**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: Transact-SQL. Instrucțiuni Procedurale**

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

**Chișinău, 2024**

**Obiective**:

**Sarcini practice:**

1. **Tipuri de blocuri in Transact-SQL**

In Transact-SQL, blocurile de instrucțiuni sunt folosite pentru a grupa și a executa mai multe instrucțiuni ca o unitate. Acestea sunt în general definite de cuvintele cheie BEGIN și END.

BEGIN

PRINT 'I am a block'

RETURN

PRINT 'I am still in a block but you will not see me'

END

[În acest exemplu, prima instrucțiune PRINT va fi executată, dar a doua instrucțiune PRINT nu va fi executată deoarece instrucțiunea RETURN întrerupe execuția blocului1](https://stackoverflow.com/questions/7263542/what-are-t-sql-statement-blocks).

[De asemenea, este important de menționat că instrucțiunea RETURN nu doar că iese din cel mai apropiat bloc BEGIN ... END (care pot fi îmbricate), dar va returna din întreaga funcție sau procedură1](https://stackoverflow.com/questions/7263542/what-are-t-sql-statement-blocks).

1. **Prezentarea variabilelor**

În SQL, variabilele locale sunt utilizate pentru a stoca date în timpul perioadei de execuție a lotului. Variabilele locale pot fi create pentru diferite tipuri de date și pot fi atribuite valori. [În plus, valorile atribuite variabilelor pot fi modificate în timpul perioadei de execuție1](https://www.sqlshack.com/sql-variables-basics-and-usage/).

Iată cum puteți declara o variabilă în SQL:

DECLARE @NumeVariabila data\_type [ = value ]

[În acest caz, @NumeVariabila este numele variabilei, data\_type este tipul de date al variabilei, iar value este o valoare opțională pe care o puteți atribui variabilei la declarare1](https://www.sqlshack.com/sql-variables-basics-and-usage/).

De exemplu, pentru a declara o variabilă numită @TestVariable de tip VARCHAR și a-i atribui valoarea inițială ‘Salvați Planeta Noastră’, ați folosi următoarea sintaxă:

DECLARE @TestVariable AS VARCHAR(100)='Salvați Planeta Noastră'

PRINT @TestVariable

[Puteți atribui o valoare unei variabile SQL folosind instrucțiunile SET sau SELECT1](https://www.sqlshack.com/sql-variables-basics-and-usage/). De exemplu:

DECLARE @TestVariable AS VARCHAR(100)

SET @TestVariable = 'O Planetă, O Viață'

PRINT @TestVariable

Sau folosind SELECT:

DECLARE @TestVariable AS VARCHAR(100)

SELECT @TestVariable = 'Salvați Natura'

PRINT @TestVariable

[De asemenea, puteți utiliza instrucțiunea SELECT pentru a atribui o valoare unei variabile dintr-un tabel, o vizualizare sau funcții cu valoare scalară1](https://www.sqlshack.com/sql-variables-basics-and-usage/).

1. **Sa se explice functionarea buclei repetitive WHILE**

[Buclele WHILE în SQL oferă posibilitatea de a executa o instrucțiune sau un set de instrucțiuni SQL în mod repetat, atâta timp cât o anumită condiție este îndeplinită1](https://www.sqlshack.com/sql-while-loop-with-simple-examples/).

Sintaxa pentru bucla WHILE în SQL este următoarea:

WHILE conditie

BEGIN

...instructiuni...

END

În fiecare iterație a buclei, condiția definită este verificată. Dacă rezultatul condiției este adevărat, instrucțiunile SQL vor fi executate. [Dacă rezultatul condiției este fals, fluxul de cod va ieși din buclă1](https://www.sqlshack.com/sql-while-loop-with-simple-examples/).

Iată un exemplu simplu de buclă WHILE în SQL:

DECLARE @Counter INT

SET @Counter=1

WHILE (@Counter <= 10)

BEGIN

PRINT 'Valoarea contorului este = ' + CONVERT(VARCHAR,@Counter)

SET @Counter = @Counter + 1

END

[În acest exemplu, bucla WHILE va afișa valoarea variabilei @Counter de zece ori, apoi bucla se va încheia1](https://www.sqlshack.com/sql-while-loop-with-simple-examples/).

