

Introducere

Calculatoarele au fost folosite încă din anii '50 pentru stocarea și procesarea cantităților mari de date. Pentru gestionarea informațiilor specifice unei activități se face de regulă apel la sisteme informatiche. Produsele software din componența acestora localizează și prelucrează datele conținute într-un ansamblu de fișiere aflate pe diferite suporturi fizice.

O bază de date este constituită dintr-un ansamblu structurat de date evolutive, organizate pentru a fi exploataate de diferite programe (aplicații).

Deși orice întreprindere face apel la baze de date pentru păstrarea și gestionarea informațiilor, câteva dintre aplicațiile acestora sunt deosebit de spectaculoase:

- bazele de date ale liniilor aeriene care sunt accesate simultan din sute de agenții pentru a realiza rezervări și vânzări de locuri pentru date și zboruri diferite;
- bazele de date ale băncilor care permit realizarea a mii de tranzacții zilnic;
- bazele de date ale supermagazinelor care sunt accesate atât de la casele de marcat cât și de la echipamentele de inventariere;
- bazele de date ale bibliotecilor care păstrază milioane de titluri și permit localizarea unei lucrări folosind diferite criterii (cuvinte cheie, titlu, autori, domeniu).

Pentru realizarea unei aplicații care folosește baze de date se poate proceda în două moduri:

- a. Se crează baza de date cu ajutorul unei aplicații de tip *server de baze de date* și se scriu apoi aplicațiile care accesează baza de date într-un limbaj care posedă funcțiile necesare accesării serverului (frecvent se folosesc limbajele C++, Java, C# sau Visual Basic);
- b. Se folosește o aplicație de tip *sistem de gestiune de baze de date* (S.G.B.D. sau D.B.M.S. - *database management system*). Un astfel de sistem oferă un ansamblu de instrumente software cu ajutorul cărora se crează atât baza de date cât și aplicațiile prin care aceasta este exploataată. Pentru utilizatorii sistemului de operare Windows cele mai cunoscute sisteme de acest fel sunt Access și Visual FoxPro.

Oracle Database Express Edition (prescurtat **Oracle XE**) este în esență o aplicație din familia serverelor de baze de date. Sesizând însă dificultățile realizării interfeței utilizator-server dintr-o aplicație, firma Oracle Co. furnizează împreună cu serverul și aplicația destinată dezvoltării unei interfețe sub forma unui ansamblu de pagini *.html* denumit **Oracle Application Express**. De altfel la fel procedează și alte companii care dezvoltă servere pentru baze de date.

În cele ce urmează se va folosi ca server de baze de date *Oracle XE*, soluție gratuită oferită de Oracle Co. iar pentru realizarea interfeței se va face apel la limbajul Java. Aplicațiile astfel construite prezintă trei avantaje majore:

- software-ul necesar este gratuit;
- aplicația astfel realizată poate fi folosită pe diferite platforme: Windows, Linux, Unix, Solaris etc.
- decuplează partea aplicației destinată stocării datelor (server de b.d.) de partea care le accesează (interfața).

Capitolul I

Oracle XE. Operații elementare

Generalități

Oracle Database Express Edition este un server de baze de date relaționale. Într-o bază de date relațională datele sunt păstrate în *fișiere de date*. Un fișier conține *articole* având fiecare aceeași structură, definită la crearea sa.



Structura articolelor unui fișier de date este definită la crearea sa, prin precizarea *câmpurilor* pe care le va conține.

Un *câmp* se caracterizează prin *nume*, *tipul informației* conținute, *lungime* și *numărul de zecimale* (pentru câmpuri numerice).

Datorită faptului că formatul articolului este fix, frecvent se folosește pentru fișier o reprezentare tabelară și chiar se folosește pentru fișierele de date denumirea de "**tabele**".

NUME	PRENUME	ID_ANGAJAT
Avram	Lucian	1
Tanase	Maria	4
Ionescu	Valer	2
Pop	Vasile	3

Un server Oracle XE operează cu o singură bază de date.

Administratorul bazei de date crează conturi ale utilizatorilor cărora le atribuie drepturile necesare operării în interiorul bazei Oracle XE. Un astfel de utilizator devine astfel proprietarul unui subdomeniu inclus în baza Oracle XE denumit de către autorii aplicației "schema" (engl. *schema*). După înregistrarea contului, beneficiarul acestuia poate realiza infrastructura pe care se va baza aplicația sa : tabele, interogări, vederi, proceduri, etc.. Domeniile diferitilor utilizatori sunt în principiu complet separate, un utilizator având însă posibilitatea de a da și altor utilizatori drepturi de acces la componentele domeniului său.

Dimensiunea bazei de date administrate de *Oracle XE* este limitată la 4 GO (4 gigaocteți).

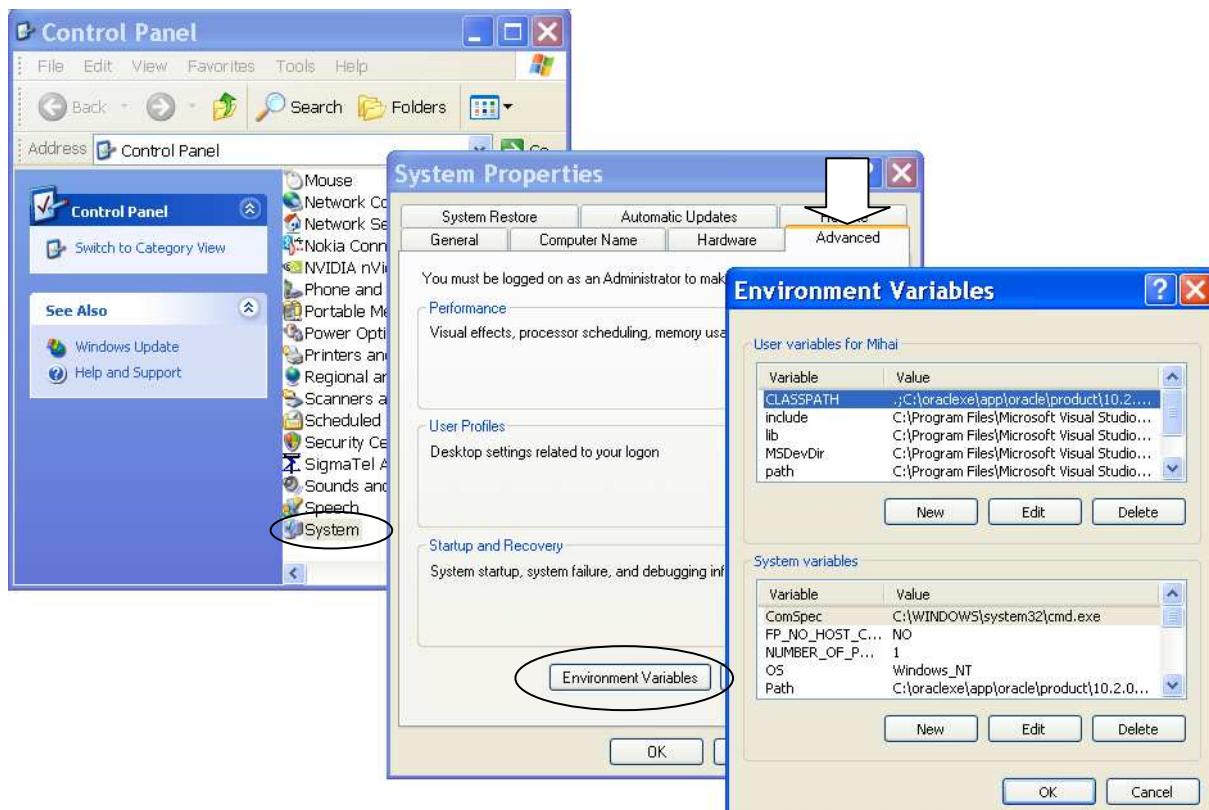
Un utilizator poate accesa un server *Oracle XE* de pe calculatorul pe care acesta este instalat sau prin rețea (rețea locală sau Internet).

Instalarea serverului Oracle XE pe un calculator funcționând sub Windows XP

Pentru instalare demarați o sesiune de lucru folosind un cont de utilizator care are drepturi de administrare și descărcați kitul pentru *Oracle Database XE*, versiunea pentru Windows, de la adresa:

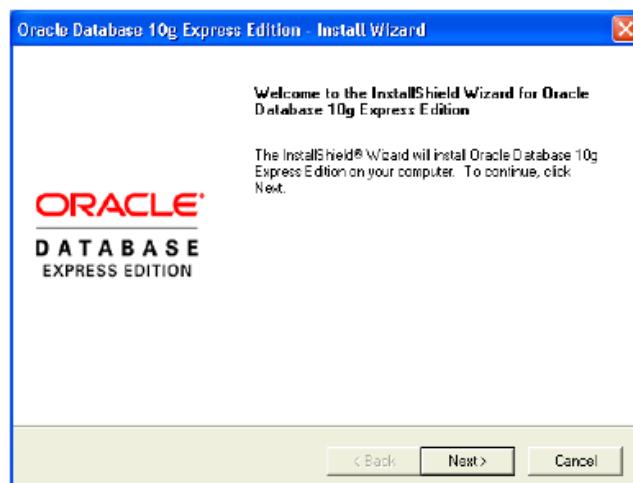
<http://www.oracle.com/technology/products/database/xe>

Dacă pe calculator a mai fost instalat *Oracle XE* sau orice altă versiune, dezinstalați-o în prealabil și ștergeți variabila de mediu **ORACLE_HOME**, dacă aceasta există. Pentru aceasta accesați intrarea *System* în *Control Panel*:

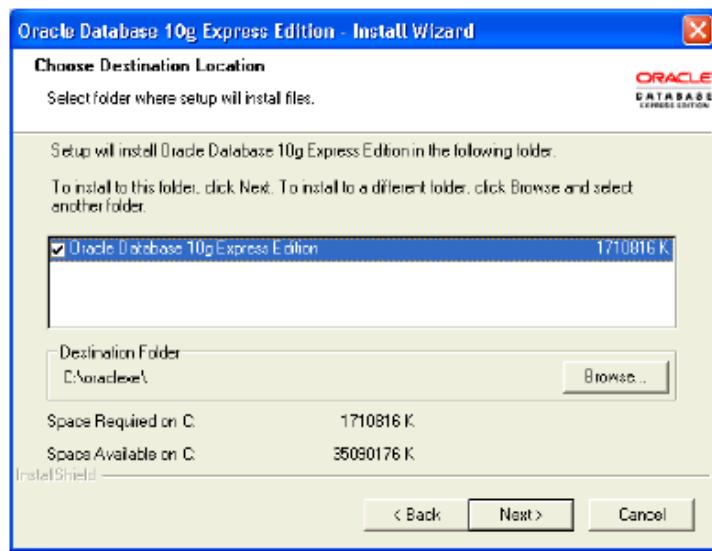


Pentru demararea instalării selectați *OracleXE.exe* (dublu clic).

Instalarea decurge apoi în mai multi pași, astfel:



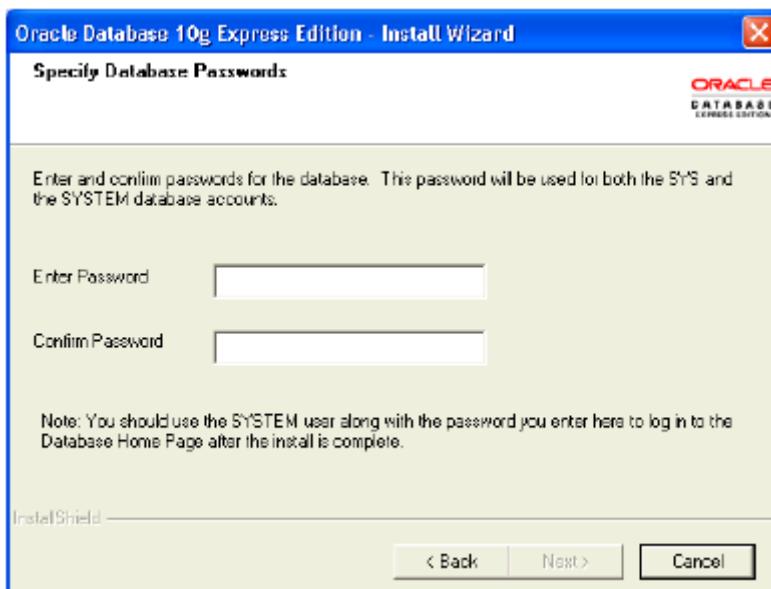
- În fereastra *Licence Agreement* selectați *I accept* și apoi *Next*;
- În fereastra *Choose Destination Location* indicați unde se va instala aplicația Locația implicită este C:.



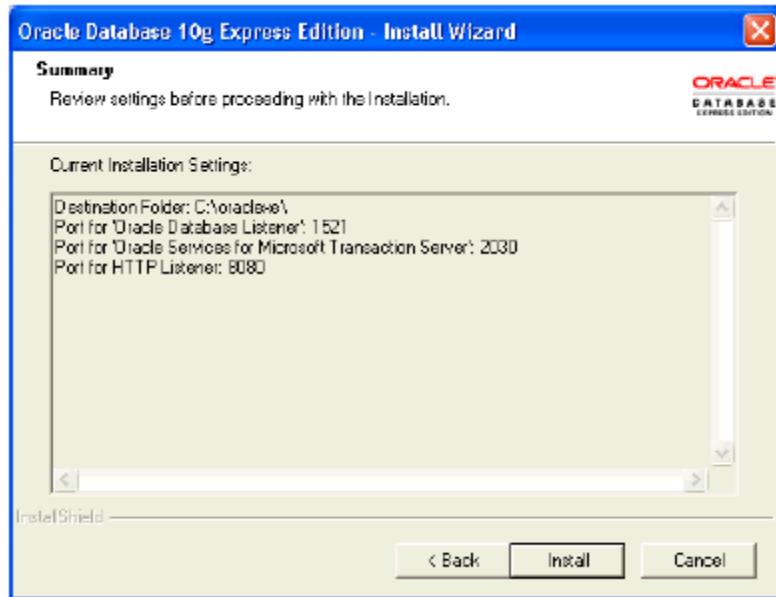
- Dacă sunteți solicitați să introduceți adresele porturilor care vor fi folosite înseamnă că unul dintre porturile隐含的:
 - 1521 alocat implicit serverului Oracle XE și
 - 8080 alocat implicit serverului HTTP prin care se realizează conectarea la serverul *Oracle XE* folosind interfața grafică și protocolul HTTP

este folosit de o altă aplicație. De exemplu portul 8080 ar putea fi folosit de exemplu de serverul *Apache Tomcat*.

- În fereastra *Specify Database Passwords* se introduce parola pentru conturile *SYSTEM* și *SYS*. Acestea vor putea fi folosite de către administratorul serverului pentru a realiza activități specifice: creare/ștergere de utilizatori, salvări de siguranță și.a. Este bine să fie păstrată undeva pentru că fără ea serverul va fi inutilizabil!

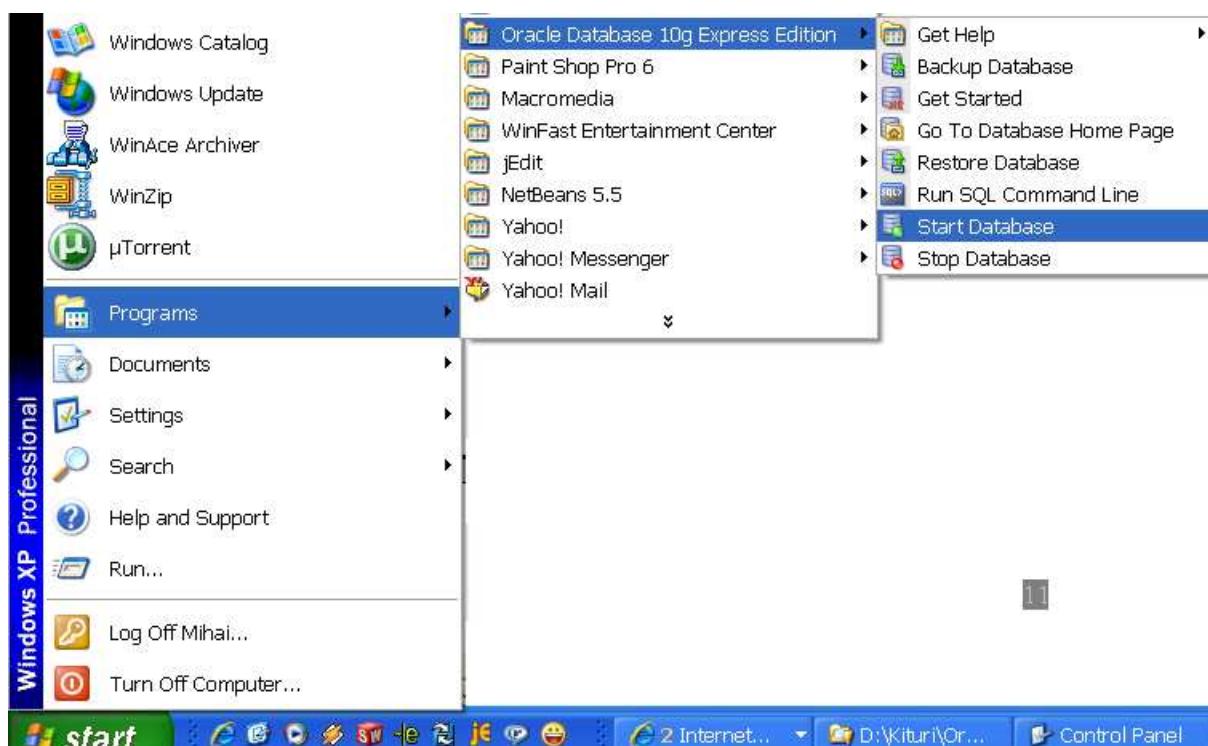


- O ultimă fereastră afișează o recapitulare a opțiunilor introduse. Se apasă *Install* pentru a se realiza instalarea.



Pornirea serverului Oracle XE

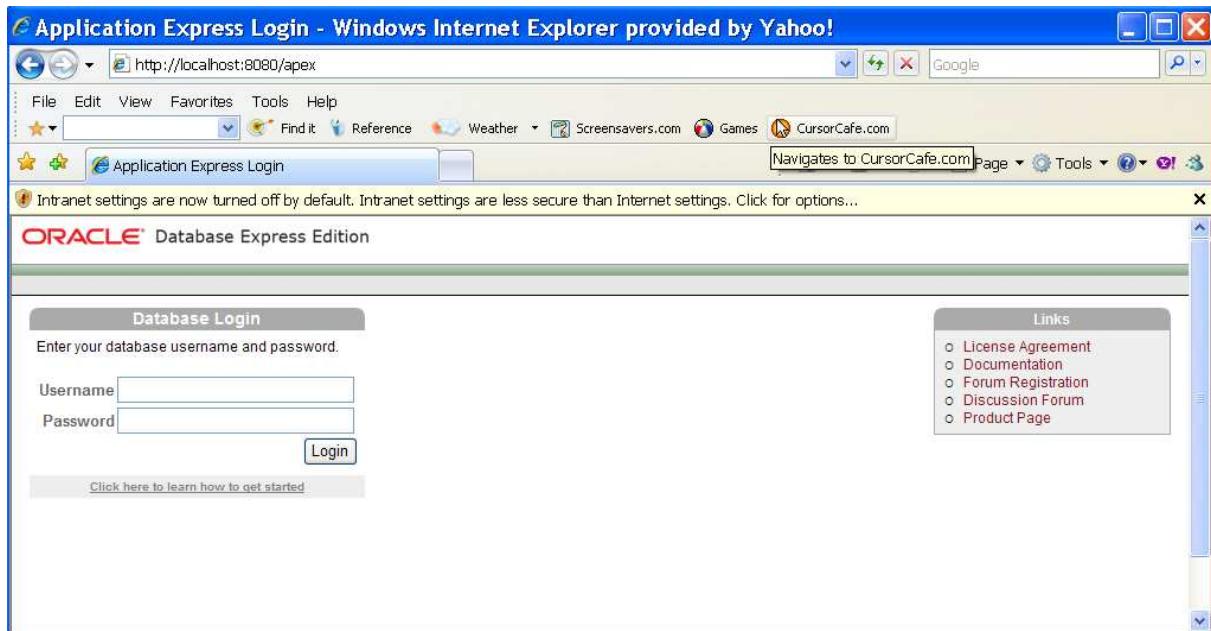
După instalare sau după repornirea calculatorului, serverul *Oracle XE* va fi în mod normal pornit. Dacă pornirea automată a acestuia a fost opriță (*Control Panel ->Administrative Tools->Services*), acesta poate fi pornit manual folosind comanda (*Start -> Programs ->Oracle database 10g Express Edition -> Start Database*).



Tot aceeași cale va fi folosită pentru a afișa fereastra *Aplication Express Login*:

(*Start -> Programs ->Oracle database 10g Express Edition -> Go To Database Home Page*)

Această fereastră va fi punctul de plecare în activitățile de administrare a serverului.



Crearea unui utilizator

În urma instalării serverului *Oracle XE* există trei utilizatori ai serverului, respectiv *SYS* și *SYSTEM* cu drept de administrare și *HR* (parola *hr*), utilizator obișnuit. Utilizatorul *HR* a fost creat pentru a se putea testa aplicațiile incluse în documentația serverului.

Pentru crearea unui nou utilizator se recomandă conectarea folosind contul *SYSYEM*. Contul *SYS* dă acces la fișiere interne ale serverului a căror modificare este interzisă.

The screenshot shows the "Create Database User" dialog and the "User Privileges" section. In the "Create Database User" dialog, the "User: SYSTEM" header indicates the session user. The "Create Database User" button is at the top right. The form fields include:

- * Username: *biblio*
- * Password: *******
- * Confirm Password: *******
- Expire Password:
- Account Status: *Unlocked* (dropdown menu)
- Default Tablespace: *USERS*
- Temporary Tablespace: *TEMP*

In the "User Privileges" section, there are two tabs:

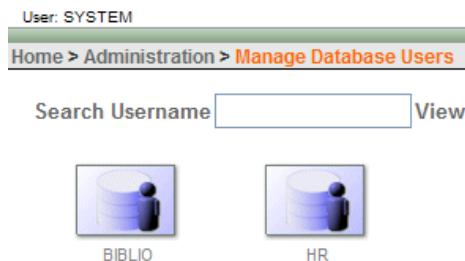
- Roles:** CONNECT RESOURCE DBA
- Direct Grant System Privileges:**

<input checked="" type="checkbox"/> CREATE DATABASE LINK	<input checked="" type="checkbox"/> CREATE MATERIALIZED VIEW	<input checked="" type="checkbox"/> CREATE PROCEDURE
<input checked="" type="checkbox"/> CREATE PUBLIC SYNONYM	<input checked="" type="checkbox"/> CREATE ROLE	<input checked="" type="checkbox"/> CREATE SEQUENCE
<input checked="" type="checkbox"/> CREATE SYNONYM	<input checked="" type="checkbox"/> CREATE TABLE	<input checked="" type="checkbox"/> CREATE TRIGGER
<input checked="" type="checkbox"/> CREATE TYPE	<input checked="" type="checkbox"/> CREATE VIEW	

At the bottom right of the "User Privileges" section are buttons for "Check All" and "Uncheck All".

Un utilizator este identificat prin nume (*Username*:), parolă (*Password*:) și are un ansamblu de drepturi. Pentru exemplele care vor urma, utilizatorul creat se numește *biblio* și are toate drepturile (selectați *Check All*) dar nu are drepturi de administrare a serverului (nu se selectează caseta *DBA - DataBase Administrator*).

După crearea noului utilizator, lista de utilizatori arată ca în figură.



Se observă că numele utilizatorului apare cu majuscule. Similar se va întâmpla și la crearea tabelelor, numele acestora vor apărea scrise cu majuscule.

Observație: După instalarea serverului, utilizatorul *HR* este implicit blocat (*Locked*). Pentru a putea folosi contul *HR* acesta trebuie deblocat. Ca urmare se selectează utilizatorul (operăția necesită conectare ca administrator, *SYSTEM*) și apoi, în fereastra cu proprietăți se impune *Unlocked*.

Crearea unui ansamblu de tabele

Pentru păstrarea informațiilor, modelul relațional presupune folosirea unui ansamblu de tabele, fiecare conținând informații de o anumită natură. Între tabele există legături realizate prin perechi de valori ale unor câmpuri. Exemplu fundamental :

Tabel cu numele angajaților + tabel cu numele copiilor acestora.

ANGAJATI			COPII			
NUME	PRENUME	ID_ANGAJAT	ID_COPIIL	NUME	PRENUME	ID_ANG
Avram	Lucian	1	1	Ionescu	Marian	2
Tanase	Maria	4	2	Ionescu	Lucia	2
Ionescu	Valer	2	3	Pop	Marius	3
Pop	Vasile	3				

În tabelul *ANGAJATI* fie căruia angajat i se înregistrează numele, prenumele și un identificator unic, *ID_angajat*. Identificatorul servește la regăsirea unui angajat, dar și la diferențierea eventualilor angajați care au același nume și același prenume.

În tabelul *COPII* se păstrează numele și prenumele copiilor. Pe lângă cele două câmpuri necesare păstrării acestor informații principale mai sunt două

câmpuri: *ID_copil* care asociază fiecărui copil un număr unic și *ID_ang*. Câmpul *ID_ang* asociază fiecărui copil un articol din tabelul de angajați. Folosind această structură se pot afișa mai multe rapoarte:

- lista copiilor,
- lista copiilor unui angajat,
- lista angajaților care au copii,
- numărul de copii pe care îi are fiecare angajat și probabil și altele.

Cheia primară, cheie străină

În tabele *ANGAJATI* și *COPII* există câte un câmp care are valori distincte pentru toate articolele: *ID_angajat* și *ID_copil*. Când un tabel posedă un câmp care are rolul de a diferenția articolele, câmpul va fi declarat ca fiind **cheie primară** iar serverul de baze de date va verifica unicitatea valorilor adăugate. Concret, la fiecare adăugare a unui nou articol serverul de baze de date va verifica dacă valoarea cheii din noul articol nu apare într-o înregistrare deja prezentă. Dacă aceeași valoare mai apare, adăugarea este refuzată. Un tabel poate avea o singură cheie primară.

În fișierul *COPII* mai există însă un câmp, *ID_ang*. Aceasta primește valori din coloana *ID_angajat* din tabelul corespondent *ANGAJATI*. Primește deci *valori ale cheii primare* dintr-un tabel corespondent. Din acest motiv el poartă numele de **cheie străină**, deoarece valorile sunt ale unei chei primare, dar din alt tabel.

Într-un tabel al unei baze de date unele valori pot lipsi. De exemplu în tabelul pentru angajați am putea avea un câmp destinat memorării adresei de e-mail. Dar nu toți angajații au o adresă de e-mail. La declararea caracteristicilor unui câmp se poate preciza dacă se acceptă valori *nule* (lipsa valorii) sau dimpotrivă. Evident că valorile din câmpurile declarate chei primare sau străine nu pot lipsi.

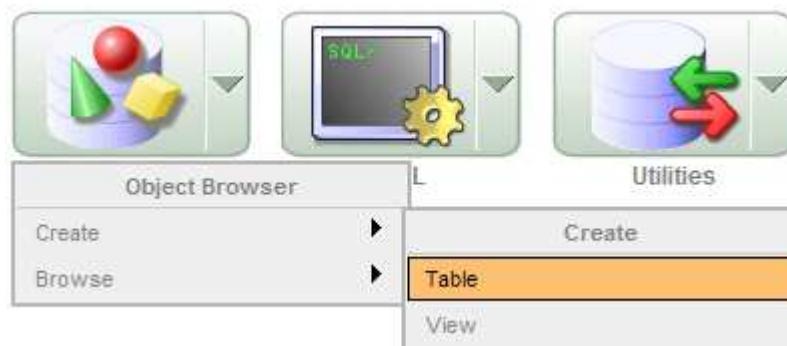
Integritatea referențială

Un server de date are pe lângă rolul de a păstra date și rolul de a menține coerentă acestora. Dacă în exemplul dat angajatul *Ionescu Valer* (*ID_angajat=2*) ar pleca din firmă, deci linia corespunzătoare din fișierul *ANGAJATI* ar fi ştersă, ce se întâmplă cu înregistrările din tabelul *COPII* pentru care *ID_ang=2*? Dacă serverul de baze de date verifică validitatea referințelor

va trebui să suprime în mod automat și liniile corespunzătoare din tabelul *COPII*. În termeni specifici domeniului bazelor de date relaționale spunem că serverul verifică și asigură *integritatea referențială* a bazei de date. Deși verificarea integrității referențiale complică semnificativ realizarea unui server de baze de date, toate aplicațiile majore suportă această caracteristică.

Crearea tabelelor

a. Crearea tabelului ANGAJATI



The screenshot shows the 'Create Table' dialog box. On the left, there is a sidebar with buttons for 'Columns', 'Primary Key', 'Foreign Key', 'Constraints', and 'Confirm'. The main area is titled 'Create Table' and has a 'Cancel' button and a 'Next >' button. The 'Table Name' field is set to 'ANGAJATI' with the 'Preserve Case' checkbox unchecked. The table definition grid has columns for 'Column Name', 'Type', 'Precision', 'Scale', 'Not Null', and 'Move'. The first three columns are populated with 'Nume' (VARCHAR2, 20), 'Prenume' (VARCHAR2, 30), and 'ID_angajat' (NUMBER, 4, 0). The remaining five columns have dropdown menus for 'Select Datatype' and are marked with green up/down arrows for reordering. An 'Add Column' button is located at the bottom right of the grid.

Column Name	Type	Precision	Scale	Not Null	Move
Nume	VARCHAR2	20		<input checked="" type="checkbox"/>	▼▲
Prenume	VARCHAR2	30		<input checked="" type="checkbox"/>	▼▲
ID_angajat	NUMBER	4	0	<input checked="" type="checkbox"/>	▼▲
	- Select Datatype -				▼▲
	- Select Datatype -				▼▲
	- Select Datatype -				▼▲
	- Select Datatype -				▼▲
	- Select Datatype -				▼▲

Primary Key

Table name: ANGAJATI

Primary Key: Not populated
 No Primary Key
 Populated from a new sequence
 Populated from an existing sequence

* Primary Key Constraint Name: ANGAJATI_PK

* Primary Key: ID_ANAJAT(NUMBER)

Composite Primary Key - Select Composite Primary Key -

Create Table

Please confirm your request.

Schema: BIBLIO
Table name: ANGAJATI

SQL

Rezultat:

User: BIBLIO

Home > Object Browser

Tables

ANGAJATI

Column Name	Data Type	Nullable	Default	Primary Key
ID_ANAJAT	NUMBER(5,0)	No	-	1
NUME	VARCHAR2(4000)	No	-	-
PRENUME	VARCHAR2(4000)	No	-	-

1 - 3

Create Table

* Table Name: COPII

Column Name Type Precision Scale Not Null Move

ID_COPII	NUMBER	4	0	<input checked="" type="checkbox"/>	▼▲
NUME	VARCHAR2		20	<input checked="" type="checkbox"/>	▼▲
PRENUME	VARCHAR2		30	<input checked="" type="checkbox"/>	▼▲
ID_ANG	NUMBER	4	0	<input checked="" type="checkbox"/>	▼▲
	- Select Datatype -			<input checked="" type="checkbox"/>	▼▲
	- Select Datatype -			<input checked="" type="checkbox"/>	▼▲
	- Select Datatype -			<input checked="" type="checkbox"/>	▼▲

Add Column

Columns
Primary Key
Foreign Key
Constraints
Confirm

Primary Key
Table name: COPII
Primary Key: No Primary Key
 Populated from a new sequence
 Populated from an existing sequence
 Not populated

* Primary Key Constraint Name: COPII_PK
* Primary Key: ID_COPIL(NUMBER)
Composite Primary Key: - Select Composite Primary Key -

Columns
Primary Key
Foreign Key
Constraints
Confirm

Foreign Keys
Foreign Key Columns Referenced Table Referenced Columns Action

Add Foreign Key Add

* Name: COPII_fk
 Disallow Delete
 Cascade Delete
 Set Null on Delete

Select Key Column(s): ID_COPIL, NUME, PRENUME
* Key Column(s): ID_ANG

* References Table: ANGAJATI

Select Reference Column(s): NUME, PRENUME
* Referenced Column(s): ID_ANAJAT

Comportamentul serverului Oracle XE în momentul ștergerii unui articol din tabelul corespondent poate fi impus prin selectarea uneia dintre opțiunile următoare:

- **Disallow Delete** - împiedică ștergerea articolelor din tabelul corespondent care contin referințe spre articolele din tabelul curent;
- **Cascade Delete** - impune ștergerea ștergerea articolelor din tabelul curent la ștergerea articolului corespondent din tabelul referit ;

- **Set to Null on Delete** impune păstrarea articolelor din fișierul curent la ștergerea articolului corespondent din tabelul referit. Valorile cheii străine în aceste articole va fi *null* (va lipsi!).

The screenshot shows the Oracle SQL Developer Object Browser interface. On the left, there is a sidebar with buttons for 'Columns', 'Primary Key', 'Foreign Key', 'Constraints', and 'Confirm'. Below this is a 'SQL' button. The main area has a title bar 'Create Table' with 'Cancel' and 'Create' buttons. It displays a confirmation message: 'Please confirm your request.' Below it, it shows 'Schema: BIBLIO' and 'Table name: COPII'. At the bottom of the main window, there is a status bar with 'User: BIBLIO'.

Object Browser:

- Tables dropdown: Tables
- Search bar: ANGAJATI
- Table list: ANGAJATI, COPII

Table Definition:

Column Name	Data Type	Nullable	Default	Primary Key
ID_COPIL	NUMBER(5,0)	No	-	1
NUME	VARCHAR2(4000)	No	-	-
PRENUME	VARCHAR2(4000)	No	-	-
ID_ANG	NUMBER(5,0)	No	-	-

Bottom right corner of the table definition grid: 1 - 4

Adăugarea datelor în tabelele

The screenshot shows the Oracle SQL Developer Object Browser interface. On the left, there is a sidebar with buttons for 'Tables', 'Data', 'Indexes', 'Model', 'Constraints', 'Grants', 'Statistics', and 'UI Defaults'. Below this is a 'Table' button. The main area has a title bar 'Home > Object Browser'.

Object Browser:

- Tables dropdown: Tables
- Search bar: ANGAJATI
- Table list: ANGAJATI

Action Bar:

- Table
- Data (highlighted)
- Indexes
- Model
- Constraints
- Grants
- Statistics
- UI Defaults

Create Row Dialog:

- Buttons: Create Row, Cancel, Create, Create and Create Another
- Table: ANGAJATI
- Fields:
 - * Nume: Ionescu
 - * Prenume: Valer
 - * Id Angajat: 2

EDIT	NUME	PRENUME	ID_ANGAJAT
	Tanase	Maria	4
	Ionescu	Valer	2
	Pop	Vasile	3

row(s) 1 - 3 of 3

[Download](#)

Articolele adăugate pot fi editate dacă se selectează butonul *EDIT*.

Suprimarea unui articol

Table:	ANGAJATI
Numé *	Ionescu
Prenume *	Valer
Id Angajat *	2

[Cancel](#) [Delete](#)

Windows Internet Explorer

Confirm delete.

[OK](#) [Cancel](#)

Ca urmare a ștergerii angajatului având $ID_angajat = 2$, din tabelul corespondent *COPII* s-au șters liniile pentru care cheia străină *ID_ang* avea valoarea 2.

EDIT	ID_COPIL	NUME	PRENUME	ID_ANG
	3	Pop	Marius	3

row(s) 1 - 1 of 1

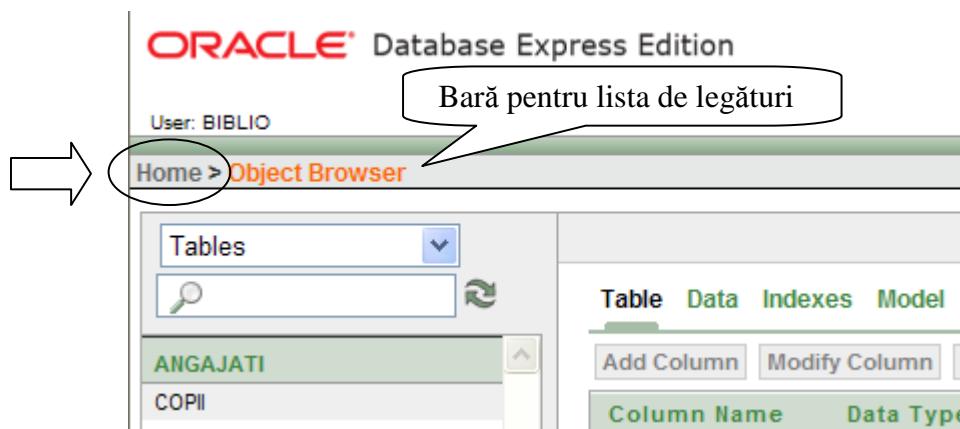
[Download](#)

Navigarea în OracleXE

Fereastra afişată după conectare conține un ansamblu de butoane, fiecare având în dreapta o săgeată care permite afişarea unui meniu derulant. Meniul derulant afişează o serie de instrumente accesibile utilizatorului în momentul respectiv. Acestea sunt afişate tot ca pagini web.



Revenirea la o pagină web anterioară se poate realiza prin selectarea acesteia din lista de legături afişată pe bara dispusă în partea de sus a paginii.



În exemplul din figură se poate reveni la pagina principală selectând legătura *Home*.

Salvarea/restaurarea conținutului baxei de date XE

Salvarea integrală a bazei de date XE se realizează periodic de către administratorul serverului folosind o procedură care va fi prezentată la sfârșitul cursului, în capitolul destinat administrării serverului.

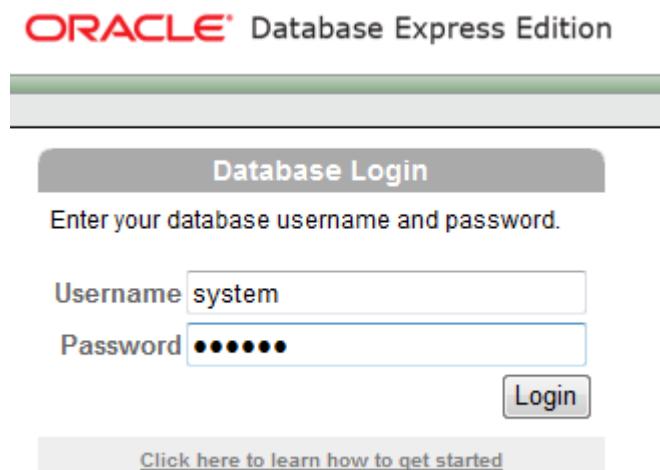
În continuare vor fi expuse soluții de salvare parțială, respectiv salvarea unei scheme sau a unui număr de tabele.

1. Exportul / importul unei scheme

Exportul schemei sub forma unui fișier

Se crează un director care va păstra fișierul conținând schema salvată. Precupunând că acest director este d:\tmp, după crearea directorului se va continua astfel:

Se realizează o conexiune ca SYSTEM:



Se dau următoarele două comenzi SQL :

```
CREATE OR REPLACE DIRECTORY dmpdir AS 'd:\tmp';
GRANT READ,WRITE ON DIRECTORY dmpdir TO hr;
```

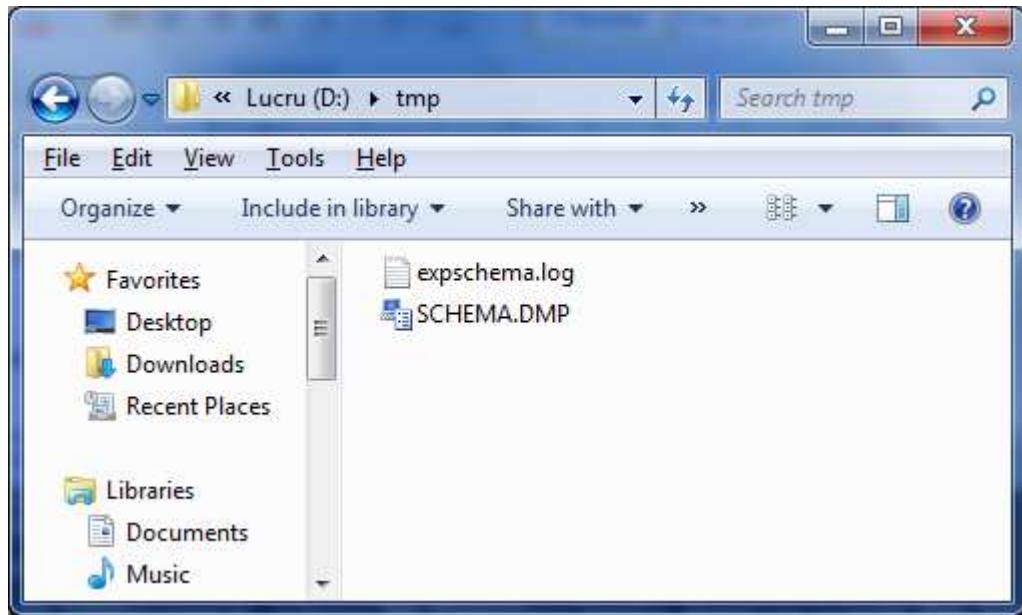
Într-o fereastră *Command Prompt* se introduce comanda de export a schemei (pe un rând!) :

```
C:\>expdp system/oracle schemas=hr directory=dmpdir dumpfile=schema.dmp logfile=expschema.log
```

The screenshot shows a Microsoft Windows Command Prompt window titled "C:\ Command Prompt". The window displays the following text:
 Microsoft Windows [Version 6.1.7601]
 Copyright © 2009 Microsoft Corporation. All rights reserved.
 C:\>expdp system/oracle schemas=hr directory=dmpdir dumpfile=schema.dmp logfile=expschema.log

```
expdp SYSTEM/password SCHEMAS=hr DIRECTORY=dmpdir DUMPFILE=schema.dmp
LOGFILE=expschema.log
```

Rezultat:



Restaurarea schemei :

Importul schemei pe un alt server Oracle

Într-o fereastră *Command Prompt* se introduce comanda *impdp* de mai jos:

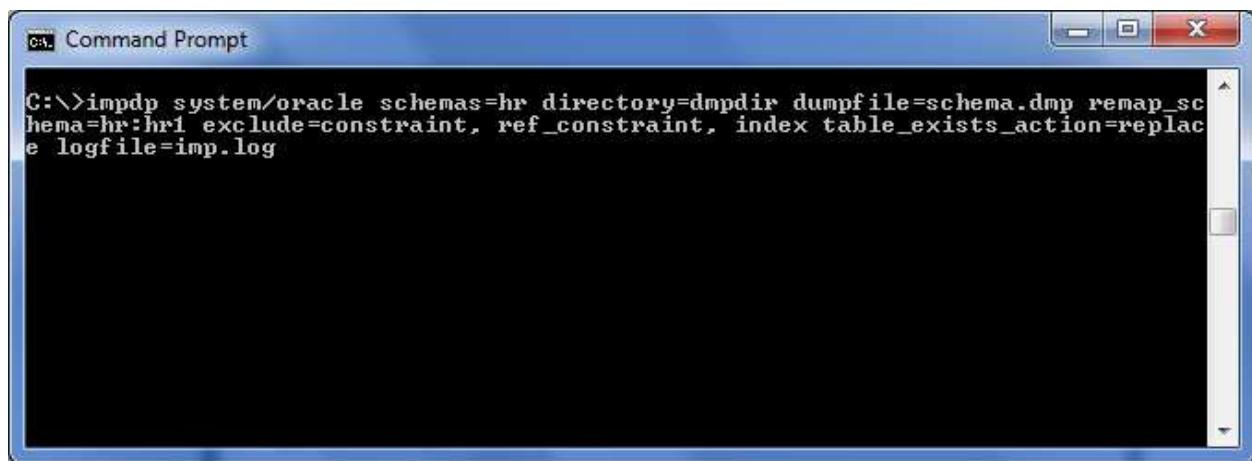
```
C:\>impdp system/oracle schemas=hr directory=dmpdir dumpfile=schema.dmp exclude=
constraint, ref_constraint, index table_exists_action=replace logfile=imp.log_
```

```
impdp      SYSTEM/password      SCHEMAS=hr      DIRECTORY=dmpdir
DUMPFILE=schema.dmp      EXCLUDE=constraint,      ref_constraint,      index
TABLE_EXISTS_ACTION=replace LOGFILE=impschema.log
```

Importul aceleiasi scheme sub alt nume (copie de rezervă)

Realizarea schemei *hr1*, o copie de siguranță a schemei *hr*.

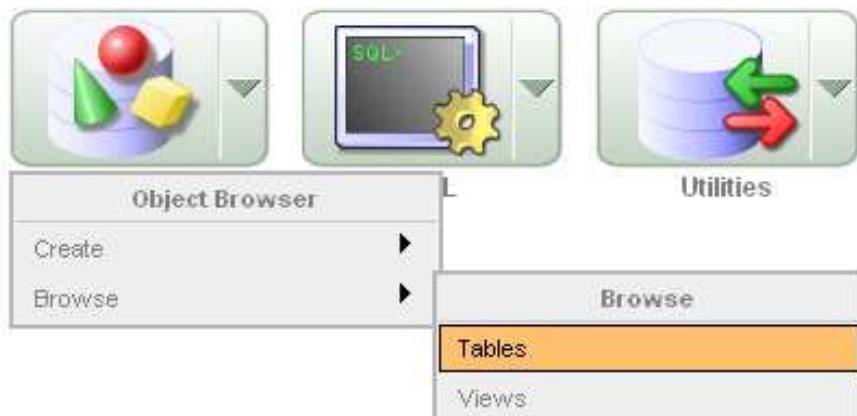
```
impdp SYSTEM/password SCHEMAS=hr DIRECTORY=dmpdir
DUMPFILE=schema.dmp REMAP_SCHEMA=hr:hr1 EXCLUDE=constraint,
ref_constraint, index TABLE_EXISTS_ACTION=replace LOGFILE=impschema.log
```



Deoarece pentru realizarea aplicațiilor pe calculatorul propriu este necesară preluarea structurii de tabele create la laborator, în cele ce urmează va fi prezentată procedura de salvare a tabelelor.

Salvarea tabelelor

Refacerea unora dintre tabelele unei scheme se va realiza folosind un ansamblu de comenzi SQL salvate într-un fișier ASCII. Pentru crearea fișierului cu comenzi se selectează în fereastra *Home* opțiunea *Object Browser / Tables*:



Apoi se selectează succesiv tabelele ale căror comenzi de creare trebuie salvate și pentru fiecare tabel selectat se afișează comanda de creare corespunzătoare selectând legătura **SQL**.

The screenshot shows the Oracle SQL Developer interface. On the left, there's a tree view of tables: AUTCARTI, AUTORI, CARTI, CITITORI (which is selected and highlighted in green), and EDITURI. The main panel displays the 'CITITORI' table structure. The 'SQL' tab at the top of the main panel is circled in red. The table definition includes columns COD_CIT, CNP, NUME, PRENUME, LOCALITATEA, ID_JUD, ADRESA, TELEFON, and E_MAIL, with COD_CIT as the primary key.

Aplicația va afișa comanda *CREATE* corespunzătoare iar dacă tabelul selectat este indexat, după fraza *CREATE* sunt afișate frazele care realizează refacerea indecșilor.

CITITORI					
Table	Data	Indexes	Model	Constraints	Grants
Statistics	UI Defaults	Triggers	Dependencies	SQL	
<pre> CREATE TABLE "CITITORI" ("COD_CIT" NUMBER(6,0) NOT NULL ENABLE, "CNP" VARCHAR2(15), "NUME" VARCHAR2(30) NOT NULL ENABLE, "PRENUME" VARCHAR2(50), "LOCALITATEA" VARCHAR2(20), "ID_JUD" NUMBER(2,0), "ADRESA" VARCHAR2(100), "TELEFON" NUMBER(10,0), "E_MAIL" VARCHAR2(20), CONSTRAINT "CITITORI_PK" PRIMARY KEY ("COD_CIT") ENABLE) / CREATE INDEX "CITITORI_IDX1" ON "CITITORI" ("CNP") / CREATE INDEX "CITITORI_IDX2" ON "CITITORI" ("NUME") / </pre>					

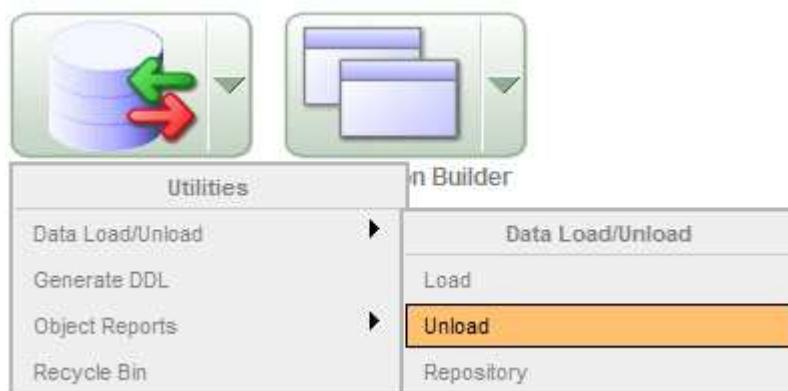
Frazele SQL afișate vor fi selectate cu mouse-ul și copiate în *Clipboard (copy)* și apoi inserate într-un fișier ASCII (.txt). Pentru aceasta se poate deschide aplicația *Notepad*.

Procedeul se va repeta pentru fiecare tabel în parte, comenziile SQL corespunzătoare fiind inserate una după alta, în același fișier .txt, care apoi va fi salvat în directorul propriu.

Salvarea datelor

Pentru a putea reface ulterior conținutul tabelelor, Oracle XE permite copierea acestora în fișiere .txt. Deoarece un astfel de fișier poate conține date provenind dintr-un singur tabel, da rezulta un număr de fișiere egal cu numărul de tabele salvate.

Pentru a iniția copierea datelor unui tabel se selectează în fereastra *Home* opțiunea *Utilities / Data Load/Unload / Unload*:



In fereastra care se afișează se selectează *Unload to Text*:



Se indică schema (implicit schema curenta)

<input type="button" value="Schema"/> <input type="button" value="Table Name"/> <input type="button" value="Columns"/> <input type="button" value="Options"/>	<p>Unload to Text</p> <p>* Schema BIBLIO</p>	<input type="button" value="Cancel"/> <input type="button" value="Next >"/>
--	--	--

c. Se indică tabelul (*CITITORI*)

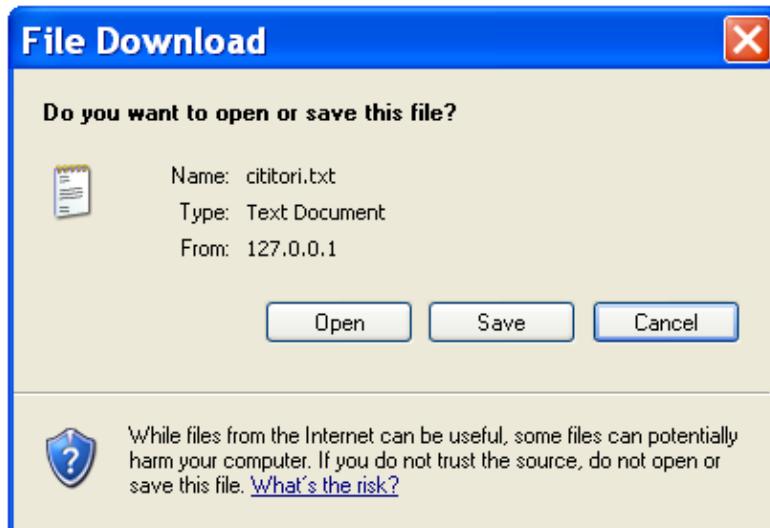
<input type="button" value="Schema"/> <input type="button" value="Table Name"/> <input type="button" value="Columns"/> <input type="button" value="Options"/>	<p>Unload to Text</p> <p>* Table CITITORI</p>	<input type="button" value="Cancel"/> <input type="button" value="< Previous"/> <input type="button" value="Next >"/>
<input type="button" value="Schema"/> <input type="button" value="Table Name"/> <input type="button" value="Columns"/> <input type="button" value="Options"/>	<p>Unload to Text</p> <p>* Columns</p> <ul style="list-style-type: none"> COD_CIT CNP NUME PRENUME LOCALITATEA ID_JUD ADRESA TELEFON E_MAIL 	<input type="button" value="Cancel"/> <input type="button" value="< Previous"/> <input type="button" value="Next >"/>

e. Se indică separatorul (în mod normal *tab*, *\t*) și se validează *Include Column Names*.

