***Baza de date pentru gestionarea***

***serverului Dummy Chef***

**- proiect –**

**Proiect realizat de:** sd. cap**.** Văduva Marius Octavian

**Grupa de studii:** C 112B

1. Motivația alegerii proiectului

Am ales această temă de proiect întrucât consider că există o nevoie în creștere pentru soluții digitale care să faciliteze accesul la rețete culinare diverse și să simplifice procesul de gătit, atât pentru bucătarii amatori (utilizatorii), cât și pentru profesioniști. Dummy Chef este o platformă centralizată care își propune să ofere utilizatorilor acces la o bază de date extinsă de rețete culinare, să le permită să navigheze și să aleagă rețete în funcție de preferințele și necesitățile lor, să genereze liste de cumpărături personalizate și să faciliteze gestionarea ingredientelor și a furnizorilor.

Funcționalitățile aplicației pot acoperi mai multe necesități din societate:

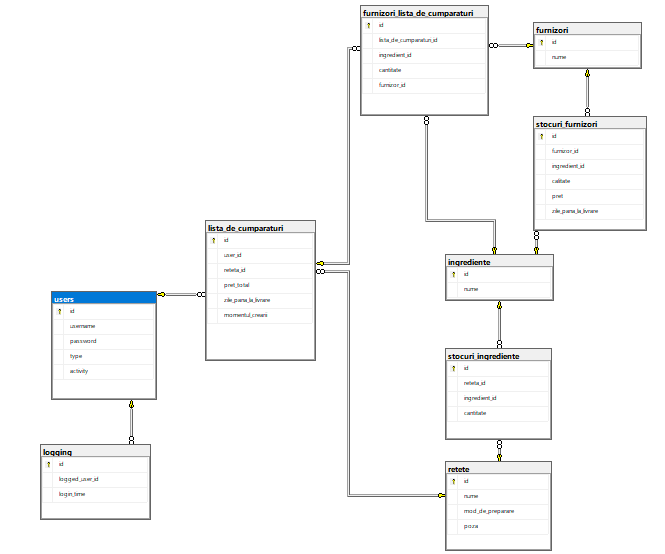
1. **Eficiență în gătit:** O platformă care oferă rețete variate și personalizate poate ajuta oamenii să gătească mai eficient și să încerce noi preparate culinare, înlocuind rutina culinară cu opțiuni mai diverse și creative.
2. **Economie de timp și resurse:** Generarea listelor de cumpărături pe baza preferințelor și bugetului utilizatorilor poate reduce timpul petrecut în magazine și poate ajuta la gestionarea mai eficientă a resurselor.
3. **Accesibilitate:** Prin separarea rolurilor între bucătari și clienți, platforma poate oferi atât experiențe personalizate pentru bucătarii profesioniști, cât și pentru cei amatori, facilitând accesul la informații și resurse specifice fiecărui tip de utilizator.
4. **Transparență în achiziții:** Integrarea informațiilor despre furnizori și stocuri în platformă poate oferi utilizatorilor transparență în privința surselor de alimente și poate ajuta la luarea deciziilor informate în procesul de cumpărare.
5. **Gestionarea eficientă a inventarului:** Funcționalitățile de administrare a ingredientelor și furnizorilor pot fi utile și pentru profesioniștii din industria alimentară, facilitând gestionarea stocurilor și aprovizionarea.

Astfel, baza de date pentru gestionarea serverului Dummy Chef are ca scop gestionarea resurselor acestei aplicații și furnizarea constanta de date catre aplicatie. Industria alimentară este în continuă creștere și evoluție, iar gestionarea eficientă a rețetelor, ingredientelor și furnizorilor este crucială pentru succesul în acest domeniu. O bază de date dedicată poate fi o unealtă valoroasă pentru profesioniștii din industria alimentară.

Potențialul de extindere și personalizare: Tema proiectului oferă oportunitatea de a explora și implementa funcționalități suplimentare sau personalizate în funcție de cerințele specifice ale utilizatorilor și bucătarilor. Această flexibilitate poate fi motivantă pentru a crea o soluție adaptată nevoilor și preferințelor utilizatorilor.

În scop didactic, în cadrul materiei „Programare orientată pe obiecte – proiect” va fi implementată o aplicație care sa utilizeze această bază de date și să permită automatizarea aceteia.

1. Prezentarea și creerea tabelelor



-- Crează tabelul "users"

CREATE TABLE users (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

username VARCHAR(50) UNIQUE, -- Câmpul pentru username, unic

password VARCHAR(100), -- Câmpul pentru parolă

type VARCHAR(20), -- Câmpul pentru tip (bucătar, utilizator, admin)

activity BIT -- Câmpul pentru activitate (1 pentru activ, 0 pentru inactiv)

);

Acest script va crea tabela **users** în baza de date **DummyChef**, cu următoarele caracteristici:

* Câmpul **username** va fi unic și va fi cheie primară. Este de tip **VARCHAR(50)**, ceea ce înseamnă că poate stoca până la 50 de caractere pentru fiecare username.
* Câmpul **password** va fi de tip **VARCHAR(100)**, ceea ce înseamnă că poate stoca până la 100 de caractere pentru fiecare parolă.
* Câmpul **type** va fi de tip **VARCHAR(20)** și va stoca tipul utilizatorului (bucătar, utilizator, admin).
* Câmpul **activity** va fi de tip **BIT**, care este un tip de date boolean în SQL Server (1 pentru activ, 0 pentru inactiv).

CREATE TABLE logging (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

logged\_user\_id INT, -- ID-ul utilizatorului care s-a logat

login\_time DATETIME, -- Momentul de timp la care s-a realizat logarea

FOREIGN KEY (logged\_user\_id) REFERENCES users(id) -- Cheie străină către tabela users

);

CREATE TABLE ingrediente (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

nume VARCHAR(100) UNIQUE -- Câmpul pentru nume, unic

);

Acest script va crea tabela ingrediente în baza de date DummyChef, cu următoarele caracteristici:

Câmpul id va fi cheie primară și va fi auto-incrementat. Acesta va servi ca identificator unic pentru fiecare înregistrare din tabel.

Câmpul nume va stoca numele ingredientului și va fi unic (UNIQUE). Este de tip VARCHAR(100), ceea ce înseamnă că poate stoca până la 100 de caractere pentru fiecare nume de ingredient.

