DIMA NARCISA

APLICAȚII PL/SQL ÎN ORACLE APEX

EDITURA EVOMIND

APLICAŢII PL/SQL ÎN ORACLE APEX

Autor: DIMA NARCISA

ISBN 978-606-94690-9-5

Editura Evomind, 2020

https://evomind.org/

CUPRINS

1. CREAREA UNEI APLICAȚII ÎN ORACLE APEX 18.2

În cele ce urmează, vom crea o aplicație cu baze de date, folosind versiunea 18.2 a utilitarului ORACLE APEX, lansată în septembrie 2018. Vom considera, spre exemplificare, crearea unei aplicații în care să ținem evidența medicamentelor dintr-o farmacie. Vom crea patru tabele în care vom reține denumirea, categoria, producătorii, precum și evidența stocului pentru fiecare medicament în parte, după cum urmează:

Tabela ***Medicamente***

```
create table  "medicamente"
  (                  "cod" number not null enable,
                     "medicament" varchar2(40) not null enable,
                     "grupa_terapeutica" number,
                      constraint "medicamente_pk" primary key ("cod")
  using index  enable
   )
alter table  "medicamente" add constraint "medicamente_fk" foreign key ("grupa_terapeutica")
                      references  "grupe_terapeutice" ("cod") on delete cascade enable
create or replace editionable trigger  "bi_medicamente"
  before insert on "medicamente"
  for each row
begin
  if :new."cod" is null then
    select "medicamente_seq".nextval into :new."cod" from sys.dual;
  end if;
end;
alter trigger  "bi_medicamente" enable
```

Tabela ***Grupe_Terapeutice*** în care vom reține grupele terapeutice din care fac parte medicamentele:

```
create table  "grupe_terapeutice"

  (        "cod" number not null enable,
           "grupa_terapeutica" varchar2(40) not null enable,
           constraint  "grupe_terapeutice_pk"  primary  key
("cod")
  using index  enable,
           constraint       "grupe_terapeutice_uk1"       unique
("grupa_terapeutica")
  using index  enable
  )
create or replace editionable trigger  "bi_grupe_terapeutice"
  before insert on "grupe_terapeutice"
  for each row
begin
  if :new."cod" is null then
    select  "grupe_terapeutice_seq".nextval  into  :new."cod"
from sys.dual;
  end if;
end;
alter trigger  "bi_grupe_terapeutice" enable
```

Tabela ***Firme*** în care reținem denumirea firmelor producătoare de medicamente

```
create table  "firme"
  (       "cod" number not null enable,
          "firma" varchar2(20) not null enable,
           constraint "firme_pk" primary key ("cod")
 using index  enable,
          constraint "firme_uk1" unique ("firma")
 using index  enable
  )
create or replace editionable trigger  "bi_firme"
 before insert on "firme"
 for each row
begin
 if :new."cod" is null then
   select "firme_seq".nextval into :new."cod" from sys.dual;
 end if;
end;

alter trigger  "bi_firme" enable
```

7 Aplicaţii PL/SQL în ORACLE APEX

Crearea aplicației presupune parcurgerea următorilor pași:

Alegerea interfeței grafice, denumirea aplicației

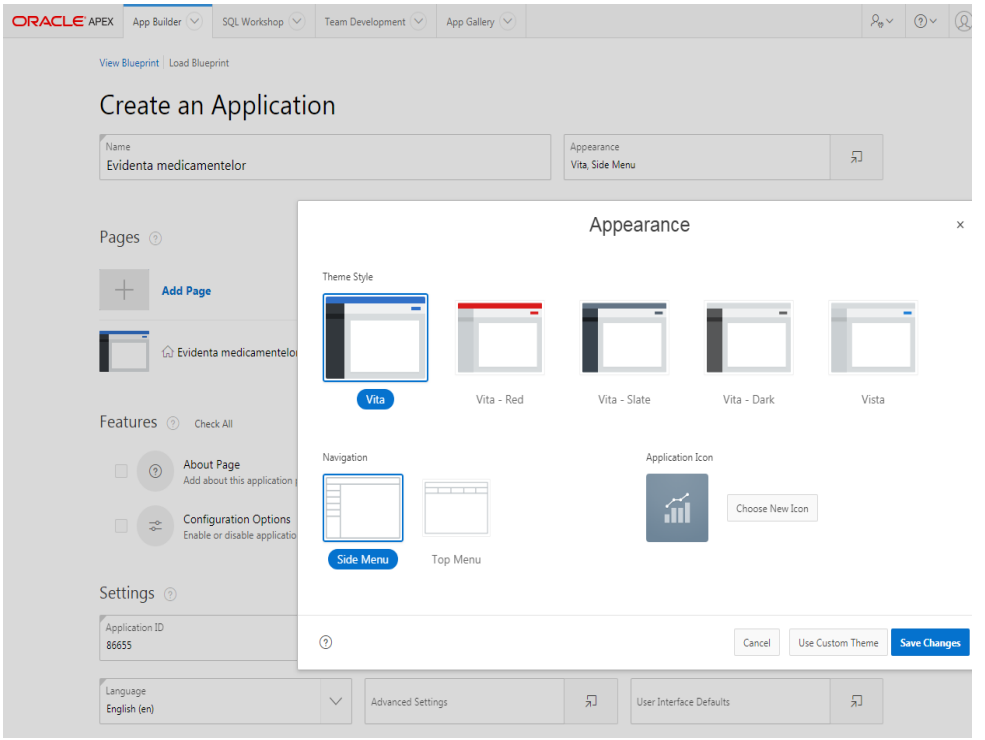


Figura 1.1 Alegerea interfeței grafice, denumirea aplicației

Lansarea în execuție a aplicației:

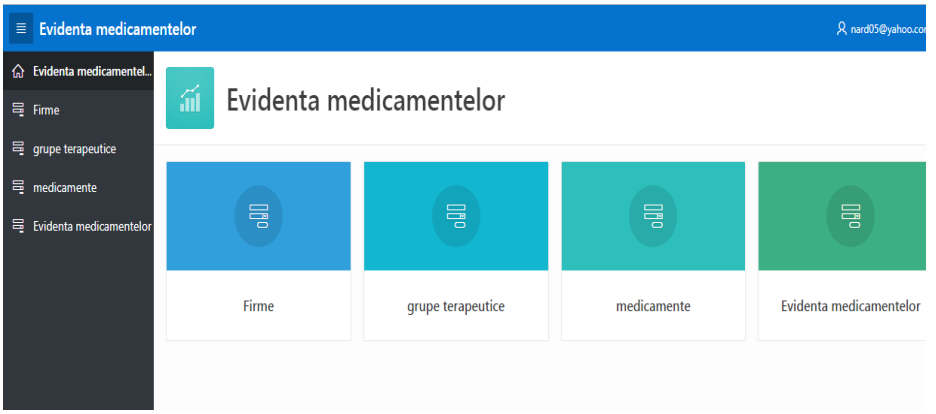


Figura 1.2 Lansarea în execuție a aplicației

Crearea listelor de valori și atașarea acestora la itemi

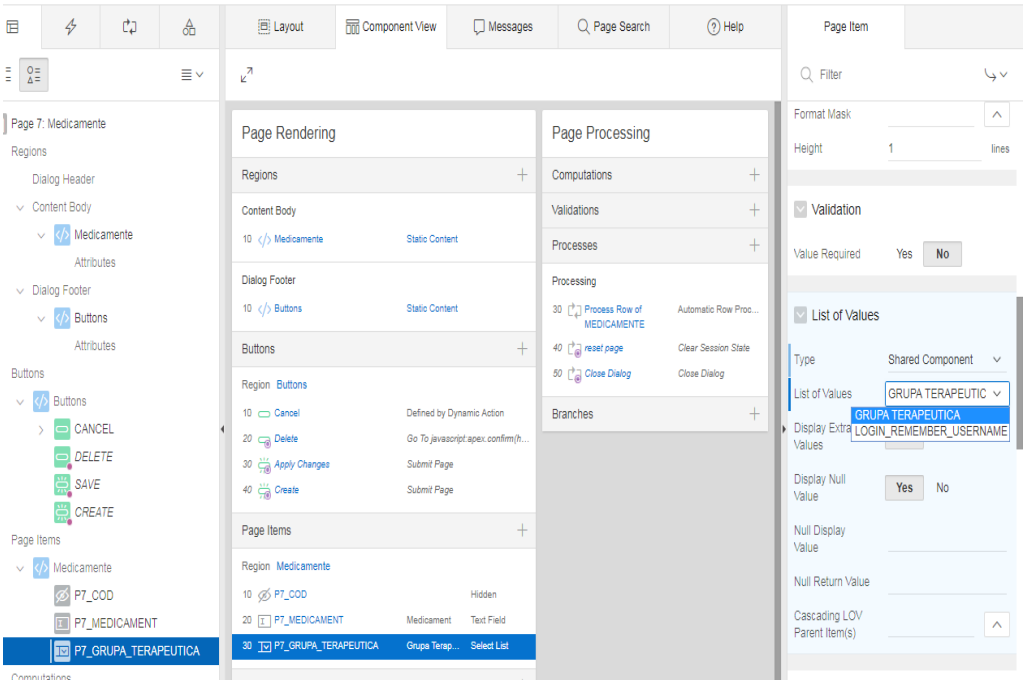


Figura 1.3 Crearea listelor de valori și atașarea acestora la itemi

Crearea unui grafic în aplicație

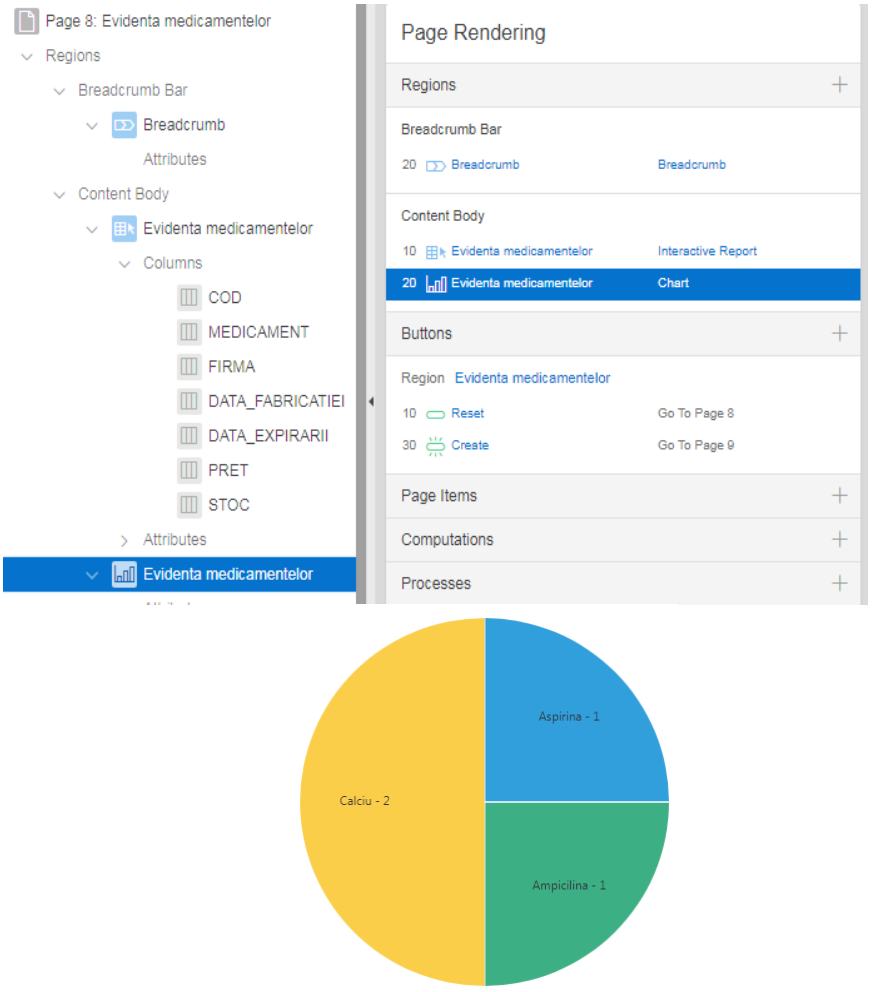


Fig. 1.4 Crearea unui grafic în aplicație

Raport cu grafic atașat

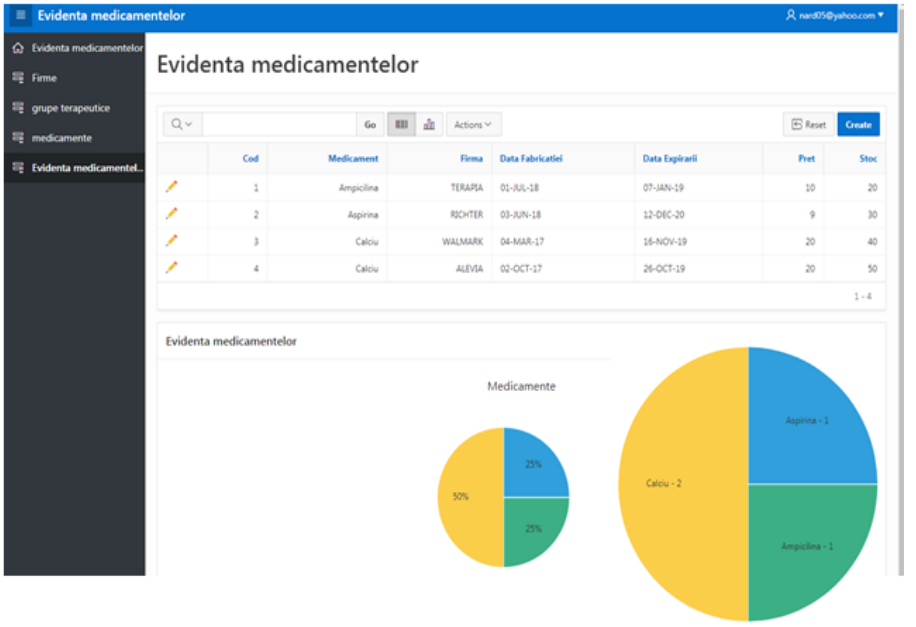


Figura 1.5 Raport cu grafic atașat

2. ACTUALIZAREA TABELELOR PRIN ACTIUNI DINAMICE IN ORACLE APEX 19.1

În cele ce urmează, ne propunem să actualizăm o tabelă prin intermediul unei acțiuni dinamice ce se va executa la crearea unei înregistrări în altă tabelă. Spre exemplificare, vom crea o aplicație ce va ține evidența vânzărilor dintr-o librărie. Tabela *Evidemțe* reține evidența stocului existent pentru fiecare carte prezentă în librărie, iar tabela *Vânzări* reține vânzările efectuate. Ne propunem ca la introducerea unei vânzări, să se actualizeze automat stocul cărții respective din tabela *Evidențe*. Pentru aceasta, vom crea o acțiune dinamică ce se va executa la introducerea unei vânzări.

```
Cazul TRUE:
declare
s number;
BEGIN
select stoc into s from evidente
where cod=:P13_EVIDENTA;
IF (s>=:P13_EXEMPLARE and
:P13_EXEMPLARE>0)
THEN
BEGIN
update evidente
set stoc= stoc- :P13_EXEMPLARE
where cod= :P13_EVIDENTA;
```

```
:P13_REZULTAT:='VANZARE
EFECTUATA';
        END;
        END IF;
        :P13_AUTORUL:=NULL;
        :P13_EVIDENTA:=NULL;
        END;
Cazul False:
begin
:P13_REZULTAT:='NU SE
POATE EFECTUA VANZAREA';
end;
```

