

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.

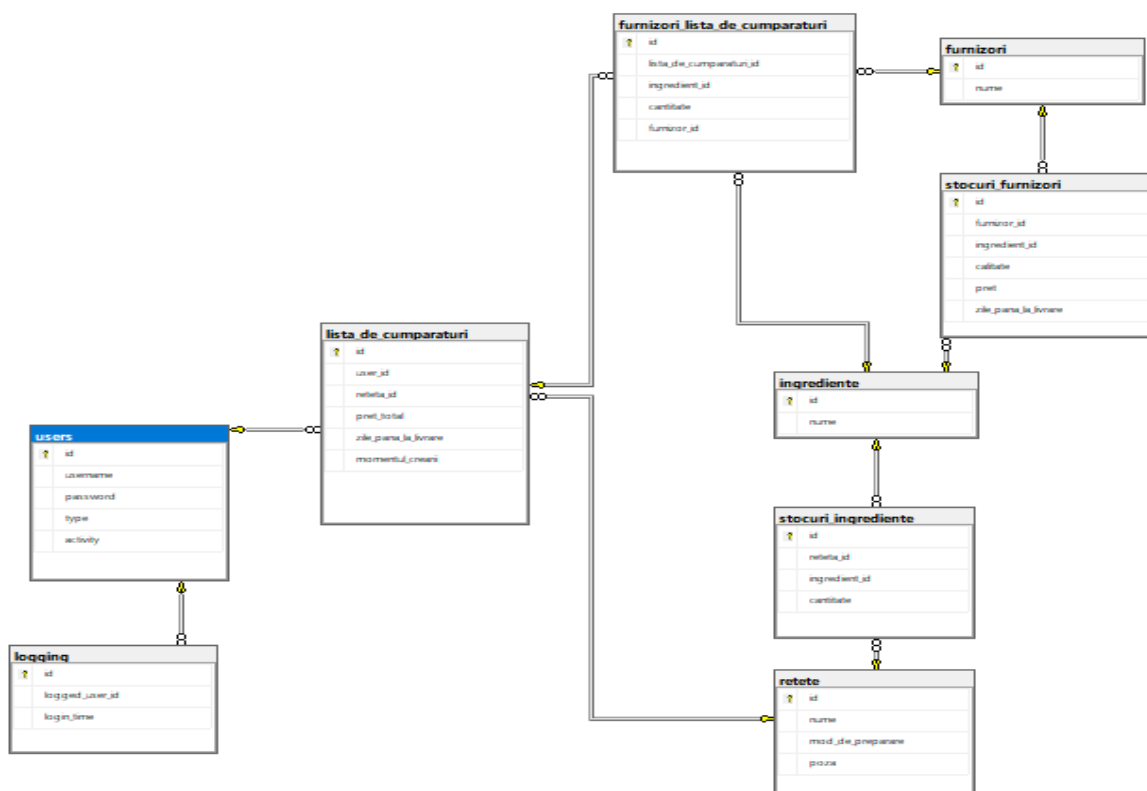
5. **Gestionarea eficientă a inventarului:** Funcționalitățile de administrare a ingredientelor și furnizorilor pot fi utile și pentru profesioniștii din industria alimentară, facilitând gestionarea stocurilor și aprovizionarea.

Astfel, baza de date pentru gestionarea serverului Dummy Chef are ca scop gestionarea resurselor acestei aplicații și furnizarea constantă de date către aplicație. 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 să utilizeze această bază de date și să permită automatizarea aceteia.

2. 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_ingredientes (
    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 rețeta 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 rețeta respectivă, definită ca un număr cu două zecimale.

Acest tabel "stocuri_ingredientes" va permite să gestionezi relația între rețete și ingredientele necesare pentru a le pregăti, păstrând înregistrări ale cantităților de ingrediente necesare pentru fiecare rețetă.

```

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.

3. 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 rețetă 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.

!!!!!!!

4. Select

- Select cu filtrare de date calendaristice:

```
SELECT *
FROM logging
WHERE login_time >= '2024-03-19' AND login_time < '2024-03-20';
```

100 %

Results Messages

	id	logged_user_id	login_time
1	2	2	2024-03-19 10:30:00.000
2	3	3	2024-03-19 11:45:00.000

- Select cu prelucrare de date de tip caracter:

```
SELECT id, nume, UPPER(nume) AS nume_mare
FROM ingrediente;
```

100 %

Results Messages

	id	nume	nume_mare
1	1	faina	FAINA
2	2	oua	OUA
3	3	zahar	ZAHAR

- Select folosind clauza Order By:

```
SELECT *
FROM retete
ORDER BY nume;
```

100 %

Results Messages

	id	nume	mod_de_preparare	poza
1	2	Omleta	Se bat ouale, se adauga sare si piper, apoi se praje...	NULL
2	1	Placinta cu mere	Se amesteca faina cu ouale si zaharul, apoi se ada...	NULL
3	3	Tiramisu	Se amesteca mascarpone cu zaharul si galbenusuri...	NULL

- Select cu filtrarea datelor folosind TOP:

```
SELECT TOP 2 *
FROM users;
```

100 %

Results Messages

	id	username	password	type	activity
1	1	utilizator1	parola123	utilizator	1
2	2	bucatar1	parola456	bucatar	1

- Select cu filtrarea și ordonarea datelor:

```
SELECT *
FROM furnizori_lista_de_cumparaturi
WHERE cantitate > 100
ORDER BY cantitate DESC;
```

100 %

Results Messages

	id	lista_de_cumparaturi_id	ingredient_id	cantitate	furnizor_id
1	1	1	1	500.00	1
2	3	3	3	300.00	3

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

```
SELECT *
FROM lista_de_cumparaturi
WHERE user_id = 1 AND pret_total IS NOT NULL;
```

100 %

Results Messages

	id	user_id	reteta_id	pret_total	zile_pana_la_livrare	momentul_crearii
1	1	1	2	2995.00	2	2024-03-20 11:36:21.457

- Select cu ordonare pe mai multe coloane:

```
SELECT *
FROM stocuri_furnizori
ORDER BY furnizor_id, ingredient_id;
```

100 %

Results Messages

	id	furnizor_id	ingredient_id	calitate	pret	zile_pana_la_livrare
1	1	1	1	buna	5.99	2
2	2	2	2	foarte buna	7.50	3
3	3	3	3	excelenta	10.25	1

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

```
SELECT
    CONCAT_WS(' - ', i.nume, ISNULL(calitate, 'Necunoscută')) AS detalii_ingredient
FROM stocuri_furnizori sf
INNER JOIN ingrediente i ON sf.ingredient_id = i.id;
```

100 %

Results Messages

	detalii_ingredient
1	faina - buna
2	oua - foarte buna
3	zahar - excelenta

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

```
SELECT DISTINCT username, type
FROM users
WHERE activity = 1
ORDER BY type;
```

00 %

Results Messages

	username	type
1	admin1	admin
2	bucatar1	bucatar
3	utilizator1	utilizator

Partea II

Jonctiuni pe cel puțin 3 tabele

1. Jonctiune între tabela **lista_de_cumparaturi**, **retete** și **users** pentru a obține detalii despre lista de cumpărături împreună cu numele utilizatorului și numele retetei.

```
SELECT lc.id AS lista_id, u.username, r.num AS reteta_num
FROM lista_de_cumparaturi lc
JOIN users u ON lc.user_id = u.id
JOIN retete r ON lc.reteta_id = r.id;
```

121 %

Results Messages

	lista_id	username	reteta_num
1	1	utilizator1	Omleta
2	2	bucatar1	Omleta
3	3	admin1	Tiramisu

- Joncțiune între tabela **stocuri_ingrediente**, **ingredientes** și **retete** pentru a obține detalii despre ingredientele necesare pentru fiecare rețetă.

```
SELECT si.reteta_id, i.nume AS ingredient_nume, si.cantitate, r.nume AS reteta_nume
FROM stocuri_ingredientes si
JOIN ingrediente i ON si.ingredient_id = i.id
JOIN retete r ON si.reteta_id = r.id;
```

	reteta_id	ingredient_nume	cantitate	reteta_nume
1	1	faina	300.00	Placinta cu mere
2	1	oua	4.00	Placinta cu mere
3	1	zahar	150.00	Placinta cu mere
4	2	oua	6.00	Omleta
5	3	oua	12.00	Tiramisu
6	3	faina	200.00	Tiramisu
7	3	zahar	250.00	Tiramisu

- Joncțiune între tabela **furnizori_lista_de_cumparaturi**, **furnizori** și **ingredientes** pentru a obține detalii despre furnizorii pentru fiecare ingredient din lista de cumpărături.