CREATE TABLE furnizori (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

nume VARCHAR(100) UNIQUE -- Câmpul pentru nume, unic

);

CREATE TABLE stocuri\_furnizori (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

furnizor\_id INT, -- Cheie străină către tabelul furnizori

ingredient\_id INT, -- Cheie străină către tabelul ingrediente

cantitate DECIMAL(10,2), -- Cantitatea de ingredient în stoc

calitate VARCHAR(50), -- Calitatea ingredientului

pret DECIMAL(10,2), -- Prețul ingredientului

zile\_pana\_la\_livrare INT, -- Numărul de zile până la livrare

FOREIGN KEY (furnizor\_id) REFERENCES furnizori(id), -- Cheie străină către tabelul furnizori

FOREIGN KEY (ingredient\_id) REFERENCES ingrediente(id) -- Cheie străină către tabelul ingrediente

);

* **id**: Cheia primară a tabelului **stocuri\_furnizori**, care este auto-incrementată și servește drept identificator unic pentru fiecare înregistrare din tabel.
* **furnizor\_id**: Cheie străină către tabela **furnizori**, care indică furnizorul asociat cu acest stoc.
* **ingredient\_id**: Cheie străină către tabela **ingrediente**, care indică ingredientul asociat cu acest stoc.
* **cantitate**: Cantitatea de ingredient în stoc, definită ca un număr cu două zecimale.
* **calitate**: Descrierea calității ingredientului, de exemplu, "proaspăt", "congelat", etc.
* **pret**: Prețul ingredientului, definit ca un număr cu două zecimale.
* **zile\_pana\_la\_livrare**: Numărul de zile până la livrare pentru acest stoc de ingredient.

Această structură de tabel respectă relațiile dintre **stocuri\_furnizori**, **furnizori** și **ingrediente**, folosind chei străine către tabelele corecte.

ALTER TABLE stocuri\_furnizori

DROP COLUMN cantitate; -- eliminam cantitate din tabelul stocuri\_furnizori

CREATE TABLE retete (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

nume VARCHAR(100) UNIQUE, -- Numele retetei, unic

mod\_de\_preparare TEXT, -- Modul de preparare al retetei (text lung)

poza VARBINARY(MAX) -- Imaginea retetei (binar)

);

* **id**: Cheia primară a tabelului "retete", care este auto-incrementată și servește drept identificator unic pentru fiecare înregistrare din tabel.
* **nume**: Numele retetei, definit ca un șir de caractere de maxim 100 de caractere și unic, pentru a asigura că fiecare rețetă are un nume distinct.
* **mod\_de\_preparare**: Descrierea modului de preparare al retetei, definită ca un câmp de tip TEXT, care permite stocarea unor cantități mai mari de text.
* **poza**: Imaginea retetei, definită ca un câmp de tip VARBINARY(MAX), care poate stoca imagini de dimensiuni variabile.

CREATE TABLE stocuri\_ingrediente (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

reteta\_id INT, -- Cheie străină către retete

ingredient\_id INT, -- Cheie străină către ingrediente

cantitate DECIMAL(10,2), -- Cantitatea de ingredient necesară

FOREIGN KEY (reteta\_id) REFERENCES retete(id), -- Cheie străină către tabela retete

FOREIGN KEY (ingredient\_id) REFERENCES ingrediente(id) -- Cheie străină către tabela ingrediente

);

* **reteta\_id**: Cheie străină către tabela "retete", care indică reteta asociată cu aceste stocuri de ingrediente.
* **ingredient\_id**: Cheie străină către tabela "ingrediente", care indică ingredientul asociat cu aceste stocuri de ingrediente.
* **cantitate**: Cantitatea de ingredient necesară pentru reteta respectivă, definită ca un număr cu două zecimale.

Acest tabel "stocuri\_ingrediente" va permite să gestionezi relația între retete și ingredientele necesare pentru a le pregăti, păstrând înregistrări ale cantităților de ingrediente necesare pentru fiecare retetă.

CREATE TABLE lista\_de\_cumparaturi (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

user\_id INT, -- Cheie străină către tabelul users

reteta\_id INT, -- Cheie străină către tabelul retete

pret\_total DECIMAL(10,2), -- Prețul total al ingredientelor pentru reteta respectivă

zile\_pana\_la\_livrare INT, -- Numărul de zile până la livrare

momentul\_crearii DATETIME, -- Momentul de timp la care s-a realizat logarea

FOREIGN KEY (user\_id) REFERENCES users(id), -- Cheie străină către tabela users

FOREIGN KEY (reteta\_id) REFERENCES retete(id) -- Cheie străină către tabela retete

);

* **id**: Cheie primară auto-incrementată pentru identificarea unică a fiecărei înregistrări.
* **user\_id**: Cheie străină care face referire la utilizatorul care a creat lista de cumpărături.
* **reteta\_id**: Cheie străină care face referire la reteta pentru care este creată lista de cumpărături.
* **pret\_total**: Prețul total al ingredientelor necesare pentru reteta respectivă.
* **zile\_pana\_la\_livrare**: Numărul de zile până la livrarea ingredientelor.

Acest tabel va fi util pentru a urmări ce utilizatori au creat liste de cumpărături pentru anumite rețete, cât costă ingredientele pentru fiecare listă și câte zile mai sunt până la livrarea acestora.

CREATE TABLE furnizori\_lista\_de\_cumparaturi (

id INT PRIMARY KEY IDENTITY, -- Cheie primară auto-incrementată

lista\_de\_cumparaturi\_id INT, -- Cheie străină către tabela lista\_de\_cumparaturi

ingredient\_id INT, -- Cheie străină către tabela ingrediente

cantitate DECIMAL(10,2), -- Cantitatea de ingredient necesară

furnizor\_id INT, -- Cheie străină către tabela furnizori

FOREIGN KEY (lista\_de\_cumparaturi\_id) REFERENCES lista\_de\_cumparaturi(id), -- Cheie străină către tabela lista\_de\_cumparaturi

FOREIGN KEY (ingredient\_id) REFERENCES ingrediente(id), -- Cheie străină către tabela ingrediente

FOREIGN KEY (furnizor\_id) REFERENCES furnizori(id) -- Cheie străină către tabela furnizori

);

* **id**: Cheie primară auto-incrementată pentru identificarea unică a fiecărei înregistrări.
* **lista\_de\_cumparaturi\_id**: Cheie străină care face referire la lista de cumpărături pentru care se face achiziția.
* **ingredient\_id**: Cheie străină care face referire la ingredientul achiziționat de la furnizor.
* **furnizor\_id**: Cheie străină care face referire la furnizorul de la care se achiziționează ingredientul.

Acest tabel va ajuta la urmărirea legăturii dintre furnizori și listele de cumpărături, precum și legătura dintre ingredientele achiziționate și furnizori.