<input type="button" value="Schema"/> <input type="button" value="Table Name"/> <input type="button" value="Columns"/> <input type="button" value="Options"/>	<p>Unload to Text</p> <p>Separator \t Optionally Enclosed By <input checked="" type="checkbox"/> Include Column Names</p> <p>File Format DOS</p> <p>File Character Set Unicode UTF-8</p>	<input type="button" value="Cancel"/> <input type="button" value="< Previous"/> <input type="button" value="Unload Data"/>
--	--	---

Se apasă apoi *Unload Data*. În fereastra care se afișează se selectează *Save* și apoi se indică locul pe disc și numele fișierului. Numele propus de

aplicație va fi *citorii.txt*. Este bine ca numele fișierului în care sunt memorate datele să coincidă cu numele tabelului din care provin.



Refacerea tabelelor bazei de date

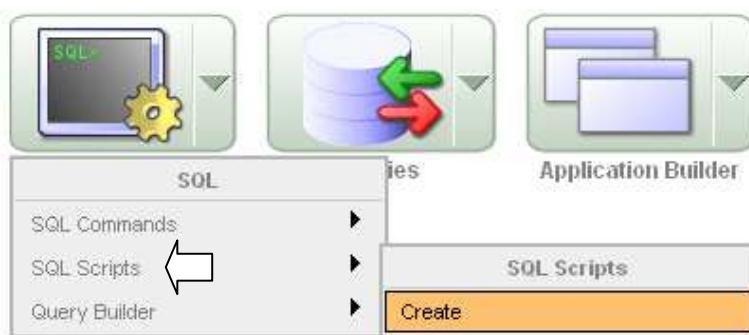
Refacerea tabelelor bazei de date se realizează în două etape:

1. Se refac tabela și
2. Se inserează datele.

Refacerea tabelelor se poate realiza folosind comenziile *CREATE* salvate anterior într-un fișier .txt. Comenziile pot fi introduse una câte una copiindu-le în fereastra destinată introducerii comenziilor SQL sau toate odată, folosind posibilitatea aplicației de creare și execuție a scripturilor SQL.

A doua variantă fiind mai productivă va fi prezentată în continuare.

Se afișează fereastra în care pot fi introduse comenziile unui script (*Home / SQL / SQL Scripts / Create*).



Se deschide în Notepad fișierul care conține comenziile SQL de creare și se copiază conținutul său în fereastra Script Editor, ca mai jos. Se dă de asemenea un nume scriptului deoarece el va fi salvat în baza de date.

Script Name **CreareTabele****Cancel****Download****Save****Run****Undo** **Redo** **Find**

```

1 CREATE TABLE "CITITORI"
2   ( "COD_CIT" NUMBER(6, 0) NOT NULL ENABLE,
3     "CNP" VARCHAR2(15),
4     "NUME" VARCHAR2(30) NOT NULL ENABLE,
5     "PRENUME" VARCHAR2(50),
6     "LOCALITATEA" VARCHAR2(20),
7     "ID_JUD" NUMBER(2, 0),

```

In continuare se execută scriptul (*Run*).**Run Script****Cancel****Run**

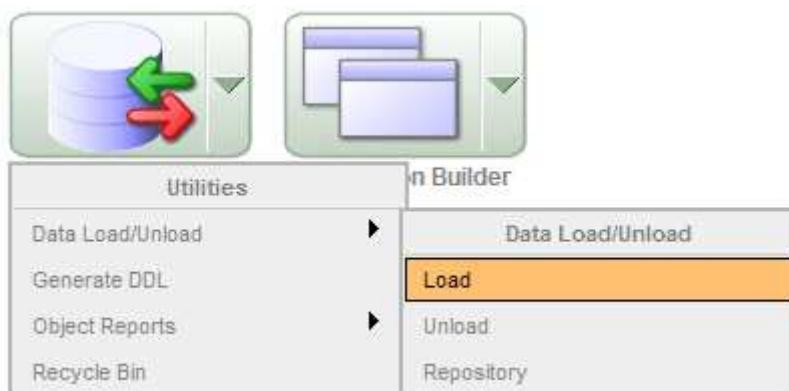
You have requested to run the following script. Please confirm your request.

Script Name	CreareTabele
Created	on 09/15/2007 05:27:45 PM by BIBLIO
Updated	on 09/15/2007 05:27:54 PM by BIBLIO
Number of Statements	4
Script Size in Bytes	889

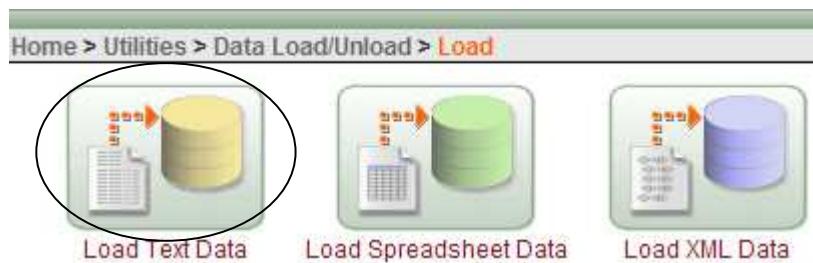
Inserarea datelor în tabele

După refacerea tabelelor bazei de date se poate trece la refacerea conținutului acestora folosind fișierele .txt care conțin datele. Pentru aceasta se poate proceda astfel:

- a. Se selectează în fereastra *Home* opțiunea *Utilities / Data Load/Unload / Load*:



b. În fereastra care se afișează se selectează *Load Text Data*:



c. Se indică faptul că tabelul în care vor fi adăugate datele există (se selectează opțiunea *Existing table*):

Load To:
 Existing table
 New table

Load From:
 Upload file (comma separated or tab delimited)
 Copy and paste (up to 30KB)

c. Se indică schema (Biblio):

* Schema BIBLIO

d. Se selectează tabelul care urmează să fie repopulat cu date:

* Table Name - Select Table -
 - Select Table -
 AUTCARTI (table)
 AUTORI (table)
 CARTI (table)
CITITORI (table)
 EDITURI (table)

e. Se indică numele fișierului (*Browse*) și separatorul (*tab*, *\t*);

e. Se indică numele fișierului și separatorul folosit:

The screenshot shows the 'Load Data' dialog in Oracle SQL Developer. On the left, there's a vertical stack of tabs: Schema, Table Name, File Details (which is currently selected), and Column Mapping. The main area is titled 'Load Data' and contains the following fields:

- * File: D:\Mihaile\Postuniversitar\Cursuri\Bda (with a 'Browse...' button)
- * Separator: \t
- Optionally Enclosed By: (empty field)
- First row contains column names.
- File Character Set: Unicode UTF-8 (with a dropdown arrow)

f. Aplicația afișează în continuare informațiile recuperate. Pentru încărcarea acestora în tabelul indicat se va apăsa butonul *Load data*.

The screenshot shows the 'Load Data' dialog with the 'Schema' and 'Table Name' sections visible. Below them is the 'Column Mapping' section, which contains a table for defining how columns from the source file map to the target table columns:

Column Names	COD_CIT - number *	CNP - varchar2(15)	NUME - varchar2(30)
Format			
Upload	Yes	Yes	Yes
Row 1	3	1234567890123	Ionescu
Row 2	4	0987654321123	Pop

Observație. La refacerea tabelelor și a conținuturilor acestora folosind metoda expusă anterior, serverul *Oracle XE* face toate verificările privind integritatea datelor. Implicațiile acestui fapt sunt următoarele:

1. Salvarea comenziilor de creare a tabelelor trebuie să urmeze ordinea creerii inițiale a acestora. Astfel un tabel care conține chei străine va fi creat după crearea tabelelor de care acesta depinde.
2. Restaurarea datelor trebuie să urmeze aceeași logică. Datele dintr-un tabel care conține o cheie străină vor fi încărcate după încărcarea datelor din tabelul principal, ale cărui înregistrări sunt referite prin valorile cheii străine.

Astfel de exemplu tabelul *Edituri* trebuie creat înaintea tabelului *Carti*. Datele din tabelul *Carti* nu pot fi încărcate înaintea încărcării datelor din tabelul *Edituri*.

Capitolul II

Structurarea bazelor de date

Scurt istoric

În anii '60, A.F. Codd, cercetător la I.B.M., a studiat posibilitatea ameliorării modului de stocare a datelor în fișiere deoarece existența informațiilor redundante predispunea la erori greu de depistat. În 1970 a publicat un articol, **"A Relational Model of Data for Large Shared Databanks"** care a schimbat complet concepțiile privind structura bazelor de date. Un programator, Larry Ellison, sesizând importanța practică a teoriei lui Codd, a realizat un produs software, *Oracle*, și o firmă pentru promovarea acestuia, *Oracle Corporation*. L. Ellison a devenit astfel unul dintre cei mai bogăți oameni de pe planetă.

Anomalii rezolvate de modelul relațional

Să presupunem că într-o bibliotecă, bibliotecara păstrează evidența cărților cu ajutorul calculatorului, într-un fișier pentru cărți având următoarea structură:

NrInv	Autor	Titlu	Editura	Locul	Anul apariției

Având în vedere posibilitățile unui calculator legate de parcurgerea fișierelor, s-ar părea că vor putea fi realizate ușor o mulțime de rapoarte. În timpul exploatarii pe calculator a fișierului s-au depistat însă probleme cauzate de *modul de structurare a informațiilor*.

1. Anomalia la actualizarea datelor:

Conducerea bibliotecii a impus ca numele autorului să fie scris cu majuscule. Pentru autorii mai puțin prolifici nu e nici o problemă, dar unii se regăsesc în fișier de zeci sau chiar sute de ori. Modificarea în sute de locuri a aceluiași nume poate conduce la erori greu de depistat. Această anomalie poartă numele de **anomalia la actualizare**. Pentru eliminarea ei, modelul relațional propune structurarea informațiilor în două tabele: un tabel cu numele autorilor și un tabel cu cărți:

CodAutor	Autor

NrInv	CodAutor	Titlu	Editura	Locul	Anul apariției

În momentul înregistrării unei cărți, dacă autorul există deja în tabelul de autori, i se notează codul care apoi este folosit la înregistrarea cărții. Dacă autorul nu există în fișierul de autori, el va fi adăugat, în momentul adăugării atribuindu-se un cod. În acest mod, modificarea numelui autorilor se realizează simplu, numele fiecărui autor apărând o singură dată.

2. Anomalia la ștergerea datelor:

Această anomalie poate apărea dacă se cere ștergerea din fișierul de cărți a înregistrării corespunzând unei cărți pierdute. Odată cu titlul cărții se șterge și editura care a publicat-o. Ulterior, dacă se dorește realizarea unui raport privind editurile cu care biblioteca are relații, în raport nu va mai figura editura care a realizat cartea suprimată. Această anomalie poartă numele de *anomalie la ștergerea datelor*.

3. Anomalia la adăugarea datelor:

Dacă se dorește înregistrarea în baza de date a bibliotecii a datelor unei noi edituri, în modelul elaborat acest lucru nu este posibil fără adăugarea unei prime cărți achiziționate de la aceasta. Dacă nu s-a cumpărat nici o carte, baza noastră prezintă o *anomalie la adăugarea datelor*.

Normalizarea

Modelul relațional elaborat de Codd propune soluții pentru eliminarea acestor anomalii. Procesul de structurare a bazei de date în vederea eliminării anomaliei sesizate poartă numele de **normalizare**. Normalizarea constă în aducerea bazei de date într-una dintre formele normale, cele mai importante fiind cele 3 forme prezentate în continuare.

1. Prima formă normală

Prima formă normală cere ca tabelele în care sunt păstrate informațiile să satisfacă următoarelor cerințe:

- Fiecare coloană trebuie să păstreze o informație elementară (care nu se mai poate descompune). În exemplul prezentat, dacă o carte are mai mulți autori coloana *CodAutor* ar trebui să conțină mai multe coduri, deci acest mod de structurare nu respectă această cerință.
- Fiecare coloană trebuie să aibă un nume unic;
- Tabelul nu poate avea două linii conținând informații identice. Fiecare tabel din compoziția unei baze de date normalizate conține o *cheie primară*. **O cheie primară este un câmp care are valori distincte pentru toate liniile tabelului.** Uneori, mai rar, cheia primară este obținută prin alăturarea valorilor dintr-un ansamblu de mai multe câmpuri. Pot exista în baza de date tabele care nu au o cheie primară. Tabelele pentru care s-a definit o cheie primară respectă, de regulă, cerința enunțată.

d. Într-un tabel nu se admit grupuri de informații care se repetă. În exemplul dat *Editura și Locul* formează un grup care probabil se repetă pentru toate cărțile provenind de la aceeași editură.

2. A doua formă normală

A doua formă normală se referă la tabelele care au o cheie compusă din valorile mai multor câmpuri. Astfel, în exemplul dat dacă datele editurii sunt păstrate într-un tabel separat având structura:

NumeEd	Oras	Adresa	NumeScurt	Telefon

atunci adăugarea unui nou sediu pentru o editură va face ca valoarea din coloana "NumeScurt" să apară repetat, deoarece ea nu depinde de sediul editurii ci de numele acesteia. Pentru a satisface cerințele celei de-a doua forme normale, coloanele unui tabel trebuie să depindă direct de toate câmpurile care formează o cheie primară compusă.

3. A treia formă normală

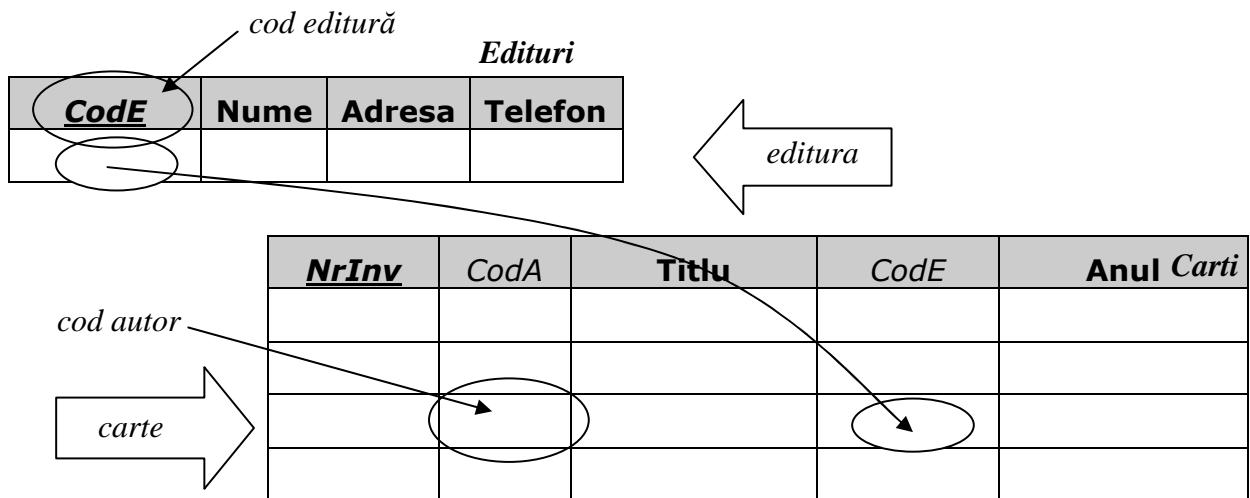
A treia formă normală rezolvă anomalia care apare în cazul în care într-un tabel există una sau mai multe coloane ale căror valori nu depind direct de valoarea cheii primare. În exemplul dat, să presupunem că în tabelul în care sunt înregistrate cărțile există o coloană care conține numele furnizorului (firma prin care s-a achiziționat cartea). Cu siguranță că prin același furnizor s-au achiziționat și alte cărți, deci numele acestuia va apărea pe mai multe linii deoarece furnizorul nu depinde direct de carte. Coloanele unui tabel care satisface a treia formă normală conțin informații care depind direct de cheia primară.

Relații între tabelele unei baze de date

Pentru a satisface cerințele impuse de cele 3 forme normale prezentate, informațiile sunt de regulă păstrate într-un ansamblu de tabele între care există relații de diferite tipuri.

1. Relații 1 la mai mulți (1 la n, one to many)

Dacă în exemplul considerat datele privind o editură sunt păstrate într-un fișier iar cărțile sunt înregistrate în alt fișier, între cele două tabele se stabilește o dependență de tip "unul la mai mulți".



Acest tip de relație este cel mai frecvent întâlnit și stă la baza modelului relational elaborat de Codd.

Cheile primare din cele două tabele sunt **CodE** pentru **Edituri** și **NrInv** pentru **Cărți**. În tabelul de cărți, **CodE** și **CodA** (cod autor) sunt *chei străine*. O cheie străină dintr-un tabel A permite regăsirea unei linii dintr-un tabel asociat, B. În tabelul asociat, B, cheia străină din tabelul A este de regulă cheie primară. În cazul dat, cheia străină **CodA** permite regăsirea în tabelul de autori a numelui acestuia iar cheia străină **CodE** permite găsirea numelui editurii care a publicat cartea.

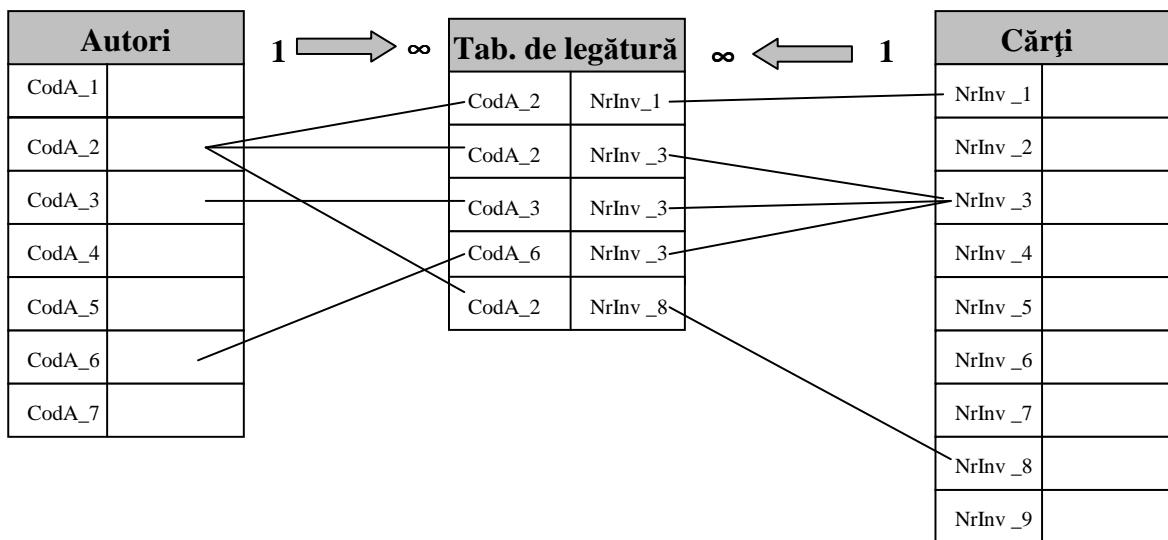
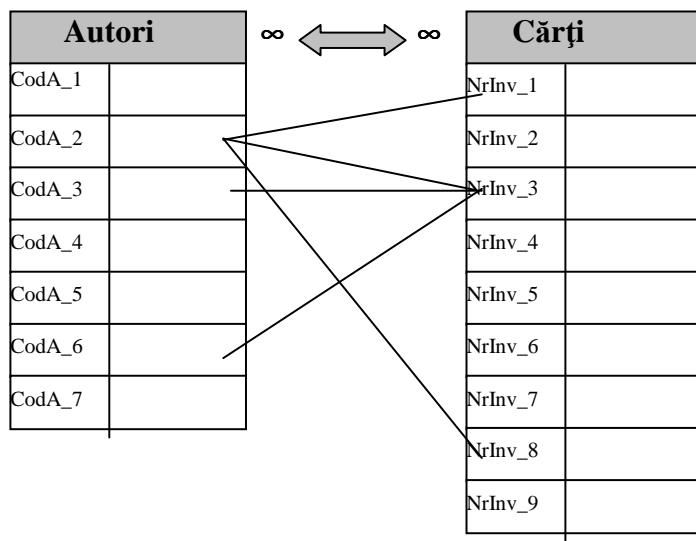
2. Relații 1 la 1 (one to one)

O relație de tipul 1 la 1 apare în cazul în care unei linii dintr-un tabel îi corespunde o singură linie în tabelul cu care acesta este în legătură. În cazul în care fiecare înregistrare dintr-un tabel îi corespunde o înregistrare în al doilea tabel nu este necesară înregistrarea informațiilor în două tabele separate.

3. Relații mai mulți la mai mulți (m la n, many to many)

O relație de acest tip apare în exemplul dat între tabela de autori și cea de cărți. Astfel un scriitor poate fi autor la mai multe cărți iar o carte poate avea mai mulți autori (**CodA_2** respectiv **NrInv_3** de exemplu).

În astfel de situații se va proceda la crearea unui tabel suplimentar, de legătură, care va transforma relația evidențiată (*many to many*) în relații *one to many*. Fiecare articol din tabelul de legătură va contine o pereche de *chei străine*, una fiind cheie primară în primul tabel și una în al doilea tabel:



Capitolul III

Limbajul SQL

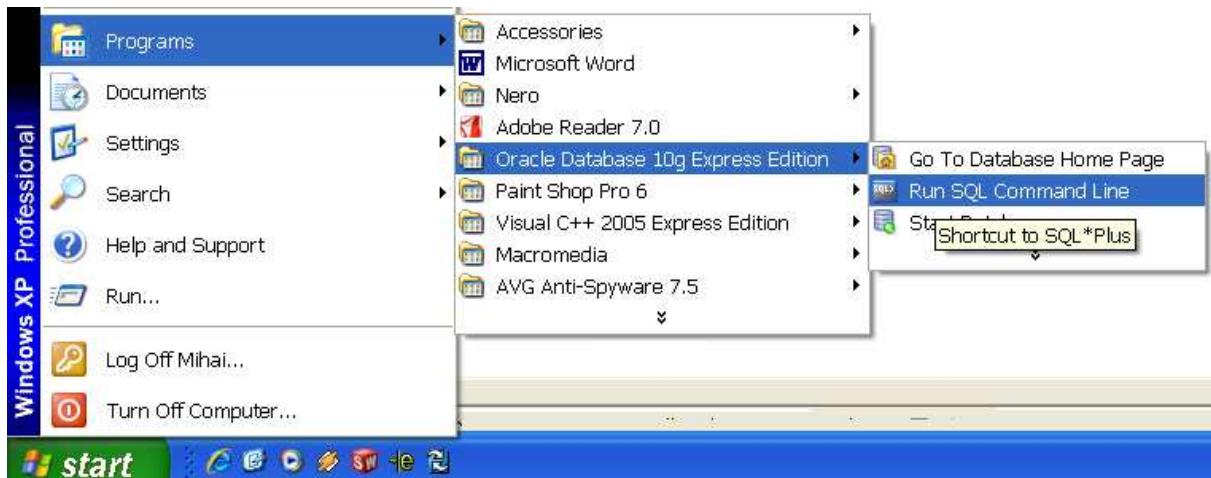
În 1992 ANSI (*American National Standards Institute*) a definitivat varianta standard a unui limbaj destinat sistemelor de gestiune a bazelor de date denumit *Structured Query Language*, prescurtat SQL (pronunțați "es-q-el"). Frazele SQL permit crearea, actualizarea, interogarea și distrugerea bazelor de date relaționale. Deși toate S.G.B.D. folosesc SQL, adesea ele implementează și funcții care nu există în standard.

Comenzile de bază ale limbajului SQL care sunt **Create, Alter, Select, Insert, Update, Delete** și **Drop** sunt suportate de toate sistemele de gestiune de baze de date și permit realizarea tuturor activităților majore legate de crearea și exploatarea bazelor de date relaționale. Cuvintele cheie ale limbajului SQL pot fi scrise atât cu litere mici cât și cu majuscule.

Odată instalat, serverul Oracle XE oferă două interfețe pentru introducerea comenziilor SQL:

a. Introducerea comenziilor SQL folosind interpretorul de comenzi SQL *Plus executabil într-o fereastră de tip Command Prompt

Lansarea în execuție a interpretorului se realizează accesând *Run SQL Command Line* din grupul de comenzi *Oracle Database 10g Express Edition*:



După lansarea în execuție a interpretorului este necesară conectarea la serverul Oracle XE printr-o comandă *connect*, ca în exemplul următor, după care pot fi introduse comenzi. Comenziile SQL pot fi introduse pe mai multe linii, marcarea sfârșitului introducerii unei comenzi realizându-se printr-un caracter '/'.

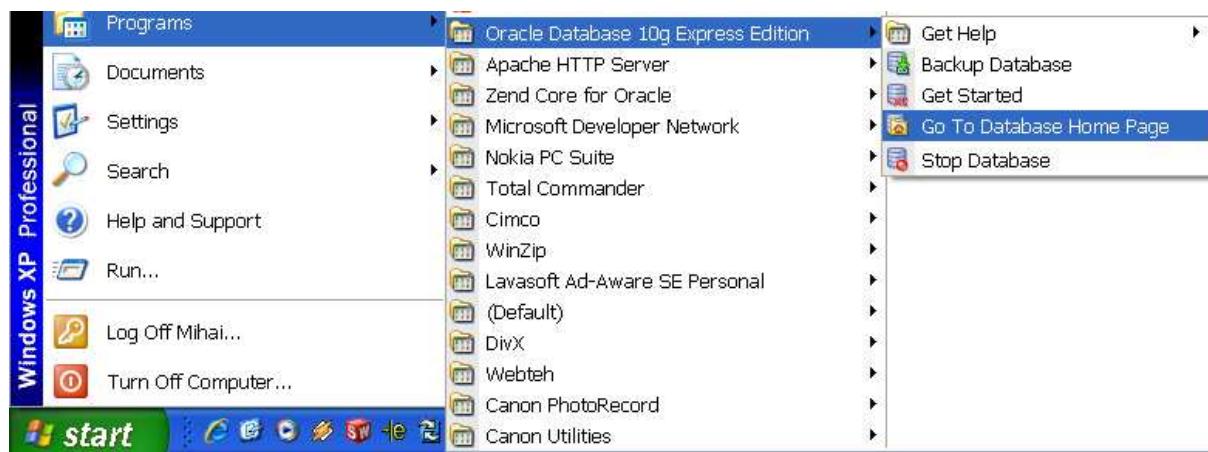
Run SQL Command Line

```
SQL*Plus: Release 10.2.0.1.0 - Production on Sun Feb 11 19:26:45 2007
Copyright (c) 1982, 2005, Oracle. All rights reserved.

SQL> connect
Enter user-name: biblio
Enter password:
Connected.
SQL> create table cafea
  2  (nume varchar(50),
  3  ziua varchar(8),
  4  nr_cafele number(2,0)
  5  )
  6  /
Table created.

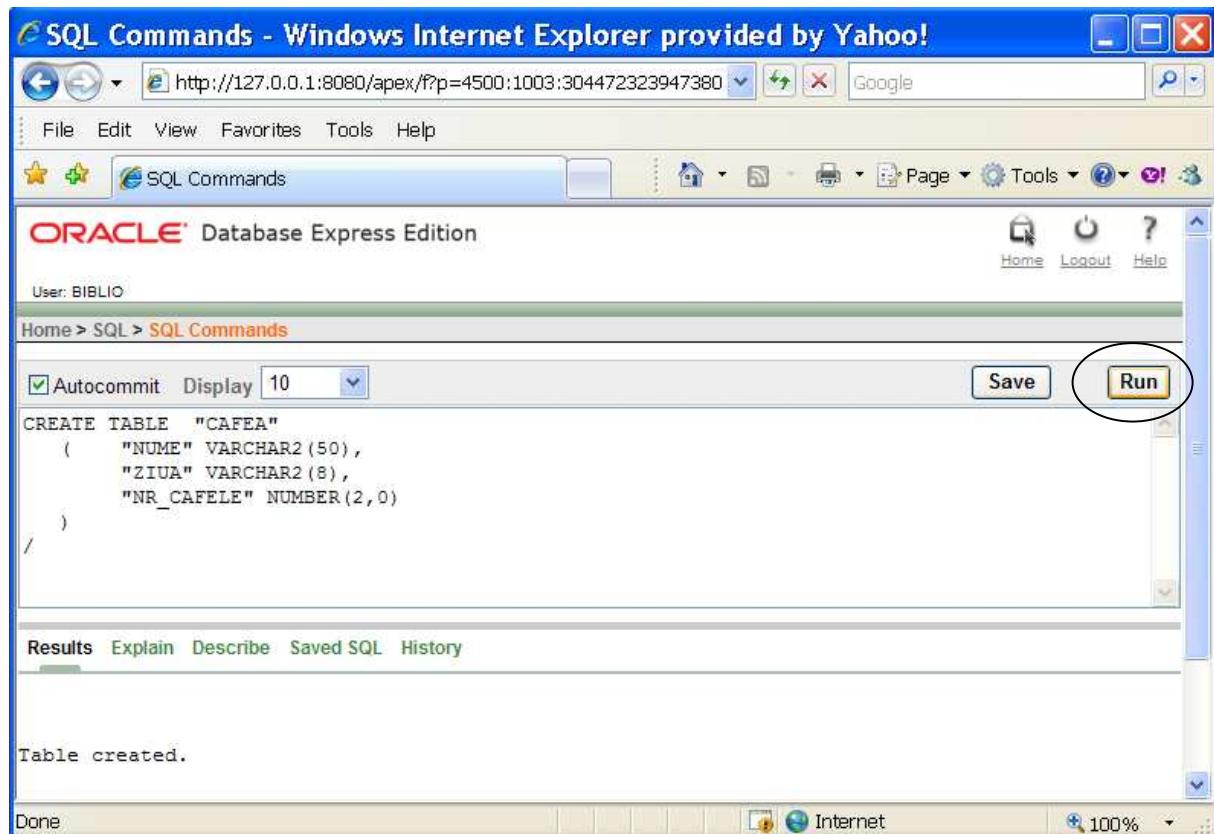
SQL> -
```

b. Introducerea comenzilor SQL folosind interfață grafică:



După conectare se selectează *SQL Commands / Enter Command*.





Comenzile introduse în fereastra *SQL Commands* pot fi terminate prin ';' ca în MySQL de exemplu sau se poate chiar omite caracterul terminal. Astfel pentru suprimarea tabelului *cafea* se poate scrie:

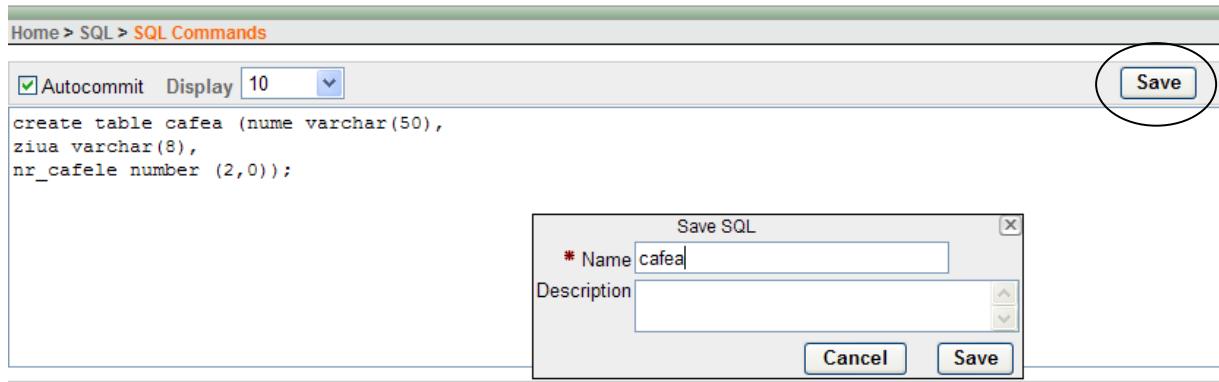
drop table cafea;

sau pur și simplu

drop table cafea

efectul fiind același.

Comenzile SQL pot fi memorate individual apăsând butonul *Save*:



Comenzile memorate pot fi reexecutate prin selectare cu mouse-ul (dublu clic):

The screenshot shows the Oracle SQL developer interface. In the top-left corner, there's a breadcrumb navigation: Home > SQL > SQL Commands. Below it is a toolbar with Autocommit checked, a Display dropdown set to 10, and Save/Run buttons. The main area contains a SQL editor with the following code:

```
create table cafea (nume varchar(50),
ziua varchar(8),
nr_cafele number (2,0));
```

Below the editor is a results grid titled "Results". It has columns: Owner (dropdown set to All Users), Name, Description, SQL, Updated By, and Last Updated. There are two rows:

- Row 1: Owner BIBLIO, Name cafea, Description -, SQL: create table cafea (nume varchar(50), ziua varchar(8), nr_cafele number (2,0));, Updated By BIBLIO, Last Updated 10 minutes ago.
- Row 2: Owner BIBLIO, Name cafea_drop, Description -, SQL: drop table cafea, Updated By BIBLIO, Last Updated 9 minutes ago.

A red circle highlights the "Name" column header, and a hand cursor is positioned over the "cafea" entry in the first row.

Tipuri de date suportate de Oracle

a. Siruri de caractere

Pentru păstrarea sirurilor de caractere, Oracle definește patru tipuri de date:

- VARCHAR2 - pentru siruri de caractere de lungime variabilă,
- NVARCHAR2 pentru siruri de caractere de lungime variabilă în format UNICODE (16 biți / caracter),
- CHAR - pentru siruri de caractere având lungime fixă,
- NCHAR - pentru siruri de caractere având lungime fixă, în format UNICODE.

Indiferent de tip, la declararea unui câmp trebuie precizată lungimea:

nume varchar2(50)

Pentru VARCHAR2 lungimea specificată este cea maximă admisă în timp ce pentru CHAR ea va fi efectiv utilizată, sirurile de lungime mai mică fiind completate la dreapta cu spații. Rezultă că VARCHAR2 este mai eficient, în exemplele prezentate în continuare acest tip fiind utilizat sistematic.

b. Date numerice

Datele numerice pot fi declarate în Oracle folosind unul dintre tipurile următoare:

- NUMBER - pentru numere zecimale,
- BINARY_FLOAT - pentru numere reale (memorate fără conversie în baza 10)
- BINARY_DOUBLE - pentru numere reale în dublă precizie (memorate fără conversie).

Cel mai frecvent se folosește tipul NUMBER. Pentru declararea unei date de tip NUMBER se poate scrie:

inaltime number(3)

sau,

inaltime number(3,0)

Pentru numere care au o parte întreagă și una zecimală se scrie:

pret NUMBER(7,2)

Însemnând reprezentarea câmpului *pret* folosind 7 poziții zecimale, ultimele două fiind folosite pentru partea reală.

Exemple:

Număr	Declarație	Valoare memorată
123.89	NUMBER	123.89
123.89	NUMBER(3)	124
123.89	NUMBER(6 , 2)	123.89
123.89	NUMBER(6 , 1)	123.9
123.89	NUMBER(3)	eroare (depășire)
123.89	NUMBER(4 , 2)	eroare (depășire)

BINARY_FLOAT și BINARY_DOUBLE sunt tipuri recomandate în cazul în care datele astfel reprezentate sunt folosite la calcule mai complicate.

c. Tipuri pentru timp și dată calendaristică

Pentru declararea câmpurilor care vor păstra data calendaristică sau timpul, Oracle folosește tipurile DATE și TIMESTAMP.

Formatele implicite de introducere a datei și a timpului rezultă din tabelul următor.

Valoare	Tip dată	Val. memorată
12-MAR-2007	DATE	12-MAR-07
2007-02-01 01:02:04.1234	TIMESTAMP	01-FEB-07 01.02.04.123400 AM

d. Tipuri pentru memorarea imaginilor

Pentru declararea câmpurilor destinate păstrării în baza de date a imaginilor, Oracle folosește tipurile CLOB sau BLOB.

Crearea tabelelor în Oracle XE

Comanda CREATE TABLE

Comanda CREATE TABLE servește la crearea unui nou tabel și la descrierea câmpurilor acestuia. Ea are formatul general:

```
CREATE TABLE nume
  (nume_câmp tip_câmp [(marime [,precizie]])
  [NULL | NOT NULL]
  [PRIMARY KEY | UNIQUE]
  [,nume_câmp tip_câmp [(marime [,precizie])]]
  [NULL | NOT NULL]
  )
```

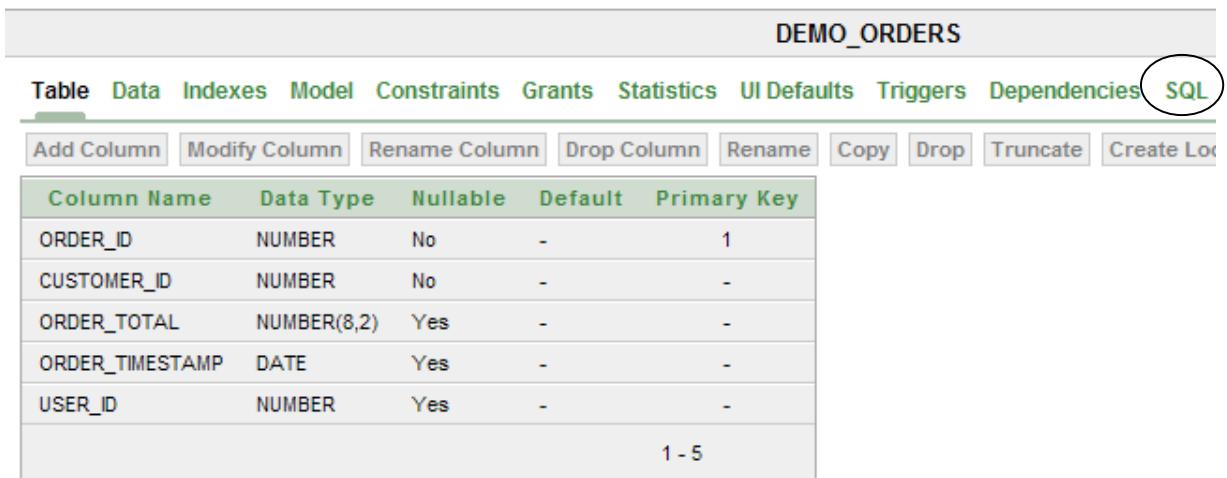
Exemplu:

```
CREATE TABLE regiune
```

```
  (ID_regiune CHAR(2) NOT NULL PRIMARY KEY,
   nume VARCHAR2(40)
  )
```

Crearea tabelelor va fi realizată mai ușor folosind interfața serverului Oracle XE, aşa cum s-a procedat în primele cursuri. În cazul în care în momentul creării unui tabel se impun restricții asupra câmpurilor, se declară chei străine sau se declară o cheie primară, fraza SQL corespunzătoare va conține un număr de restricții introduse folosind cuvântul rezervat "*constraints*".

Exemplu:



The screenshot shows the Oracle SQL Developer interface with the 'DEMO_ORDERS' table selected. The 'SQL' tab is highlighted with a red oval. The table structure is as follows:

Column Name	Data Type	Nullable	Default	Primary Key
ORDER_ID	NUMBER	No	-	1
CUSTOMER_ID	NUMBER	No	-	-
ORDER_TOTAL	NUMBER(8,2)	Yes	-	-
ORDER_TIMESTAMP	DATE	Yes	-	-
USER_ID	NUMBER	Yes	-	-
1 - 5				

```
CREATE TABLE "DEMO_ORDERS"
(
  "ORDER_ID" NUMBER NOT NULL ENABLE,
  "CUSTOMER_ID" NUMBER NOT NULL ENABLE,
  "ORDER_TOTAL" NUMBER(8,2),
```

```

"ORDER_TIMESTAMP" DATE,
"USER_ID" NUMBER,
CONSTRAINT "DEMO_ORDER_PK" PRIMARY KEY ("ORDER_ID") ENABLE,
CONSTRAINT "DEMO_ORDER_TOTAL_MIN" CHECK (order_total >= 0) ENABLE,
CONSTRAINT "DEMO_ORDERS_CUSTOMER_ID_FK" FOREIGN KEY ("CUSTOMER_ID")
REFERENCES "DEMO_CUSTOMERS" ("CUSTOMER_ID") ENABLE,
CONSTRAINT "DEMO_ORDERS_USER_ID_FK" FOREIGN KEY ("USER_ID")
REFERENCES "DEMO_USERS" ("USER_ID") ENABLE
)
/

```

Obs. În fraza SQL afișată denumirile câmpurilor, constrângerilor sau denumirea tabelului apar între ghilimele. Această scriere permite folosirea de denumiri conținând spații (de evitat!). Dacă denumirile nu pot da naștere la confuzii, ghilimelele pot fi omise.

Cele patru constrângerile declarate se referă la cheia primară, limitarea unei valori și declară două câmpuri ca fiind chei străine, precizând și tabelele legate. În primul exemplu cheia primară a fost declarată adăugând descrierii câmpului ID_regiune cuvintele PRIMARY KEY.

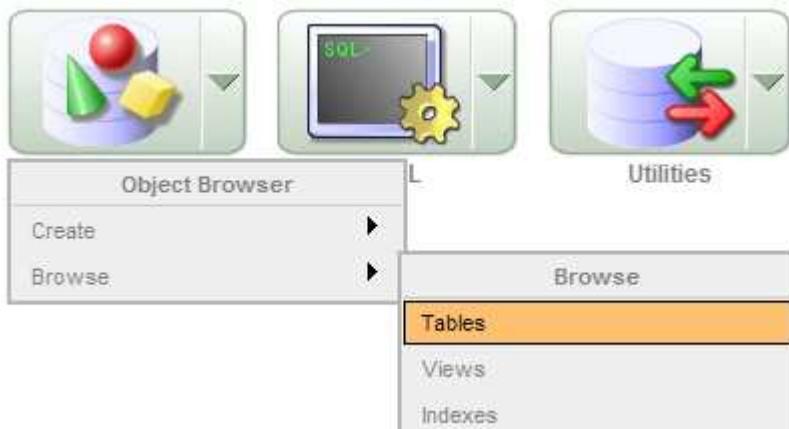
Din aceste exemple se observă diversitatea modatităților de scriere a unei comenzi SQL. Pentru a limita volumul de cunoștințe legate de sintaxa limbajului, în cele ce urmează comenzile vor fi realizate și testate folosind interfața aplicației. Așa cum s-a văzut deja, interfața permite și afișarea comenziilor create. Deoarece în timpul realizării unei baze de date se pot produce incidente mergând până la distrugerea serverului Oracle XE, este bine să fie păstrate într-un fișier având extensia .sql un număr cât mai mare de fraze a căror executare să permită refacerea tabelelor și repopularea lor cu date.

Comanda DROP TABLE

Comanda DROP TABLE permite suprimarea unui tabel. Sintaxa comenzi este:

```
DROP TABLE Nume_tabel
```

Pentru a da comanda folosind interfața grafică se selectează succesiv *Object Browser / Browse / Tables*, se selectează apoi tabelul și se apasă butonul *Drop*:



Tables

DEMOS_STATES

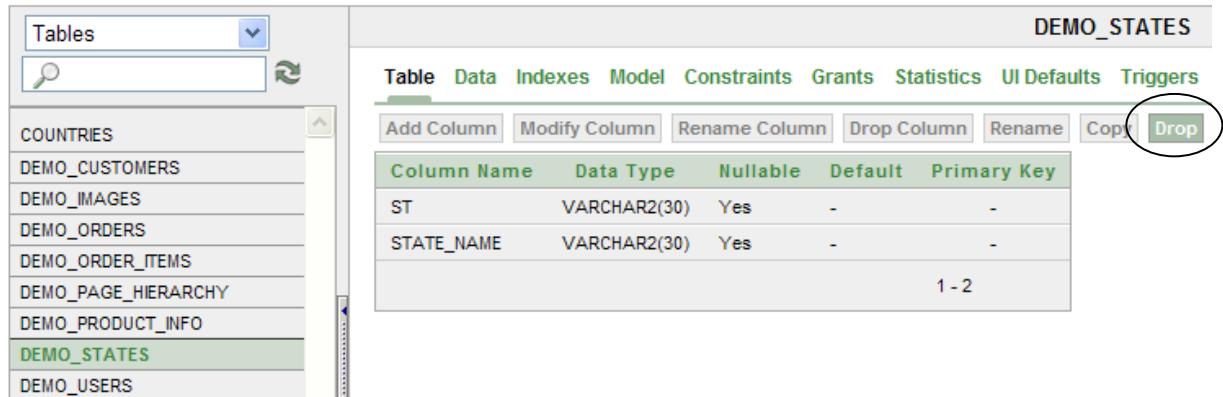
Table Data Indexes Model Constraints Grants Statistics UI Defaults Triggers

Add Column Modify Column Rename Column Drop Column Rename Copy Drop

Column Name	Data Type	Nullable	Default	Primary Key
ST	VARCHAR2(30)	Yes	-	-
STATE_NAME	VARCHAR2(30)	Yes	-	-

1 - 2

COUNTRIES
DEMO_CUSTOMERS
DEMO_IMAGES
DEMO_ORDERS
DEMO_ORDER_ITEMS
DEMO_PAGE_HIERARCHY
DEMO_PRODUCT_INFO
DEMO_STATES
DEMO_USERS



Comanda SELECT

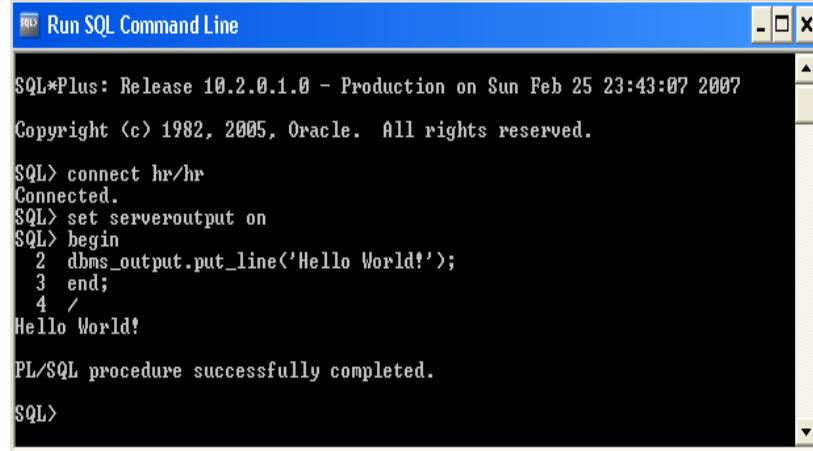
Comanda SELECT creează o mulțime de selecție. Aceasta poate conține un articol, mai multe articole sau niciunul. Articolele din mulțimea de selecție conțin numai câmpurile indicate de programator. Din acest punct de vedere se poate considera că SELECT creează un sub tabel conținând numai informațiile specificate. Câmpurile înregistrărilor mulțimii de selecție pot proveni dintr-un tabel sau din mai multe tabele legate.

Comanda SELECT are formatul general:

```
SELECT [DISTINCT] coloana1 [,coloana2]
      FROM tabel_1[,tabel_2, ...]
      [WHERE condiții]
      [GROUP BY listă-coloane]
      [HAVING conditii]
      [ORDER BY listă-coloane [ASC | DESC] ]
```

Dintre cele 5 clauze ale comenzii SELECT numai clauza FROM este obligatorie. Fiecare dintre clauze are la rândul ei reguli și parametri pentru construcție, făcând din SELECT cea mai complexă comandă a limbajului SQL. În frazele SELECT sirurile de caractere se pun între caractere ' (apostrof).

Pentru construirea unei comenzi SELECT se poate folosi asistentul *Query Builder*.



The screenshot shows a window titled "Run SQL Command Line". The console output is as follows:

```
SQL*Plus: Release 10.2.0.1.0 - Production on Sun Feb 25 23:43:07 2007
Copyright (c) 1982, 2005, Oracle. All rights reserved.

SQL> connect hr/hr
Connected.
SQL> set serveroutput on
SQL> begin
 2  dbms_output.put_line('Hello World!');
 3  end;
 4 /
Hello World!

PL/SQL procedure successfully completed.

SQL>
```

Aplicația va afișa trei panouri:

- în stânga un panou conținând tabelele cuprinse în schema curentă :



EMPLOYEES
COUNTRIES
DEMO_CUSTOMERS
DEMO_IMAGES
DEMO_ORDERS
DEMO_ORDER_ITEMS
DEMO_PAGE_HIERARCHY
DEMO_PRODUCT_INFO
DEMO_STATES
DEMO_USERS
DEPARTMENTS
EMPLOYEES

- În centrul un panou conținând tabelele selectate:

EMPLOYEES		JOBS	
EMPLOYEE_ID	789	JOB_ID	A
FIRST_NAME	A	JOB_TITLE	A
LAST_NAME	A	MIN_SALARY	789
EMAIL	A	MAX_SALARY	789
PHONE_NUMBER	A		
HIRE_DATE	31		
JOB_ID	A		

Notă: *JOB_ID* din *Jobs* = *JOB_ID* din *Employees*. Pentru impunerea legăturii se va selecta succesiv cu mouse-ul *JOB_ID* din *Employees* și din *Jobs*.

- În partea de jos un panou cuprindând câmpurile selectate și alte specificații:

Column	Alias	Object	Condition	Sort Type	Sort Order	Show
▲ ▼ FIRST_NAME	FIRST_NAME	EMPLOYEES		Asc		<input checked="" type="checkbox"/>
▲ ▼ LAST_NAME	LAST_NAME	EMPLOYEES		Asc	1	<input checked="" type="checkbox"/>
▲ ▼ EMAIL	EMAIL	EMPLOYEES		Asc		<input checked="" type="checkbox"/>
▲ ▼ JOB_TITLE	JOB_TITLE	JOBS		Asc		<input checked="" type="checkbox"/>

Selectarea dintr-un singur tabel

Exemple fondamentale:

SELECT Nume, Prenume **FROM** cititor

SELECT * FROM cititor

SELECT * FROM cititor **WHERE** nume = 'Popescu'

SELECT nume, prenume, adresa **FROM** cititor **WHERE** nume **LIKE** 'Po%'

SELECT Nume, Prenume **FROM** cititor **ORDER BY** Nume

Numele coloanelor scrise după *SELECT* precizează câmpurile mulțimii de selecție. Dacă se dorește ca mulțimea de selecție să aibă aceeași structură cu tabelul, în locul listei de coloane se pune *.

Clauza WHERE permite limitarea mulțimii de selecție prin introducerea unei condiții care specifică înregistrările căutate. În cazul selectării din mai multe tabele, această clauză poate fi folosită și pentru a introduce legăturile dintre tabele. La scrierea condiției se pot folosi operatorii logici AND, OR sau NOT precum și operatorii relationali:

=	Egal
>	Mai mare
<	Mai mic
>=	Mai mare sau egal
<=	Mai mic sau egal
<> sau !=	Diferit de
LIKE	*Vezi nota

Notă: Operatorul *LIKE* permite selectarea liniilor care conțin în câmpul testat siruri de caractere în care se regăsește o succesiune de caractere precizată. Pentru definirea succesiunii de caractere căutată se poate folosi caracterul generic "%". De exemplu clauza *WHERE nume LIKE 'Po%*' va permite selectarea liniilor în care *nume = Pop* și *nume=Popescu*. Clauza *WHERE prenume LIKE '%a'* permite selectarea tuturor liniilor în care prenumele se termină în "a".

Exemple:

```
SELECT * FROM cititor WHERE nume = 'Popescu' AND
    prenume='Mihai'
```

```
SELECT * FROM cititor WHERE nume = 'Popescu' OR nume='Mihai'
```

```
SELECT * FROM cititor WHERE nume = 'Popescu' AND prenume  
IN('Ioan','Vasile','Griqore')
```

IN specifică o mulțime căreia trebuie să-i aparțină câmpul specificat (*prenume*).

```
SELECT * FROM cititor WHERE nume = 'Popescu' AND data_nasterii
BETWEEN '12-MAY-1968' AND '29-MAY-1980'
```

Clauza *BETWEEN* permite definirea unui interval căruia trebuie să-i aparțină câmpul specificat (*data_nasterii*).