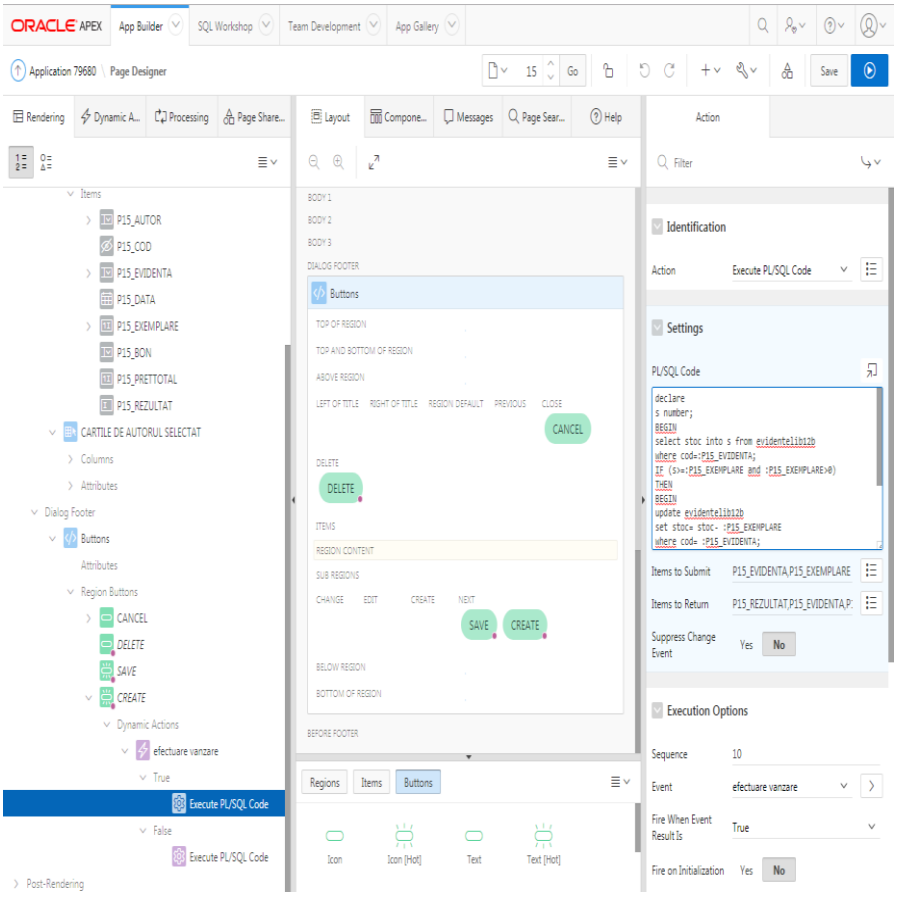


Figura 2.1 Acţiune dinamică ataşată butonului CREATE din formularul VÂNZĂRI

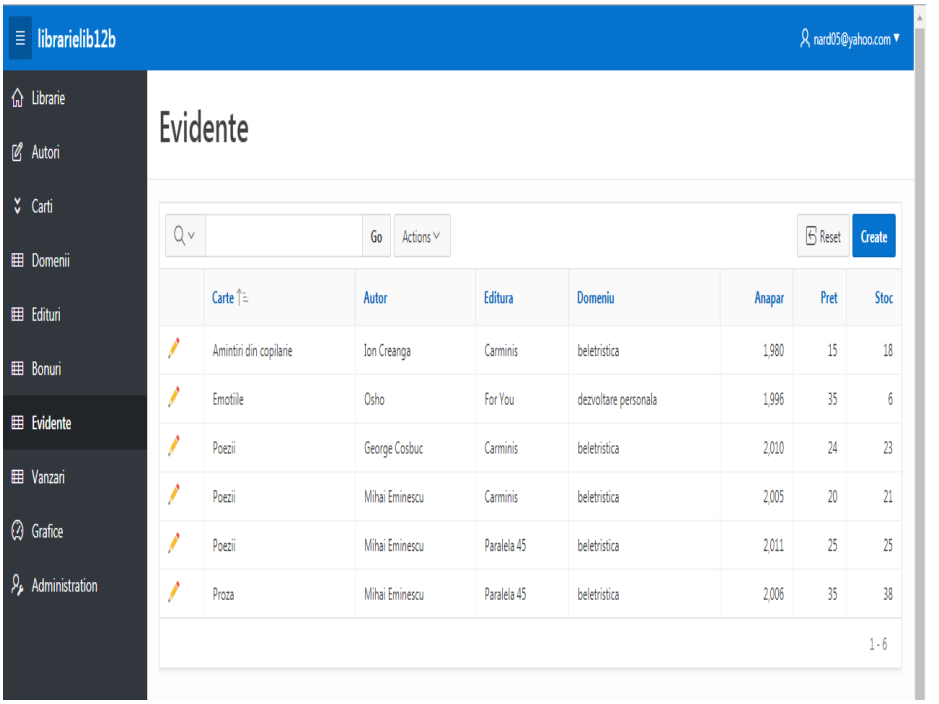


Figura 2.2 Tabela Evidențe

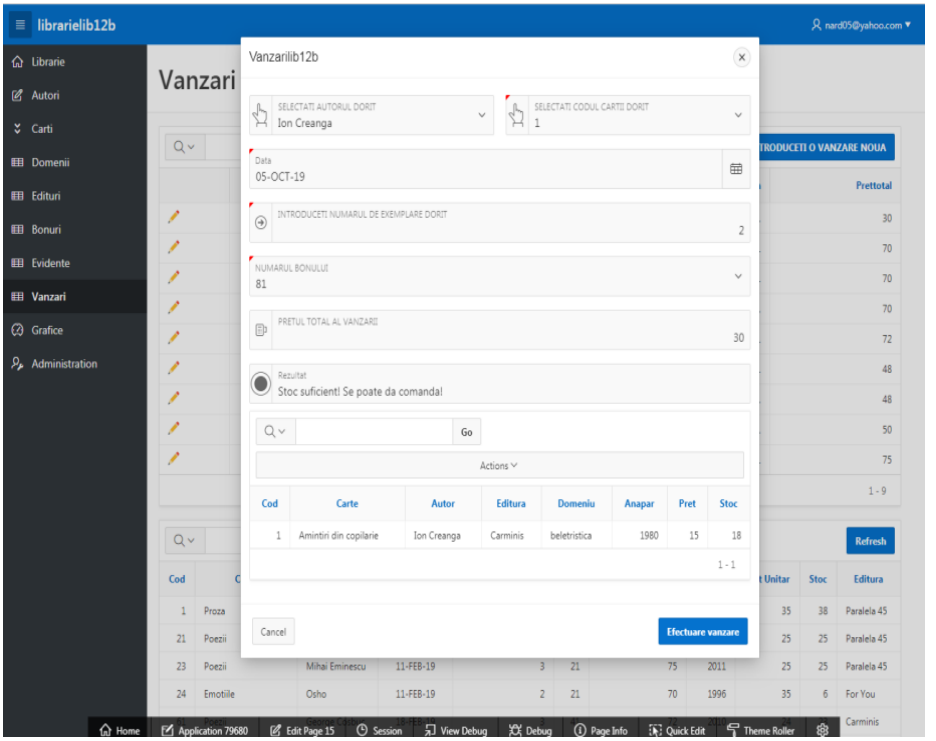


Figura 2.3 Introducerea unei vânzări prin intermediul formularului Vânzări

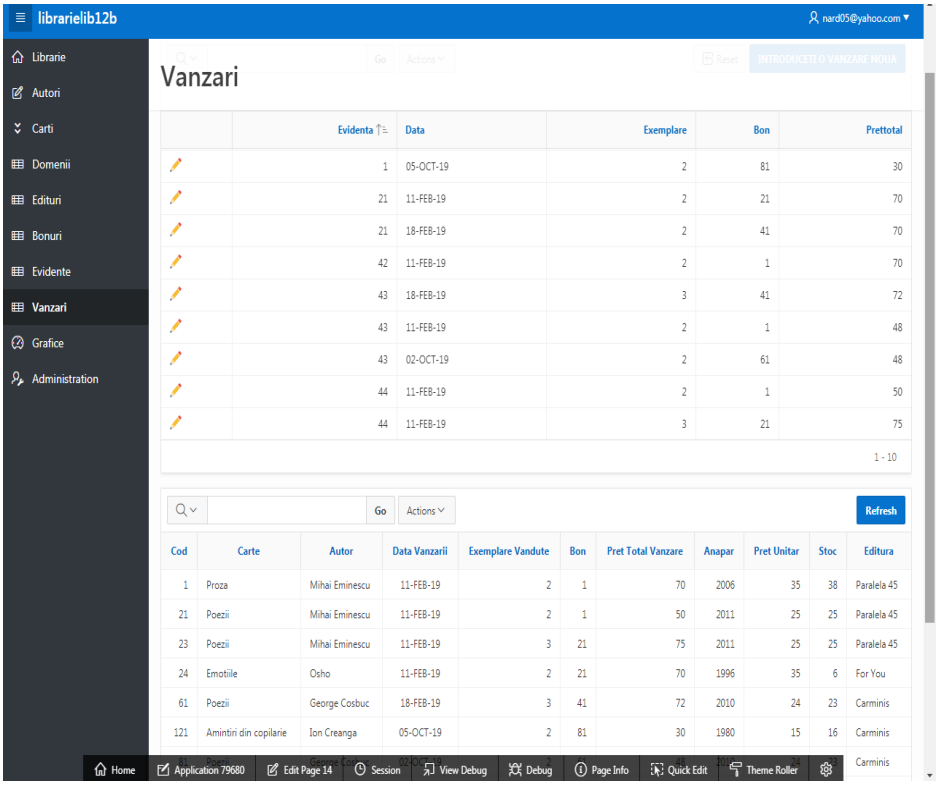


Figura 2.4 Actualizarea automată a stocului pentru cartea vândută

3. TRATAREA EXCEPTIILOR IN ORACLE PL/SQL

Apariția unei erori în timpul execuției unui bloc de program în ORACLE PL/SQL, are ca efect întreruperea acestuia. Restabilirea execuției programului presupune tratarea erorii respective prin folosirea unei excepții. O *excepție* reprezintă un cod special ce se adaugă la finalul blocului de program unde se anticipează apariția erorii. În scrierea codului respectiv se va ține cont de tipul erorii ce ar putea să apară în acel bloc.

În funcție de erorile ce pot apărea în cadrul execuției unui bloc de program, există două tipuri de excepții: generate de server-ul ORACLE și cele definite de utilizator. Excepțiile generate de server-ul ORACLE pot fi la rândul lor de două tipuri: predefinite sau non-predefinite. Cele predefinite au un nume, un cod de eroare, un mesaj și se generează automat de server-ul ORACLE, neavând nevoie de o declarare explicită. Ca exemplu, în cazul unei comenzi SQL ce nu returnează nicio valoare, se generează excepția NO_DATA_FOUND asociată codului de eroare ORA-01403.

Spre deosebire de excepțiile predefinite, cele non-predefinite generate de server-ul ORACLE, nu au un nume. De aceea, acestea trebuie declarate explicit de către utilizator în secțiunea declarativă

a blocului de program, asociind numele excepției la codul erorii respective prin folosirea funcției PRAGMA EXCEPTION_INIT. Ca exemplu, în *Figura 3.1* s-a generat o eroare non-predefinita de către server-ul ORACLE când s-a încercat introducerea valorii NULL în câmpul clasa ce nu admite astfel de valoare.