Trebuie să fiți atenți la buclele infinite, unde condiția nu devine niciodată falsă, astfel încât bucla să continue la nesfârșit. [În astfel de cazuri, va trebui să opriți manual execuția interogării1](https://www.sqlshack.com/sql-while-loop-with-simple-examples/).

1. **Sa se explice functionarea expresiei CASE**

Expresia `CASE` în Transact-SQL (T-SQL) este o instrucțiune flexibilă folosită pentru a efectua evaluări condiționale și pentru a returna valori în funcție de aceste evaluări. Este similară cu un bloc `switch` din alte limbaje de programare.

Iată structura de bază a expresiei `CASE`:

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

...

ELSE default\_result

END

Expresia `CASE` începe cu cuvântul cheie `CASE` și se încheie cu `END`. Între acestea, specificăm o serie de clauze `WHEN` care conțin o condiție și rezultatul asociat cu acea condiție. Dacă nicio condiție `WHEN` nu este adevărată, se returnează valoarea din clauza `ELSE` (dacă este prezentă). Dacă niciuna dintre condițiile `WHEN` nu este adevărată și nu există clauză `ELSE`, expresia `CASE` returnează `NULL`.

Iată un exemplu simplu de utilizare a expresiei `CASE` în T-SQL:

SELECT

ProductID,

ProductName,

QuantityInStock,

CASE

WHEN QuantityInStock > 0 THEN 'In stock'

WHEN QuantityInStock = 0 THEN 'Out of stock'

ELSE 'Unknown'

END AS StockStatus

FROM Products;

În acest exemplu, `CASE` evaluează câteva condiții pe baza cantității în stoc (`QuantityInStock`) și returnează o etichetă corespunzătoare stocului.

Expresia `CASE` poate fi utilizată și în alte moduri, cum ar fi în clauzele `UPDATE`, `INSERT`, `WHERE` etc., permițând manipularea datelor în funcție de condiții complexe.

În concluzie, expresia `CASE` în T-SQL oferă o modalitate flexibilă și puternică de a gestiona evaluările condiționale și de a returna valori corespunzătoare în funcție de aceste evaluări.

1. **Utilizarea si actiunile structurii alternative IF...ELSE**

Structura alternativă `IF...ELSE` este o parte esențială a Transact-SQL (T-SQL) care permite executarea de acțiuni condiționate în funcție de o expresie logică. Acesta are o sintaxă similară cu alte limbaje de programare, cum ar fi C sau Java.

Iată structura de bază a instrucțiunii `IF...ELSE` în T-SQL:

IF condition

statement1

ELSE

statement2

- Dacă condiția este evaluată ca fiind adevărată, `statement1` este executat.

- Dacă condiția este evaluată ca fiind falsă, `statement2` este executat.

În plus față de acest format simplu, putem folosi instrucțiuni multiple `IF...ELSE IF...ELSE` pentru a gestiona mai multe condiții:

IF condition1

statement1

ELSE IF condition2

statement2

...

ELSE

statementN

Iată un exemplu simplu de utilizare a structurii `IF...ELSE` în T-SQL:

DECLARE @score INT = 85;

IF @score >= 90

PRINT 'Grade: A';

ELSE IF @score >= 80

PRINT 'Grade: B';

ELSE IF @score >= 70

PRINT 'Grade: C';

ELSE

PRINT 'Grade: D';

Aici, în funcție de valoarea variabilei `@score`, se afișează un mesaj corespunzător gradei corespunzătoare.

Principalele utilizări ale structurii `IF...ELSE` în T-SQL includ:

1. Controlul fluxului logic: Permite executarea de acțiuni diferite în funcție de condiții specifice.

2. Validarea datelor de intrare: Se poate utiliza pentru a valida datele introduse înainte de a continua cu o operațiune.

3. Manipularea datelor: Poate fi utilizată pentru a decide asupra modului în care se efectuează operații de manipulare a datelor, cum ar fi inserarea, actualizarea sau ștergerea acestora din baza de date.

