**MINISTERUL EDUCAŢIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Programul de studii: Tehnologia informației**



**RAPORT**

**Disciplina „Baza de Date 2”**

**Tema: Crearea și Utilizarea Declanșatoarelor**

|  |  |  |
| --- | --- | --- |
| **Student(ă):** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Vlași**ț**chi Ștefan , TI-216** |
|  |  |  |
| **Coordonator universitate:** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Cebotar Gabriela, asist.univ.** |

**Chișinău, 2024**

**Obiective**:

**Sarcini practice:**

1. Să se creeze o procedură stocată fără parametri. Procedura calculează numărul tuturor profesorilor la facultate. Rezultatul execuției in figura 1

-- Șterge procedura stocată existentă (dacă dorești să o înlocuiești)

-- DROP PROCEDURE IF EXISTS pentru a elimina procedura stocată existentă

IF OBJECT\_ID('CalculeazaNumarProfesori', 'P') IS NOT NULL

BEGIN

DROP PROCEDURE CalculeazaNumarProfesori;

END

GO

-- CREATE PROCEDURE pentru a crea procedura stocată

CREATE PROCEDURE CalculeazaNumarProfesori

AS

BEGIN

DECLARE @NumarProfesori INT

SELECT @NumarProfesori = COUNT(\*) FROM profesori;

PRINT 'Numarul total al profesorilor la facultate este: ' + CAST(@NumarProfesori AS VARCHAR);

END;



Figura 1 – Rezultatul codului executat

1. Să se creeze o procedură stocată, care nu are niciun parametru de intrare și posedă un parametru de ieșire. Parametrul de ieșire trebuie să returneze numărul de studenți, care nu au susținut cel puțin o formă de evaluare (nota mai mică de 5 sau valoare NULL). Rezultatul execuției in figura 2

CREATE PROCEDURE NumarStudentiFaraEvaluare

@NumarStudentiFaraEvaluare INT OUTPUT

AS

BEGIN

-- Declara variabila pentru stocarea numarului de studenti fara evaluare

DECLARE @CountStudentiFaraEvaluare INT;

-- Calculeaza numarul de studenti care nu au sustinut cel putin o forma de evaluare

SELECT @CountStudentiFaraEvaluare = COUNT(\*)

FROM studenti\_reusita

WHERE Nota < 5 OR Nota IS NULL;

-- Seteaza valoarea parametrului de iesire

SET @NumarStudentiFaraEvaluare = @CountStudentiFaraEvaluare;

END;

DECLARE @NumarStudenti INT;

EXEC NumarStudentiFaraEvaluare @NumarStudenti OUTPUT;

PRINT 'Numarul de studenti fara evaluare este: ' + CAST(@NumarStudenti AS VARCHAR);



Figura 2 - Rezultatul codului executat

1. Să se creeze o procedură stocată cu un parametru de intrare. De exemplu, trebuie numiți (numele, prenumele) profesorii care predau cel puțin două discipline. Parametrul (număr\_discipline) specifică un număr de întreaga mai mare decât 1. Rezultatul execuției in figura 3

CREATE PROCEDURE ProfesoriCuMaiMulteDiscipline

@numar\_discipline INT

AS

BEGIN

-- Verifică dacă numărul de discipline este valid (mai mare decât 1)

IF @numar\_discipline <= 1

BEGIN

PRINT 'Numărul de discipline trebuie să fie mai mare decât 1.';

RETURN;

END;

-- Selectează profesorii care predau cel puțin numărul specificat de discipline distincte

SELECT p.Nume\_Profesor, p.Prenume\_Profesor

FROM profesori p

JOIN (

SELECT Id\_Profesor

FROM studenti\_reusita

GROUP BY Id\_Profesor

HAVING COUNT(DISTINCT Id\_Disciplina) >= @numar\_discipline

) AS SubQ

ON p.Id\_Profesor = SubQ.Id\_Profesor;

END;

EXEC ProfesoriCuMaiMulteDiscipline @numar\_discipline = 2; -- Specifici numărul minim de discipline (de exemplu, 2)

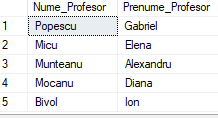


Figura 3 - Rezultatul codului executat

1. Să se elaboreze procedura stocată, care calculează numărul de examene la care s-a prezentat un student al cărui nume e dat ca parametru. Rezultatul execuției in figura 4

CREATE PROCEDURE CalculNumarExamenteStudent

@Nume\_Student NVARCHAR(100)

AS

BEGIN

-- Declara variabila pentru stocarea numarului de examene ale studentului

DECLARE @NumarExamente INT;