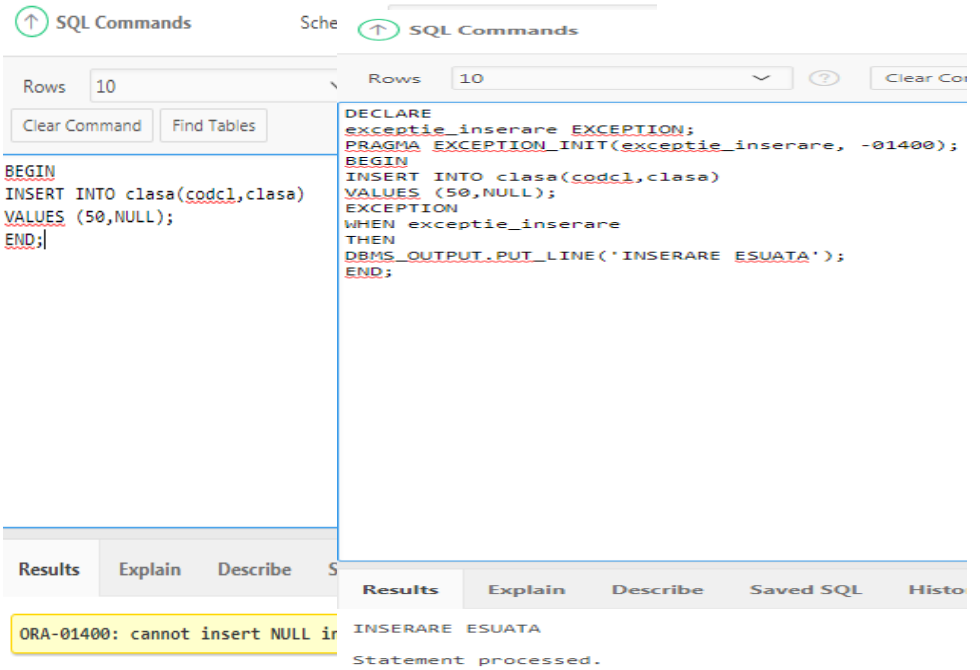


Fig. 3.1 Excepție non-predefinită generată de server-ul ORACLE

Excepțiile definite de utilizator se declară în secțiunea declarativă a blocului de program în care se anticipează că pot apărea erori la execuție și se lansează explicit prin folosirea cuvântului cheie RAISE, ca in *Figura 3.2.*

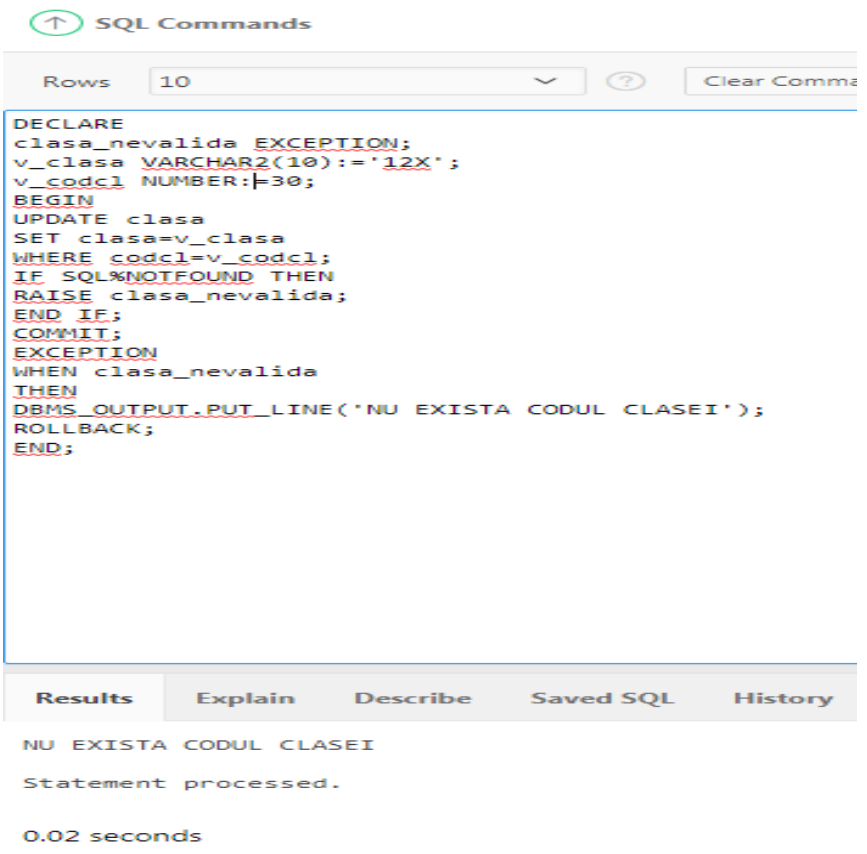


Figura 3.2 Excepție definită de utilizator

4. CURSORI ÎN ORACLE PL/SQL

Pentru afișarea mai multor rânduri de valori dintr-o tabelă prin intermediul unui bloc de program ORACLE PL/SQL, vom folosi un cursor explicit. Spre exemplificare, vom crea o aplicație ce va reține evidența cărților dintr-o bibliotecă(Figura 4.1).

Figura 4.1 Aplicația Biblioteca

Se vor crea două tabele, *carti_t* si *editura_t* în care se vor reține datele despre cărți, având structura prezentată în Figura 4.2 respectiv Figura 4.3.

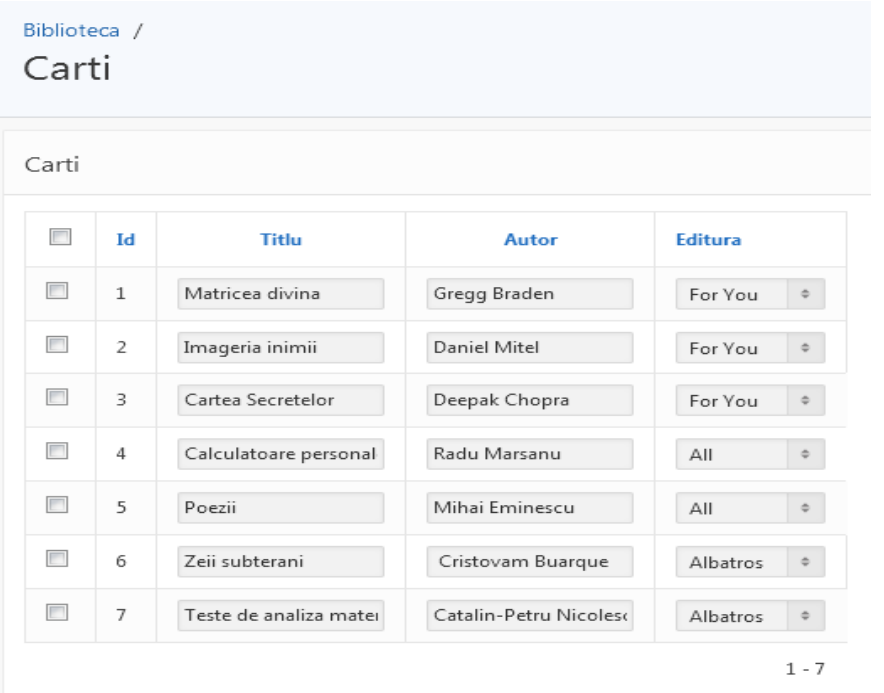


Figura 4.2 Tabela Cărți_t

***Figura 4.3** Tabela Editura_t*

În continuare, dorim să afișăm toate cărțile unei edituri date, printr-un bloc de program PL/SQL. Pentru aceasta, vom crea un cursor explicit(*Figura 4*-din SQL Command și *Fig 4.5*-din aplicație).

```
DECLARE
CURSOR carti_t_cursor IS
 SELECT  a.titlu, a.autor, b.editura from carti_t a, editura_t b
where (a.editura=b.id) and (b.editura=:Editura);
v_carti_t_record carti_t_cursor%ROWTYPE;
BEGIN
 OPEN carti_t_cursor;
 LOOP
 FETCH carti_t_cursor INTO v_carti_t_record;
 EXIT WHEN carti_t_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(v_carti_t_record.titlu || ' ' ||
v_carti_t_record.autor ||' '||v_carti_t_record.editura);
 END LOOP;
 CLOSE carti_t_cursor;
 END;
```

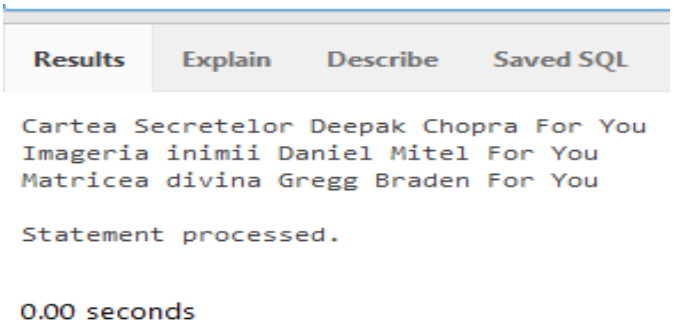


Figura 4.4 Cursor explicit

În aplicație, pentru crearea cursorului explicit vom folosi aceeași comandă ca în Fig 4.4, cu deosebirea că la afișare vom utiliza comanda *htp.p()* în locul comenzii *dbms_output.put_line().* Pentru alegerea editurii din lista editurilor disponibile în tabela editura_t, vom crea un item de tip listă pe care îl vom asocia acestei tabele. La alegerea editurii din itemul respectiv, se vor afișa cărțile disponibile din acea editură(Figura 4.5).

```
DECLARE
CURSOR carti_t_cursor IS
 SELECT a.titlu, a.autor, b.editura from carti_t a, editura_t b
where (a.editura=b.id) and (a.editura=:P5_EDITURA);
v_carti_t_record carti_t_cursor%ROWTYPE;
BEGIN
 OPEN carti_t_cursor;
 LOOP
 FETCH carti_t_cursor INTO v_carti_t_record;
 EXIT WHEN carti_t_cursor%NOTFOUND;
 htp.p('Titlu: ' || v_carti_t_record.titlu || ' - Autor: ' ||
v_carti_t_record.autor ||' - Editura: '||v_carti_t_record.editura);
htp.nl();
 END LOOP;
 CLOSE carti_t_cursor;
 END;
```

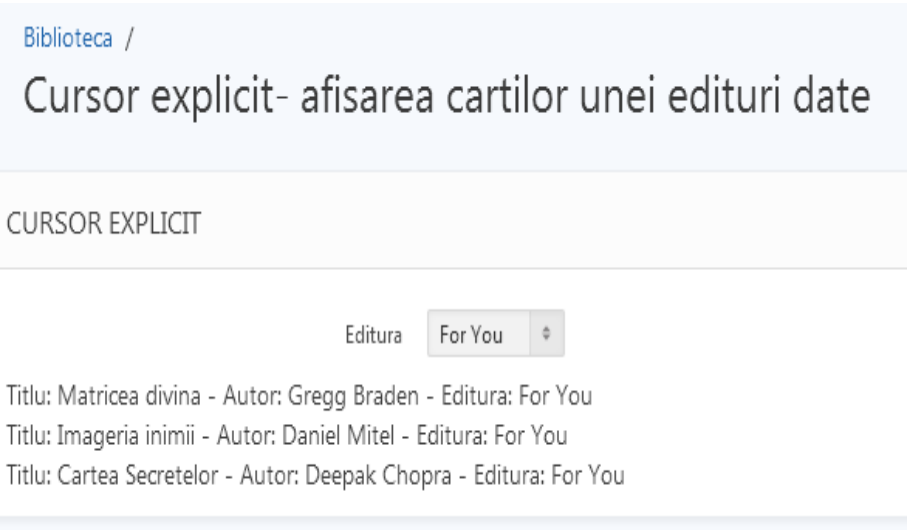


Figura 4.5 Cursor explicit creat din aplicație

5. ROW TRIGGERS IN ORACLE APEX

Ne propunem, în cele ce urmează, să creăm o aplicaţie în care să ţinem evidenţa jucătorilor şi a echipelor în care aceştia activează. În cazul în care un jucător se transferă la o altă echipă, acesta să nu fie primit dacă a mai evoluat anterior acolo. Pentru aceasta, este necesar să reţinem istoricul fiecărui jucător într-o tabelă ale cărei date se vor încărca automat la modificarea tabelei ce reţine evidenţa jucătorilor şi a echipelor în care aceştia activează.

Fie tabela *jucatori1* care reţine evidenţa jucătorilor şi a echipelor în care activează aceştia (Figura 5.1)

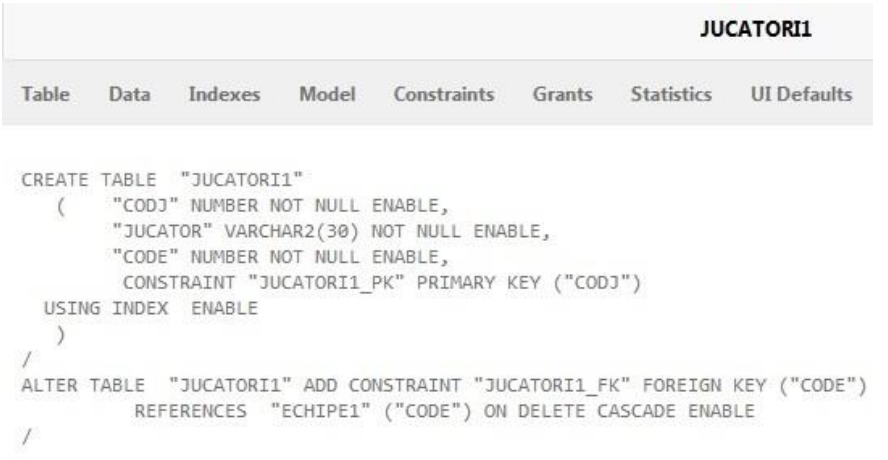


***Figura 5.1** Crearea tabelei Jucatori1*

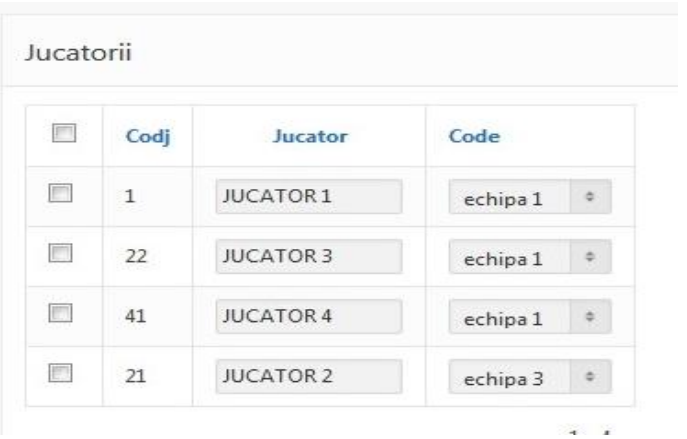


***Figura 5.2** Tabela Jucatori1*

Tabela *echipe1* reţine echipele în care pot activa jucătorii(Figura 5.4).

