pressed*, *when-list-changed*, etc.
- *trigger-i* pentru interceptarea erorilor – definesc acțiunea ce trebuie întreprinsă la apariția unui mesaj sau a unei erori (exemplu: *On-Error*, *On-Message*).

2.11.1. Trigger-i de interogare (*query triggers*)

Evenimentele asociate interogărilor asupra tabelelor de bază ale blocurilor de date pot fi controlate prin intermediul *trigger-ilor* de interogare, care permit personalizarea modului de lucru *query*. Pentru a înțelege tipologia acestor *trigger-i*, ar trebui să cunoaștem modul în care are loc procesul de interogare a bazei de date de către formular. Când, într-un bloc de date, este inițiată o cerere (de către operator sau programatic) formularul parcurge următoarele etape:

1. aflat în modul *enter-query*, *Forms Developer* declanșează *trigger-ul pre-query*, dacă acesta există; în cazul eșuării, interogarea este abandonată; dacă *trigger-ul* se termină cu succes, se trece la pasul următor;

2. *Forms*-ul construiește, apoi execută fraza *select* utilizată pentru interogarea tabelelor bazei de date; aceasta este creată folosind filtrele definite în blocul de date, filtre care sunt introduse de utilizator sau/și de *trigger*-ul *pre-query*;
3. o linie întoarsă de cerere este încărcată într-o înregistrare nouă, care va fi marcată ca validă;
4. este declanșat *trigger*-ul *post-query*; atenție: acest *trigger* se declanșează pentru fiecare înregistrare în parte și este folosit, în general, pentru popularea elementelor *non based-table*, pentru efectuarea de calcule statistice. Dacă acest *trigger* eșuează, înregistrarea curentă este abandonată și eliminată din blocul de date. Se revine la pasul 3.
5. dacă *trigger*-ul se termină cu succes, vor fi validate modificările efectuate în înregistrarea curentă.

În afara celor doi *trigger*-i prezenți anterior, specifici modului de funcționare *query* al formularului, se pot declanșa și alți *trigger*-i, dacă proprietatea *Fire in Query Mode* asociată lor este setată la valoarea *Yes*. În cazul când un *trigger* poate fi declanșat în ambele moduri (*normal* și *query*), este util de detectat starea în care se află formularul; în funcție de aceasta, vor fi efectuate acțiuni diferite. Variabila sistem *mode* permite detectarea modului de lucru în care se găsește formularul, având una din următoarele trei valori:

Valoarea variabilei :SYSTEM.MODE	Semnificație
<i>NORMAL</i>	Formularul se găsește în modul de procesare <i>normal</i>
<i>ENTER-QUERY</i>	Formularul este în modul <i>enter-query</i> , se așteaptă introducerea criteriului pentru interogare
<i>QUERY</i>	Formularul se află în etapa de procesare a înregistrărilor aduse de pe <i>server</i>

Să presupunem că dorim să creăm propriul buton pentru lansarea de cereri asupra blocului *bl_clienti*. Vom crea un buton, numit *cerere*, care poate fi poziționat în orice *canvas*, dar va trebui să aparțină blocului asupra căruia se va executa interogarea. Motivul este acela că, într-un *trigger* care se declanșează în modul *query* nu avem voie să folosim anumite *build-ins*-uri, printre care și cele care execută navigarea (*go_block*, *go_record*, *next_block*, *next_record* etc.). În *trigger* vom folosi procedurile predefinite *enter_query*,

pentru comutarea în modul *query* și *execute_query*. Abandonarea modului *query* și comutarea la modul *normal* se face cu ajutorul *build-in*-ului *abort_query*. Dacă butonul creat va fi în alt bloc decât cel care se dorește a fi interogat, acționarea lui înseamnă schimbarea *focus*-ului pe blocul căruia îi aparține butonul și interogarea acelui bloc. Creăm, deci, butonul *cerere* în blocul *bl_clienti*, iar secvența de cod *PL/SQL* a *trigger*-ului *when-button-pressed* va fi următoarea:

```

EX1: BUTOANE (BL_CLIENTI)
PL/SQL Editor
Name: WHEN-BUTTON-PRESSED
Type: Trigger Object: BL_CLIENTI CERERE
if :SYSTEM.MODE='NORMAL' then
  set_item_property('cerere',label,'executa cerere');
  enter_query;
else
  set_item_property('cerere',label,'introdu cerere');
  execute_query;
end if;

```

2.11.2. Trigger-i de validare

Forms Developer execută un proces de validare implicit, la diferite niveluri, pentru a se asigura că înregistrările și elementele individuale respectă anumite reguli (definite în paletele de proprietăți ale elementelor, în *trigger*-i etc). *Forms*-ul declanșează fenomenul de validare la nivel de:

- element (*item*) – fiecare element are un *status* care indică validitatea acestuia. Dacă un *item* a fost modificat și nu este încă marcat ca valid (validat), *Forms* efectuează o validare a sa, verificând dacă valoarea este conformă cu condițiile impuse prin paleta de proprietăți a elementului (*format mask*; *required*; *data type*; *range*; *validate from list*). Aceste verificări preced declanșarea *trigger*-ului *when-validate-item*.
- înregistrare (*record*) – la încercarea (inițiată programatic sau de către utilizator) de a părăsi o înregistrare, este verificat *status*-ul acesteia. În cazul când nu a fost validată, este verificat *status*-ul fiecărui *item*, apoi este declanșat un eventual *trigger* *when-validate-record*. După ce toate aceste etape sunt parcurse, înregistrarea este marcată ca validă;

- bloc sau modul – toate înregistrările aflate sub acest nivel vor fi validate (de exemplu, la efectuarea unui *commit*).

Unitatea de validare definește maximul de date ce pot fi introduse înainte de declanșarea automată a procesului de validare. Implicit, aceasta este setată la nivel de *item*, dar poate fi definită, de asemenea, la nivel de *record*, *block* sau *form*.

Procesul de validare se declanșează la navigarea din unitatea de validare (programatic, printr-un *trigger*, sau de către utilizator) sau la rularea *built-in-ului enter*. Acesta este un subprogram creat pentru forțarea imediată a validării la nivelul la care se află definit. De exemplu, crearea unui buton pentru care *trigger-ul when-button-pressed* invoca *built-in-ul enter* va efectua validarea imediată a elementului pe care se află *focus-ul*.

Odată cu aducerea de pe *server* a înregistrărilor ce sunt folosite în formular, *Forms-ul* încarcă adițional câteva pseudo-coloane (numite *flag-uri interne*), cum ar fi *row_id* (un număr ce definește unic înregistrarea în baza de date) și *valid* (reprezentând *status-ul* obiectului curent). Astfel, *Forms-ul* urmărește cu ușurință acțiunile efectuate de utilizator. Pentru a determina dacă validarea datelor trebuie sau nu efectuată, *Forms-ul* utilizează *statutul* înregistrării sau al *item-ului* curent. Astfel, statutul unui *item* poate avea una din valorile:

<i>Status</i>	<i>Explicație</i>
<i>NEW</i>	La crearea unei înregistrări noi, fiecărui <i>item</i> îi este acordat acest <i>status</i> .
<i>CHANGED</i>	Un <i>item</i> este marcat cu acest <i>status</i> dacă valoarea sa este modificată (programatic, de către un <i>trigger</i> , sau de către utilizator); de asemenea, în cazul în care un <i>item</i> este modificat, toate <i>item-urile</i> aparținând aceleiași înregistrări vor fi marcate ca <i>changed</i> .
<i>VALID</i>	<i>Forms-ul</i> marchează un <i>item</i> ca valid dacă: <ul style="list-style-type: none"> ▪ toate <i>item-urile</i> din înregistrarea încărcată din baza de date sunt marcate ca fiind valide; ▪ procesul de validare al <i>item-ului</i> curent s-a terminat cu succes; ▪ după o instrucțiune <i>post</i> sau <i>commit</i> terminată cu succes; ▪ fiecare <i>item</i> dintr-o înregistrare duplicat moștenește <i>status-ul</i> de la sursă

Validarea are loc, după cum precizam anterior, și la nivel de înregistrare. Orice înregistrare primește un statut, după regulile:

<i>Status</i>	<i>Explicație</i>
<i>NEW</i>	La crearea unei înregistrări noi, acesteia îi este acordat statutul <i>new</i> .
<i>CHANGED</i>	Ori de câte ori un element dintr-o înregistrare este marcat ca <i>changed</i> , acest statut îi este acordat întregii înregistrări.
<i>VALID</i>	<p><i>Forms</i>-ul marchează o înregistrare ca <i>valid</i> dacă:</p> <ul style="list-style-type: none"> ▪ toate <i>item</i>-urile din înregistrare au fost validate cu succes; ▪ după o instrucțiune <i>post</i> sau <i>commit</i> terminată cu succes; ▪ o înregistrare duplicat moștenește <i>status</i>-ul de la sursa sa.

Statutul unui element poate fi modificat programatic de către *trigger*-i care conțin *built-in*-uri ce influențează procesul de validare:

- *clear_block*, *clear_form*, *exit_form* – primul parametru al acestor proceduri indică modul în care vor fi operate modificările făcute atunci când un bloc sau un modul sunt reinitializate, respectiv când formularul se închide (în cazul procedurii *exit_form*). Atunci când este folosită valoarea *NO_VALIDATE*, eventualele modificări nu vor fi operate. Absența unui parametru va produce afișarea unei casete de dialog care oferă posibilitatea alegerii unei opțiuni de salvare.
- *Item_is_valid* – se pot folosi procedurile *get_item_property* și *set_item_property* pentru a afla sau modifica proprietatea *item_is_valid* a unui element. Acest mod de setare a *status*-ului unui element nu poate fi aplicat și înregistrărilor. Totuși, setând *status*-ul fiecărui *item* al respectivei înregistrări, avem controlul asupra *status*-ului înregistrării însăși.
- *Enter* – este un *built-in* care forțează validarea imediată a unității curente de validare;
- *Validate (domeniu)* – produce validarea imediată la nivel de *domeniu*, care poate lua una din valorile: *default_scope*, *block_scope*, *record_scope*, *item_scope*.

2.11.3. Trigger-ii navigaționali

Forms Builder oferă o varietate de modalități pentru deplasarea *focus*-ului. Această acțiune, inițiată de utilizator sau generată programatic, poartă numele de navigare.

Pentru a defini statutul navigațional al unui modul a fost definită noțiunea de unitate navigațională: aceasta este un obiect intern, invizibil, folosit pentru a reține poziția curentă a *focus*-ului și poate fi setată la una din valorile (se respectă ierarhia următoare): *outside the form* (în afara formularului), *form*, *block*, *record*, *item*. Când se navighează în afara unui obiect, se schimbă unitatea navigațională; *Forms*-ul parcurge ierarhia precedentă până când elementul pe care se află cursorul este atins.

Spre deosebire de unitatea navigațională, cursorul este un obiect vizibil, extern, care indică poziția curentă a *focus*-ului. În timpul procesului navigațional, *Forms*-ul părăsește un obiect și intră într-un altul. Pătrunderea într-un obiect înseamnă modificarea unității navigaționale de la obiectul anterior către cel curent. *Forms*-ul nu va deplasa cursorul până când unitatea navigațională nu ia noua valoare. Dacă navigarea eşuează, *Forms*-ul modifică unitatea navigațională la vechea valoare. Dacă această tentativă eşuează, *Forms*-ul închide formularul. Cursorul rămâne pe poziția inițială până când unitatea navigațională devine stabilă.

Când utilizatorul inițiază rularea unui formular, acesta observă cursorul poziționat pe primul *item* din primul bloc navigațional (în ordinea din *Object Navigator*). Accesarea primului item al formularului a implicat, însă, desfășurarea unor procese navigaționale complexe; adițional navigației externe (care poate fi observată de utilizator) are loc și o navigare internă, care constă în:

- intrarea *focus*-ului în formular;
- intrarea *focus*-ului în bloc;
- intrarea pe prima înregistrare;
- intrarea în *item*.

Părăsirea *item*-ului curent și accesarea unui alt *item*, aparținând altor înregistrări, implică următoarele acțiuni:

- părăsirea *item*-ului curent;
- părăsirea înregistrării curente;
- intrarea *focus*-ului pe noua înregistrare;
- intrarea *focus*-ului pe *item*-ul dorit.

Modul în care se realizează navigarea se poate defini cu ajutorul proprietăților obiectelor. Au fost detaliate anterior intrări din paletele de proprietăți cum ar fi:

- la nivel de modul: *Mouse Navigation Limit* (determină cât de departe relativ la *item*-ul curent poate naviga utilizatorul cu *mouse*-ul), *First Navigation Block* (neprecizarea unei valori pentru această proprietate va determina navigarea, la inițierea formularului, în primul bloc, în ordinea în care sunt așezate în *Object Navigator*);
- la nivel de bloc de date: *Navigation Style*, *Previous Navigation Data Block*, *Next Navigation Data Block*;
- la nivel de *item*: *Enabled*, *Keyboard Navigable*, *Mouse Navigate*, *Previous Navigation Item*, *Next Navigation Item*.

Navigarea dintr-un formular este influențată, în afara proprietăților specifice ale obiectelor, și de *trigger*-ii naavigaționali, care se împart în două categorii:

- *Post-* și *Pre-* – când ne deplasăm din obiectul *x* în obiectul *y*, pentru primul se declanșează *trigger*-ul *post*-, iar pentru cel de al doilea *trigger*-ul *pre*-. Aceștia nu se declanșează dacă sunt definiți pentru un nivel ierarhic inferior unității curente de validare. La eșuarea oricărui din acești *trigger*-i *focus*-ul se întoarce pe obiectul părăsit anterior.
- *When-new-<object>-instance* – se declanșează imediat ce *focus*-ul este pe obiectul *<object>* și unitatea de navigare este stabilă.

Adesea poate avea loc, în timpul procesului naavigațional, un fenomen interesant: să presupunem că, la navigarea din obiectul *x* spre obiectul *y*, *trigger*-ul *pre*- definit pentru al doilea obiect eșuează. Atunci unitatea naavigațională tinde să se întoarcă la obiectul *x*. Dacă acesta are un *trigger* *pre*- definit și care eșuează, cursorul logic nu are unde să se întoarcă și este generată o eroare. Eșuarea în lanț a două *trigger*-e se numește capcana naavigării.

Procesul de navigare poate fi inițiat și programatic, folosind *built-ins*-uri restricționate (aceleia care afectează navigarea), cum ar fi: *go_form*, *go_block*, *next_block*, *up* (navighează către instanța *item*-ului curent aflată în înregistrarea anterioară), *down*, *next_set* etc. Acestea nu pot fi, însă, utilizate în *trigger*-ii *pre*- și *post*-.

2.11.4. Trigger-ii tranzacționali

Procesul tranzacțional este inițiat la apăsarea de către utilizator a butonului *Save* (sau, echivalent, selectarea opțiunii *Action → Save* din meniu) sau la apelarea, într-un *trigger*, a procedurii *commit_form*.

La încercarea de a salva modificările făcute de utilizator într-un formular, se declanșează o secvență de acțiuni care poate fi algoritmizată după următoarea schemă:

1. validarea formularului;

2. declanșarea *trigger-ului pre-commit*;
3. validarea blocurilor de date (în ordinea apariției acestora în *Object Navigator*);
4. executarea operațiilor DML:
 - a. în cazul unui *delete*:
 - i. declanșarea *trigger-ului pre-delete*;
 - ii. ștergerea rândului din tabela de bază sau declanșarea *trigger-ului On-Delete*;
 - iii. declanșarea *trigger-ului Post-delete*;
 - b. în cazul unui *insert*:
 - i. verifică proprietatea *Copy Value from Item*;
 - ii. declanșează *trigger-ul pre-insert*;
 - iii. verifică unicitatea înregistrării;
 - iv. inserează rândul în tabela de bază sau declanșează *trigger-ul on-insert*;
 - v. se declanșează *trigger-ul post-insert*
 - c. în cazul unui *update*:
 - i. se declanșează *trigger-ul pre-update*;
 - ii. se verifică unicitatea înregistrării;
 - iii. se modifică rândul corespunzător din tabela de bază sau se declanșează *trigger-ul on-update*;
 - iv. se declanșează *trigger-ul post-update*;
5. se declanșează *trigger-ul post-form-commit*; dacă operația următoare este *commit*, atunci:
6. se execută o comandă *SQL commit*;
7. se declanșează *trigger-ul post-database-commit*.

Enumerăm în tabela următoare câteva din caracteristicile celor mai uzitați *trigger-i* tranzacționali:

Trigger-i	Caracteristici	
<i>Pre-commit</i>	Se declanșează o singură dată, înainte ca blocurile de date să fie procesate.	Verificarea privilegiilor utilizatorului.

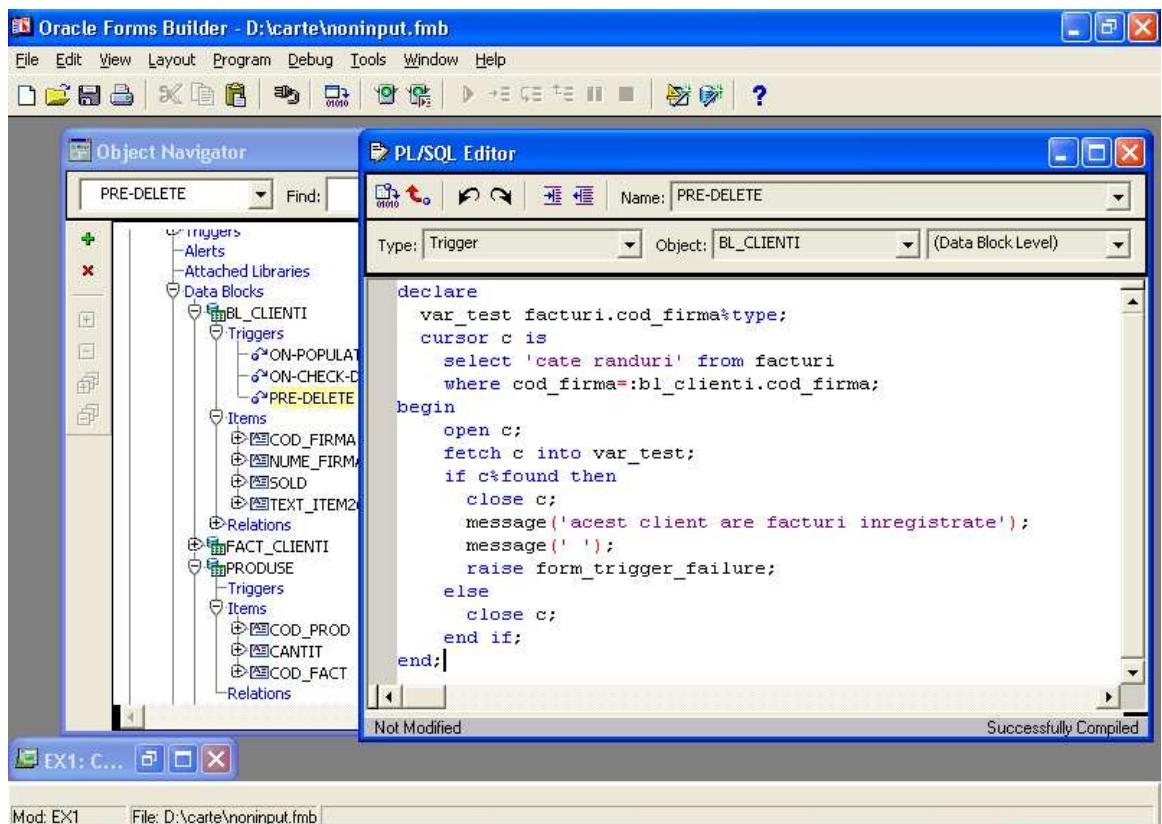
<i>Pre-delete</i>		Să presupunem că utilizatorul vrea să șteargă o înregistrare care are fii; <i>server-ul</i> va returna o eroare, dar putem preveni această situație căutând eventualii fii și ștergându-i prin acest <i>trigger</i> .
<i>Pre-insert</i>	Se declanșează o singură dată pentru fiecare înregistrare marcată pentru o operație <i>DML</i> ; <i>trigger-ii pre-DML</i> sunt folosiți și pentru jurnalizare (memorarea, într-o altă tabelă, a operațiilor <i>DML</i> efectuate asupra tabeliei de bază).	Dacă vrem să populăm un <i>item</i> dintr-o secvență, putem seta proprietatea <i>Initial Value</i> cu numele secvenței, ca în care de fiecare dată când rulăm formularul valoarea curentă a secvenței se incrementează; putem preveni această situație folosind acest <i>trigger</i> . Doar la declanșarea lui va fi incrementată secvența.
<i>Pre-update</i>		Permite implementarea regulii cheii externe. Putem simula un <i>update</i> în cascadă, în <i>trigger</i> actualizând fiii, apoi, în formular, tatăl.
<i>Post-DML</i> (<i>update</i> , <i>delete</i> , <i>insert</i>)		
<i>On-DML</i>	Se declanșează pentru fiecare înregistrare marcată pentru <i>insert</i> , <i>update</i> sau <i>delete</i> , înlocuind instrucțiunile <i>DML</i> standard cu cele precizate în codul <i>trigger-ului</i> . De obicei, includ un apel la <i>built-ins</i> urile <i>insert_record</i> , <i>update_record</i> , <i>delete_record</i> .	
<i>Post-forms-commit</i>	Se declanșează o singură dată, după ce blocurile de date sunt procesate, dar înainte de execuția instrucțiunii <i>SQL commit</i> .	
<i>Post-database-commit</i>	Se declanșează o singură dată, după executarea unui <i>SQL-commit</i> .	