Clauza ORDER BY servește la impunerea ordinii de afișare a înregistrărilor mulțimii de selecție. Ordonarea poate fi realizată după valorile unuia sau a mai multor câmpuri, crescător (ASC) sau descrescător (DESC). În cazul în care se folosesc mai multe câmpuri, acestea sunt separate prin virgulă.

```
SELECT * FROM Cititor ORDER BY Nume,Prenume,ID_cititor
SELECT * FROM Angajati ORDER BY salar DESC, Nume ASC, Prenume
ASC
```

Clauza DISTINCT permite realizarea unei mulțimi de selecție care conține înregistrări distincte, care diferă prin cel puțin o valoare a unui câmp. De exemplu :

```
SELECT DISTINCT Localitate FROM cititor
```

Ultima frază va provoca afișarea numelor localităților în care domiciliază cititorii. Fără clauza *DISTINCT*, numele orașului Cluj-Napoca ar fi fost repetat de mai multe ori.

Clauza GROUP BY permite gruparea înregistrărilor după valoarea unui câmp.

Exemplul 1:

```
SELECT Localitate, COUNT(*) as Nr_Cititori FROM Cititor GROUP BY
Localitate
```

În cazul în care se realizează o grupare, fiecare dintre liniile mulțimii de selecție se referă la un grup de înregistrări și nu la înregistrări simple. În exemplul precedent, *GROUP BY Localitate* precizează câmpul după care se realizează gruparea. În afară valorii câmpului după care se face gruparea, în astfel de situații liniile mulțimii de selecție pot conține rezultatul aplicării unor funcții matematice asupra articolelor care formează grupul (suma valorilor dintr-un câmp, media valorilor, valoarea maximă sau minimă, numărul de articole care formează grupul etc). Pentru câmpurile care vor conține rezultatul aplicării unor

funcții se recomandă folosirea clausei *AS nume* pentru atribuirea unui nume relevant coloanei respective. În exemplul dat s-a afișat rezultatul funcției *COUNT(*)*, care numără înregistrările din grup. Funcțiile care pot fi apelate pentru câmpuri numerice sunt prezentate în tabelul de mai jos.

MIN	<i>returnează cea mai mică valoare dintr-o coloană (câmp numeric)</i>
MAX	<i>returnează cea mai mare valoare dintr-o coloană (câmp numeric)</i>
SUM	<i>returnează suma valorilor numerice dintr-o coloană (câmp numeric)</i>
AVG	<i>returnează media valorilor dintr-o coloană (câmp numeric)</i>
COUNT	<i>returnează numărul de valori dintr-o coloană</i>
COUNT DISTINCT	<i>returnează numărul de valori distincte dintr-o coloană</i>

Exemplul 2:

```
SELECT Sectie, MAX(salar) as Sal_Max FROM Angajati
      GROUP BY Sectie
```

Se pot însă aplica aceleași funcții și întregului fișier, fără gruparea articolelor, caz în care mulțimea de selecție va conține o singură înregistrare:

```
SELECT AVG(salar) FROM Angajati
SELECT AVG(salar) FROM Angajati WHERE functie='Zidar'
SELECT COUNT(*) FROM Cititor
```

Clauza **HAVING** servește la precizarea unui filtru care se aplică grupurilor de articole, dacă este prezentă clauza **GROUP BY**.

Exemplu:

```
SELECT Sectie, AVG(Salar) FROM Angajati GROUP BY Sectie HAVING
AVG(Salar) > 500
```

Dacă în fraza SELECT lipsește clauza GROUP BY, folosirea clauzei HAVING nu se justifică, ea având același efect ca și clauza WHERE.

O situație aparte prezintă comenziile **SELECT** care nu realizează o mulțime de selecție ci un calcul matematic. În acest caz *from nume_tabel* va fi înlocuit prin *from dual*, ca în exemplul următor:

Home > SQL > SQL Commands

Autocommit Display 10 ▾

```
select sin(3.1415926535/2.)/4. from dual
```

←

Results Explain Describe Saved SQL History

SIN(3.1415926535/2.)/4.
.2499999999999999748036697699714405

1 rows returned in 0.00 seconds [CSV Export](#)

Cuvântul rezervat *dual* astfel folosit permite respectarea sintaxei comenzi *SELECT*.

Selectarea din mai multe tabele

Principial o bază de date relațională presupune repartizarea datelor în mai multe tabele. Așa cum s-a văzut deja, acest mod de stocare permite eliminarea informațiilor redondante. Pentru a găsi articolele dintr-un tabel legate de o înregistrare din alt tabel trebuie însă indicată în frazele *SELECT* condiția de legătură dintre cele două tabele.

Exemple:

```
SELECT Edituri.Nume, Carti.Titlu, Carti.Anul FROM Edituri, Carti WHERE Nume='Minerva' AND Edituri.CodE=Carti.CodE
```

A doua condiție conținută în clauza *WHERE* este cea care leagă tabelele *Edituri* și *Carti*.

Limbajul SQL permite înlocuirea unei mulțimi care contribuie la selecție printr-o altă frază *SELECT*, ca în exemplul următor.

```
SELECT Edituri.Nume, Carti.Titlu, Carti.Anul FROM Edituri, Carti WHERE Nume='Minerva' AND Edituri.CodE=Carti.CodE AND Titlu.Data IN (SELECT Data FROM Carti WHERE Data BETWEEN '12-MAY-1975' AND '29-MAY-1980')
```

Odată cu standardizarea limbajului SQL, pentru introducerea legăturilor stabilite între tabele prin definirea de chei primare și străine s-a definit și o altă modalitate, respectiv clauza *JOIN*. Folosind *JOIN*, exemplul anterior poate fi scris astfel:

```
SELECT Edituri.Nume, Carti.Titlu, Carti.Anul
FROM Edituri INNER JOIN Carti ON Edituri.CodE=Carti.CodE
WHERE Nume='Minerva'
```

Clauza *JOIN* este însă și soluție într-o altă situație, respectiv când într-un tabel se acceptă pentru o cheie străină valori nule. Un exemplu simplu este cel al tabelelor *Angajati* și *Sotii* prezentate în continuare.

ANGAJATI					SOTI		
EDIT	ID_ANG	NUME	PRENUME	ID_SOT	EDIT	NUME_PRENUME	ID_SOT
	1	Avram	Lucian	-		Ionescu Marcel	2
	2	Breban	Ioan	-		Popescu Simona	1
	5	Ionescu	Carmen	2	row(s) 1 - 2 of 2		
	6	Banu	Mircea	-			
	4	Anghel	Mihai	-			
	3	Popescu	Ilie	1			
row(s) 1 - 6 of 6							

Dintre angajații din primul tabel doi sunt căsătoriți, în cazul lor ID_SOT are valori nenule.

Dacă se dorește realizarea unei multimi de selecție care să cuprindă angajații, iar pentru cei căsătoriți numele și prenumele soțului/sotiei, fraza *SELECT* ar putea fi următoarea:

```
select Angajati.Nume, Angajati.Prenume,Soti.Nume_prenume
from Angajati, Soti where Angajati.ID_SOT=SOTI.ID_SOT
```

Rezultat:

NUME	PRENUME	NUME_PRENUME
Ionescu	Carmen	Ionescu Carmen
Popescu	Ilie	Popescu Ilie

Rezultatul nu este cel dorit deoarece prin condiția pusă sunt excluși din mulțimea de selecție angajații necăsătoriți.

Soluția este oferită de clauza *LEFT OUTER JOIN*:

```
select Angajati.Nume, Angajati.Prenume,Soti.Nume_prenume
from Angajati left outer join Soti on Angajati.ID_SOT=SOTI.ID_SOT
```

Rezultatul va cuprinde toate liniile din primul tabel iar pentru liniile din primul tabel care au corespondent în al doilea tabel, va include și informațiile corespunzătoare din al doilea tabel.

NUME	PRENUME	NUME_PRENUME
Avram	Lucian	-
Breban	Ioan	-
Ionescu	Carmen	Ionescu Marcel
Banu	Mircea	-
Anghel	Mihai	-
Popescu	Ilie	Popescu Simona

Similar se poate folosi clauza *RIGHT OUTER JOIN* pentru includerea tuturor înregistrărilor din al doilea tabel și selectarea din primul doar a celor care satisfac relația de legătură.

Utilizarea parametrilor

SQL permite ca la scrierea comenzi SELECT să se folosească parametrii. Valorile acestora sunt cerute într-o fereastră separată, în momentul executării interogării.

Exemplu :

```
sql>select * from edituri where cod_edit > :coded;
```

The screenshot shows a web-based SQL command interface. At the top, there's a navigation bar with 'Home > SQL > SQL Commands'. Below it is a configuration panel with 'Autocommit' checked and 'Display' set to 10. The main area contains the SQL query:

```
select * from edituri where cod_edit > :coded;
```

Below the query, a browser window titled 'http://127.0.0.1:8080 - Enter Bind Variables ...' displays a form with a single input field containing ':CODED 5'. To the right of the input field is a 'Submit' button. The browser status bar at the bottom shows 'Done' and 'Internet'.

COD_EDIT	NUME
6	Eminescu
7	Cartea Romaneasca
8	Miron
9	Gramond

Comanda INSERT

Comanda INSERT adaugă un rând într-un tabel existent.

Exemplu:

```
INSERT INTO Edituri(CodE,Nume,Adresa,Telefon) VALUES(121,
'Albatros','B-dul Tomis Nr. 32 Constanta','0745654765')
```

Lista de câmpuri de după numele tabelei poate fi omisă dacă toate câmpurile primesc valori iar acestea sunt scrise în ordinea definită la creare (în care ele apar în capul de tabel). Dacă un câmp nu primește valoare el va primi implicit valoarea NULL, dacă la creare, prin modul de declarare a câmpului, introducerea unei astfel de valori este autorizată.

Comanda DELETE

Comanda DELETE suprimă una sau mai multe înregistrări dintr-un fișier. Ca și în cazul comenzi SELECT, pentru definirea unui set de înregistrări care vor fi șterse se utilizează clauza WHERE.

Exemple:

```
DELETE FROM Autori WHERE CodAut=7
```

```
DELETE FROM Edituri
```

Ultima comandă suprimă toate înregistrările din tabelul *Edituri*.

Comanda UPDATE

Comanda UPDATE permite modificarea unei înregistrări sau a unui set de înregistrări. Pentru precizarea setului de înregistrări afectate se folosește clauza WHERE.

Exemplu:

```
UPDATE Edituri SET Adresa='str. 1 Mai Nr. 12', Telefon='0745343435'
WHERE Nume='Dacia'
```

```
UPDATE Angajati SET Salar=Salar*1.2 WHERE Salar<450
```

Capitolul IV

Limbajul PL/SQL

Generalități

Limbajul SQL (Structured Query Language) este un *limbaj declarativ* orientat pe obținerea de mulțimi de selecție într-o aplicație în arhitectură client-server. Eficiența sa este remarcabilă dar în cazul în care elementele unei mulțimi de selecție trebuie accesate pe rând, nu în grup, utilizarea sa nu mai este posibilă. Pe de altă parte SQL nu oferă nici suportul necesar realizării interfeței unei aplicații. În contrast cu SQL, *limbajele procedurale* pot manipula linii individuale dintr-o mulțime de selecție. Ele dispun de funcții care permit trecerea de la o linie la alta și ramificarea codului în funcție de rezultatul testării valorilor preluate din tabelele bazei de date.

Limbajul PL/SQL (*Procedural Language extensions to SQL*) a fost dezvoltat de compania Oracle ca extensie a limbajului declarativ SQL. PL/SQL este un limbaj de programare procedural care poate fi folosit pentru descrierea unor prelucrări complexe sau pentru programarea unor secvențe de cod care trebuie executate automat la apariția unui eveniment (*triggere*).

O caracteristică remarcabilă a limbajului PL/SQL este faptul că procedurile sau funcțiile scrise vor fi memorate în baza de date. Aceleași prelucrări pot fi realizate de regulă și în cadrul aplicațiilor client care accesează datele din bază, dar în cazul modificării unei metode de calcul trebuie refăcute aplicațiile client afectate și suportate toate costurile pe care le implică redistribuirea unor noi versiuni ale aplicațiilor client.

Având în vedere faptul că de multe ori aplicațiile din domeniul bazelor de date sunt în arhitectură client-server și aplicația client accesează aplicația server (Oracle XE) prin intermediul unei rețele, utilizarea pe cât posibil a procedurilor scrise în PL/SQL poate ameliora semnificativ viteza de prelucrare. În cazul unei proceduri care ar implica transferul spre aplicația client a unui volum mare de date (rezultatul unei interogări de exemplu), întârzierile cauzate de rețea prin care se accesează serverul de baze de date pot fi mari. Dacă prelucrarea datelor nu presupune afișarea acestora în aplicația client, mult mai eficientă este soluția prelucrării datelor pe server, într-o procedură scrisă în PL/SQL.

PL/SQL este bazat pe limbajul ADA, o parte dintre construcțiile sale sintactice provenind din Pascal.

Scrierea unui bloc anonim

De regulă programarea în PL/SQL înseamnă definirea unor proceduri sau funcții, în sensul pe care acestea le au în oricare dintre limbajele procedurale

cunoscute. O procedură sau o funcție poate fi introdusă folosind fereastra SQL Plus sau folosind interfața grafică, în *SQL Command* sau *Script Editor*.

O succesiune de instrucțiuni în PL/SQL care nu sunt destinate definirii unei funcții sau proceduri formează un *bloc anonim* care va fi executat doar o singură dată.

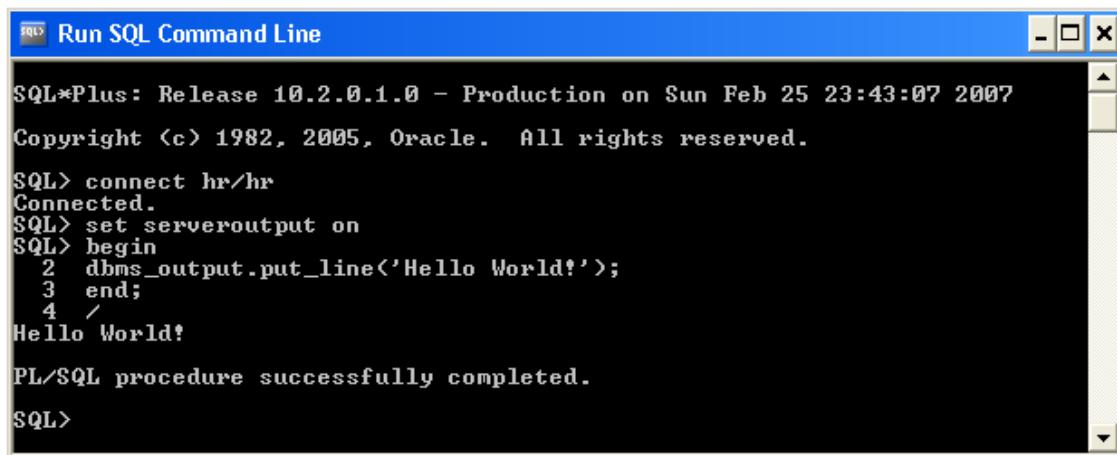
Exemplu:

```
begin
    dbms_output.put_line('Hello World');
end;
/
```

În fereastra SQL*Plus secvența de cod scrisă va fi precedată de comanda

```
set serveroutput on
```

Aceasta trebuie dată la începutul sesiunii de lucru și realizează activarea afișării la consolă a rezultatului operațiilor de ieșire.



The screenshot shows a Windows application window titled "Run SQL Command Line". The title bar has standard minimize, maximize, and close buttons. The main area is a black terminal window with white text. It displays the following SQL*Plus session:

```
SQL*Plus: Release 10.2.0.1.0 - Production on Sun Feb 25 23:43:07 2007
Copyright (c) 1982, 2005, Oracle. All rights reserved.

SQL> connect hr/hr
Connected.
SQL> set serveroutput on
SQL> begin
 2  dbms_output.put_line('Hello World!');
 3  end;
 4 /
Hello World!
PL/SQL procedure successfully completed.

SQL>
```

Dacă se folosește interfața grafică nu mai este necesară comanda *set serveroutput on*.

User: HR

Home > SQL > SQL Commands

Autocommit Display 10 ▾

```
begin
dbms_output.put_line('Hello World');
end
/
```

Results Explain Describe Saved SQL History

Hello World

Statement processed.

Exemplul următor conține declarația unei variabile, *nr*. Pentru a afișa valoarea acesteia se realizează conversia ei în sir de caractere folosind funcția *to_char()*:

Home > SQL > SQL Commands

Autocommit Display 10 ▾

```
declare
nr number;
begin
nr := 4;
dbms_output.put_line('Nr. = ' || to_char(nr));
end;
/
```

Results Explain Describe Saved SQL History

Nr. = 4

Statement processed.

Elementele limbajului PL/SQL

Codul PL/SQL poate fi conținut în *blocuri anonime* sau în blocuri care conțin *subprograme memorate* în baza de date (proceduri sau funcții).

Un *bloc anonim* se introduce în fereastra în care se introduc comenzi SQL. El nu este memorat în mod normal în baza de date în vederea reutilizării ulterioare. Interfața serverului Oracle XE permite totuși memorarea unui bloc, întocmai ca și memorarea unei comenzi SQL oarecare. Cele două exemple anterioare sunt blocuri anonime.

Un *subprogram memorat* (i se zice uneori stocat) este un bloc PL/SQL pe care serverul Oracle îl compilează și îl memorează în baza de date. Subprogramul poate fi ulterior apelat dintr-o aplicație sau dintr-un alt bloc PL/SQL. Subprogramele pot fi proceduri sau funcții. Diferența dintre cele două este faptul că o funcție returnează o valoare.

Un *pachet* (engl. *package*) este format dintr-un grup de subprograme și de declarații de variabile. Serverul Oracle memorează elementele conținute într-un pachet, acestea putând fi apelate din alte pachete sau subprograme.

Sintaxa unui bloc PL/SQL

Un bloc anonim PL/SQL are sintaxa următoare:

<i>Bloc anonim</i>
<pre>DECLARE declaratii de variabile BEGIN cod program EXCEPTION cod tratare exceptii END;</pre>

Dacă blocul conține o procedură memorată în baza de date, sintaxa sa este următoarea:

<i>Subprogram memorat de tip procedură</i>
<pre>CREATE OR REPLACE PROCEDURE "nume" (lista_parametri) IS <i>declaratii variabile</i> BEGIN <i>cod program</i> EXCEPTION <i>cod tratare exceptii</i> END;</pre>

Secțiunea *DECLARE* cuprinde declarații de variabile simple, variabile structurate, variabile tip cursor, funcții sau proceduri ajutătoare. Zona de declarații este opțională. Pentru funcții, proceduri și triggere cuvântul *DECLARE* lipsește, blocul de declarații fiind implicit cuprins între linia de declarare a funcției sau a procedurii și *BEGIN*.

Blocul introdus prin *EXCEPTION* este opțional și realizează tratarea erorilor apărute în timpul execuției codului programului.

Caracterul ';' se folosește pentru a marca sfârșitul unei instrucțiuni sau a unui bloc, dacă apare după *END*.

Codul poate cuprinde blocuri interioare. Exemplu:

```
DECLARE
    declaratii variable
BEGIN
    -- cod program
    BEGIN
        codul blocului inclus
    EXCEPTION
        tratare exceptii
    END;
    -- cod program (continuare)
END;
```

Pentru transformarea exemplelor din următoarele subcapitole în exemple executabile, înaintea declarațiilor de variabile va fi introdus fie numele secțiunii de date, *DECLARE* fie secvența de declarare a unei proceduri stocate (CREATE OR REPLACE PROCEDURE "nume" IS).

Comentarii

În PL/SQL comentariile în linie se introduc prin două caractere '-' iar comentariile pe mai multe linii sunt cuprinse între /* respectiv */, ca în limbajul C.

Exemplu:

The screenshot shows a SQL developer interface with the following details:

- Toolbar:** Home > SQL > SQL Commands
- Autocommit:** Checked
- Display:** Set to 10
- Code Area:**

```

DECLARE
    salar NUMBER(6);
    nr_zile_lucrare NUMBER(2);
    salar_zilnic NUMBER(6,2);
-- Urmeaza blocul de calcul
BEGIN
    salar := 1480;
    nr_zile_lucrare := 21;
    salar_zilnic := salar / nr_zile_lucrare;
-- Se afiseaza rezultatul
    DBMS_OUTPUT.PUT_LINE('Salarul pa zi este ' || to_char(salar_zilnic));
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        salar_zilnic := 0;
END;

```
- Annotations:** Two arrows point to the multi-line comment /* ... */. The first arrow points to the opening /*, and the second points to the closing */.
- Bottom Navigation:** Results (highlighted), Explain, Describe, Saved SQL, History

Output results:

```

Salarul pa zi este 70.48
Statement processed.

```

Tipuri de date în PL/SQL

Tipurile de date folosite în PL/SQL pot fi:

- cele folosite în SQL (VARCHAR2, NVARCHAR2, CHAR, NCHAR, NUMBER, BINARY_FLOAT, BINARY_DOUBLE, DATE, TIMESTAMP, CLOB și BLOB),
- tipuri specifice limbajului PL/SQL : BOOLEAN, NUMBER (scris simplu, fără dimensiune) și PLS_INTEGER (pentru variabile având valori întregi),
- tipuri specifice structurate: TABLE, VARRAY și RECORD.

Exemplu:

```
-- Declaratii de variabile
nume VARCHAR2(30);
prenume VARCHAR2(25);
marca NUMBER(6);
activ BOOLEAN;
salar_lunar NUMBER(6);
nb_zile_lucrare NUMBER(2);
salar_zilnic NUMBER(6,2);
medie_zile_lucr CONSTANT NUMBER(2) := 21; -- o cconstantă
BEGIN
    NULL;          -- NULL indica lipsa corpului. Este permisa pt. testare.
END;
/
```

Obs. Pentru a defini constanta *medie_zile_lucr* s-a folosit cuvântul rezervat *CONSTANT* și imediat s-a atribuit valoarea corespunzătoare.

Variabilele declarate servesc de multe ori la memorarea unor valori din tabelele bazei de date, obținute folosind comenzi *SELECT*. În astfel de cazuri este esențial ca tipul declarat pentru o astfel de variabilă să coincidă cu tipul coloanei tabelului din care va primi valori. Pentru a evita erorile greu de depistat cauzate de declararea eronată a acestor variabile, PL/SQL oferă soluția simplă a preluării tipului câmpului care va furniza valori folosind *%TYPE*, astfel:

```
coded edituri.cod_edit%TYPE;
```

Variabila *coded* va fi *NUMBER(5)*, de același tip cu câmpul *cod_edit* din tabelul *edituri*.

O situație asemănătoare apare în cazul variabilelor structurate (de tipul *RECORD*). O astfel de variabilă va avea mai multe câmpuri. Dacă variabila trebuie să preia valorile conținute într-o linie a unui tabel, la declararea ei se va folosi *%ROWTYPE*, astfel:

```
editura edituri%ROWTYPE;
```

Variabila *editura* va fi de tip RECORD și va avea aceleași câmpuri cu tabelul *edituri*. Accesul la câmpuri se realizează folosind operatorul '.' (punct). Exemplu: *editura.cod_edit*, *editura.nume* etc.

Cursoare

Rezultatul unei interogări este păstrat de PL/SQL într-o zonă de memorie denumită *cursor*.

O aplicație poate crea mai multe cursoare, din acest motiv fiind necesară declararea acestora. Declararea unui cursor se realizează în secțiunea de declarații a blocului (procedurii, funcției), astfel:

```
CURSOR c1 IS SELECT ... ;
```

Exemplu:

```
CURSOR c1 IS SELECT nume, prenume FROM angajati WHERE functia = 'zidar';
```

Comanda SQL *SELECT* va fi executată în momentul deschiderii cursorului folosind instrucțiunea *OPEN*. Liniile multșimii de selecție conținute în cursor vor fi prelucrate individual, accesul la linia curentă realizându-se folosind instrucțiunea *FETCH*.

După terminarea prelucrării datelor conținute într-un cursor, acesta trebuie închis folosind instrucțiunea *CLOSE*.

Câteva exemple privind folosirea cursoarelor vor fi inserate după prezentarea instrucțiunilor repetitive.

Identificatori

Numele unei variabile constă dintr-un sir având maximum 30 caractere și format dintr-o literă urmată optional de alte litere, cifre, \$, _. Caracterele '&', '-', '/' și ' ' (spațiu) nu sunt permise. PL/SQL nu este sensibil la tipul literelor - majuscule sau litere mici.

Exemple:

```
-- Declaratii de variabile
numeprenume VARCHAR2(30); -- identificator acceptat
```

```

nume_prenume VARCHAR2(30); -- identificator acceptat, _ permis
nume$prenume VARCHAR2(30); -- identificator acceptat, $ permis
nume#prenume VARCHAR2(30); -- identificator acceptat, # permis
-- nume-prenume identificator neacceptat, minus nepermis
-- nume/prenume identificator neacceptat, / nepermis
-- nume prenume identificator neacceptat, spatiu nepermis
-- NUMEPRENUME identif. neacceptat, acelasi cu numeprenume si NumePrenume
-- NumePrenume identif. neacceptat, acelasi cu numeprenume si NUMEPRENUME
BEGIN
    NULL;
END;
/

```

Operatori

La scrierea codului în PL/SQL se folosesc următoarele tipuri de operatori:

- aritmetici: + - * /
- logici : '=' '>' '<' '>=' '<=' , '!= ' (sau '<>')
- de concatenare a sirurilor '||'
- de atribuire: ':='

Exemple:

```

-- Declaratii de variabile
salar NUMBER(6,2);
ore_lucrate NUMBER := 40;
salar_orar NUMBER := 22.50;
bonus NUMBER := 150;
tara VARCHAR2(128);
numarator NUMBER := 0;
gata BOOLEAN := FALSE;
id_valid BOOLEAN;

BEGIN
    salar := (ore_lucrate * salar_orar) + bonus; -- calcul salar
    tara := 'Suedia'; -- atrib. un literal de tip string

```

```
tara := UPPER('Canada'); -- idem, caracterele transf in majuscule
gata := (contor > 100); -- atrib. BOOLEAN, literalul FALSE
id_valid := TRUE; -- atrib. BOOLEAN

END;
/
```

Literali

Exemple:

```
-- Declaratii de variabile
number1 PLS_INTEGER := 32000; -- literal numeric, val întreagă
number2 NUMBER(8,3);

BEGIN
    number2 := 3.125346e3; -- literal numeric
    number2 := -8300.00; -- literal numeric
    number2 := -14; -- literal numeric
END;
```

```
char1 VARCHAR2(1) := 'x'; -- literal caracter
char2 VARCHAR2(1);

BEGIN
    char2 := '5'; -- literal caracter
END;
/
```

```
-- Declaratii de variabile
string1 VARCHAR2(1000);
string2 VARCHAR2(32767);

BEGIN
    string1 := '555-111-2323';
-- daca un sir contine apostrof, acesta se dubleaza
    string2 := 'Here"s an example of two single quotation marks.';
END;
/
```

```

-- Declaratii de variabile
gata BOOLEAN := TRUE; -- BOOLEAN literal
complet BOOLEAN;
true_or_false BOOLEAN;

BEGIN
    gata := FALSE; -- literal BOOLEAN
    complet := NULL; -- literal BOOLEAN (valoare nedefinita)
    true_or_false := (3 = 4);
    true_or_false := (3 < 4);

END;
/

```



```

-- Declaratii de variabile
date1 DATE := '11-AUG-2005'; -- literal DATE
time1 TIMESTAMP;

BEGIN
    time1 := '11-AUG-2005 11:01:01 PM'; -- literal TIMESTAMP
END;
/

```

Atribuirea folosind SELECT ... INTO

Dacă valoarea unei variabile se determină în funcție de valori dintr-o linie a unui tabel al bazei de date, PL/SQL permite atribuirea valorii acesteia folosind o construcție

SELECT expresie INTO variabilă FROM tabel WHERE condiție_selectare_linie

Expresia folosită poate folosi variabile, literali și valori din linia corespunzătoare din tabel.

Exemplu:

```

-- Declaratii de variabile
procent_bonus CONSTANT NUMBER(2,3) := 0.05;

```

```

bonus NUMBER(8,2);
ang_id NUMBER(6) := 120; -- atribuie o val. pt testare

BEGIN
/* preia salar din tabelul angajati, calculeaza bonusul și atribuie
rezultatul -> variabila bonus */
    SELECT salar * procent_bonus INTO bonus FROM angajati
        WHERE angajat_id = ang_id;
-- listeaza codul angajat_id, bonusul si procent_bonus
    DBMS_OUTPUT.PUT_LINE ( 'Angajat: ' || TO_CHAR(ang_id)
    || ' Bonus: ' || TO_CHAR(bonus) || ' Procent bonus: ' ||
    TO_CHAR(procent_bonus*100));
END;
/

```

În aceeași frază SELECT pot fi atribuite valori mai multor variabile. Exemplu:

```

SELECT Nume, Prenume, salar*procent_bonus INTO m_nume, m_prenume, m_salar
FROM angajati WHERE angajat_id = ang_id;

```

Funcții uzuale

PL/SQL cuprinde un număr mare de funcții. În tabelul de mai jos sunt incluse cele mai des folosite.

Funcția	Descriere
Funcții pentru prelucrarea șirurilor	
upper(s), lower(s)	convertește s în majuscule/minuscule
ltrim(s), rtrim(s)	înlătura spațiile de la stânga / dreapta
substr(s, start, lungime)	returnează un subșir definit prin poz. start și lungime
length(s)	returnează lungimea șirului s
Funcții pentru prelucrarea datei calendaristice	
sysdate	data din sistem
to_date(data, format)	returneaza o dată formatată conform formatului Ex. : to_date('31-12-2007', 'dd-mm-yyyy')
to_char(data, format)	Convertește o dată calendaristică în șir, conform formatului <i>format</i> . Exemplu:

to_date(d, 'dd-mm-yyyy')

Funcții pentru date numerice

round(x)	rotunjește x
mod(n, p)	returnează restul împărțirii întregi n/p
abs(x)	returnează val. absolută
dbms_random.random()	generează un număr aleator întreg

Conversii de tip

to_char(n)	convertește n în sir de caractere
to_number(s)	convertește sirul s în număr

Alte funcții

user	returnează numele utilizatorului serverului Oracle
------	--

Limbajul PL/SQL

(continuare)

Instrucțiuni

Instrucțiunea de decizie - IF

În limbajul PL/SQL instrucțiunea IF poate prezenta trei firme.

a. IF-THEN

Exemplu:

```
-- Declaratii de variabile
vanzari NUMBER(8,2) := 10100;
cota NUMBER(8,2) := 10000;
bonus NUMBER(6,2);
ang_id NUMBER(6) := 120;

BEGIN
    IF vanzari > (cota + 200) THEN
        bonus := (vanzari - cota)/4;
        UPDATE angajati SET salar = salar + bonus WHERE
            angajat_id = ang_id;
    END IF;
END;
/
```

Astfel scris IF servește la condiționarea unei acțiuni. Acțiunea este declanșată dacă expresia logică are valoarea *adevărat*. Secvența de cod care descrie acțiunea este plasată între THEN și END IF.

b. IF-THEN-ELSE

Exemplu:

```

-- Declaratii de variabile
vanzari NUMBER(8,2) := 12100;
cota NUMBER(8,2) := 10000;
bonus NUMBER(6,2);
ang_id NUMBER(6) := 120;

BEGIN
    IF vanzari > (cota + 200) THEN
        bonus := (vanzari - cota)/4;
    ELSE
        bonus := 50;
    END IF;
    UPDATE angajati SET salar = salar + bonus WHERE
        angajat_id = ang_id;
END;
/

```

În secvența anterioară prezența clauzei ELSE permite definirea a două blocuri de instrucțiuni, unul cuprins între THEN și ELSE și al doilea cuprins între ELSE și END IF. În funcție de valoarea de adevăr a expresiei logice va fi executat primul bloc sau al doilea.

c. IF-THEN-ELSIF

Uneori este necesară alegerea unei secvențe de cod din mai multe, fiecare fiind condiționată de câte o expresie logică. În acest caz se recomandă folosirea structurii IF-THEN-ELSIF.

Exemplu:

```

-- Declaratii de variabile
vanzari NUMBER(8,2) := 20000;
bonus NUMBER(6,2);
ang_id NUMBER(6) := 120;

BEGIN
    IF vanzari > 50000 THEN
        bonus := 1500;
    ELSIF vanzari > 35000 THEN

```

```

        bonus := 500;
    ELSE
        bonus := 100;
    END IF;
    UPDATE angajati SET salar = salar + bonus WHERE
        angajat_id = ang_id;
END;
/

```

Instrucțiunea CASE

Instrucțiunea CASE permite impunerea blocului care va fi executat în funcție de valoarea unei expresii. De cele mai multe ori expresia se reduce la o variabilă.

Exemplu:

```

-- Declaratii de variabile
grade CHAR(1);
BEGIN
    nivel := 'B';
    CASE nivel
        WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excelent');
        WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Foarte bun');
        WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Bun');
        WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Corect');
        WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Slab');
        ELSE DBMS_OUTPUT.PUT_LINE('Nu ma pot pronunta');
    END CASE;
END;
/

```

În exemplul prezentat valoarea variabilei *nivel* condiționează blocul care va fi executat. Valorile posibile sunt introduse prin clauze WHEN iar blocul de instrucțiuni care trebuie executat se scrie după THEN. Dacă valoarea variabilei *nivel* nu se regăsește printre valorile introduse prin WHEN, se va executa blocul introdus prin clauza ELSE, plasată ultima. Instrucțiunea se încheie cu END CASE.

Instrucțiuni de ciclare

În limbajul PL/SQL există trei instrucțiuni de coclare: *LOOP*, *WHILE-LOOP* și *FOR-LOOP*

a. LOOP

Un bloc *LOOP* - *END LOOP* permite definirea unui ciclu din care se ieșe prin executarea unei instrucțiuni *EXIT*. Instrucțiunea *EXIT* este de regulă plasată într-un *IF* a cărui condiție trebuie să ajungă să fie satisfăcută pentru a se evita rularea blocului *LOOP* în ciclu infinit.

Exemplu:

```
credit_rating NUMBER := 0;
BEGIN
    LOOP
        credit_rating := credit_rating + 1;
        IF credit_rating > 3 THEN
            EXIT; -- exit loop immediately
        END IF;
    END LOOP;
    -- după EXIT, executia se continuă de aici
    DBMS_OUTPUT.PUT_LINE ('Credit rating: ' || TO_CHAR(credit_rating));
    IF credit_rating > 3 THEN
        RETURN; -- în afara blocului LOOP se folosește RETURN, nu EXIT
    END IF;
    DBMS_OUTPUT.PUT_LINE ('Credit rating: ' || TO_CHAR(credit_rating));
END;
/
```

Dacă ieșirea din ciclu se poate realiza printr-un simplu *IF*, se poate înlocui *IF* prin instrucțiunea *EXIT WHEN conditie_iesire*.

Exemplu:

```
IF count > 100 THEN EXIT; ENDIF;
EXIT WHEN count > 100;
```

Exemplu:

```

LOOP
    credit_rating := credit_rating + 1;
    EXIT WHEN credit_rating > 3
END LOOP;

```

b. WHILE - LOOP

În PL/SQL pentru realizarea unei structuri repetitive condiționate anterior se utilizează instrucțiunea WHILE - LOOP. Sintaxa unei astfel de structuri repetitive este următoarea:

```

WHILE condiție LOOP
    bloc de instrucțiuni
END LOOP

```

c. FOR - LOOP

Ca și în alte limbaje, parcurgerea unui sir de valori se realizează folosind o instrucțiune FOR - LOOP. Exemplu de utilizare:

```

-- Declaratii de variabile
p NUMBER := 0;
BEGIN
    FOR k IN 1..500 LOOP -- calcul pi ca suma a 500 termeni
        p := p + ( ( (-1) ** (k + 1) ) / ((2 * k) - 1) );
    END LOOP;
    p := 4 * p;
    DBMS_OUTPUT.PUT_LINE( 'pi este aproximativ : ' || p );
END;
/

```

În exemplul dat variabila *i* va lua valori în intervalul de valori întregi introdus prin IN. Valorile sunt de regulă consecutive și crescătoare. Dacă sirul valorilor introduse prin IN trebuie parcurs în ordine inversă se va folosi clauza REVERSE, ca în exemplul următor:

```

BEGIN
    FOR i IN REVERSE 1..5 LOOP
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));
    END LOOP;
END;
/

```

Se interzice modificarea variabilei folosite la controlul ciclului în interiorul acestuia.

Un exemplu tipic de utilizare a ciclului FOR în domeniul bazelor de date este cel în care se actualizează într-un ciclu înregistrările dintr-un tabel.

Exemple:

```

-- Declaratii de variabile
CURSOR ang_cursor IS SELECT * FROM angajati;
REC angajati%ROWTYPE; -- var. structurata de tip record

BEGIN
    FOR REC IN ang_cursor LOOP
        UPDATE angajati
            SET salar = ...
        WHERE ang_id = REC.ang_id;
    END LOOP;
END;
/

```

```

CREATE OR REPLACE PROCEDURE Afisez
AS
    CURSOR c1 IS SELECT * FROM edituri order by nume;
    REC edituri%ROWTYPE; -- var. structurata de tip record

BEGIN
    FOR REC IN c1 LOOP
        DBMS_OUTPUT.PUT_LINE (rec.nume);
    END LOOP;
END;

```

/

```

BEGIN
  OPEN c2;
  LOOP
    FETCH c2 INTO REC;           -- preia linia curenta in REC
    EXIT WHEN c2%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( RPAD(REC.prenume, 25, ' ') ||
                          REC.id_meserie );
  END LOOP;
  CLOSE c2;

```

Notă Funcția *RPAD()* construiește un sir de caractere având lungimea dată de al doilea argument. Sirul se obține prin completarea la dreapta a sirului dat ca prim argument cu sirul dat ca al treilea argument.

În exemplul următor cursorul c1 conține doar două valori din tabelul angajați.. Instrucțiunea *FETCH* va prelua cele două valori în două variabile locale.

```

jobid angajati.job_id%TYPE; -- variabila pentru id_functie
pren angajati.prenume%TYPE; -- variabila pentru prenume
CURSOR c1 IS SELECT prenume, id_functie FROM angajati
WHERE id_functie LIKE '%ZIDAR';
BEGIN
  OPEN c1; -- open the cursor before fetching
  LOOP
    FETCH c1 INTO pren, jobid; -- preiau 2 coloane in variabile
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( RPAD(pren, 25, ' ') || jobid );
  END LOOP;
  CLOSE c1;

```

Definirea unei proceduri

De regulă secvențele de cod PL/SQL sunt conținute în proceduri sau funcții. Blocurile anonime pot fi folosite eventual pentru testarea corectitudinii unei secvențe de cod, dar după încheierea testării ele vor fi transformate în proceduri sau funcții memorate în baza de date.

Codul poate fi scris în fereastra destinată scrierii comenzi SQL (*SQL Command*), în *Script Editor* sau folosind *Object Browser / Create / Procedure*. Indiferent de soluția adoptată, rezultatul va fi o procedură stocată în *Oracle XE* și accesibilă folosind meniul *Object Browser*.

Exemplul 1. Crearea unei proceduri folosind *SQL Command*:

The screenshot shows the Oracle SQL Command window. At the top, there is a toolbar with 'Autocommit' checked, a 'Display' dropdown set to 10, a 'Save' button, and a 'Run' button which is highlighted with a blue oval. Below the toolbar is a code editor containing the following PL/SQL code:

```
create or replace procedure "SALUT"
(num IN VARCHAR2)
is
begin
dbms_output.put_line('Salut, ' || num || '!');
end;
/
```

Below the code editor, there is a navigation bar with links: 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History'. Underneath the navigation bar, the message 'Procedure created.' is displayed, followed by '0.53 seconds'.

Declararea unei proceduri folosind fereastra SQL Command sau SQL*Plus începe cu secvența :

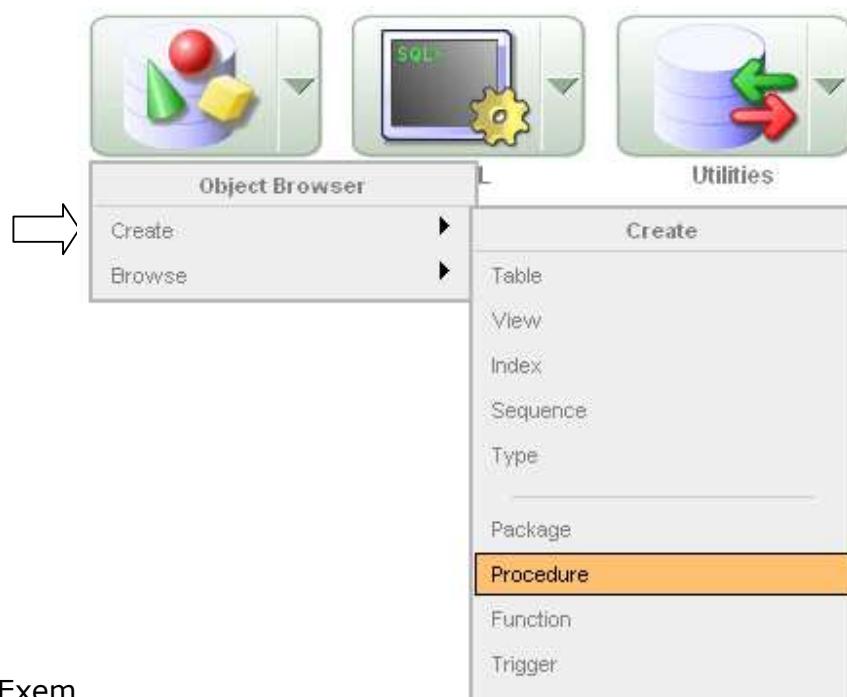
CREATE OR REPLACE PROCEDURE nume ...

Pentru verificarea creerii procedurii se selectează *Object Browser / Browse / Procedures* și în fereastra care se afișează se selectează procedura (*SALUT*).

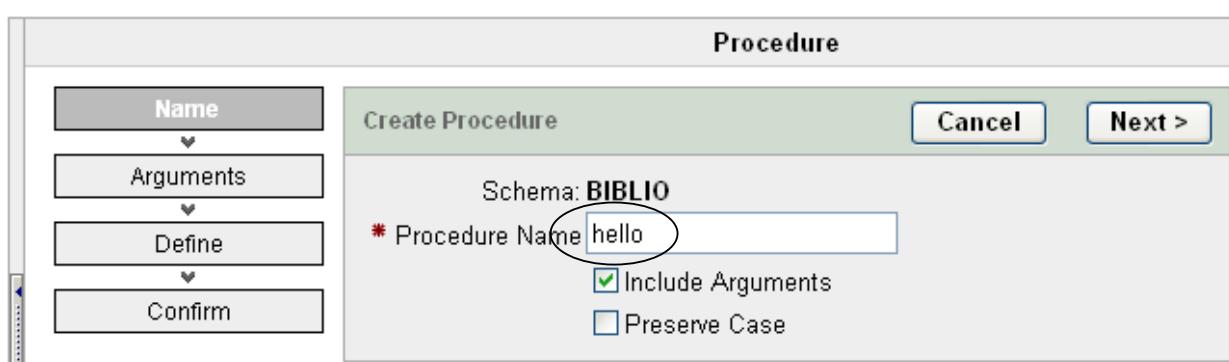
The screenshot shows the Oracle Object Browser window. On the left, there is a tree view under the 'Procedures' tab, showing nodes for 'HELLO', 'SALUT' (which is selected and highlighted in green), and 'TEST'. On the right, there is a details pane with tabs: 'Code', 'Dependencies', 'Errors', and 'Grants'. The 'Code' tab is selected. Below the tabs are buttons: 'Compile', 'Download', 'Drop', 'Edit', 'Undo', 'Redo', and 'Find'. The code area displays the same PL/SQL code as in the previous screenshot:

```
1 create or replace procedure "SALUT"
2 (nume IN VARCHAR2)
3 is
4 begin
5 dbms_output.put_line('Salut, ' || nume || '!');
6 end;
```

Exemplul 2. Crearea unei proceduri folosind *Object Browser / Create / Procedure*:



Exem.



În pasul următor se definesc parametrii de intrare și de ieșire ai procedurii, dacă este cazul.

Argument Name	In/Out	Argument Type	Default	Move
nume	IN	VARCHAR2		▼
		VARCHAR2		▼ ▲
		VARCHAR2		▼ ▲

În continuare se introduce corpul procedurii:

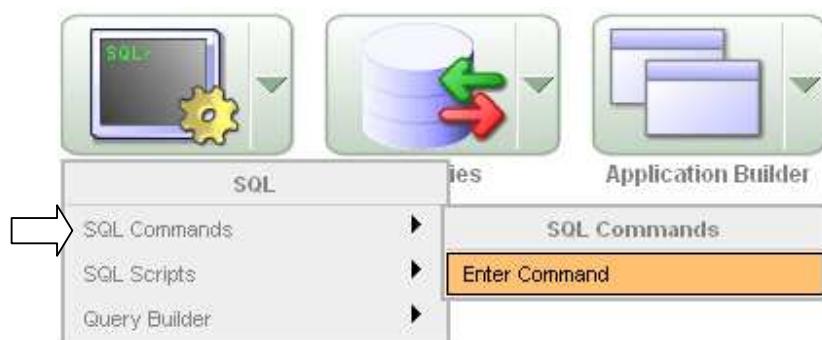
```
* Procedure Body:
dbms_output.put_line('Salut, ' || nume || '!');
```

Rezultat:

```
1 create or replace procedure "HELLO"
2 (nume IN VARCHAR2)
3 is
4 begin
5 dbms_output.put_line('Salut, ' || nume || '!');
6 end;
```

Astfel generată, procedurii îi lipsește secțiunea destinată declarării variabilelor locale. Pentru adăugarea acestora se selectează *Edit* și se inserează declarațiile necesare.

Apelul procedurii se poate realiza din fereastra destinată introducerii de comenzi SQL (*SQL / SQL Commands*) :



```

begin
hello('Mihai');
end;
/

```

The screenshot shows the 'Enter Command' window of SQL*Plus. It displays the following PL/SQL code:

```

begin
hello('Mihai');
end;
/

```

Below the code, the 'Results' tab is selected, showing the output:

Salut, Mihai!

Din SQL*Plus apelul se poate face și folosind comanda SQL *CALL* :

```
SQL> set serveroutput on;
SQL> call hello('Mihai');
Salut, Mihai!
Call completed.
SQL>
```

Parametrii procedurilor PL/SQL

Procedurile pot avea un număr de parametri de intrare (IN), parametri de ieșire (OUT) sau de intrare / ieșire (IN OUT).

Exemple:

```
PROCEDURE maj  ( v1 IN VARCHAR2, v2 IN VARCHAR2) IS ...
PROCEDURE majusc ( v1 IN VARCHAR2, v2 OUT VARCHAR2) IS ...
PROCEDURE majuscule ( v1 IN OUT VARCHAR2, v2 IN OUT VARCHAR2) AS ...
```

Dacă un parametru este declarat de intrare (IN) modificarea sa în subprogram nu afectează valoarea eventualei variabile folosite ca parametru efectiv la apelarea funcției.

La apelul unei proceduri având parametri de tip OUT sau IN OUT, pe pozițiile corespunzătoare acestora din lista de parametri efectivi trebuie utilizate variabile de același tip cu cel al parametrilor procedurii.

Un bloc anonim sau o procedură poate avea în zona de declarații o declarație de procedură. Exemplu :

```
DECLARE -- declaratii de variabile si subprograme
    nume VARCHAR2(20) := 'ionescu';
    prenume VARCHAR2(25) := 'marin';

    PROCEDURE nume_maj ( v1 IN OUT VARCHAR2, v2 IN OUT VARCHAR2) AS
    BEGIN
        v1 := UPPER(v1); -- trec sirurile in majuscule
        v2 := UPPER(v2);
    END;

    BEGIN
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume); -- afisez val. initiale
        nume_maj (nume, prenume); -- apelez procedura cu parameteri
        DBMS_OUTPUT.PUT_LINE(nume || ' ' || prenume); -- afisez noile valori
    END;
/

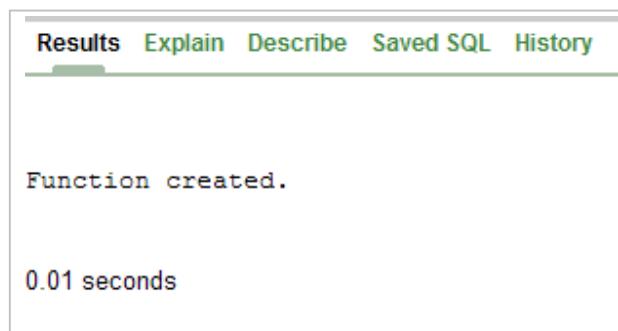
```

Deoarece procedura *nume_maj* este cuprinsă într-un bloc anonim, executarea blocului nu va determina memorarea acesteia în baza de date.

Definirea unei funcții

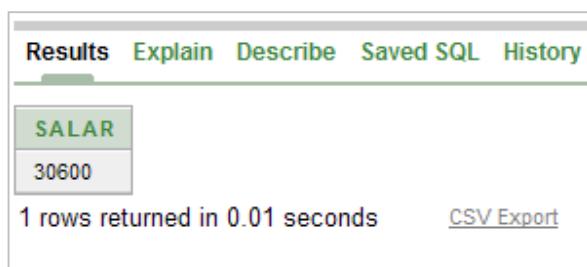
Funcțiile sunt subprograme care returnează o valoare. Ca și în cazul procedurilor, o funcție poate fi scrisă în fereastra destinată scrierii comenzi SQL (*SQL Command*), în *Script Editor* sau folosind interfața afișată prin selectarea opțiunii *Object Browser / Create / Function*. Exemplu:

```
CREATE OR REPLACE FUNCTION salar
RETURN NUMBER AS
  x  NUMBER;
BEGIN
  x := 30600;
  RETURN x;
END;
/
```



Funcția *salar* poate fi apelată în mai multe moduri, cel mai simplu fiind în fraze *select*:

a. `select salar from dual|`



`SELECT * FROM employees WHERE salary < (salar);`

b.

Ca și procedura, o funcție poate avea parametri de intrare. Lista acestora este

```

CREATE OR REPLACE FUNCTION nume_prenume ( nume IN VARCHAR2,
                                         prenume IN VARCHAR2, casatorit IN VARCHAR2)
RETURN VARCHAR2 AS
    nup VARCHAR2(45); -- variabila locala
BEGIN
    -- Se construiește un sir cuprinzând numele și prenumele
    nup := upper(nume) || ' ' || prenume;
    if length(casatorit) > 0 THEN -- dacă e casatorit, adaug casatorit
        nup := nup || ' ' || casatorit;
    END IF;
    RETURN nup; -- returnează valoarea lui nup
END;
/

```

The screenshot shows a SQL command window with the following details:

- Header:** Home > SQL > SQL Commands
- Autocommit:** checked
- Display:** 10
- Buttons:** Save, Run
- SQL Statement:** select nume_prenume('Dragan', 'Mircea', '') from dual
- Results Tab:** Active (highlighted in green)
- Result Data:** DRAGAN Mircea
- Timing:** 1 rows returned in 0.01 seconds
- Export:** CSV Export

Oracle JDBC

Generalități

Pentru scrierea aplicațiilor care accesază serverul *Oracle XE* o soluție bună este limbajul Java. De altfel firma *Oracle Co.* pune la dispoziția utilizatorilor mediul *JDeveloper* care posedă multiple facilități în acest sens. În cele ce urmează se va folosi mediul de programare *Netbeans*.