Figura 5.3 Crearea tabelei Echipe1

Figura 5.4 Tabela Echipe1

Tabela *istoric_echipe1* reţine istoricul fiecărui jucător şi echipele în care acesta a mai evoluat(Figura 5.6

```
CREATE TABLE  "ISTORIC_ECHIPE1"
   (    "COD" NUMBER NOT NULL ENABLE,
        "CODJ" NUMBER NOT NULL ENABLE,
        "CODE" NUMBER NOT NULL ENABLE,
         CONSTRAINT "ISTORIC_ECHIPE1_PK" PRIMARY KEY ("COD")
  USING INDEX  ENABLE
   )
/
```

Figura 5.5 Crearea tabelei Istoric_echipe1

Figura 5.6 Tabela istoric_echipe1

Pentru încărcarea automată a informaţiilor în tabela *istoric_exhipe1*, la fiecare actualizare a tabelei *jucători1* se execută trigger-ul *JUCATORI1_T1(Figura 5.7).*

Object Details Code Errors SQL

```
CREATE OR REPLACE TRIGGER  "JUCATORI1_T1"
AFTER
insert or update or delete on "JUCATORI1"
for each row
begin
BEGIN
   INSERT INTO istoric_echipe1
    (codj, code) VALUES (:NEW.codj, :NEW.code);
END;
end;

/
ALTER TRIGGER  "JUCATORI1_T1" ENABLE
/
```

Figura 5.7 Trigger-ul JUCATORI1_T1

În tabela *istoric_echipe1* se observă că *jucătorul 1* a mai evoluat anterior la *echipa 2*, iar în prezent face parte din *echipa 1*. În cazul în care dorim să-l transferăm înapoi la *echipa 2* în tabela *jucatori1*, se declanşează trigger-ul *jucatori1_T3*(Figura 5.7) care nu permite acest lucru(Figura 5.8).

```
create or replace trigger "JUCATORI1_T3"
BEFORE
UPDATE OF "CODE" ON "JUCATORI1"
for EACH ROW
DECLARE
    nrechipe INTEGER;
BEGIN
    SELECT count(*) into nrechipe
    FROM istoric_echipe1
    where (codj= :OLD.codj)
    AND (code= :NEW.code);
    IF (nrechipe>0) THEN
        RAISE_APPLICATION_ERROR (-20201,'Acest jucator ' || :OLD.jucator || ' a mai jucat in acea echipa' || :NEW.code);
    END IF;
END;
```

Figura 5.8 Trigger-ul jucatori1_T3

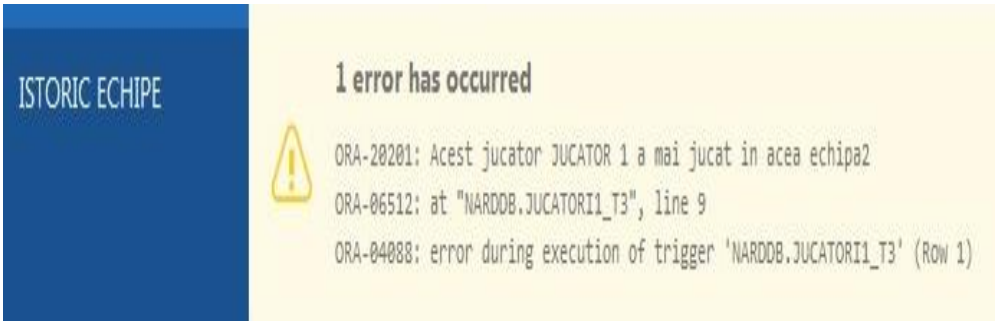


Figura 5.9 Declanșarea trigger-ului jucatori1_T3

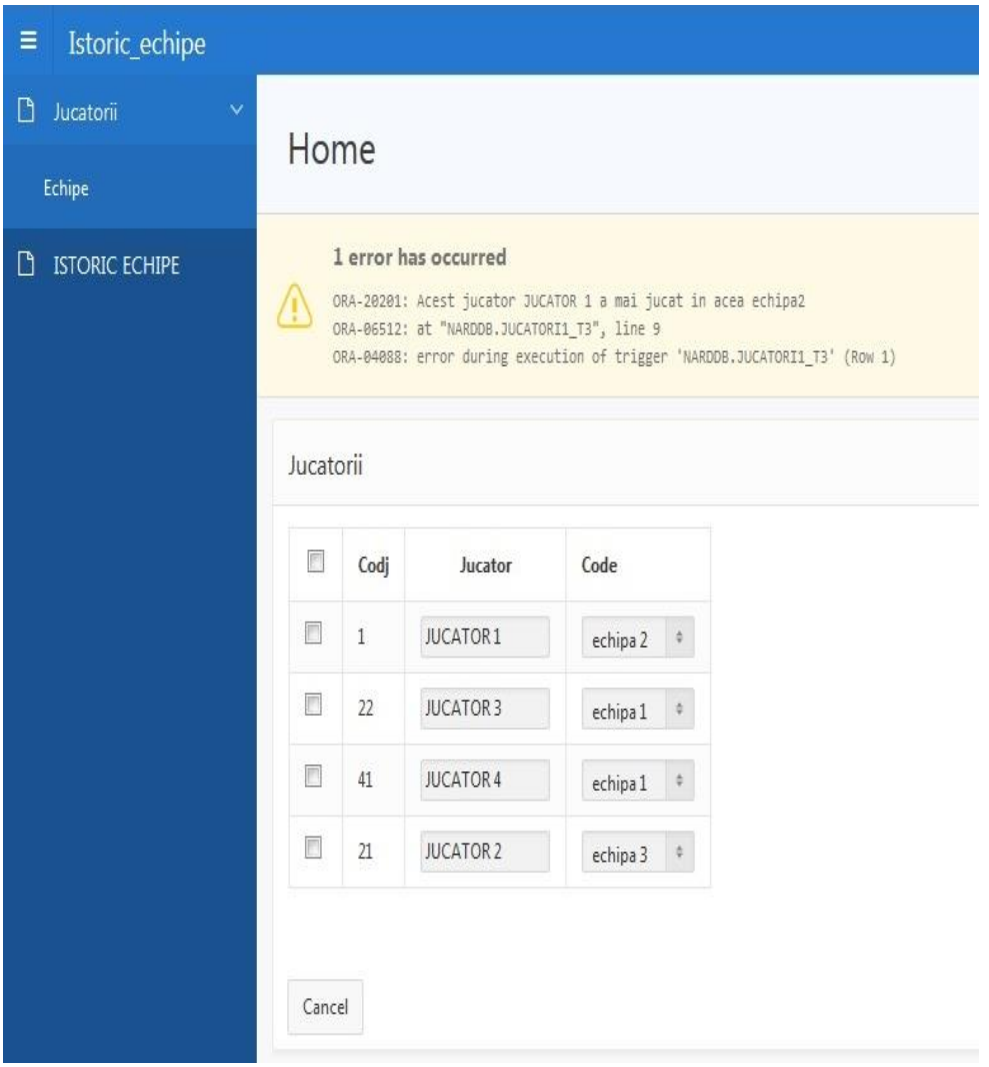


***Figura 5.10** Execuţia trigger-ului JUCATORI1_T3*

6. GRAFICE ȘI PROCESE ÎN APLICAŢII ORACLE APEX

În aplicaţia descrisă mai jos, ne propunem să construim o tabelă pe baza informaţiilor dintr-o altă tabelă, precum şi un grafic al acesteia.

Se consideră o bază de date în care se reţine evidenţa rezultatelor unei clase de elevi la testul iniţial pe o disciplină. Pentru fiecare elev se cunosc următoarele informaţii: *numele şi prenumele,* precum şi *nota* acestuia la testul iniţial respectiv. Ne propunem să realizăm o aplicaţie în ORACLE APEX prin care să putem gestiona şi vizualiza datele respective, precum şi rezultatele pe tranşe de note, la care vom ataşa un grafic.

Se vor crea tabelele *TEST_INIŢIAL1* şi *TRANŞE_DE_NOTE* în care vom reţine informaţiile de care avem nevoie.

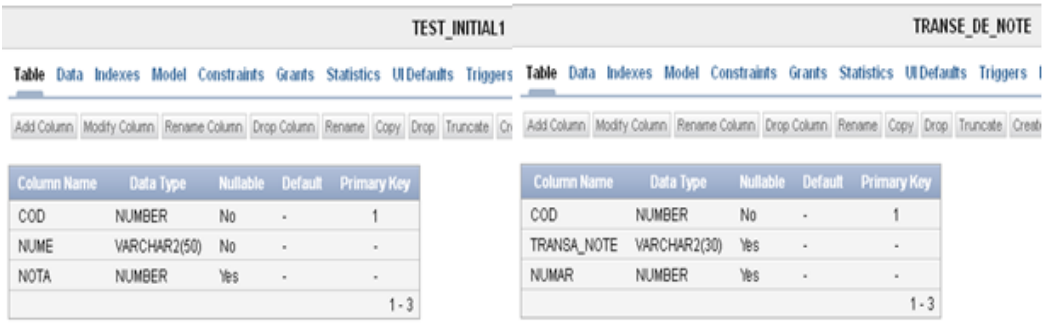


Figura 6.1 Structură tabele TEST_INIŢIAL1 și TRANŞE_DE_NOTE

În continuare vom construi aplicaţia *Tranşe de note* ce va include trei pagini:

- *O pagină de tip* Blank *ce va conţine intrările către celelalte două pagini*
- *O pagină de tip* Tabular form *prin intermediul căreia vom gestiona informaţiile din tabela* Test_initial1 *(pagina* rezultate*)*
- *O pagină de tip* Tabular form *prin intermediul căreia vom gestiona informaţiile din tabela* Transe_de_note*. Informaţiile din câmpul* Număr *vor conţine numărul de note cuprinse în intervalul specificat în câmpul* Transa_note*. Aceste informaţii*

vor fi introduse automat prin intermediul unui proces creat şi ataşat acestui tabular form.