1. Insert

-- Inserare în tabelul "users"

INSERT INTO users (username, password, type, activity)

VALUES ('utilizator1', 'parola123', 'utilizator', 1),

('bucatar1', 'parola456', 'bucatar', 1),

('admin1', 'parola789', 'admin', 1);

-- Inserare în tabelul "logging"

INSERT INTO logging (logged\_user\_id, login\_time)

VALUES (1, GETDATE()), -- Presupunând că utilizatorul cu ID-ul 1 s-a logat în momentul curent

(2, '2024-03-19 10:30:00'), -- Un alt exemplu cu un timp specific

(3, '2024-03-19 11:45:00');

-- Inserare în tabelul "ingrediente"

INSERT INTO ingrediente (nume)

VALUES ('faina'),

('oua'),

('zahar');

-- Inserare în tabelul "furnizori"

INSERT INTO furnizori (nume)

VALUES ('Furnizor1'),

('Furnizor2'),

('Furnizor3');

-- Inserare în tabelul "stocuri\_furnizori"

INSERT INTO stocuri\_furnizori (furnizor\_id, ingredient\_id, calitate, pret, zile\_pana\_la\_livrare)

VALUES (1, 1, 'buna', 5.99, 2),

(2, 2, 'foarte buna', 7.50, 3),

(3, 3, 'excelenta', 10.25, 1);

-- Inserare în tabelul "retete"

INSERT INTO retete (nume, mod\_de\_preparare, poza)

VALUES ('Placinta cu mere', 'Se amesteca faina cu ouale si zaharul, apoi se adauga merele taiate cubulete. Se coace la 180°C timp de 40 de minute.', NULL),

('Omleta', 'Se bat ouale, se adauga sare si piper, apoi se prajesc intr-o tigaie incinsa.', NULL),

('Tiramisu', 'Se amesteca mascarpone cu zaharul si galbenusurile, apoi se aranjeaza straturi de piscoturi insiropate cu cafea si crema de mascarpone.', NULL);

-- Inserare în tabelul "stocuri\_ingrediente"

INSERT INTO stocuri\_ingrediente (reteta\_id, ingredient\_id, cantitate)

VALUES (1, 1, 300), -- Placinta cu mere: 300 grame de faina

(1, 2, 4), -- Placinta cu mere: 4 oua

(1, 3, 150), -- Placinta cu mere: 150 grame de zahar

(2, 2, 6), -- Omleta: 6 oua

(3, 2, 12), -- Tiramisu: 12 oua

(3, 1, 200), -- Tiramisu: 200 grame de faina

(3, 3, 250); -- Tiramisu: 250 grame de zahar

-- Inserare în tabelul "lista\_de\_cumparaturi"

INSERT INTO lista\_de\_cumparaturi (user\_id, reteta\_id, pret\_total, zile\_pana\_la\_livrare,momentul\_crearii)

VALUES (1, 2, NULL, NULL,GETDATE()), -- Utilizatorul 1 are în lista de cumpărături reteta 1, prețul total și zilele până la livrare vor fi actualizate automat

(2, 2, NULL, NULL,'2024-03-19 10:30:00'), -- Utilizatorul 2 are în lista de cumpărături reteta 2, prețul total și zilele până la livrare vor fi actualizate automat

(3, 3, NULL, NULL,'2024-03-19 10:35:00'); -- Utilizatorul 3 are în lista de cumpărături reteta 3, prețul total și zilele până la livrare vor fi actualizate automat

-- Inserare în tabelul "furnizori\_lista\_de\_cumparaturi"

-- Presupunând că fiecare retetă necesită un ingredient de la un furnizor

INSERT INTO furnizori\_lista\_de\_cumparaturi (lista\_de\_cumparaturi\_id, ingredient\_id, cantitate, furnizor\_id)

VALUES (1, 1, 500, 1), -- Lista de cumpărături 1 necesită 500 grame de făină de la furnizorul 1

(2, 2, 8, 2), -- Lista de cumpărături 2 necesită 8 ouă de la furnizorul 2

(3, 3, 300, 3); -- Lista de cumpărături 3 necesită 300 grame de zahăr de la furnizorul 3

!!!!!!!!

UPDATE lista\_de\_cumparaturi

SET pret\_total = (

SELECT SUM(ROUND(sf.pret \* flc.cantitate, 0))

FROM furnizori\_lista\_de\_cumparaturi flc

INNER JOIN stocuri\_furnizori sf ON flc.furnizor\_id = sf.furnizor\_id AND flc.ingredient\_id = sf.ingredient\_id

WHERE flc.lista\_de\_cumparaturi\_id = lista\_de\_cumparaturi.id

)

WHERE id IN (

SELECT lista\_de\_cumparaturi\_id

FROM furnizori\_lista\_de\_cumparaturi

WHERE reteta\_id = lista\_de\_cumparaturi.reteta\_id

);

Această declarație **UPDATE** va actualiza toate înregistrările din **lista\_de\_cumparaturi**, calculând prețul total pentru fiecare listă de cumpărături. Subinterogarea calculează suma prețurilor din **stocuri\_furnizori** pentru fiecare listă de cumpărături în funcție de rețeta asociată, folosind furnizorii și ingredientele din **furnizori\_lista\_de\_cumparaturi**. Apoi, aceste valori calculate sunt actualizate în câmpul **pret\_total** al fiecărei liste de cumpărături. Suma este calculată înmulțind prețul ingredientului cu cantitatea necesară și rotunjind rezultatul la integer

UPDATE lista\_de\_cumparaturi

SET zile\_pana\_la\_livrare = (

SELECT MAX(sf.zile\_pana\_la\_livrare)

FROM furnizori\_lista\_de\_cumparaturi flc

INNER JOIN stocuri\_furnizori sf ON flc.furnizor\_id = sf.furnizor\_id AND flc.ingredient\_id = sf.ingredient\_id

WHERE flc.lista\_de\_cumparaturi\_id = lista\_de\_cumparaturi.id

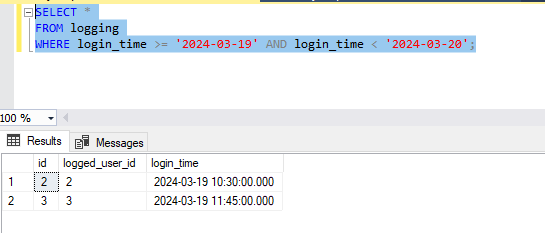
)

Această declarație **UPDATE** va actualiza câmpul **zile\_pana\_la\_livrare** în **lista\_de\_cumparaturi** cu timpul maxim de livrare găsit în **stocuri\_furnizori** pentru fiecare listă de cumpărături. Subinterogarea caută furnizorii și ingredientele corespunzătoare fiecărei liste de cumpărături din **furnizori\_lista\_de\_cumparaturi**, apoi găsește timpul maxim de livrare pentru acești furnizori din **stocuri\_furnizori**. Acest timp maxim este apoi actualizat în câmpul **zile\_pana\_la\_livrare** al fiecărei liste de cumpărături.

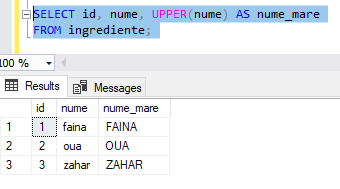
!!!!!!!!