În concluzie, instrucțiunea `IF...ELSE` este o unealtă puternică în Transact-SQL care permite controlul fluxului de execuție în funcție de condiții logice.

1. **Ce este o exceptie? Captarea si tratarea exceptiilor**

În Transact-SQL (T-SQL), o excepție este o situație neașteptată sau o eroare care apare în timpul executării unei instrucțiuni sau a unui bloc de cod. Aceste excepții pot fi rezultatul unor erori de sintaxă, erori de timp de execuție, erori legate de integritatea datelor sau orice altă condiție care împiedică execuția normală a unei instrucțiuni sau a unui bloc de cod.

Captarea și tratarea excepțiilor sunt procese utilizate pentru a gestiona aceste situații neașteptate și pentru a permite codului să facă față erorilor în mod elegant și controlat. Iată câteva elemente cheie legate de captarea și tratarea excepțiilor în T-SQL:

1**. Capturarea excepțiilor**: Acest proces implică identificarea și interceptarea excepțiilor în timpul execuției codului. În T-SQL, excepțiile sunt adesea capturate în blocuri `TRY...CATCH`. Blocul `TRY` conține codul care poate genera excepții, iar blocul `CATCH` este folosit pentru a trata și a gestiona excepțiile care apar în blocul `TRY`.

2**. Blocul TRY**: În acest bloc sunt plasate instrucțiunile care ar putea genera excepții.

3**. Blocul CATCH**: Acesta este utilizat pentru a trata excepțiile care sunt generate în blocul `TRY`. În acest bloc, puteți specifica acțiunile pe care doriți să le întreprindeți pentru a gestiona excepția. Aceste acțiuni pot include afișarea unui mesaj de eroare, înregistrarea excepției într-un jurnal, derularea la un alt bloc de cod sau reluarea execuției codului.

Exemplu de utilizare a blocurilor TRY...CATCH în T-SQL:

BEGIN TRY

-- Instrucțiuni care ar putea genera excepții

SELECT 1 / 0; -- O împărțire la zero va genera o excepție

END TRY

BEGIN CATCH

-- Tratarea excepțiilor

PRINT 'A apărut o excepție: ' + ERROR\_MESSAGE();

END CATCH;

În acest exemplu, împărțirea la zero în blocul `TRY` va genera o excepție. Această excepție este prinsă în blocul `CATCH`, iar un mesaj de eroare este afișat.

Capturarea și tratarea excepțiilor în T-SQL este esențială pentru a asigura fiabilitatea și robustețea codului, permițându-vă să gestionați erorile în mod controlat și să oferiți feedback corespunzător utilizatorilor sau sistemului în cazul unei probleme.

1. Functia instructiunii RAISEFOR

În Transact-SQL (T-SQL), instrucțiunea `RAISEERROR` este utilizată pentru a genera o eroare personalizată în cadrul unei proceduri stocate, a unui trigger sau a unui bloc de cod T-SQL. Ea poate fi folosită pentru a semnala o problemă în timpul execuției și poate fi capturată și tratată ulterior.

Iată sintaxa instrucțiunii `RAISEERROR`:

RAISEERROR (message\_id, severity, state);

- `message\_id`: Este un număr întreg care indică mesajul de eroare din catalogul de mesaje. Acesta poate fi un număr de eroare definit de utilizator sau un număr de eroare de sistem.

- `severity`: Este un număr întreg cuprins între 0 și 25, care indică gravitatea erorii. 0 este cea mai scăzută gravitate și 25 este cea mai mare. Gravitatea 11 sau mai mare oprește execuția curentă a blocului de cod.

- `state`: Este un număr întreg care oferă informații suplimentare despre contextul erorii.