Desigur, utilitatea acestor *trigger-i* se definește programatic, în funcție de cerințele utilizatorului, care pot fi cu totul altele decât cele prezentate în tabelul anterior.

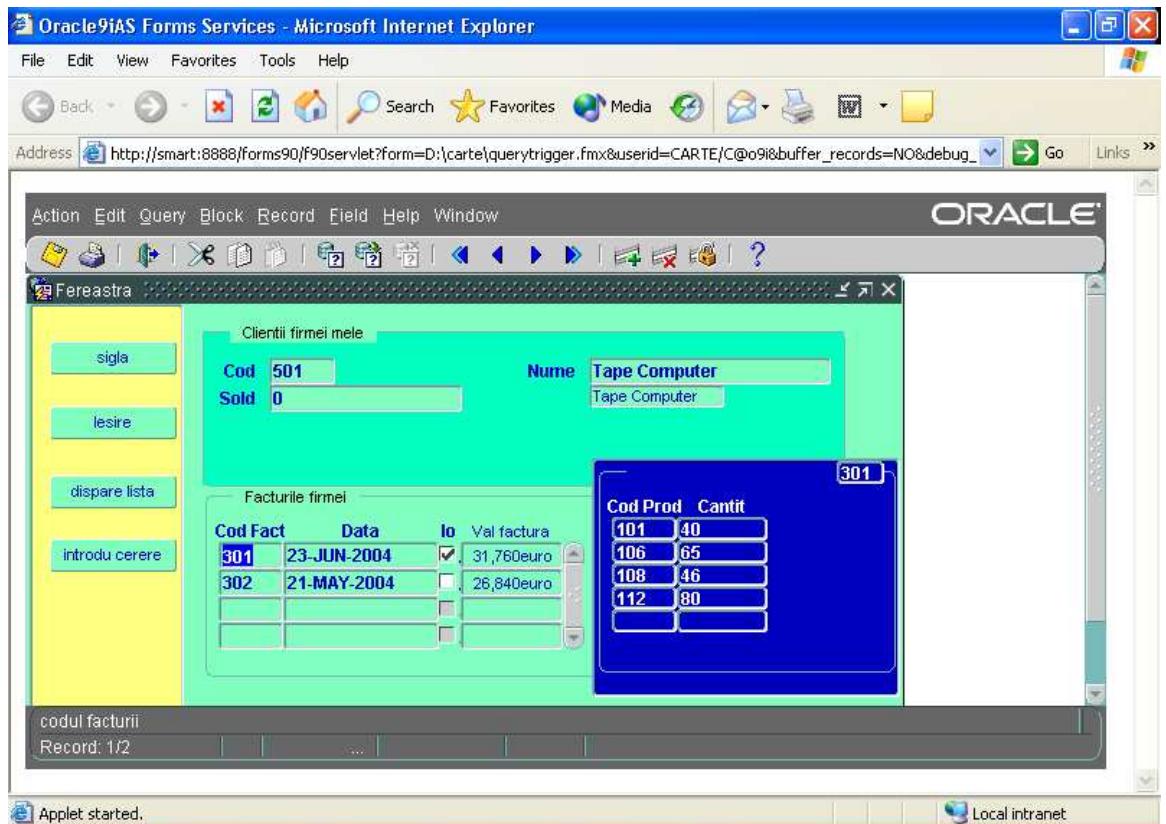
Să premiem acum cititorii care au avut răbdarea necesară atingerii acestei ultime pagini a acestei părți a lucrării de față cu cîteva *task-uri*:

1 ..Ce efect credeți că are definirea, la nivel de bloc de date, a *trigger-ului on-delete* conținând doar instrucțiunea *null*;?

2 ..Ce efect va avea definirea, la nivel de bloc *bl_clienti*, a *trigger-ului*:



3 ..Creați un *canvas* de tip *toolbar* vertical, unde așezați butoanele necesare în formular. Creați un buton la a cărui apăsare formularul va intra în modul *enter-query*. Utilizatorul va introduce criteriul de căutare, butonul își va schimba eticheta, iar acțiunea din nou a sa va avea efectul *execute_query*. Dublu *click* pe codul unei facturi va determina apariția unui *canvas* de tip *stacked* care va conține informații referitoare la datele respectivei facturi:



3. Oracle9i Reports Developer

3.1. Introducere

Raportarea este procesul de oferire a informației către clienți într-o formă accesibilă, bine structurată. *Oracle9i Reports Developer* reprezintă soluția pentru publicarea atât pe *web*, cât și pe hârtie a oricărui tip de date, din orice sursă, în orice format. Cu *Oracle9i Reports* nu efectuăm modificări în tabele, ci doar extragem datele în scopul prelucrării lor și a afișării într-o formă atrăgătoare.

Într-un mediu bazat pe *Internet*, clienții așteaptă un răspuns prompt la cererile lor. Una dintre modalitățile de reducere a timpului de așteptare este dezvoltarea rapidă a rapoartelor, fără necesitatea scrierii a sute de linii de cod sau a formatării manuale a structurilor. În *Oracle9i Reports* întregul mediu de dezvoltare este bazat pe instrumente de tip *wizard*, urmate de generarea automată de cod și posibilitatea transmiterii către *browser* a rapoartelor în formate diverse: *Hypertext Markup Language (HTML)* cu *Cascade Style Sheets (CSS)*, *Extensible Markup Language (XML)*, *Rich Text Format (RTF)*, *PostScript*, *Portable Document Format (PDF)*. Productivitatea muncii programatorului este îmbunătățită și prin posibilitatea construirii, în afara modulelor de tip raport, a încă două tipuri de module: *template-uri* (un schelet conținând reguli generale de afișare a datelor într-un raport) și biblioteci de programe *PL/SQL* (conțin unități de program ce pot fi apoi apelate din raport).

Tot mai mulți utilizatori caută informația în primul rând pe *Internet*. Totuși, printarea informației are importanță să de necontestat. Aceste două modalități de publicare sunt complementare una alteia, neexcluzându-se reciproc. În majoritatea cazurilor, nu putem afirma despre un fișier *HTML* care arată excelent într-o pagină *web* că va avea același aspect și la printare, după cum macheta unui raport creat pentru a fi printat nu este satisfăcătoare pentru afișarea într-un *browser*. Instrumentul *Reports Builder*, care permite proiectarea rapoartelor, construite, în același timp, două machete: una specifică *web*-ului, cealaltă în scopul printării. Macheta *web* este optimizată pentru *HTML*, pe când macheta *paper* este proiectată pentru formatele *PDF* și *PostScript*. Avantajul major constă în faptul că modelul de date este unic, productivitatea fiind astfel îmbunătățită.

Ca și în cazul formularelor, raportul poate fi construit (cu ajutorul instrumentului *Reports Builder*) și testat pe mașina *client* (amintiți-vă, trebuie startată instanța *OC4J*), apoi transferat pe *server-ul* de aplicații, care, fizic, se poate afla pe alt calculator. Orice *browser* accesează raportul, aflat pe *Oracle Application Server*, printr-o adresă *web*. Raportul va accesa datele, aflate pe *server-ul* de baze de date. Algoritmul folosit pentru

crearea, testarea și rularea aplicațiilor *Oracle9i Reports Developer* este similar cu cel utilizat de *Oracle9i Forms Developer*.

Oracle9i Reports Developer este, deci, o colecție de programe folosite pentru a ușura munca de raportare. Componența *Report Builder* permite definirea modelului de date pentru raport, crearea și testarea machetelor *web* și *paper*, posibilitatea atașării de conținut dinamic unui raport *HTML* prin atașarea de marcaje *Java Server Page* (JSP) etc. Componența *Oracle9iAS Reports Services* oferă mediul pentru rularea, distribuirea și publicarea aplicațiilor *Oracle9i Reports Developer*.

Executabilele incluse în *Reports Developer* au fost denumite în *Windows* conform regulii *rw<?>.exe* și sunt prezentate pe scurt în tabelul următor:

	Numele fisierului executabil	Numele aplicației	Descriere
<i>Developer</i>	<i>rwbuilder</i>	<i>Reports Builder</i>	Creează și rafinează raportul.
	<i>rwrn</i>	<i>Reports runtime</i>	Mediu <i>runtime</i> pentru testare.
	<i>rwconverter</i>	<i>Reports Converter</i>	Convertește raportul în diferite formate de stocare.
<i>Reports Services</i>	<i>rwserver</i>	<i>Reports Server</i>	Instalează/apeleză componente de pe <i>Internet Application Server (IAS)</i> care gestionează rapoartele ce se vor executa.
	<i>rwclient</i>	<i>Reports Client</i>	Accesează <i>server-ul</i> de rapoarte și trimită acolo linia de comandă pentru executarea raportului curent.
	<i>rwrqm</i>	<i>Reports Queue Manager</i>	Vizualizează și programează la ore stabilite execuția rapoartelor.
	<i>rwservlet</i>	<i>Reports Servlet</i>	Rulează raportul pentru <i>web</i> , asigurând procesarea <i>server-side</i> ce constă, de obicei, în accesarea bazei de date.
	<i>rwcgi</i>	<i>Reports CGI</i>	Este menținut doar pentru compatibilitatea cu versiunile anterioare; oferă o conexiune între <i>server-ul web</i> și <i>Reports Services</i> .

Crearea unui raport include definirea a două elemente (distingute în faza de proiectare) ce vor determina, la rulare, conținutul și aspectul raportului. Cele două părți logice ale unui raport sunt:

- Modelul de date, constând în datele ce vor fi afișate la ieșire și structura lor;

- Macheta (*layout*), ce definește modul de afișare a datelor la ieșire. Macheta poate fi de tip *paper* (utilizată pentru rapoartele destinate printării) sau de tip *web* (utilizată pentru rapoartele ce se vor publica pe *web*). Macheta de tip *web* are în spate cod sursă (*HTML* și *JSP*).

Un raport poate conține:

- Un model de date și o macheta *paper*;
- Un model de date și o macheta *web*;
- Un model de date, o macheta *web* și una *paper*.

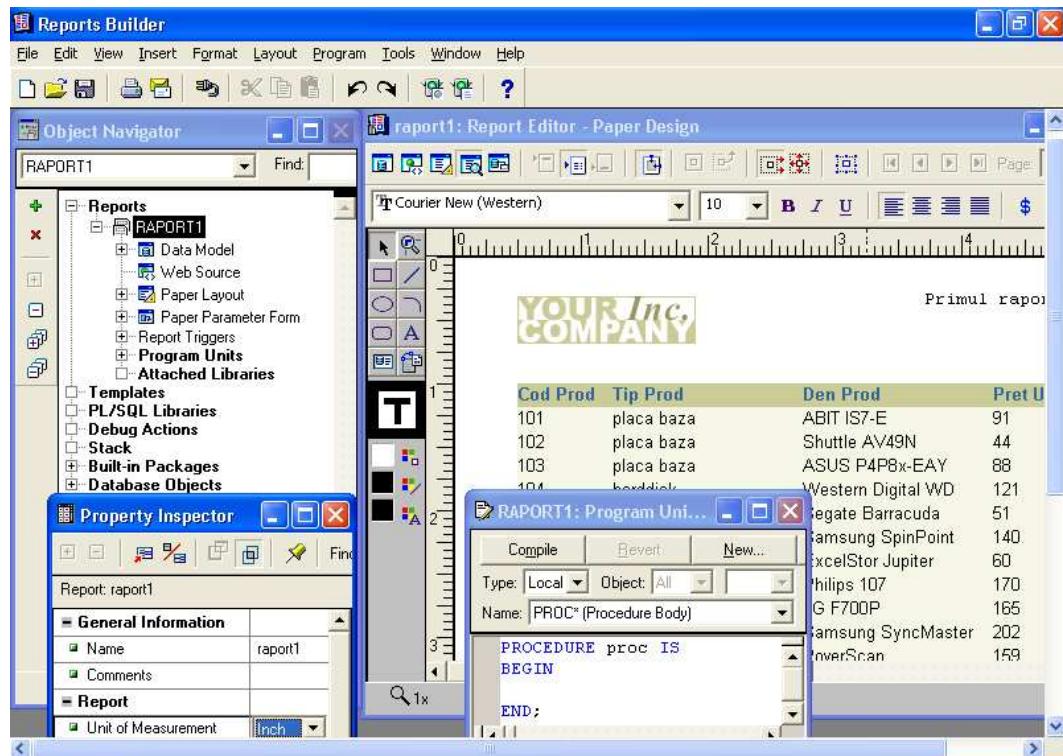
3.2. Proiectarea rapoartelor

3.2.1. Componentele *Reports Builder*

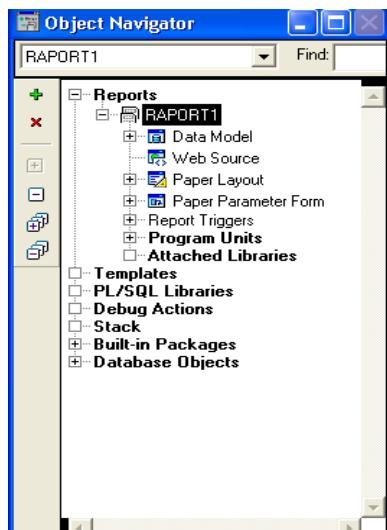
Aplicația *Report Builder* include patru componente:

1. *Object Navigator*;
2. *Report Editor*;
3. *Property Inspector*;
4. Editorul *PL/SQL*.

Încercați să le identificați în captura de ecran următoare:



Object Navigator

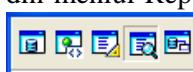


Acesta constă, ca și în cazul produsului *Forms Builder*, într-o prezentare ierarhică a tuturor obiectelor aplicației, permitând accesarea și manipularea lor rapidă. Toate nodurile din *Object Navigator*, mai puțin *Database Objects*, conțin obiecte aflate pe parte de *client*. La crearea unui raport nou (prin *click* pe butonul marcat cu o cruce verde și aflat în partea superioară a barei de instrumente), acestuia îi va fi automat asignat numele *ModuleN*, *N* fiind numărul său. Numele raportului nu poate fi modificat la fel ca în cazul formularelor. Salvarea raportului va produce modificarea automată a numelui său în *Object Navigator* acordându-i-se aceeași valoare cu numele fișierului. Obiectele conținute în raport se împart în mai multe categorii (reflectate și în ierarhia din *Object Navigator*):

- *Raport* – se află proprietăți generale, legate de setările pentru previzualizare, dimensiuni ale paginilor etc.;
- *Data Model* – obiectele aflate la acest nivel definesc datele publicate în raport și structura lor. Există 11 parametri-sistem care așteaptă să fie setați, se pot defini parametri-utilizator, sunt conținute frazele *select*, împreună cu eventualele grupuri returnate de acestea. Ierarhia de la nodul *Groups* indică tipul raportului: *master-detail* sau simplu. Putem avea mai multe fraze *select*, dar acestea trebuie conectate logic; în acest scop există nodul *Data Links*.
- *Web Layout* – bazându-se pe frazele *select*, *Reports Builder* construiește două machete: *Paper Layout* și *Web Layout*, în spatele căreia este cod *HTML* “împănat” cu marcaje *JSP*;
- *Paper Layout* – obiectele aflate sub acest nod definesc formatul raportului: poziționarea și frecvența de apariție a datelor, text și elemente grafice;
- *Paper Parameter Form* – definește apariția în *runtime* a unui formular pentru parametri; creăm, de exemplu, un raport care să afișeze toate comenziile unui client. La rularea raportului va apărea o formă de parametri, în care utilizatorul va introduce codul clientului, iar raportul va afișa doar comenziile acestuia. Astfel, este creat un singur raport și rulat pentru mai multe seturi de date;
- *Report Triggers* – conține proceduri *PL/SQL* care se vor declanșa în timpul populării cu date și al formatării lor. *Reports Builder* conține cinci *trigger-i*, care sunt:
 - *Before Report trigger*;
 - *After Report trigger*;
 - *Between Pages trigger*;
 - *Before Parameter Form trigger*;
 - *After Parameter Form trigger*.
- *Program Units*
- *Attached Libraries* .

Report Editor

Acesta este un utilitar care conține cinci moduri de vizualizare utilizate pentru gestionarea modelului de date și a machetelor raportului. Fiecare din acestea poate fi accesat printr-un buton aflat în colțul din stânga sus al ferestrei editorului de rapoarte sau din meniul Reports Builder, selectând *View → Change View*, apoi modul de vizualizare dorit. Să le prezintăm pe scurt, urmând să revenim asupra lor, în ordinea apariției în fereastra editorului de rapoarte:



- *Data Model* – (nu confundați modul de vizualizare *Data Model* cu nodul *Data Model* din *Object Navigator*!) constă în reprezentarea simbolică a structurilor de date afișate în raport. Sunt marcate special tipul obiectelor și relațiile dintre acestea. Obiectele aflate aici nu vor apărea ca atare în raport, la rularea acestuia, dar vor influența macheta.
- *Web Source* – acest mod permite vizualizarea codului sursă care produce publicarea pe *web* a raportului. După cum precizam, la definirea cu ajutorul instrumentului *Report Wizard* a raportului se poate alege crearea ambelor machete (*web* și *paper*), însă modificarea ulterioară a oricareia dintre acestea nu se va reflecta și în cealaltă machetă, care va trebui modificată manual.
- *Paper Layout* – reprezintă o zonă de lucru în care putem defini modul de afișare a datelor. Se lucrează cu obiecte specifice machetelor, cum ar fi: cadre, cadre repetitive, câmpuri, ancore și obiecte grafice. La rulare, *Reports Builder* folosește această machetă ca pe un schelet de bază, pe care îl expandează astfel încât datele să poată fi poziționate în el.
- *Paper Design* – este un mod de vizualizare care oferă un *preview* al raportului și permite manipularea conținutului efectiv al acestuia (acest mod de vizualizare este ideal pentru a modifica, de exemplu, lățimea unei coloane).
- *Paper Parameter Form* – permite crearea, cu ajutorul instrumentului *Paper Parameter Builder*, a unui formular unde utilizatorul va introduce propriii parametri sau va fi invitat să aleagă valorile dintr-un *LOV*.