1. Select

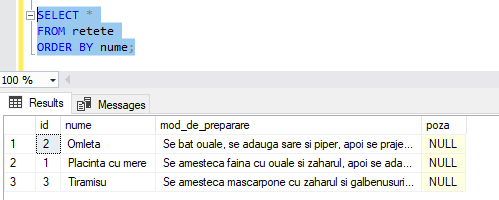
* **Select cu filtrare de date calendaristice:**



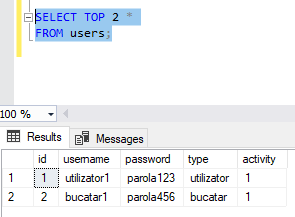
* **Select cu prelucrare de date de tip caracter:**



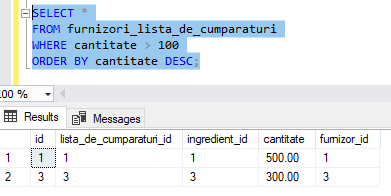
* **Select folosind clauza Order By:**



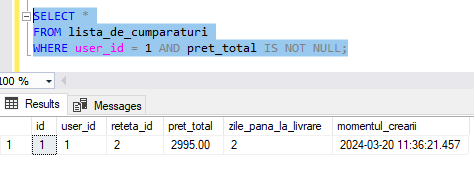
* **Select cu filtrarea datelor folosind TOP:**



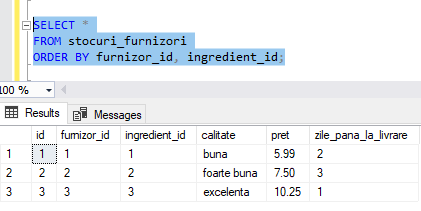
* **Select cu filtrarea și ordonarea datelor:**



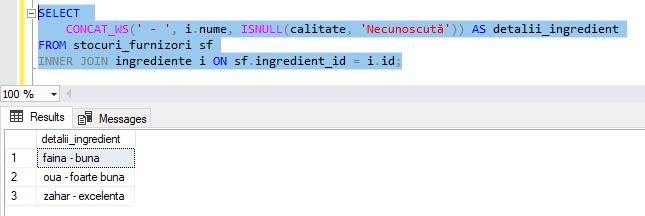
* **Select cu filtrare de date în care se combină cel puțin 2 predicate:**



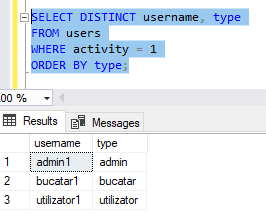
* **Select cu ordonare pe mai multe coloane:**



* **Select cu concatenare de coloane în care este permis NULL, cu obținerea unui rezultat relevant:**



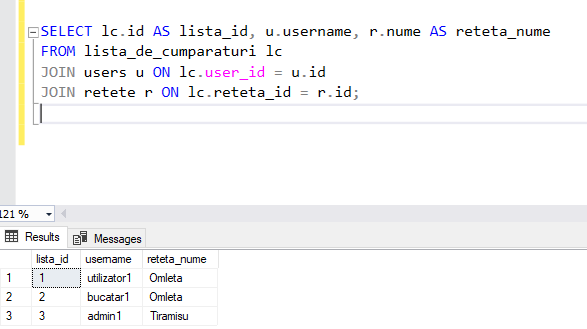
* **Select cu eliminarea duplicatelor rămase după aplicarea a două filtre. Afișați datele în mod convenabil:**



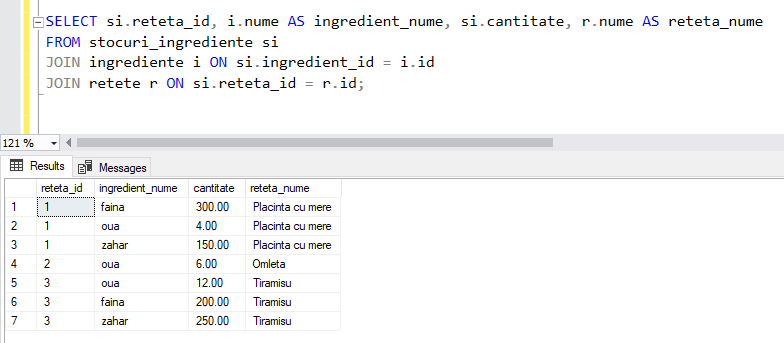
Partea II

**Jonctiuni pe cel putin 3 tabele**

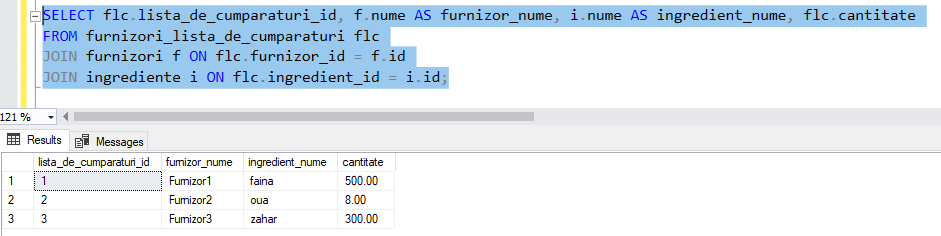
1. Joncțiune între tabela **lista\_de\_cumparaturi**, **retete** și **users** pentru a obține detalii despre lista de cumpărături împreună cu numele utilizatorului și numele retetei.



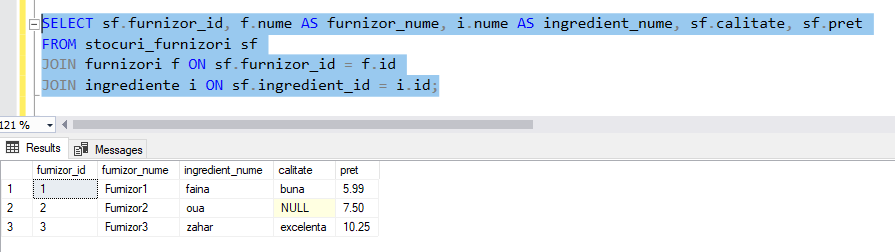
1. Joncțiune între tabela **stocuri\_ingrediente**, **ingrediente** și **retete** pentru a obține detalii despre ingredientele necesare pentru fiecare retetă.



1. Joncțiune între tabela **furnizori\_lista\_de\_cumparaturi**, **furnizori** și **ingrediente** pentru a obține detalii despre furnizorii pentru fiecare ingredient din lista de cumpărături.

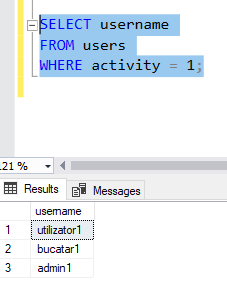


1. Joncțiune între tabela **stocuri\_furnizori**, **furnizori** și **ingrediente** pentru a obține detalii despre stocurile furnizorilor pentru fiecare ingredient.

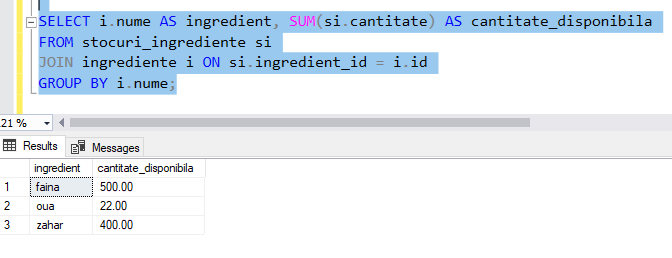


**Selecturi cu operatori pe seturi de date**

1. **Utilizatori activi:** Această interogare va returna utilizatorii care sunt inactivi (cu activitatea setată la 1) din tabela **users**.