Exemplu de utilizare a instrucțiunii `RAISEERROR`:

BEGIN TRY

-- Verificăm o condiție și generăm o eroare dacă nu este îndeplinită

IF (SELECT COUNT(\*) FROM Employees) < 10

RAISEERROR (50001, 16, 1); -- Mesaj de eroare personalizat

END TRY

BEGIN CATCH

-- Capturăm și tratăm eroarea

PRINT 'A apărut o eroare: ' + ERROR\_MESSAGE();

END CATCH;

În acest exemplu, dacă numărul de angajați din tabelul `Employees` este mai mic de 10, instrucțiunea `RAISEERROR` generează o eroare personalizată cu mesajul și informațiile despre gravitate specificate.

Instrucțiunea `RAISEERROR` este utilă în situațiile în care este necesar să semnalați o problemă sau să furnizați un mesaj informativ în timpul execuției unui bloc de cod T-SQL. Aceasta permite controlul și gestionarea adecvată a erorilor în aplicațiile bazate pe baze de date SQL Server.

1. Calculați valoarea totală a notelor tuturor studenților, puneți rezultatul într-o variabilă, afișați valoarea variabilei e ecran.

DECLARE @var INT

SET @var = 0;

-- Calculează suma notelor și stochează rezultatul în variabila definită

SELECT @var = SUM(Nota)

FROM studenti\_reusita;

--Afișează valoarea totală a notelor

SELECT @var AS Valoare\_Totala\_Note;



Figura 1 – Rezultatele codului executat

1. Calculați numărul de grupe Ia facultate, puneți rezultatul într-o variabilă, afișați valoarea variabilei pe ecran.

DECLARE @numar\_grupe INT;

SET @numar\_grupe = (SELECT COUNT(\*) FROM grupe);

-- Afișează numărul de grupe

SELECT @numar\_grupe AS Numar\_Grupe;

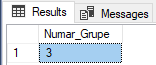


Figura 2 – Rezultatul execuției

1. Completați următorul cod pentru a afișa cel mai mare număr dintre cele trei numere prezentate:

DECLARE @N1 INT, @N2 INT, @N3 INT;

DECLARE @MAI\_MARE INT;

SET @N1 = 60 \* RAND();

SET @N2 = 60 \* RAND();

SET @N3 = 60 \* RAND();

-- Aici ar trebui plasate IF-urile ---

--------------------------------

PRINT @N1;

PRINT @N2;

PRINT @N3;

PRINT 'Mai mare = ' + CAST(@MAI\_MARE AS VARCHAR(2));

DECLARE @N1 INT, @N2 INT, @N3 INT;

DECLARE @MAI\_MARE INT;

SET @N1 = 60 \* RAND();

SET @N2 = 60 \* RAND();

SET @N3 = 60 \* RAND();

IF (@N1 >= @N2 AND @N1 >= @N3)

SET @MAI\_MARE = @N1;

ELSE IF (@N2 >= @N1 AND @N2 >= @N3)

SET @MAI\_MARE = @N2;

ELSE

SET @MAI\_MARE = @N3;

PRINT @N1;

PRINT @N2;

PRINT @N3;

PRINT 'Cel mai mare = ' + CAST(@MAI\_MARE AS VARCHAR(2));

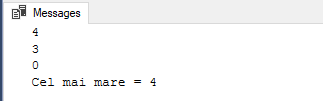


Figura 3 – Rezultatul execuției

1. Afișați primele zece date (numele, prenumele studentului) în funcție de valoarea notei (cu excepția notelor 6 și 8) a studentului Ia primul test al disciplinei Baze de date, folosind structura de alternativă IF...ELSE. Să se folosească variabilele.

DECLARE @Disciplina NVARCHAR(50) = 'Baze de date', @Test NVARCHAR(50) = 'Testul 1';

DECLARE @NotaMin INT = 6, @NotaMax INT = 8, @ExistaStud INT = 0;

CREATE TABLE #TopTenStudents (

Nume NVARCHAR(100),

Prenume NVARCHAR(100)

);

IF EXISTS (

SELECT 1

FROM studenti\_reusita sr

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

INNER JOIN discipline d ON sr.Id\_Disciplina = d.Id\_Disciplina

WHERE d.Disciplina = @Disciplina

AND sr.Tip\_Evaluare = @Test

AND sr.Nota NOT IN (@NotaMin, @NotaMax)

)

BEGIN

SET @ExistaStud = 1;