```
SELECT flc.lista_de_cumparaturi_id, f.nume AS furnizor_nume, i.nume AS ingredient_nume, flc.cantitate
FROM furnizori_lista_de_cumparaturi flc
JOIN furnizori f ON flc.furnizor_id = f.id
JOIN ingrediente i ON flc.ingredient_id = i.id;
```

	lista_de_cumparaturi_id	furnizor_nume	ingredient_nume	cantitate
1	1	Furnizor1	faina	500.00
2	2	Furnizor2	oua	8.00
3	3	Furnizor3	zahar	300.00

- Joncțiune între tabela **stocuri_furnizori**, **furnizori** și **ingredientes** pentru a obține detalii despre stocurile furnizorilor pentru fiecare ingredient.

```
SELECT sf.furnizor_id, f.nume AS furnizor_nume, i.nume AS ingredient_nume, sf.calitate, sf.pret
FROM stocuri_furnizori sf
JOIN furnizori f ON sf.furnizor_id = f.id
JOIN ingrediente i ON sf.ingredient_id = i.id;
```

	furnizor_id	furnizor_nume	ingredient_nume	calitate	pret
1	1	Furnizor1	faina	buna	5.99
2	2	Furnizor2	oua	NULL	7.50
3	3	Furnizor3	zahar	excelenta	10.25

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**.

```
SELECT username
FROM users
WHERE activity = 1;
```

	username
1	utilizator1
2	bucatar1
3	admin1

2. **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**.

```
SELECT i.nume AS ingredient, SUM(si.cantitate) AS cantitate_disponibila
FROM stocuri_ingrediente si
JOIN ingrediente i ON si.ingredient_id = i.id
GROUP BY i.nume;
```

	ingredient	cantitate_disponibila
1	faina	500.00
2	oua	22.00
3	zahar	400.00

3. **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.numa;
```

4. **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**.

The screenshot shows a SQL query in a query editor and its results in a table. The query is:

```
SELECT DISTINCT f.numa AS furnizor
FROM stocuri_furnizori sf
JOIN furnizori f ON sf.furnizor_id = f.id
WHERE sf.calitate = 'excelenta';
```

The results table has two columns: 'furnizor' and 'numa'. The first row shows the value 'Furnizor3' in the 'furnizor' column and '1' in the 'numa' column.

	furnizor	numa
1	Furnizor3	1

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

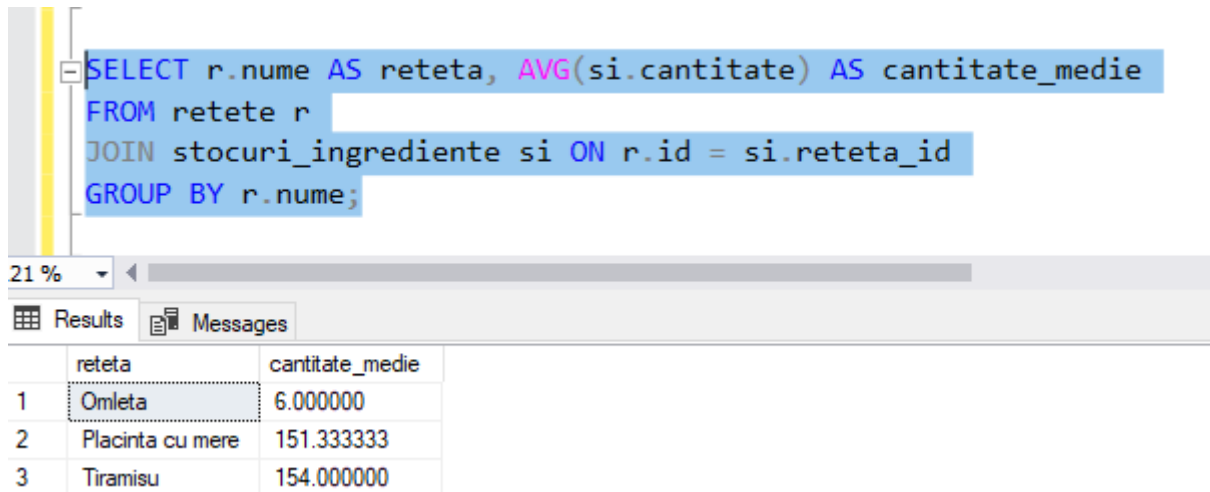
The screenshot shows a SQL query in a query editor and its results in a table. The query is:

```
SELECT u.type, COUNT(r.id) AS numar_retete
FROM users u
JOIN rețete r ON u.id = r.id
GROUP BY u.type;
```