1. **Ingredientele disponibile în stoc pentru toate retetele:** Această interogare va returna numele ingredientelor și cantitățile disponibile în stoc pentru toate retetele din tabela **stocuri\_ingrediente**.



1. **Retetele care necesită ingrediente disponibile în stoc:** Această interogare va returna numele retetelor și ingredientele necesare pentru retetele care au toate ingredientele disponibile în stoc.

SELECT r.nume AS reteta, SUM(sf.pret \* si.cantitate) AS pret\_total

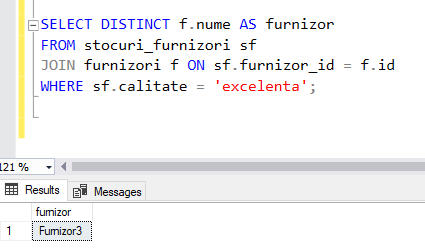
FROM retete r

JOIN stocuri\_ingrediente si ON r.id = si.reteta\_id

JOIN stocuri\_furnizori sf ON si.ingredient\_id = sf.ingredient\_id

GROUP BY r.id, r.nume;

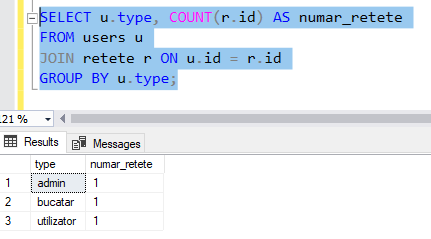
1. **Furnizori care oferă cel puțin un ingredient de calitate excelentă:** Această interogare va returna numele furnizorilor care oferă cel puțin un ingredient de calitate excelentă din tabela **stocuri\_furnizori**.



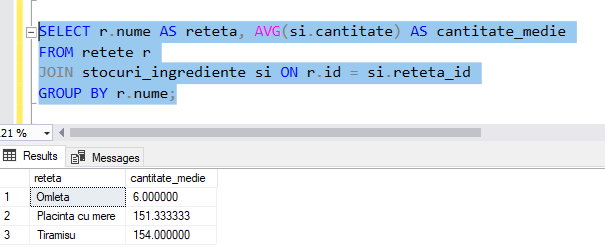
Aceste interogări utilizează operatorii pe seturi de date precum **WHERE**, **GROUP BY**, **HAVING**, **DISTINCT** pentru a extrage informații specifice din baza de date în funcție de criteriile specificate.

**Selecturi cu grupări de date**

1. **Numărul total de retete pentru fiecare tip de utilizator:** Această interogare va returna numărul total de retete create de fiecare tip de utilizator din tabela **users** și **retete**.



1. **Media cantității necesare de ingrediente pentru fiecare retetă:** Această interogare va returna media cantității necesare de ingrediente pentru fiecare retetă din tabela **stocuri\_ingrediente**.



1. **Prețul total al ingredientelor necesare pentru fiecare retetă:** Această interogare va returna prețul total al ingredientelor necesare pentru fiecare retetă din tabela **stocuri\_furnizori**.

SELECT r.nume AS reteta, SUM(sf.pret \* si.cantitate) AS pret\_total

FROM retete r

JOIN stocuri\_ingrediente si ON r.id = si.reteta\_id

JOIN stocuri\_furnizori sf ON si.ingredient\_id = sf.ingredient\_id

GROUP BY r.nume;

1. Cantitatea totală de ingrediente disponibile în stoc pentru fiecare furnizor: Această interogare va returna cantitatea totală de ingrediente disponibile în stoc pentru fiecare furnizor din tabela stocuri\_furnizori.

SELECT f.nume AS furnizor, SUM(sf.cantitate) AS cantitate\_totala

FROM furnizori f

JOIN stocuri\_furnizori sf ON f.id = sf.furnizor\_id

GROUP BY f.nume;

Aceste interogări utilizează clauza **GROUP BY** pentru a grupa datele în funcție de anumite criterii și pentru a aplica funcții de agregare cum ar fi **COUNT**, **SUM**, **AVG** pentru a prezenta informații structurate sau agregate din baza de date.

**Selecturi cu filtre pe grupuri**

1. Utilizatori activi care au creat retete: Această interogare va returna utilizatorii activi care au creat retete din tabela users și retete.

SELECT r.nume AS reteta, COUNT(si.ingredient\_id) AS numar\_ingrediente

FROM retete r

JOIN stocuri\_ingrediente si ON r.id = si.reteta\_id

GROUP BY r.nume

HAVING COUNT(si.ingredient\_id) > 5;

1. Retetele cu mai multe de 5 ingrediente: Această interogare va returna retetele care au mai mult de 5 ingrediente din tabela retete și stocuri\_ingrediente.

SELECT r.nume AS reteta, COUNT(si.ingredient\_id) AS numar\_ingrediente

FROM retete r

JOIN stocuri\_ingrediente si ON r.id = si.reteta\_id

GROUP BY r.nume

HAVING COUNT(si.ingredient\_id) > 5;

1. Retetele cu mai multe de 5 ingrediente: Această interogare va returna retetele care au mai mult de 5 ingrediente din tabela retete și stocuri\_ingrediente.

WITH UtilizatoriCuRetete AS (

SELECT u.username, r.nume AS reteta

FROM users u

JOIN retete r ON u.id = r.id

)

SELECT \* FROM UtilizatoriCuRetete;

1. Retetele cu mai multe de 5 ingrediente: Această interogare va returna retetele care au mai mult de 5 ingrediente din tabela retete și stocuri\_ingrediente.

WITH UtilizatoriCuRetete AS (

SELECT u.username, r.nume AS reteta

FROM users u

JOIN retete r ON u.id = r.id

)

SELECT \* FROM UtilizatoriCuRetete;

**Selecturi cu CTE**

1. **Utilizatori și retetele create de ei:** Această interogare utilizează o CTE pentru a defini temporar o listă de utilizatori și retetele create de ei din tabelele **users** și **retete**

WITH UtilizatoriCuRetete AS (

SELECT u.username, r.nume AS reteta

FROM users u

JOIN retete r ON u.id = r.id

)

SELECT \* FROM UtilizatoriCuRetete;

1. Ingredientele și preturile disponibile în stoc: Această interogare utilizează o CTE pentru a defini temporar o listă de ingrediente și preturile disponibile în stoc din tabelele ingrediente și stocuri\_furnizori.

WITH PretStoc AS (

SELECT i.nume AS ingredient, sf.pret

FROM ingrediente i

JOIN stocuri\_furnizori sf ON i.id = sf.ingredient\_id

)

SELECT \* FROM PretStoc;

1. Lista de cumpărături cu prețul total pentru fiecare utilizator: Această interogare utilizează o CTE pentru a defini temporar lista de cumpărături și prețul total pentru fiecare utilizator din tabelele lista\_de\_cumparaturi și users.