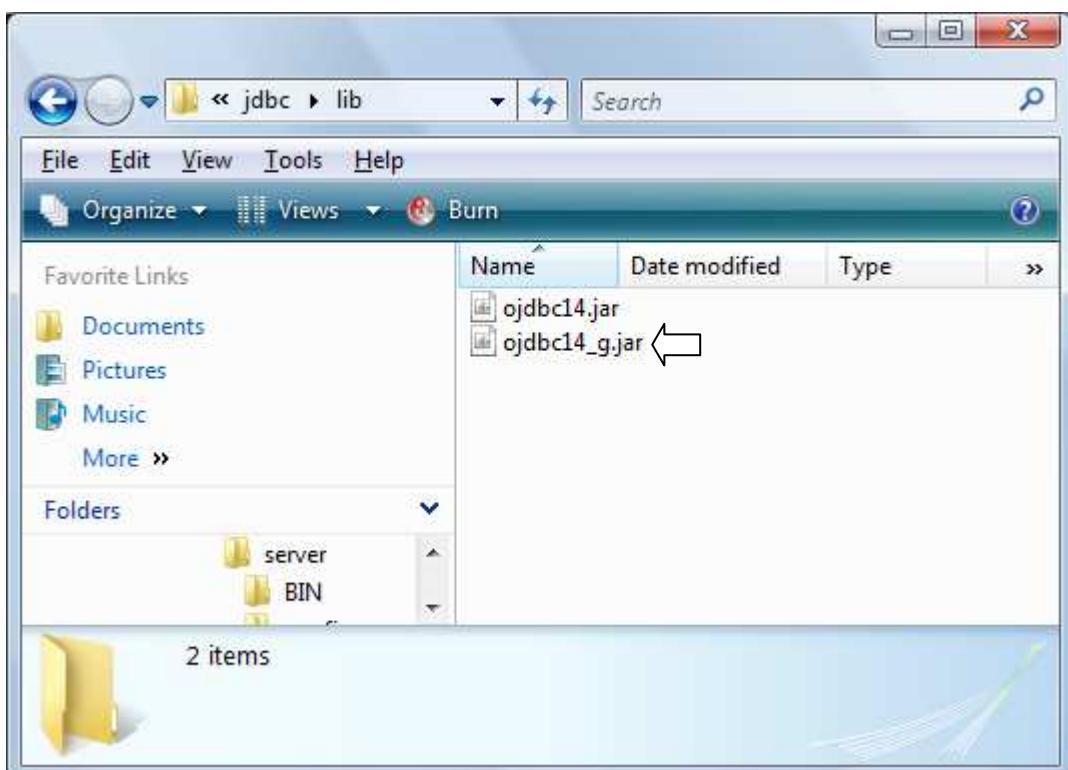
JDBC (Java DataBase Connectivity) este o colecție de clase care permite unei *aplicații client* scrisă în Java să acceseze un *server de baze de date relationale*. Folosind aceste clase o aplicație poate să se conecteze la serverul de baze de date, poate trimite comenzi SQL și poate prelucra rezultatele furnizate de server ca urmare a executării acțiunilor comandate. Spre deosebire de *ODBC (Open DataBase Connectivity)* – colecție de proceduri scrise în C și dependente de sistemul de operare), colecția de clase conținute în JDBC este scrisă în Java fiind astfel realizată independentă față de sistemul de operare.

Fișierele sursă care folosesc clase din JDBC trebuie să includă pachetul *java.sql*.

Pentru fiecare server de baze de date există implementări specifice ale JDBC. Astfel în cazul accesării serverului *Oracle XE* proiectului aplicației client trebuie să i se adauge una dintre arhivele *.jar* existente în directorul [ORACLE_HOME]\jdbc\lib. Dacă instalarea serverului Oracle XE s-a făcut în C:, atunci [ORACLE_HOME] este:

C:\oraclexe\app\oracle\product\10.2.0\server

În exemplele care vor fi prezentate în continuare a fost folosită arhiva *ojdbc14_g.jar*.



Principalele clase din JDBC

Pentru realizarea unei aplicații care accesează un server de baze de date JDBC definește o serie de clase, cele mai utilizate fiind *Connection*, *Statement*, *PreparedStatement* și *ResultSet*.

Notă: Este esențial ca după utilizarea lor obiectele JDBC să fie suprimate prin apelul metodei *close()* definită pentru toate clasele menționate.

Clasa Connection

Pentru a accesa serverul Oracle XE, clasa principală a aplicațiilor care vor fi realizate va conține un obiect din clasa *Connection*. Crearea sa va fi sistematic realizată folosind secvența de cod următoare:

```
try {
    OracleDataSource ods = new OracleDataSource();

    ods.setURL("jdbc:oracle:thin:biblio/bibpas@193.226.7.210:1521/XE");
    cnx = ods.getConnection(); // cnx apartine clasei Connection
} catch ( SQLException sqlException ) {
    System.out.println("Conectare imposibila.");
    System.exit( 1 );
}
```

Șirul de caractere subliniat conține *schema (biblio)*, parola (*bibpas*), IP-ul serverului pe care rulează *Oracle XE* (193.226.7.210) și portul pe care se va realiza conectarea (1521).

Înaintea încheierii aplicației este necesară închiderea conexiunii prin apelul metodei *close()* a acestieia:

```
cnx.close();
```

Clasa Statement

Pentru a trimite comenzi SQL individuale serverului de baze de date este necesară crearea unui obiect din clasa *Statement*. Crearea obiectului se realizează prin apelul metodei *createStatement()* a clasei *Connection* :

```
Statement stmt1 = cnx.createStatement() ;
sau
Statement stmt2 = cnx.createStatement(tip, acces) ;
```

Dacă va fi folosit pentru trimiterea de comenzi *SQL select*, obiectul ***stmt1*** va permite crearea unor mulțimi de selecție care vor putea fi parcuse doar dinspre primul element spre ultimul, nu și în sens invers. Dacă se dorește crearea ulterioară

a unor multimi de selecție care pot fi parcuse în ambele sensuri se va utiliza varianta metodei `createStatement()` cu doi parametri.

Primul parametru, *tip*, va avea una dintre valorile următoare:

- `ResultSet.TYPE_FORWARD_ONLY`, care indică faptul că mulțimea de selecție creată ulterior va putea fi parcursă doar înainte,
- `ResultSet.TYPE_SCROLL_INSENSITIVE`, care permite realizarea unei multimi de selecție care va putea fi parcursă în ambele sensuri dar nu va reflecta modificările operate de alți utilizatori pe tabelele folosite la interogare.

Al doilea parametru, *acces*, poate lua valorile:

- `ResultSet.CONCUR_READ_ONLY` semnificând faptul că mulțimile de selecție care vor fi realizate nu vor putea fi folosite la actualizarea datelor din tabele (nu se vor putea folosi la trimiterea de comenzi *update*) sau
- `ResultSet.CONCUR_UPDATABLE` care indică faptul că mulțimile de selecție realizate ulterior folosind *stmt2* vor putea servi la actualizarea datelor din tabelele folosite la interogare (corectare, inserare de noi linii).

După crearea unui obiect din clasa *Statement* acesta va putea fi folosit pentru a apela metoda `executeQuery()` (pentru a trimite serverului comenzi *SELECT*) sau metoda `executeUpdate()` (pentru a trimite serverului comenzi *CREATE*, *UPDATE*, *INSERT* sau *DELETE*). Metoda `executeQuery()` returnează o mulțime de selecție în timp ce metoda `executeUpdate()` returnează un număr întreg reprezentând numărul înregistrărilor afectate de comanda SQL dată.

După încheierea folosirii sale, obiectul *Statement* trebuie suprimat prin apelul metodei `close()`.

```
stmt.close();
```

Clasa *ResultSet*

Un obiect din clasa *ResultSet* conține o mulțime de selecție furnizată de serverul de baze de date ca urmare a executării unei comenzi de interogare (*SELECT*).

Pentru parcurgerea liniilor mulțimii de selecție un *ResultSet* integrează un cursor care indică linia curentă a mulțimii. Inițial, după executarea interogării, cursorul este poziționat înaintea primei linii. Pentru a avansa pe o nouă linie se folosește metoda `next()`. Deoarece metoda `next()` returnează *null* la terminarea liniilor mulțimii de selecție, aceasta permite scrierea ușoară a unui ciclu de parcurgere a liniilor mulțimii de selecție.

```
try {
    Statement stmt = cnx.createStatement();
    ResultSet rs = stmt.executeQuery( "SELECT * FROM Edituri order by nume"
);
    while ( rs.next() ) {
        System.out.println( rs.getString("nume") );
    }
}
```

```

stmt.close();
} catch (SQLException ex) {}

```

Un obiect din clasa *ResultSet* este inițializat prin executarea metodei *executeQuery()* a clasei *Statement*. Dacă crearea s-a făcut ca mai sus, mulțimea de selecție poate fi parcursă doar înainte, apelând metoda *.next()*.

Dacă mulțimea de selecție trebuie să fie parcursă în ambele sensuri se va folosi varianta cu parametri a metodei *createStatement()*:

```

Statement stmt =
    cnx.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                       ResultSet.CONCUR_READ_ONLY) ;

```

După crearea unei mulțimi de selecție care poate fi parcursă în ambele sensuri, aceasta va putea fi parcursă folosind metodele:

- *first()* pentru trecerea pe prima linie,
- *last()* pentru trecere pe ultima linie,
- *next()* pentru trecere pe următoarea linie sau
- *previous()* pentru trecerea pe linia precedentă
- *absolute(nr)* pentru plasarea cursorului pe linia *nr* a mulțimii de selecție.

Valoarea unui câmp al liniei curente din mulțimea de selecție se obține apelând una dintre metodele JDBC *get*()* din tabelul de mai jos. O metodă *get*()* poate avea ca argument fie un număr întreg care specifică a cătă valoare trebuie extrasă, fie un sir de caractere conținând numele câmpului. La specificarea numelui câmpului nu contează cu ce caractere este scris - litere nici sau majuscule.

Preluarea valorilor câmpurilor liniei curente din mulțimea de selecție se realizează apelând metoda *get*()* adecvată, conform tabelului prezentat în continuare.

Metode JDBC <i>get*()</i>	Tip date SQL	Tip Java
getBit()	BIT	boolean
getByte()	TINYINT	byte
getShort()	SMALLINT	short
getInt()	INTEGER	int
getLong()	BIGINT	long
getFloat()	REAL	float
getDouble()	DOUBLE	double
getString()	CHAR	String

Metode JDBC <i>get*</i> ()	Tip datea SQL	Tip Java
getString()	VARCHAR	String
getString()	LONGVARCHAR	String
getDate()	DATE	java.sql.Date
getTimeStamp()	TIME	java.sql.Date
getObject()	BLOB	Object

Observație: La scrierea metodelor *get**() având argument numeric se va ține cont de faptul că numerotarea câmpurilor liniei curente din mulțimea de selecție începe cu valoarea 1. Dacă tabelul edituri are două câmpuri, *Cod_edit* și *Nume*, pentru exemplul dat același rezultat se putea obține deci scriind :

```
System.out.println( rs.getString(2) );
```

Column Name	Data Type	Nullable	Default	Primary Key
COD_EDIT	NUMBER(5,0)	No	-	1
NUME	VARCHAR2(300)	No	-	-
				1 - 2

O mulțime de selecție poate servi la actualizarea datelor din tabele folosite la crearea ei. Pentru aceasta obiectul de tip *Statement* folosit la trimiterea comenzi *select* trebuie declarat astfel:

```
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
```

Corectarea datelor din tabele se realizează ca în exemplul următor:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Edituri order by nume
");
rs.absolute(5); // mută cursorul pe a linia 5 din rs
rs.updateString("NUME", "Teora"); // corecteaza câmpul NUME
rs.updateRow(); // se trimit sursei de date linia 5
```

Pentru inserarea unei noi linii folosind o mulțime de selecție se va folosi o linie creată în acest scop prin apelul metodei *moveToInsertRow()*. În această linie pot fi apoi adăugate valorile dorite folosind metode *update*()* (*updateInt()*, *updateString()* etc). Exemplu:

```
rs.moveToInsertRow(); // mută cursorul pe linia suplimentară creată
rs.updateInt(1, "23"); // impune valoarea pt. primul câmp
rs.updateString(2,"Casa cartii"); // idem, câmpul 2
rs.insertRow();
rs.moveToCurrentRow();
```

Un *ResultSet* este închis prin apelul metodei *close()* sau automat, la închiderea obiectului *Statement* folosit la crearea sa respectiv la reconstruire (reexecutarea interogării).

Clasa PreparedStatement

Deoarece trimiterea efectivă a unei comenzi SQL folosind un obiect din clasa *Statement* presupune derularea în prealabil a unui proces de analiză sintactică și de construire a unei comenzi acceptate de serverul de baze de date, în cazul trimiterii repetate a aceleiași comenzi SQL se poate câștiga timp dacă procesul de pregătire menționat se derulează o singură dată pentru tot setul de comenzi. Pentru a realiza acest lucru este necesar însă ca în locul obiectului din clasa *Statement* să se declare un obiect din clasa înrudită *PreparedStatement*. Exemplu:

```
PreparedStatement ps = cnx.prepareStatement("INSERT INTO Edituri
VALUES(?,?)");
ps.setInt(1, 12);
ps.setString(2, "Minerva");
ps.executeUpdate();
ps.setInt(1, 13);
ps.setString(2, "Orizonturi");
ps.executeUpdate();
ps.setInt(1, 14);
ps.setString(2, "Dacia");
ps.executeUpdate();

ps.close;
```

Observație: Sirul de caractere care conține comanda care urmează să fie executată repetat poate conține caractere '?' care indică pozițiile pe care vor fi inserate valori noi înaintea reapelării metodei *executeUpdate()* (sau *executeQuery()*, după caz). Pentru inserare se apelează una dintre metodele *set**() din tabelul următor, în funcție de tipul valorii inserate.

Tip dată Oracle	metodă <i>set*</i> ()
CHAR	<i>setString()</i>
VARCHAR2	<i>setString()</i>
NUMBER	<i>setBigDecimal()</i>
	<i>setBoolean()</i>
	<i>setByte()</i>
	<i>setShort()</i>
	<i>setInt()</i>
	<i>setLong()</i>
	<i>setFloat()</i>
	<i>setDouble()</i>
INTEGER	<i>setInt()</i>

FLOAT	setDouble()
CLOB	setClob()
BLOB	setBlob()
RAW	getBytes()
LONGRAW	getBytes()
	setDate()
DATE	setTime()
	setTimestamp()

Primul parametru al metodei `set*()` indică în câte dintre pozițiile marcate cu '?' se va face inserarea, numerotarea acestora începând cu '1'.

Clasa CallableStatement

Un obiect de tip `CallableStatement` permite lansarea în execuție a unei proceduri stocate (scrisă în PL SQL de exemplu).

Crearea unui obiect din clasa `CallableStatement` se realizează ca în exemplele următoare:

- a. Procedură stocată fără parametri:

```
CallableStatement cs = cnx.prepareCall("{call procst}");
```

- b. Procedură stocată având 2 parametri:

```
CallableStatement cs = cnx.prepareCall("{call procstoc(?, ?)}");
cs.setString(1,theuser);
cs.setString(2,password);
cs.executeQuery();
```

- c. Procedură având doi parametri de intrare, unul de ieșire și care returnează o mulțime de selecție

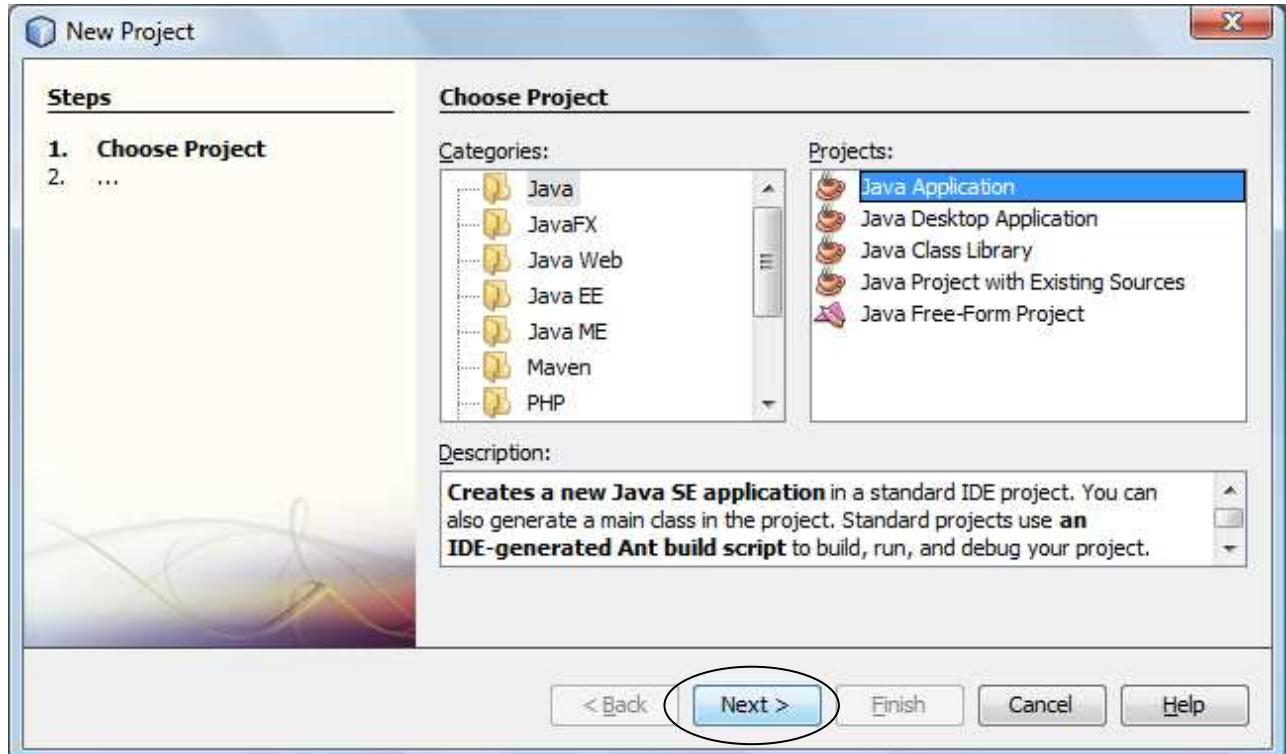
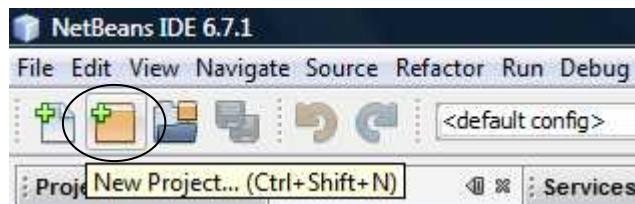
```
CallableStatement cs = cnx.prepareCall("{call proc3p(?, ?, ?)}");
cs.setString(1,theuser);
cs.setString(2,password);
cs.registerOutParameter(3,Types.DATE);

cs.executeQuery();
Date dataconect = cs.getDate(3);
```

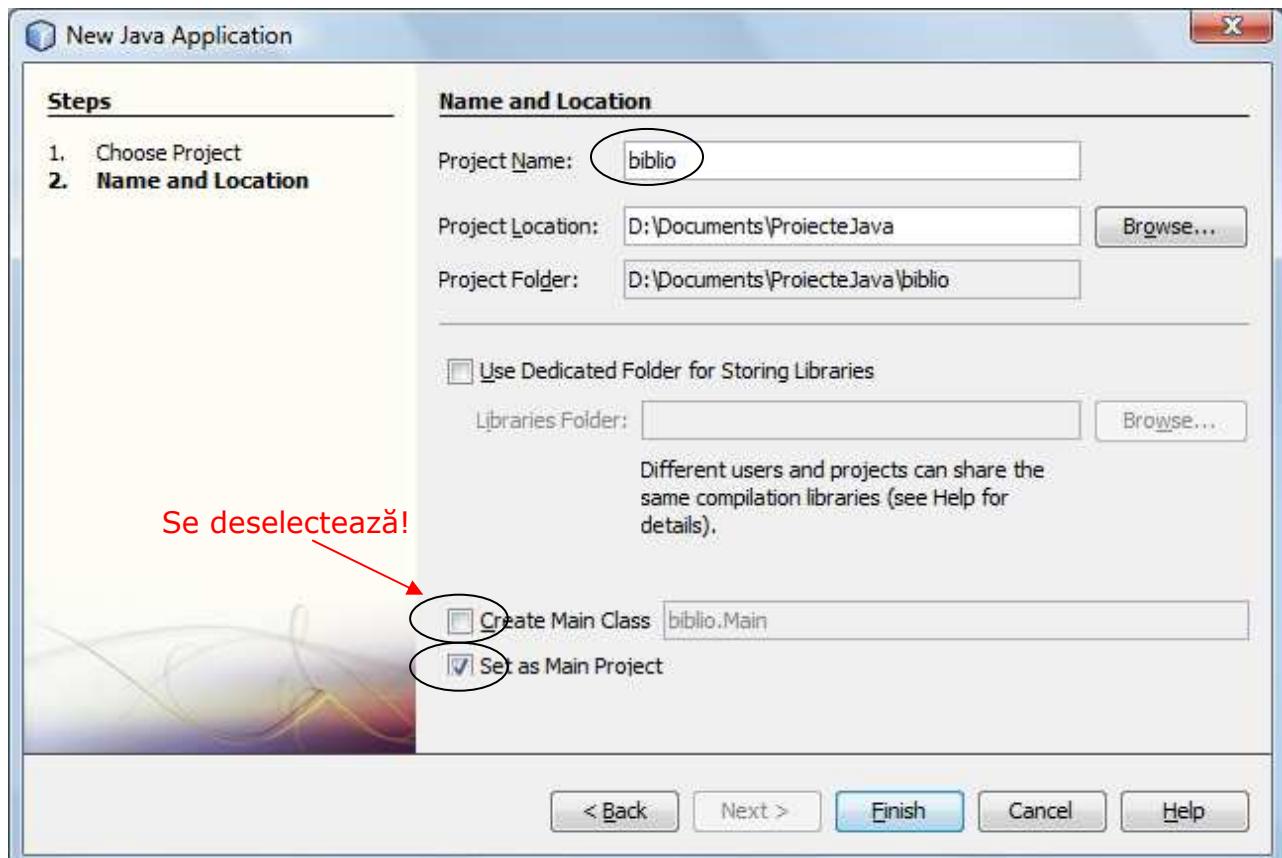
Realizarea interfeței aplicației

1. Se pornește mediul de programare (*NetBeans 6.7.1 IDE* sau o versiune ulterioară) ;

2. Se demarează un nou proiect (*File / New Project ...* sau se apasă butonul );

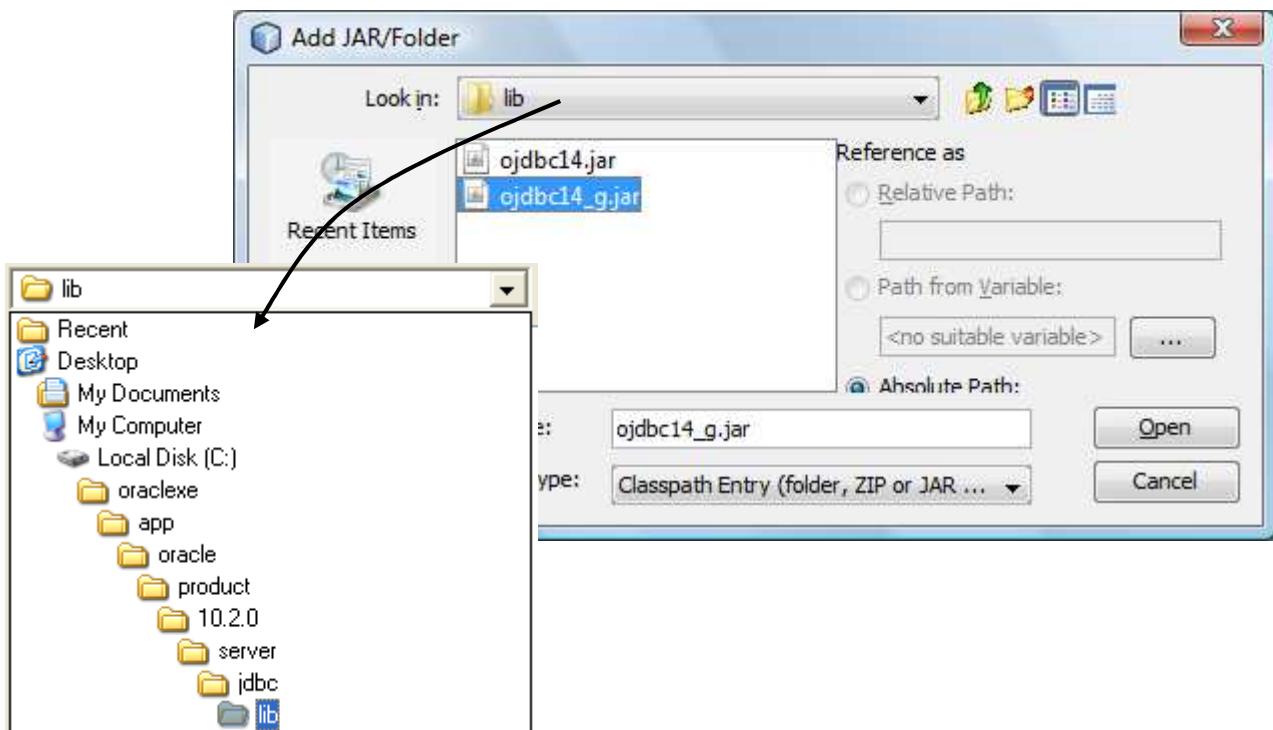
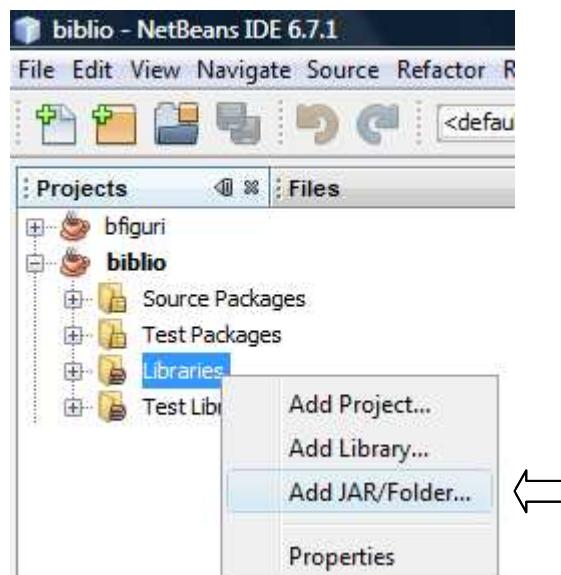


3. Se definește numele proiectului (Biblio), se deselectează caseta de validare *Create Main Class* și se apasă *Finish*. Deselectarea casetei *Create Main* va opri crearea unei clase principale.

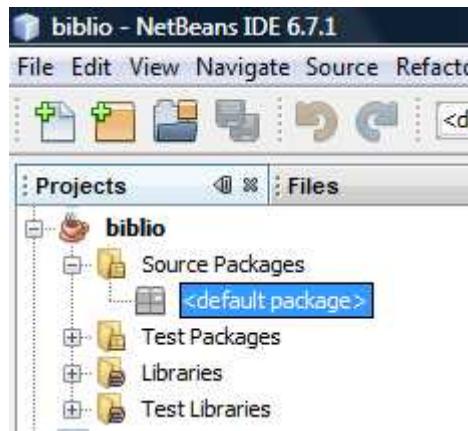


Aplicația care trebuie realizată având interfață grafică, este mai simplu să se creeze ulterior o clasă, derivată din clasa *Jframe*, pe care mediul de programare o va crea ca principală (va conține *main()*).

4. Se adaugă proiectului arhiva *ojdbc14_g.jar*:

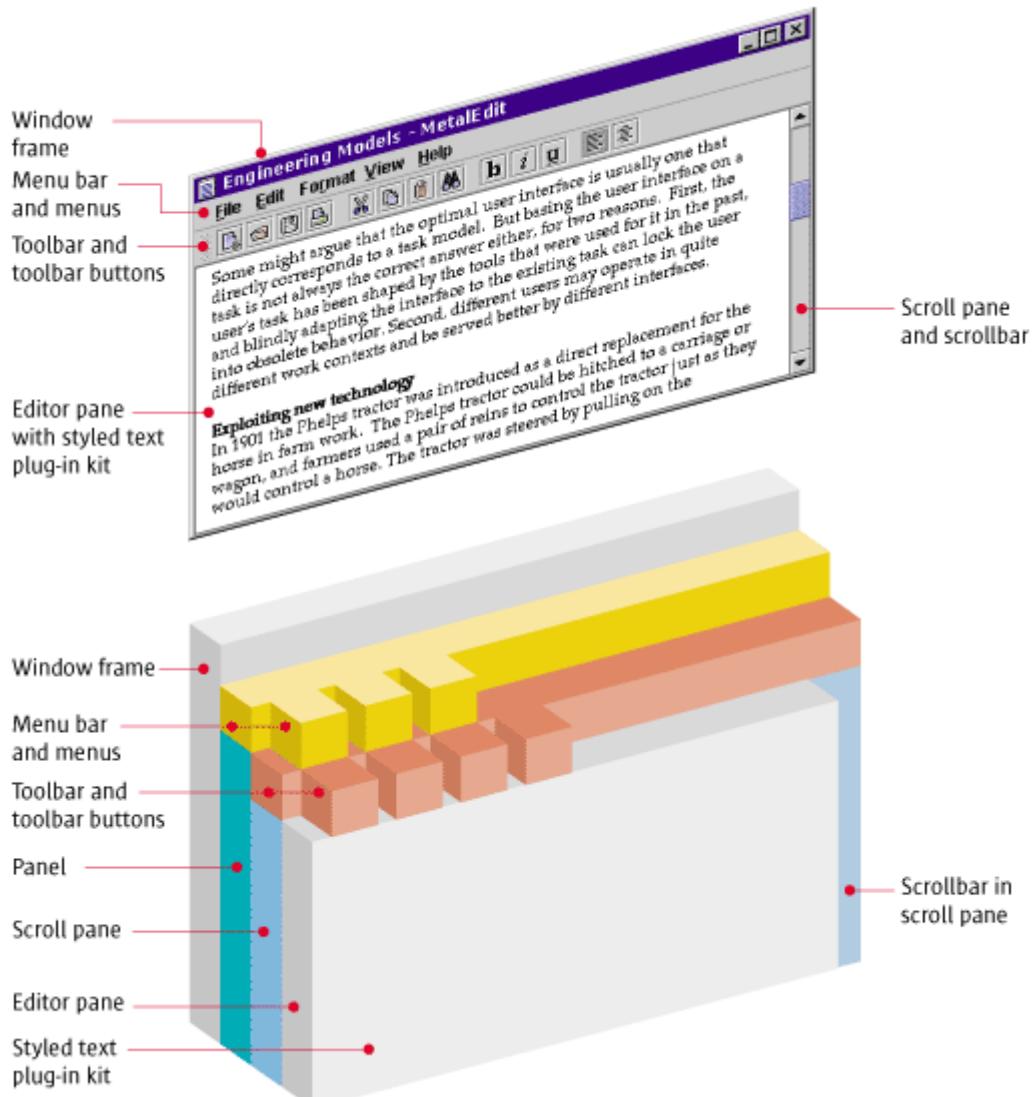


Pașii realizati asigură crearea pe disc a unei structuri de directoare care vor conține fișierele proiectului. Proiectul nu conține încă nici o clasă, în arborele acestuia figurând intrarea *Source Packages* cu o entitate, *<default package>*.



Crearea ferestrei principale

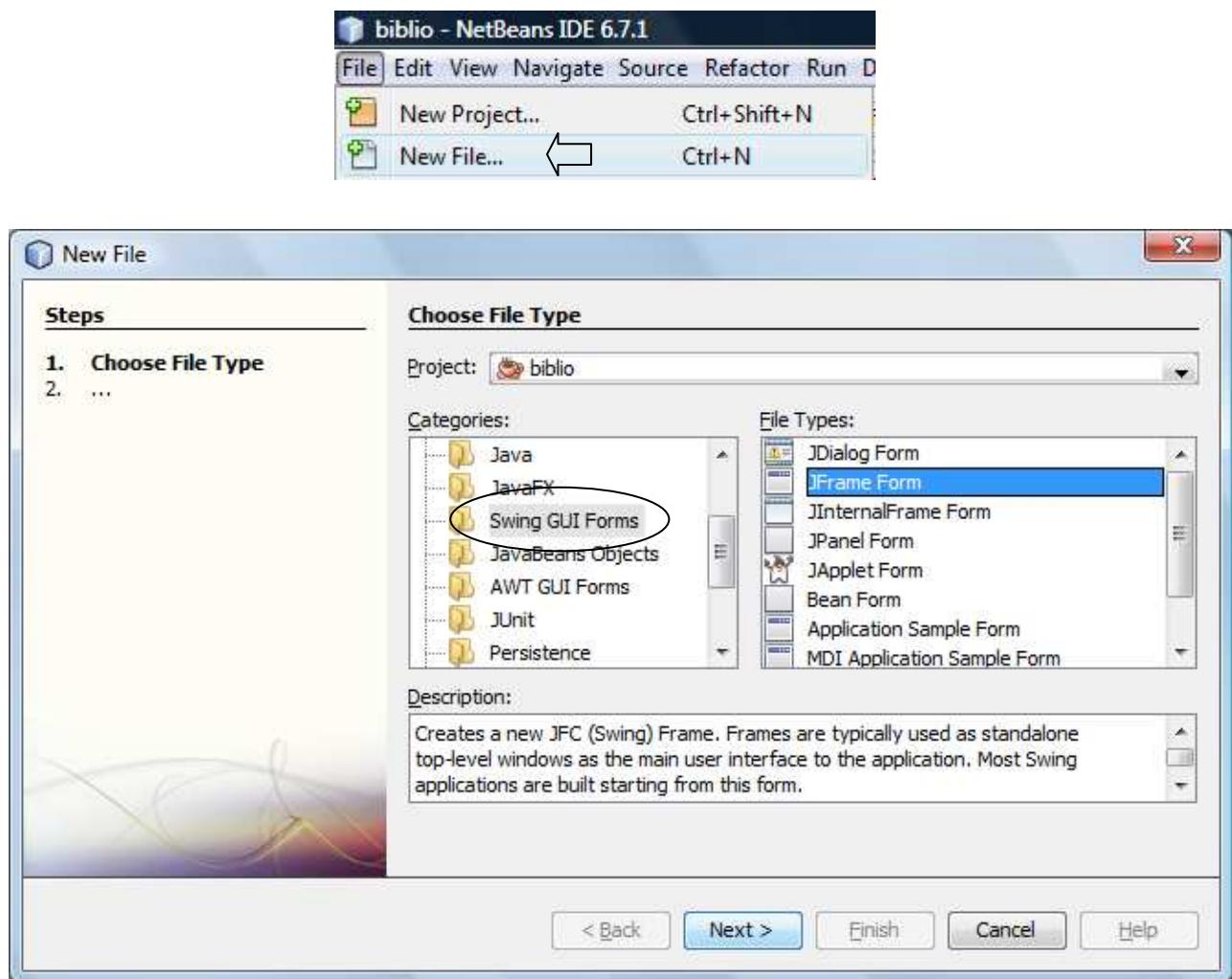
O aplicație care accesează un server de baze de date are caracteristicile unei aplicații Windows obișnuite, respectiv afișează o fereastră principală care conține o bară cu meniu derulant, o bară cu instrumente pentru comenzi rapide și o zonă grafică de afișare.



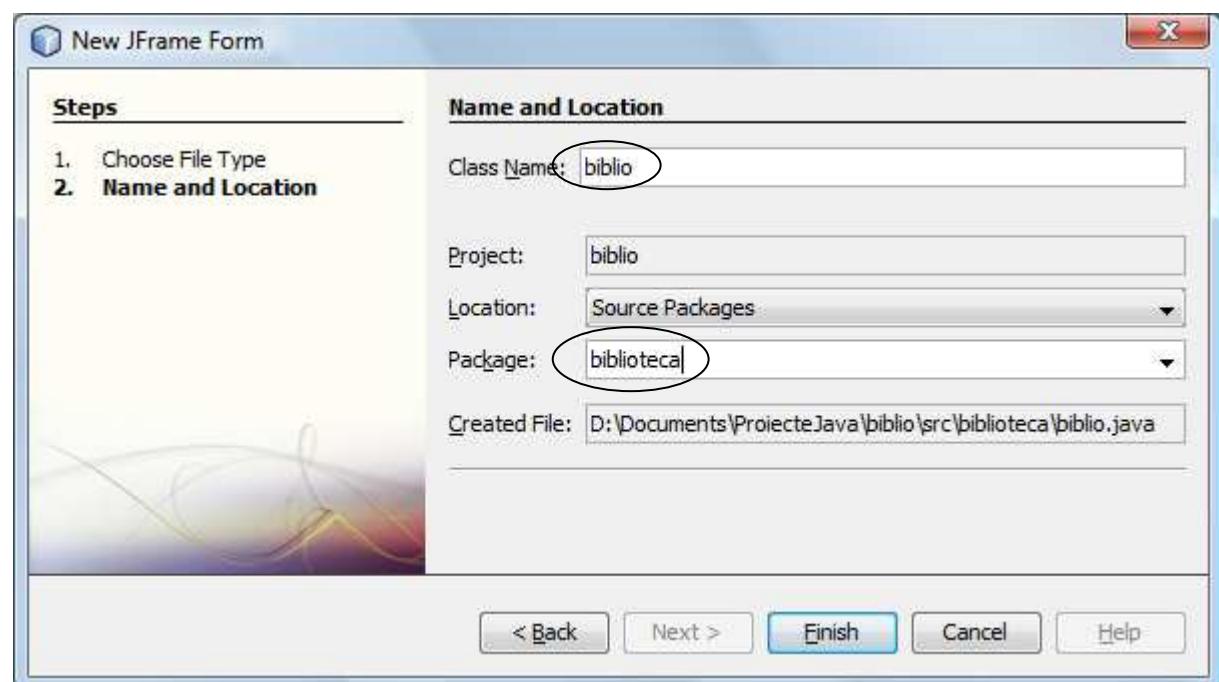
Crearea interfeței aplicației va începe cu crearea unei clasei principale. În aplicațiile scrise în Java aceasta poartă numele proiectului și conține metoda statică `main()`. Clasa principală **va fi declarată ca derivată din clasa JFrame**. O instanță a clasei principale, creată în cadrul metodei statice `main()`, va putea atunci servi drept container principal al aplicației (*Window frame* în figură).

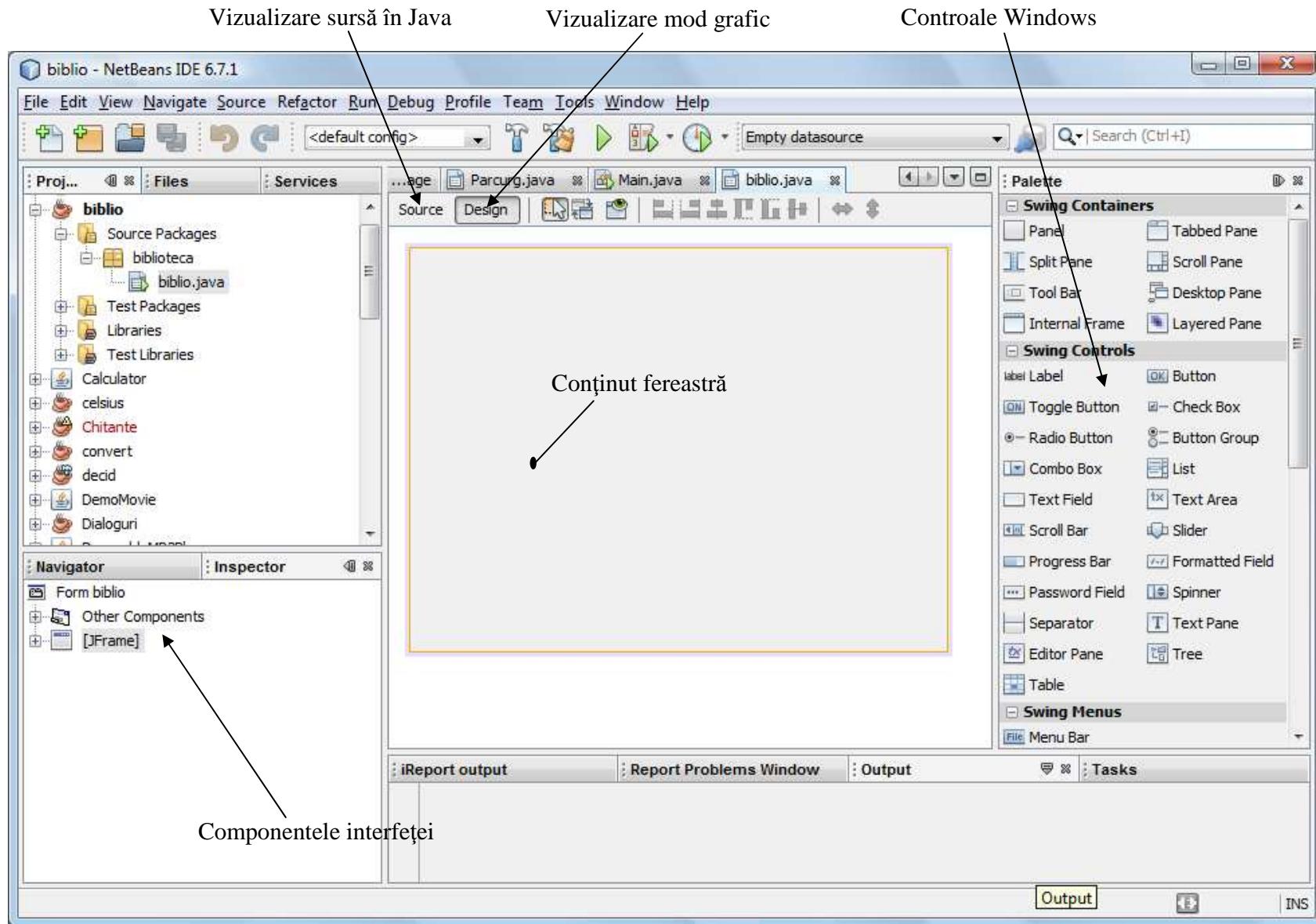
Pentru crearea acestei clase se procedează astfel:

- În meniul *File* se selectează *New File* :

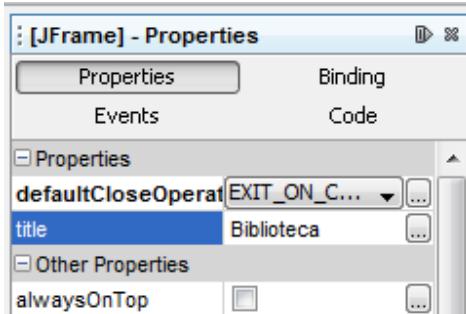


În fereastra care se afișează se introduce numele noii clase. Fiind clasa principală a aplicației, ea va purta numele proiectului, *biblio*. În fereastra *New JFrame Form* este bine să se indice de asemenea numele unui nou pachet care va înlocui pachetul implicit și va coține fișierele sursă ale aplicației.



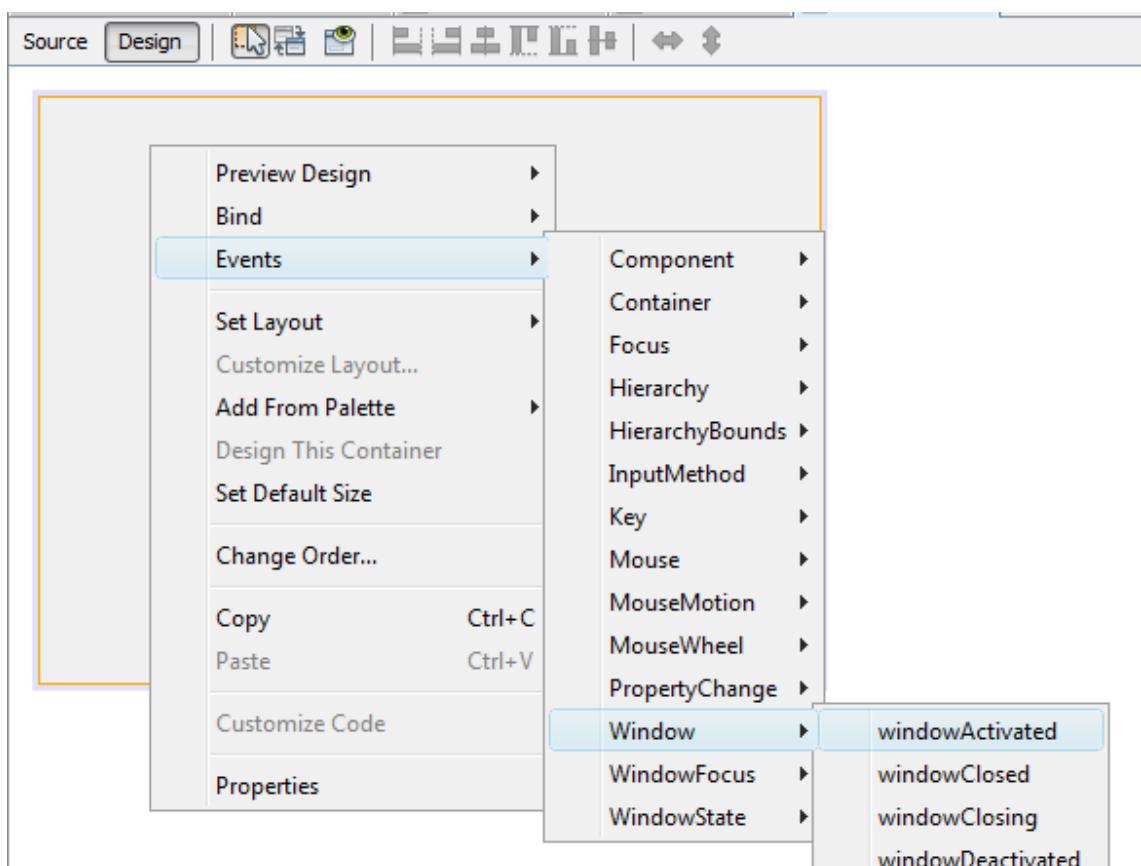


Pentru a da un titlu ferestrei se va selecta un punct din fereastră și se va modifica proprietatea *title*.



Prin demersuri oarecum asemănătoare vor fi adăugate ferestrele de dialog care vor conține formularele și rapoartele necesare exploatarii bazei de date *Oracle XE*.

Pentru a impune poziția în care se va afișa fereastra creată se va selecta fereastra (clic dreapta) și în meniu contextual se va selecta succesiv *Events / Window / windowActivated*.



Mediul de dezvoltare va genera metoda `formWindowActivated()` care va fi modificată astfel:

```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    this.setLocation(150, 100);
    this.pack();
}
```

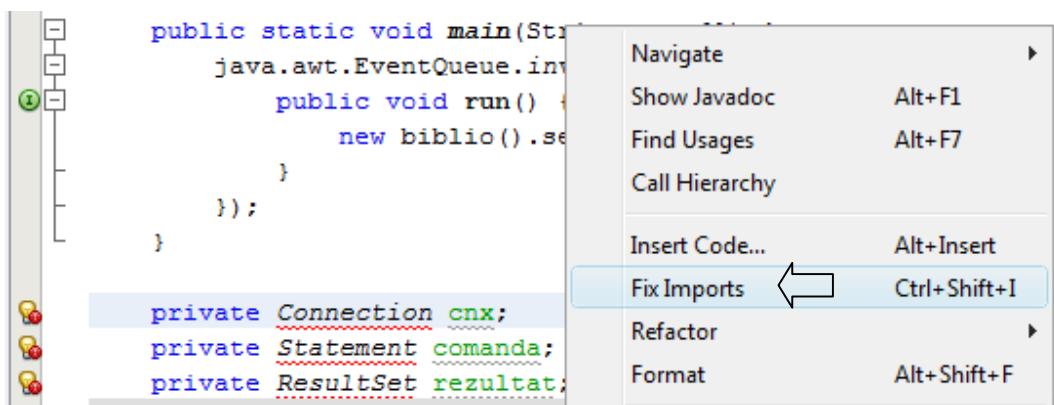
```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    this.setLocation(150, 100);
    this.pack();
}
```

În această etapă se poate testa aplicația precum și posibilitatea conectării la serverul *Oracle XE*. Pentru aceasta se va proceda astfel:

- Se inserează la începutul fișierului *Biblio.java* liniile:

```
import java.sql.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.OracleDataSource;
```

Observație: Aceste linii sunt inserate și de mediul de programare dacă se include codul care conține declarații ale unor obiecte aparținând unor clase declarate în aceste pachete: Connection, ResultSet, Statement etc.. După declararea obiectelor se selectează cu butonul drept un punct din fereastra de editare și în meniul contextual se selectează *Fix Imports*.



Mediul de programare va afișa o fereastră în care va arăta ce pachete va include. Pentru cazurile în care există mai multe soluții se va alege varianta din figură.



- Se adaugă clasei principale variabila *cnx* de tip *Connection*. Pentru aceasta se merge în codul sursă la sfârșit, în zona de declarații de variabile și se tastează declarația:

```
private Connection cnx;
// Variables declaration - do not modify
// End of variables declaration
```

}

- Se completează constructorului clasei *Biblio* cu liniile necesare creării obiectului *cnx*. În prima fază unele linii vor fi evidențiate ca fiind eronate. Mediul de programare cere ca toate liniile legate de accesarea unei baze de date să fie integrate în secvențe *try - catch*.



Mediul poate însă adăuga automat secvența *try - catch* corespunzătoare. Pentru aceasta se selectează cu butonul stâng al mouse-ului unul dintre mesajele de eroare și apoi se selectează *Surround Block with try-catch*.

The screenshot shows a Java code editor with the following code:

```

    /**
     * Creates new form biblio
     */
    unreported exception java.sql.SQLException; must be caught or declared to be thrown
    OracleDataSource ods = new OracleDataSource();
    biblio/biblio@localhost:1521/XE");
    Add throws clause for java.sql.SQLException
    Surround Statement with try-catch
    Surround Block with try-catch
    > Ok");
    initComponents();
}

```

A tooltip is displayed over the line `ods = new OracleDataSource();`, containing three suggestions:

- Add throws clause for `java.sql.SQLException`
- Surround Statement with try-catch
- Surround Block with try-catch

The third suggestion, "Surround Block with try-catch", is highlighted with a blue background.

Rezultat:

```

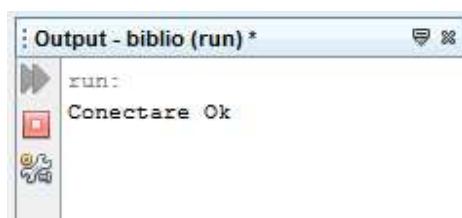
public biblio() {
    try {
        OracleDataSource ods = new OracleDataSource();
        ods.setURL("jdbc:oracle:thin:biblio/biblio@localhost:1521/XE");
        cnx = ods.getConnection();
        System.out.println("Conectare Ok");
        initComponents();
    } catch (SQLException ex) {
        Logger.getLogger(biblio.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

Liniile evidențiate realizează inițializarea variabilei `cnx`. Aceasta va fi folosită în continuare pentru accesul la baza de date.

În continuare se poate lansa aplicația în execuție.

Dacă nu sunt incidente legate de conectarea la serverul de baze de date, aplicația va afișa fereastra principală (goală!) iar în regiunea mediului de programare afectată consolii va apărea sirul de caractere "*Conectare Ok*":



Realizarea formularelor

Accesul la datele conținute într-o bază de date se realizează prin *formulare* și *rapoarte*. Formularele asigură următoarele funcții principale :

- Afisează date individuale;
- Permit modificarea datelor afișate;
- Permit suprimarea articolelor ;
- Permit adăugarea unui articol.

Rapoartele prezintă într-un format impus date conținute în baza de date. În unele cazuri formatul în care datele trebuie prezentate datele este definit prin lege. Aplicația trebuie să asigure afișarea pe ecran și imprimarea rapoartelor.

Controale Windows

Formularele prin care sunt accesate datele conținute într-o bază de date pot fi destinate realizării unei funcții elementare (afișare date, modificare, adăugare etc.) sau pot realiza o funcție complexă. Un formular destinat realizării unei funcții complexe conține un mare număr de controale Windows, în anumeite stări ale aplicației putându-se pune chiar problema ascunderii unora dintre ele.

O aplicație care conține formulare simple, fiecare destinat realizării unei acțiuni elementare, poate pune probleme de operare dar practica programării recomandă această abordare deoarece atât depanarea cât și întreținerea aplicațiilor astfel concepute este mai simplă.