The results table has two columns: 'type' and 'numar_retete'. The first three rows show the values 'admin', 'bucatar', and 'utilizator' in the 'type' column, each with a value of '1' in the 'numar_retete' column.

	type	numar_retete
1	admin	1
2	bucatar	1
3	utilizator	1

2. **Media cantității necesare de ingrediente pentru fiecare rețetă:** Această interogare va returna media cantității necesare de ingrediente pentru fiecare rețetă din tabela **stocuri_ingrediente**.



```

SELECT r.nume AS reteta, AVG(si.cantitate) AS cantitate_medie
FROM retete r
JOIN stocuri_ingredientes si ON r.id = si.reteta_id
GROUP BY r.nume;

```

	reteta	cantitate_medie
1	Omleta	6.000000
2	Placinta cu mere	151.333333
3	Tiramisu	154.000000

3. **Prețul total al ingredientelor necesare pentru fiecare rețetă:** Această interogare va returna prețul total al ingredientelor necesare pentru fiecare rețetă din tabela **stocuri_furnizori**.

```

SELECT r.nume AS reteta, SUM(sf.pret * si.cantitate) AS pret_total
FROM retete r
JOIN stocuri_ingredientes si ON r.id = si.reteta_id
JOIN stocuri_furnizori sf ON si.ingredient_id = sf.ingredient_id
GROUP BY r.nume;

```

4. 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 rețete: Această interogare va returna utilizatorii activi care au creat rețete din tabela **users** și **retete**.

```

SELECT r.nume AS reteta, COUNT(si.ingredient_id) AS numar_ingredientes
FROM retete r
JOIN stocuri_ingredientes si ON r.id = si.reteta_id
GROUP BY r.nume

```

```
HAVING COUNT(si.ingredient_id) > 5;
```

2. 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.num AS reteta, COUNT(si.ingredient_id) AS numar_ingrediente
FROM retete r
JOIN stocuri_ingredient si ON r.id = si.reteta_id
GROUP BY r.num
HAVING COUNT(si.ingredient_id) > 5;
```

3. 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.num AS reteta
    FROM users u
    JOIN retete r ON u.id = r.id
)
SELECT * FROM UtilizatoriCuRetete;
```

4. 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.num 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.num AS reteta
    FROM users u
    JOIN retete r ON u.id = r.id
)
SELECT * FROM UtilizatoriCuRetete;
```


2. 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;
```

3. 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;
```

4. 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.num 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;

```

2. 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.num AS furnizor, i.num 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;

```

3. 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.num AS reteta, si.cantitate AS cantitate_oua
FROM retete r
JOIN stocuri_ingredientes si ON r.id = si.reteta_id
JOIN ingrediente i ON si.ingredient_id = i.id
WHERE i.num = 'oua';

```

4. 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.num AS furnizor, sf.calitate
FROM furnizori f
JOIN stocuri_furnizori sf ON f.id = sf.furnizor_id
WHERE sf.calitate = 'excelenta';

```

5. 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.num AS reteta, flc.cantitate, i.num AS ingredient, f.num 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 rețetele 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. Creați o procedură stocată care primește doi parametri de intrare pe baza cărora se fac actualizări pe două tabele. Procedura returnează doi parametri de ieșire reprezentând numărul de linii afectate de fiecare dintre actualizări. Exemplificați un apel al acestei proceduri

--2. Scrieți un trigger de tip AFTER sau INSTEAD OF. (Acesta să facă altceva decât verificarea dublurii unei anumite valori în tabela pe care este creat triggerul). Creați evenimentul necesar declansării triggerului.

--3. În cadrul unei proceduri stocate creați o tranziție care să realizeze o inserare pe baza a doi parametri de intrare, dintre care unul reprezintă valoarea cheii primare care se dorește a fi inserată. gestionați eroarea de cheie primară. exemplificați apeluri ale acestei proceduri care să determine tratamentul acesteia.

--4. Creați o tranziție în care să se facă 3 INSERT-uri urmate de un update și un delete. Operațiunile de update și delete să se realizeze, fiecare din ele, pe baza a cel puțin 2 join-uri. Tratați erorile în 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

```