WITH PretTotalLista AS (

SELECT user\_id, SUM(pret\_total) AS pret\_total

FROM lista\_de\_cumparaturi

GROUP BY user\_id

)

SELECT u.username, pt.pret\_total

FROM users u

JOIN PretTotalLista pt ON u.id = pt.user\_id;

1. Numarare furnizori

WITH NumarFurnizori AS (

SELECT COUNT(\*) AS numar\_furnizori

FROM furnizori

)

SELECT \* FROM NumarFurnizori;

**View-uri**

1. Lista de cumpărături cu prețul total pentru fiecare utilizator: Această interogare utilizează o CTE pentru a defini temporar lista de cumpărături și prețul total pentru fiecare utilizator din tabelele lista\_de\_cumparaturi și users.

CREATE VIEW ListaCumparaturiUtilizatoriActivi AS

SELECT lc.id, u.username, r.nume AS reteta, lc.pret\_total

FROM lista\_de\_cumparaturi lc

JOIN users u ON lc.user\_id = u.id

JOIN retete r ON lc.reteta\_id = r.id

WHERE u.activity = 1;

1. View pentru stocurile disponibile în stocurile furnizorilor: Acest view va afișa stocurile disponibile în stocurile furnizorilor din tabela stocuri\_furnizori.

CREATE VIEW StocuriDisponibileFurnizori AS

SELECT sf.furnizor\_id, f.nume AS furnizor, i.nume AS ingredient, sf.cantitate

FROM stocuri\_furnizori sf

JOIN furnizori f ON sf.furnizor\_id = f.id

JOIN ingrediente i ON sf.ingredient\_id = i.id;

1. View pentru stocurile disponibile în stocurile furnizorilor: Acest view va afișa stocurile disponibile în stocurile furnizorilor din tabela stocuri\_furnizori.

CREATE VIEW ReteteCuOua AS

SELECT r.nume AS reteta, si.cantitate AS cantitate\_oua

FROM retete r

JOIN stocuri\_ingrediente si ON r.id = si.reteta\_id

JOIN ingrediente i ON si.ingredient\_id = i.id

WHERE i.nume = 'oua';

1. View pentru stocurile disponibile în stocurile furnizorilor: Acest view va afișa stocurile disponibile în stocurile furnizorilor din tabela stocuri\_furnizori.

CREATE VIEW FurnizoriIngredienteExcelente AS

SELECT DISTINCT f.nume AS furnizor, sf.calitate

FROM furnizori f

JOIN stocuri\_furnizori sf ON f.id = sf.furnizor\_id

WHERE sf.calitate = 'excelenta';

1. View pentru stocurile disponibile în stocurile furnizorilor: Acest view va afișa stocurile disponibile în stocurile furnizorilor din tabela stocuri\_furnizori.

CREATE VIEW ListaCumparaturiDetaliiIngrediente AS

SELECT lc.id, u.username, r.nume AS reteta, flc.cantitate, i.nume AS ingredient, f.nume AS furnizor

FROM lista\_de\_cumparaturi lc

JOIN users u ON lc.user\_id = u.id

JOIN retete r ON lc.reteta\_id = r.id

JOIN furnizori\_lista\_de\_cumparaturi flc ON lc.id = flc.lista\_de\_cumparaturi\_id

JOIN ingrediente i ON flc.ingredient\_id = i.id

JOIN furnizori f ON flc.furnizor\_id = f.id;

--TRIGGERE

--Atunci când un utilizator se autentifică, acest trigger înregistrează într-un jurnal momentul autentificării și identitatea utilizatorului, oferind o modalitate de monitorizare a accesului la sistem și de urmărire a activității utilizatorilor.

CREATE TRIGGER trg\_UserLoginLogging

ON users

AFTER INSERT

AS

BEGIN

INSERT INTO logging (logged\_user\_id, login\_time)

SELECT id, GETDATE()

FROM inserted;

END;

DROP TRIGGER trg\_UserLoginLogging

--Acest trigger actualizează automat prețul total al ingredientelor pentru o rețetă în lista de cumpărături în funcție de modificările aduse stocurilor de ingrediente, asigurând o actualizare precisă și eficientă a costurilor asociate pregătirii unei rețete.

CREATE TRIGGER trg\_UpdateTotalPrice

ON stocuri\_ingrediente

AFTER INSERT, UPDATE

AS

BEGIN

UPDATE l

SET l.pret\_total = (SELECT SUM(si.cantitate \* sf.pret)

FROM inserted i

INNER JOIN lista\_de\_cumparaturi l ON i.reteta\_id = l.reteta\_id

INNER JOIN stocuri\_ingrediente si ON l.reteta\_id = si.reteta\_id

INNER JOIN stocuri\_furnizori sf ON si.ingredient\_id = sf.ingredient\_id

WHERE l.id = i.reteta\_id)

FROM lista\_de\_cumparaturi l

INNER JOIN inserted i ON l.reteta\_id = i.reteta\_id;

END;

DROP TRIGGER trg\_UpdateTotalPrice

-- Prin acest trigger, orice modificare adusă stocurilor de ingrediente este înregistrată într-un jurnal, oferind o urmărire detaliată a schimbărilor și a actualizărilor efectuate asupra stocurilor, astfel încât să se poată gestiona eficient resursele disponibile.

CREATE TRIGGER trg\_MonitorStockChanges

ON stocuri\_furnizori

AFTER UPDATE

AS

BEGIN

INSERT INTO logging (logged\_user\_id, login\_time)

VALUES (1, GETDATE()); -- Schimbați id-ul utilizatorului și data după necesități

END;

DROP TRIGGER trg\_MonitorStockChanges

-- Acest trigger împiedică ștergerea accidentală a utilizatorilor activi, înlocuind operația de ștergere cu o acțiune care elimină doar utilizatorii inactivi din baza de date. Astfel, se asigură integritatea și continuitatea datelor pentru utilizatorii activi.

CREATE TRIGGER trg\_PreventActiveUserDeletion

ON users

INSTEAD OF DELETE

AS

BEGIN

DELETE FROM users

WHERE id IN (SELECT id FROM deleted WHERE activity = 0)

END;

DROP TRIGGER trg\_PreventActiveUserDeletion

--Atunci când se înregistrează o nouă intrare în jurnalul de activitate, acest trigger actualizează automat statutul de activitate al utilizatorului asociat, marcându-l ca activ. Această funcționalitate permite menținerea și actualizarea continuă a stării de activitate a utilizatorilor în sistem.

CREATE TRIGGER trg\_UpdateUserActivity

ON logging

AFTER INSERT

AS

BEGIN

UPDATE u

SET u.activity = 1

FROM users u

INNER JOIN inserted i ON u.id = i.logged\_user\_id;

END;

DROP TRIGGER trg\_UpdateUserActivity

-- PROCEDURI STOCATE

-- Procedura stocată pentru a obține toate retetele disponibile pentru un anumit utilizator:

CREATE PROCEDURE GetAvailableRecipesForUser

@username VARCHAR(50)

AS

BEGIN

SELECT r.\*

FROM retete r

INNER JOIN lista\_de\_cumparaturi lc ON r.id = lc.reteta\_id

INNER JOIN users u ON lc.user\_id = u.id

WHERE u.username = @username;

END;

-- Această procedură stocată primește un nume de utilizator ca parametru și returnează toate rețetele disponibile pentru acel utilizator.