The figure consists of four screenshots of the Oracle Reports Builder interface:

- Data Model:** Shows a diagram of data objects and their relationships. A central object is labeled "G_COD_PROD" with attributes "788 COD_PROD", "A TIP_PROD", and "A DEN_PROD". A relationship line connects it to another object labeled "CANTITPerRe".
- Web Source:** Shows the XML code generated for a web-based report. The code includes tags for report definition, objects, and an HTML section with a title, meta content, and a table with a logo.
- Paper Layout:** Shows the visual layout of the report. It features a table with columns for "Cod Prod", "Tip Prod", "Den Prod", "Pret Unit", "Cantit", and "SumCANTITPerRe". The "Den Prod" column contains an image of a computer monitor.
- Paper Design:** Shows a preview of the report with the heading "YOUR Inc. COMPANY". The table data is identical to the layout view, showing various computer components and their details.

Reamintim că *Data Model* este responsabil cu partea logică a raportului, livrând informația, în timp ce *paper layout* și *web layout* se ocupă de modul de afișare a acesteia. Legătura dintre modelul de date și machetă se face la rularea raportului în modul următor: pentru fiecare grupare din *Data Model*, *Report Builder* construiește în machetă câte un cadru repetitiv (*repeating frame*) și reciproc, orice cadru repetitiv are la bază o grupare. În cadrul unui *repeating frame* avem câmpuri (*fields*). Ele sunt populate conform instrucțiunii *select* definite la acest pas.

Property Inspector

Toate obiectele din *Object Navigator* au câte o paletă de proprietăți specifice, care descriu comportamentul obiectului. Paleta de proprietăți a oricărui obiect se obține apăsând tasta *F4*, atunci când obiectul este selectat sau alegând opțiunea *Property Inspector* din meniul de context al obiectului.

Editorul PL/SQL

Acest instrument este identic cu cel folosit în *Forms Builder* și permite scrierea și compilarea de cod *PL/SQL* asignat unei proceduri definite de utilizator sau unui *trigger*.

3.2.2. Instrumente de tip *wizard*

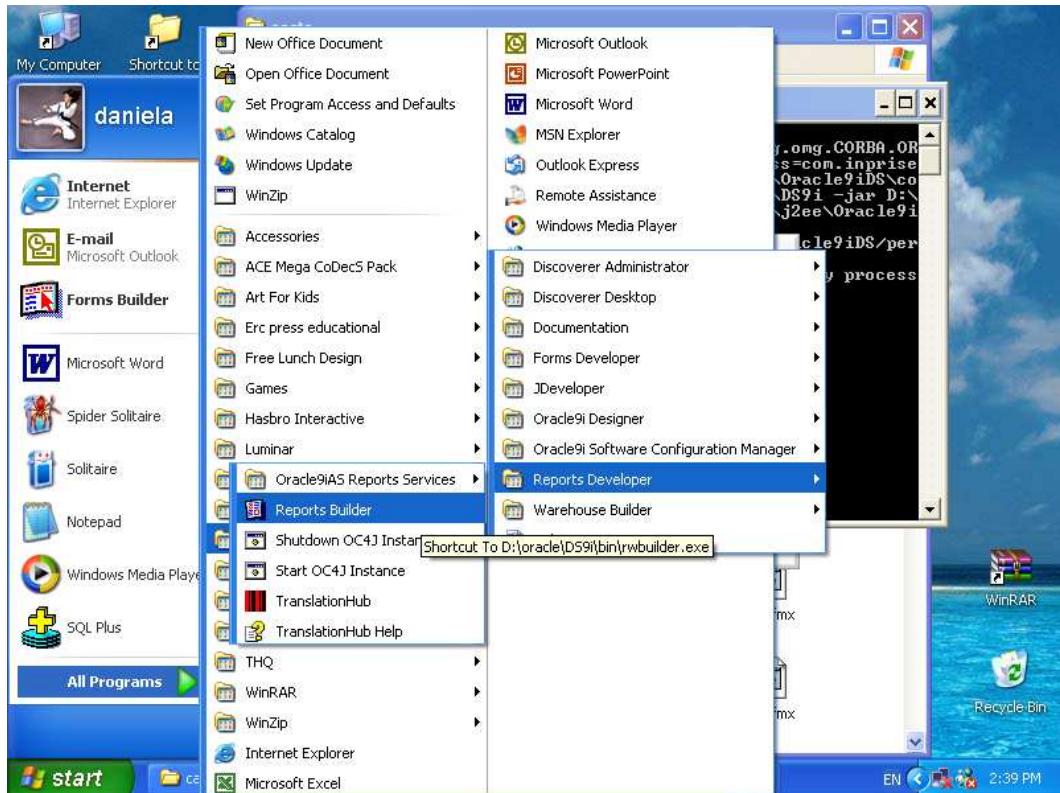
Indiferent de modul de stocare a raportului, definiția acestuia trebuie să conțină un model de date, o sursă *web* și/sau o machetă pentru printare, unități de program și formulare pentru parametri. Modelul de date și unitățile de program sunt comune ambelor machete (*web* și *paper*).

În *Oracle9i Reports* avem mai multe posibilități pentru a crea un raport:

- folosind *Reports Builder*;
- definind modelul de date și/sau macheta într-un fișier *XML*.

Ne vom opri asupra primei posibilități și ne vom ușura munca folosind instrumentele de tip *wizard* incluse în *Report Builder*, apoi vom rafina manual raportul obținut. Reamintim faptul că orice modificare manuală într-una din machetele *paper* sau *web* nu se va reflecta și în celalătă.

Să purcedem, deci, la construirea unui raport. Din aceleași considerente cu cele prezentate în capitolul *Oracle9i Forms Developer* trebuie startată instanța *OC4J* (care creează un mediu virtual ce permite rularea rapoartelor, în vederea testării), apoi aplicația *Report Builder*, al cărei *shortcut* poate fi găsit la meniul *Start* al *Windows*-ului:

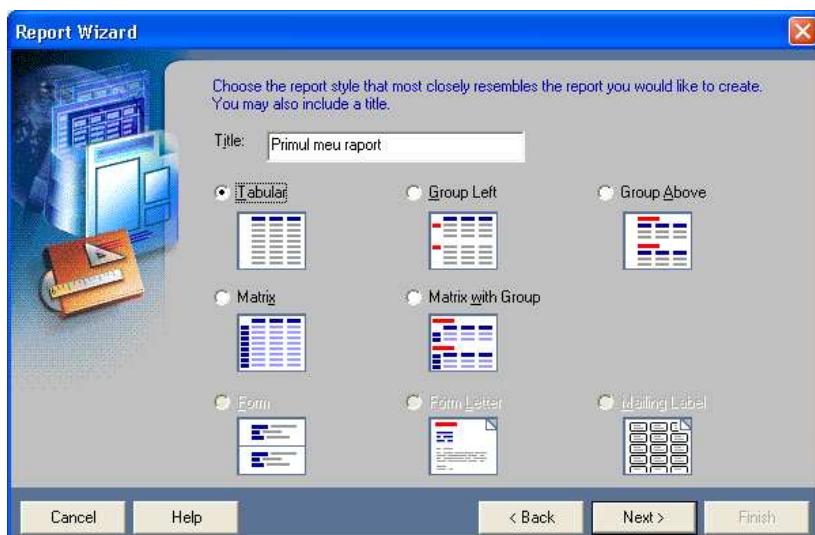


Invocarea *Reports Builder*-ului va produce afișarea unei casete de dialog cu opțiuni privind modul de creare a raportului:



Una dintre cele mai uzitate modalități este folosirea instrumentului *Report Wizard*, urmată de rafinarea manuală a raportului. Să lăsăm, deci, opțiunea *Use the Report Wizard* bifată și să continuăm. *Report Wizard* oferă o interfață de tip *step-by-step* pentru crearea unui raport. Prima casetă de dialog cuprinde un mesaj introductiv și poate fi inhibată prin deselectarea casetei de validare “*Display at Startup*”. Pagina următoare culege informație referitoare la modul de publicare a raportului, în funcție de care creează machetele *paper*, *web* sau pe amândouă.

Acționarea butonului *Next* determină afișarea casetei *Report Style*, în care specificăm titlul și tipul raportului ce va fi creat:



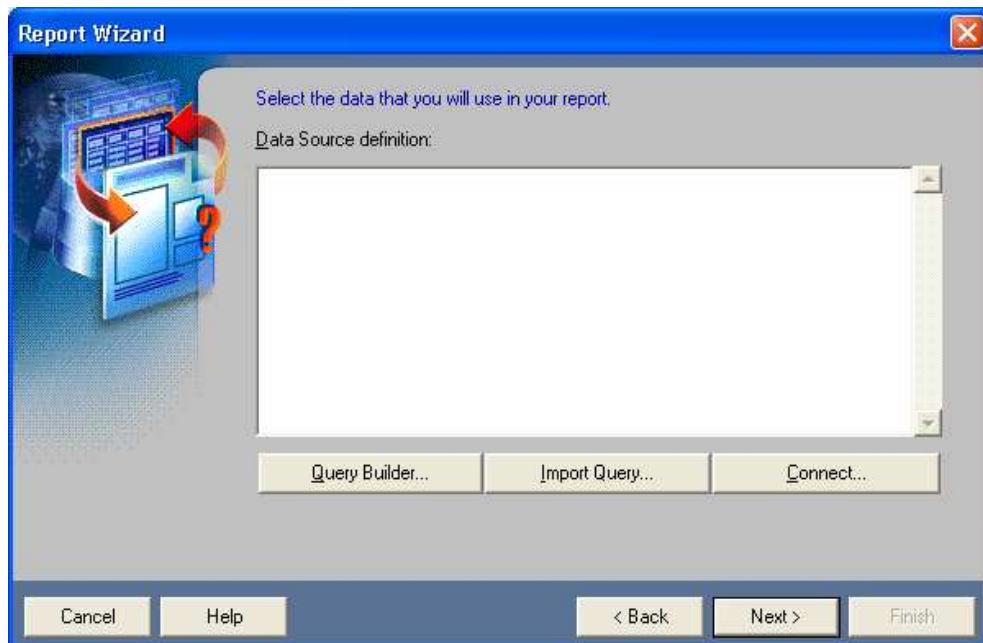
În campul *Title* completăm textul titlului raportului, aşa cum vrem să apară la afişare (partea de formatare a sa se defineşte ulterior), apoi selectăm un buton din grupul radio ce permite specificarea tipului raportului ce va fi creat. Observați faptul că, la acest moment, ultimele trei opțiuni (legate de etichete și scrisori) sunt inhibate, deoarece am ales și generarea machetei *web*, unde nu are sens publicarea a astfel de documente. În cazul când, în caseta de dialog anterioară (Layout Type) am fi ales *Paper Layout Only* am fi avut, la acest pas, și ultimele trei posibilități disponibile.

Pentru o înțelegere mai bună a acestor tipuri de rapoarte, să reamintim că se lucrează cu un model de date, bazat pe o frază *select* executată în *runtime* și care are ca efect încărcarea de pe *server* a setului de date necesar raportului. Aceste date sunt afișate conform unei machete. *Report Builder* organizează datele, creând grupuri. La scrierea unei instrucțiuni *select* pe o tabelă este creat automat un grup, care conține coloanele selectate în cerere. Se pot defini grupuri adiționale (pentru a crea niveluri de diviziune ale raportului) manual sau folosind *Report Wizard*, opțiunile *group-above* și *group-left*.

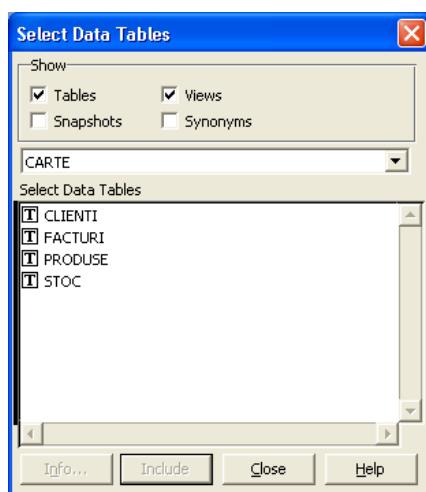
Revenim la caseta de dialog ce permite determinarea tipului raportului. Există, după cum observați, mai multe tipuri de rapoarte:

- *tabulare* – cel mai des întâlnite; fiecare coloană a raportului corespunde unei coloane dintr-o tabelă;
- *group-left* – împarte rândurile din tabelă în submulțimi, în fiecare submulțime fiind conținute rândurile pentru care valoarea uneia dintre coloane este aceeași; se folosește pentru evitarea repetării afișării informației;
- *group-above* – conține două sau mai multe grupuri de date; pentru fiecare valoare din grupul *master*, sunt afișate valorile corespunzătoare din grupurile *detail*;
- *matrix* – un raport de tip matricial conține un rând de etichete, o coloană de etichete, iar informația aflată la intersecția acestora este corelată cu semnificația etichetelor. O caracteristică importantă a acestui tip de rapoarte este acela că numărul de coloane nu este cunoscut până la încărcarea datelor din baza de date;
- *matrix with groups* – este un raport de tip *group-above* cu o matrice pentru fiecare valoare a grupului *master*.

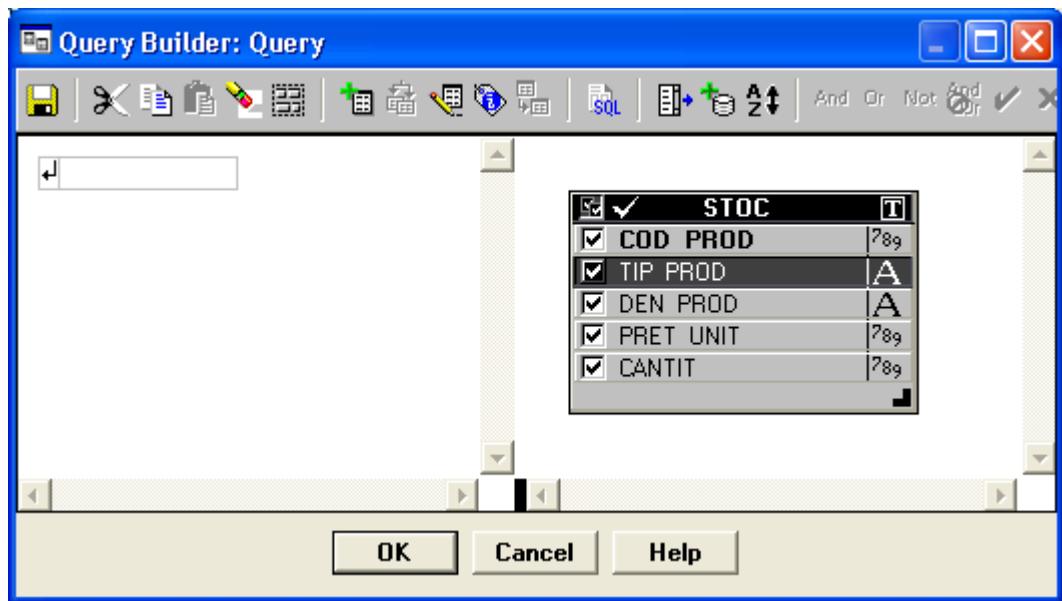
Să construim un raport tabular lăsând, deci, bifat primul buton al grupului radio, *tabelar*. Următoarea casetă de dialog, *Data Source Type*, permite specificarea sursei de date a raportului. Vom construi un raport pe baza unei instrucțiuni *select*, deci selectăm opțiunea *SQL Query* și apăsăm butonul *Next*, care permite definirea sursei de date:



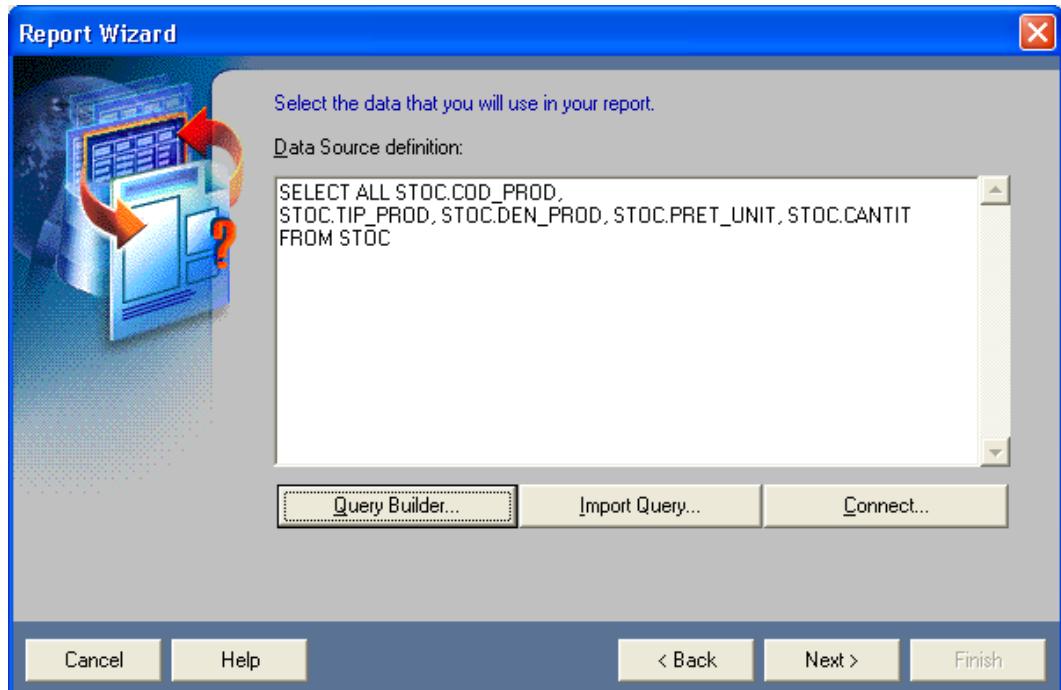
Pentru a accesa datele din baza de date trebuie să ne conectăm la aceasta (observați butonul *Connect*), apoi putem tasta fraza *select* ce va popula raportul. Pentru utilizatorii nu prea familiarizați cu limbajul *SQL* avem la dispoziție un utilitar pentru automatizarea creării cererilor, numit *Query Builder*. Invocarea sa (prin acționarea butonului corespunzător) va determina apariția unei casete ce conține toate tabelele din schema utilizatorului curent, în vederea selectării acelora care vor participa la fraza *select*:



Să creăm un raport simplu, care va afișa toate produsele aflate în stoc. Selectăm tabela *stoc*, apoi acționăm butonul *Include*. Procedăm la fel cu toate tabelele ce vor oferi date în vederea publicării, apoi dăm *click* pe butonul *Close*. Următoarea fereastră conține toate tabelele selectate la pasul anterior și permite specificarea coloanelor care vor intra în cerere și a relațiilor dintre tabele (în cazul rapoartelor *master-detail*). Putem da dublu *click* pe numele tabelei, rezultatul fiind selectarea tuturor coloanelor:

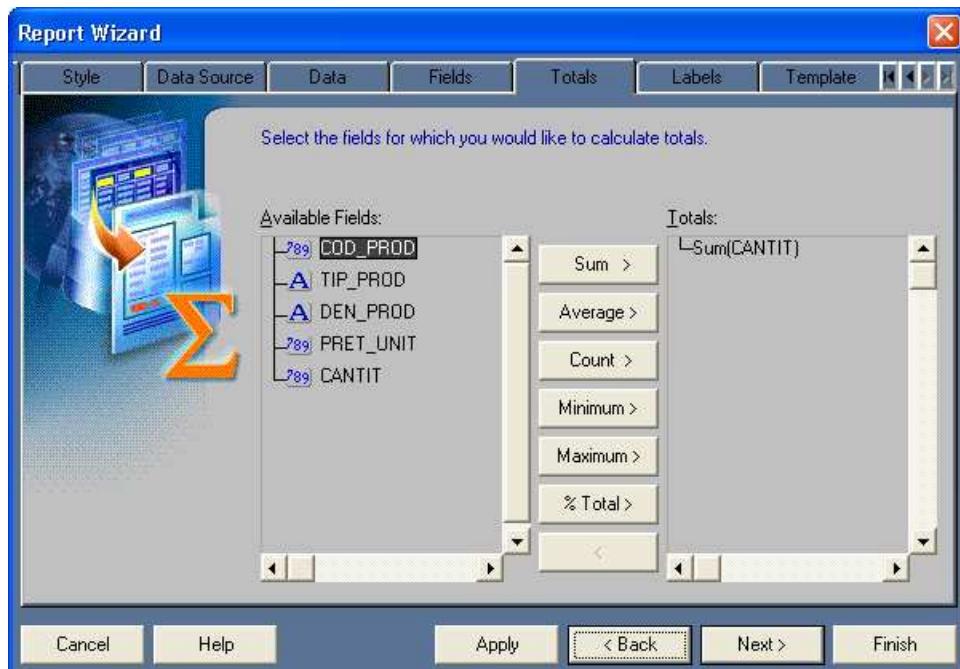


Dăm *click* pe *OK* și automat *Query Builder* va copia sintaxa cererii în *Report Wizard*:



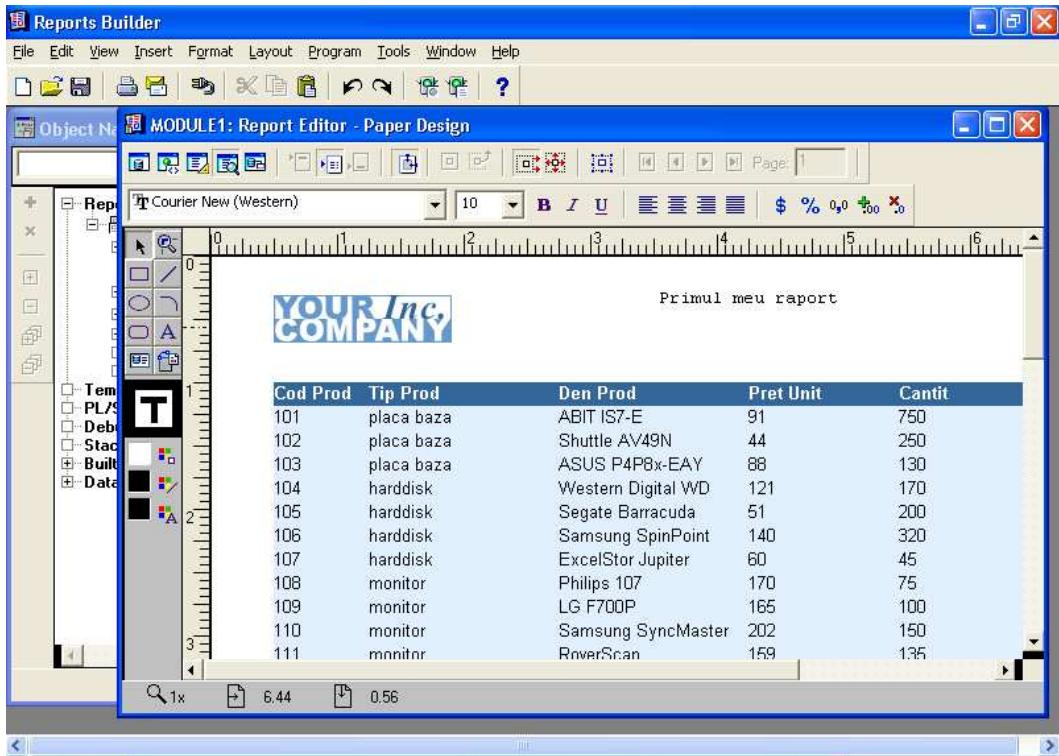
Casetă următoare de dialog, numită *Fields*, permite selectarea câmpurilor care vor participa la *output*. Câmpurile selectate în zona *Available Fields* pot fi mutate în zona câmpurilor ce vor fi afișate (*Displayed Fields*) prin acționarea butonului *>*. Pentru selectarea rapidă a tuturor coloanelor se apasă butonul *>>*. Ordinea apariției câmpurilor în zona *Displayed Fields* va determina ordinea apariției coloanelor raportului în *runtime*. Într-un raport tabelar, acestea vor fi așezate în ordinea specificată, de la stânga la dreapta. Folosind tehnica *drag* putem modifica ordinea de apariție a coloanelor în raport. Câmpurile rămase în zona *Available Fields* pot fi referite în raport (ele se numesc ascunse) și folosite în calcule statistice sau în codul *trigger*-ilor.

Acționând butonul *Next* ajungem la următoarea casetă de dialog, *Totals*, unde putem crea totaluri bazate pe funcțiile statistice *SQL* standard afișate de *Report Wizard*:

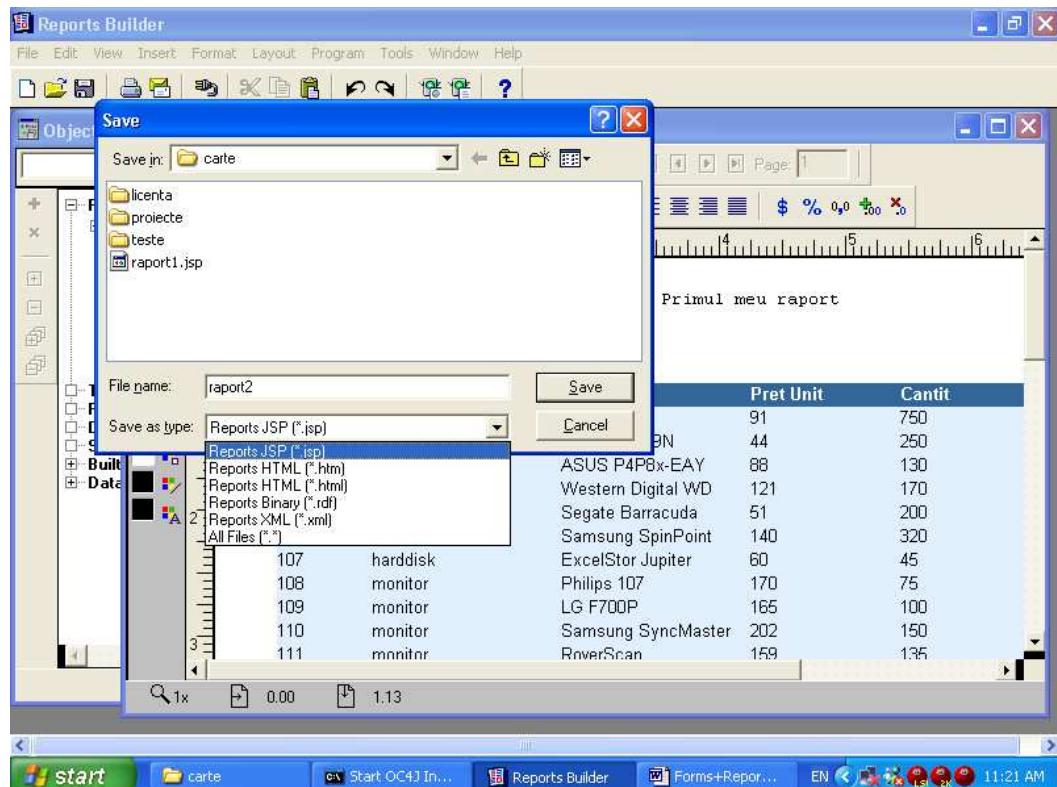


Toate funcțiile, mai puțin *count*, folosesc valori numerice. Acționând butonul corespunzător funcției dorite, aceasta va fi aplicată câmpului selectat în zona *Available Fields*. *Wizard*-ul va crea câte un total la fiecare nivel al raportului, adică: pentru întregul raport și pentru fiecare grup în parte, în cazul existenței acestora.

Următoarea casetă de dialog permite atașarea unor etichete câmpurilor, precum și setarea dimensiunii acestora, iar ultimul pas al *wizard*-ului oferă posibilitatea alegerii unei machete pentru publicarea raportului, dintr-o serie de *template*-uri disponibile. Utilizatorul își poate defini propriile machete, pe care le va salva ca *template*-uri în vederea aplicării lor ulterior asupra altor rapoarte. Acționarea butonului *Finish* va determina invocarea utilitarului *Report Editor*, în modul *Paper Design*, care permite o vizualizare a raportului:



Este momentul să salvăm rezultatul muncii noastre de până acum. Ceea ce vom salva de fapt nu este raportul, aşa cum apare el în *runtime*, ci definiția sa. La fiecare rulare va fi executată fraza *select* ce va popula raportul, iar informația returnată de aceasta va fi afișată conform specificațiilor machetei. Să selectam, deci, din meniul *Report Builder* opțiunea *File → Save*. Această acțiune va invoca o bine-cunoscută casetă de dialog specifică mediului *Windows*, ce permite alegerea locului unde va fi stocat raportul, a numelui său și a formatului în care este salvat:

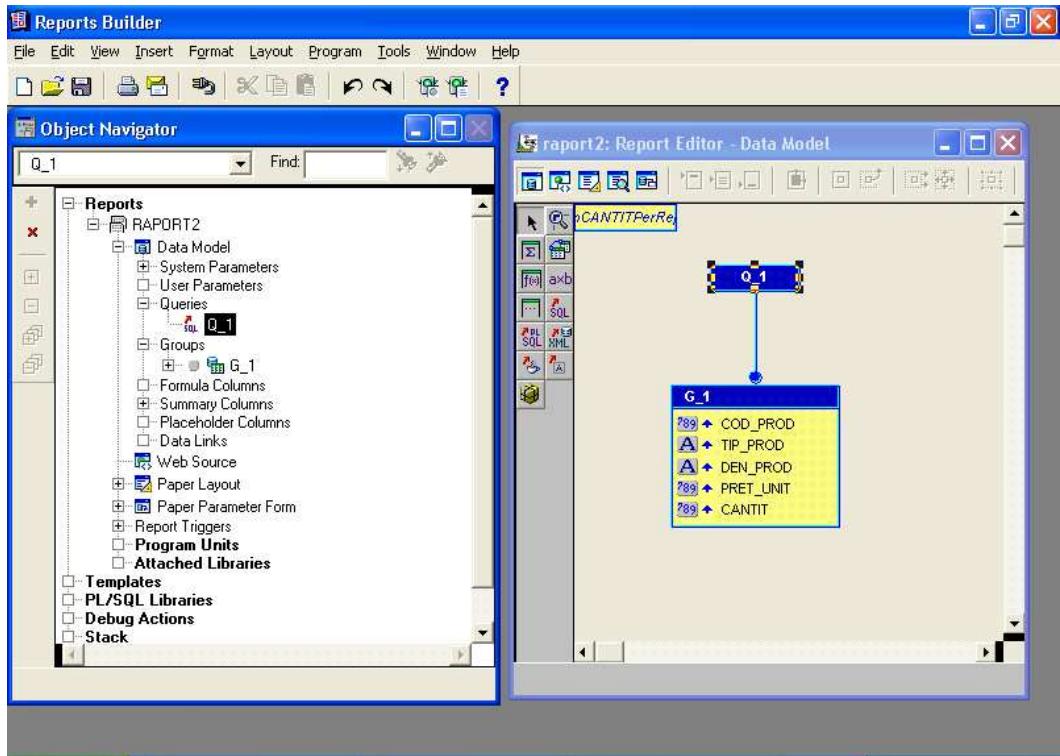


Formatul implicit este *.jsp*. Remarcăți în *LOV*-ul afișat anterior existența formatului *.rdf*, specific versiunii 6 a produsului *Oracle Reports* și introdus în această versiune doar pentru compatibilitatea cu cele anterioare.

3.3. *Report Editor – 5 moduri de vizualizare a raportului*

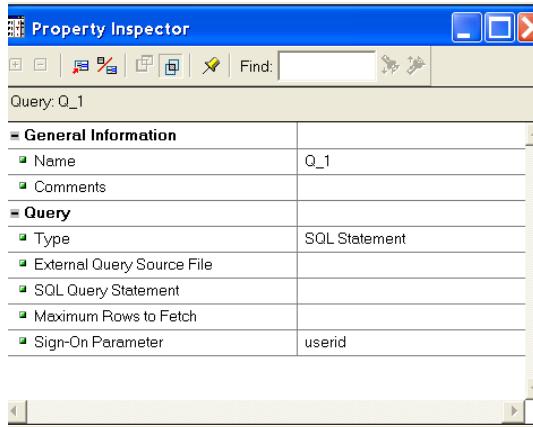
Macheta raportului fiind salvată, ne întoarcem la utilitarul *Report Editor*, unde vom analiza modurile de vizualizare a raportului și modificările ce se pot efectua în fiecare din acestea.

Primul, în ordinea apariției butoanelor care le apelează, este modul ce permite vizualizarea modelului de date:



A fost creată o cerere, numită implicit Q_1 , iar ea a generat un grup, numit G_1 . (Fiecare cerere generează un grup propriu!) Toate coloanele returnate de fraza *select* se află în grupul G_1 . Coloanele singulare (create de utilizator pentru calcule statistice) se află în afara grupului. Observați săgeata orientată în sus și aflată în stânga fiecărui element al grupului G_1 . Aceasta este adăugată implicit și determină o clauza *order by*. Pentru creșterea vitezei de rulare a raportului este recomandată anularea acestei clauze pentru coloanele unde ea nu este necesară (setând proprietatea *Break Order* a coloanei la valoarea dorită: *Null*, *Ascending* sau *Descending*).

Proprietățile obiectului Q_1 și ale grupului generat de acesta pot fi modificate prin intermediul inspectorului de proprietăți. Proprietățile specifice unei cereri pot fi observate în figura următoare:

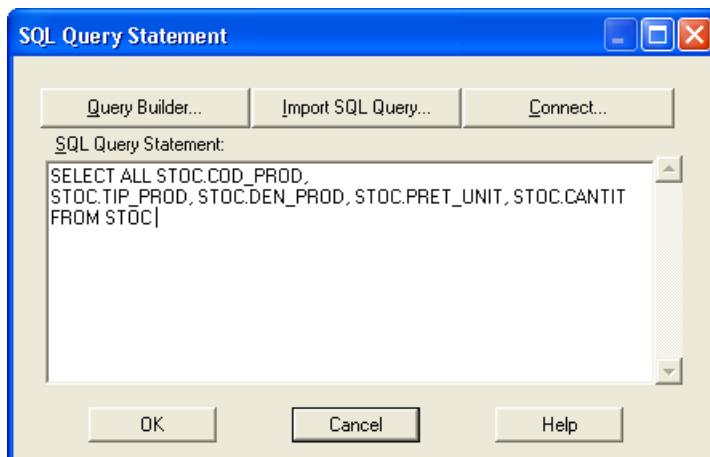


delimitatorii:

-- pentru comentariu la sfârșitul liniei curentă;

/* comentariu */ pentru comentarii multi-linie.

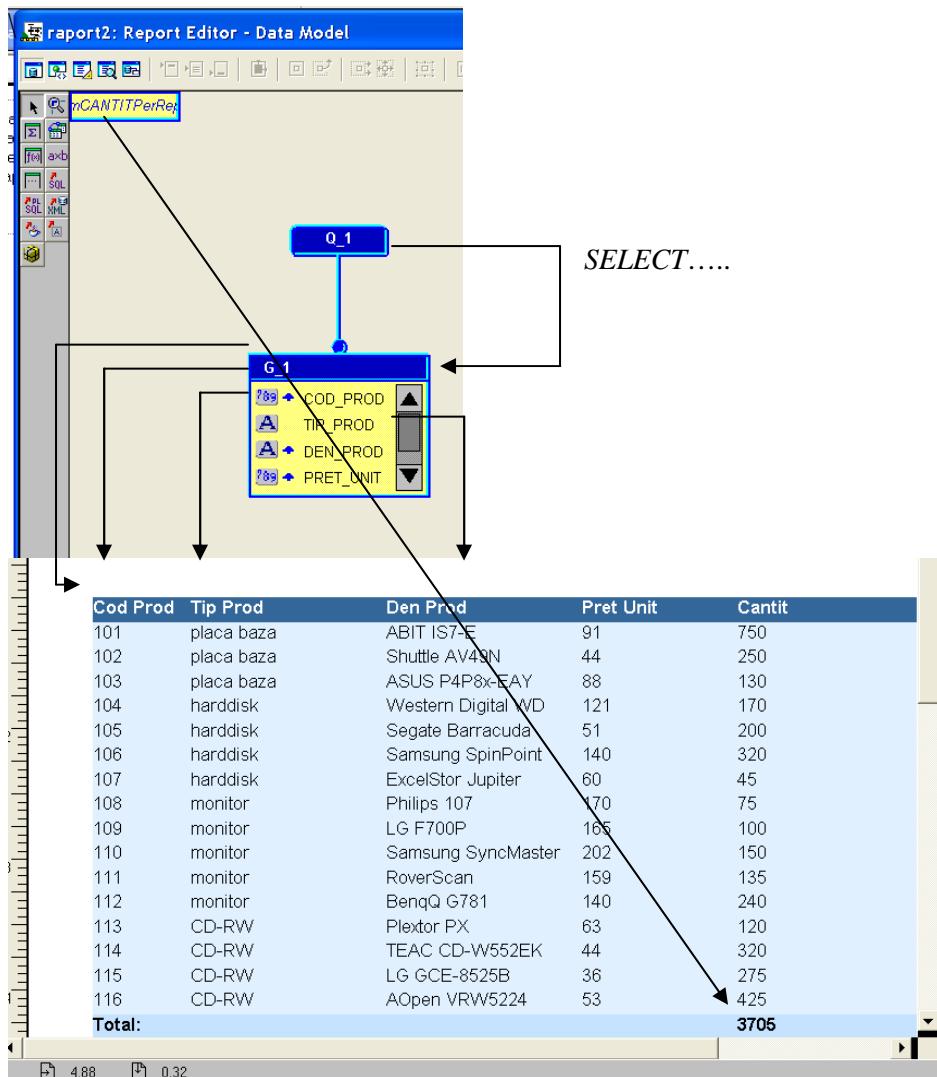
Coloanele returnate de fraza *select* nu pot fi șterse cu tasta *Del* în *Data Model*. Doar coloanele create de utilizator pot fi șterse în acest mod; pentru primul caz, trebuie modificată chiar fraza *select*: la efectuarea unui *click* pe proprietatea *SQL Query Statement* vom obține aceeași casetă de dialog cu cea întâlnită în *Report Wizard*, în care utilizatorul poate modifica manual sau cu ajutorul utilitarului *Query Builder*, conținutul cererii:



Aceeași casetă de dialog poate fi invocată și dând dublu *click* pe numele cererii, *Q_1*.

Acesta este locul de unde putem modifica numele cererii, tipul și numărul maxim de rânduri returnate de aceasta (opțiune extrem de utilă în cazul în care suntem în faza de testare a raportului, iar baza de date conține mii de înregistrări), precum și fraza *select*. Este recomandabil ca într-un model de date complex numele atribuite automat cererilor să fie modificate de către utilizator, pentru o mai bună înțelegere a acestora. Comentariile pot fi inserate la proprietatea *Comments* sau direct în fraza *SQL*, folosind

Urmăriți, în figura de mai jos, legătura dintre elementele modelului de date și câmpurile afișate la rularea raportului: obiectul Q_1 generează o frază *select* ce populează grupul de înregistrări G_1 . Fiecare înregistrare încărcată din baza de date va deveni o instanță a grupului. Fiecare element al unei înregistrări este încărcat în coloana corespunzătoare a grupului de date. Fiecare coloană întoarce valori care sunt afișate în câmpurile corespunzătoare. Un câmp trebuie să afișeze toate instanțele coloanei asociate; în consecință, fiecare înregistrare – instanță a grupului este reprezentată de un cadru repetitiv (*repeating frame*).