JMenuBar

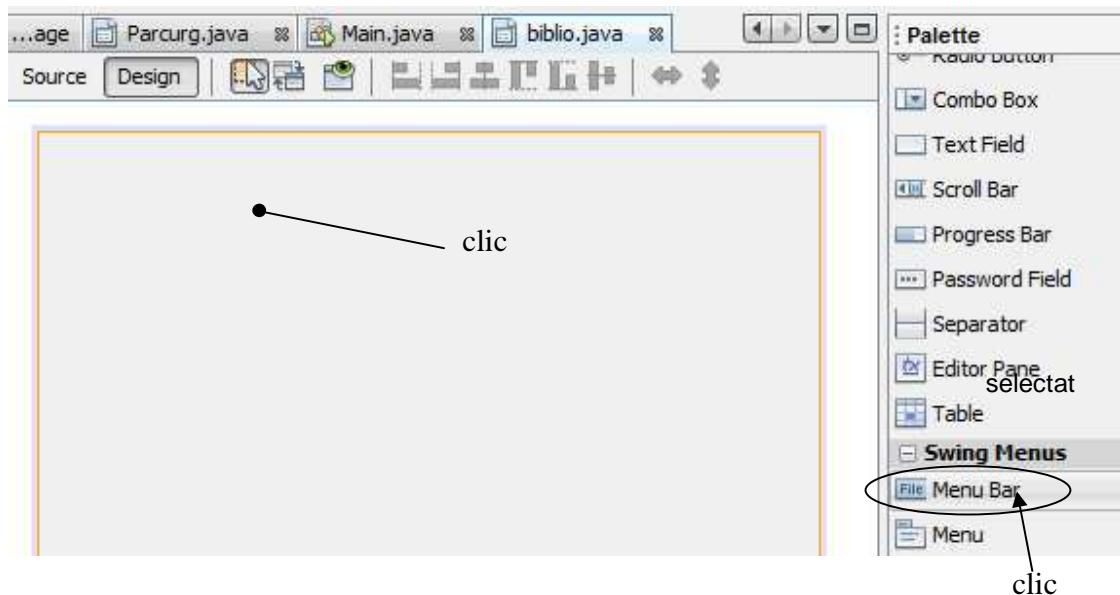
Inițierea diferențelor acțiuni se realizează în principal folosind meniuri derulante și butoane. Meniurile derulante vor declanșa principalele funcții ale aplicației (afișarea formularelor și rapoartelor, salvarea bazei de date, oprirea aplicației) iar butoanele vor declanșa acțiuni în interiorul formularelor.

Pentru a defini meniul unei aplicații trebuie creată o bară de meniuri (obiect aparținând clasei *JMenuBar*), un ansamblu de meniuri derulante (obiecte de tip *JMenu*) și pentru fiecare meniu derulant, un ansamblu de opțiuni (obiecte aparținând clasei *JMenuItem*).

Pentru a realiza o aplicație care accesează baza de date Oracle XE, schema *Biblio*, se va proceda astfel:

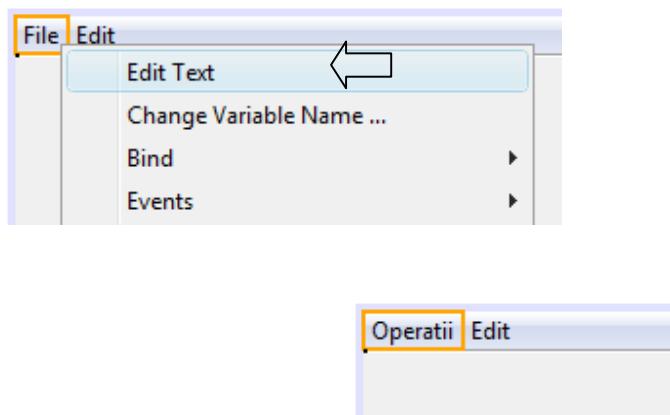
- a. Se adaugă proiectului bara cu meniuri derulante:

Se va selecta cu mouse-ul *Menu Bar* și se va realiza adăugarea sa printr-un clic cu mouse-ul în fereastra aplicăției.

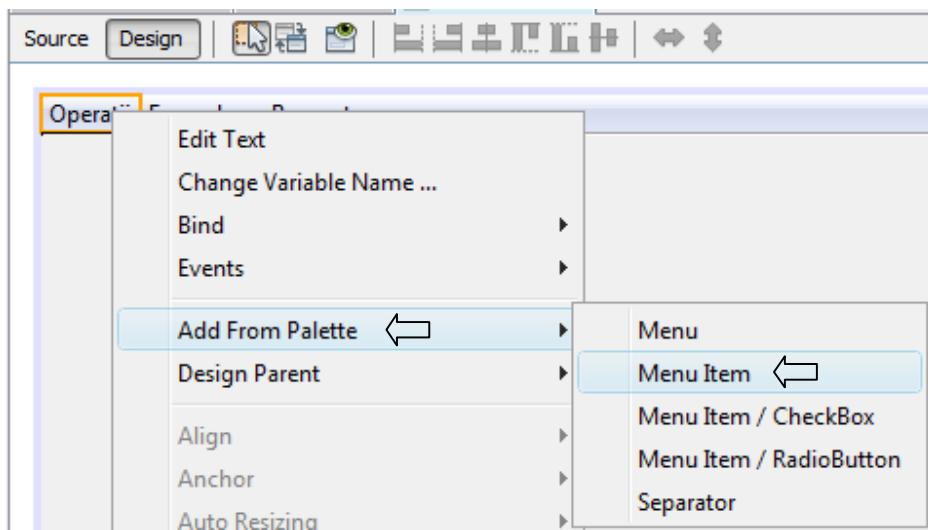


În continuare se va edita conținutul barei cu meniu folosind reprezentarea arborescentă afișată prin selectarea tabului *Inspector*.

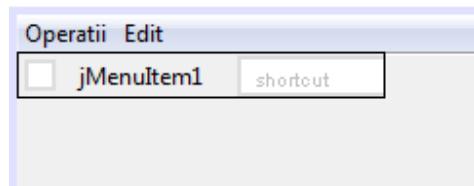
b. Se modifică numele primului meniu derulant (proprietatea *text*):



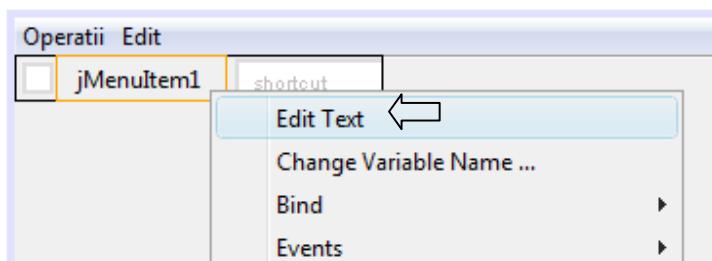
c. Se adaugă meniului *Operații* opțiunea *Iesire*. Efectul selectării acesteia va fi oprirea aplicăției. Pentru a iniția adăugarea noii opțiuni se selectează cu butonul drept al mouse-ului meniu *Operații* și apoi se selectează în meniul contextual *Add From Palette / Menu Item*.



Rezultat:

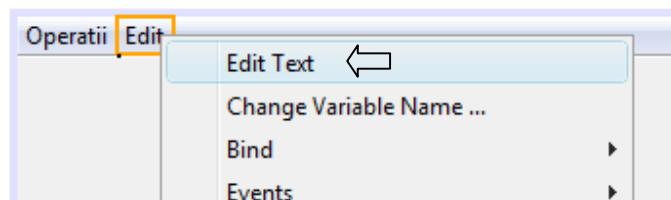


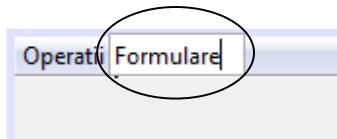
Se modifică apoi proprietatea *text* a opțiunii nou adăugate în /eșire:



d. Se editează apoi al doilea meniu de pe bara cu meniu derulante.

Se modifică proprietatea *text* a noului obiect din *Edit* în *Formular*:

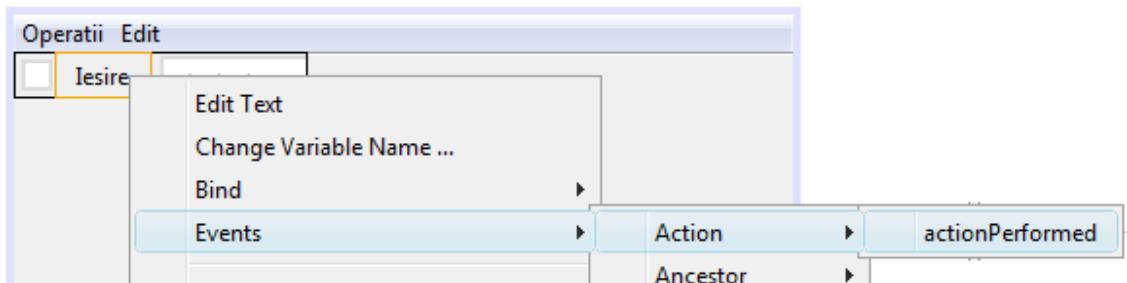




Celui de-al doilea meniu derulant i se vor adăuga în timp mai multe intrări, fiecare având rolul de a declanșa afișarea uneia dintre formularele care vor fi realizate.

În același mod va fi adăugat aplicației un al treilea meniu derulant, *Rapoarte*, ale cărui intrări vor fi folosite pentru a declanșa afișarea rapoartelor.

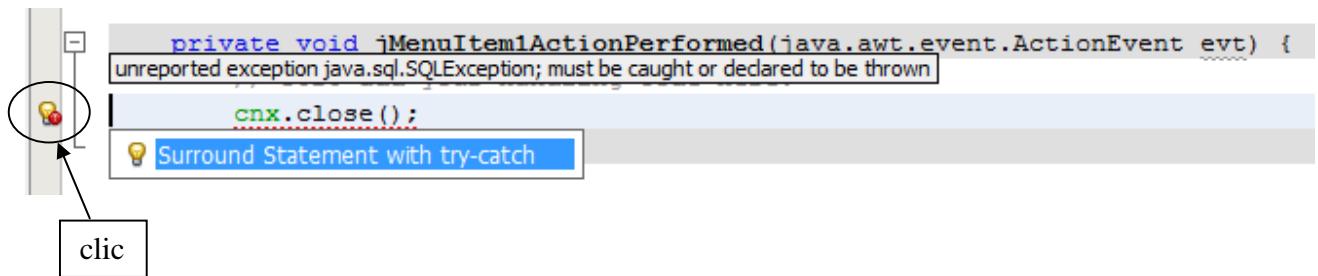
Pentru a crea metoda care trebuie executată la selectarea unei opțiuni dintr-un meniu derulant se va selecta opțiunea cu un dublu clic în fereastra *Inspector* sau se va selecta opțiunea în fereastra *Design* cu butonul drept al mouse-ului și în meniul contextual se va selecta *Events / Action / actionPerformed*.



Pentru opțiunea *Iesire* din primul meniu derulant funcția astfel adăugată trebuie completată cu secvența de cod care închide conexiunea cu serverul și apoi apelează metoda *System.exit(0)* a cărui efect este oprirea imediată a aplicației.

```
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt)
{
    try {
        // TODO add your handling code here:
        cnx.close();
    } catch (SQLException ex) {
        Logger.getLogger(biblio.class.getName()).log(Level.SEVERE,
null, ex);
    }
    System.exit(0);
}
```

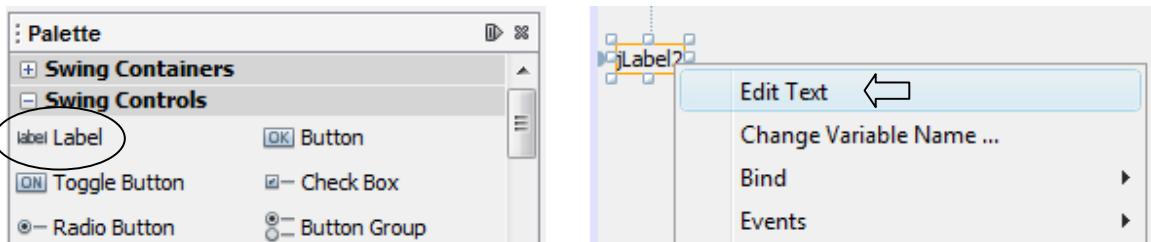
Secvența *try-catch* a fost adăugată de mediul de programare. Practic s-a scris *cnx.close();* după care s-a selectat cu mouse-ul simbolul care indică o eroare și s-a selectat opțiunea *Surround Statement with try-catch*.



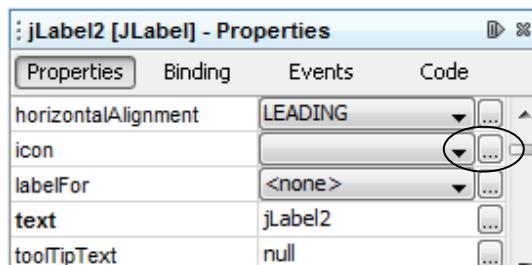
JLabel

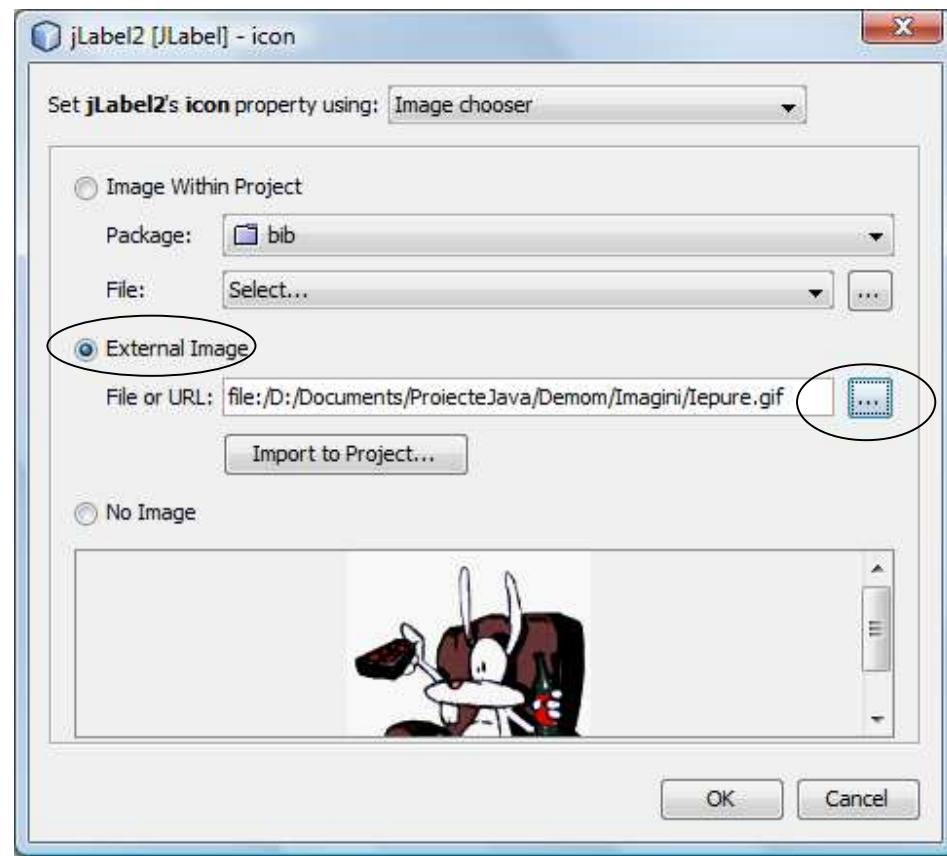
Controlul de tip *JLabel* servește la plasarea într-un formular a unui text sau a unei imagini.

După includerea controlului trebuie setate principalele proprietăți ale acestuia, respectiv *text* sau *icon* pentru definirea conținutului, *font*, *foreground*, *background* și.a.

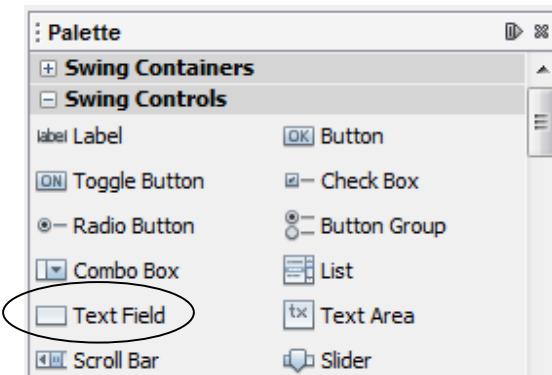


Proprietățile *icon* și *text* se pot folosi împreună. Dacă se selectează *icon* trebuie indicat fișierul care conține imaginea de afișat, aceasta fiind în format JPEG (*Joint Photographic Experts Group*, având extensia *.jpg*) sau GIF (*Graphics Interchange Format* având extensia *.gif*).





JTextField



Controlul de tip *JTextField*, este folosit la afişarea sau introducerea şirurilor de caractere.

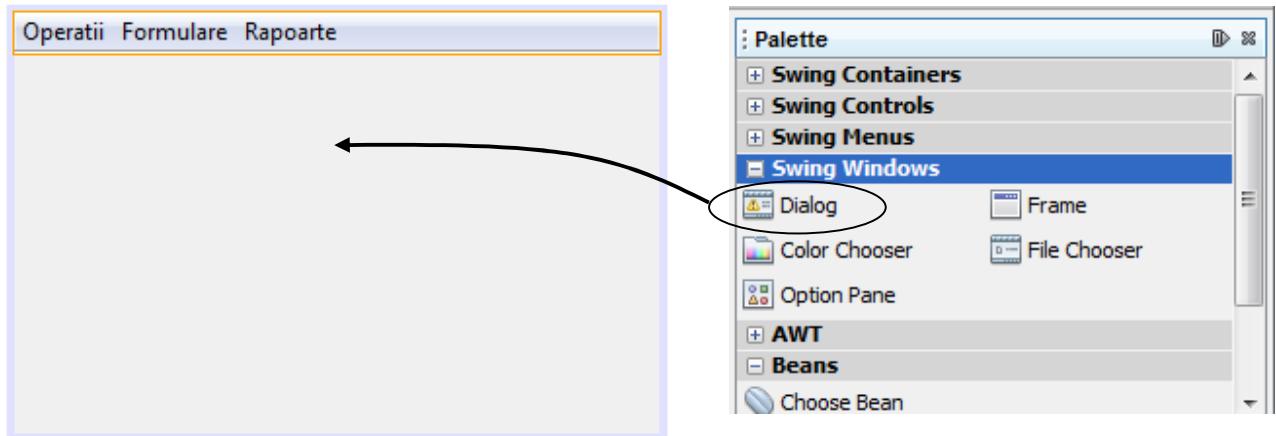
Accesul la conținutul unui control de tip *JTextField* se realizează cu perechea de metode *setText()* (impunere *String* conținut) și *getText()* (preluare *String* conținut) ca în exemplul următor:

```
nume_ed.setText(nume);
...
String nume = nume_ed.getText();
```

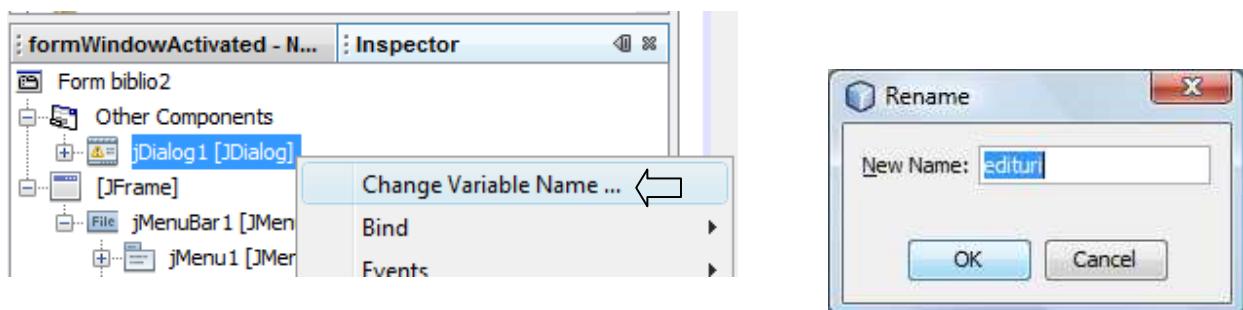
Exemplu fundamental : Să se adauge aplicației un formular care permite navigarea în tabelul *Edituri*. Pentru navigare se vor folosi două butoane având pentru proprietatea *text* valoarea *Inainte* respectiv *Inapoi*.

Rezolvare:

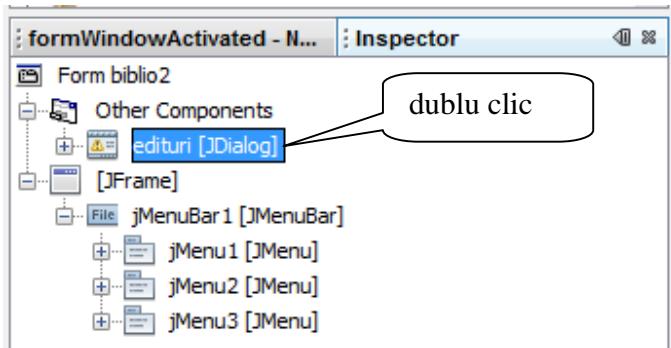
În NetBeans soluția rapidă pentru adăugarea unei ferestre de dialog constă în selectarea succesivă cu mouse-ul a butonului *JDialog* din fereastra *Palette* și a ferestrei principale (sau, mai general, fereastra aplicației în cadrul căreia se va afișa noua fereastră).



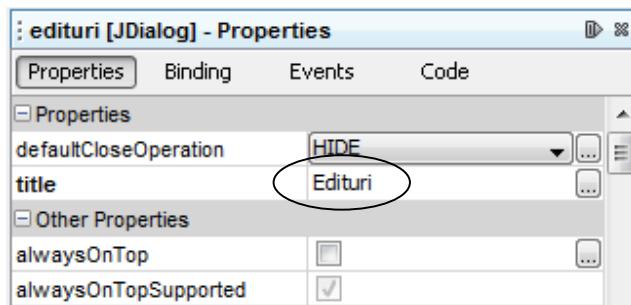
După adăugarea ferestrei de dialog se începe configurarea acesteia. Prima operație va fi schimbarea numelui implicit din *jDialog1* în *edituri*. Pentru aceasta se selectează cu butonul drept al mouse-ului în fereastra *Inspector* noua fereastră și apoi, în meniul contextual, se selectează *Change Variable Name...*



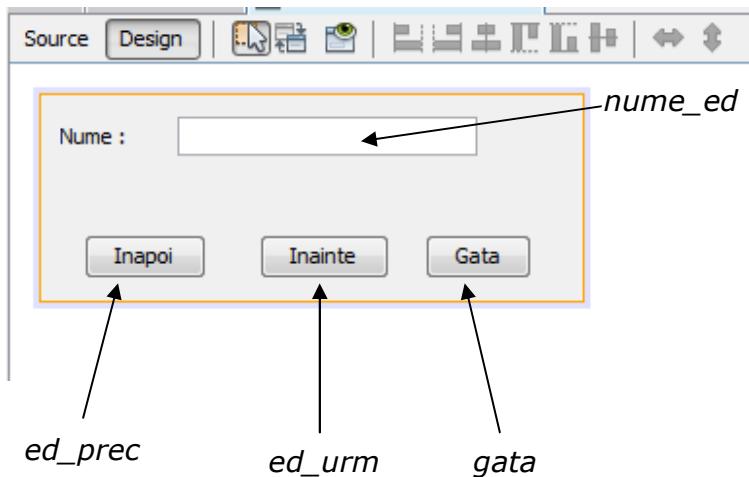
Pentru a edita conținutul ferestrei de dialog se selectează cu un dublu clic în fereastra *Inspector* identificatorul acestia:



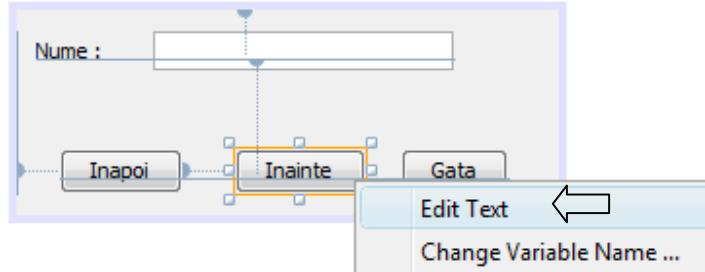
Ca și în cazul ferestrei principale, pentru a impune un titlu care se va afișa pe bara ferestrei se selectează cu mouse-ul un punct din fereastră și se scrie titlul dorit în dreptul proprietății *Title*.



Folosind instrumentele din fereastra *Palette*, se adaugă noi ferestre cinci controale Windows: o etichetă (*JLabel* având *text=Nume*), o casetă de text (*JTextBox* având *Name=nume_ed*) și trei butoane având numele *ed_prec*, *ed_urm* respectiv *gata* și proprietățile *text* ca în figura următoare.



Pentru a modifica etichetele butoanelor se accesează proprietatea *text* a acestora sau se selectează controlul cu butonul stâng al mouse-ului și se selectează *Edit Text* în meniul contextual.



Înainte de a continua cu scrierea secvențelor de cod necesare funcționării noii ferestre se va verifica în fereastra *Inspector* corecitudinea configurării.



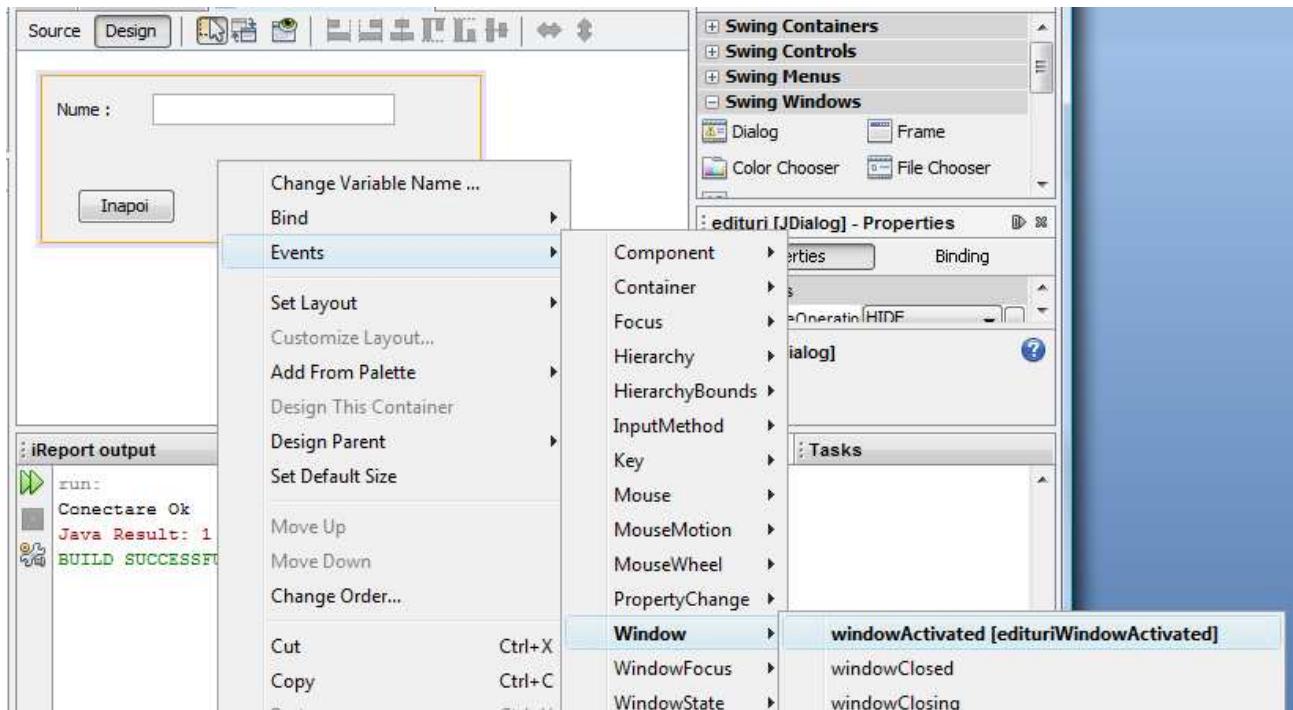
Deoarece rolul acestui formular este parcurgerea tabelului *Edituri*, clasei principale i se va adăuga un obiect din clasa *ResultSet* denumit *rs_edituri*. Acesta va fi inițializat într-o secvență de cod executată la deschiderea formularului și va fi închis (se va apela metoda *close()*) la închiderea acestuia. De asemenea se va adăuga clasei principale obiectul *stm_edituri* aparținând clasei *Statement*. Acesta va permite crearea obiectului *rs_edituri* care va conține mulțimea de selecție rezultată în urma executării de către serverul *Oracle XE* a comenzi SQL de selectare a editurilor din baza de date.

Zona din codul clasei destinată declarației variabilelor clasei va conține deci variabilele următoare:

```
private Connection cnx;
private Statement stm_edituri;
private ResultSet rs_edituri;
```

Pentru a realiza acțiuni **înaintea afișării unei ferestre** (*JFrame* sau *JDialog*) este necesară adăugarea unei metode asociate evenimentului *windowActivated*.

Pentru aceasta se va selecta cu butonul drept al mouse-ului un punct din fereastra în care nu există nici un control Windows și în meniul contextual se va selecta *Events / Window / windowActivated*.

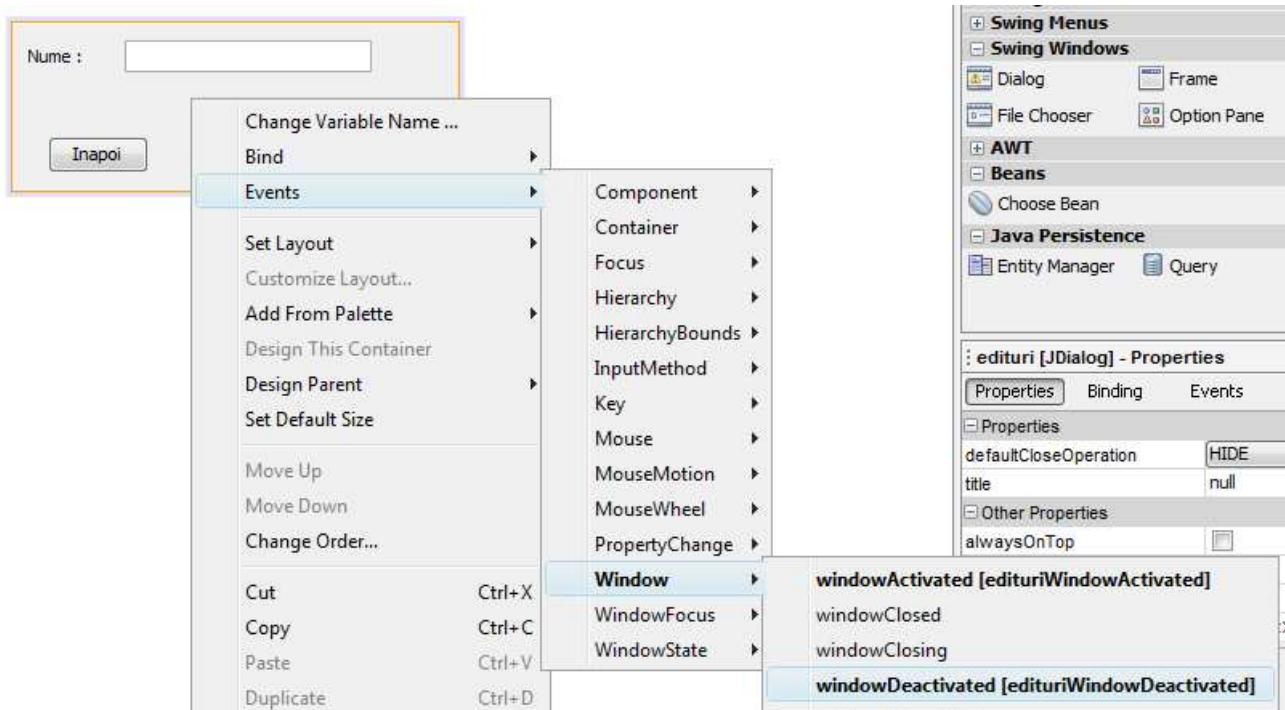


În cazul formularului *edituri*, metoda *edituriWindowActivated()* astfel adăugată va inițializa succesiv variabilele *stm_edituri* (*Statement*) și *rs_edituri* (*ResultSet*) și va plasa în controlul *nume_ed* numele primei edituri.

```
private void edituriWindowActivated(java.awt.event.WindowEvent evt) {
    try {
        // TODO add your handling code here:
        stm_edituri =
            cnx.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                               ResultSet.CONCUR_UPDATABLE);
        // resultSet-ul poate fi parcurs în ambele sensuri și
        // permite update
        rs_edituri = stm_edituri.executeQuery("SELECT * FROM edituri");
        if (rs_edituri.next()) {
            // preiau numele
            String nm = rs_edituri.getString("nume");
            nume_ed.setText(nm);
        }
    } catch (SQLException ex) {
        Logger.getLogger(biblio2.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

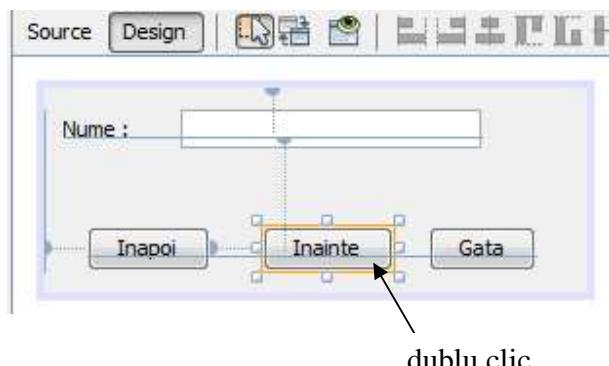
Practic s-a adăugat doar codul evidențiat, adăugarea instrucțiunii *try-catch* realizându-se de mediul de programare, ca în exemplele precedente.

La închiderea ferestrei este necesară apelarea metodelor *close()* pentru obiectele *rezultat* și *comanda*. Pentru a adăuga aplicației o metodă care se apelează la închiderea formularului *edituri* se procedează ca mai sus, evenimentul selectat fiind *windowDeactivated*.



```
private void edituriWindowDeactivated(java.awt.event.WindowEvent evt) {
    try {
        // TODO add your handling code here:
        rs_edituri.close();
        stm_edituri.close();
    } catch (SQLException ex) {
        Logger.getLogger(biblio2.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Pentru a iniția introducerea secvenței de cod care trebuie executată la acționarea unui buton se modifică proprietatea *actionPerformed* sau, mai simplu, se selectează controlul cu un dublu clic. Mediul de dezvoltare va include automat o funcție de tratare a evenimentului al cărui conținut trebuie apoi editat.



```

private void ed_urmActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        if (!rs_edituri.isLast()) {
            ed_prec.setEnabled(true);
            if (rs_edituri.next()) {
                // preiau numele
                String nm = rs_edituri.getString("nume");
                nume_ed.setText(nm);
            }
        } else {
            ed_urm.setEnabled(false);
        }
    } catch (SQLException ex) {
        Logger.getLogger(biblio2.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

La fel se inițiază scrierea codului pentru celelalte două butoane, *ed_prec* și *gata*:

```

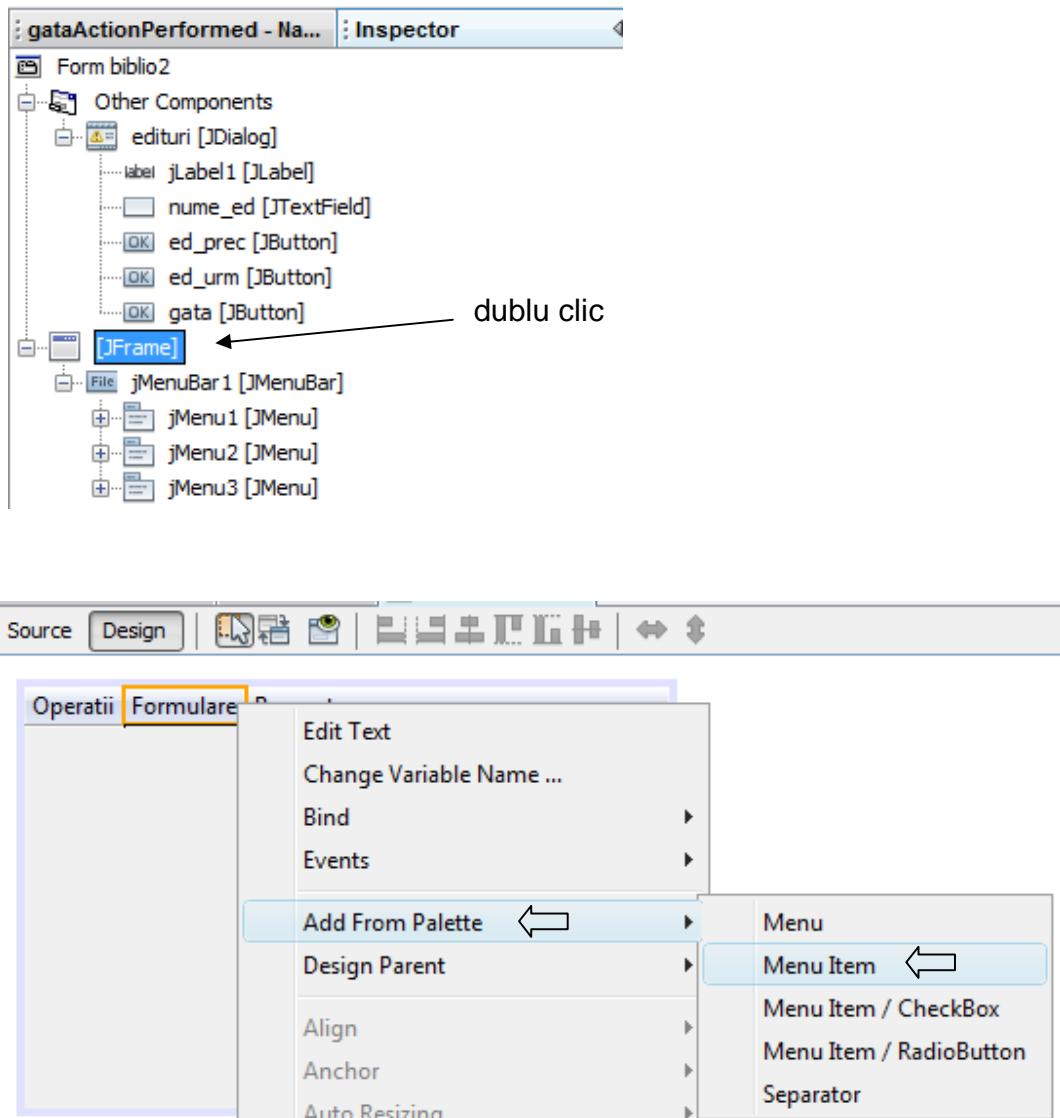
private void ed_precActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        if (!rs_edituri.isFirst()) {
            ed_urm.setEnabled(true);
            if (rs_edituri.previous()) {
                // preiau numele
                String nm = rs_edituri.getString("nume");
                nume_ed.setText(nm);
            }
        } else {
            ed_prec.setEnabled(false);
        }
    } catch (SQLException ex) {
        Logger.getLogger(biblio2.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void gataActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    edituri.setVisible(false);
}

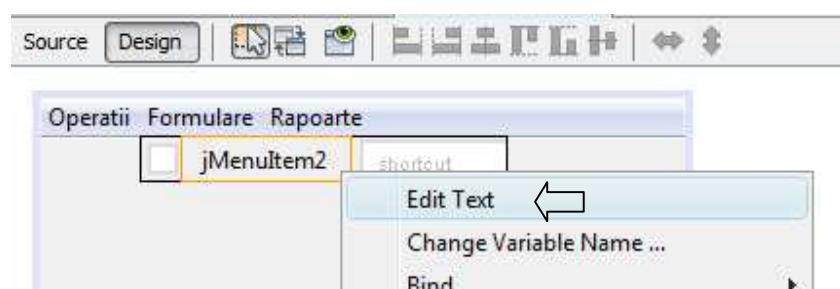
```

Metoda *setVisible(false)* realizează ascunderea ferestrei de dialog.

Pentru a realiza afişarea ferestrei de dialog create se revine la fereastra principală (dublu clic pe *JFrame* în fereastra *Inspector*) și se adăugă meniului Formulară o primă opțiune, *Edituri*.

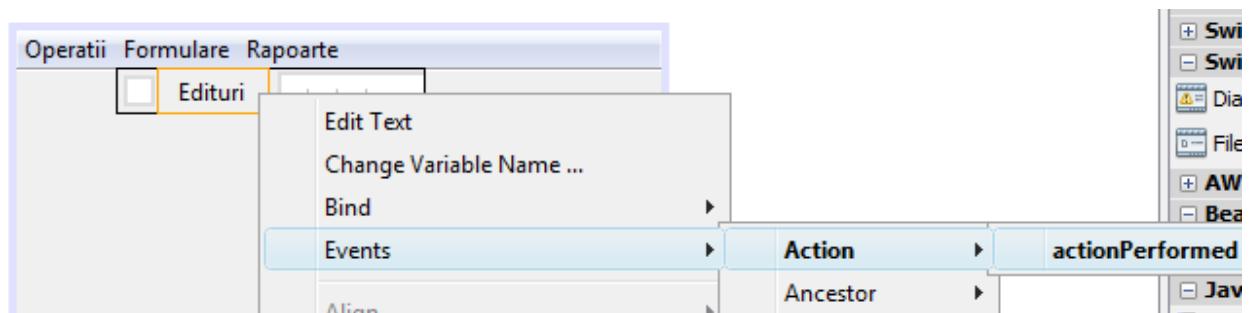


În continuare se modifică proprietatea *text* a noii opțiuni în *Edituri*:





În continuare se crează metoda care trebuie executată la selectarea noii opțiuni. Pentru aceasta se selectează opțiunea cu butonul drept al mouse-ului și în meniu contextual afișat se selectează *Events / Action / Action Performed*.



Conținutul metodei *AEdituriActionPerformed()* este următorul:

```
private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    edituri.setLocation(170,140);
    edituri.pack();
    edituri.setVisible(true);
}
```

Codul cuprinde o linie care impune poziția noii ferestre (*edituri.setLocation()*), determină afișarea ferestrei (*edituri.setVisible(true)*) și impune reconstruirea interfeței conținute (*edituri.pack()*) în acord cu managerul de dispunere a obiectelor grafice (eng. *layout manager*).

În continuare se poate executa aplicația apăsând butonul .



Alte controale Windows

JPasswordField



Controlul de tip *JPasswordField* este folosit pentru preluarea parolelor. Proprietățile sale sunt similare controlului de tip *JTextField*. Practic, într-o aplicație care accesază o bază de date se preia o parolă care este folosită ulterior la stabilirea nivelului de acces al utilizatorului aplicației. Astfel dacă parola este a unui simplu operator care culege date o parte din elementele de control ale aplicației (opțiuni din meniuri, butoane de pe barele cu instrumente sau conținute în formulare) devin inactive. Ele vor fi desigur activate dacă parola introdusă este cea a inginerului de baze de date care întreține aplicația. Între cele două extreme pot fi definite diverse nivele de prioritate, fiecare cu caracteristicile sale.

JTextArea



Controlul de tip *JTextArea* se utilizează pentru introducerea sirurilor de caractere lungi, pe mai multe linii. Proprietățile folosite pentru accesarea conținutului sunt aceleași ca și în cazul controlul de tip *JTextField*.

JCheckBox



Un control de tip casetă de validare (*JCheckBox*) poate avea două stări: selectat sau neselectat. Starea de selectare poate fi stabilită în momentul realizării interfeței grafice sau în timpul execuției aplicației, prin apelul metodei *setChecked()*. Testarea stării controlului se realizează prin apelul metodei *isSelected()*.

Ca și în cazul controalelor de tip *JButton*, controlului de tip *JCheckBox* î se poate asocia o secvență de tratare a evenimentului declanșat la selectarea acestuia cu mouse-ul.



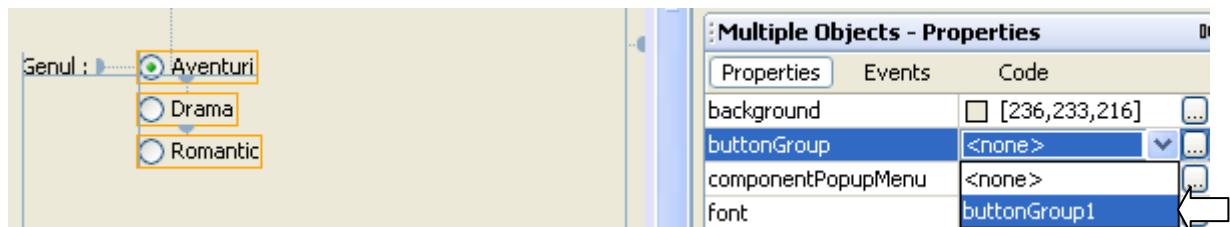
```

private void dispoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if(dispo.isSelected()) {
        // Actiune pentru trecerea din neselectat in selectat
    } else {
        // Actiune pentru trecere din selectat in neselectat
    }
}

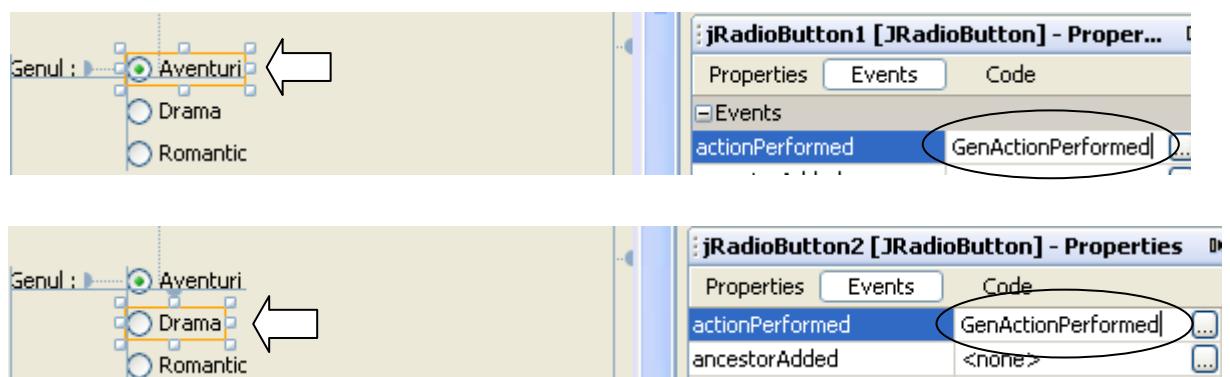
```

JRadioButton

Cadroalele de tip *JRadioButton* (butoane *radio*) servesc la selectarea unei valori dintr-un set de valori posibile. Pentru a impune acest comportament, înaintea adăugării butoanelor de tip *JRadioButton* care vor compune un grup de butoane se va adăuga interfeței un control de tip *ButtonGroup*. Pentru a realiza gruparea, se selectează succesiv, cu tasta *Control* apăsată, butoanele radio care compun grupul și apoi, în fereastra de proprietăți, se selectează pentru proprietatea *buttonGroup* numele obiectului de tip *ButtonGroup* adăugat anterior.



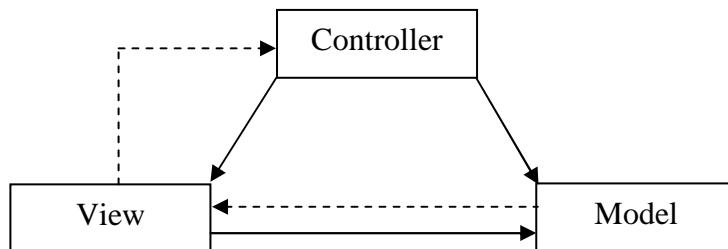
Uneori metoda de tratare a evenimentului declanșat la selectarea unuia dintre butoane trebuie să fie aceeași pentru toate butoanele care formează grupul.. Pentru a impune acest lucru se scrie metoda folosind unul dintre butoane, după care se selectează succesiv butoanele și se editează numele dat implicit metodei de tratare a evenimentului *actionPerformed*.



Starea inițială a butoanelor, respectiv butonul care apare selectat la afișarea interfeței se impune folosind proprietatea *selected* a butoanelor. Evident, numai un buton din grup poate avea starea selectată. Ca și în cazul controalelor de tip *JCheckBox*, se poate modifica prin program starea de selectare a uneia dintre butoanele care formează grupul apelând metoda *setSelected()* și se poate testa starea uneia dintre butoane prin apelul metodei *isSelected()*.

Controale Windows cu listă

Componentele Swing folosite în limbajul Java începând cu versiunea 1.2 implementează arhitectura *Model-View-Controller* (MVC). Modelul MVC constă în esență în separarea datelor de reprezentarea lor. Datele sunt conținute într-un obiect (*model*) asociat unui control Windows care realizează reprezentarea lor (*view*). Cea de-a treia componentă, *Controller* realizează interceptarea și procesarea evenimentelor.



Această abordare stă de altfel la baza independenței aplicațiilor Java de platforma pe care se execută aplicația.

Dacă pentru controale simple obiectul care conține datele este realizat automat, pentru controale complexe, care pot conține un număr mare și variat de obiecte (*JComboBox*, *JList*, *JTable*, *JTree*), programatorul poate opta între folosirea unui obiect aparținând unei clase隐式 (DefaultListModel, DefaultTableModel, DefaultTreeModel) și definirea unei clase proprii, derivată dintr-o clasă abstractă predefinită. Evident complexitatea programării crește dacă nu se folosește clasa implicită, dar în cazul unor reprezentări mai deosebite este singura soluție.

JComboBox

Controlul de tip *JComboBox* permite selectarea unei valori dintr-o listă predefinită.

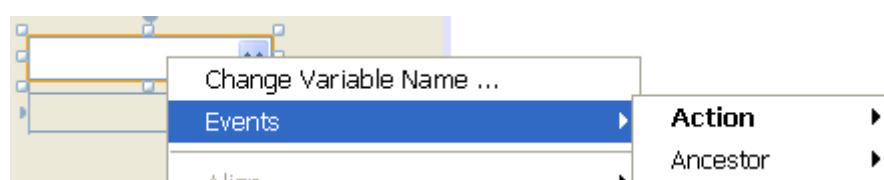
Definirea listei se realizează de obicei în etapa de proiectare a aplicației, conținutul acesteia fiind accesibil prin intermediul proprietății *model*. Lista poate fi construită și editată și dinamic, în timpul execuției, folosind metodele din tabelul următor.

Tip returnat	Metoda	ACTION
void	addItem(Object obj)	Adaugă un element în listă
void	insertItemAt(Object obj, int index)	Inserează un element în listă
Object	getSelectedItem()	Returnează ob. selectat
int	getSelectedIndex()	Returnează poz. elem. selectat
void	removeAllItems()	Golește lista
void	removeItem(Object obj)	Suprimă prima apariție a obj.
void	removeItemAt(int index)	Suprimă el.. de la poziția index
int	getItemCount()	Returnează nr. de elemente
void	setEditable()	În funcție de argument caseta de text va fi editabilă sau needitabilă

În aplicațiile destinate realizării interfeței cu baze de date prezintă un mare interes popularea listei prin preluarea valorilor dintr-o mulțime de selecție, ca în exemplul următor:

```
rezultat =
    comanda.executeQuery("SELECT * FROM edituri order by nume");
while (rezultat.next()) // Trec pe urmatoarea linie
{
    // preiau numele
    String nm = rezultat.getString("nume");
    listaed.addItem(nm); // listaed este un JComboBox
}
```

Pentru a insera o funcție de tratare a evenimentului produs la selectarea unui element folosind un control de tip *JComboBox* se procedează ca și în cazurile deja prezentate, respectiv se selectează controlul cu butonul drept al mouse-ului și în meniul contextual se selectează *Events / Action*:



```

private void listaedActionPerfomed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Copiez din combo in variabila numeed de tip String
    numeed = (String)listaed.getSelectedItem();
}

```

Există situații în care caseta de text a controlului trebuie să primească doar o parte din șirul conținut în linia selectată. În exemplul următor s-a reținut în caseta de text doar primul element din șirul de caractere conținut în linia selectată:

```

private void listacitActionPerfomed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Preiau sirul de car. din lista
    String nm = (String)listaed.getSelectedItem();
    // Separ subsirurile din sir
    String[] sir = nm.split(" ");
    listaed.setEditable(true); // caseta combo devine editabila
    listaed.setSelectedItem(sir[0]); // preiau numai primul element
    listaed.setEditable(false); // caseta combo devine needitabila
}

```

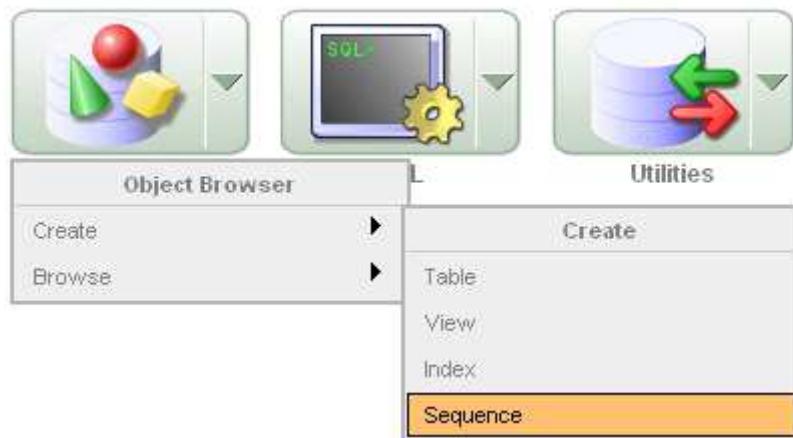
Metoda `split()` a clasei `String` permite separarea unui șir de caractere într-un număr de subsiruri, separatorii permisi fiind date într-un șir (al doilea argument). În exemplul prezentat șirul folosit ca argument conține un spațiu.

Pentru a impune valoarea din caseta de text a controlului combinat trebuie modificat tipul acestuia, din `needitabil` în `editabil`. După impunerea valorii se revine la starea inițială. Modificarea acestei proprietăți se realizează folosind metoda `setEditable()`.

Exemplu : Se cere să se realizeze un formular pentru introducerea cititorilor bibliotecii în tabelul *Cititori* având structura :

CITITORI					
Add Column	Modify Column	Rename Column	Drop Column	Rename	
Column Name	Data Type	Nullable	Default	Primary Key	
COD_CIT	NUMBER(6,0)	No	-	1	
CNP	VARCHAR2(15)	Yes	-	-	
NUME	VARCHAR2(30)	No	-	-	
PRENUME	VARCHAR2(50)	Yes	-	-	
LOCALITATEA	VARCHAR2(20)	Yes	-	-	
ID_JUD	NUMBER(2,0)	Yes	-	-	
ADRESA	VARCHAR2(100)	Yes	-	-	
TELEFON	NUMBER(10,0)	Yes	-	-	
E_MAIL	VARCHAR2(20)	Yes	-	-	
1 - 9					

Câmpul *COD_CIT* fiind de tip *autoincrement*, este comod să se creeze pentru initializarea lui cu valori corecte un obiect de tip *sequence* (secvență). Declararea în Oracle XE a unui obiect de tip *sequence* se realizează selectând *Object Browser / Create / Sequence* :



În noua fereastră se va defini numele obiectului (*citi_seq*), valoarea de start și incrementul.

Create Sequence

Schema BIBLIO

* Sequence Name:

Preserve Case

Start With:

Minimum Value:

Maximum Value:

Increment By:

CYCLE

Number to Cache:

ORDER

Cancel **Next >**

Evidențierea funcționării noului obiect se poate realiza tastând o comandă SQL SELECT.

ORACLE® Database Express Edition

User: BIBLIO

Home > SQL > **SQL Commands**

Autocommit Display 10

```
select citi_seq.nextval from dual
```

Funcția *NEXTVAL* incrementează secvența cu o unitate iar funcția *CURRVAL* returnează valoarea curentă a acesteia.

Tot ca element ajutător s-a definit tabelul *Judete*. Acesta are două câmpuri și va fi folosit pentru popularea cu date a unui control *JComboBox* care va servi la alegerea județului. În tabelul *Cititori* va fi introdusă valoarea numerică *ID_JUD* aferentă județului selectat.

JUDETE					
Add Column	Modify Column	Rename Column	Drop Column	Rename	
Column Name	Data Type	Nullable	Default	Primary Key	
ID_JUD	NUMBER(2,0)	No	-	1	
NUME	VARCHAR2(25)	No	-	-	
1 - 2					

Ca și în cazul primului formular realizat, se va adăuga clasei principale un obiect din clasa *JDialog* căruia îi vor fi apoi adăugate controalele necesare, ca în figură.

The screenshot shows a JDialog window titled "Cititori". The window contains the following fields:

- Group 1 (Left): `cit_nume, cit_prenume`, `cit_cnp`, `cit_jud`, `cit_loc`, `cit_adr`.
- Group 2 (Right): `Nome :` [Text Box], `Prenume :` [Text Box], `C.N.P. :` [Text Box], `Judet :` [ComboBox], `Localitate :` [Text Box].
- Group 3 (Bottom): `Adresa :` [Text Area with scroll bars].
- Group 4 (Bottom): `Telefon :` [Text Box], `E-mail :` [Text Box].
- Buttons at the bottom: `Adauga`, `Imprima`, `Gata`.

Numele ferestrei de dialog este *ACititor* iar în stânga imaginii noii ferestre sunt scrise numele obiectelor pe care le conține (exceptând etichetele, desigur!). Pentru a putea imprima conținutul formularului *ACititor*, partea utilă a acestuia (zona de

deasupra butoanelor de comandă) a fost dispusă pe un control de tip *JPanel* (denumit *cit_imprim*).

Pentru afișarea formularului se va adăuga încă o opțiune meniului derulant *Formulare*, metoda de tratare a evenimentului declanșat la selectarea acestuia fiind prezentată în continuare.