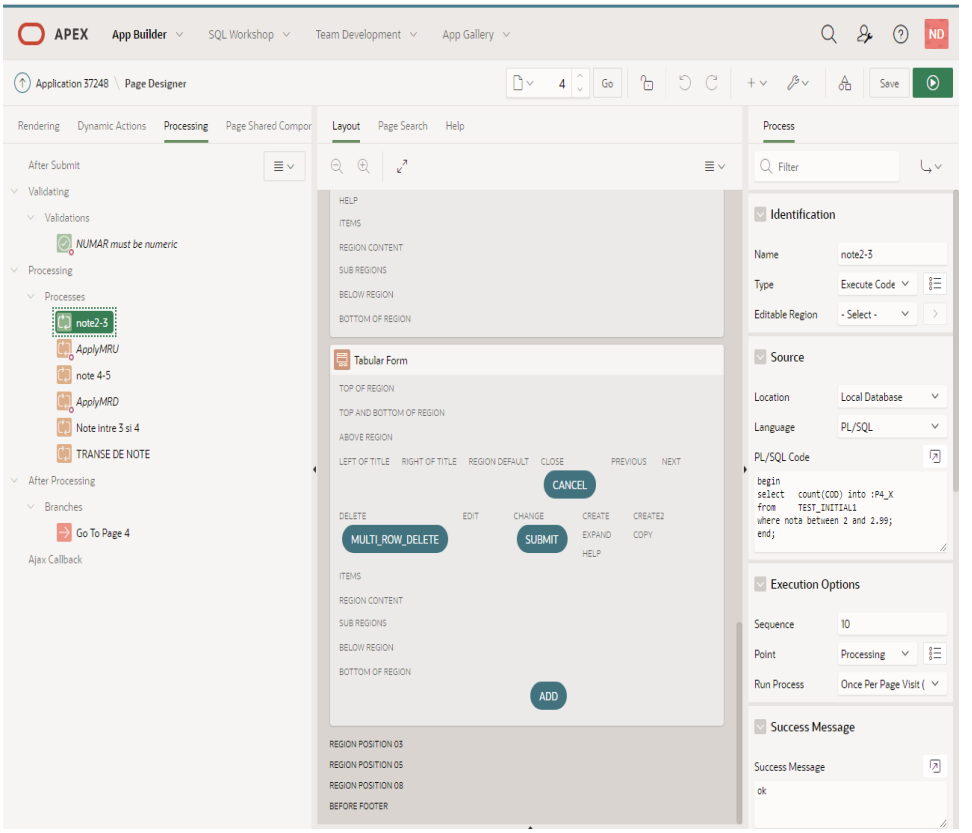


Figura 6.2 Procesul Note2-3

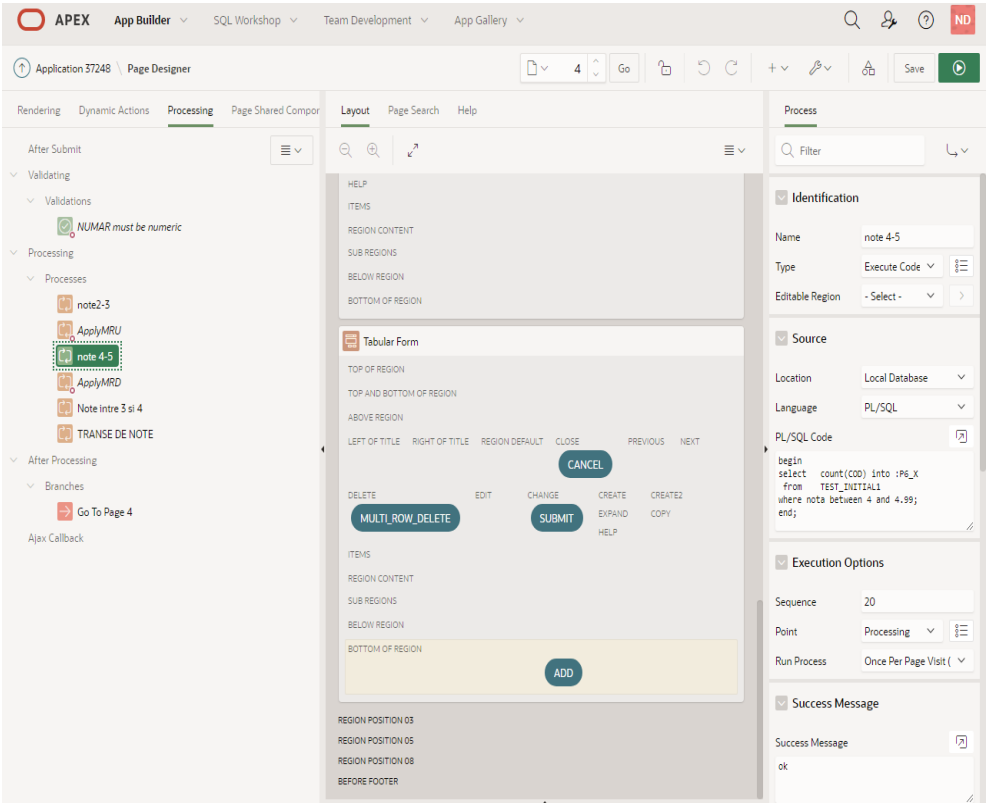


Figura 6.3 Procesul Note 4-5

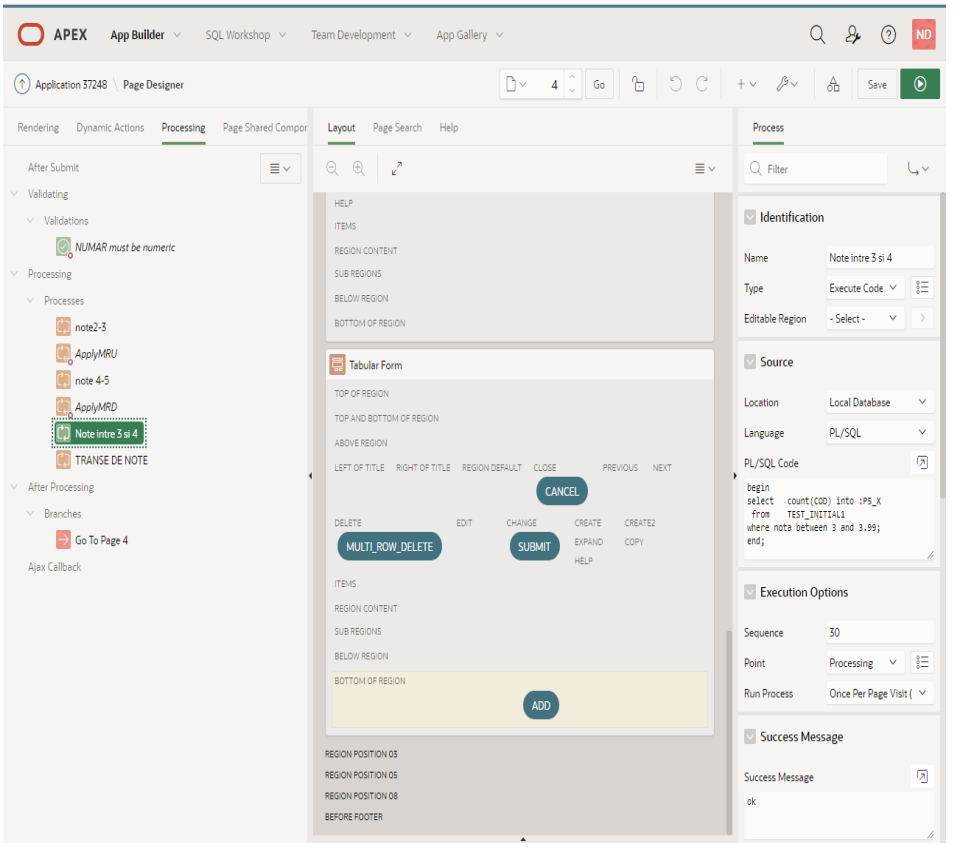


Figura 6.4 Procesul Note intre 3 si 4

Procesul TRANSE DE NOTE

```
declare
v1 number;
v2 number;
v3 number;
v4 number;
v5 number;
v6 number;
v7 number;
v8 number;
v9 number;
v10 number;
begin
select count(cod) into v1 from test_initial1 where nota between 1 and 1.99;
select count(cod) into v2 from test_initial1 where nota between 2 and 2.99;
select count(cod) into v3 from test_initial1 where nota between 3 and 3.99;
select count(cod) into v4 from test_initial1 where nota between 4 and 4.99;
select count(cod) into v5 from test_initial1 where nota between 5 and 5.99;
select count(cod) into v6 from test_initial1 where nota between 6 and 6.99;
 select count(cod) into v7 from test_initial1 where nota between 7 and 7.99;
select count(cod) into v8 from test_initial1 where nota between 8 and 8.99;
```

```
select count(cod) into v9 from test_initial1 where nota between 9 and 9.99;
select count(cod) into v10 from test_initial1 where nota=10;
update transe_de_note
set numar = v1 where transa_note = 'intre 1-1.99';
update transe_de_note
set numar = v2 where transa_note = 'intre 2-2.99';
update transe_de_note
set numar = v3 where transa_note = 'intre 3-3.99';
update transe_de_note
set numar = v4 where transa_note = 'intre 4-4.99';
update transe_de_note
set numar = v5 where transa_note = 'intre 5-5.99';
update transe_de_note
set numar = v6 where transa_note = 'intre 6-6.99';
update transe_de_note
set numar = v7 where transa_note = 'intre 7-7.99';
update transe_de_note
set numar = v8 where transa_note = 'intre 8-8.99';
update transe_de_note
set numar = v9 where transa_note = 'intre 9-9.99';
update transe_de_note
set numar = v10where transa_note = 'nota 10';
end;
```

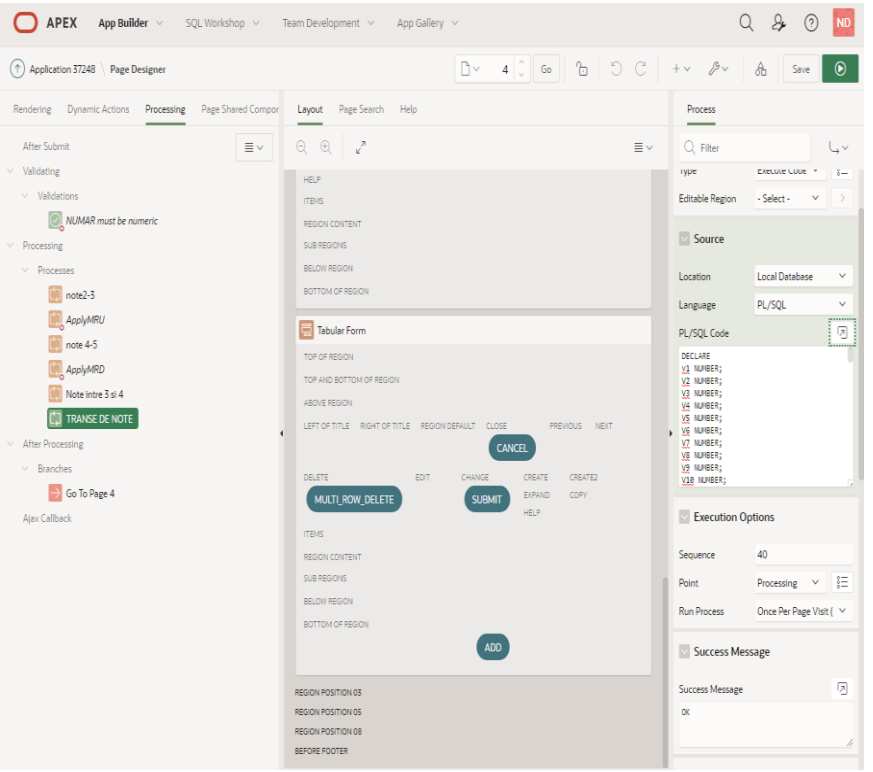


Figura 6.5 Procesul TRANSE DE NOTE

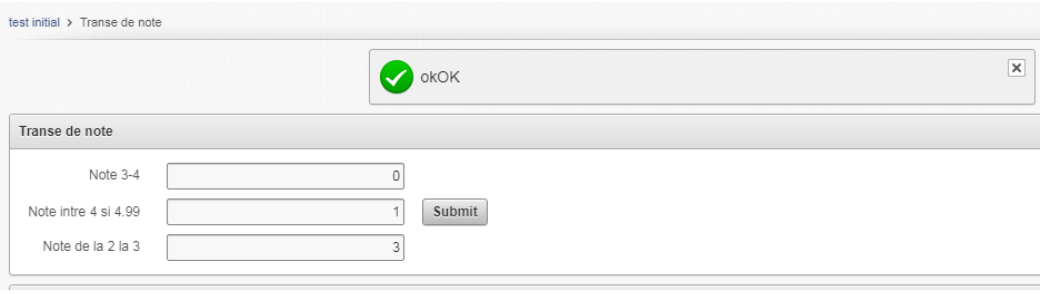


Figura 6.6 Executia proceselor prin intermediul unor itemi

Ne propunem în continuare să ataşăm un grafic la tabela *Transe_de_note*. În graficul respective dorim să afişăm numărul de note din fiecare tranşă prezentă în tabela *Transe_de_note*.

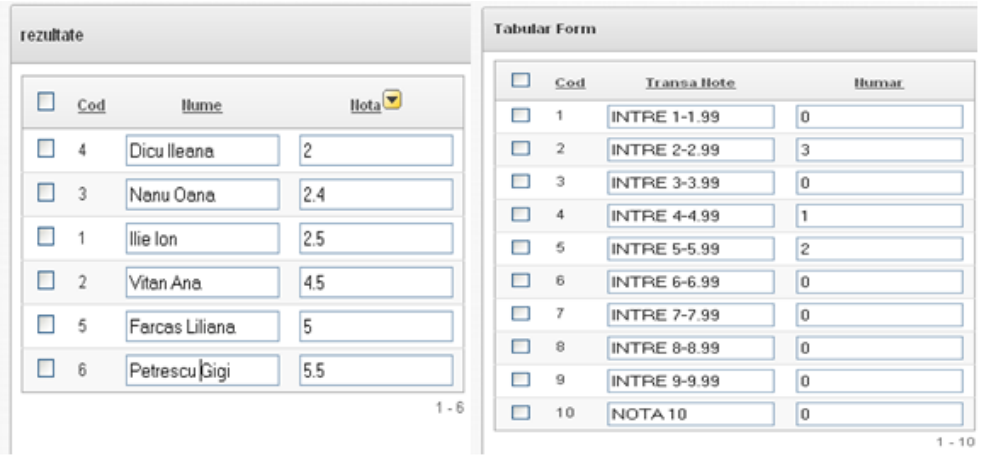


Figura 6.7 Tabelele Test_initial1 si Transe_de_note

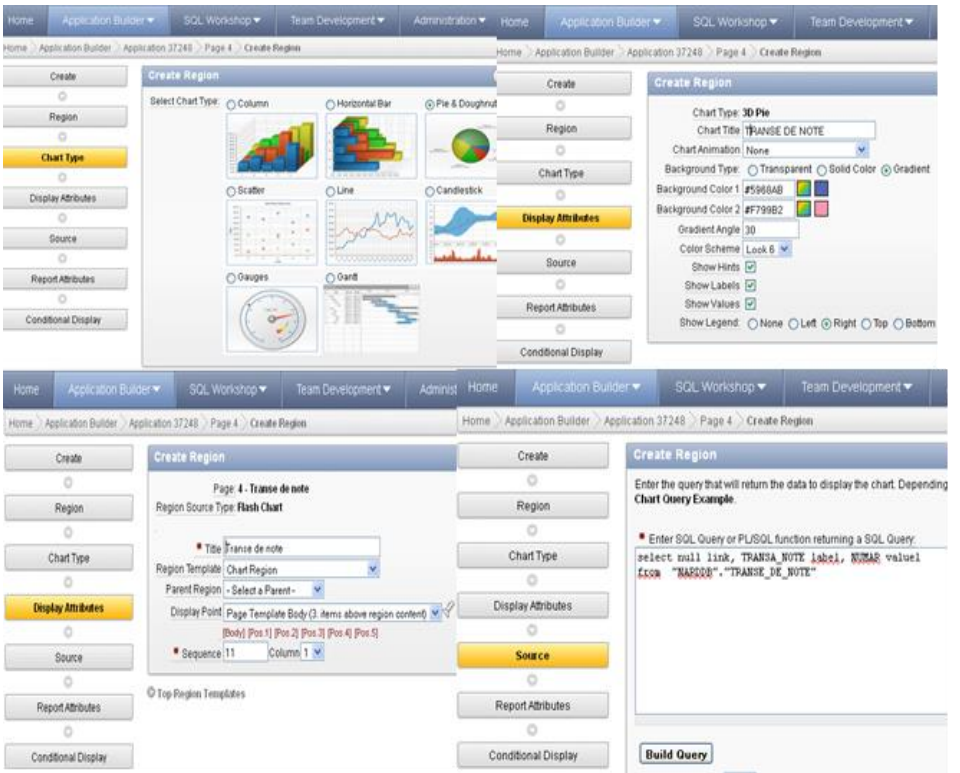


Figura 6.8 Ataşarea unui grafic la tabela Transe_de_note

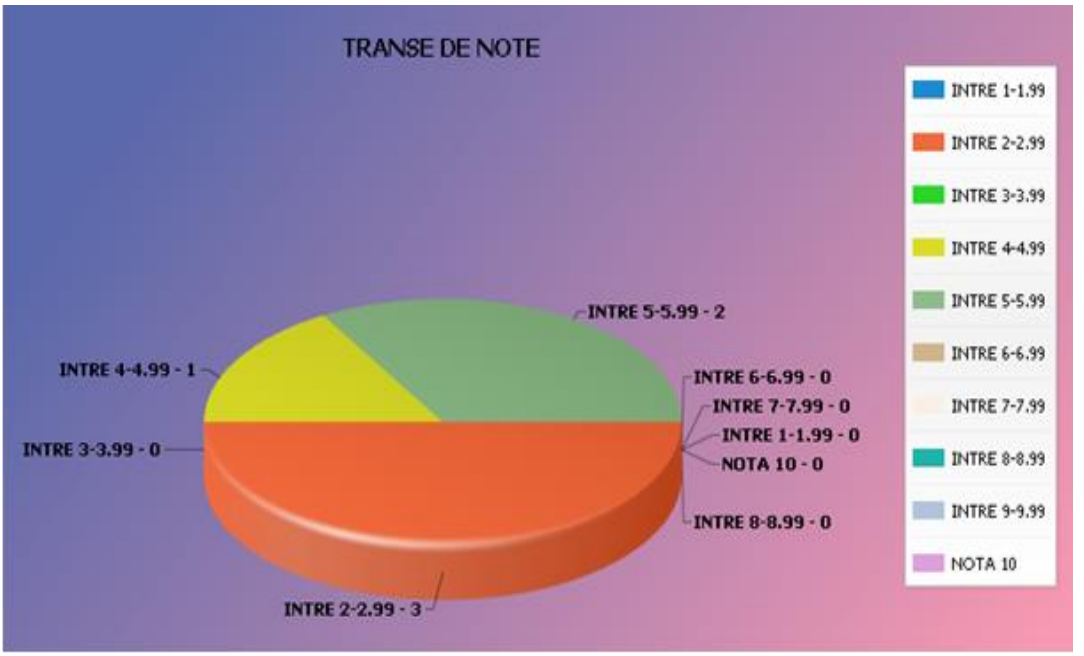


Figura 6.9 Vizualizarea graficului

La o modificare a tabelei *Test_initial1* se vor modifica automat datele din câmpul *Număr* al tabelei *Transe_de_note*, precum şi graficul ataşat acesteia.