-- Calculeaza numarul de examene la care s-a prezentat studentul

SELECT @NumarExamente = COUNT(\*)

FROM studenti\_reusita sr

JOIN studenti s ON sr.Id\_Student = s.Id\_Student

WHERE s.Nume\_Student = @Nume\_Student;

-- Afișează rezultatul

PRINT 'Numărul de examene la care s-a prezentat studentul ' + @Nume\_Student + ' este: ' + CAST(@NumarExamente AS VARCHAR);

END;

EXEC CalculNumarExamenteStudent @Nume\_Student = 'NumeleStudentului';



Figura 4 - Rezultatul codului executat

1. Să se creeze o procedură stocată, care ar insera în baza de date informații despre un student nou. În calitate de parametrii de intrare să servească datele personale ale studentului nou și Cod\_Grupa. Sâ se genereze toate întrările-cheie necesare în tabelul studenți\_reusita. Notele de evaluare să fie inserate ca NULL. Rezultatul execuției in figura 5

DROP PROCEDURE IF EXISTS InserareStudentNou;

GO

-- Ștergem datele despre studentul inserat

IF EXISTS (

SELECT 1

FROM studenti Where Id\_Student = 175

)

BEGIN

DELETE FROM studenti\_reusita WHERE Id\_Student = 175;

DELETE FROM studenti WHERE Id\_Student = 175;

END

GO

CREATE PROCEDURE InserareStudentNou

@Id\_Student INT,

@Nume\_Student NVARCHAR(100),

@Prenume\_Student NVARCHAR(100),

@Data\_Nastere\_Student DATE,

@Adresa\_Postala\_Student NVARCHAR(255),

@Cod\_Grupa NVARCHAR(50),

@Id\_Profesor INT,

@Disciplina NVARCHAR(100)

AS

BEGIN

DECLARE @Id\_Grupa INT;

DECLARE @Id\_Disciplina INT;

BEGIN TRANSACTION;

-- Obținem ID-ul grupei

SELECT @Id\_Grupa = Id\_Grupa

FROM grupe

WHERE Cod\_Grupa = @Cod\_Grupa;

-- Obținem ID-ul disciplinei

SELECT @Id\_Disciplina = Id\_Disciplina

FROM discipline

WHERE Disciplina = @Disciplina;

-- Inserăm studentul

INSERT INTO studenti (Id\_Student, Nume\_Student, Prenume\_Student, Data\_Nastere\_Student, Adresa\_Postala\_Student)

VALUES (@Id\_Student, @Nume\_Student, @Prenume\_Student, @Data\_Nastere\_Student, @Adresa\_Postala\_Student);

-- Inserăm datele despre student în tabelul studenti\_reusita

INSERT INTO studenti\_reusita (Id\_Student, Id\_Disciplina, Id\_Profesor, Id\_Grupa, Tip\_Evaluare, Nota, Data\_Evaluare)

VALUES (@Id\_Student, @Id\_Disciplina, @Id\_Profesor, @Id\_Grupa, 'Testul 1', NULL, NULL),

(@Id\_Student, @Id\_Disciplina, @Id\_Profesor, @Id\_Grupa, 'Testul 2', NULL, NULL),

(@Id\_Student, @Id\_Disciplina, @Id\_Profesor, @Id\_Grupa, 'Examen', NULL, NULL);

COMMIT TRANSACTION;

END;

GO

EXEC InserareStudentNou 175, 'A', 'A', '2002-08-13', 'mun. Chisinau, str. Albisoara, 33/3', 'TI171', 101, 'Baze de date';

--Afisarea datelor despre studentul inserat

SELECT \*

FROM studenti\_reusita

WHERE Id\_Student = 175;



Figura 5 - Rezultatul codului executat

1. Să se creeze un declanșator care să verifice dacă au fost modificate coloanele 5 și 6 din tabelul studenti\_reusita prin instrucțiuni UPDATE.