INSERT INTO #TopTenStudents (Nume, Prenume)

SELECT TOP 10

s.Nume\_Student,

s.Prenume\_Student

FROM

studenti s

INNER JOIN

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

INNER JOIN

discipline d ON sr.Id\_Disciplina = d.Id\_Disciplina

WHERE

d.Disciplina = @Disciplina

AND sr.Tip\_Evaluare = @Test

AND sr.Nota NOT IN (@NotaMin, @NotaMax)

ORDER BY

sr.Nota DESC;

END

IF @ExistaStud = 0

BEGIN

PRINT 'Nu există studenți care să îndeplinească condițiile.'

END

ELSE

BEGIN

SELECT \* FROM #TopTenStudents;

END

IF OBJECT\_ID('tempdb..#TopTenStudents') IS NOT NULL

BEGIN

DROP TABLE #TopTenStudents;

END

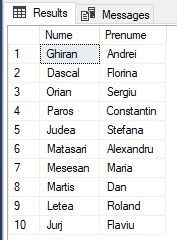


Figura 4 – Rezultatul execuției

1. Rezolvați aceeași sarcină, 4, apelând la structura selectivă CASE.

DECLARE @Disciplina NVARCHAR(50) = 'Baze de date', @Test NVARCHAR(50) = 'Testul 1';

DECLARE @NotaMin INT = 6, @NotaMax INT = 8;

CREATE TABLE #TopTenStudents (

Nume NVARCHAR(100),

Prenume NVARCHAR(100)

);

INSERT INTO #TopTenStudents (Nume, Prenume)

SELECT TOP 10

s.Nume\_Student,

s.Prenume\_Student

FROM

studenti s

INNER JOIN

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

INNER JOIN

discipline d ON sr.Id\_Disciplina = d.Id\_Disciplina

WHERE

d.Disciplina = @Disciplina

AND sr.Tip\_Evaluare = @Test

AND sr.Nota NOT IN (@NotaMin, @NotaMax)

ORDER BY

CASE

WHEN sr.Nota >= 5 THEN sr.Nota

ELSE 0

END DESC;

SELECT \* FROM #TopTenStudents;

IF @@ROWCOUNT = 0

BEGIN

PRINT 'Nu există studenți care să îndeplinească condițiile.'

END

DROP TABLE #TopTenStudents;



Figura 5 – Rezultatul execuției

1. Modificați exercițiile din sarcinile 4 și 5 pentru a include procesarea erorilor cu TRY și CATCH, și RAISERROR.

-- Sarcina Ex.4 - Modificata

DECLARE @Disciplina NVARCHAR(50) = 'Baze de date', @Test NVARCHAR(50) = 'Testul 1';

DECLARE @NotaMin INT = 6, @NotaMax INT = 8, @ExistaStud INT = 0;

BEGIN TRY

CREATE TABLE #TopTenStudents (

Nume NVARCHAR(100),

Prenume NVARCHAR(100)

);

IF EXISTS (

SELECT 1

FROM studenti\_reusita sr

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

INNER JOIN discipline d ON sr.Id\_Disciplina = d.Id\_Disciplina

WHERE d.Disciplina = @Disciplina

AND sr.Tip\_Evaluare = @Test

AND sr.Nota NOT IN (@NotaMin, @NotaMax)

)

BEGIN

SET @ExistaStud = 1;

INSERT INTO #TopTenStudents (Nume, Prenume)

SELECT TOP 10

s.Nume\_Student,

s.Prenume\_Student

FROM

studenti s

INNER JOIN

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

INNER JOIN

discipline d ON sr.Id\_Disciplina = d.Id\_Disciplina

WHERE

d.Disciplina = @Disciplina

AND sr.Tip\_Evaluare = @Test

AND sr.Nota NOT IN (@NotaMin, @NotaMax)

ORDER BY

sr.Nota DESC;

IF @ExistaStud = 1

BEGIN

SELECT \* FROM #TopTenStudents;

END

END

IF @ExistaStud = 0

BEGIN

THROW 50000, 'Nu există studenți care să îndeplinească condițiile.', 1;