rezultate

Cod	Nume	Nota
4	Dicu Ileana	2
3	Nanu Oana	2.4
1	Ilie Ion	2.5
2	Vitan Ana	4.5
5	Farcas Liliana	5
6	Petrescu Gigi	5.5
24	Tinca Eduard	8
21	Mihaila Carmen	9.55
22	Sandu Ion	10
23	Draghici Cristina	10

1 - 10

Figura 6.10 Tabela Test_initial1 modificată

Tabular Form

Cod	Transa Note	Numar
1	INTRE 1-1.99	0
2	INTRE 2-2.99	3
3	INTRE 3-3.99	0
4	INTRE 4-4.99	1
5	INTRE 5-5.99	2
6	INTRE 6-6.99	0
7	INTRE 7-7.99	0
8	INTRE 8-8.99	1
9	INTRE 9-9.99	1
10	NOTA 10	2

Figura 6.11 Tabela Transe_de_note modificată

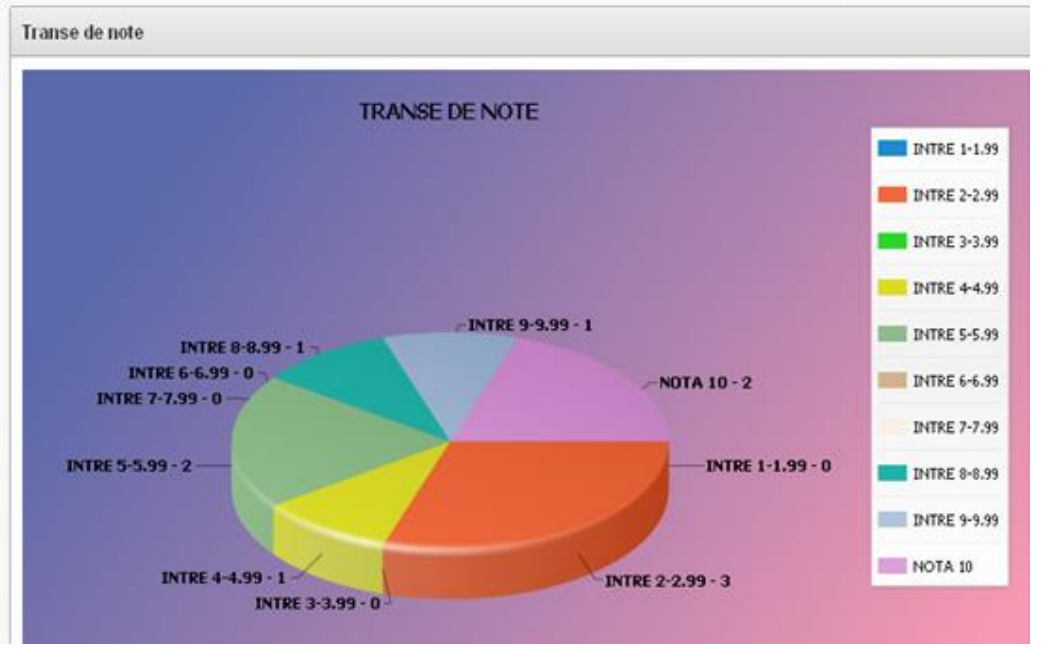


Figura 6.12 Noul grafic rezultat în urma modificării

7. FOLOSIREA LISTELOR DE VALORI SI A PROCESELOR IN APLICATII ORACLE APEX

Nu de puţine ori, în cadrul unei aplicaţii ORACLE, atunci când dorim să introducem valori într-un câmp ce reprezintă o cheie străină într-o tabelă, ne întrebăm: *Ce valoare are câmpul de legătură ce reprezintă cheia primară în tabela de referinţă(tabela părinte), pentru ceea ce dorim să introducem în câmpul respectiv din tabela-fiu?* Ar fi cu mult mai uşor dacă am avea la dispoziţie o listă din care să putem selecta aceste valori, la introducerea datelor într-un câmp ce reprezintă o cheie străină. Ne propunem în aplicaţia descrisă mai jos, să exemplificăm acest lucru, prin folosirea listelor de valori. Totodată vom realiza completarea automată a datelor din anumite câmpuri ale unei tabele, prin intermediul proceselor create în cadrul aplicaţiei.

Se consideră o bază de date în care se reţine evidenţa elevilor la un concurs şcolar. Pentru fiecare elev se cunosc următoarele informaţii: *numele şi prenumele, clasa, nota1, nota2, media* precum şi *rezultatul* obţinut în cadrul concursului. Ne propunem să realizăm o aplicaţie în APEX prin care să putem gestiona şi vizualiza datele respective.

Se vor crea două tabele *CLASE,* respectiv *ELEVII* în care vom reţine informaţiile de care avem nevoie. Legătura dintre cele două tabele se va face prin intermediul câmpului *codclasa* ce va fi cheie străină în tabela *ELEVII* şi care va face referire la câmpul *cod*, cheia primară din tabela *CLASE*.

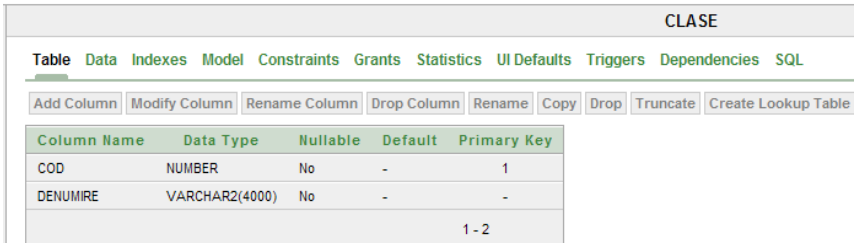


Figura 7.1 Tabela CLASE

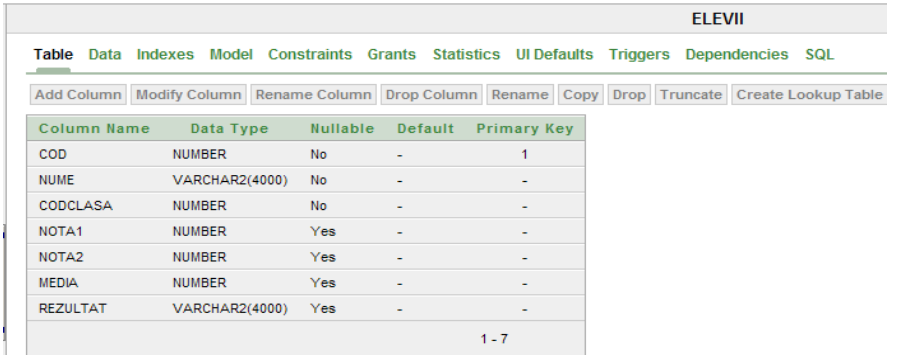


Figura 7.2 Tabela ELEVII

În continuare vom construi aplicaţia *CONCURS* ce va include 4 pagini:

- pagină de tip *Blank* ce va conţine un cuprins din care vom accesa celelalte pagini

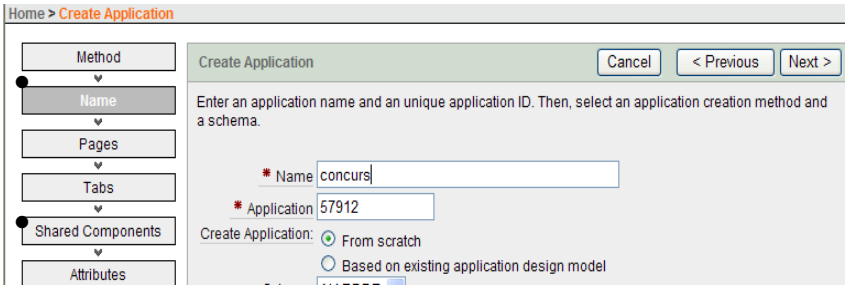


Figura 7.3 Crearea aplicaţiei CONCURS

- Două pagini de tip Tabular Form prin intermediul cărora vom actualiza datele din cele două tabele

- O pagină de tip *Report* prin intermediul căreia vom vizualiza informaţiile din cele două tabele

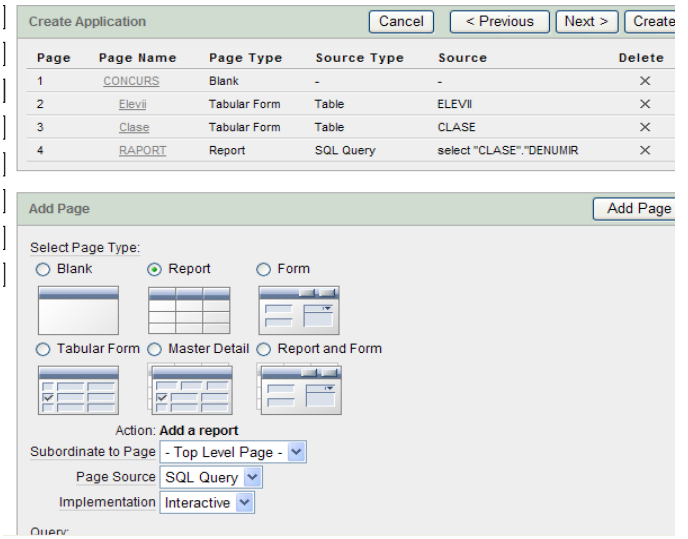


Figura 7.4 Paginile aplicaţiei

Aplicaţia va arăta în felul următor:

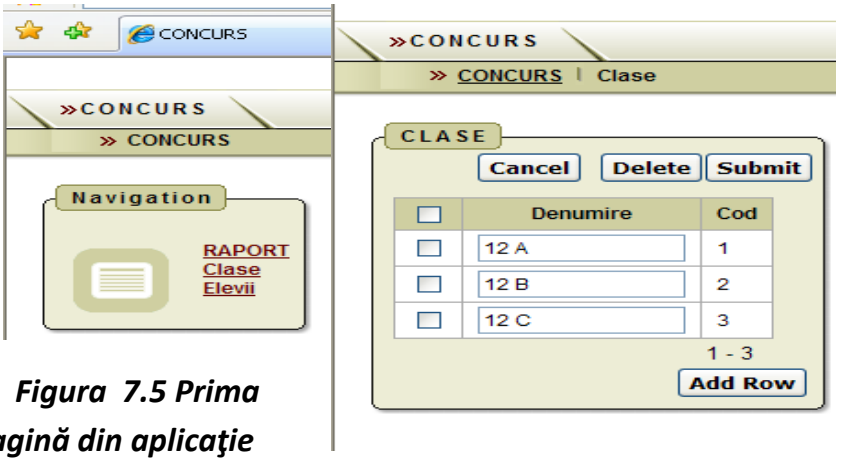


Figura 7.5 Prima pagină din aplicaţie

Figura 7.6 A treia pagină din aplicaţie

Pentru introducerea datelor în tabela ELEVII ne propunem să creăm o listă de valori pentru a selecta clasa din care face parte fiecare elev.

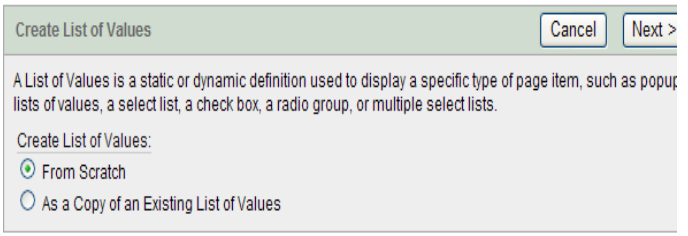


Figura 7.7 Crearea listei de valori

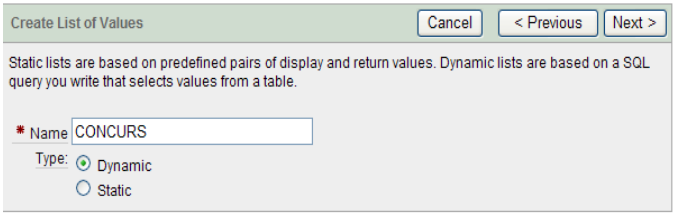


Figura 7.8 Selectarea tipului listei de valori

Selectăm tabela- referinţă, câmpul ale cărui valori dorim să se afişeze în listă, precum şi câmpul ale cărui valori se vor introduce în fundal(în câmpul FK).