-- Verificați dacă declanșatorul există și, dacă da, eliminați-l

-- Efectuați o actualizare care va declanșa trigger-ul

UPDATE studenti\_reusita

SET Nota = 9.5

WHERE Id\_Student = 12345;

DROP TRIGGER IF Exists VerificareModificareStudentiReusita;

Go

-- Creare declanșator

CREATE TRIGGER VerificareModificareStudentiReusita

ON studenti\_reusita

AFTER UPDATE

AS

BEGIN

-- Verificăm dacă coloanele 5 și 6 au fost modificate

IF UPDATE(Tip\_Evaluare) OR UPDATE(Nota)

BEGIN

-- Afișăm un mesaj de avertizare

PRINT 'Coloanele Tip\_Evaluare și/sau Nota au fost modificate!';

END

END;



Figura 6 - Rezultatul codului executat

1. Modificați declanșatorul tr\_inregistrare\_noua astfel încât, în cazul actualizării auditoriului, să afișeze un mesaj de informare care să includă codul grupei afectate, ziua, blocul, auditoriul vechi și auditoriul nou. Rezultatul execuției in figura 7

CREATE TABLE orar (

Id\_Disciplina INT,

Id\_Profesor INT,

Id\_Grupa SMALLINT,

Zi VARCHAR(50),

Ora TIME(0),

Auditoriu INT,

Bloc Char(1) NOT NULL DEFAULT ('B'),

CONSTRAINT ux\_prof UNIQUE (Zi, Ora, Id\_Grupa, Id\_Profesor),

CONSTRAINT ux\_disc UNIQUE (Zi, Ora, Id\_Grupa, Id\_Disciplina),

CONSTRAINT PK\_Id PRIMARY KEY (Id\_Grupa, Zi, Ora, Auditoriu),

CONSTRAINT FK\_disc FOREIGN KEY (Id\_Disciplina) REFERENCES discipline(Id\_Disciplina),

CONSTRAINT FK\_prof FOREIGN KEY (Id\_Profesor) REFERENCES profesori(Id\_Profesor)

);

DROP TRIGGER IF EXISTS tr\_inregistrare\_noua;

GO

CREATE TRIGGER tr\_inregistrare\_noua

ON orar

AFTER UPDATE

AS

BEGIN

SET NOCOUNT ON;

IF UPDATE(Auditoriu)

BEGIN

SELECT 'Lectia la disciplina "' + UPPER(d.Disciplina) + '" de la ora ' + CAST(inserted.Ora AS VARCHAR(5)) +

' a fost transferata in aula ' + CAST(inserted.Auditoriu AS CHAR(3)) +

', grupa ' + CAST(inserted.Id\_Grupa AS VARCHAR(10)) +

', ziua ' + inserted.Zi +

', blocul ' + inserted.Bloc +

', auditoriul vechi ' + CAST(deleted.Auditoriu AS CHAR(3)) +

', auditoriul nou ' + CAST(inserted.Auditoriu AS CHAR(3))

FROM inserted

JOIN discipline d ON inserted.Id\_Disciplina = d.Id\_Disciplina

JOIN deleted ON inserted.Id\_Grupa = deleted.Id\_Grupa AND inserted.Zi = deleted.Zi AND inserted.Ora = deleted.Ora

END

END;

GO

UPDATE orar

SET Auditoriu = 2

WHERE Id\_Disciplina = 101;

Figura 7 - Rezultatul codului executat

1. Să se creeze declanșatorul, care ar asigura popularea corectă (consecutivă) a tabelelor studenti și studenti\_reusita, și ar permite evitarea erorilor la nivelul cheilor externe.

DROP TRIGGER IF EXISTS Tr\_NewInsertStudenti

GO

CREATE TRIGGER Tr\_NewInsertStudenti

ON studenti

INSTEAD OF INSERT

AS

BEGIN

SET NOCOUNT ON;

-- Temporar dezactivăm cheile externe pentru a evita erorile de cheie străină în timpul inserării

ALTER TABLE studenti\_reusita NOCHECK CONSTRAINT FK\_studenti\_reusita\_Studenti;

-- Inserăm în tabelul studenti

INSERT INTO studenti (Id\_Student, Nume\_Student, Prenume\_Student, Data\_Nastere\_Student, Adresa\_Postala\_Student)

SELECT Id\_Student, Nume\_Student, Prenume\_Student, Data\_Nastere\_Student, Adresa\_Postala\_Student

FROM inserted;

-- Inserăm în tabelul studenti\_reusita

INSERT INTO studenti\_reusita (Id\_Student, Id\_Disciplina, Id\_Profesor, Id\_Grupa, Tip\_Evaluare, Nota, Data\_Evaluare)

SELECT i.Id\_Student, r.Id\_Disciplina, r.Id\_Profesor, r.Id\_Grupa, r.Tip\_Evaluare, r.Nota, r.Data\_Evaluare

FROM inserted i

JOIN studenti\_reusita r ON i.Id\_Student = r.Id\_Student;

-- Re-activăm cheile externe după inserare

ALTER TABLE studenti\_reusita WITH CHECK CONSTRAINT FK\_studenti\_reusita\_Studenti;

END;

Figura 8 - Rezultatul codului executat

1. Să se creeze un declanșator, care ar interzice micșorarea notelor în tabelul studenti\_reusita și modificarea valorilor câmpului Data\_Evaluare, unde valorile acestui câmp sunt nenule. Declanșatorul trebuie să se lanseze, numai dacă sunt afectate datele studenților din grupa ”CIB171”. Se va afișa un mesaj de avertizare în cazul tentativei de a încălca constrângerea.