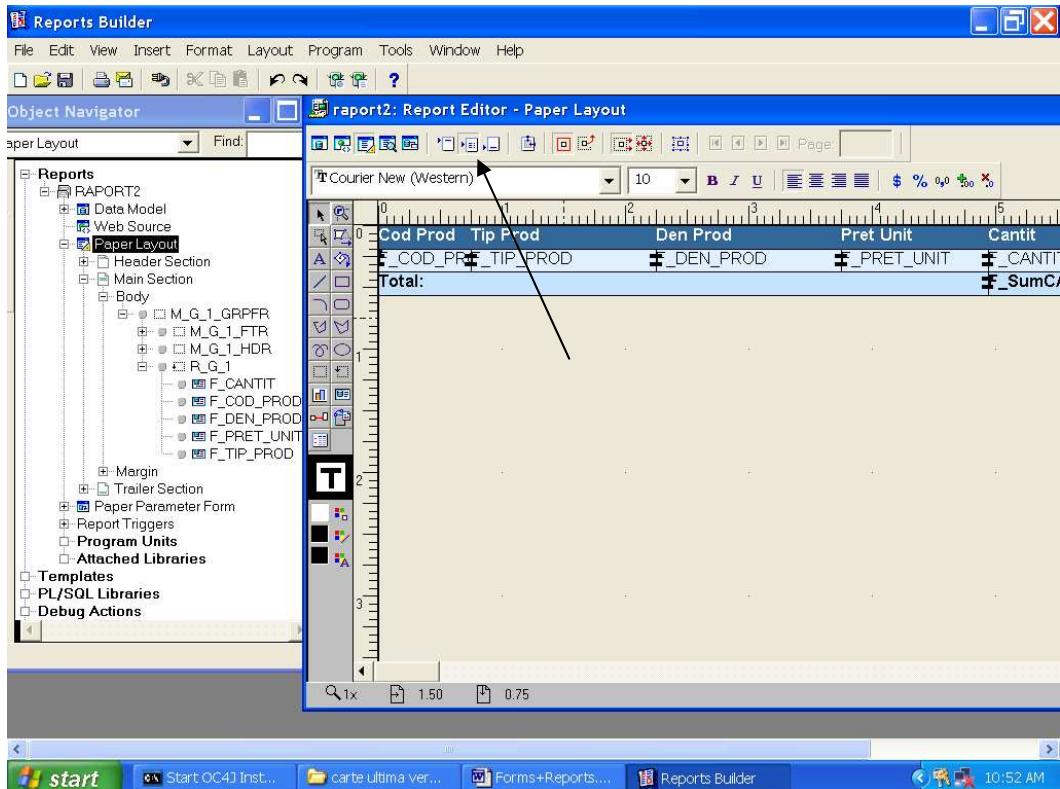


Modificările făcute în *Data Model*, nu se vor reflecta automat în *Layout*. Toate aceste eventuale modificări din *Data Model* trebuie făcute manual și în *Layout*: o coloană care a fost ștearsă din modelul de date va trebui ștearsă și în *layout*; dacă adăugăm manual în fraza *select* o coloană, aceasta va apărea în modelul de date la nivelul cel mai de jos. Utilizatorul va trebui să o tragă cu *mouse-ul* la nivelul dorit, apoi să facă modificarea corespunzătoare și în *layout*. Din păcate, la acest nivel nu ne putem permite să invocăm *Layout Wizard*-ul, care îmi recreează macheta, distrugând-o pe cea veche (asupra căreia se presupune că s-au efectuat rafinări complexe ce vor fi pierdute).

Următorul mod de vizualizare a raportului este accesat cu ajutorul butonului *Web Source*. Vom reveni ulterior asupra acestuia, trecând acum la macheta pe care și-o construiește *Report Builder*-ul pentru a printa raportul și definită în modul *Paper Layout*. Obiectele specifice acestui mod de vizualizare definesc poziționarea elementelor în pagină, precum și modul în care vor fi afișate datele, elementele text și graficele. Cele mai des întâlnite obiecte, specifice modului *Paper Layout* sunt:

- *Cadru repetitiv (repeating frame)* – conține obiecte care sunt afișate o singură dată pentru fiecare înregistrare a grupului asociat; cadrele repetitive sunt figurate ca dreptunghiuri pe ale căror laturi se află săgeți, în funcție de modul de expandare (găsiți explicații detaliate mai jos);
- *cadru (frame)* - grupează mai multe obiecte și se afișează o singură dată;
- *câmp (field)* – conține date și modul lor de formatare;
- *boilerplate* – conține text sau imagini ce pot apărea oriunde în raport.

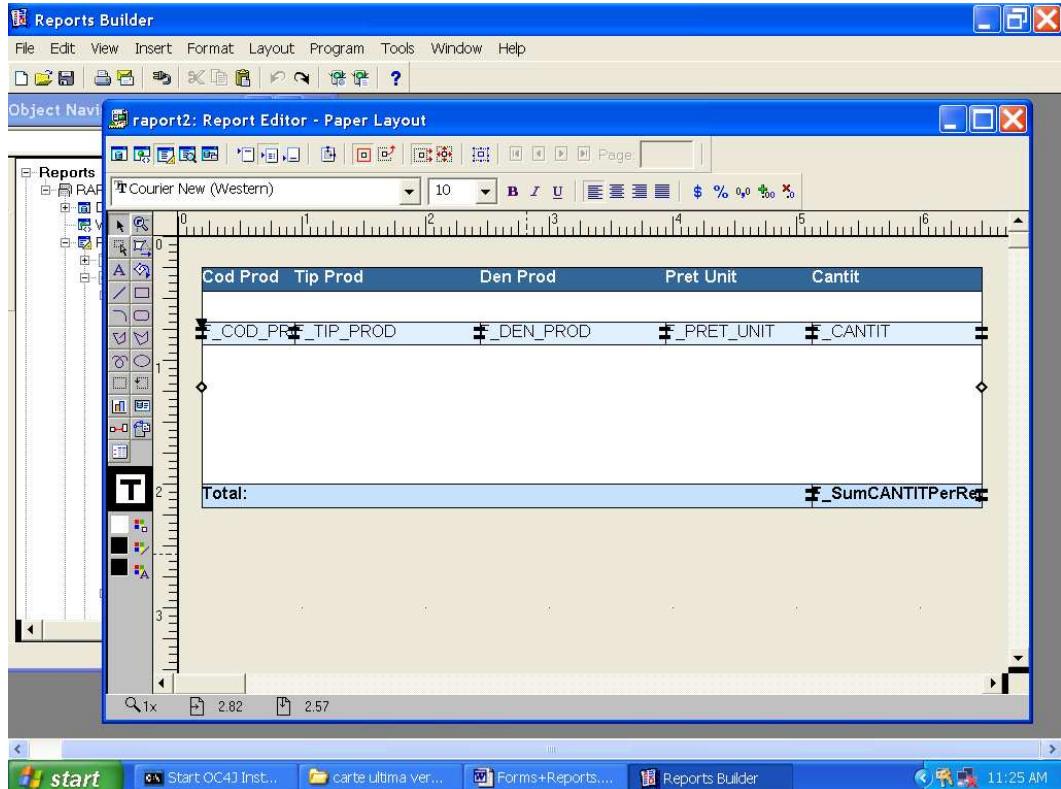
ACTIONÂND, ÎN FERESTRA *Report Editor*, butonul corespunzător modului de vizualizare *Paper Layout* va fi afișată următoarea fereastră:



Remarcați, în *Object Navigator*, existența a trei sub-noduri ale nodului *Paper Layout*: *Header Section* (conținând informația ce va fi afișată la începutul raportului), *Main Section* (conținutul efectiv al raportului) și *Trailer Section* (conținând informația ce va fi afișată la sfârșitul raportului). În fereastra *Report Editor* este apăsat butonul corespunzător secțiunii *Main* (vezi săgeata). Pentru afișarea celorlalte secțiuni se vor actiona butoanele corespunzătoare (alăturate celui prezentat anterior).

Să rearanjăm obiectele în machetă, mărind spațiul dintre ele. Această acțiune are ca scop înțelegerea tipurilor de obiecte ce pot fi manipulate în modul de vizualizare curent. Mutarea obiectelor trebuie făcută însă cu multă atenție. Cele mai grave și dese erori provin din poziționarea unui câmp care trebuie să afișeze o întreagă coloană de date în afara cadrului repetitiv ce îl conține. Selectăm în *Object Navigator* întregul cadrul *M_G_1_GRPFR* (dând click pe el). În fereastra *Report Editor* mutăm întregul grup, folosind săgețile (jos și dreapta). Același rezultat poate fi obținut și cu ajutorul mouse-ului. Selectăm în continuare cadrul *M_G_1_FTR*, care generează în raport ultima linie, în care este afișat totalul. În fereastra *Report Editor* mutăm și acest cadrul mai jos. În sfârșit, selectăm cadrul repetitiv *R_G_1* și procedăm în același mod. Observați, în *Object Navigator*, dreptunghiul marcat cu săgeată asociat cadrului *R_G_1*, care indică faptul că acesta este de tip repetitiv. Cadrele anterioare pot fi selectate, de asemenea, prin

selectarea unui obiect conținut în ele și acționarea butonului *Select Parent Frame*. Rezultatul deplasărilor anterioare de cadre este următorul:



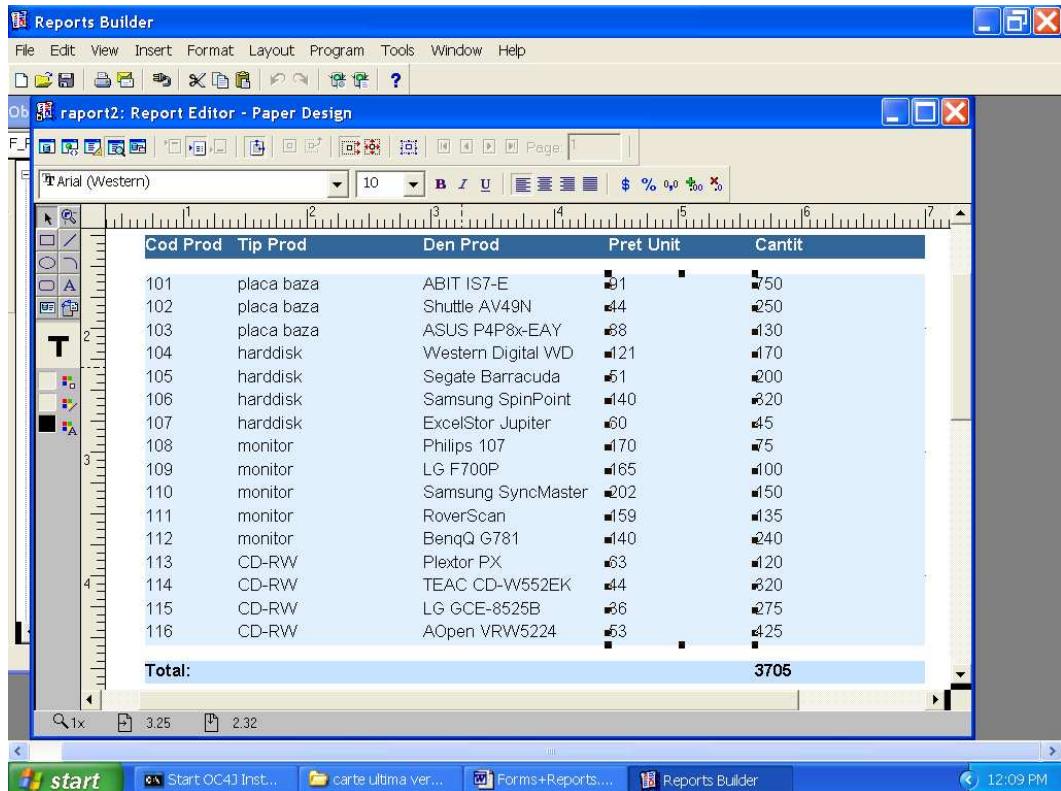
Să urmărim încă o dată cadrele generate pentru acest raport: cadrul părinte este numit implicit *M_G_1_GPRFR* și este încadrat în *Report Editor* într-un dreptunghi marcat pe laturile verticale cu câte un romb. Acest romb indică tipul de elasticitate a cadrului (în cazul nostru, se expandează pe verticală, astfel încât să poată cuprinde întregul volum de date). Există mai multe tipuri de elasticitate, iar acestea pot fi setate cu ajutorul inspectorului de proprietăți al cadrului, capitolul *General Layout*:

- cadrul cu elasticitate fixă (*fixed elasticity*) - mărimea cadrului nu variază în funcție de dimensiunea sau numărul obiectelor pe care le include; este reprezentat printr-un dreptunghi fără marcaje adiționale;
- cadrul ce se poate contracta (*contracting elasticity*) – dimensiunea cadrului se poate micșora în funcție de mărimea obiectelor pe care le include, dar nu poate crește; este reprezentat printr-un dreptunghi având câte un cerc pe două laturi opuse;

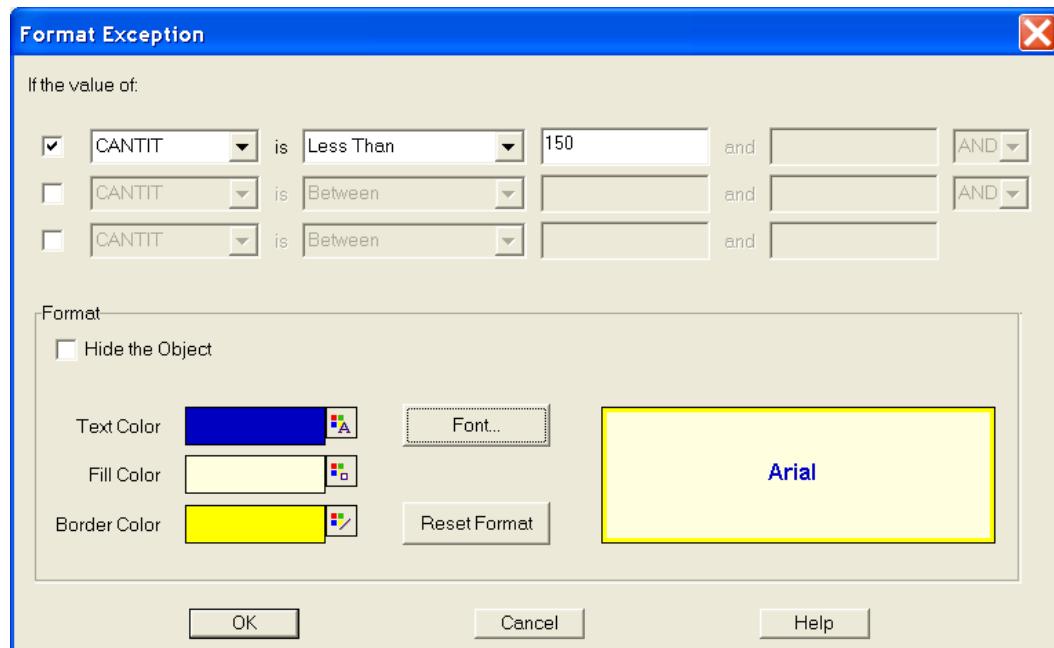
- cadru ce se poate expanda (*expanding elasticity*) - dimensiunea cadrului se poate mări în funcție de mărimea obiectelor pe care le include, dar nu se poate micșora; este reprezentat printr-un dreptunghi având câte două liniuțe paralele pe două laturi opuse;
- cadru cu elasticitate variabilă (*variable elasticity*) – dimensiunea cadrului se ajustează (mărinind sau micșorându-se) în funcție de obiectele conținute în cadrului; este reprezentat printr-un dreptunghi având câte un romb pe două laturi paralele.

În funcție de laturile dreptunghiului pe care se află marcajele, elasticitatea poate acționa pe verticală sau pe orizontală. Adăugând doar faptul că săgeata în jos atașată dreptunghiului cadrului *G_R_I* semnifică faptul că acesta este repetitiv, suntem în măsură să înțelegem toate elementele ce apar în fereastra *Report Editor*, modul *Paper Layout*.

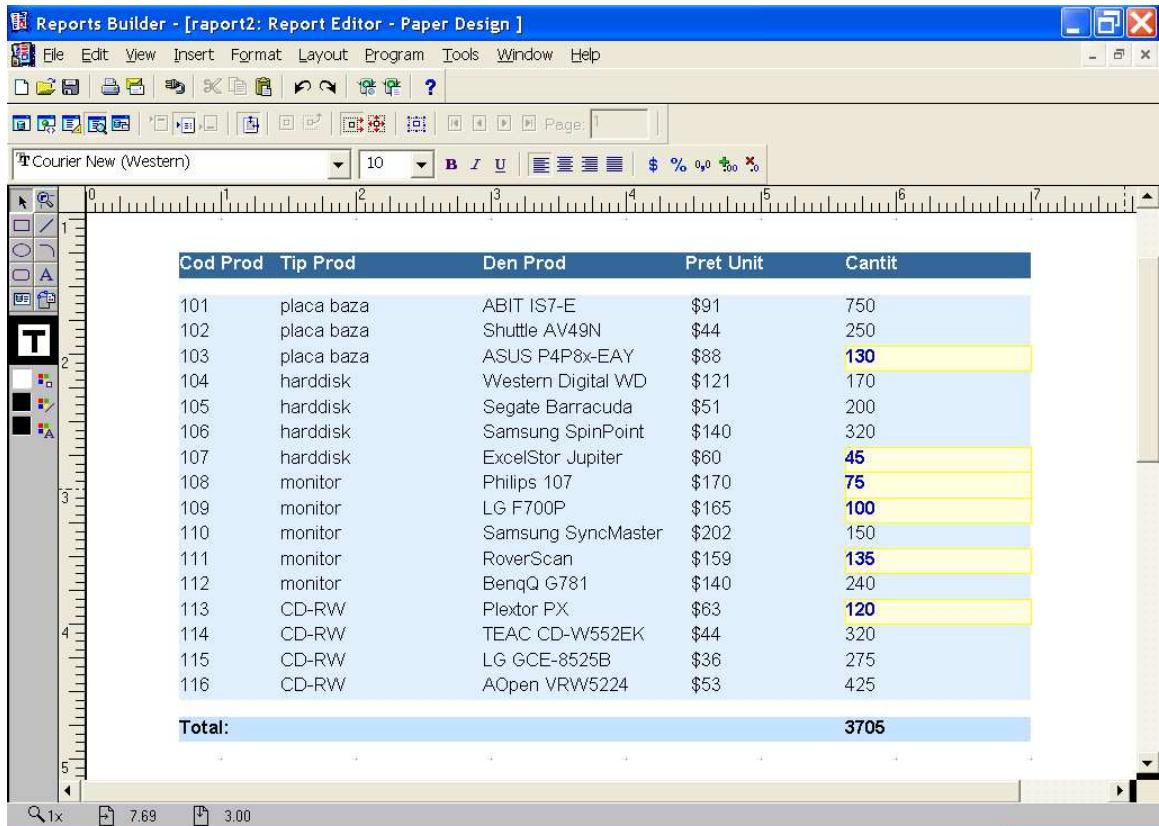
Acționând următorul buton, *Paper Design*, obținem un *preview* al raportului proiectat pentru tipărire. Acesta este un editor de tipul *WYSIWYG*, în care putem face modificări direct asupra setului de date reale.