DROP PROCEDURE GetAvailableRecipesForUser

--Procedura stocată pentru a adăuga un nou furnizor:

CREATE PROCEDURE AddNewSupplier

@supplier\_name VARCHAR(100)

AS

BEGIN

INSERT INTO furnizori (nume)

VALUES (@supplier\_name);

END;

-- Această procedură stocată primește numele unui nou furnizor și îl adaugă în tabela furnizori.

DROP PROCEDURE AddNewSupplier

-- Procedura stocată pentru a actualiza parola utilizatorului

CREATE PROCEDURE UpdateUserPassword

@username VARCHAR(50),

@new\_password VARCHAR(100)

AS

BEGIN

UPDATE users

SET password = @new\_password

WHERE username = @username;

END;

-- Această procedură stocată primește numele de utilizator și noua parolă și actualizează parola utilizatorului corespunzător.

DROP PROCEDURE UpdateUserPassword

-- Procedura stocată pentru a șterge un furnizor împreună cu toate legăturile sale:

CREATE PROCEDURE DeleteSupplierAndRelatedData

@supplier\_id INT

AS

BEGIN

DELETE FROM furnizori WHERE id = @supplier\_id;

DELETE FROM stocuri\_furnizori WHERE furnizor\_id = @supplier\_id;

DELETE FROM furnizori\_lista\_de\_cumparaturi WHERE furnizor\_id = @supplier\_id;

END;

--Această procedură stocată primește ID-ul unui furnizor și șterge furnizorul împreună cu toate înregistrările legate de el din tabelele relevante.

DROP PROCEDURE DeleteSupplierAndRelatedData

-- Procedura stocată pentru a adăuga un nou ingredient în stocul furnizorului:

CREATE PROCEDURE AddIngredientToSupplierStock

@supplier\_id INT,

@ingredient\_id INT,

@quality VARCHAR(50),

@price DECIMAL(10,2),

@delivery\_days INT

AS

BEGIN

INSERT INTO stocuri\_furnizori (furnizor\_id, ingredient\_id, calitate, pret, zile\_pana\_la\_livrare)

VALUES (@supplier\_id, @ingredient\_id, @quality, @price, @delivery\_days);

END;

DROP PROCEDURE AddIngredientToSupplierStock

-- Această procedură stocată primește ID-ul furnizorului, ID-ul ingredientului, calitatea, prețul și numărul de zile până la livrare și adaugă un nou ingredient în stocul furnizorului corespunzător.

-- tranzacții cu prelucrarea erorilor

-- Tranzacție pentru adăugarea unui nou utilizator și înregistrarea în jurnal (logging):

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO users (username, password, type, activity)

VALUES ('new\_user', 'password123', 'utilizator', 1);

INSERT INTO logging (logged\_user\_id, login\_time)

VALUES (SCOPE\_IDENTITY(), GETDATE());

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

-- Putem înregistra sau gestiona eroarea aici

SELECT ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

--Tranzacție pentru actualizarea prețului unui ingredient în stocul furnizorului:

BEGIN TRY

BEGIN TRANSACTION;

UPDATE stocuri\_furnizori

SET pret = 15.99

WHERE furnizor\_id = 1 AND ingredient\_id = 2;

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

-- Puteți înregistra sau gestiona eroarea aici

SELECT ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

--Tranzacție pentru adăugarea unui nou furnizor împreună cu legăturile sale:

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO furnizori (nume)

VALUES ('NumeFurnizor');

DECLARE @supplier\_id INT = SCOPE\_IDENTITY();

INSERT INTO stocuri\_furnizori (furnizor\_id, ingredient\_id, calitate, pret, zile\_pana\_la\_livrare)

VALUES (@supplier\_id, 1, 'Bună', 10.99, 3);

-- Se pot adăuga și alte legături aici

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

-- Puteți înregistra sau gestiona eroarea aici

SELECT ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

-- Tranzacție pentru ștergerea unui utilizator și a tuturor datelor asociate:

BEGIN TRY

BEGIN TRANSACTION;

DECLARE @deleted\_user\_id INT;

SELECT @deleted\_user\_id = id FROM users WHERE username = 'user\_to\_delete';

DELETE FROM lista\_de\_cumparaturi WHERE user\_id = @deleted\_user\_id;

DELETE FROM logging WHERE logged\_user\_id = @deleted\_user\_id;

DELETE FROM users WHERE id = @deleted\_user\_id;

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

-- Puteți înregistra sau gestiona eroarea aici

SELECT ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

-- Tranzacție pentru adăugarea unui nou ingredient în stocul furnizorului cu verificare de existență a furnizorului:

BEGIN TRY

BEGIN TRANSACTION;

DECLARE @supplier\_id INT;

SELECT @supplier\_id = id FROM furnizori WHERE nume = 'FurnizorExist';

IF @supplier\_id IS NULL

BEGIN

RAISERROR('Furnizorul nu există.', 16, 1);

END

INSERT INTO stocuri\_furnizori (furnizor\_id, ingredient\_id, calitate, pret, zile\_pana\_la\_livrare)

VALUES (@supplier\_id, 1, 'Bună', 10.99, 3);

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

-- Puteți înregistra sau gestiona eroarea aici

SELECT ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

-- Aceste tranzacții gestionează erorile care pot apărea în timpul executării și revin la starea anterioară a datelor în caz de eroare.

--1. Creati o procedura stocata care primeste doi parametri de intrare pe baza carora se fac actualizari pe doua tabele. Procedura returneaza doi parametri de iesire reprezentand numarul de linii afectate de fiecare dintre actualizari. Exemplificati un apel al acestei proceduri

--2. Scrieti un trigger de tip AFTER sau INSTEAD OF. (Acesta sa faca altceva decat verificarea dublurii unei anumite valori in tabela pe care este creat triggerul). Creati evenimentul necesar declansarii triggerului.

--3. In cadrul unei proceduri stocate creati o tranzitie care sa realizeze o inserare pe baza a doi parametrii de intrare, dintre care unul reprezinta valoarea cheii primare care se doreste a fi inserata. gestionati eroarea de cheie primara. exemplificati apeluri ale acestei proceduri care sa determine tratarea acesteia.

--4. Creati o tranzitie in care sa se faca 3 INSERT-uri urmate de un update si un delete. Operatiunile de update si delete sa se realizeze, fiecare din ele, pe baza a cel putin 2 join-uri. Tratati erorile in mod structurat.

--1.