UPDATE studenti\_reusita

SET Nota = 4

WHERE Id\_Student = 175;

DROP TRIGGER IF EXISTS Tr\_InterzicereModificareNote

GO

CREATE TRIGGER Tr\_InterzicereModificareNote

ON studenti\_reusita

AFTER UPDATE

AS

BEGIN

SET NOCOUNT ON;

-- Verificăm dacă actualizarea afectează studenții din grupa "CIB171"

IF EXISTS (SELECT 1 FROM inserted i INNER JOIN grupe g ON i.Id\_Grupa = g.Id\_Grupa WHERE g.Cod\_Grupa = 'CIB171')

BEGIN

-- Verificăm dacă nota a fost micșorată sau câmpul Data\_Evaluare a fost modificat pentru note nenule

IF EXISTS (SELECT 1 FROM inserted i JOIN deleted d ON i.Id\_Student = d.Id\_Student

WHERE (i.Nota < d.Nota AND d.Nota IS NOT NULL) OR (i.Data\_Evaluare <> d.Data\_Evaluare AND i.Data\_Evaluare IS NOT NULL))

BEGIN

-- Afișăm un mesaj de avertizare în cazul încălcării constrângerii

PRINT 'Modificarea notelor sau a valorilor de Data\_Evaluare pentru studenții din grupa "CIB171" este interzisă!';

-- Anulăm actualizarea

ROLLBACK TRANSACTION;

END;

END;

END;



Figura 9- Rezultatul codului executat

1. Să se creeze un declanșator DDL care ar interzice modificarea coloanei Id\_Disciplina în tabelele bazei de date universitatea cu afișarea mesajului respectiv.

DROP TRIGGER IF EXISTS tr\_Interzicere ON DATABASE;

GO

CREATE TRIGGER tr\_Interzicere

ON DATABASE

FOR ALTER\_TABLE, DROP\_TABLE

AS

BEGIN

PRINT 'Atentie, modificarea schemei bazei de date este interzisa, pentru a continua adresasati-va administratorului de system';

SELECT EVENTDATA().value('(/EVENT\_INSTANCE/TSQLCommand/CommandText)[1]', 'nvarchar(max)');

ROLLBACK;

END;

GO



Figura 10- Rezultatul codului executat

**Concluzie:**

In urma efectuării lucrării de laborator și examinării procesului de creare și utilizare a declanșatoarelor în bazele de date, se conturează o serie de observații și concluzii relevante.

Declanșatoarele reprezintă componente esențiale în gestionarea și controlul datelor într-o bază de date relațională. Ele oferă flexibilitate și automatizare în implementarea regulilor de afaceri, asigurând consistența și integritatea datelor.

Un aspect important al utilizării declanșatoarelor este capacitatea lor de a răspunde automat la evenimente specifice, cum ar fi operațiile de inserare, actualizare sau ștergere din tabelele bazei de date. Această reactivitate este esențială în menținerea coerenței datelor și aplicarea politicilor definite de întreprindere.

Pe lângă funcționalitatea lor de a executa acțiuni în mod automat, declanșatoarele pot fi folosite și pentru validarea datelor înainte de a fi introduse în baza de date, prevenind astfel introducerea informațiilor incorecte sau incomplete.

De asemenea, declanșatoarele pot fi utilizate pentru implementarea politicilor de securitate, restricționând accesul la date sau impunând reguli specifice de confidențialitate și integritate.

Cu toate acestea, utilizarea excesivă sau incorectă a declanșatoarelor poate duce la complexitate sporită în administrarea și întreținerea bazei de date, precum și la performanțe slabe în anumite scenarii.

Este important ca dezvoltatorii și administratorii de baze de date să fie conștienți de potențialele avantaje și limitări ale declanșatoarelor și să le utilizeze în mod rațional, luând în considerare cerințele specifice ale aplicației și mediul în care aceasta operează.

Prin urmare, în concluzie, declanșatoarele reprezintă un instrument puternic în arsenalul de administrare a bazei de date, însă utilizarea lor ar trebui să fie bine planificată și gestionată pentru a asigura eficiența și integritatea sistemului de gestionare a datelor.