Selectați coloana *Pret Unit* și încercați să formatați modul de afișare a sa acționând grupul de butoane din bara de instrumente *formatting* (asemănătoare cu cea din aplicațiile Microsoft):  Ultimul grup de butoane permite adăugarea automată a unuia din semnele dolar, procent sau virgulă. Să adăugăm semnul monedei americane și să presupunem că dorim formatări mai complexe, cum ar fi: toate cantitățile mai mici decât 150 să fie marcate (avertizându-ne, astfel, că respectivul produs tinde să fie epuizat). Ne poziționăm pe coloana *Cantit* și din meniu de context (obținut dând *click* dreapta pe respectiva coloană) alegem opțiunea *Conditional Formatting*.... Din caseta de dialog afișată alegem opțiunea *new* (creăm o nouă condiție de formatare) și trecem la următorul pas, specificarea condițiilor:



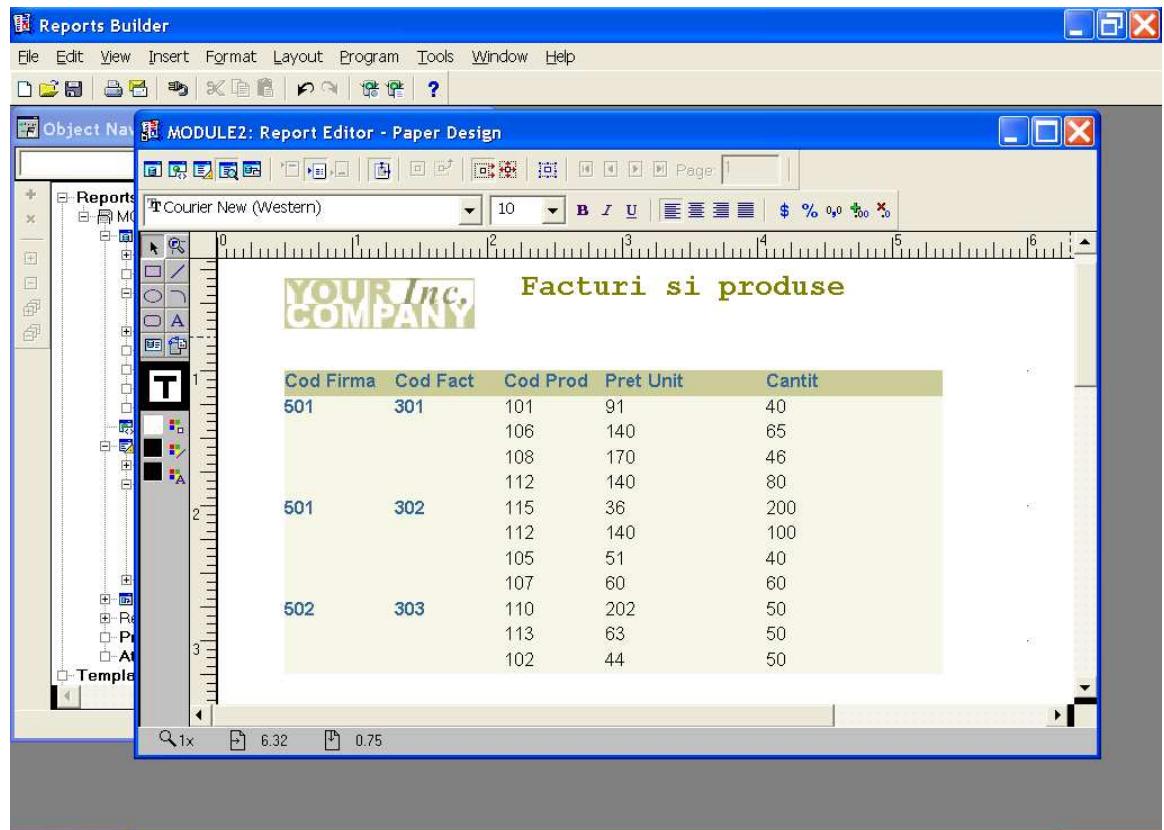
Specificăm, așa ca în figura anterioară, condiția (valoarea câmpului *cantit* mai mică decât 150), iar în partea de jos a ferestrei precizăm modul în care va fi formatat câmpul dacă îndeplinește condiția anterioară. Am ales culoarea textului - albastră, iar pentru hașurare și chenar – două nuanțe de galben. Rezultatul va fi următorul:



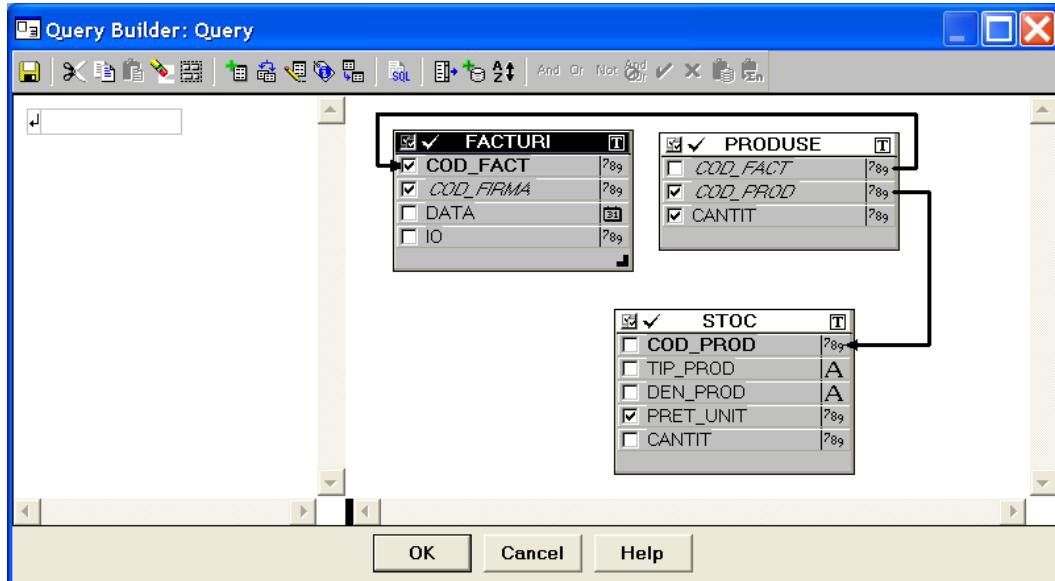
Putem adăuga, de asemenea, numărul paginii curente, precum și data, eventual ora la care a fost generat raportul. Acest lucru se face într-un mod asemănător aplicațiilor *Microsoft Office*, prin selectarea din meniul *Insert* a opțiunilor *Page Number*, respectiv *Date and Time*. Opțiunea *AutoText* permite stocarea și inserarea automată a unor porțiuni de text, grafice, câmpuri și a altor elemente care sunt folosite frecvent.

3.4. Rapoarte cu grupuri master-detail

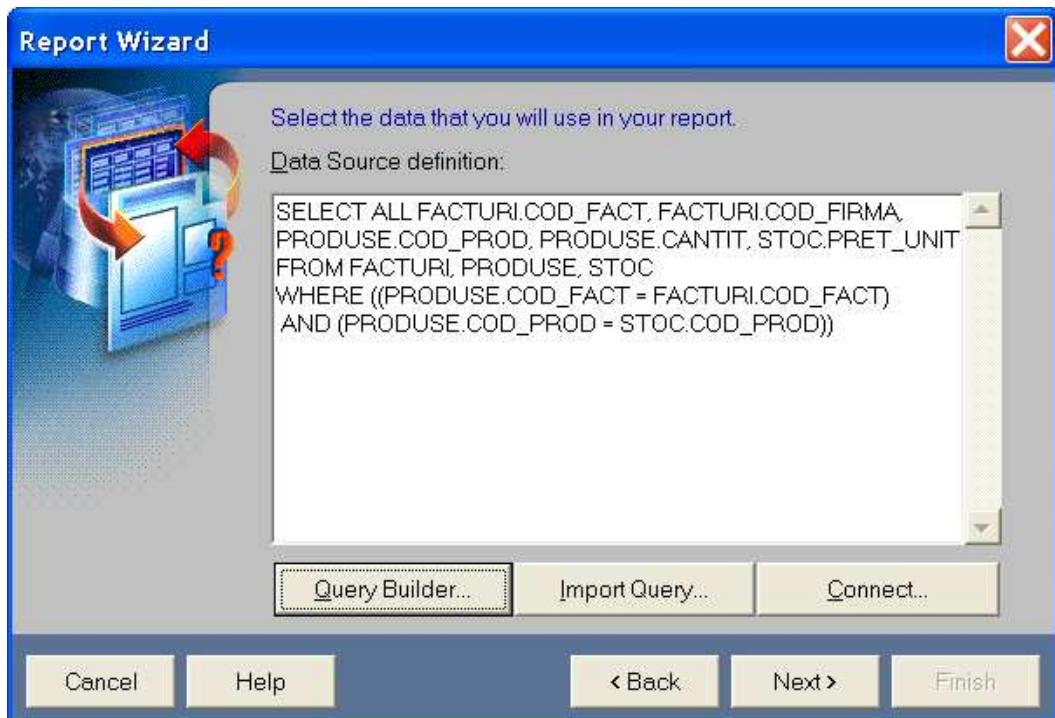
Am creat și analizat în capitolele precedente un raport simplu, bazat pe un singur grup de date. O să continuăm cu rapoarte mai complexe, analizând structura datelor care sunt afișate și modul în care poate fi modificat raportul prin manipularea modelului de date. Să generăm un nou raport, în care să afișăm pentru fiecare firmă toate facturile existente (de intrare sau ieșire). Pentru fiecare factură vom afișa produsele corespunzătoare, precum și cantitatea. Raportul final va arăta astfel:



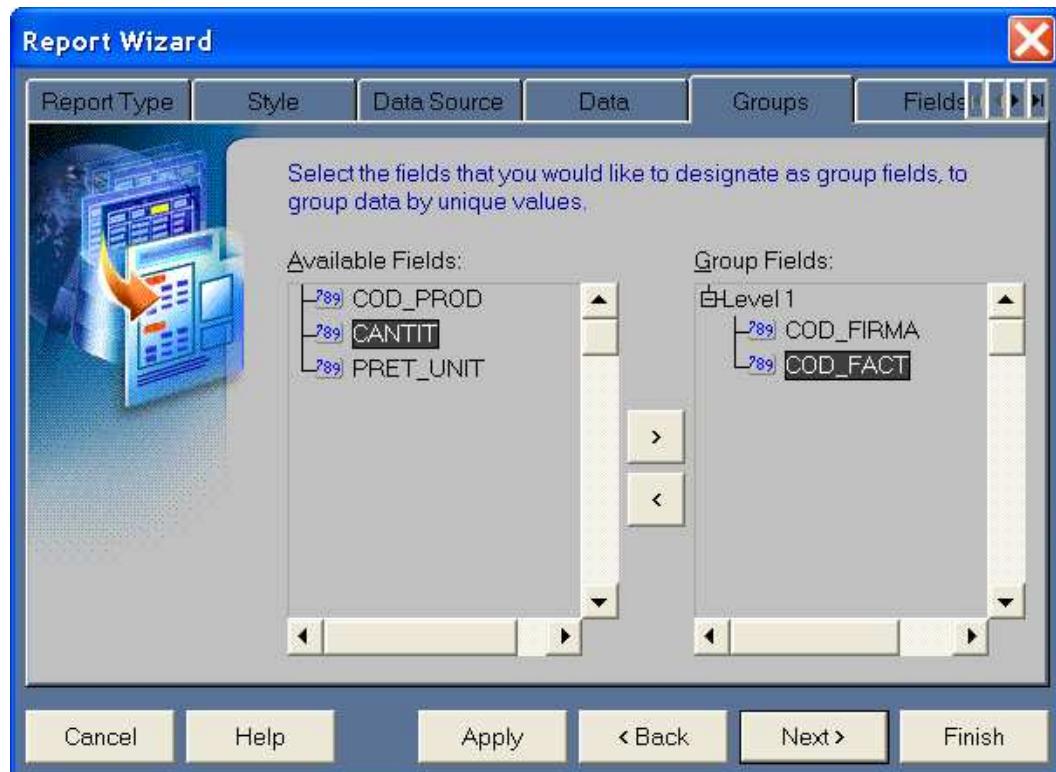
Să invocăm bine-cunoscutul *Report Wizard*. În pagina *Style* dăm un titlu raportului (de exemplu, *Facturi și produse*) și alegem tipul de raport *Group Left*. Sursa de date va fi, ca și până acum, rezultatul unei cereri *SQL*. La pasul următor invocăm *Query Builder*-ul și selectăm trei tabele: *produse*, *facturi* și *stoc*. Relația de *join* dintre cele trei tabele este generată automat, iar noi precizăm ce coloane dorim să apară în raport, bifând casetele de validare din stânga fiecăreia:



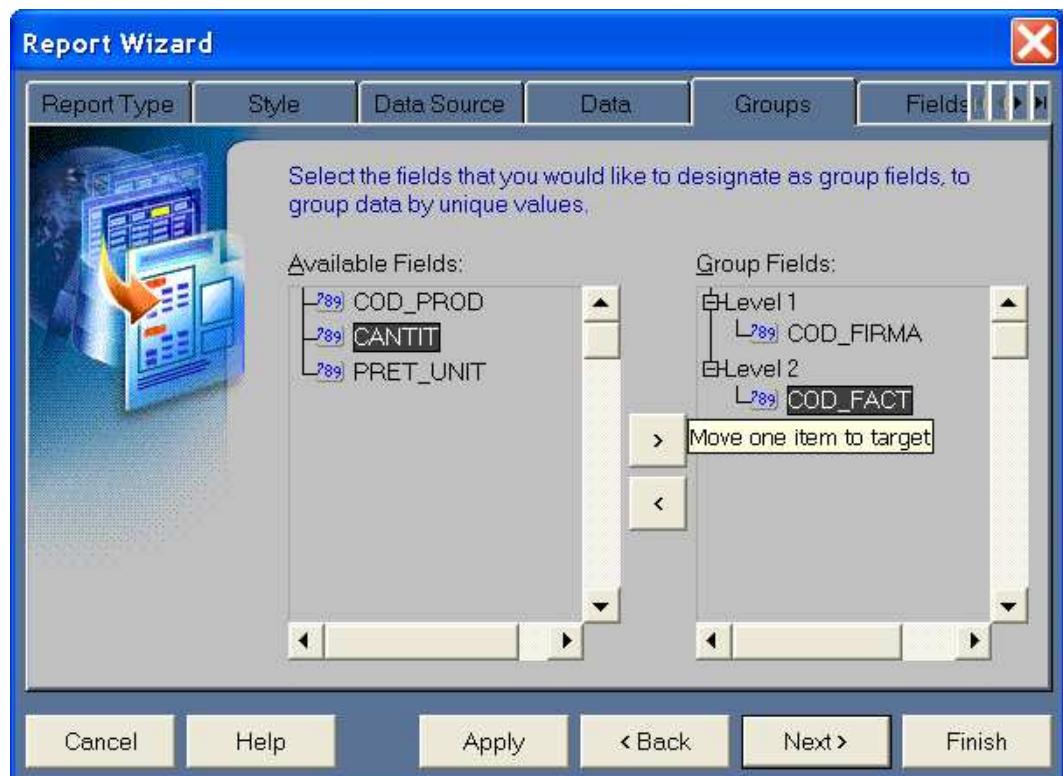
Acționând butonul *OK*, va fi generată automat fraza *SQL*:



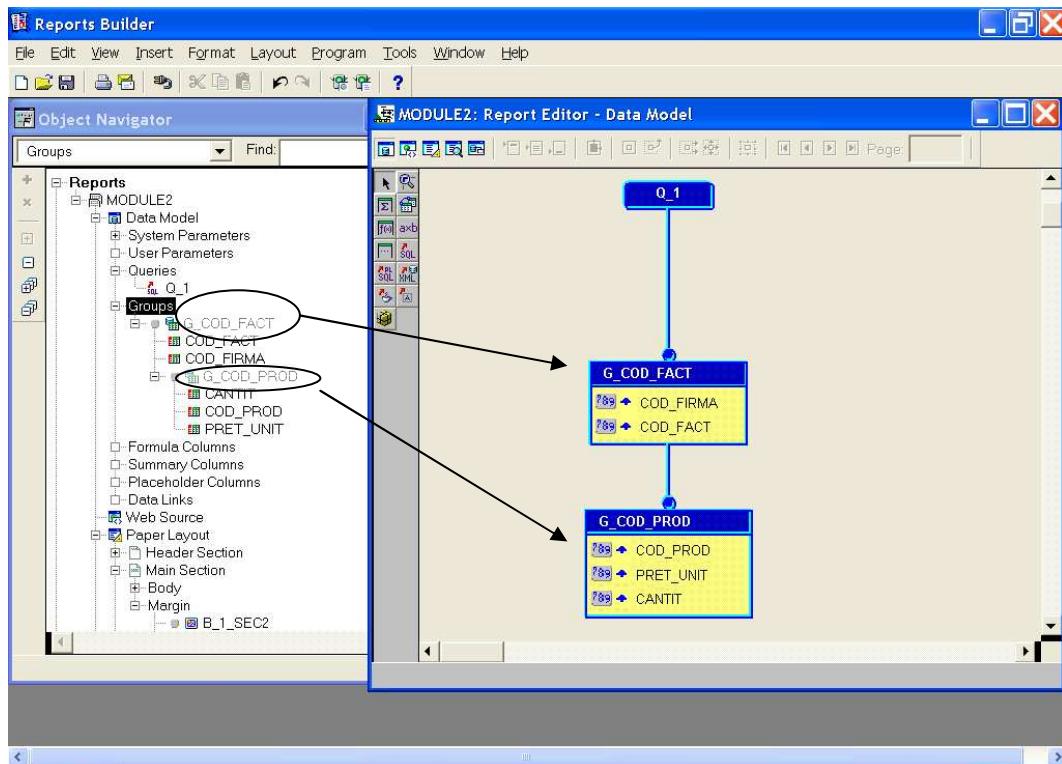
Următorul pas al *wizard*-ului permite definirea unor grupuri de date; dorim ca produsele să fie grupate pe firme, iar în cadrul fiecărei firme – după factură. Pentru a realiza acest lucru, procedăm în felul următor: în caseta *Groups* a *wizard*-ului mutăm din zona *Available Fields* în zona *Group Fields* câmpul *cod_firma*. Astfel, va fi creat un grup după acest câmp. Dorim ca, în cadrul acestui grup, produsele să fie grupate după codul facturii. Putem proceda în unul din două moduri: având selectat grupul *cod_firma* din caseta din dreapta, mutăm câmpul *cod_factura* din zona câmpurilor disponibile în cea a grupărilor. Rezultatul va fi următorul:



Al doilea mod în care putem proceda este să mutăm câmpul *cod_fact* în zona *Groups* având selectat în caseta din dreapta *Level 1*. Rezultatul va fi altul:



Diferența dintre cele două metode este că în primul caz vor fi create în *Data Model* doar două grupuri de date, pe când în cazul al doilea vor fi create trei grupuri. Care din cele două metode este cea corectă? Analizându-le pe amândouă, vor hotărî împreună acest lucru, pe parcursul prezentului capitol. Deci, să procedăm conform primului caz; în caseta următoare, *Fields*, precizăm care din câmpuri vor fi folosite la afișare. Le selectăm pe toate și le mutăm în zona *Displayed Fields*. Ordinea în care sunt câmpurile afișate la printarea raportului este aceeași cu ordinea în care se găsesc în zona *Displayed Fields* și poate fi schimbată în mai multe moduri; unul dintre ele este rearanjarea câmpurilor chiar în această casetă, prin *drag&drop*. Următorii pași ai *wizard-ului* sunt cei deja cunoscuți, iar rezultatul final ar trebui să fie raportul din figura de la începutul prezentului subcapitol. Să analizăm modelul de date pe care se bazează acest raport:

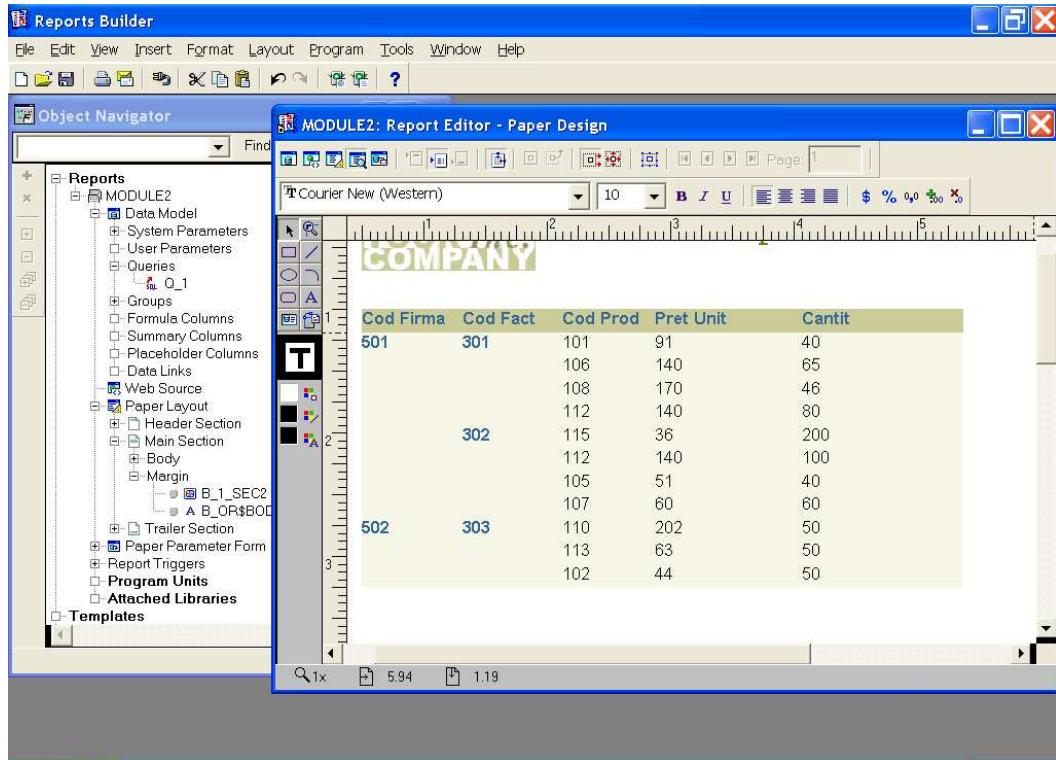


A fost generată o cerere *SQL* (prezentată anterior) și două grupări, unul *master*, celălalt *detail*. Această structură poate fi observată atât în *Data Model*, cât și în *Object Navigator*. Să ne reamintim că un raport tabular se bazează pe o cerere care generează un singur grup, pe când rapoartele de tip *Group Above* și *Group Left* conțin o cerere și două sau mai multe grupuri.