--putem utiliza această procedură pentru a actualiza starea de activitate a utilizatorilor (cum ar fi bucătarii, ospătarii etc.) și cantitatea de ingrediente disponibile în stocuri. Aceasta poate fi utilă în situații precum gestionarea utilizatorilor care sunt activi sau inactivi în sistem, sau actualizarea cantităților de ingrediente din stocuri în urma unei livrări de la furnizor sau în urma unei comenzi de la clienți. Folosind această procedură, putem economisi timp și efort, asigurând în același timp consistența datelor în mai multe tabele.

CREATE PROCEDURE UpdateTables2

@newActivity BIT,

@newCantitate DECIMAL(10,2),

@affectedRowsUsers INT OUTPUT,

@affectedRowsStocuri INT OUTPUT

AS

BEGIN

SET NOCOUNT ON;

-- Actualizare în tabela users

UPDATE users

SET activity = @newActivity;

SET @affectedRowsUsers = @@ROWCOUNT;

-- Actualizare în tabela stocuri\_ingrediente

UPDATE stocuri\_ingrediente

SET cantitate = @newCantitate;

SET @affectedRowsStocuri = @@ROWCOUNT;

END;

DROP PROCEDURE UpdateTables2

DECLARE @affectedUsers INT, @affectedStocuri INT;

EXEC UpdateTables2 @newActivity = 1, @newCantitate = 20.5, @affectedRowsUsers = @affectedUsers OUTPUT, @affectedRowsStocuri = @affectedStocuri OUTPUT;

SELECT 'Numărul de linii afectate în tabela users: ' + CONVERT(VARCHAR, @affectedUsers), 'Numărul de linii afectate în tabela stocuri\_ingrediente: ' + CONVERT(VARCHAR, @affectedStocuri);

--2

-- Pentru a monitoriza și a înregistra activitățile de conectare în sistemul de management culinar, se dorește crearea unui trigger AFTER în SQL Server care să fie declanșat după fiecare inserare a unui nou eveniment de conectare în tabela 'logging'. Triggerul trebuie să afișeze un mesaj de confirmare atunci când este declanșat și poate fi extins pentru a include și alte acțiuni ulterioare, cum ar fi actualizarea altor tabele sau trimiterea de notificări către administratorii sistemului.

CREATE TRIGGER TriggerLoggingAfterInsert

ON logging

AFTER INSERT

AS

BEGIN

-- Codul de acțiune al triggerului

PRINT 'Un nou eveniment a fost înregistrat în tabela logging.';

END;

INSERT INTO logging (logged\_user\_id, login\_time)

VALUES (1, GETDATE());

DROP TRIGGER TriggerLoggingAfterInsert

--3

-- În cadrul unui sistem de gestionare a utilizatorilor, este necesară crearea unei proceduri stocate în baza de date pentru a gestiona inserarea unui nou utilizator. Scopul este de a trata în mod adecvat cazurile în care se încearcă inserarea unei chei primare care există deja în tabelul "users". Procedura trebuie să ofere o soluție care să permită continuarea operației fără a afecta integritatea datelor. Pentru acest lucru, se propune implementarea unei logici care să identifice eroarea de cheie primară duplicată și să trateze această situație în mod corespunzător, astfel încât să se asigure adăugarea utilizatorului nou fără pierderi de date sau conflicte.

CREATE PROCEDURE InserareCuGestionareEroare (

@id INT,

@username VARCHAR(50),

@password VARCHAR(100),

@type VARCHAR(20),

@activity BIT

)

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

INSERT INTO users (id, username, password, type, activity)

VALUES (@id, @username, @password, @type, @activity);

END TRY

BEGIN CATCH

-- Verificăm dacă eroarea este de tip cheie primară duplicată

IF ERROR\_NUMBER() = 2627

BEGIN

PRINT 'Eroare: Cheia primară este duplicată. Se încearcă gestionarea...';

-- Aici poți trata eroarea cum dorești, poți genera o altă cheie unică sau poți lăsa procedura să continue fără să facă nimic.

END

ELSE

BEGIN

-- Dacă eroarea nu este de tip cheie primară duplicată, afișăm mesajul de eroare

THROW;

END

END CATCH

END;

DROP PROCEDURE InserareCuGestionareEroare

-- codul de testare merge doar daca pe coloana id, este anulata proprietatea de autoincrement. Dupa testare, am refacut baza de date la starea initiala

-- Exemplu de apel al procedurii cu o cheie primară care nu există deja

EXEC InserareCuGestionareEroare @id = 1, @username = 'utilizator1', @password = 'parola1', @type = 'utilizator', @activity = 1;

-- Exemplu de apel al procedurii cu o cheie primară care există deja (va genera o eroare gestionată)

EXEC InserareCuGestionareEroare @id = 1, @username = 'utilizator2', @password = 'parola2', @type = 'utilizator', @activity = 1;

-- 4

-- Pentru a face testing pe baza de date, administartorul dorește implementarea unei tranzacții în cadrul unei baze de date pentru a efectua mai multe operațiuni de modificare a datelor. Aceste operațiuni includ inserarea unor noi înregistrări în tabelele "users", "furnizori" și "ingrediente", actualizarea unor înregistrări existente în tabela "furnizori" și ștergerea unor înregistrări din tabela "ingrediente". Fiecare operațiune de modificare este bazată pe cel puțin două join-uri cu alte tabele. Scopul este de a asigura consistența datelor și de a gestiona în mod structurat orice eroare care ar putea apărea în timpul tranzacției. Soluția propusă constă în implementarea unei proceduri stocate care să încadreze toate aceste operațiuni într-o singură tranzacție și să trateze în mod adecvat orice eroare care ar putea interveni, asigurând astfel integritatea și fiabilitatea datelor din baza de date.

CREATE PROCEDURE TranzitieCuTratareErori

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

BEGIN TRANSACTION; -- Începe tranzacția

-- Inserare 1

INSERT INTO users (username, password, type, activity)

VALUES ('utilizator1', 'parola123', 'utilizator', 1);

-- Inserare 2

INSERT INTO furnizori (nume) VALUES ('Furnizor1');

-- Inserare 3

INSERT INTO ingrediente (nume) VALUES ('Ingrediente1');

-- Update

UPDATE furnizori

SET nume = 'NoulNume'

FROM furnizori AS f

INNER JOIN stocuri\_furnizori AS sf ON f.id = sf.furnizor\_id

WHERE sf.calitate = 'Buna';

-- Delete

DELETE FROM ingrediente

WHERE id IN (

SELECT i.id

FROM ingrediente AS i

INNER JOIN stocuri\_ingrediente AS si ON i.id = si.ingredient\_id

INNER JOIN retete AS r ON si.reteta\_id = r.id

WHERE r.nume = 'Reteta1'

);

COMMIT TRANSACTION; -- Confirmă tranzacția dacă toate instrucțiunile sunt executate cu succes

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION; -- Anulează tranzacția în caz de eroare

-- Afisează mesajul de eroare

PRINT 'Eroare: ' + ERROR\_MESSAGE();

END CATCH;

END;

DROP PROCEDURE TranzitieCuTratareErori