```
private void AdCititorActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ACititor.setLocation(120, 170);
    ACititor.setVisible(true);
    ACititor.pack();
}
```

Ca și în cazurile anterioare, metoda declanșată de selectarea butonului *Gata* (obiectul *cit_gata*) realizează ascunderea ferestrei.

```
private void cit_gataActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    ACititor.setVisible(false);
}
```

Pentru a pregăti formularul în vederea culegerii de date trebuie populată lista controlului *cit_jud*. Pentru aceasta vor fi adăugate clasei principale două noi obiecte, *cit_cmd* de tip *Statement* și *cit_rs* de tip *ResultSet*. Cele două noi variabile vor fi inițializate în metoda asociată evenimentului *windowActivated*. (*ACititorWindowActivated*).

```
private void ACititorWindowActivated(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    try {
        // TODO add your handling code here:
        cit_cmd =
            cnx.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        cit_rs = cit_cmd.executeQuery("SELECT * from judete order by nume");
        // Golesc lista controlului cit_jud
        cit_jud.removeAllItems();
        while (cit_rs.next()) {
            // Preiau numele
            String nm = cit_rs.getString("nume");
            // Adaug numele in lista
            cit_jud.addItem(nm);
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

În metoda executată la ascunderea ferestrei de dialog, obiectul *cit_cmd* din clasa *Statement* este suprimat.

```

private void ACititorWindowDeactivated(java.awt.event.WindowEvent evt) {
    try {
        // TODO add your handling code here:
        cit_cmd.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}

```

În continuare se pot scrie metodele apelate la acționarea butoanelor *Adauga* și *Imprimă*.

Pentru primul buton, secvența de cod care trebuie scrisă va realiza construirea unei comenzi *INSERT* folosind valorile din câmpurile formularului. Exemplu:

```

INSERT into Cititori VALUES (citi_seq.NEXTVAL, '0987654321123',
    'Pop', 'Vasile', 'Drobeta', 12,
    'Str. Mica Nr. 14, Bl. B4 Ap. 6', '0258123456', '')

```

Pentru a construi valorile necesare introducerii cheii primare s-a utilizat secvența *citi_seq* declarată împreună cu tabelul *Cititori*.

Deoarece în comanda SQL șirurile de caractere sunt plasate între apostroafe, este convenabilă adăugarea unei metode care să returneze un șir format prin încardarea între apostroafe a șirului primit ca argument.

Metoda adăugată a fost denumită *apostrof()* și are următoarea definiție:

```

private String apostrof(String p)
{
    String c = "" + p + "";
    return c;
}

```

Metoda *cit_adaugaActionPerformed()* apelată la apăsarea butonului *Adauga* este următoarea:

```

private void cit_adaugaActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // Preiau valorile din controale
        String cnp, nume, prenume, localitatea, adresa, telefon, email;
        int jud;
        cnp = cit_cnp.getText();
        nume = cit_nume.getText();
        prenume = cit_prenume.getText();
        localitatea = cit_loc.getText();
        // Preiau județul din multimea de selectie
        int poz = cit_jud.getSelectedIndex(); // Poz. in lista
        // Caut in cit_rs poz. indicata
        cit_rs.absolute(poz+1); // mut? cursorul pe a linia poz+1 din cit_rs
        jud = cit_rs.getInt("ID_JUD");
    }
}

```

```

adresa = cit_adr.getText();
telefon = cit_tel.getText();
email = cit_email.getText();
// Formulez comanda INSERT
String frazaSQL = "INSERT into Cititori VALUES (citi_seq.NEXTVAL";
frazaSQL += "," + apostrof(cnp);
frazaSQL += "," + apostrof(nume);
frazaSQL += "," + apostrof(prenume);
frazaSQL += "," + apostrof(localitatea);
frazaSQL += "," + jud;
frazaSQL += "," + apostrof(adresa);
frazaSQL += "," + apostrof(telefon);
frazaSQL += "," + apostrof(email)+ ")";
// Creez un Statement
Statement stm = cnx.createStatement();
// Apelez metoda executeUpdate() pt. a trimite comanda INSERT
//System.out.println(frazaSQL);
stm.executeUpdate(frazaSQL);
stm.close();
} catch (SQLException ex) {
    ex.printStackTrace();
}
}
}

```

Dacă se testează aplicația, un rezultat posibil ar fi :

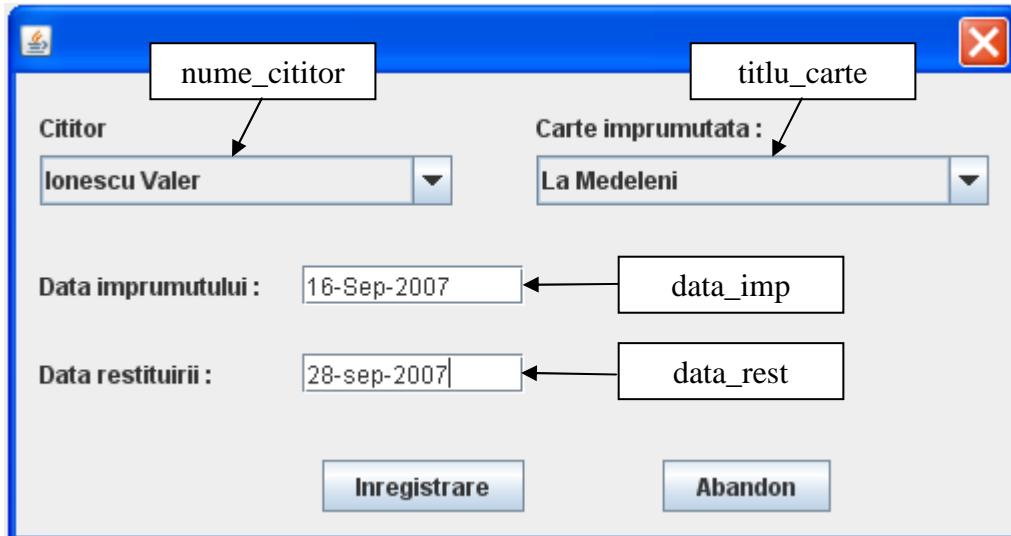
CITITORI								
Table	Data	Indexes	Model	Constraints	Grants	Statistics	UI Defaults	Triggers
Query	Count Rows	Insert Row						
EDIT	COD_CIT	CIP	HUME	PRENUME	LOCALITATEA	ID_JUD		
	3	1234567890123	Ionescu	Valer	Apahida	13		
	4	0987654321123	Pop	Vasile	Drobeta	12		

Preluarea datei calendaristice

Tabelul *IMPRUMUT* al utilizatorului *BIBLIO* conține două câmpuri de tip *DATE*.

Column Name	Data Type	Nullable	Default	Primary Key
COD_CARTE	NUMBER(7,0)	No	-	-
COD_CIT	NUMBER(6,0)	No	-	-
DATA_IMPRUMUT	DATE	No	-	-
DATA_REST	DATE	No	-	-
RESTITUITA	NUMBER(1,0)	Yes	-	-
1 - 5				

Pentru inserarea unei înregistrări în acest tabel s-a realizat formularul *Imprumuturi*.



După crearea interfeței și impunerea numelor variabilelor ca în figură, s-a scris metoda aferentă evenimentului *windowActivated*. Aceasta va popula cu date cele două controale cu listă, *nume_cititor* și *titlu_carte*. Înaintea începerii programării s-au adăugat clasei principale variabilele *impstat* și *impstatac* din clasa *Statement* respectiv *imprs* și *imprsc* din clasa *ResultSet*.

```
private void ImprumuturiWindowActivated(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    try {
        // Populez cele doua liste. Pentru ambele, multimile de selectie
        // trebuie sa ramana active pana la terminare.
        // Populez prima lista
        impstat = cnx.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        imprs = impstat.executeQuery("SELECT * from Cititori order by Nume");
        nume_cititor.removeAllItems();
        while (imprs.next()) {
            String n = imprs.getString("nume")+" "+imprs.getString("prenume");
            nume_cititor.addItem(n);
        }
        // Populez a doua lista
        impstatac = cnx.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        imprsc = impstatac.executeQuery("SELECT * from Carti order by Titlu");
        titlu_carte.removeAllItems();
        while (imprsc.next()) {
            String n = imprsc.getString("Titlu");
            titlu_carte.addItem(n);
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

```
    }
```

După selectarea unei cărți va fi înregistrată data împrumutului în caseta de text *data_imp*. Secvența de cod corespunzătoare va constitui corpul metodei asociată evenimentului *actionPerformed* declanșat de selectarea unei cărți folosind controlul *titlu_carte* (*JComboBox*).

Caseta de text *data_imp* este inițializată cu data calendaristică furnizată de sistemul de operare. Pentru aceasta s-a creat variabila locală *azi* din clasa *Calendar* apelând metoda statică *getInstance()*. În continuare s-a definit variabila *fmt* din clasa *SimpleDateFormat* destinată impunerii formatului dorit de afișare a datei calendaristice.

Șirul de caractere care va conține data calendaristică în formatul impus se obține prin apelul *fmt.format(azi.getTime())*). El va fi memorat în caseta de text *data_imp*.

Metoda se încheie prin transferul controlului intrărilor spre *data_rest* (metoda *requestFocus()*).

```
private void titlu_carteActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Calendar azi = Calendar.getInstance();
    SimpleDateFormat fmt = new SimpleDateFormat("dd-MMM-yyyy");
    data_imp.setText(fmt.format(azi.getTime()));
    data_rest.requestFocus();
}
```

Crearea unei noi înregistrări în tabelul *IMPRUMUT* se realizează printr-o comandă *INSERT* creată în metoda asociată evenimentului *actionPerformed* declanșat de apăsarea butonului "Inregistrare". Corpul acesteia este prezentat în continuare, singura remarcă fiind aceea că numărarea elementelor unei multimi de selecție începe de la 1 iar poziția primului element din lista unei casete combinate este 0. De aceea trecerea într-o multime de selecție la o linie corespunzând unei poziții *poz* din lista asociată unui control *JComboBox* se va realiza apelând *multime.absolute(poz+1)*.

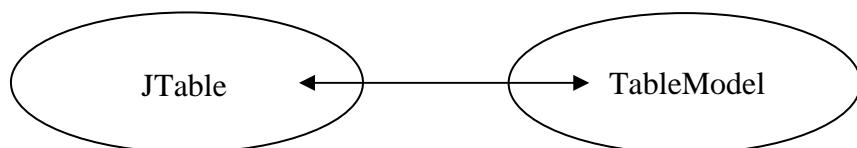
```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
        // Inserez o linie in tabelul IMPRUMUT
        // Preiau codul cititorului
        int poz = nume_cititor.getSelectedIndex();
        imprs.absolute(poz + 1); // Merg pe linia selectata
        int codcit = imprs.getInt("cod_cit");
        // Preiau codul cartii
        poz = titlu_carte.getSelectedIndex();
        imprsc.absolute(poz + 1); // Merg pe linia selectata
        int codcarte = imprsc.getInt("cod_carte");
        // Creez fraza INSERT
        String frazaSQL = "INSERT into Imprumut VALUES (" +
            frazaSQL += String.valueOf(codcarte) + "," + String.valueOf(codcit);
        frazaSQL += "," + apostrof(data_imp.getText());
        frazaSQL += "," + apostrof(data_rest.getText());
        frazaSQL += "," + 0 +")"; // 0 = FALSE, 1 = TRUE
    }
}
```

```
// Creez un Statement
Statement stm = cnx.createStatement();
// Apelez metoda executeUpdate() pt. a trimite comanda INSERT
// System.out.println(frazaSQL);
stm.executeUpdate(frazaSQL);
stm.close();
} catch (SQLException ex) {
    ex.printStackTrace();
}
}
```

JTable

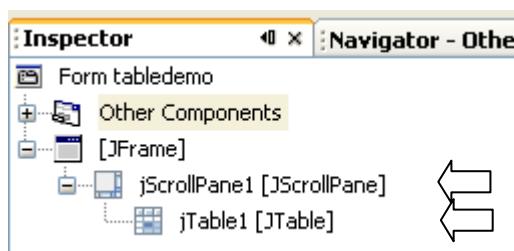
Obiectele din clasa *JTable* sunt controale Windows destinate afișării datelor sub formă tabelară. Deoarece comanda SQL *SELECT* destinată interogării serverului de baze de date furnizează datele în formă tabelară, este evident că interfața grafică a aplicației va conține astfel de controale. Controlul de tip *JTable* este însă unul deosebit de complex. Pentru păstrarea claselor și a interfețelor necesare construirii și utilizării acestor controale, în *Swing* există un pachet special, *javax.swing.table*.

Un obiect de tip *JTable* realizează doar afișarea datelor, acestea fiind conținute într-un obiect asociat (*TableModel*, de regulă un tablou bidimensional).



Înaintea creării unui obiect din clasa *JTable* este necesară crearea obiectului care conține datele. Acesta poate apartine clasei *DefaultTableModel* (variantă implicită în NetBeans) sau poate apartine unei clase definite de programator, de regulă derivată din *AbstractTableModel* (clă abstractă), aşa cum se va proceda în continuare.

În NetBeans adăugarea unui obiect din clasa *JTable* provoacă inserarea în fapt a două obiecte, un *jScrollPane* (container) și un *jTable* conținut în primul.



La crearea unei clase derivate din *AbstractTableModel* trebuie suprascrise cele trei metodele abstracte ale clasei de bază:

```

public int getRowCount();
public int getColumnCount();
public Object getValueAt(int row, int column);
  
```

Evident pot fi suprascrisă și metodele care nu sunt abstracte dacă este necesară specializarea modului lor de operare. În exemplul prezentat în continuare a fost

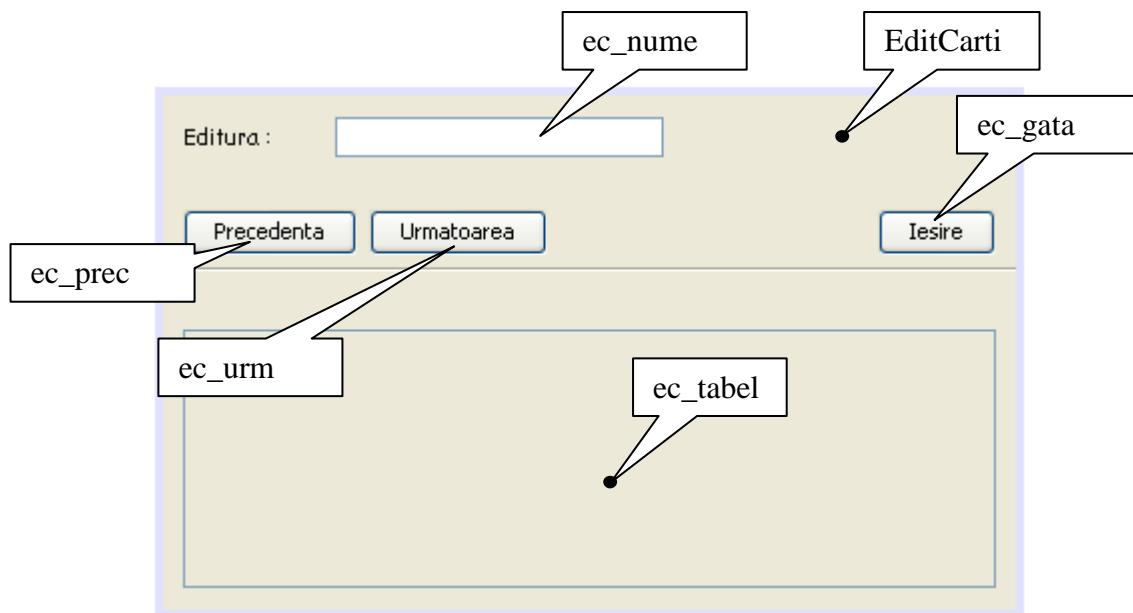
suprascrisă și metoda `getColumnName()`. Ea permite impunerea denumirilor din capul de tabel.

Exemplu fundamental:

Pentru evidențierea cărților provenind de la diferite edituri se dorește realizarea unui formular care să permită navigarea în tabelul *Edituri* și afișarea într-un tabel a înregistrărilor din tabelul *Carti* asociate înregistrării curente din tabelul *Edituri*.

Rezolvare.

1. Se adaugă aplicației formularul *EditCarti*. (obiect din clasa *JDialog*). Dispunerea și denumirile controalelor acestuia sunt cele din figura următoare.



Celoralte controale (*JLabel* și *JScrollPane*) nu li s-au schimbat numele.

Datele care vor fi conținute în tabel pot fi obținute prin executarea interogării

```
SELECT titlu, an_apar from Carti where cod_edit = coded
```

Parametrul *coded* este de tip *int* și va avea valoarea cheii primare din tabelul *Edituri*, *cod_edit*. Practic la trecerea în mulțimea de selecție care conține datele din tabelul *edituri* de pe o linie pe alta, *coded* va prelua din linia curentă a mulțimii de selecție valoarea câmpului *cod_edit* și folosind această valoare va repopula tabelul.

Datele din mulțimea de selecție creată prin comanda *SELECT* prezentată vor fi memorate într-o structură de tip listă. În Java există două clase care oferă interfață

necesară realizării de liste, *ArrayList* și *Vector*. Deoarece varianta utilizării clasei *Vector* a fost deja studiată în cadrul cursului *Programarea interfețelor utilizator*, în cele ce urmează datele care vor fi afișate cu ajutorul controlului de tip *JTable* vor fi memorate într-un obiect din clasa *ArrayList*. Pentru aceasta se va adăuga la sfârșirul clasei principale, în zona destinață declarației variabilelor clasei, declarația următoare:

```
private ArrayList dateContinute; // va deveni ArrayList de ArrayList
```

Pentru afișarea formularului realizat se va adăuga meniului derulant *Formulare* o intrare suplimentară, metoda asociată selectării acesteia realizând afișarea ferestrei *EditCarti*.

```
private void CartiEdActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    EditCarti.setLocation(120, 170);
    EditCarti.setVisible(true);
    EditCarti.pack();
}
```

Aplicația mai trebuie completată cu câteva metode asociate unor evenimente: activarea sau dezactivarea ferestrei *EditCarti* sau acționarea butoanelor din fereastră.

Deoarece repopularea tabelului ca urmare a reconstruirii mulțimii de selecție corespunzătoare unei valori a cheii primare din tabelul *Edituri*, *cod_edit* se realizează repetat, în trei metode ale clasei (la afișarea formularului și la apăsarea butoanelor *Precedenta* și *Urmatoarea*), este bine să se programeze o metodă ajutătoare. Metoda realizată a fost denumită *populareTablou()* și conținutul ei este prezentat în continuare.

```
private void populareTablou(int coded) {
    Statement carti_stmt = null;
    ResultSet rs = null;
    try {
        carti_stmt = cnx.createStatement();
        rs =
            carti_stmt.executeQuery("SELECT titlu, an_apar FROM Carti WHERE cod_edit = " +
                String.valueOf(coded));

        // Adaug valorile in ArrayList
        dateContinute = new ArrayList();
        try {
            while (rs.next()) {
                ArrayList linie = new ArrayList();
                String titlu = rs.getString("TITLU");
                int an_apar = rs.getInt("AN_APAR");
                linie.add(titlu);
                linie.add(an_apar);
                dateContinute.add(linie);
            }
        }
    }
}
```

```

        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    // Creez si impun obiectul model pentru ec_tabel
    // Obiectul apartine unei clase anonime derivata din AbstractTableModel
    ec_tabel.setModel(new AbstractTableModel() {           // Definesc clasa anonima

        private String[] colNume = {"Titlu", "Anul aparitiei"};

        public int getRowCount() {
            return dateContinute.size();
        }

        public int getColumnCount() {
            return colNume.length;
        }

        public Object getValueAt(int linie, int col) {
            // Preiau linia rowIndex
            ArrayList rand_cautat = (ArrayList) dateContinute.get(linie);
            return rand_cautat.get(col);
        }

        public String getColumnName(int col) {
            return colNume[col];
        }
    });

    carti_stmt.close();

    // Redimensionez prima coloana
    TableColumn column = null;
    column = ec_tabel.getColumnModel().getColumn(0);
    column.setPreferredWidth(300);
} catch (SQLException ex) {
    ex.printStackTrace();
}
}

```

Pentru a crea obiectul *model*, care conține datele care trebuie reprezentate cu ajutorul controlului *ec_tabel* (*JTable*) s-a definit o clasă anonimă derivată din clasa abstractă *AbstractTableModel*. Clasa abstractă suprascrie cele trei metode abstracte ale clasei *AbstractTableModel* (*getRowCount()*, *getColumnCount()* și *getValueAt()*) precum și metoda *getColumnName()*.

După impunerea conținutului tabelului *ec_tabel* se realizează redimensionarea primei coloane a tabelului. Această operație este necesară deoarece în mod implicit coloanele sunt create de lățimi egale. Fiecare coloană dintr-un *JTable* este un obiect din clasa *TableColumn*. Clasa *TableColumn* conține metode pentru preluarea sau impunerea lățimii unei coloane. În exemplul dat取得 obiectul *TableColumn* asociat primei

coloane ("Titlu") se realizează scriind `ec_tabel.getColumnModel().getColumn(0)`. Lățimea acestaia este ulterior modificată prin apelul metodei `setPreferredWidth()`.

Secvența de program care trebuie să se execute la activarea ferestrei realizează crearea mulțimii de selecție necesară navigării în tabelul *Edituri* și, după preluarea codului primei edituri, apelează metoda `populareTablou()`. Obiectele *comanda* și *rezultat*, aparținând claselor *Statement* respectiv *ResultSet*, au fost declarate la sfârșitul clasei principale, în zona destinață declarațiilor variabilelor clasei.

Apelul metodei `close()` pentru cele două obiectele va fi realizat la închiderea ferestrei, asa cum se va vedea în continuare.

La afișarea ferestrei pe ecran este necesară popularea cu date a tuturor controalelor. Secvența de cod va crea obiectele comanda și rezultat, va prelua codul primei edituri și îi va afișa numele în caseta de text `ec_num`. După aceasta va apela metoda `populareTablou()`.

```
private void EditCartiWindowActivated(java.awt.event.WindowEvent evt) {
// TODO add your handling code here:
    try {
        comanda = cnx.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        rezultat = comanda.executeQuery("SELECT * FROM edituri order by nume");
        // carti_stmt = cnx.createStatement();
        if (rezultat.next()) // Trec pe prima linie
        {
            String nm = rezultat.getString("nume");
            ec_num.setText(nm);
            int coded = rezultat.getInt("cod_edit");
            populareTablou(coded);
        }
    } catch ( SQLException e ) {
        e.printStackTrace();
        System.exit( 1 );
    }
}
```

La închiderea formularului va fi închis obiectul *comanda*.

```
private void EditCartiWindowDeactivated(java.awt.event.WindowEvent evt) {
    try {
        // TODO add your handling code here:
        comanda.close();
    } catch ( SQLException ex ) {
        ex.printStackTrace();
    }
}
```

Metodele asociate butoanelor "Precedenta" respectiv "Urmatoarea" realizează deplasarea în mulțimea de selecție conținând înregistrările din tabelul *Edituri*. La trecerea pe o nouă linie a mulțimii de selecție se preia numele editurii și se reconstruiește tabelul cu cărți prin apelul metodei *populareTablou()*.

```

private void ec_urmActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    try {
        if(!rezultat.isLast()) // nu este pe ultima linie
        {
            ec_prec.setEnabled(true);
            if (rezultat.next()) // Trec pe urmatoarea linie
            { // preiau numele
                String nm = rezultat.getString("nume");
                ec_nume.setText(nm);
                int coded = rezultat.getInt("cod_edit");
                populareTablou(coded);
            }
        } else {
            ec_urm.setEnabled(false);
        }
    } catch (SQLException esq) {}
}

private void ec_precActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    try {
        if(!rezultat.isFirst()) // nu este pe prima linie
        {
            ec_urm.setEnabled(true);
            if (rezultat.previous()) // Trec pe linia anterioara
            { // preiau numele
                String nm = rezultat.getString("nume");
                ec_nume.setText(nm);
                int coded = rezultat.getInt("cod_edit");
                populareTablou(coded);
            }
        } else {
            ec_prec.setEnabled(false);
        }
    } catch (SQLException esq) {}
}

```

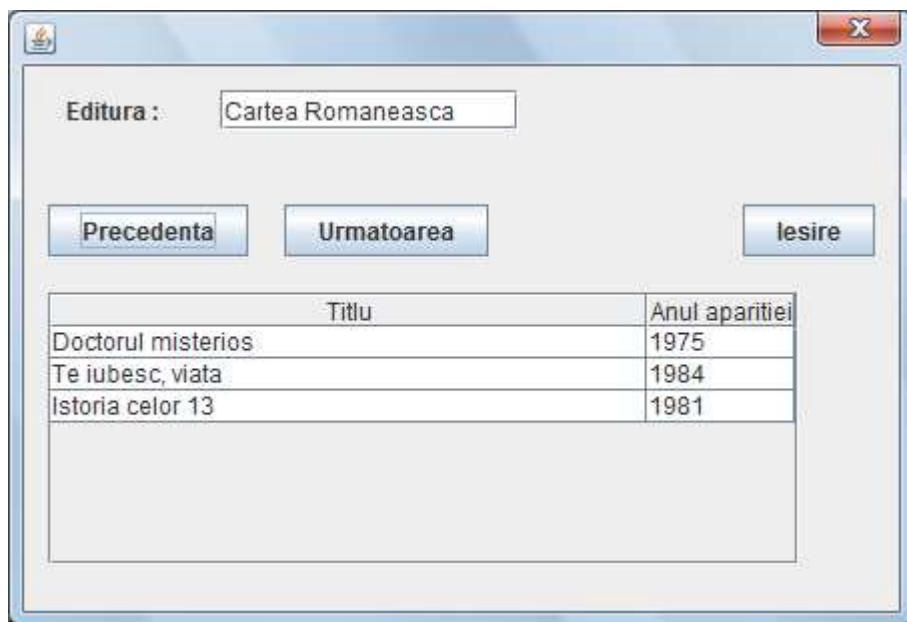
Butonul *Iesire* realizează ascunderea ferestrei *EditCarti*:

```

private void ec_gataActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    EditCarti.setVisible(false);
}

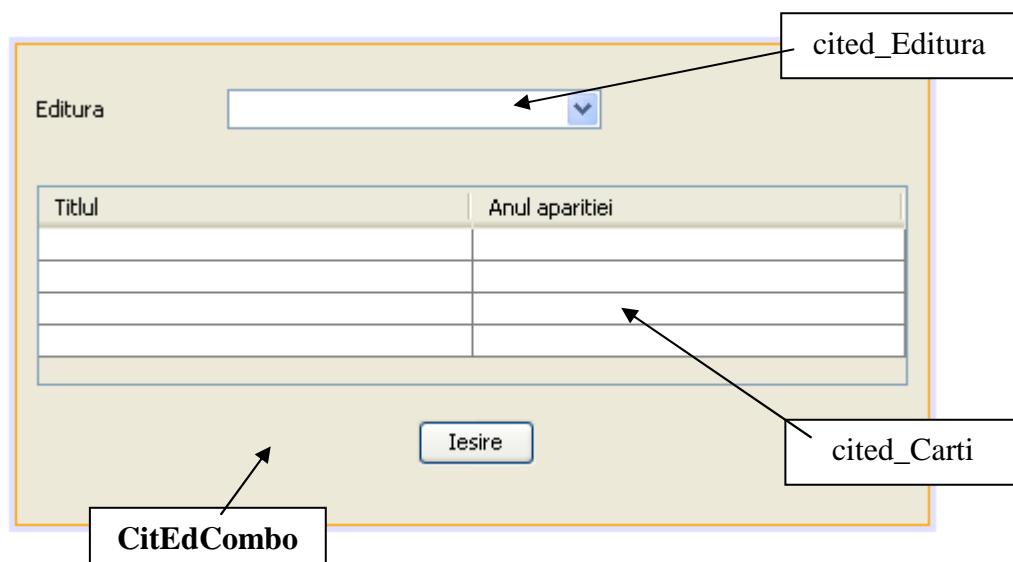
```

Rezultat posibil:



Observație

Foemularul realizat oferă o soluție pentru afișarea informațiilor din două tabele aflate în relație 1 la n. De obicei însă selectarea în primul tabel se realizează folosind o casetă combinată (*JComboBox*). Selectarea unei valori în caseta combinată provoacă repopularea controlului din clasa *JTable* în care sunt afișate datele din al doilea tabel. Exemplul precedent poate fi ușor modificat astfel încât editura să fie selectată într-un control *JComboBox*. O variantă posibilă este fereastra de dialog *CitEdCombo* prezentată în continuare.



După adăugarea controalelor și schimbarea numelor acestora ca în figură, se va scrie metoda *populareTablou1()*, similară metodei *populareTablou()*. Diferența constă în numele tabloului căruia i se atașează obiectul model.

```

public void populareTablou1(int coded) {
    Statement carti_stmt = null;
    ResultSet rs = null;
    try {
        carti_stmt = cnx.createStatement();
        rs = carti_stmt.executeQuery("SELECT titlu, an_apar FROM Carti WHERE cod_edit = " +
            String.valueOf(coded));
        // Adaug valorile in ArrayList
        dateContinute = new ArrayList();
        try {
            while (rs.next()) {
                ArrayList linie = new ArrayList();
                String titlu = rs.getString("TITLU");
                int an_apar = rs.getInt("AN_APAR");
                linie.add(titlu);
                linie.add(an_apar);
                dateContinute.add(linie);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

cited_Carti.setModel(new AbstractTableModel() {

    private String[] colNume = {"Titlu", "Anul aparitiei"};

    public int getRowCount() {
        return dateContinute.size();
    }

    public int getColumnCount() {
        return colNume.length;
    }

    public Object getValueAt(int linie, int col) {
        // Preiau linia rowIndex
        ArrayList rand_cautat = (ArrayList) dateContinute.get(linie);
        return rand_cautat.get(col);
    }

    public String getColumnName(int col) {
        return colNume[col];
    }
});
carti_stmt.close();
// Redimensionez prima coloana
TableColumn column = null;
```

```

        column = cited_Carti.getColumnModel().getColumn(0);
        column.setPreferredWidth(300);
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}

```

În continuare sunt generate și editate patru metode

1. Metoda pentru evenimentul *WindowActivated*:

```

private void CitEdComboWindowActivated(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    try {
        comanda = cnx.createStatement	ResultSet.TYPE_SCROLL_INSENSITIVE,
                           ResultSet.CONCUR_READ_ONLY);
        rs = comanda.executeQuery("SELECT * from Edituri order by Nume");
        cited>Editura.removeAllItems();
        while (rs.next()) {
            String n = rs.getString("nume");
            cited>Editura.addItem(n);
        }
        // Preiau codul primei edituri din lista
        rs.first();
        int coded = rs.getInt("Cod_Edit");
        // Populez tabelul
        populareTablou1(coded);
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}

```

Pentru popularea cu date a controlului de tip *JComboBox*, se reutilizează variabilele clasei *comanda* (*Statement*) și *rs* (*ResultSet*) declarate deja când a fost realizată fereastra de dialog precedentă.

2. Metoda pentru evenimentul *WindowDeactivated*

La închiderea formularului trebuie apelate metodele close aferente obiectelor de tip *Statement* și *ResultSet* folosite.

```

private void CitEdComboWindowDeactivated(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    try {
        comanda.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}

```

3. Metoda pentru evenimentul **ActionPerformed** (declanșat de *JComboBox*)

Selectarea unei edituri în caseta combinată declanșează un eveniment *ActionPerformed*. Metoda aferentă trebuie generată și editată astfel:



```
private void cited_EdituraActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
        // Se selecteaza codul noii edituri
        int linie = cited_Editura.getSelectedIndex();
        if (linie >= 0) {
            rs.absolute(linie + 1); // mut cursorul in cited_rs pe linie
            int coded = rs.getInt("Cod>Edit");
            // Repopulez tabelul
            populareTablou1(coded);
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

4. Metoda pentru evenimentul **ActionPerformed** (declanșat de butonul *Iesire*)

Apăsarea butonului *Iesire* provoacă ascunderea ferestrei de dialog *CitEdCombo*.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    CitEdCombo.setVisible(false);
}
```

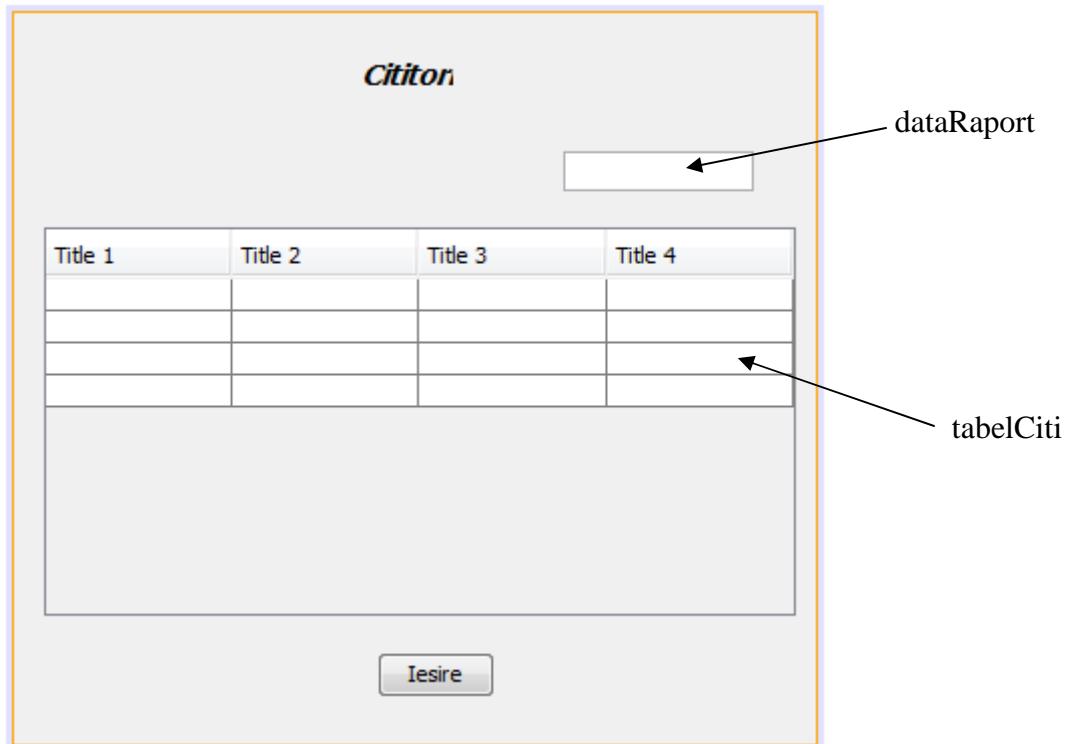
Pentru afișarea pe ecran a ferestrei realizate se va adăuga în meniul aplicației o nouă opțiune, ca și în cazul afișării ferestrei precedente.

Inserarea în *JTable* a altor tipuri de obiecte

Celulele de pe o coloană a unui tabel pot conține în principiu orice tip de obiect, mai frecvent folosite fiind pictogramele și controale de tip *JComboBox*.

Crearea unui tabel conținând astfel de obiecte necesită însă adăugarea de noi clase și sevențe de cod care asigură afișarea corectă și tratarea evenimentelor specifice.

Exemplu: Se consideră formularul **AfisareCititori** care afișează cititorii unei biblioteci:



Pentru a da posibilitate operatorului să inițieze corectarea datelor unui cititor prin simpla selectare cu mouse-ul a unei pictograme conținută în prima coloană a liniei se va proceda astfel:

1. Se adaugă aplicației o clasă nouă care va prelua sarcina afișării noului tip de dată:

```
package biblioteca;
public class Afisez {

    /** Creates a new instance of Afisez */
    public Afisez() {
    }
}
```

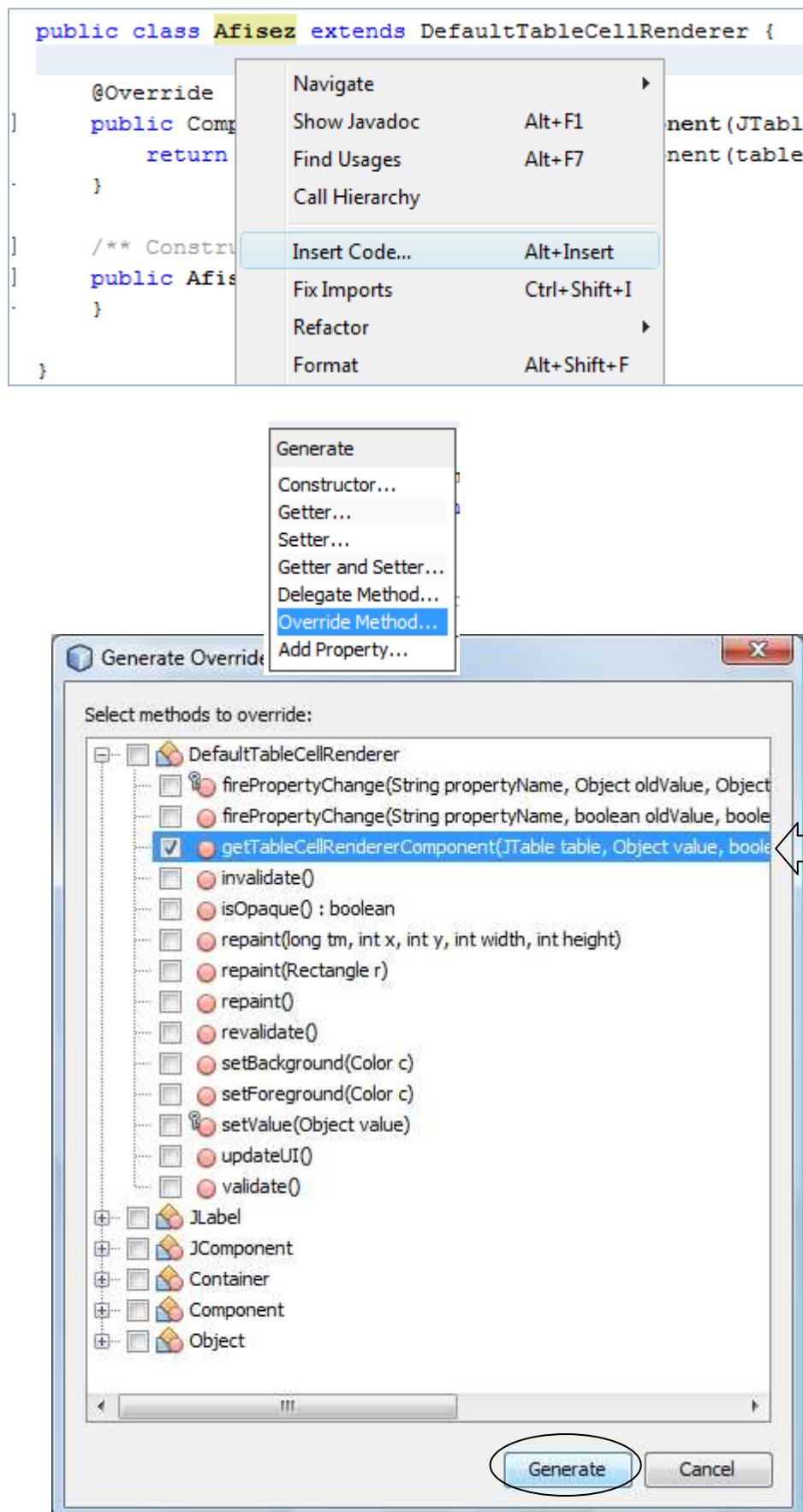
Se corectează declarația clasei astfel încât să fie derivată din *DefaultTableCellRenderer*.

```
public class Afisez extends DefaultTableCellRenderer {
```

```
    ..
```

Se elimină eroarea evidențiată de editor prin completarea listei de pachete incluse folosind opțiunea *Fix imports* a mediului de programare.

Se suprascrie metoda `getTableCellRendererComponent()` moștenită de la clasa de bază. Pentru aceasta se selectează în meniu derulant Source opțiunea *Insert Code ... / Override Methods* sau se selectează *Insert Code/ OverrideMethods* pornind din meniu contextual:



Metoda adăugată este apoi editată. Corpul acesteia va conține crearea unui obiect din clasa *ImageIcon*. Acesta va fi impus ca și conținut al celulei din tabel prin apelul metodei *setIcon()*.

```
public Component getTableCellRendererComponent(JTable table,
    Object value, boolean isSelected, boolean hasFocus, int row, int column) {
    ImageIcon icon=new ImageIcon( getClass().getResource("edit_big.gif"));
    setIcon(icon);
    return this;
}
```

Pentru testarea aplicației, fișierul referit (*edit_big.gif*) va fi plasat în directorul *biblioteca / src / evidcarti*.

2. Ca și în exemplele precedente se adaugă clasei principale o metodă, *populareTablou2()*, specializată în popularea cu date a controlului *tabelCiti* (*JTable*). Această metodă va fi similară celor deja realizate.

```
public void populareTablou2() {
    try {
        Statement comanda = cnx.createStatement	ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        ResultSet rs = comanda.executeQuery("Select * from Cititori");

        // Adaug valorile in ArrayList
        dateContinute = new ArrayList();
        try {
            while (rs.next()) {
                ArrayList linie = new ArrayList();

                String nume = rs.getString("NUME");
                String prenume = rs.getString("PRENUME");
                String cnp = rs.getString("CNP");
                linie.add(null); // aici va fi pictograma
                linie.add(nume);
                linie.add(prenume);
                linie.add(cnp);
                dateContinute.add(linie);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        comanda.close();
        tabelCiti.setModel(new AbstractTableModel() {

            private String[] colNume = {"Edit", "Nume", "Prenume", "CNP"};
            public int getRowCount() {
                return dateContinute.size();
            }
        });
    }
}
```

```

public int getColumnCount() {
    return colNume.length;
}

public Object getValueAt(int linie, int col) {
    // Preiau linia rowIndex
    ArrayList rand_cautat = (ArrayList) dateContinute.get(linie);
    return rand_cautat.get(col);
}

public String getColumnName(int col) {
    return colNume[col];
}
});

comanda.close();
// Redimensionez primele 3 coloane
TableColumn column = null;
column = tabelCiti.getColumnModel().getColumn(0);
column.setPreferredWidth(50);
column = tabelCiti.getColumnModel().getColumn(1);
column.setPreferredWidth(130);
column = tabelCiti.getColumnModel().getColumn(2);
column.setPreferredWidth(130);

} catch (SQLException ex) {
    ex.printStackTrace();
}
}
}

```

3. Se scrie funcția de tratare a evenimentului WindowActivated pentru noua fereastră.

```

private void AfisareCititoriWindowActivated(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    // Populez tabelul
    populareTablou2();
    tabelCiti.setRowHeight(32);
    tabelCiti.clearSelection();
    tabelCiti.setCellSelectionEnabled(false);
    tabelCiti.getColumnModel().getColumn(0).setCellRenderer(new Afisez());
    ListSelectionModel rowSM = tabelCiti.getSelectionModel();
    rowSM.addListSelectionListener(new ListSelectionListener() {

        public void valueChanged(ListSelectionEvent e) {
            //Ignore extra messages.
            if (e.getValueIsAdjusting()) {
                return;
            }
            ListSelectionModel lsm = (ListSelectionModel) e.getSource();
            if (!lsm.isSelectionEmpty()) {
                int selectedRow = lsm.getMinSelectionIndex();
                //selectedRow is selected
                int coloana =
                    tabelCiti.getColumnModel().getSelectionModel().getAnchorSelectionIndex();
                if (coloana == 0) {
                    String s = "Celula " + selectedRow + ", " + coloana + " e selectata.";

```

```

        JOptionPane.showMessageDialog(null, s); // Se inlocuieste
    }
    tabelCiti.clearSelection();
}
});
Calendar azi = Calendar.getInstance();
SimpleDateFormat fmt = new SimpleDateFormat("dd-MM-yyyy");
dataRaport.setText(fmt.format(azi.getTime()));
}

```

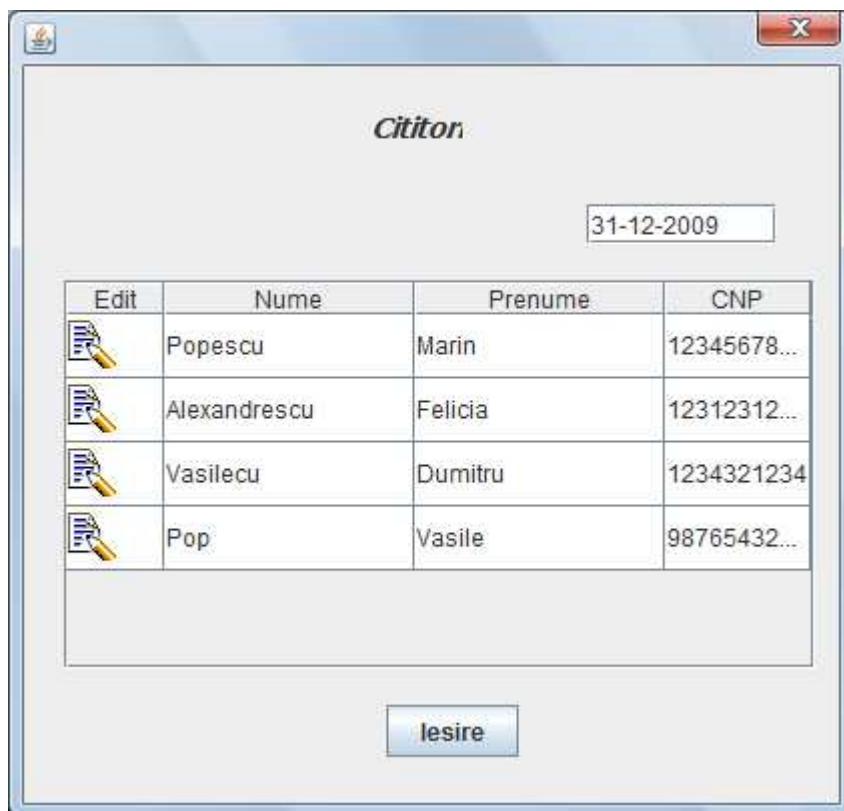
- 4.** Se adaugă în meniul ferestrei principale o nouă intrare. Selectarea acesteia va declanșa afișarea noii ferestre.