END

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

RAISERROR(@ErrorMessage, 16, 1);

END CATCH

IF OBJECT\_ID('tempdb..#TopTenStudents') IS NOT NULL

DROP TABLE #TopTenStudents;

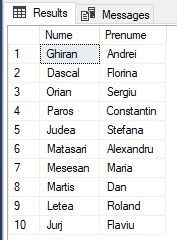


Figura 6 – Rezultatul execuției

-- Sarcina Ex.5 - Modificata

BEGIN TRY

DECLARE @Disciplina NVARCHAR(50) = 'Baze de date', @Test NVARCHAR(50) = 'Testul 1';

DECLARE @NotaMin INT = 6, @NotaMax INT = 8;

CREATE TABLE #TopTenStudents (

Nume NVARCHAR(100),

Prenume NVARCHAR(100)

);

INSERT INTO #TopTenStudents (Nume, Prenume)

SELECT TOP 10

s.Nume\_Student,

s.Prenume\_Student

FROM

studenti s

INNER JOIN

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

INNER JOIN

discipline d ON sr.Id\_Disciplina = d.Id\_Disciplina

WHERE

d.Disciplina = @Disciplina

AND sr.Tip\_Evaluare = @Test

AND sr.Nota NOT IN (@NotaMin, @NotaMax)

ORDER BY

CASE

WHEN sr.Nota >= 5 THEN sr.Nota

ELSE 0

END DESC;

SELECT \* FROM #TopTenStudents;

IF @@ROWCOUNT = 0

BEGIN

THROW 50000, 'Nu există studenți care să îndeplinească condițiile.', 1;

END

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

RAISERROR(@ErrorMessage, 16, 1);

END CATCH

DROP TABLE #TopTenStudents;

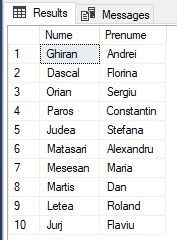


Figura 7 – Rezultatul execuției

**Concluzie:**

În concluzie, pentru o analiză aprofundată a instrucțiunilor procedurale în Transact-SQL, trebuie să ne concentrăm pe mai multe aspecte. Limbajul Transact-SQL (T-SQL) oferă un set bogat de funcționalități pentru gestionarea datelor și logică procedurală în cadrul serverului de bază de date SQL Server.

Principalele puncte de învățat includ:

**1. Instrucțiunile procedurale.** În T-SQL, instrucțiunile procedurale, cum ar fi `IF...ELSE`, `WHILE`, `BEGIN...END`, `TRY...CATCH` și altele, oferă controlul fluxului logic și gestionarea excepțiilor în cadrul blocurilor de cod.

**2. Utilizarea instrucțiunilor în mod adecvat.** Este esențial să înțelegem cum să folosim fiecare instrucțiune procedurală în mod corespunzător pentru a atinge obiectivele noastre. De exemplu, instrucțiunea `IF...ELSE` este ideală pentru decizii simple, în timp ce blocul `TRY...CATCH` este esențial pentru gestionarea erorilor.

**3. Tratarea excepțiilor.** În T-SQL, tratarea excepțiilor cu blocul `TRY...CATCH` este crucială pentru a gestiona erorile în mod controlat și pentru a menține integritatea și fiabilitatea aplicațiilor noastre. Prin capturarea și tratarea adecvată a excepțiilor, putem oferi feedback utilizatorilor sau sistemului și putem preveni întreruperile neașteptate.

**4. Generarea de erori personalizate.** Instrucțiunea `RAISEERROR` ne permite să generăm erori personalizate, oferind informații suplimentare și context relevant în cazul unor situații neașteptate.

În învățarea limbajului T-SQL, este important să practicăm utilizarea acestor instrucțiuni procedurale în scenarii reale și să înțelegem cum să le integrăm în dezvoltarea bazată pe baze de date SQL Server. Prin utilizarea corectă a acestor instrumente, putem îmbunătăți eficiența și fiabilitatea codului nostru și putem gestiona mai eficient operațiunile de gestionare a datelor și logică procedurală.