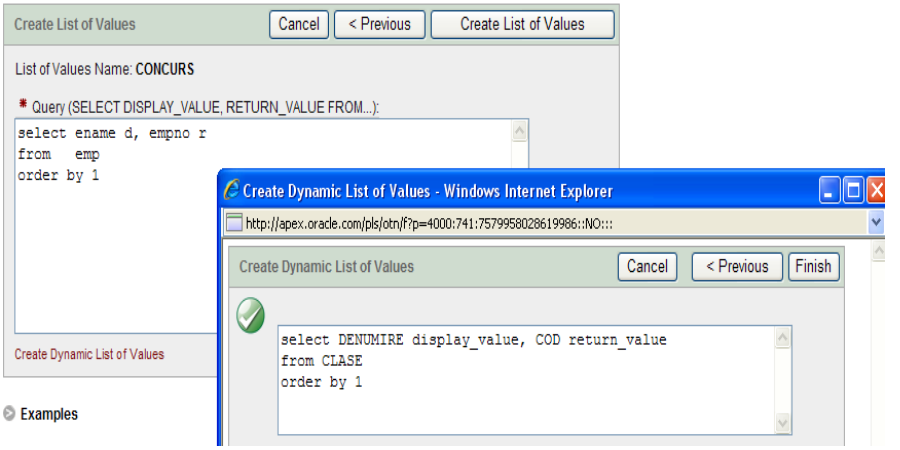


Figura 7.9 Descrierea listei de valori

În continuare, editäm atributele câmpului *codclasa* din tabela *ELEVII*, ale cărui valori se vor introduce din lista de valori creată anterior

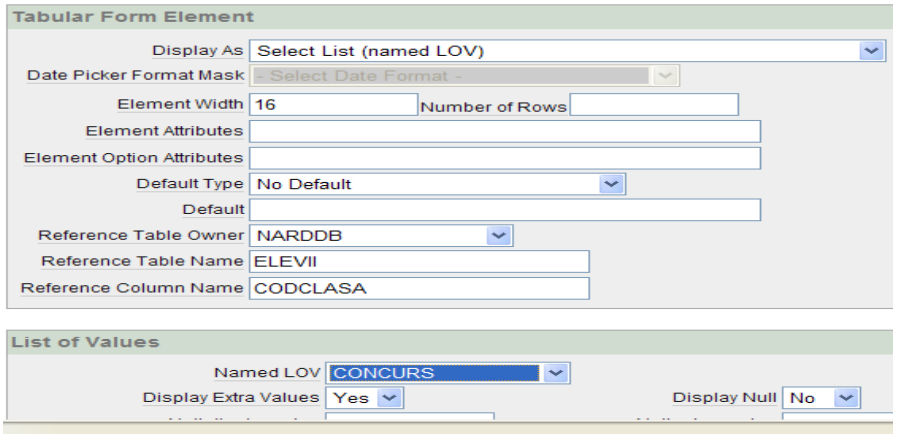


Figura 7.10 Selectarea listei de valori

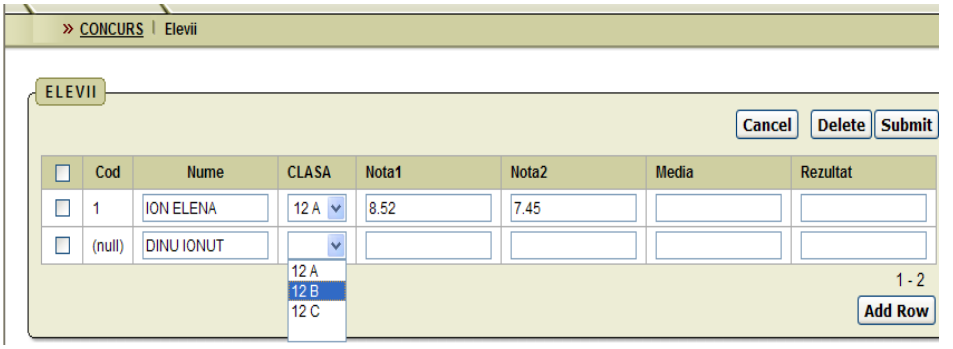


Figura 7.11 Folosirea listei de valori creată

Pentru completarea automată a valorilor din câmpurile *Media* şi *Rezultat* în cardul tabelei *Elevii*, vom crea un PL/SQL proces în aplicaţie.

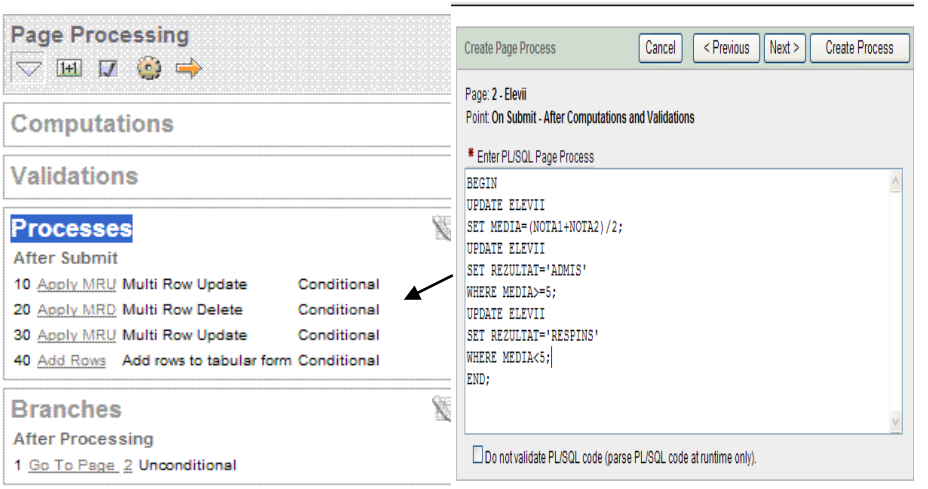


Figura 7.12 Crearea unui proces

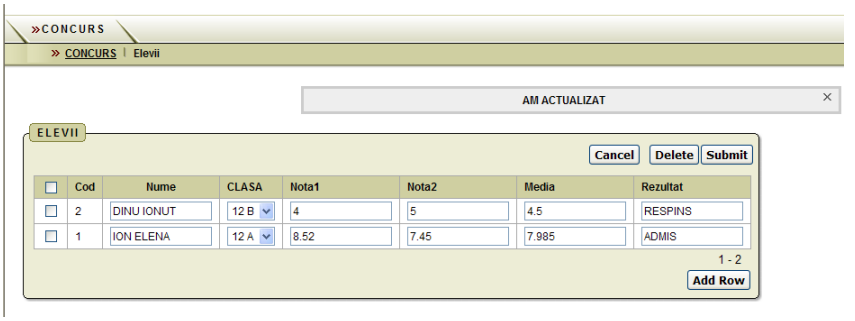


Figura 7.13 Execuţia procesului în cadrul aplicaţiei

8. ITEMI CU ACȚIUNE DINAMICĂ ÎN ORACLE APEX

La introducerea datelor în formularele unei aplicaţii cu baze de date, precum şi la afişarea acestora, apare uneori necesitatea controlării acestor operaţii, în funcţie de anumite condiţii pe care trebuie să le îndeplinească aceste date. Introducerea datelor sau afişarea acestora în câmpurile unui formular poate fi condiţionată prin crearea unor **acţiuni dinamice** pe care le vom ataşa itemilor ce reţin acele câmpuri.

În cele ce urmează, ne propunem să exemplificăm acest lucru prin intermediul unei aplicaţii create în *ORACLE APEX 4.0.* Aplicaţia respectivă va gestiona datele unui examen alcătuit din patru probe, din care primele două sunt eliminatorii şi au ca rezultat ADMIS sau RESPINS. La introducerea datelor, în cazul în care un candidat nu a promovat una dintre cele două probe eliminatorii, dorim ca următoarele câmpuri din formular să fie dezactivate, iar rezultatul să fie afişat automat. Un candidat se va considera admis, în cazul în care va promova cele două probe eliminatorii şi va avea minim 5 la următoarele două probe, iar media va fi minim 6. Valoarea mediei se va calcula automat pentru candidaţii admişi la probele eliminatorii. Aplicaţia va gastiona datele din două tabele, *Rezultat* şi *Examen* ce vor reţine valorile rezultatului unui examen, respectiv datele candidaţilor la examen.

Tabelele *Rezultat şi Examen* au structura prezentată în Figura 8.1, respectiv Figura 8.2.

Column Name	Data Type	Nullable	Default	Primary Key
COD	NUMBER	No	-	1
REZULTAT	VARCHAR2(10)	No	-	-
				1 - 2

Figura 8.1 Tabela Rezultat

Column Name	Data Type	Nullable	Default	Primary Key
COD	NUMBER	No	-	1
CANDIDAT	VARCHAR2(50)	No	-	-
PROBA_A	NUMBER	No	-	-
PROBA_B	NUMBER	Yes	-	-
PROBA_C	NUMBER	Yes	-	-
PROBA_D	NUMBER	Yes	-	-
MEDIA	NUMBER	Yes	-	-
REZULTAT	VARCHAR2(50)	Yes	-	-

Figura 8.2 Tabela Examen

Create Application

Page	Page Name	Page Type	Source Type	Source	Delete
1	Examen	Report	Table	EXAMEN	×
2	Examen	Form	Table	EXAMEN	×

Figura 8.3 Aplicaţia EXAMEN

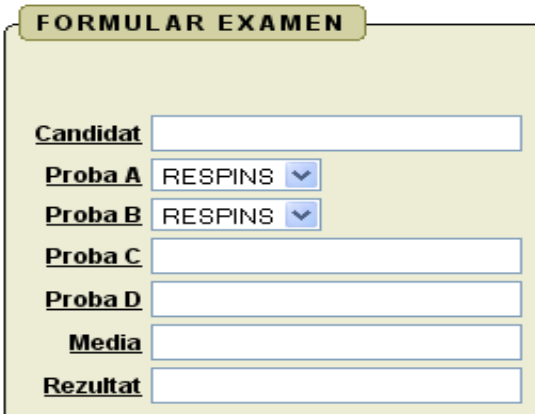


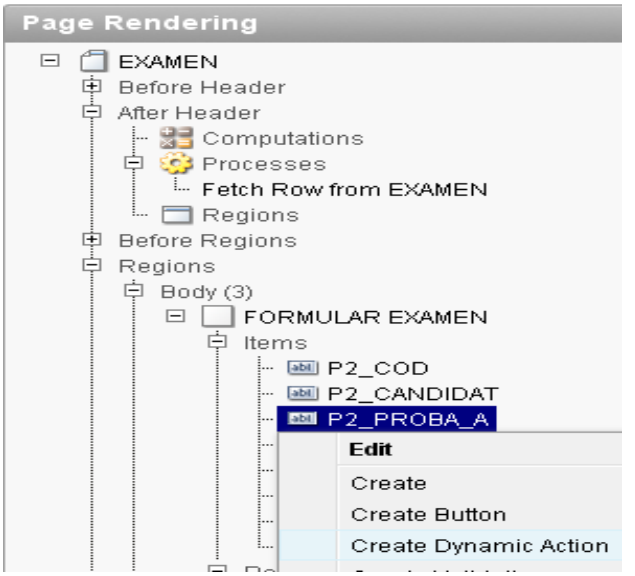
Figura 8.4 Formularul EXAMEN

Se crează aplicaţia *EXAMEN* ce are în componenţa sa un raport şi un formular.

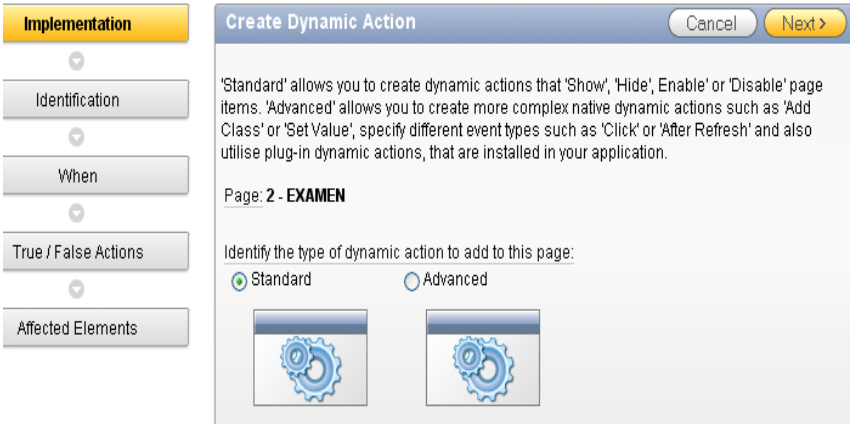
În continuare, vom edita itemul *P2_PROBA_A* corespunzător câmpului **Proba A** din cadrul formularului, astfel încât să putem selecta pentru acesta, unul din rezultatele *ADMIS* sau *RESPINS* din tabela *REZULTAT*. Pentru aceasta vom crea o listă dinamică de valori pe care o vom ataşa acestui item. Analog vom proceda pentru itemul *P2_PROBA_B* corespunzător **Probei B** din formular.

Formularului de introducere a datelor va arăta ca în Figura 8.4. Având în vedere faptul că primele două probe din examen sunt eliminatorii, dorim să blocăm introducerea datelor în continuare pentru candidaţii ce nu au promovat la una din cele două probe, dezactivând următoarele câmpuri din formular. Pentru aceasta

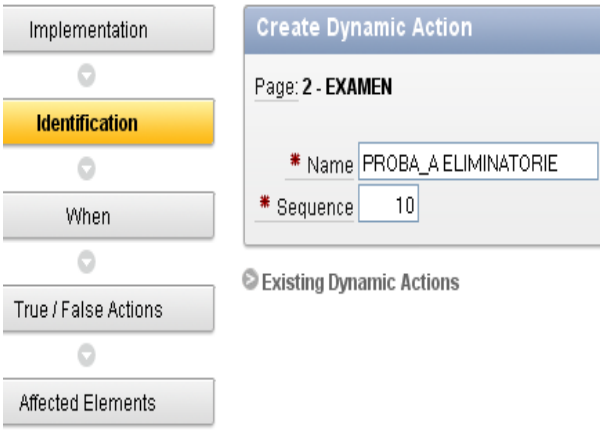
vom crea o actiune dinamică standard, ataşată itemului PROBA_A. Analog vom proceda pentru itemul PROBA_B.