```

private void afisatiActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    AfisareCititori.setLocation(120, 170);
    AfisareCititori.setVisible(true);
    AfisareCititori.pack();
}

```

Rezultat posibil:



Realizarea rapoartelor

Rapoartele prezintă într-un format impus date conținute în baza de date. În unele cazuri formatul în care datele trebuie prezentate este definit prin lege.

Aplicațiile care acceseză servere de baze de date trebuie să asigure afișarea pe ecran a rapoartelor și imprimarea lor.

Un raport conține de regulă nouă regiuni dispuse în benzi, ca în figură. Lățimile benzilor sunt astfel stabilite încât să ocupe întreaga zonă utilă (imprimabilă) a paginii.

Titlul (<i>Title</i>)
Cap pagină (<i>Page Header</i>)
Cap coloană (<i>Column Header</i>)
Detaliere (<i>Detail</i>)
Picior coloană (<i>Column Footer</i>)
Picior pagină (<i>Page Footer</i>)
Recapitulare (<i>Summary</i>)

Titlul

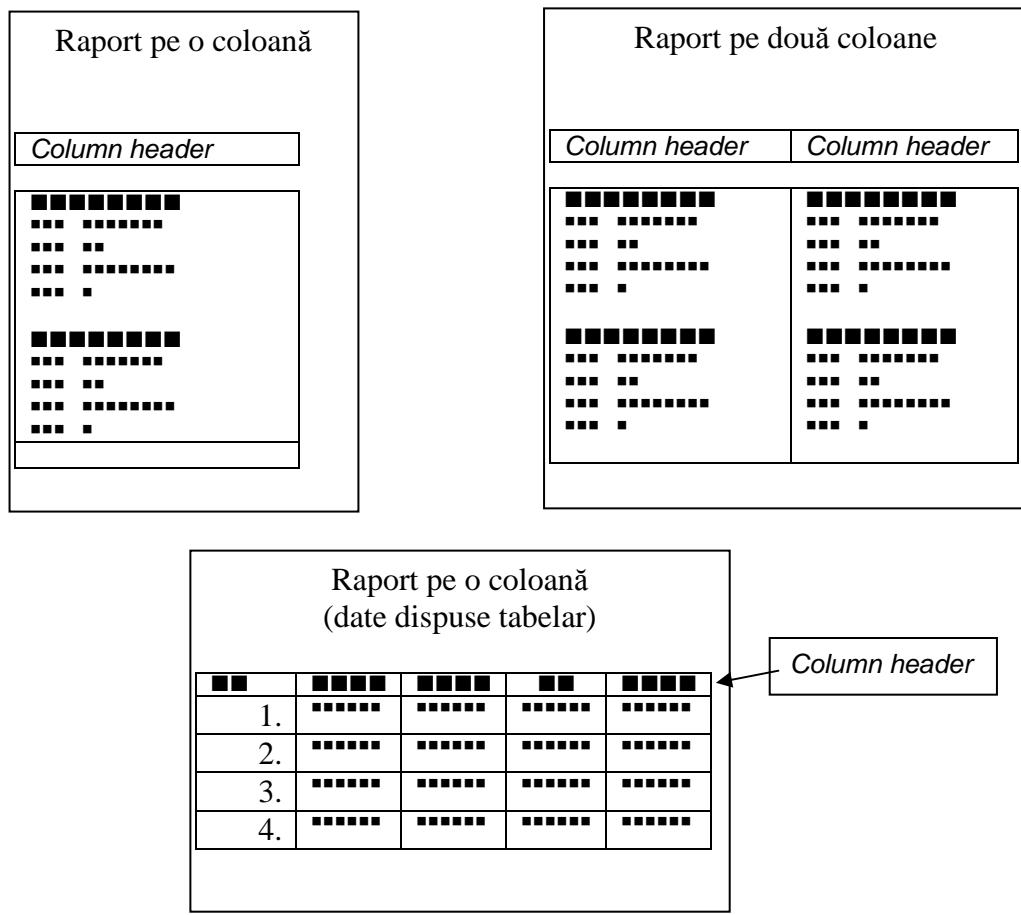
Banda pentru titlu este prima dintr-un raport. Un raport conține numai o singură bandă pentru titlu. Aceasta poate fi dispusă pe o pagină separată.

Capul paginii

Banda pentru capul paginii apare la începutul fiecărei pagini a raportului. Dacă titlul raportului sau recapitularea apar pe pagini separate, acestea nu vor conține această regiune.

Capul coloanei

Informațiile dintr-un raport pot fi dispuse pe una sau mai multe coloane.



Detaliere

În cazul rapoartelor în care informația nu este grupată, în această regiune sunt afișate rânduri conținând date preluate din baza de date.

În cazul rapoartelor în care informația apare organizată în grupuri, în această regiune vor alterna benzi conținând antetul grupului (*Group Header*) respectiv datele conținute în grup. După fiecare grup este posibilă afișarea unei benzi (*Group Footer*) conținând un rezumat legat de datele conținute în grupul respectiv.

Picioare coloană

Banda care încheie coloana poate conține informații recapitulative, de regulă sume sau numărul de articole conținute în regiunea de detaliere.

Picioare pagină

Toate paginile care conțin o bandă destinată capului paginii pot conține în partea de jos o bandă pentru informații recapitulative la nivel de pagină. Este posibilă includerea unei benzi distințe pentru ultima pagină a raportului (*Last Page Footer*).

Recapitulare

Banda pentru recapitulare (*Summary*) este adăugată la sfârșitul raportului, eventual pe o pagină distinctă.

Fundal

Fundalul raportului (*Background*) poate fi impus. De regulă acesta constă dintr-o imagine care va fi repetată pe toate paginile raportului.

Jasper Reports

Realizarea comodă a rapoartelor se poate face folosind o aplicație specializată. Există o multitudine de astfel de aplicații, mai interesantă fiind *Jasper Reports*, o soluție gratuită (*open source*) propusă de *JasperSoft Corporation* (fondată de românul Teodor Danciu).

Jasper Reports constă dintr-o colecție de clase Java care pot fi adăugate aplicațiilor scrise în acest limbaj. *Jasper Reports* oferă de altfel o mulțime de funcții. Astfel, pe lângă vizualizarea și imprimarea imediată a raportului, datele acestuia pot fi exportate într-o multitudine de formate: HTML, PDF, XML, XLS (*Microsoft Excel*), RTF (*Microsoft Word*) sau reprezentate grafic.

Principial un ciclu de lucru pentru crearea unui raport folosind *Jasper Reports* constă din pași următori:

1. Se crează un fișier de descriere a raportului în format XML, având extensia *.jrxml*;
2. Fișierul *.jrxml* este compilat pentru a-l aduce într-o formă accesibilă claselor *Jasper Reports*;
3. Se realizează interogarea sursei de date pentru aducerea datelor care vor fi conținute în raport;
4. Se vizualizează și se imprimă raportul.

Deoarece fișierul de descriere a raportului în format XML este relativ dificil de creat, mai ales pentru rapoarte având o grafică îngrijită și o structură nu tocmai banală, în practică se poate folosi o aplicație independentă, *iReport* sau se poate adăuga mediului Netbeans o extensie care afișează interfața acestei aplicații. Rezultatul lucrului cu *iReport*

sau cu extensia integrată în Netbeans este fișierul care conține descrierea raportului, în format *.jrxml*.

Realizarea descrierii raportului cu iReport

iReport este accesibil sub forma unei extensii (*plugin*) a mediului de programare Netbeans.

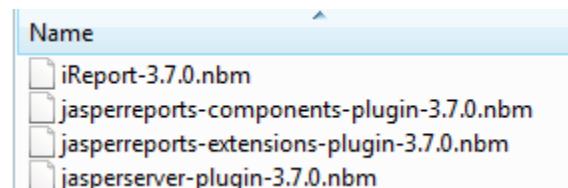
Pentru a descărca fișierul care conține această extensie se accesează saitul <http://sourceforge.net/projects/ireport/files/> și se descarcă fișierele *iReport-3.7.0-plugin.zip* și *iReport-3.7.0.zip* (sau o versiune ulterioară). Versiunile menționate pot fi descărcate din pagina cursului de pe saitul www.infonet.utcluj.ro (subdirectorul *curs11*).

Browse Files for iReport-Designer for JasperReports						
File/Folder Name	Platform	Size	Date ↓	Downloads	Notes/Subscribe	
Newest Files						
iReport-3.7.0-plugin.zip		50.2 MB	2009-12-09	2,374		
iReport-3.7.0.zip	others	75.7 MB	2009-12-09	1,975		

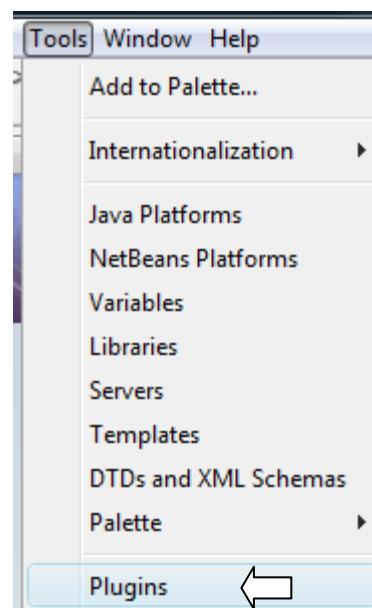
După descărcare se dezarchivează ambele fișiere.

Instalarea extensiei *iReport-3.7.0-plugin* descărcate se face astfel:

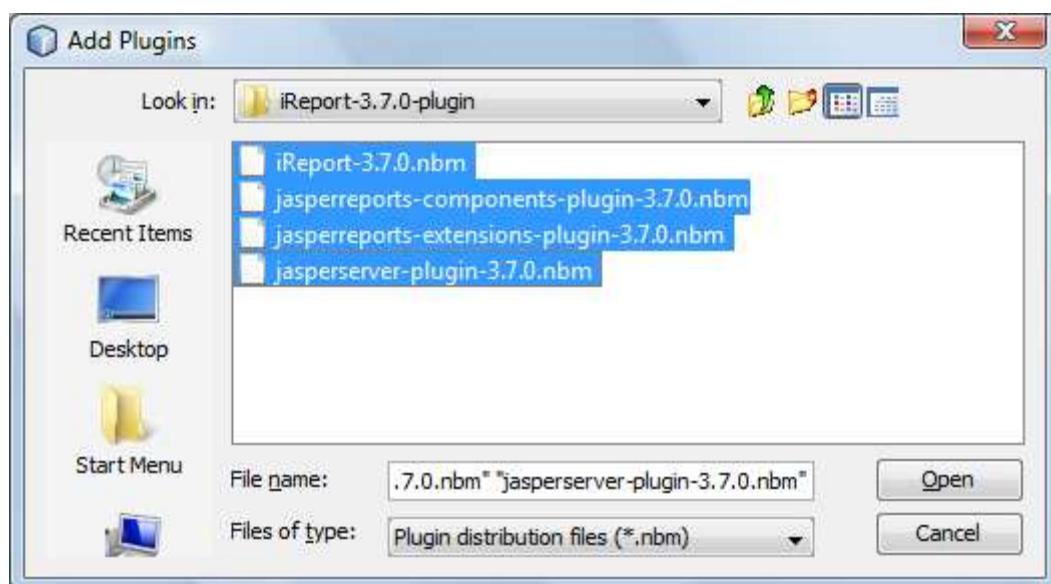
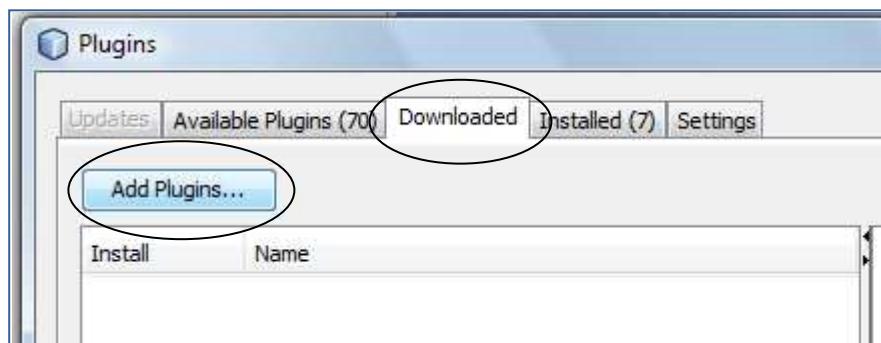
În directorul în care s-a dezarchivat *iReport-3.7.0-plugin.zip* vor apărea câteva fișiere având extensia *.nbm*.



se pornește mediul de programare *Netbeans* și se selectează *Tools / Plugins*:



În fereastra afişată (*Plugins*) se selectează tabul *Downloaded* și apoi *Add Plugins...*

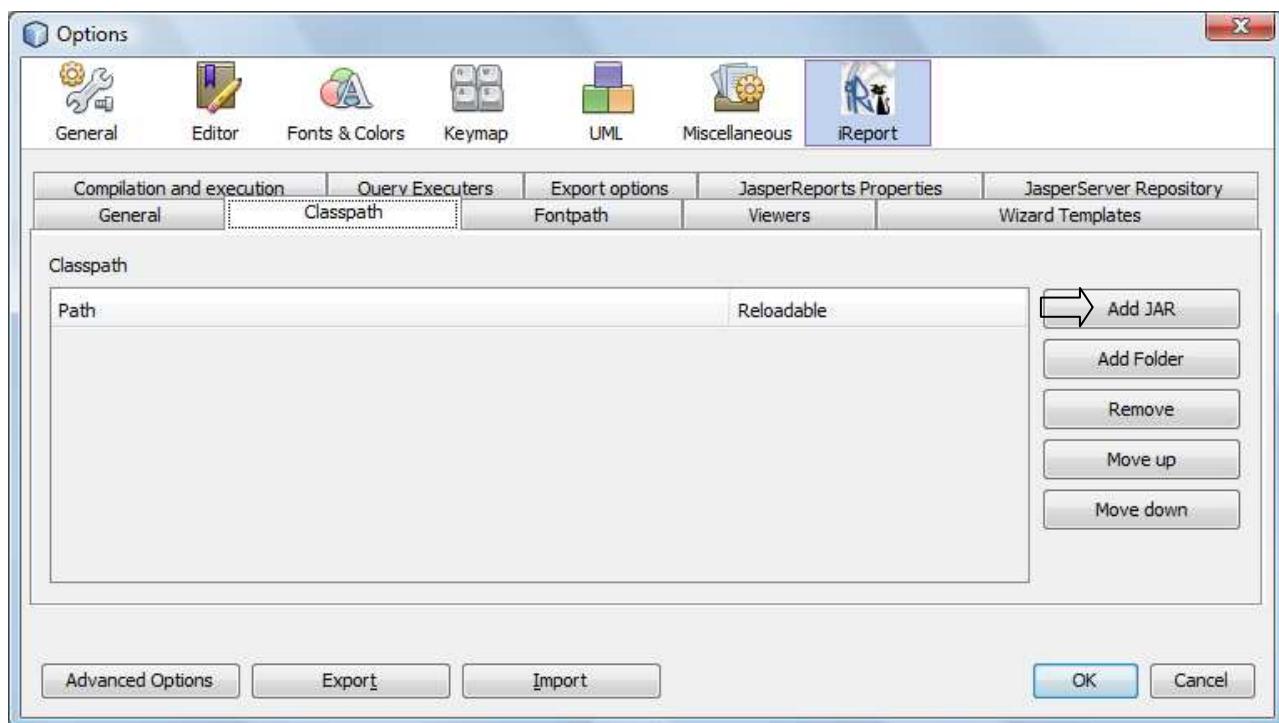


După selectarea fișierelor și apăsarea butonului *Open* se va reveni în fereastra *Plugins* în care se va selecta butonul *Install*.

După realizarea instalării și repornirea mediului de programare se poate începe realizarea unui raport.

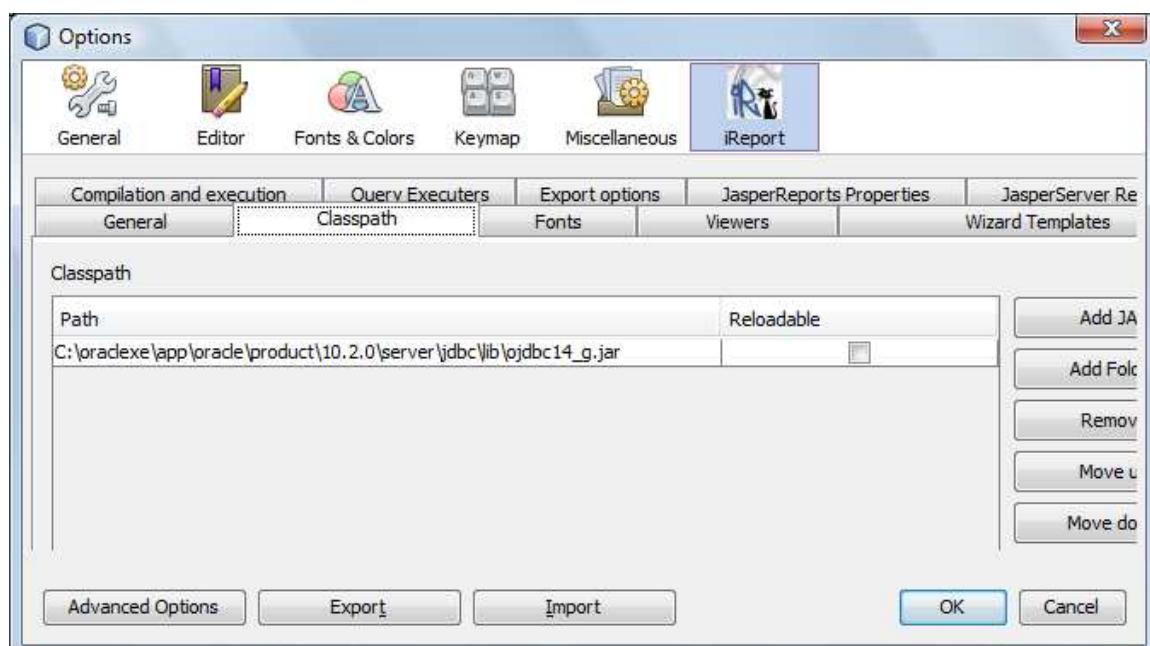
Realizarea unui raport cu iReport

Pentru a putea realiza un raport se va configura mai întâi modul de conectare al aplicației *iReport* cu baza de date. Pentru aceasta se va selecta *Tools / Options*.



În fereastra afișată se va selecta tabul *iReport* și apoi *Classpath*. Se va apăsa apoi butonul *Add JAR* și, folosind fereastra afișată, se va indica fișierul *ojdbc14_g.jar*. din directorul aplicației Oracle XE (aşa cum s-a mai făcut odată în cursul 7).

Rezultat:



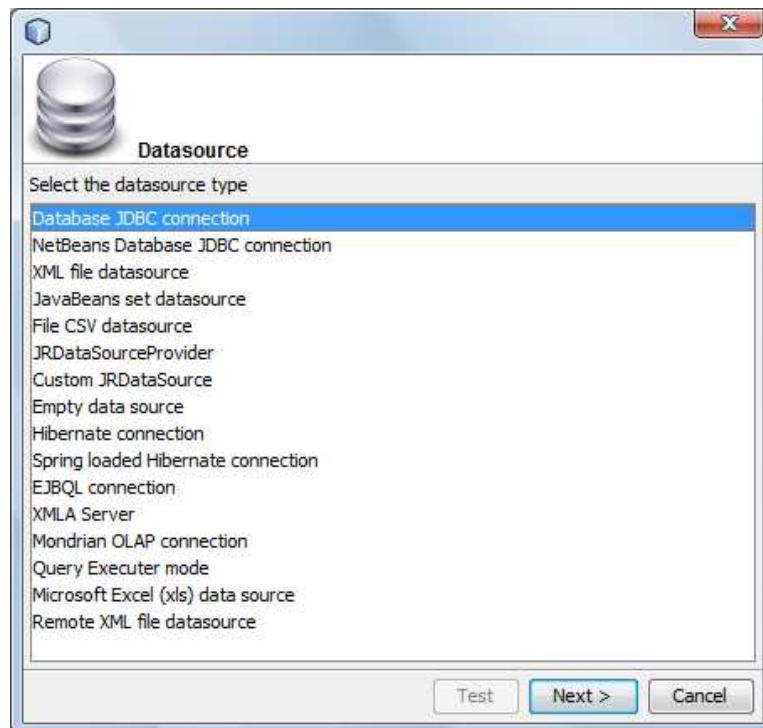
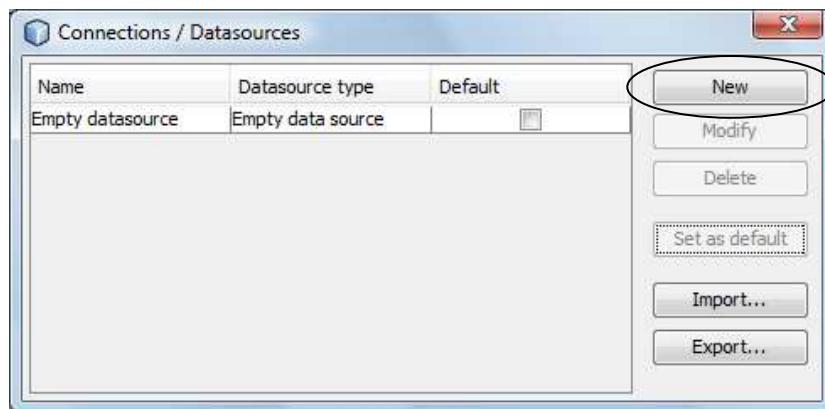
Realizarea raportului

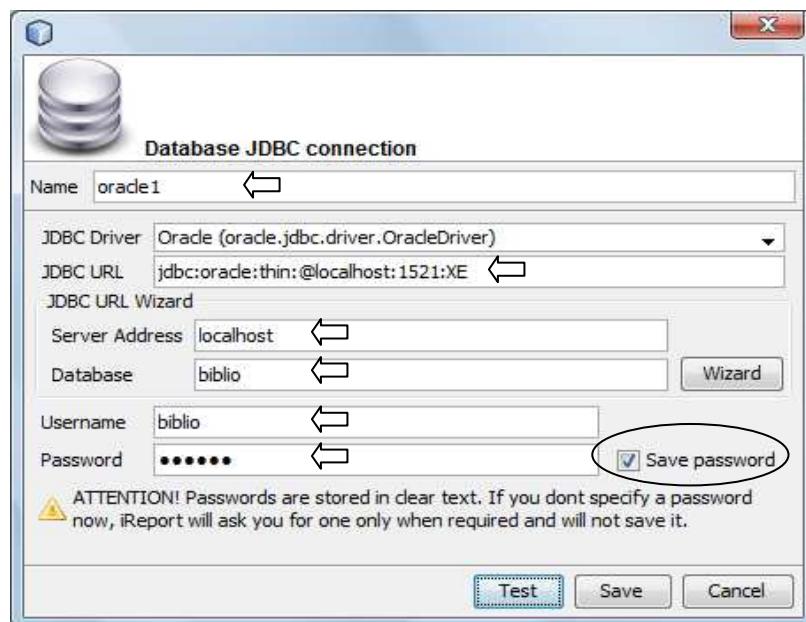
Înainte de începerea realizării rapoartelor aplicației trebuie definită o sursă de date. Practic în acest mod se va crea o legătură cu serverul de baze de date folosit.

Pentru crearea sursei de date se va selecta pictograma *Report Datasources*:

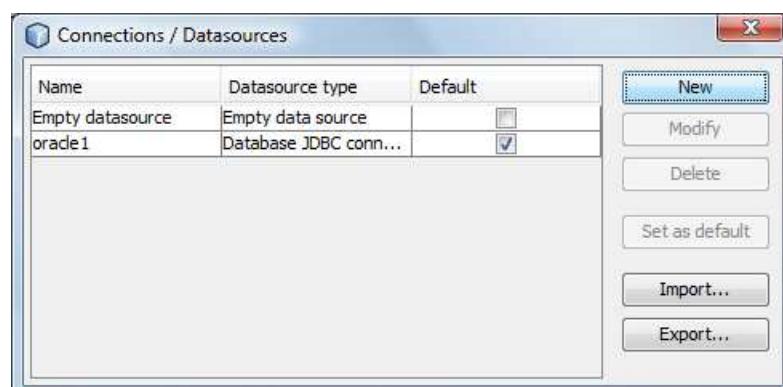


Pașii stabilirii sursei de date sunt următorii:

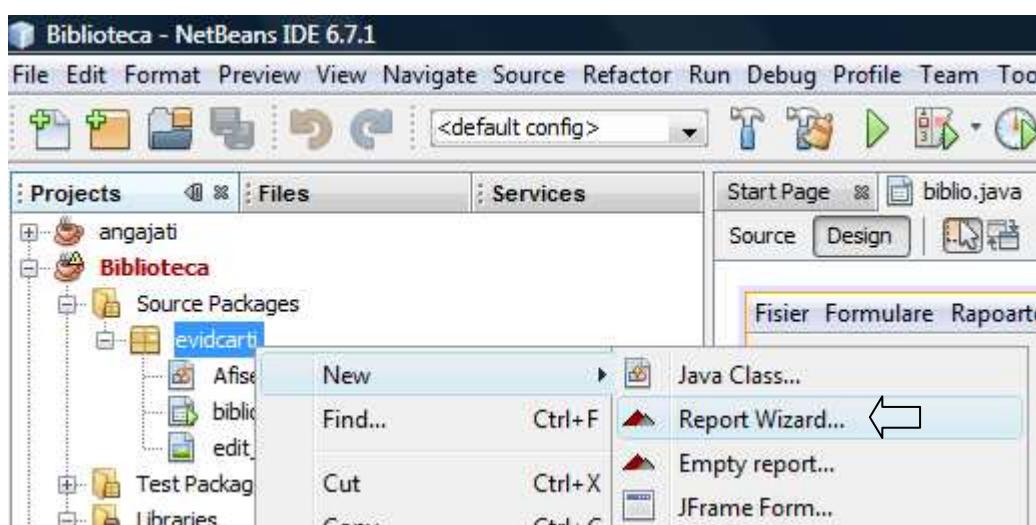




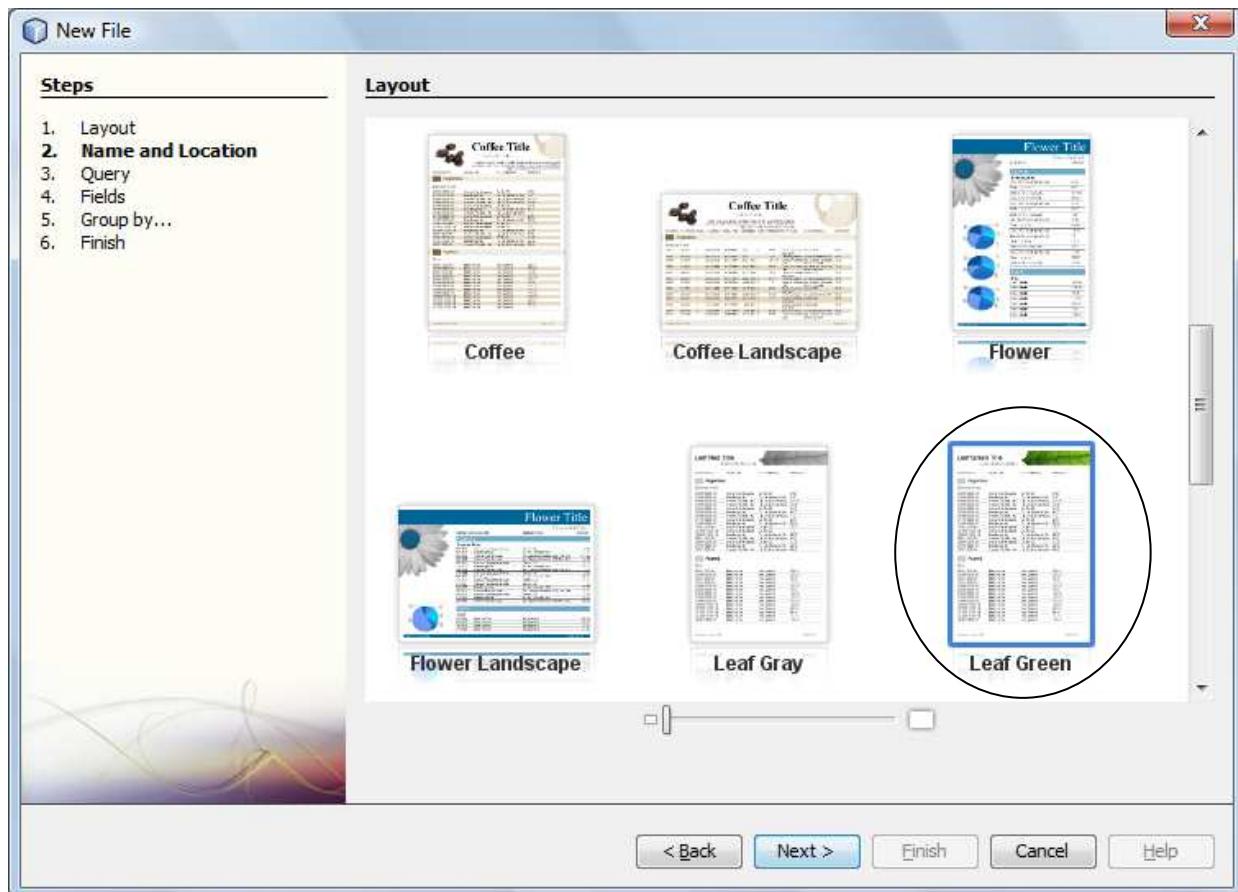
Rezultat:



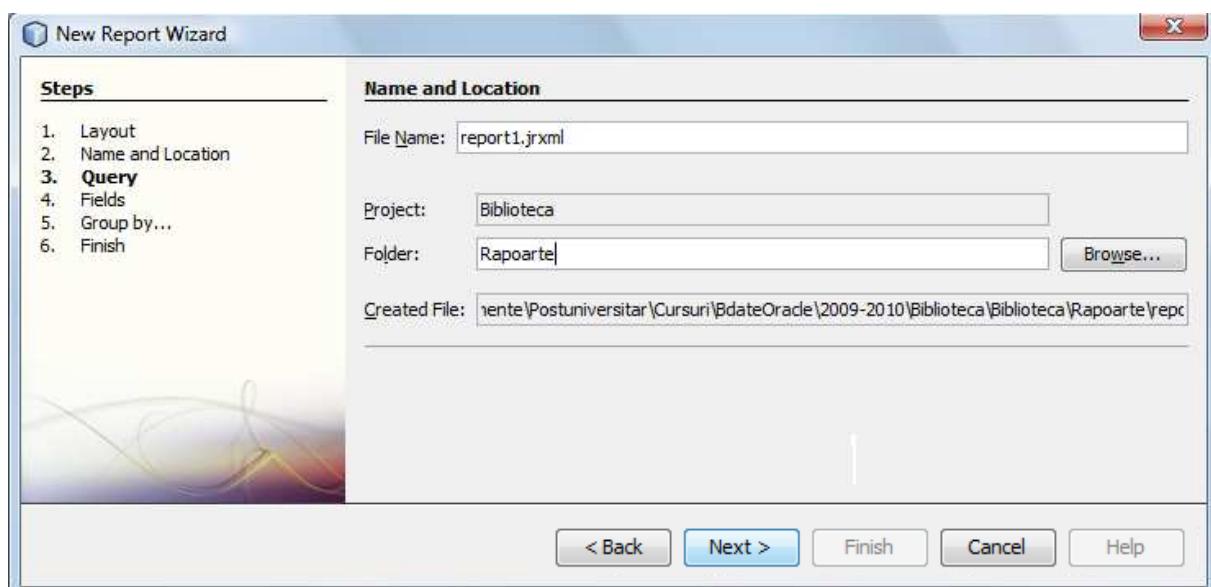
Pentru a începe realizarea raportului se va selecta cu butonul drept al mouse-ului directorul în care sunt plasate fișierele sursă (evidcarti în exemplul din figură) și în meniu contextual se selectează *New / Report Wizard....*



În fereastra mediului va fi afișată interfața aplicației expert cu ajutorul căreia va fi generat raportul. În primul ecran se alege o machetă (*layout*) pentru raportul care va fi realizat.

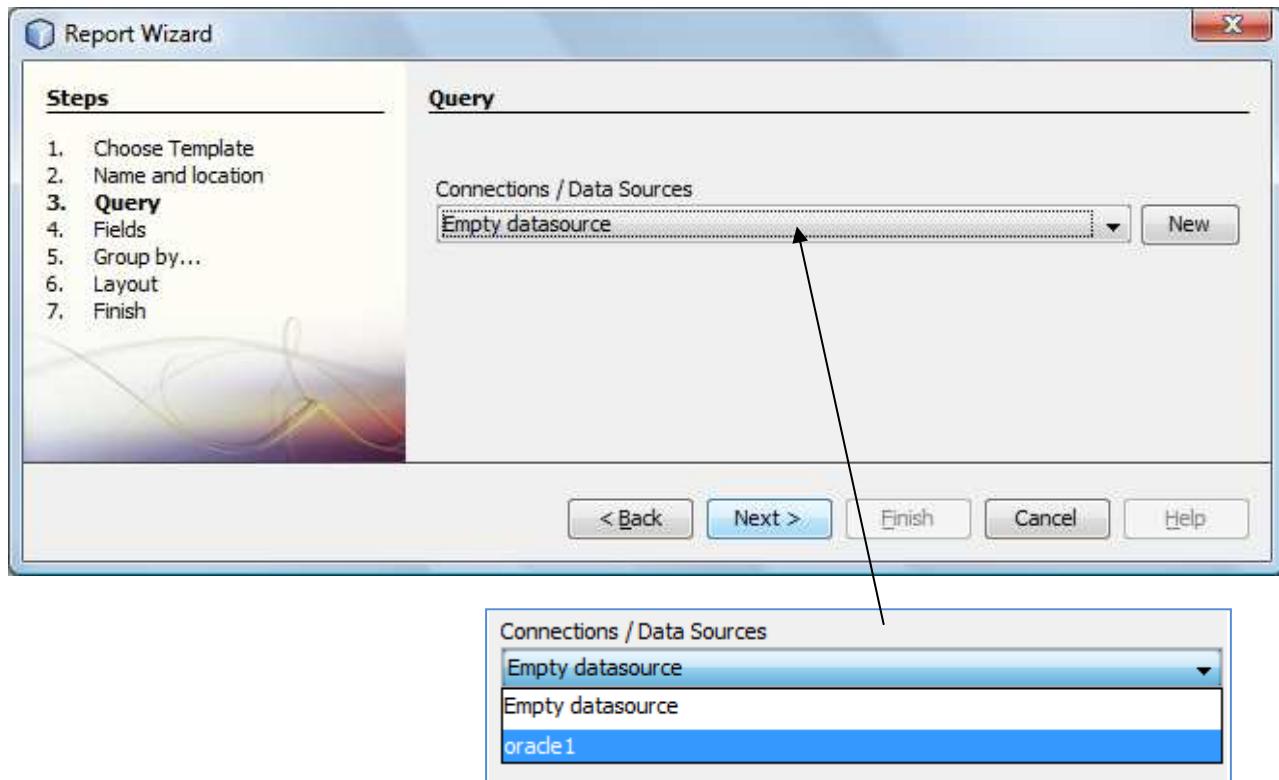


În pasul următor (după apăsarea butonului *Next*) se stabilește numele și directorul în care se va păstra fișierul care va conține descrierea raportului. Pentru aceasta în directorul rădăcină al proiectului s-a creat directorul derivat *Rapoarte*.

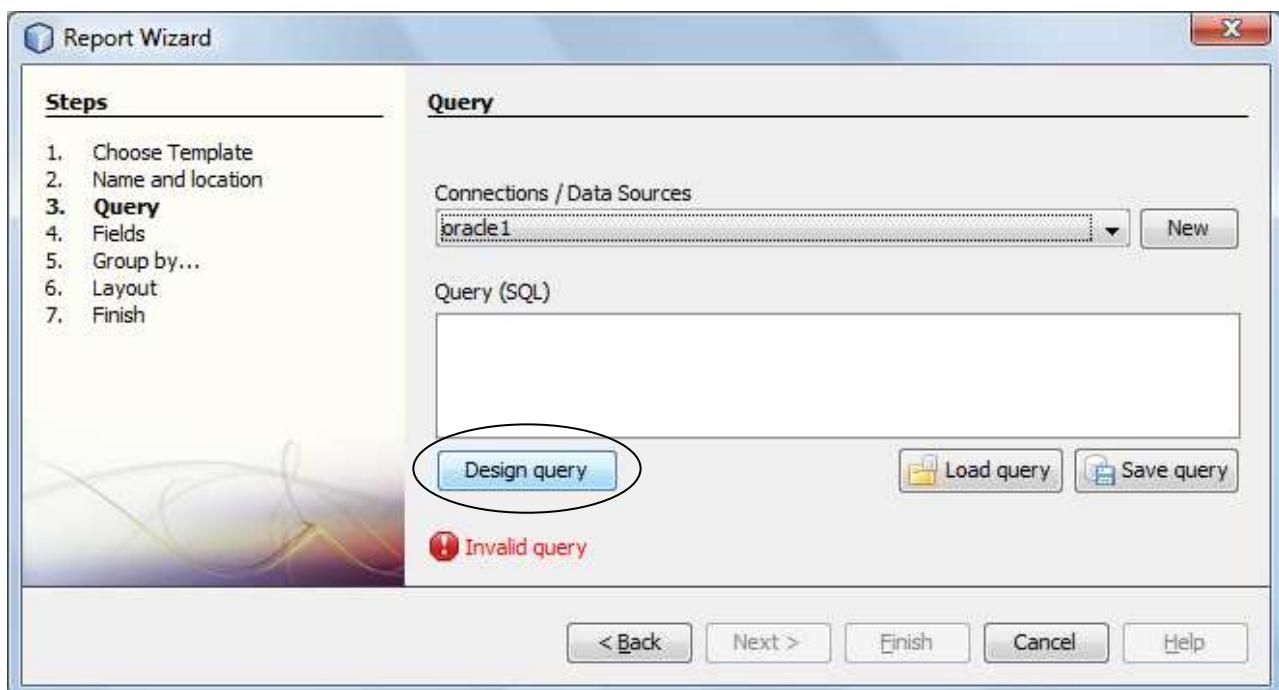


Ca nume a fost lăsat cel implicit, *report1.jrxml*.

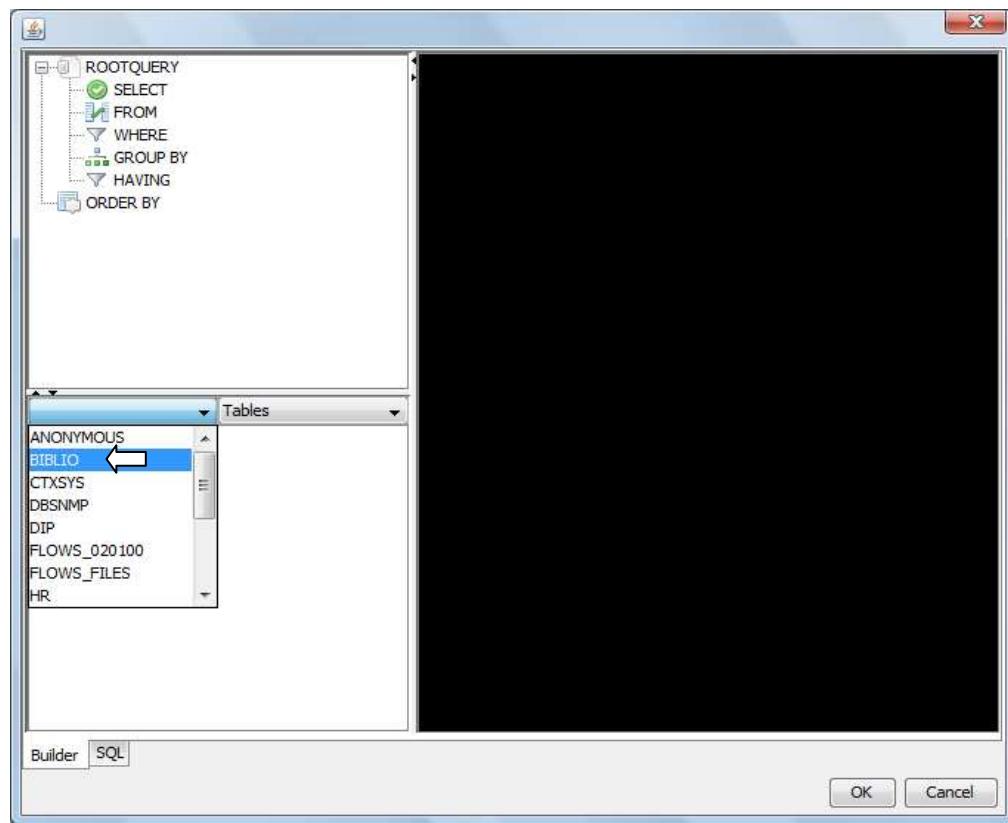
În pasul următor se va selecta sursa de date definită deja (*oracle1*).



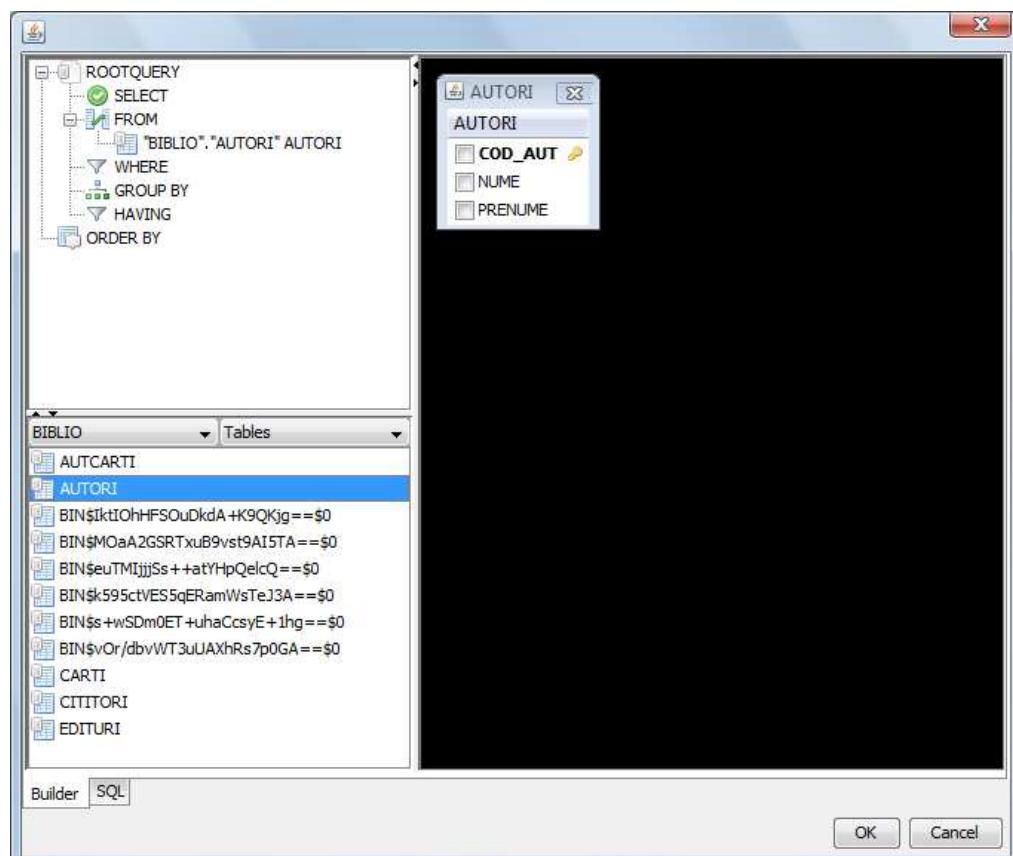
După selectarea sursei de date a raportului interfață afișată se modifică apărând o zonă de text în care ar trebui să scriem o frază SQL care extrage din baza de date mulțimea de selecție care trebuie prezentată de raport. O soluție mai simplă este apăsarea butonului *Design query* și crearea frazei SQL folosind fereastra care se afișează.



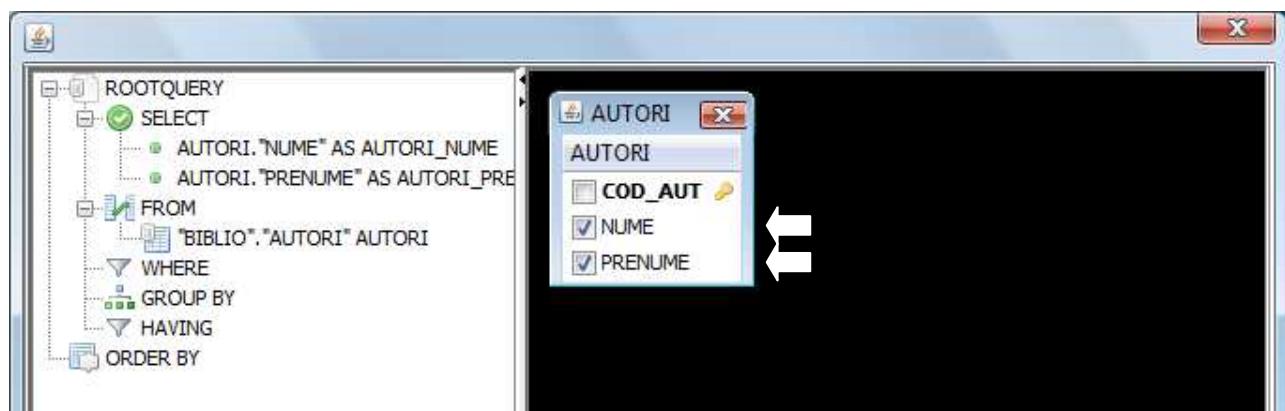
În fereastra afişată după apăsarea *Design query* se selectează mai întâi schema (*biblio*):



În fereastra *Tables* vor apărea tabelele și vederile schemei *biblio*. Cele necesare scrierii comenzi SELECT vor fi adăugate în panoul din dreapta prin selectare cu dublu clic.

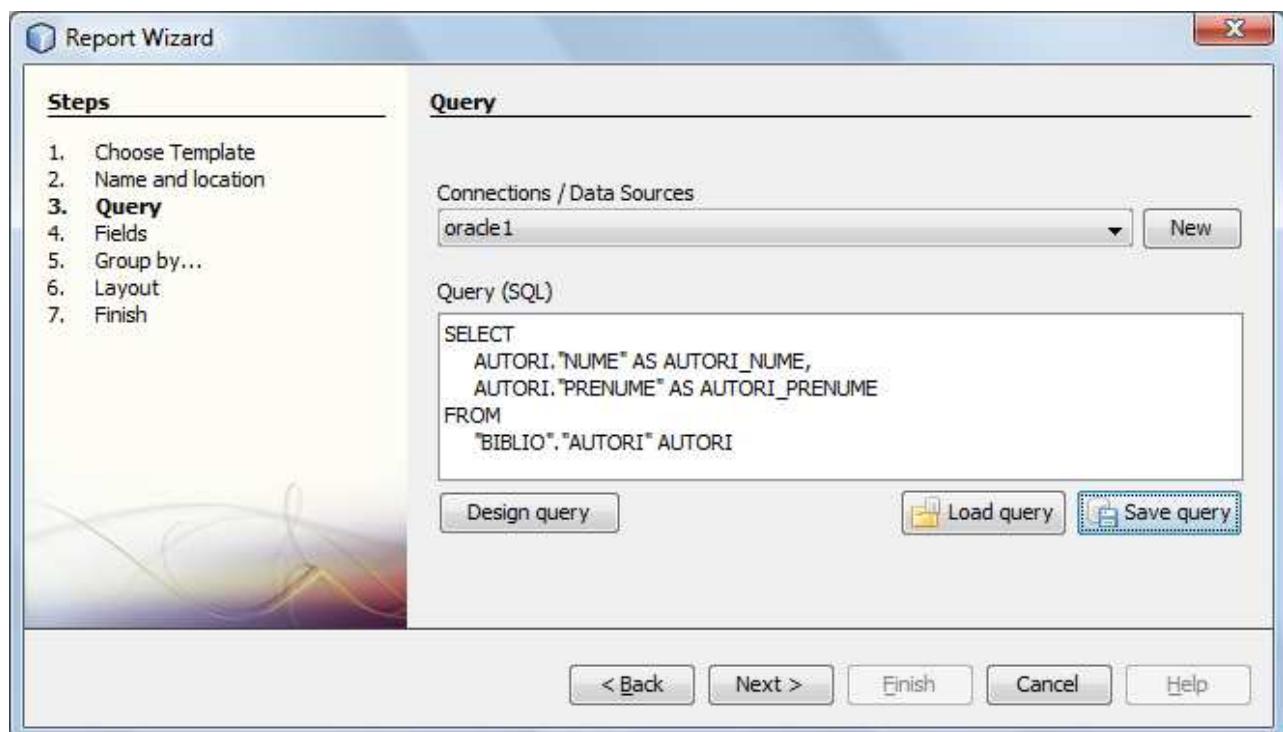


Câmpurile care vor fi utilizate în *SELECT* vor fi indicate prin selectarea casetelor de selecție din dreptul lor, ca în exemplu:

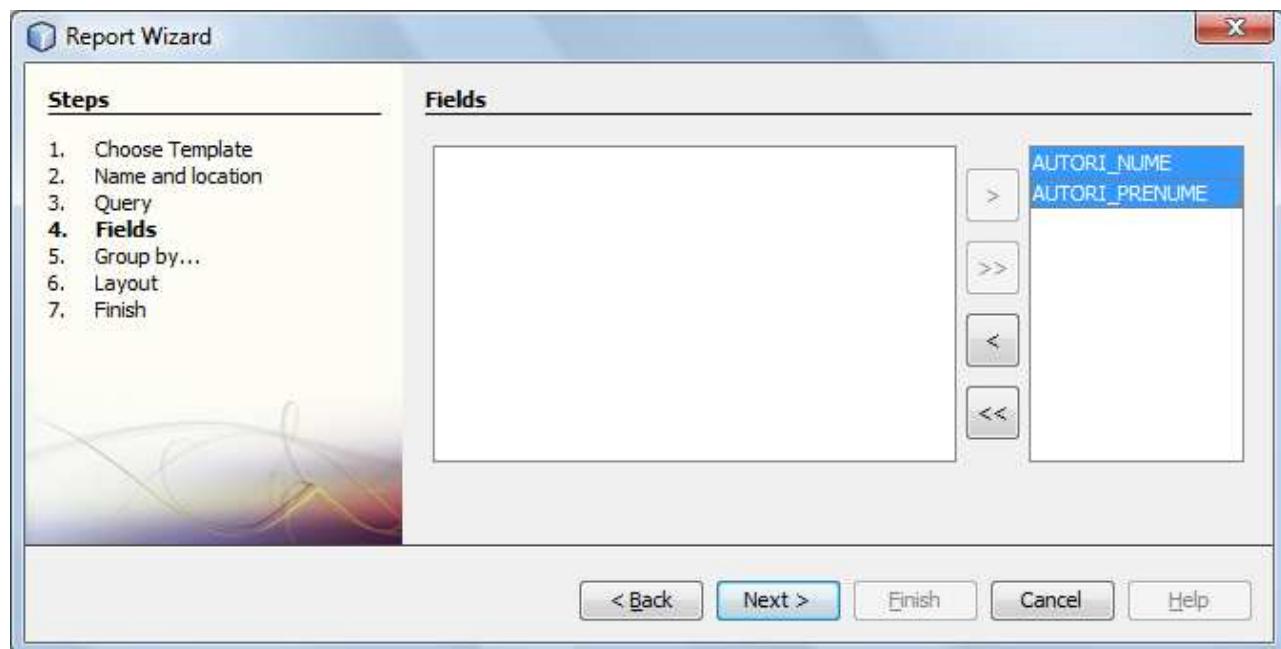


Tot folosind interfața din fereastra afișată se pot adăuga și celelalte clauze ale comenzi SQL *SELECT*. Anumite clauze sunt mai greu de impus folosind interfața afișată dar comanda SQL generată va putea fi corectată ulterior.

După validarea soluției se revine la fereastra *Report Wizard* în care se va vedea comanda *SELECT* construită.



După ce se apasă *Next* se vor putea selecta câmpurile pe care dorim să le includem în raport.



În continuare se apasă butonul *Next* până se încheie execuția aplicației expert.

Ca urmare a încheierii generării raportului, în fereastra grafică a mediului de programare apare o reprezentare a raportului în care sunt evidențiate regiunile acestuia : *Title*, *Page Header*, *Column Header*, *Detail1* etc.

The screenshot shows the report designer interface. At the top, there are tabs for 'Designer', 'XML', and 'Preview', along with various icons and a font selection dropdown set to 'DejaVu Sans'. Below the tabs is a horizontal ruler from 0 to 450. The main area contains a green leaf graphic. A table is being designed with two columns. The first column is labeled 'AUTORI_NUME' and contains the expression '\$F{AUTORI_NUME}'. The second column is labeled 'AUTORI_PRENUME' and also contains the expression '\$F{AUTORI_PRENUME}'. Below the table, there are sections labeled 'Column Header', 'Column Footer', and 'Page Footer'. The page footer section contains the expression 'new java.util.Date()'. The left side of the interface has a vertical ruler from 0 to 100.

Selectând *Preview* putem vedea imediat raportul:

AUTORI_NUME	AUTORI_PRENume
Puskin	Alexandr
Steinbeck	John
Steinhardt	Nicolae
Marcel	Proust

 The preview window has a toolbar at the top with buttons for Designer, XML, Preview, and various report navigation and search functions."/>

În continuare se pot edita elementele conținute în diferitele regiuni ale raportului. Pentru a edita un element conținut în raport se va selecta elementul și apoi se vor accesa și eventual modifica proprietățile acestuia.

Obiectele conținute în raport pot fi selectate și din fereastra afișată accesând *Report Inspector*.

În raportul generat se pot redimensiona și chiar rearanja elementele conținute. Aceste modificări se realizează cu folosind mouse-ul.

Câmpuri, Parametri, variabile

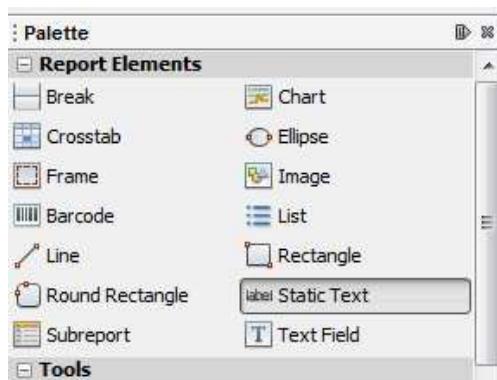
O casetă de text dintr-un raport poate conține:

- un sir simplu de caractere,
- un câmp dintr-un tabel sau dintr-o vedere,
- variabilă,
- un parametru sau,
- expresie.

În raportul realizat se pot adăuga elemente noi folosind instrumentele din panoul *Reports Elements* (în coloana din dreapta a mediului de programare). Adăugarea unui element nou se realizează prin simpla târâire cu mouse-ul a elementului dorit din panoul *Reports Elements*.

Adăugarea unui sir simplu de caractere

Dacă se dorește adăugarea în raportul început a unui antet se poate adăuga în banda de titlu o casetă de tip *Static Text*.



După adăugarea noului element se realizează configurarea acestuia (proprietăți *Font name*, *Size*, *Text* etc.).

The screenshot displays the 'Designer' view and the 'Properties' panel. In the Designer, a static text box containing the text 'Biblioteca Octavian' is selected and highlighted with a yellow border. In the Properties panel, under the 'Text' section, the font is set to 'Octavian Goga' and the size to '10'. Other properties like 'Print When Group Changes' and 'Font name' are also visible.

Câmpuri

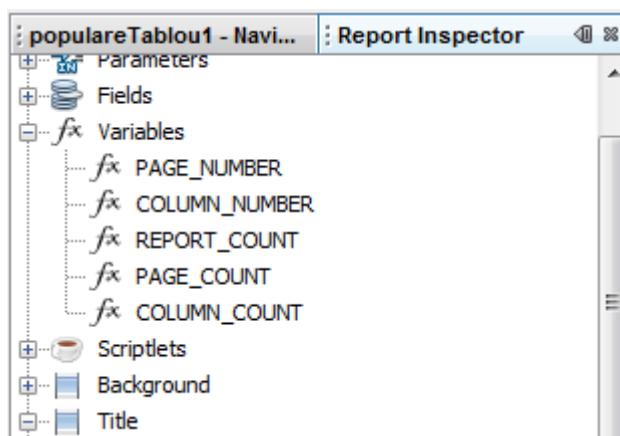
Câmpurile (*fields*) sunt incluse prin intermediul frazei SELECT care stă la baza realizării raportului. Clasa Java care le este asociată implicit depinde de natura informației din coloana tabelului sau vederii din care provin. Clasele posibile sunt următoarele:

java.lang.Boolean	java.lang.Long
java.lang.Byte	java.lang.Short
java.util.Date	java.math.BigDecimal
java.sql.Timestamp	java.lang.String
java.sql.Time	java.lang.Number
java.lang.Double	
java.lang.Float	
java.lang.Integer	

Cele mai folosite sunt *java.lang.String* (asociată tipurilor SQL *CHAR* sau *VARCHAR2*), *java.math.BigDecimal* (asociată implicit tipului SQL *NUMBER*) și *java.util.Date* (asociat tipului SQL *DATE*). Pentru referirea la câmpuri se folosește sintaxa *\$F{nume_camp}*.

Variabile

Variabilele servesc la producerea unor valori care variază de la o linie la alta (un contor de linii de exemplu) sau a unor valori calculate pe baza valorilor câmpurilor (sume, medii, valori extreme etc.). iReports definește implicit un număr de 5 variabile.

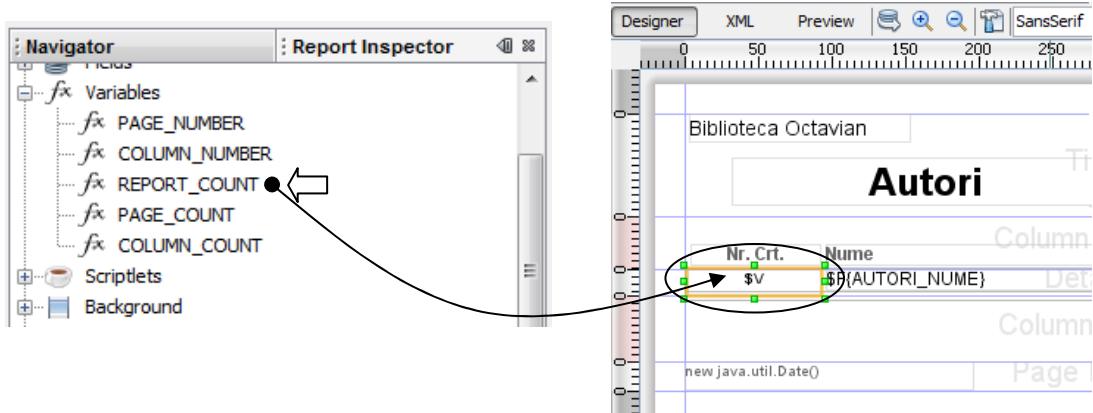


Pentru a referi o variabilă se folosește sintaxa:

\$V{nume_var}

Exemplu de utilizare: Adăugarea în banda Detail a unui control de tip *TextField* care conține valorile variabilei *REPORT_COUNT* care numără liniile din raportul realizat

se face prin deplasarea cu mouse-ul (*drag & drop*) a variabilei din fereastra *Report Inspector* în zona dorită.



Evident va trebui să adăugăm în banda *Column Header* o etichetă corespunzătoare (*Nr. crt.*).

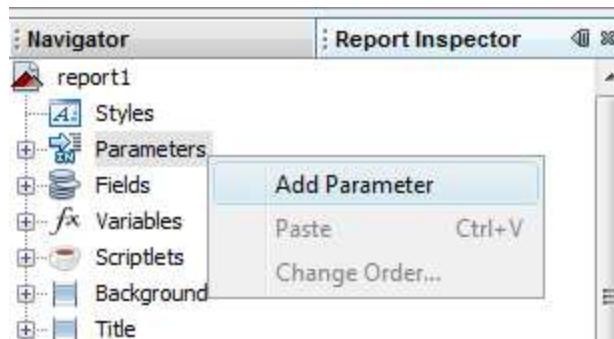
Parametri

Parametrii sunt obiecte având valori fixe, definite în momentul declarării lor. Ei vor apărea în comanda *SELECT* care realizează selectarea datelor care trebuie incluse în raport. În momentul declanșării realizării raportului, în aplicația în care acesta este integrat, valorile parametrilor declarate în etapa de construire a raportului vor fi înlocuite cu valorile lor efective (înaintea populării cu date a raportului prin interogarea bazei de date).

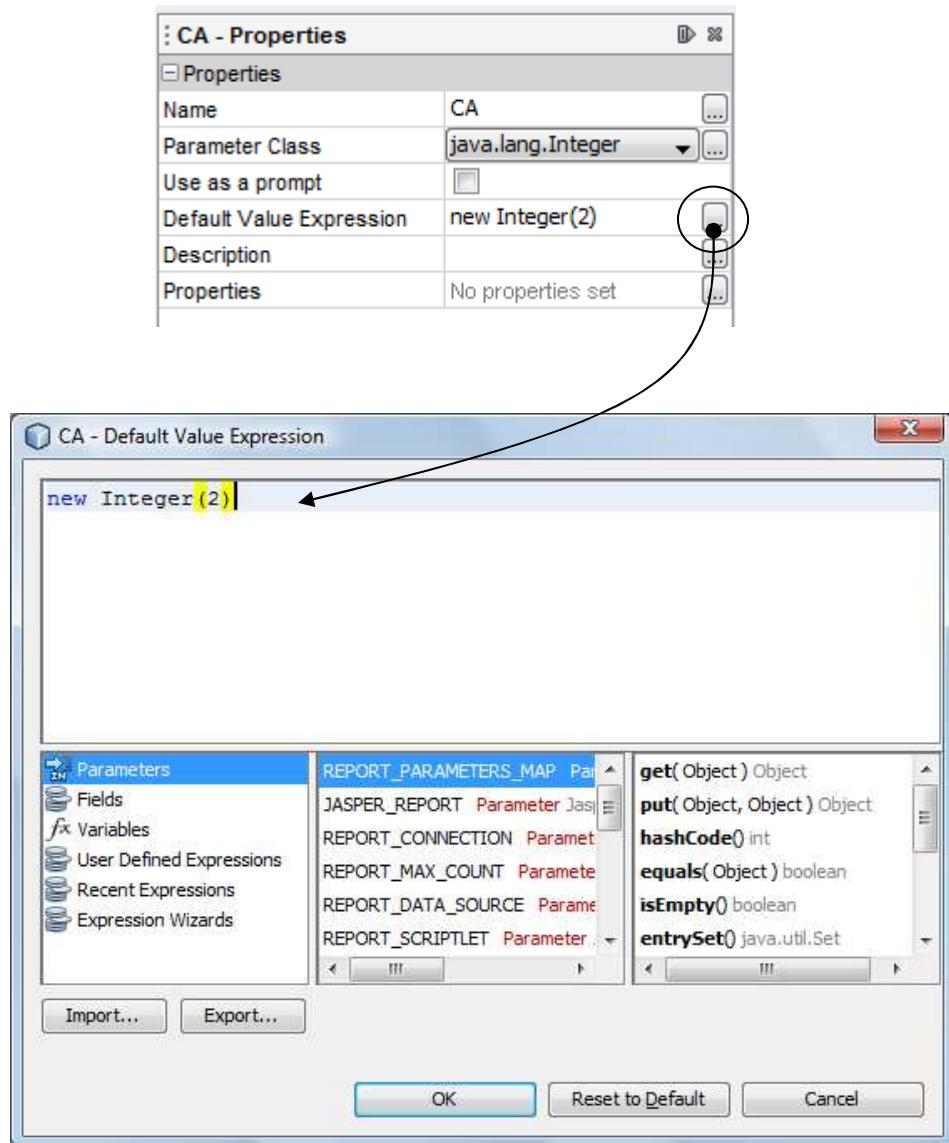
Pentru a referi un parametru se folosește sintaxa *\$P{nume_parametru}*.

Exemplu: Dacă trebuie afișați numai autorii pentru care *COD_AUT > val* se va adăuga raportului parametrul CA după care se va modifica comanda *SELECT* care realizează popularea cu date a raportului. Parametrul CA va fi inițializat cu o valoare de test, de exemplu CA=2. În momentul integrării raportului în aplicație se va transmite însă parametrului valoarea efectivă, preluată dintr-un formular lansat înaintea realizării raportului.

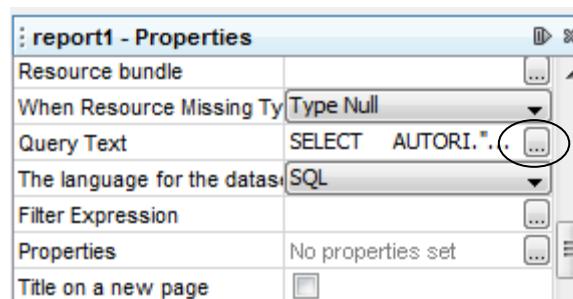
Adăugarea parametrului CA se realizează selectând în *Report Inspector* intrarea *Parameters / Add Parameter*:



Noul parametru primește numele implicit *Parameter1* care poate fi editat folosind proprietatea *Name* din fereastra de proprietăți. Tot în fereastra de proprietăți se va impune clasa noului parametru (*java.lang.Integer* în exemplul considerat) și valoarea inițială (*new Integer(2)*).

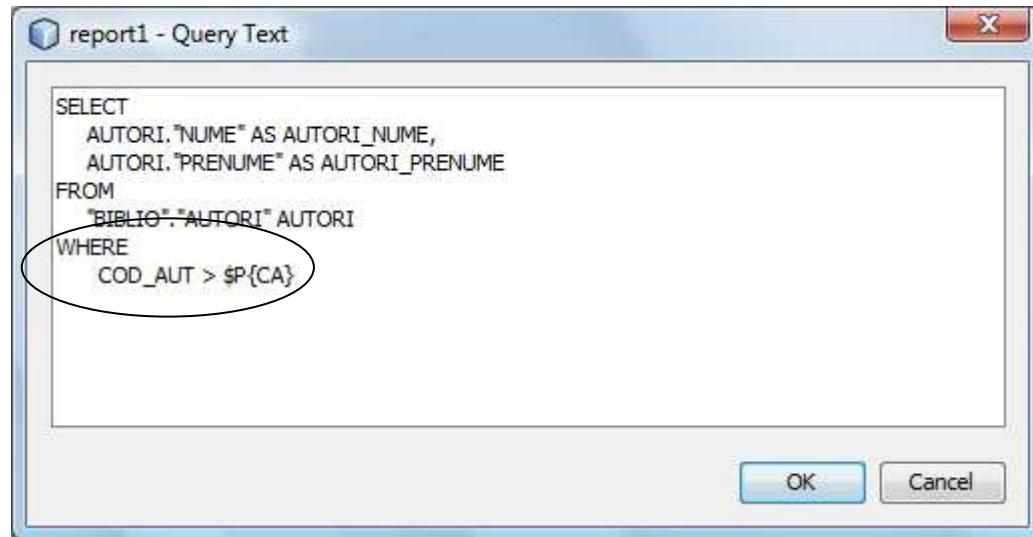


În continuare se va edita comanda *SELECT*. Pentru aceasta se selectează raportul (se selectează cu mouse-ul un punct din raport în care nu este nici o informație) și apoi, în fereastra *Properties* se selectează butonul din dreptul proprietății *Query Text*.

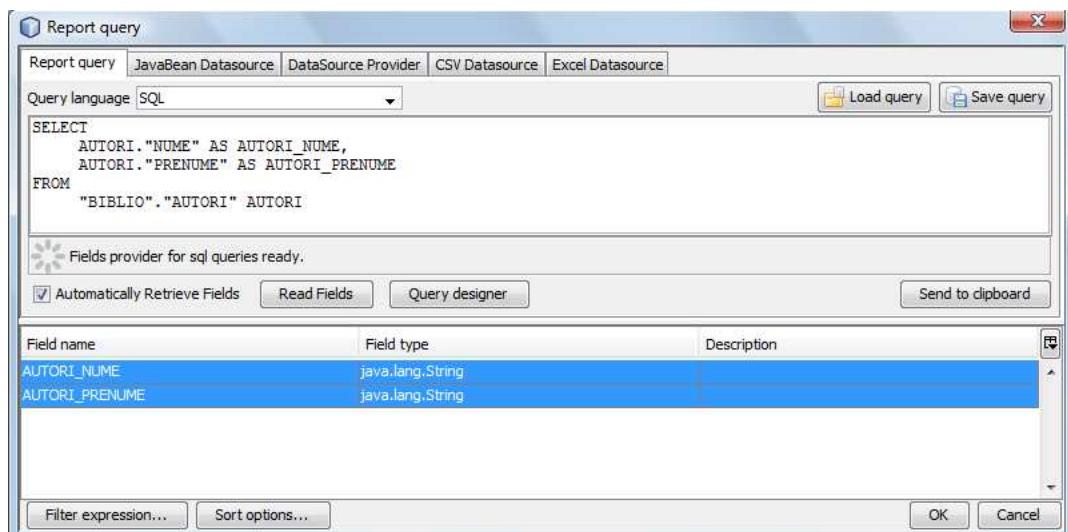
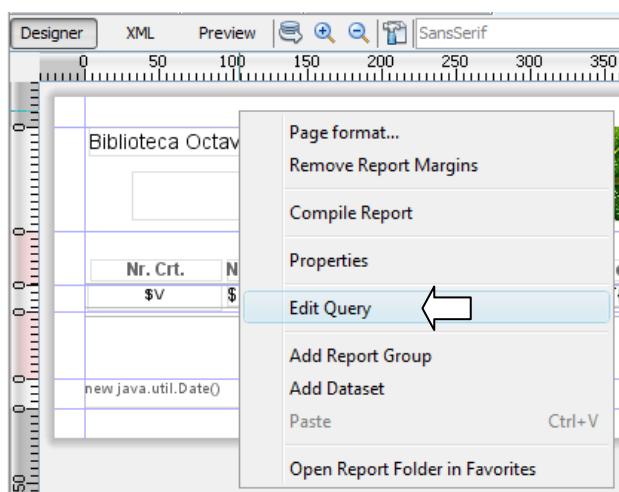


Frazei inițiale i s-a adăugat clauza *where* corespunzătoare:

... WHERE COD_AUT > \$P{CA}



Notă: Comanda SQL poate fi editată și mai ușor din fereastra afișată prin selectarea din meniu contextual a opțiunii *Edit Query*.



Expresii

Conținutul unei casete poate fi rezultatul evaluării unei expresii. Rezultatul trebuie să fie un obiect Java. Clasa acestui obiect și clasa casetei (implicită sau impusă accesându-i proprietățile) trebuie să coincidă.

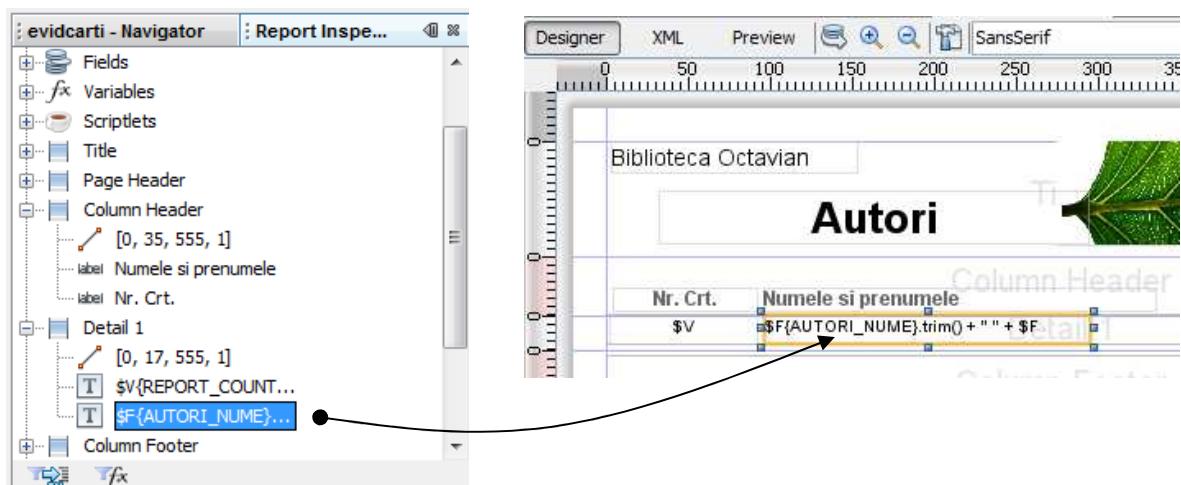
Expresiile pot conține referiri la câmpuri, parametri sau variabile.

Exemple:

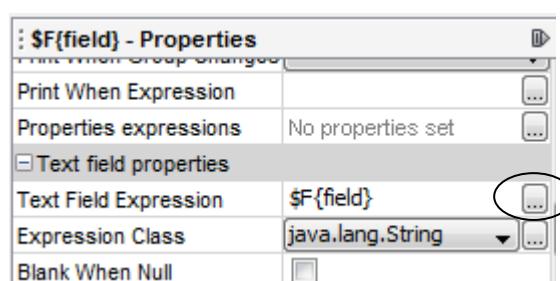
- "Debitori"
- \$F{JUDET}
- "-" + "Pagina " + \$V{PAGE_NUMBER} + " - "
- new Boolean(true)
- new Integer(3)
- ((\$P{parametru}.equals("RON")) ? "RON" : "EUR")
- new Double(\$F{f1}.doubleValue() + \$F{f2}.doubleValue())
- \$F{nume}.trim() + " " + \$F{initiala} + ". " + \$F{prenume}.trim() +
(((\$F{nume_nou}.trim().length() > 0) ? " (cas. " + \$F{nume_nou} + ")" : "")

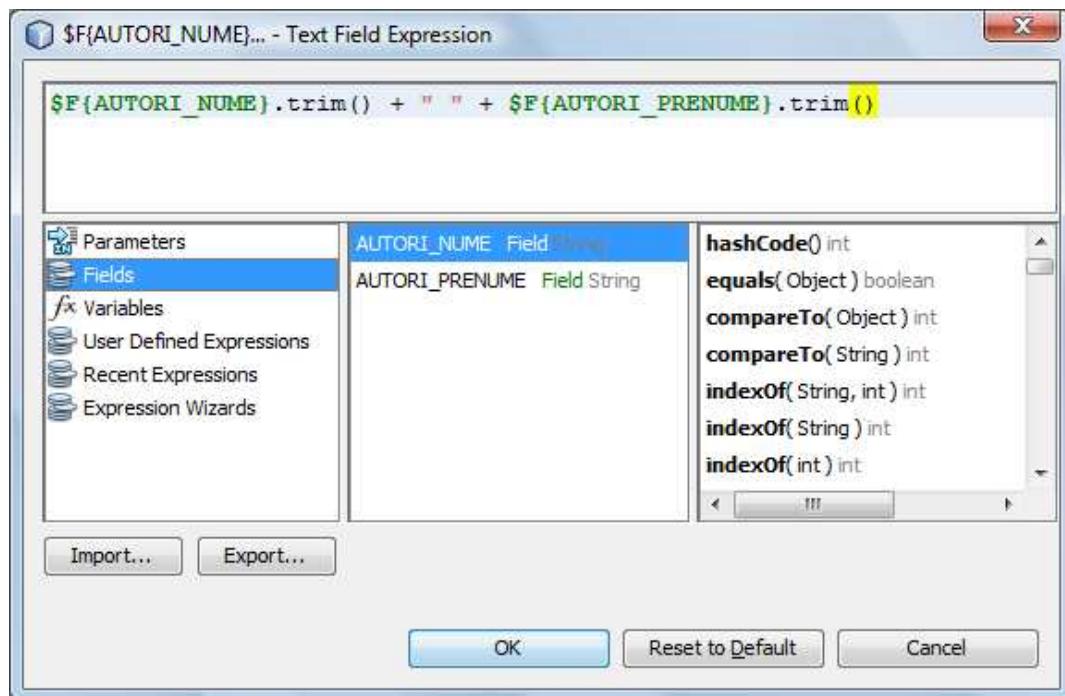
Exemplu: Adăugarea unui câmp care să conțină numele și prenumele autorilor.

1. se adaugă în banda *Detail* un control de tip *Text Field*:



2. se editează expresia:

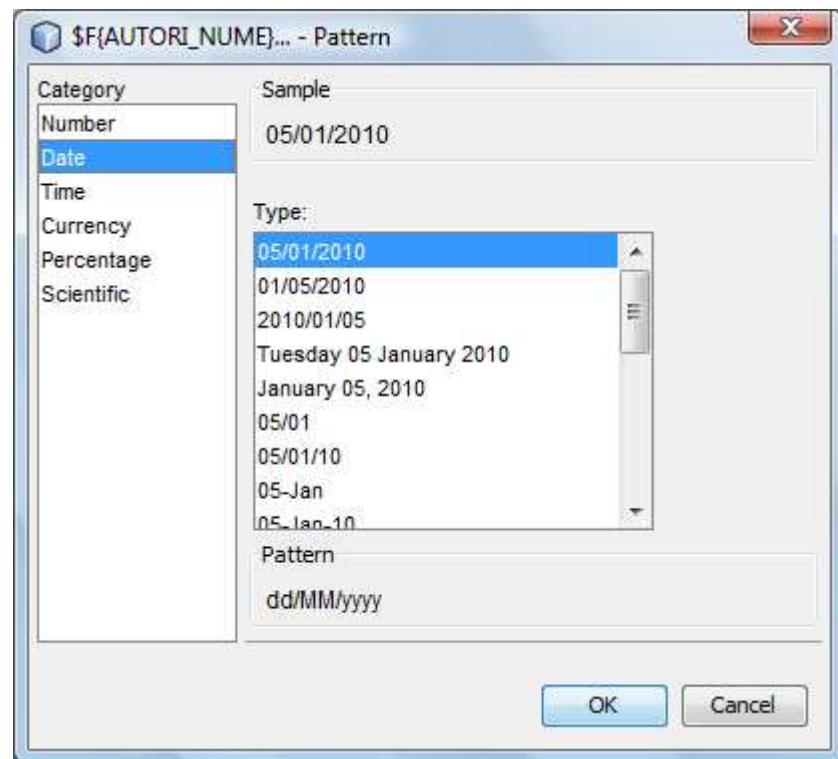




3. se deplasează obiectul în poziția dorită și se impune fontul. Rezultat posibil:

The screenshot shows the 'Preview' tab of a report. The report header reads 'Biblioteca Octavian'. Below it, the word 'Autori' is displayed in a large, bold, italicized font. To the right of 'Autori' is a decorative image of a green leaf. The report body contains a table with four rows, each with a number and a name: 1. Puskin Alexandr, 2. Steinbeck John, 3. Steinhardt Nicolae, 4. Marcel Proust.

Pentru impunerea formatului dorit pentru rezultatul unei expresii se accesează proprietatea *Pattern*.



Exemplu:

Un raport afișează după generare valorile din figură:

taxe_data	taxe_suma_lei
09/11/07 00:00	-2.5362318840579707
09/11/07 00:00	72.46376811594203
09/11/07 00:00	-2.5362318840579707

După modificarea modului de afișare folosind *Pattern* se poate ajunge la următorul mod de afișare:

taxe_data	taxe_suma_lei
09/11/2007	-2.54
09/11/2007	72.46
09/11/2007	-2.54

Integrarea raportului în aplicație

Secvențele de cod prezentate în continuare presupun că fișierul creat de extensia *iReport*, (în exemplul dat *report1.jrxml*) este conținut într-un director denumit *Rapoarte* derivat din directorul rădăcină al proiectului.

Pentru a integra raportul într-o aplicație scrisă în Java este necesar să se parcurgă pașii următori:

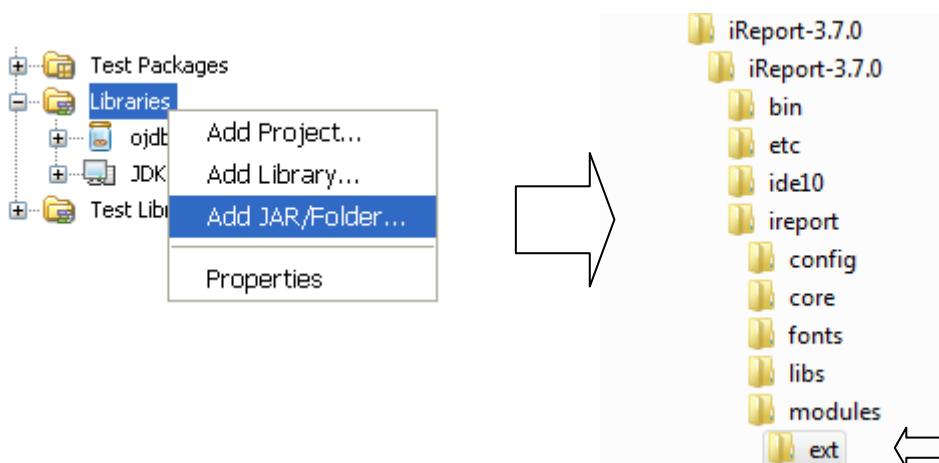
1. Se adaugă într-unul dintre meniurile derulate al aplicației o nouă intrare.

Selectarea noii opțiuni va declanșa realizarea raportului și afișarea sa într-o fereastră similară cu cea folosită la vizualizare în iReport.

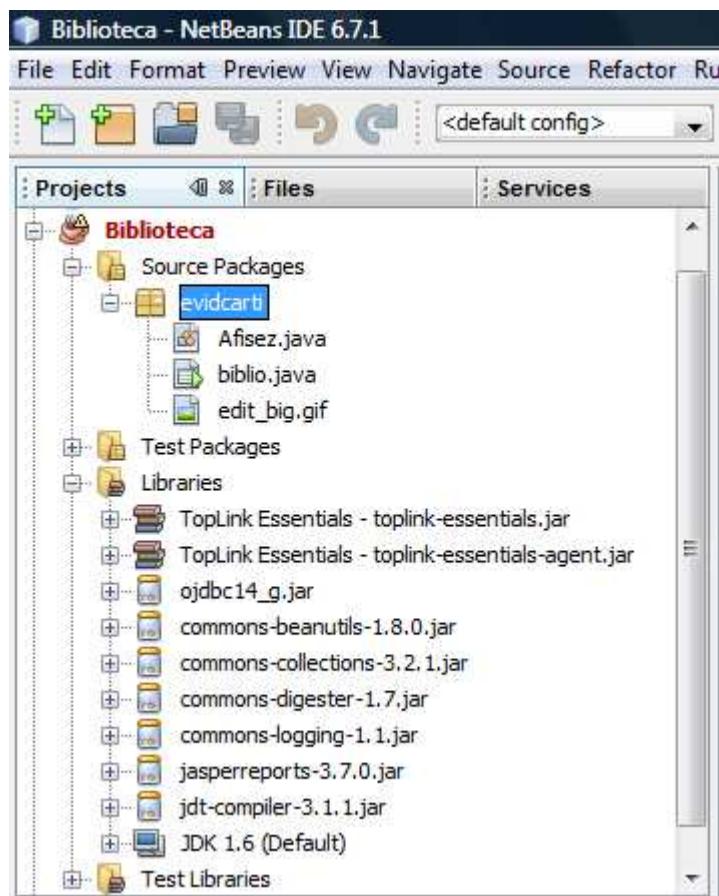
2. Se adaugă aplicației un ansamblu de arhive *.jar* : *commons-beanutils-1.8.0.jar*, *commons-collections-3.2.1.jar*, *commons-digester-1.7.jar*, *commons-logging-1.1.jar*, *jasperreports-3.7.0.jar*, *jdt-compiler-3.1.1.jar*.

Aceste arhive se găsesc în directorul descărcat împreună cu cel care conține extensia iReport pentru Netbeans. Calea spre arhivele *.jar* menționate este *iReport-3.7.0 / ireport / modules/ext*.

Observație: JasperReports și iReport fiind produse cu o dezvoltare rapidă și continuă, *versiunile fișierelor menționate* pot diferi. Atenție însă la compatibilitate! Dacă instalați o nouă versiune a extensiei *iReport* pentru Netbeans, descărcați și versiunea de *iReport* aferentă (același număr de versiune!).



Rezultat:



3. Se editează funcția executată la selectarea opțiunii din meniu adăugată pentru realizarea raportului

Funcția adăugată va conține declararea variabilei *raport* care va indica fișierul *.jrxml* generat de *iReport* și trei linii de cod realizând o compilare a fișierului *.jrxml* (metoda *compileReport()*), interogarea bazei de date pentru aducerea datelor în raport (metoda *fillReport()*) și vizualizarea raportului într-o fereastră similară celei folosite de *iReport* (metoda *viewReport()*).

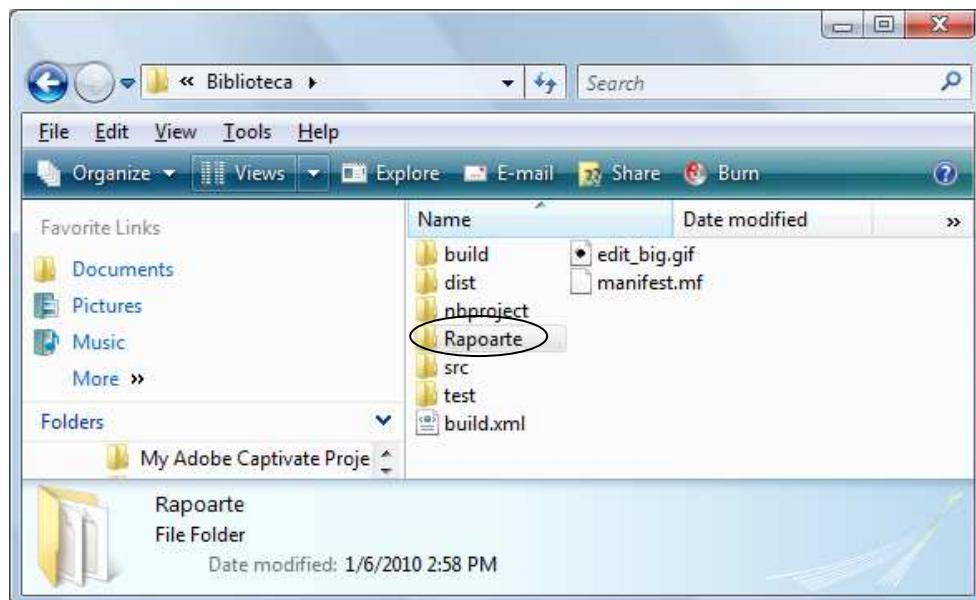
```
private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String rapport = "Rapoarte/report1.jrxml"; // Fisier realizat de iReports
    HashMap params = new HashMap(); // Va servi la transmiterea parametrilor
    try {
        JasperReport jasperReport = JasperCompileManager.compileReport(rapport);
        JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, params, cnx);
        JasperViewer.viewReport(jasperPrint, false);
    }
}
```