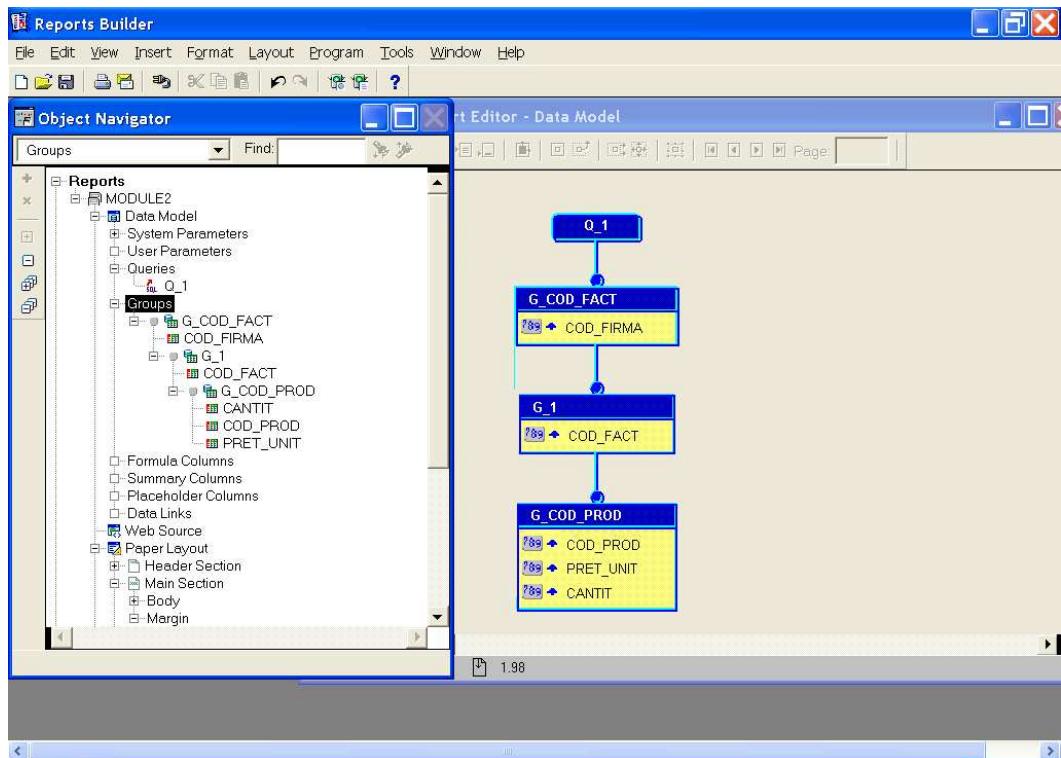
Pentru fiecare înregistrare din grupul *master* vor fi listate toate înregistrările din grupul *detail*. Săgețile afișate în *Data Model* în stânga fiecărei coloane indică faptul că înregistrările sunt sortate. Proprietatea corespunzătoare (accesibilă prin *Property Inspector*) se numește *Break Order*. Săgețile sunt, însă, valabile doar pentru grupul *master*. În grupul *detail* ele nu au nici un efect, sortarea realizându-se prin modificarea de către utilizator a cererii *SQL*, introducând manual o clauză *order by*. Este recomandată eliminarea săgeților care nu sunt necesare, pentru îmbunătățirea performanțelor raportului.

Observați, în raportul generat, un lucru mai puțin plăcut: pentru o firmă care are mai multe facturi, codul firmei va fi afișat de mai multe ori, odată cu fiecare factură. Înțelegând corect modelul de date, realizăm și de ce se întâmplă acest lucru. Folosind acest model de date, nu putem determina afișarea codului firmei o singură dată. Putem realiza însă acest lucru modificând *Data Model*: pentru fiecare firmă, trebuie să am o

grupare *detail* care să afișeze toate facturile. Acest scenariu corespunde celui de al doilea mod de creare a grupărilor, folosind două niveluri (în acest subcapitol spuneam că putem crea în *Report Wizard* grupul *cod_factura* având selectat *Level 1* în pagina *Groups*, caseta *Group Fields*. Rulați *Report Wizard* în modul *reentrant*, faceți modificarea corespunzătoare și veți observa că raportul arată în felul următor:



Modelul de date este, însă, altul față de cel prezentat anterior:

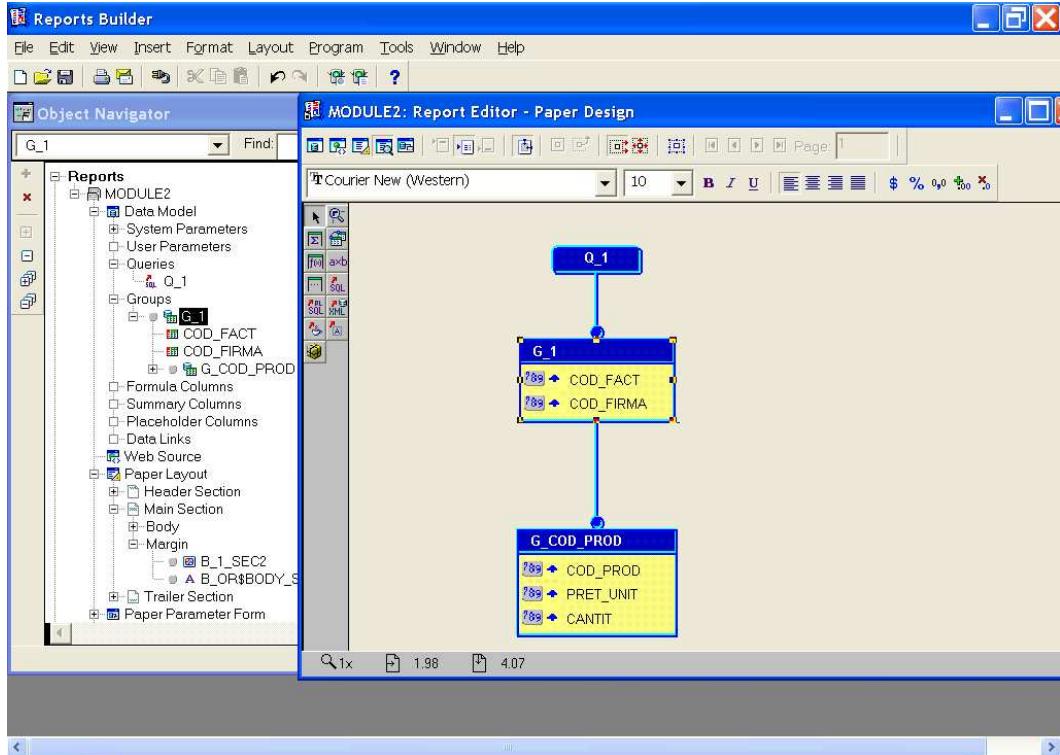


După cum observați, în *Oracle9i Reports* grupurile sunt organizate conform unei ierarhii. Aceasta poate fi modificată invocând *Report Wizard*-ul în modul *reentrant*; există însă situații când nu vrem să folosim acest mod de a modifica *Data Model* (de exemplu, în cazul când vrem să evităm pierderea unor rafinări efectuate după încheierea procesului de generare a raportului cu ajutorul *wizard*-ului). În acest caz, putem crea grupuri noi chiar în *Report Editor*.

3.5. Modificarea rapoartelor folosind *Data Model*

Modelul de date definește datele care apar în raport și ierarhia acestora, deci structura raportului, însă nu permite definirea nici unui atribut de formatare a raportului. Modelul de date este creat odată cu raportul, la rularea instrumentului *Report Wizard*. Arătam într-un capitol anterior că pasul următor rulării acestui *wizard* este rafinarea manuală a raportului. După ce toate aceste etape au fost parcurse, poate apărea necesitatea modificării structurii datelor, prin crearea unui grup nou, introducerea unor coloane suplimentare etc. Toate aceste modificări vor fi făcute manual, în *Data Model*.

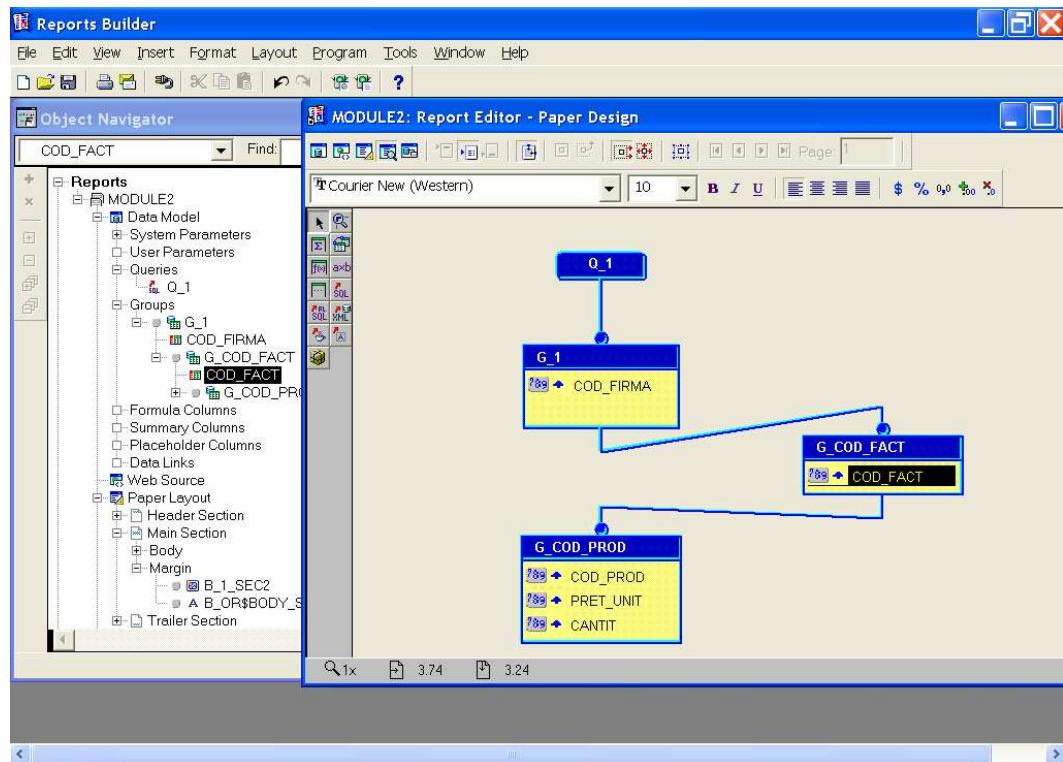
Să revenim la modelul de date anterior:



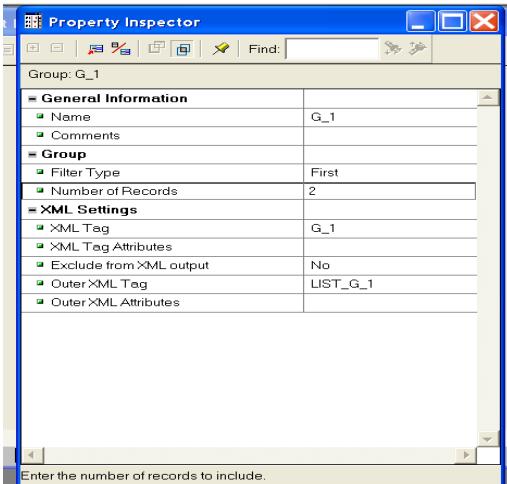
Pentru evitarea afișării de mai multe ori a codului clientului vom crea, de aceasta dată manual, un nou grup de date. Astfel, primul grup *master* va conține câmpul *cod_firma*, iar pentru fiecare firmă va fi generat grupul *detail* al facturilor asociate respectivei firme. La rândul său, acest grup va avea ca grup copil pe *G_COD_PROD*.

3.5.1. Crearea manuală a grupurilor

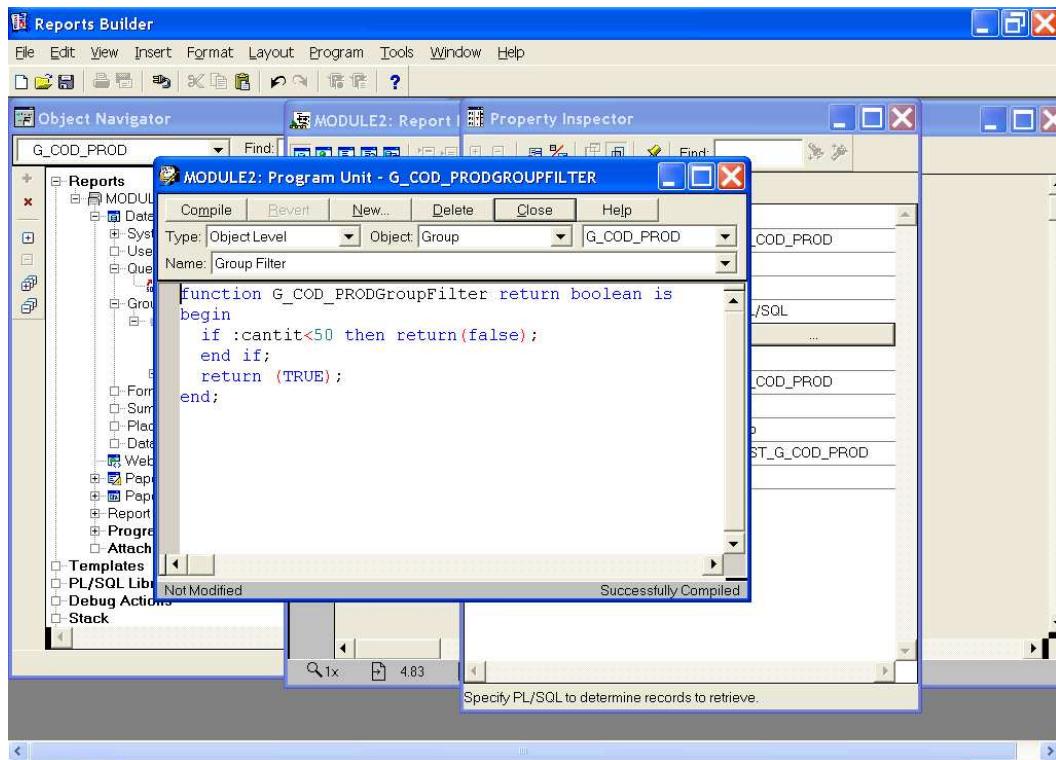
Putem crea un grup *master* al grupului *G_1* trăgând câmpul *cod_fact* în partea stângă, apoi eliberând *mouse*-ul sau putem crea un grup *detail* pentru *G_1* trăgând câmpul *cod_firma* în partea dreaptă și eliberând *mouse*-ul; în cel de al doilea caz, *Data Model* va arăta astfel:



Odată creat noul grup, în el pot fi mutate și alte coloane, trăgându-le cu *mouse-ul* în ierarhie **de sus în jos**, dintr-un grup superior în cel nou sau într-unul inferior lui. Ordinea în care grupurile apar în *Data Model* are efect asupra modului în care vor fi afișate. Trebuie să ne asigurăm că întotdeauna grupurile părinte (*master*) sunt afișate înaintea grupurilor copil (*detail*).



Ne întoarcem la primul model de date, cu doar două grupuri. Proprietățile grupurilor pot fi vizualizate și modificate folosind *Inspector Property*: observați posibilitatea introducerii unui filtru la nivel de grup, prin selectarea proprietății *Filter Type*. Există trei tipuri de filtru: *First*, urmat de specificarea unui *Number of Rows* determină afișarea, pentru fiecare înregistrare *master*, a numărului de înregistrări *detail* specificate de utilizator. Următoarea proprietate, *Last*, acționează asemănător, însă produce afișarea ultimelor rânduri din gruparea *detail*. Aceste proprietăți sunt foarte utile în faza de testare a raportului. Cea de a treia valoare a proprietății *Filter Type*, *PL/SQL*, permite inserarea de cod *PL/SQL* pentru stabilirea filtrului. Filtrul *PL/SQL* este o funcție booleană, care întoarce, deci, una din valorile *true* sau *false*. Funcția se declanșează pentru fiecare instanță: dacă este întoarsă valoarea *true*, înregistrarea este afișată; în caz contrar, ea este eliminată din raport. Să presupunem că dorim să afișăm doar acele produse pentru care cantitatea este mai mare de 50. Funcția booleană care realizează acest lucru este:



3.5.2. Adăugarea manuală a coloanelor

Coloanele existente în *Data Model* provin, aşa cum am arătat anterior, din cererea *select* care populează raportul folosind coloanele existente în tabelele bazei de date. *Reports Builder*-ul impune câteva restricții în manipularea acestor tipuri de coloane:

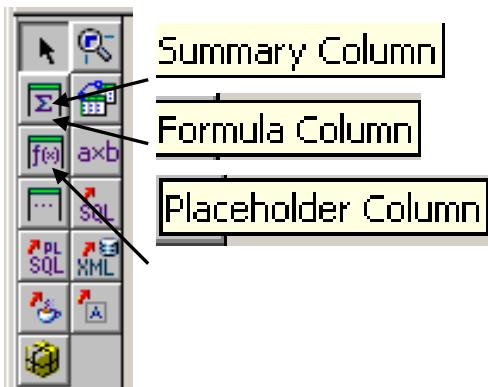
- O parte din proprietățile lor sunt *read-only*: *Name*, *Column Type*, *Data Type* și *Width*. Aceste informații provin din tabelele bazei de date și nu pot fi modificate în *Report Builder*;
- Nu pot fi șterse direct în grup (cu tasta *Del*, de exemplu), ci trebuie eliminate din cererea *select*.