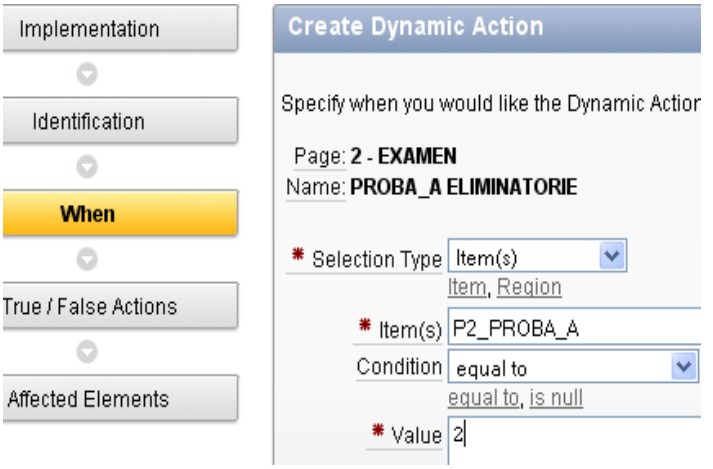
Pasul 1: Selectarea itemului şi ataşarea unei acţiuni dinamice



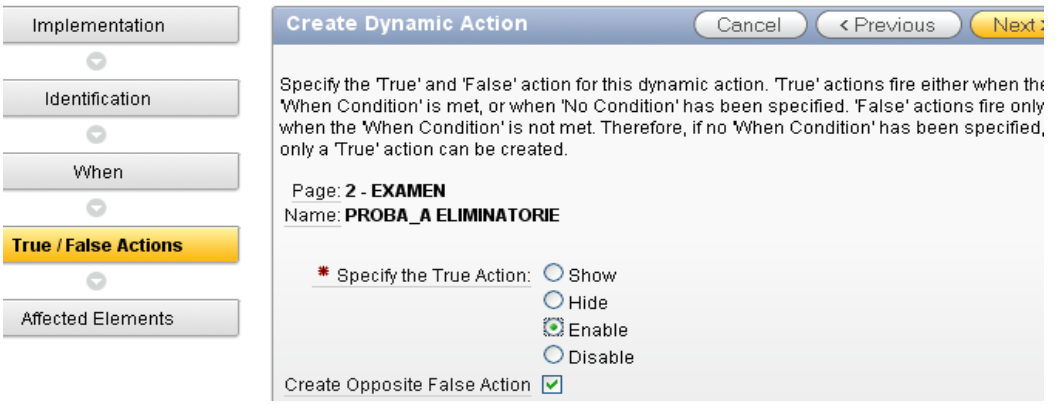
Pasul 2: Crearea unei acţiuni dinamice standard



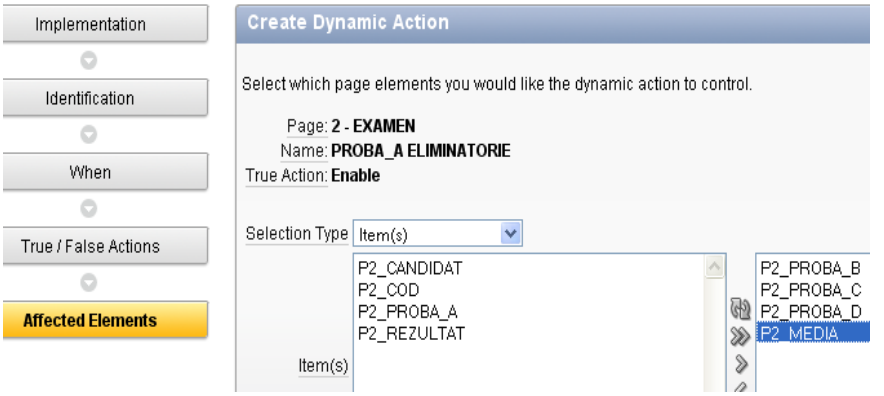
Pasul 3: Denumirea acţiunii dinamice



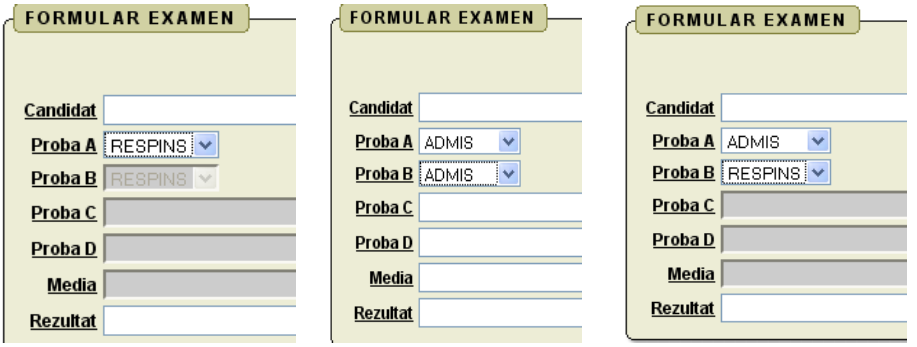
Pasul 4: Stabilirea valorii itemului respectiv, pentru care se va declanşa acţiunea dinamică



Pasul 5: Selectarea acţiunii(Enable) ce se va declanşa la valoarea selectată la pasul anterior, pentru itemul respectiv



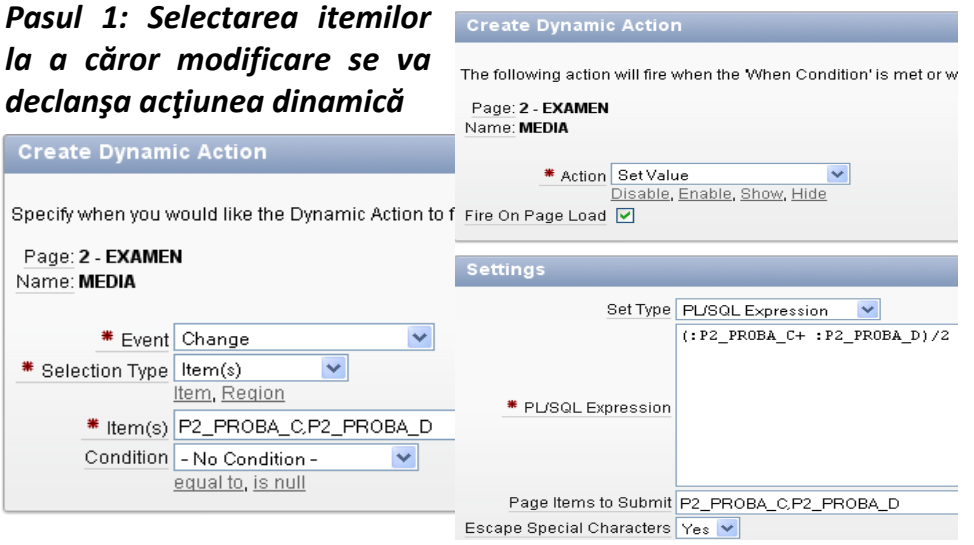
Pasul 6: Selectarea itemilor pentru care se va declanşa acţiunea dinamică nou creată

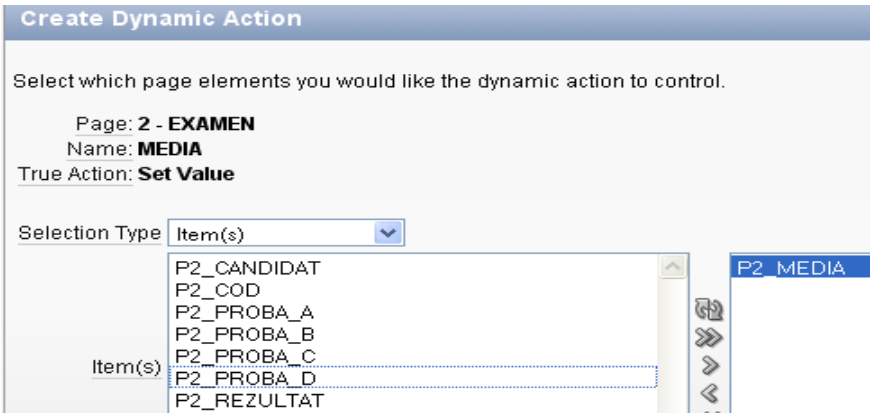


Pasul 7: Declanşarea acţiunii dinamice pentru itemul corespunzător câmpului Proba A din formularul aplicaţiei

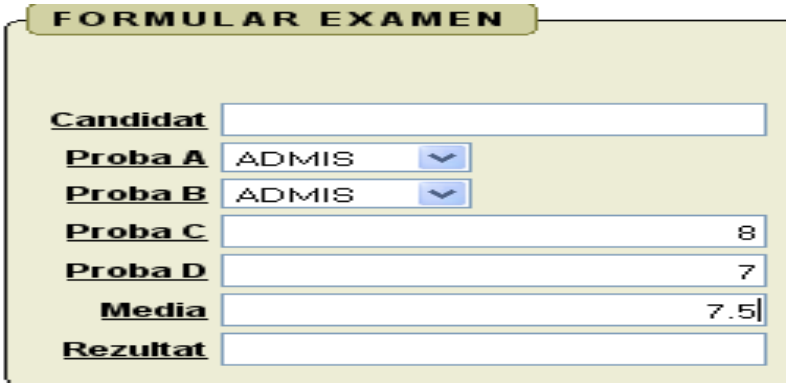
În continuare vom crea o acţiune dinamică de tip avansat, pentru câmpul **Media,** ce va avea ca efect calculul automat al mediei probelor scrise, în cazul în care se vor edita notele în câmpurile respectivelor probe.

Pasul 1: Selectarea itemilor la a căror modificare se va declanşa acţiunea dinamică





Pasul 2: Selectarea acţiunii ce se va declanşa pentru câmpul Media(calculul mediei folosind o expresie PL/SQL)
Selectarea itemului corespunzător câmpului Media, unde se va declanşa acţiunea dinamică



Pasul 3: Declanşarea acţiunii dinamice ataşată câmpului Media din cadrul formularului

În continuare, în câmpul *Rezultat* dorim să se afişeze automat rezultatul la examen pentru un candidat. Pentru aceasta, vom crea în mod analog, o acţiune dinamică de tip avansat pe care o vom ataşa acestui câmp. Această acţiune dinamică se va declanşa la orice modificare a informaţiilor din câmpurile ce reţin rezultatele la probele de examen. Pentru crearea acestei acţiuni dinamice vom folosi următoarea funcţie PL/SQL ce va întoarce valoarea pentru câmpul *Rezultat*:

```
declare
r1 Varchar2(50);
r2 varchar2(50);
r varchar2(50);
begin
select rezultat into r1 from rezultat where :P2_PROBA_A=cod;
select rezultat into r2 from rezultat where :P2_PROBA_B=cod;
if (r1=r2)and(r1='ADMIS')and(:P2_PROBA_C>=5)AND
(:P2_PROBA_D>=5)AND(:P2_MEDIA>=6)
    then r:='ADMIS' ;
    else r := 'RESPINS';
end if;
return r;
END;
```

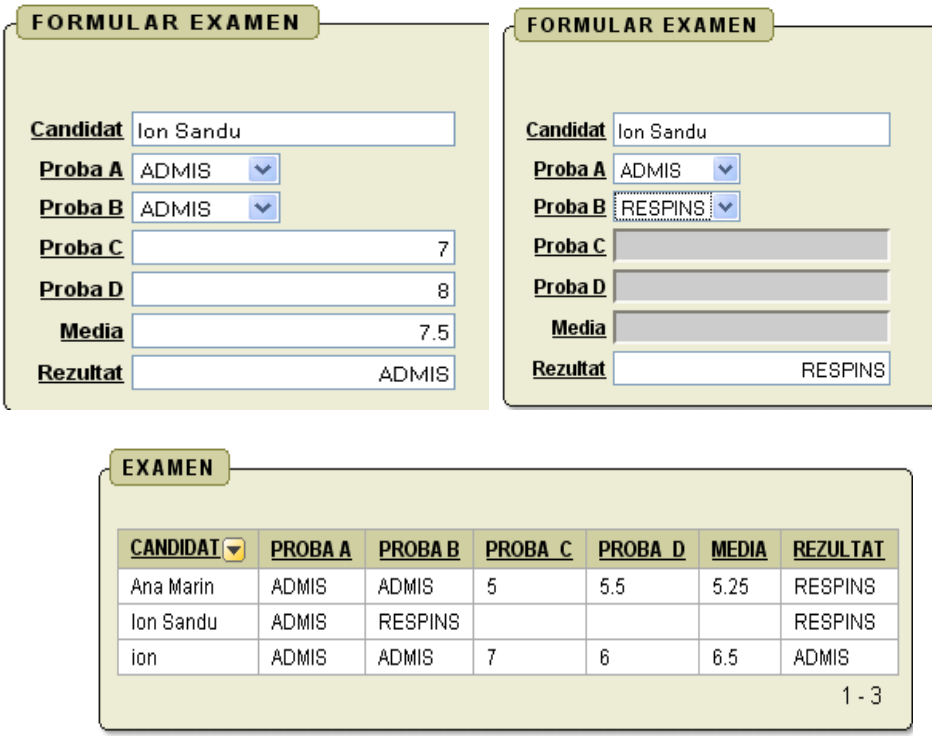


Figura 8.5 Aplicaţia Examen

BIBLIOGRAFIE

1. **ORACLE ACADEMY-** *Database Programming with PL\SQL* **,** https://academy.ORACLE.com/, 2018
2. **ORACLE® APPLICATION EXPRESS** 18.2 - http://apex.oracle.com/