```

        catch (JRException ex) {
            ex.printStackTrace();
        }
    }
}

```

Fișierul *report1.jrxml* trebuie să se afle în directorul *Rapoarte* din directorul rădăcină al proiectului.



Transmiterea parametrilor

Presupunând că dorim să realizăm un raport care să afișeze cărțile unei edituri apărute după un an precizat. La construirea raportului s-au definit doi parametri, *editura* și *an_ref*. În secvența de cod prezentată mai sus se vor modifica , înaintea generării raportului se impun valorile efective ale celor doi parametri. Acestea provin din două controale ale formularului realizat în acest scop. Secvența de cod prezentată este executată la apăsarea butonului *ButonRaport* de pe același formular.

```

private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String raport = "Rapoarte/rapcarti.jrxml"; // Fisier realizat de iReports
    HashMap params = new HashMap();
    // se impun valorile celor 2 parametri din raport, editura si an_ref
    params.put("editura", (String)edt.getSelectedItem());
    int anul = Integer.parseInt(anreferinta.getString());
    params.put("an_ref", anul); // anul de referinta
}

```

```

// Se poate face raportul
try {
    JasperReport jasperReport = JasperCompileManager.compileReport(raport);
    JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, params, cnx);
    JasperViewer.viewReport(jasperPrint, false);
}
catch (JRException ex) {
    ex.printStackTrace();
}
}
}

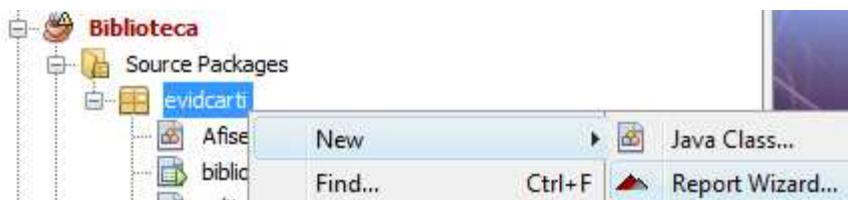
```

Pentru afişarea şi imprimarea altor rapoarte ale aplicaţiei funcţiile necesare vor fi similare celei scrise, singura diferenţă fiind numele fişierului *.jrxml* şi secvenţa de preluare a valorilor eventualilor parametri.

Realizarea rapoartelor conţinând grupuri de articole

Rapoartele pot fi cu mult mai complexe decât cel realizat. Un prim exemplu în acest sens este raportul prezentat în continuare în care se doreşte afişarea cărţilor unei biblioteci grupate pe edituri.

Pentru crearea unei prime forme a raportului se va face apel la *Report wizard*:



Numele raportului va fi *carti*, descrierea sa fiind conţinută în fişierul *carti.jrxml*.

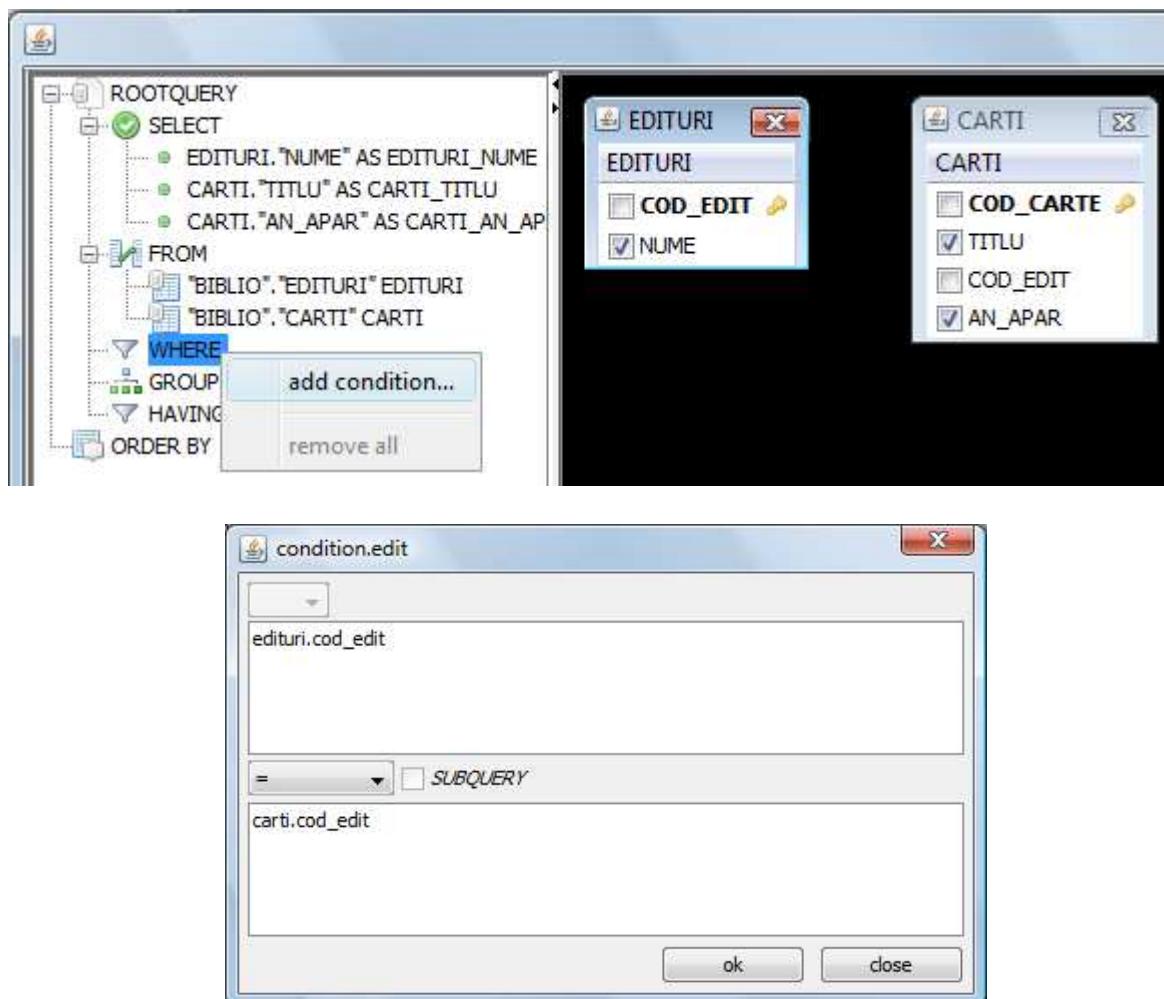
Comanda SQL *SELECT* pe care se va baza raportul este următoarea:

```

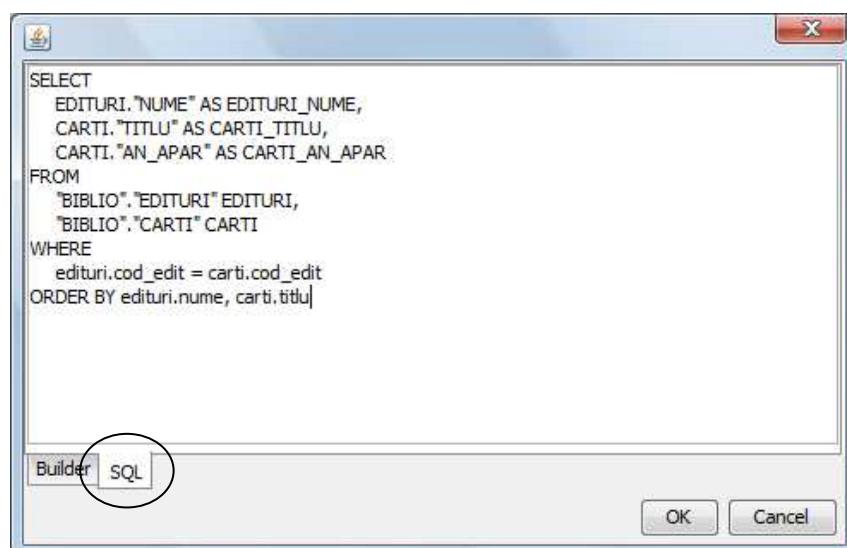
SELECT
    CARTI."TITLU" AS CARTI_TITLU,
    CARTI."AN_APAR" AS CARTI_AN_APAR,
    EDITURI."NUME" AS EDITURI_NUME,
    EDITURI."TELEFON" AS EDITURI_TELEFON
FROM
    "BIBLIO"."EDITURI" EDITURI INNER JOIN "BIBLIO"."CARTI" CARTI ON
        EDITURI."COD_EDIT" = CARTI."COD_EDIT"
ORDER BY EDITURI.NUME ASC, CARTI.TITLU ASC

```

Adăugarea clauzei *where* din comanda SQL se realizează folosind meniu contextual afișat la selectarea clauzei în fereastra editorului comenzi.

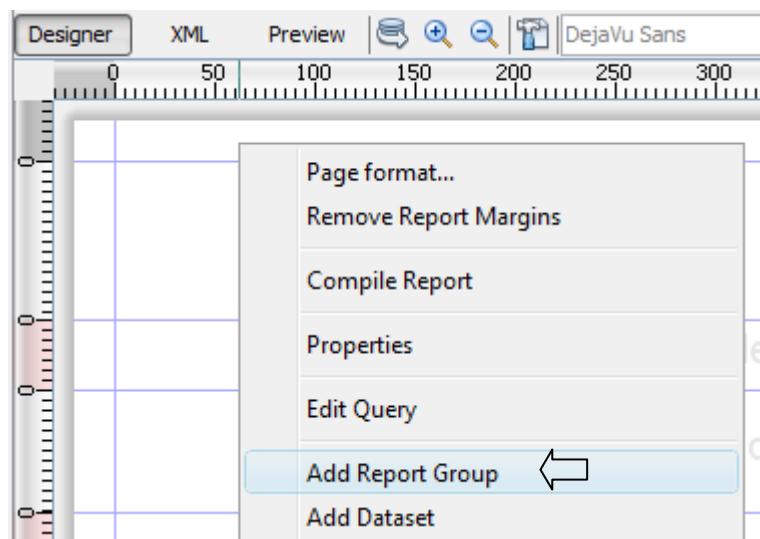


Dacă adăugarea condiției de ordonare folosind instrumentele din fereastra grafică nu este posibilă se va putea adăuga clauza direct în comanda SQL realizată.

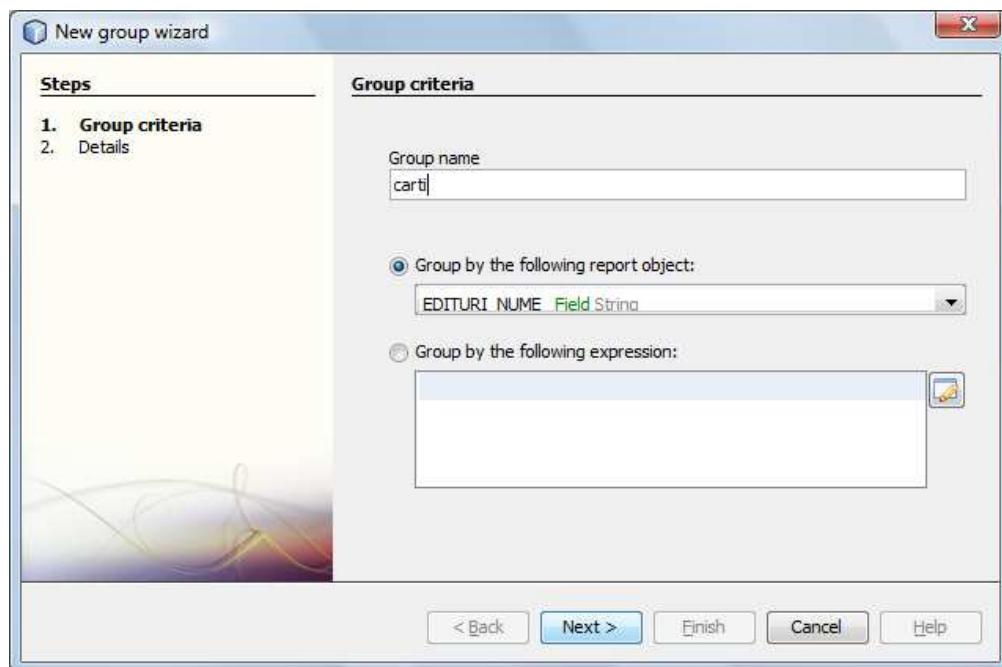


După crearea comenții SELECT și selectarea câmpurilor se va selecta butonul Next până la terminarea execuției aplicației expert și afișarea raportului.

După afișarea raportului se va adăuga acestuia o bandă suplimentară necesară afișării grupate a conținutului. Pentru aceasta se va selecta cu butonul drept al mouse-ului un punct din fereastra grafică a mediului de programare și în meniu contextual se va selecta *Add Report Group*.



În fereastra care se va afișa se va impune un nume grupului (carti) și se va selecta câmpul care va fi folosit la grupare (*edituri.nume*).



După apăsarea butonului Next, în fereastra care se afișează se va selecta caseta de selecție *Add the group header*.



Aplicația va adăuga raportului o bandă suplimentară, *Carti Group Header 1*.

În continuare va fi adăug conținutul fiecărei regiuni a raportului . În banda *Carti Group Header 1* va fi adăugat câmpul *Edituri.nume* iar în banda *Detail1* vor fi adăugate câmpurile *Carti.titlu* și *Carti.an_apar*. Adăugarea câmpurilor se realizează prin târâirea lor cu mouse-ul di fereastra *Report Inspector*.

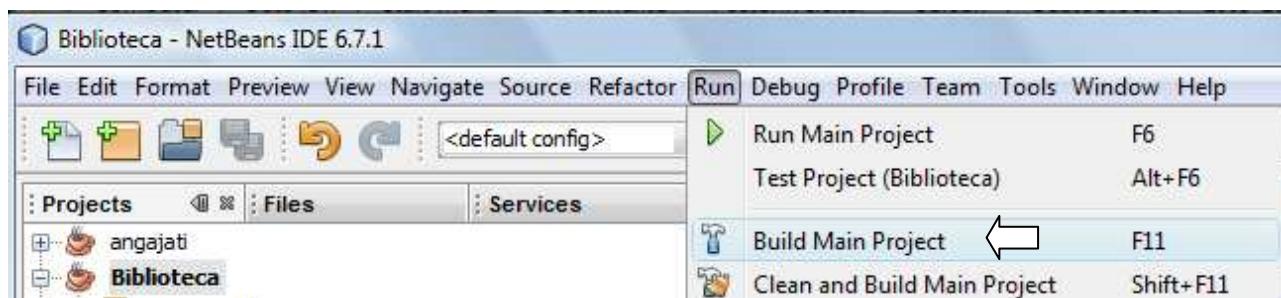
În cazul pornirii de la macheta Blank A4 s-ar putea ajunge de exemplu la următorul rezultat:

Notă: Benzile nefolosite (*Column Header* și *Page Header*) au fost ascunse.

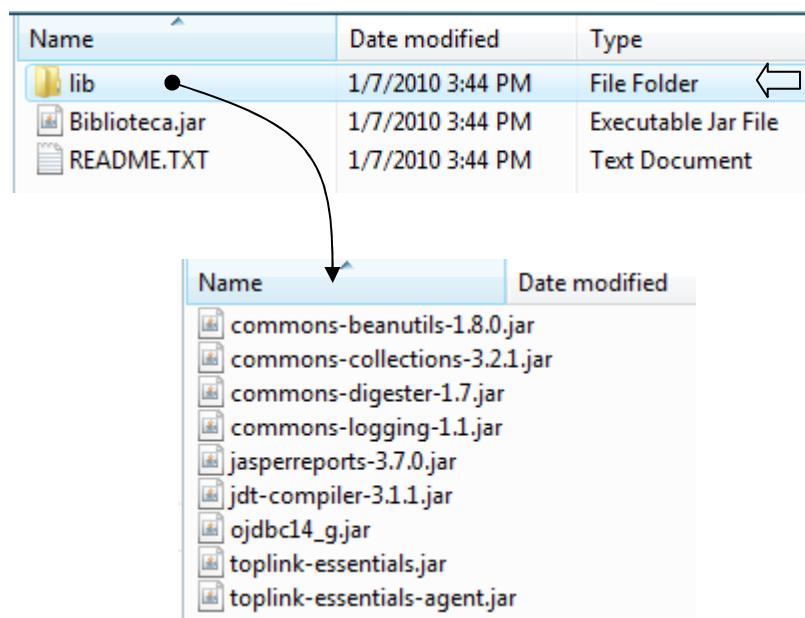
Carti grupate pe edituri		
Cartea Romaneasca		
Doctorul misterios		1975
Istoria celor 13		1981
Te iubesc, viata		1984
Dacia		
Jurnalul fericirii		1991
Orizonturi		

Pregătirea aplicației pentru distribuție

În urma testării, aplicația trebuie pregătită astfel încât să poată fi lansată în execuție și fără utilizarea mediului de programare *Netbeans*. Pentru a realiza aceasta se va selecta *Run / Build Main Project*.



Mediul de programare va adăuga directorului aplicației un subdirector denumit *dist* care va coține o arhivă *.jar* executabilă. Pe lângă aceasta directorul *dist* va mai conține subdirectorul *lib*. În acest subdirector mediul *Netbeans* va copia toate bibliotecile adăugate proiectului în timpul realizării sale (în *Libraries*).



Dacă aplicația mai conține fișiere și în alte directoare, și este de regulă cazul fișierelor *.jxml* care conțin descrierea rapoartelor, în *dist* vor trebui copiate și aceste directoare. În exemplul ales a trebuit copiat subdirectorul *Rapoarte*.

Lansarea în execuție fără *Netbeans* se realizează selectând arhiva executabilă cu butonul drept al mouse-ului și apoi *Open With / Java Platform SE binary*.