În afara coloanelor bazate pe câmpurile tabelelor, utilizatorul poate crea propriile coloane, care pot fi de mai multe feluri:

- *Summary* – aplică funcțiile statistice standard unei alte coloane sau unui grup de coloane; poate fi definită la momentul creării raportului, folosind instrumentul *Report Wizard*; numele acordat implicit de către *Reports Builder* unei astfel de coloane este *CS_1*;

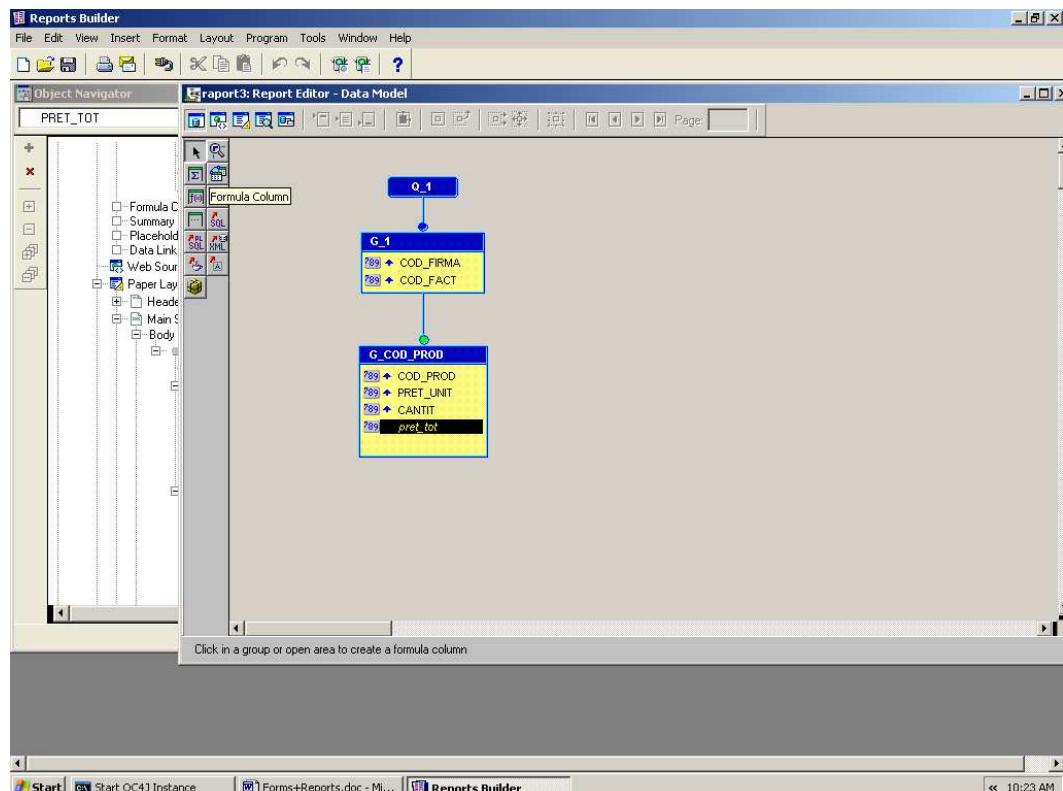
- *Formula* – se calculează ca rezultat al unei funcții scrise de utilizator în *PL/SQL*; numele acordat implicit de către *Reports Builder* unei astfel de coloane este *CF_1*;
- *Placeholder* – primește valoare în *runtime*, de la un alt obiect; numele acordat implicit de către *Reports Builder* unei astfel de coloane este *CP_1*. O coloană de tip *placeholder* este, de fapt, un *container* populat cu elemente aduse, în general, dintr-o coloană sau dintr-un *trigger*.

Fiecare din cele trei tipuri de coloane poate fi creată folosind *toolbar-ul* vertical din *Report Editor*:

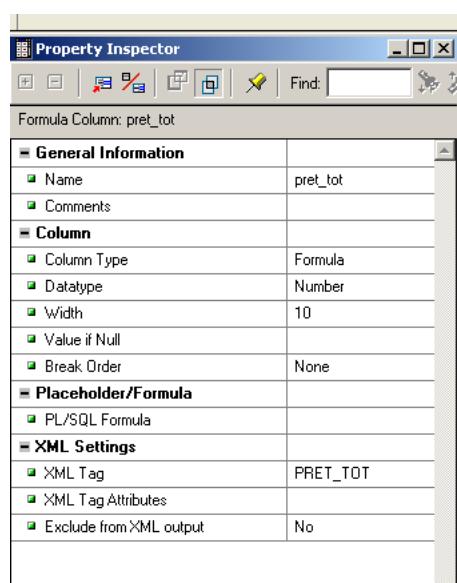


Să ne amintim că raportul asupra căruia lucrăm de vreo două subcapitole încoace afișează situația comenziilor corespunzătoare fiecărei facturi, pentru fiecare produs fiind afișate numele acestuia, cantitatea și prețul unitar. Să presupunem că, după formatare complexe asupra machetei, a apărut necesitatea inserării unei coloane noi, care să afișeze pentru fiecare produs prețul total, adică valoarea produsului (*cantitatea * prețul unitar*). Nu dorim să alterăm rafinările efectuate asupra *layout-ului*, aşa că vom introduce manual o nouă coloană de tip *formula*.

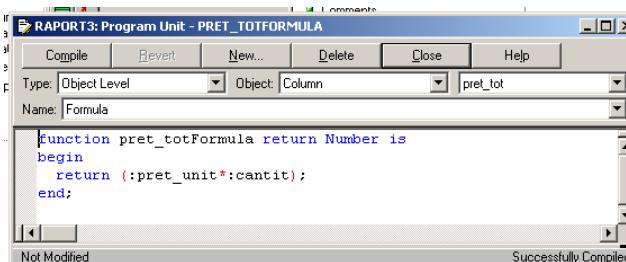
Să creăm, în *Data Model*, o nouă coloană care să calculeze produsul dintre prețul și cantitatea corespunzătoare fiecărui produs. Pentru aceasta selectăm butonul *Formula Column*, apoi dăm *click* în grupul *G_COD_PROD*, unde este locul noii coloane. Astfel, modelul de date a fost modificat:



Acționând paleta de proprietăți a coloanei nou create, putem modifica numele acestieia și îi acordăm funcționalitate. O coloană de tip *formula* va avea valorile calculate conform unei funcții PL/SQL definite de utilizator. Funcția trebuie să întoarcă o singură valoare, de tipul specificat conform proprietății *Datatype* și de lungimea definită prin proprietatea *Length*.



Acționând butonul corespunzător proprietății *PL/SQL Formula* va apărea caseta bine-cunoscută, care permite inserarea codului PL/SQL ce se va executa la fiecare instanță a grupului căruia îi aparține câmpul curent. Funcția care va calcula valoarea dorită este:



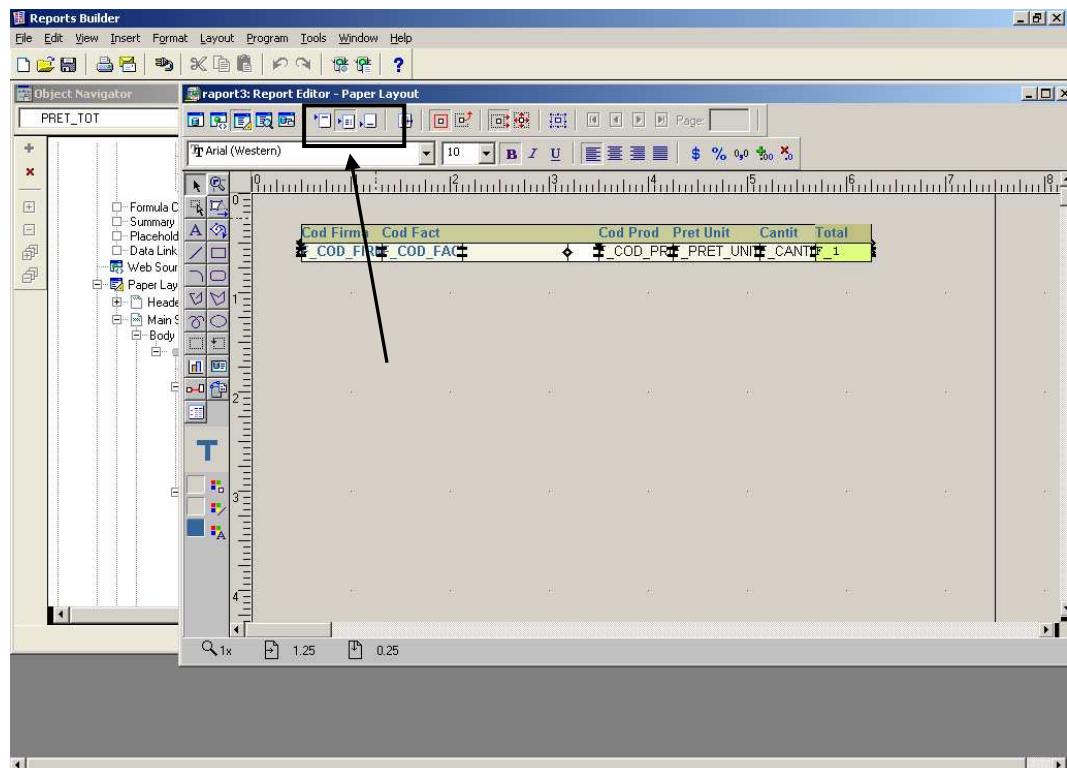
Nu uitați să compilați funcția, acționând butonul corespunzător, *Compile*. Vizualizarea raportului în modul *Paper Design* (care reprezintă, după cum precizam anterior, un instrument *WYSIWYG*) nu prezintă nici o modificare, coloana nou creată neapărând. Ne așteptăm, însă, la acest lucru, deoarece am precizat că modificările efectuate manual în *Data Model* nu se regăsesc automat în *layout*. Modificarea manuală a machetei raportului este o operație deosebit de dificilă, care trebuie efectuată cu multă atenție. Deplasarea elementului grafic ce desemnează un câmp în afara grupului căruia îi aparține respectivul câmp (în *Object Navigator*) duce la o eroare fatală și la imposibilitatea rulării raportului. Pentru a modifica cu succes macheta raportului trebuie să aprofundăm elementele ce o definesc și comportamentul acestora.

3.6. Modificarea raportului folosind *Paper layout*

3.6.1. Cele 3 secțiuni ale raportului: *header*, *main*, *trailer*

Un raport este implementat, din punctul de vedere al aplicației *Oracle Reports*, asemănător cu o carte: informația pe care cererea *select* o aduce din baza de date este depozitată într-o secțiune numită *main*. Putem crea, însă, pagini-antet și pagini ce vor fi atașate la sfârșitul raportului. Acestea pot conține simplu text sau informații din modelul de date. Secțiunea antet (*header section*) conține una sau mai multe pagini care vor fi afișate o singură dată, la începutul raportului. Secțiunea *trailer* se comportă într-un mod similar, dar informația corespunzătoare acesteia va fi afișată la sfârșitul raportului. Cele trei secțiuni ale raportului sunt afișate în *Object Navigator* la nodul *Paper Layout*.

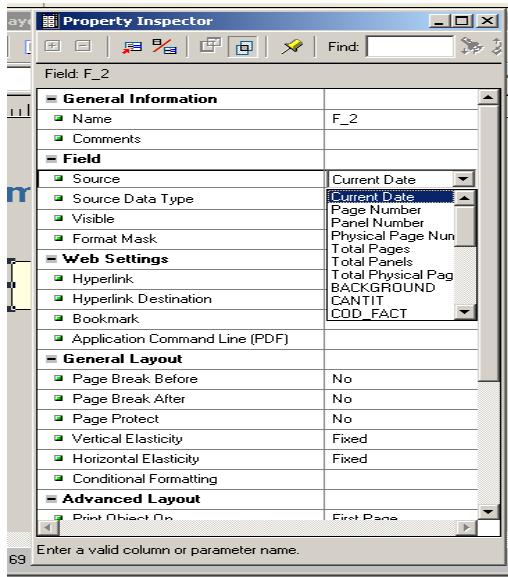
La fel ca în cazul secțiunii *main*, în secțiunile *header* și *trailer* machetele pot fi construite cu ajutorul instrumentelor de tip *wizard* sau manual. Comutarea între cele trei secțiuni ale raportului se face prin acționarea butoanelor corespunzătoare aflate în partea superioară a ferestrei *Paper Layout*:



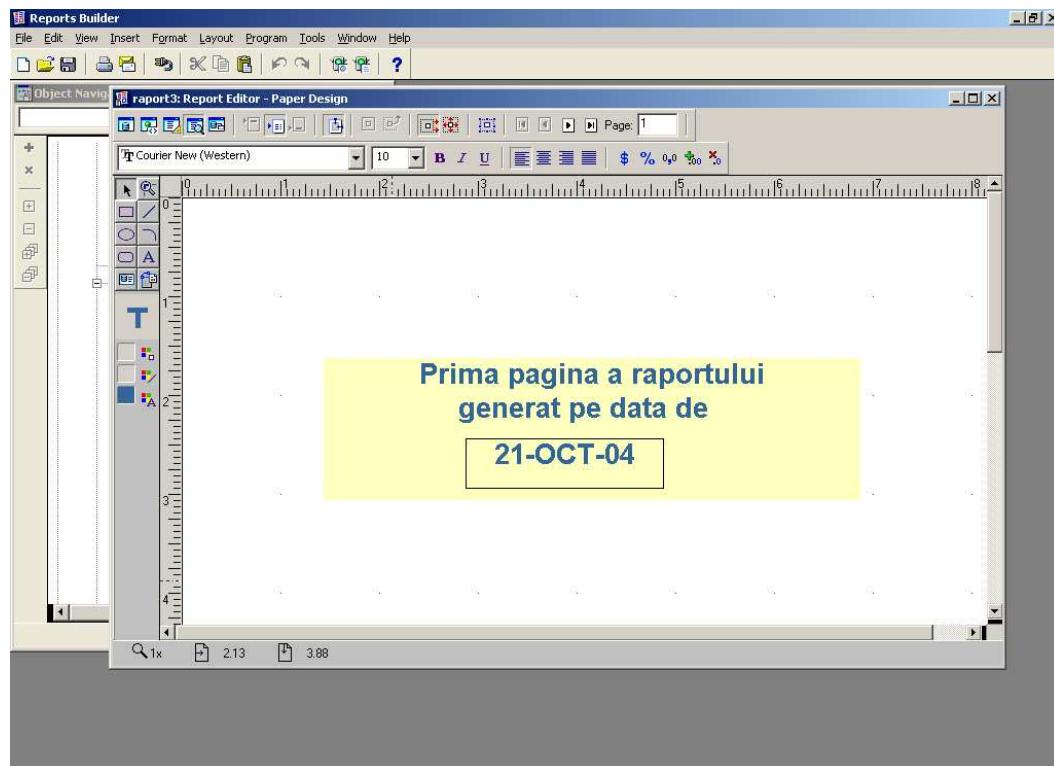
Acționați primul din cele trei butoane; va apărea o zonă fără informații, în care puteți insera informațiile ce doriți a apărea pe prima pagină a raportului. Actionând



butonul *text*,  puteți insera text static, iar cu ajutorul butonului *Field*  puteți insera informația corespunzătoare unuia din câmpurile existente la nivel de raport. Acționați paleta de proprietăți a câmpului nou creat și alegeti, la proprietatea *Source*, afișarea datei la care a fost rulat raportul:



Prima pagină a raportului poate fi vizualizată cu ajutorul instrumentului *Report Editor*, în modul *Paper Design*:



Așa cum precizam anterior, secțiunile *header* și *trailer* pot conține de asemenea informație dinamică, generată în *runtime*. Un exemplu a fost prezentat anterior și a constat în inserarea datei la care a fost generat raportul. Macheta secțiunilor sus-amintite poate fi creată manual sau automat, cu ajutorul instrumentelor de tip *wizard*. Simplă invocare a utilitarului *Report Wizard* va produce rularea lui în modul *reentrant* și afectarea machetei secțiunii *main*. Se dorește, însă, crearea unei machete adiționale, pentru secțiunea *header*, iar acest lucru se obține prin acționarea prealabilă a butonului *Report Block*, ultimul din bara verticală de instrumente aflată în fereastra *Paper Layout* și



marcat prin semnul . Apăsarea acestui buton, urmată de rularea instrumentului *Report Wizard*, va determina crearea unei machete adiționale, pentru secțiunea *header*. Pașii specifici acestui *wizard* sunt deja cunoscuți. Încercați să afișați, pe această primă pagină a raportului, numele firmelor cu care firma curentă are schimburi de marfă, precum și codurile tuturor facturilor operate (grupate pe firme).

3.6.2. Margini

Editorul *Paper Layout* permite atât modificarea corpului celor trei secțiuni ale raportului, cât și setarea marginilor lor. Fiecare pagină fizică (indiferent de secțiunea care o generează) conține două zone: corpul paginii și marginea sa. Comutarea între cele două zone se poate face, la orice moment, acționând butonul *Body/Margin* aflat în zona

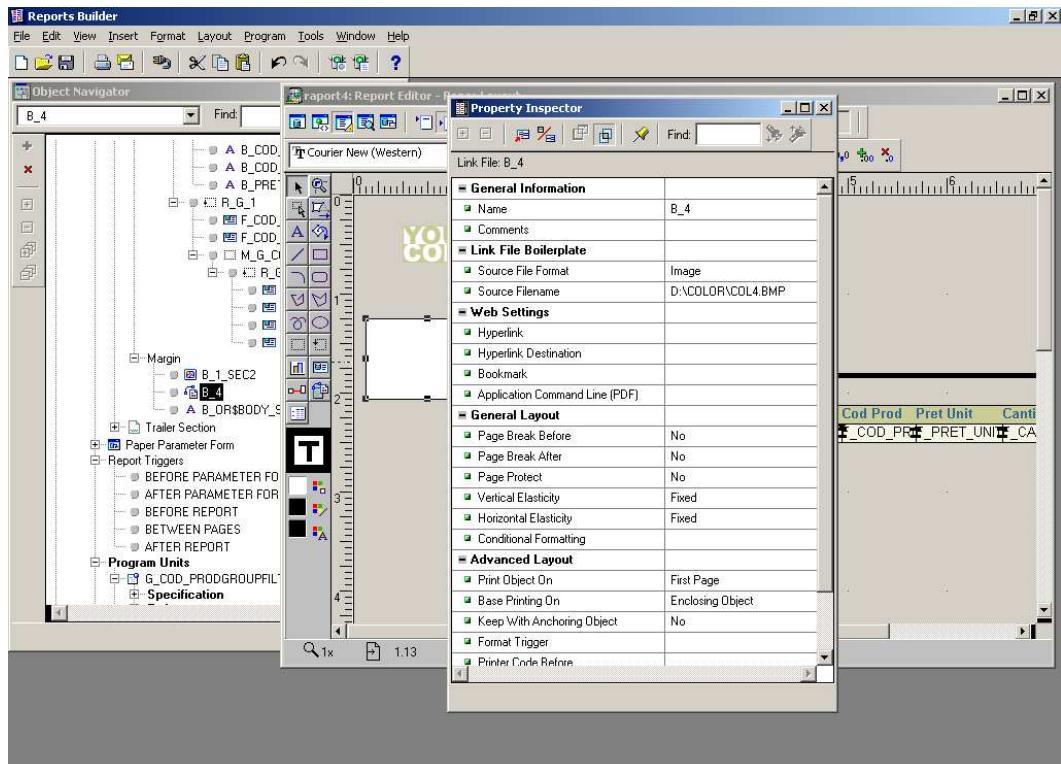


superioară a ferestrei *Paper Layout* și marcat prin icon-ul . Marginile pot conține text static, grafice, imagini, numerotarea paginilor, totaluri etc. Când utilizatorul selectează butonul *Body/Margin* pentru a vizualiza și修改 marginile, zona *body* rămâne, de asemenea, vizibilă, dar nu poate fi modificată (este *read-only*). Ea este încadrată într-un dreptunghi cu laturile negre, care stabilăște granița dintre margini și corpul paginii. Modificarea dimensiunilor marginii poate fi făcută selectând dreptunghiul negru care o încadrează și redimensionându-l, cu ajutorul mouse-ului.

Să încercăm inserarea unei imagini dintr-un fișier în zona *margins*. Apăsăm în bara verticală de instrumente a ferestrei *Paper Layout* butonul *File Link*, reprezentat prin



îmaginea , apoi dăm *click* în zona unde vrem să fie afișată figura. Va fi creat un obiect nou, a cărui funcționalitate o vom stabili folosind paleta sa de proprietăți:



Proprietatea *File Source Format* va trebui setată la valoarea *Image*, iar actionarea butonului atasat proprietății *Source filename* conduce la afișarea ferestrei *Open*, în care putem selecta fisierul al carui continut va fi afișat.

Efectul ultimelor modificari poate fi vizualizat folosind modul *Paper Design*:

