TESINA SISTEMI INFORMATIVI

la birreria

1 GENNAIO 2024

SIMONE RASTELLI
[UNIMORE]
189632

Sommario

Descrizione del problema

Il problema affrontato riguarda la digitalizzazione delle operazioni quotidiane di una birreria. Con lo scopo principale di ottimizzare e digitalizzare le seguenti funzioni:

1. **Prenotazioni**: I dipendenti possono accedere ad una pagina per creare una nuova prenotazione, tutte le prenotazioni saranno salvate in una tabella e sono modificabili ed eliminabili.

2. **Gestione delle comande**: Le comande vengono effettuate da un dipendente e sono assegnate a un tavolo e contengono informazioni sulle portate ordinate, come tipo di portata, la quantità, prezzo e disponibilità.
   È possibile visualizzare la lista di comande create con prezzo totale, orario, id_dipendente che ha effettuato la prenotazione, è anche possibile modificarle ed eliminarle.

3. **Gestione del magazzino**: Include la gestione degli ingredienti necessari per le portate, con monitoraggio delle scorte.ad ogni piatto preso il magazzino si aggiorna in automatico attraverso un trigger.

Caratteristiche dello schema

Lo schema comprende diverse entità con relazioni chiave, tra cui:

1. **Dipendente**: Può effettuare più prenotazioni. Ha una relazione di tipo "uno a molti" con le prenotazioni.

2. **Prenotazione**: associa a una prenotazione un tavolo disponibile e nell’area richiesta.

3. **Tavolo**: Ogni tavolo ha un numero massimo di posti disponibili e può essere occupato da una prenotazione. Relazione "uno a molti" con le prenotazioni.
   Difatti un tavolo può essere associato a più prenotazioni.
   La birreria possiede 3 tavoli da 2 , 4 tavoli da 4, 2 tavoli da 8 nello spazio fuori.
   5 tavoli da 2, 8 tavoli da 4 e 2 tavoli da 8 dentro il locale.

4. **Comanda**: Viene effettuata da un dipendente che segna il numero del tavolo a cui portare le portate ordinate, con il numero di portate.
   A comanda effettuata viene generato anche il prezzo totale.
   È contenuta in una relazione "uno a molti" tra tavolo e portate.

5. **Portata**: Include le portate contenute nel menu tra cui cibi, bevande e birra, con dettagli come prezzo, capacità e tipo (es. bevanda, piatto, birra).
   Il menu della birreria è il seguente:

Panini
Classic Burger: Hamburger di manzo, insalata, pomodoro, cheddar, salsa speciale.
Prezzo: €8,00
Veggie Dream: Burger vegetariano, pomodoro, insalata, salsa speciale.
Prezzo: €7,50

Hot Dog
Classic Dog: Wurstel di maiale, senape, ketchup.
Prezzo: €6,50

Tipi di Patatine
Classiche: Con sale marino. **Prezzo**: €4,00

Birre Artigianali e Bevande
Pilsner Light: Lager chiara, 4,5% vol, 330\500 ml.
Prezzo: €4,50-5,5
Stout Dream: Birra scura al caffè, 7% vol, 330\500 ml.
Prezzo: €6,00-7
Weiss Sun: Birra di frumento, 5% vol, 330\500 ml.
Prezzo: €5,50-6.5
Bevande Analcoliche
Coca-Cola (330\500 cl) - €2,50-3,5
Fanta (330\500 cl) - €2,50-3,5
Acqua Naturale (33 cl) - €1,50
Acqua Frizzante (33 cl) - €1,50

6. **Ingrediente**: specifica il nome degli ingredienti usati nelle portate.

7. **Usati**: contiene le quantità di ingredienti usati che possono essere misurate in kili, litri o porzione\bottiglia. Entità necessaria per l'aggiornamento automatico del magazzino.

8. **Stock**: Contiene le scorte degli ingredienti utilizzati per le portate e ne traccia la disponibilità, aggiornandosi in automatico ad ogni portata ordinata sottraendo gli ingredienti usati dagli ingredienti in stock.

9. **Magazzino**: Luogo contenente le scorte degli ingredienti, si sviluppa su due piani.

ANALISI DELLE SPECIFICHE

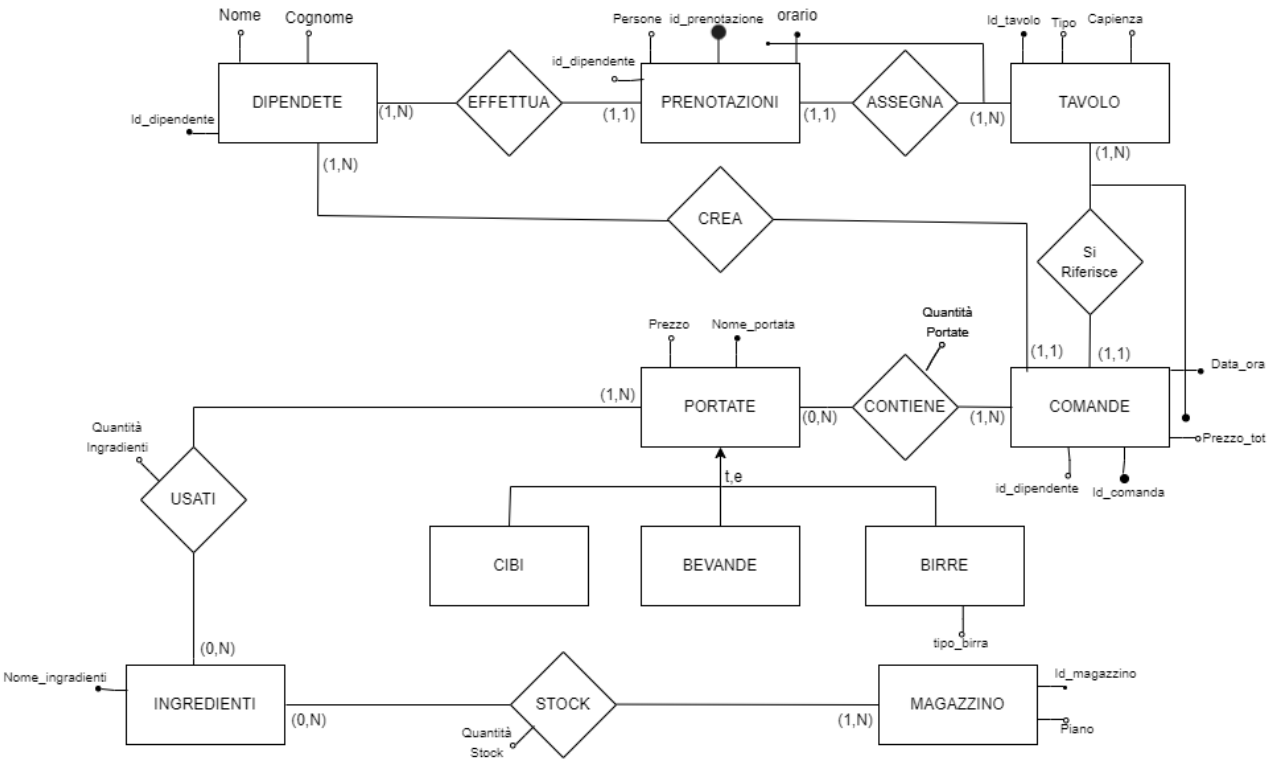
Entità	Attributi	Descrizione/Sinonimi	In relazione con
Dipendente	Id_dipendente,nome cognome	Impiegato che può effettuare prenotazioni e comande.	Prenotazione, Comanda
Prenotazione	Data_ora, Id_tavolo, numero_persone, Id_prenotazione, Id_dipendente	Associa ai clienti un tavolo e quindi lo prenota.	Dipendente, Tavolo
Tavolo	Id_tavolo, capienza, tipo	Tavoli disponibili con capienza e posizione (dentro/fuori).	Prenotazione, Comanda
Comanda	Id_comanda, Data_ora, Id_tavolo, prezzo_totale, Id_dipendente	Ordine effettuato per un tavolo, con portate e totale calcolato.	Tavolo, Dipendente, Contenuto Comanda
Contenuto Comanda	Id_comanda, nome_portata, quantità	Dettagli delle portate ordinate con quantità per una comanda.	Comanda, Portata
Portata	nome_portata, prezzo, tipo_birra	Include cibi, bevande e birre con dettagli come prezzo e capacità all'interno del menu.	Contenuto Comanda, Usati
Ingrediente	nome_ingrediente	Nome degli ingredienti utilizzati nelle portate.	Usati, Stock
Usati	nome_portata, nome_ingrediente, quantità_usata	Quantità di ingredienti utilizzata per ogni portata.	Portata, Ingrediente
Magazzino	id_magazzino, piano	Magazzino che traccia la disponibilità degli ingredienti.	Stock
Stock	nome_ingrediente, id_magazzino, quantità_stock	Quantità di ingredienti disponibili nel magazzino.	Ingrediente, Magazzino

FUNZIONALITÀ DEL PROGRAMMA

1. **Gestione delle comande:**
    - Esegue una query per recuperare tutte le comande e le mostra tutte.
    - Permette la creazione di una nuova comanda e la sua registrazione nel database.
    - Calcola automaticamente il prezzo totale basato sulle portate ordinate e aggiorna il totale nella tabella delle comande.
    - Consente di recuperare i dettagli di una comanda, inclusi portate, quantità, ID tavolo e ID dipendente che ha creato la comanda.
    - Consente la modifica di una comanda già esistente, permettendo di aggiungere nuove portate o eliminarne.
    - Presenta un pulsante per eliminare comande esistenti.
2. **Gestione delle prenotazioni:**
    - Esegue una query per recuperare tutte le prenotazioni e restituisce un elenco di tutte le prenotazioni effettuate.
    - Consente di creare una nuova prenotazione e inserirla nel database, impedendo la prenotazione doppia dello stesso tavolo.
    - Permette di modificare una prenotazione esistente, consentendo di aggiornare la data, l'orario o il numero di persone.
    - Presenta un pulsante per eliminare prenotazioni esistenti.
3. **Gestione degli ingredienti:**
    - Mostra l'elenco delle scorte di ingredienti disponibili in magazzino.
    - Aggiorna automaticamente le scorte di ingredienti in base alle portate ordinate.
    - Permette l'inserimento di nuovi ingredienti nel sistema.

SCHEMA ER

Per la creaziazione dello schema E-R ho usato una strategia mixed, cioè ho utilizzato sia strategie top-down che bottom-up



1. **DIPENDENTE - PRENOTAZIONI (Relazione: EFFETTUA)**

Descrizione della Relazione:

- La relazione EFFETTUA collega l'entità DIPENDENTE con l'entità PRENOTAZIONI.

Cardinalità:

- (1,N) dal lato DIPENDENTE: Un dipendente può effettuare più prenotazioni.
- (1,1) dal lato PRENOTAZIONI: Ogni prenotazione è effettuata da un solo dipendente.

Funzionalità:

- Traccia quale dipendente ha effettuato una prenotazione.

---

2. PRENOTAZIONI - TAVOLO (Relazione: ASSEGNA)

Descrizione della Relazione:

- La relazione ASSEGNA collega l'entità PRENOTAZIONI con l'entità TAVOLO.

Cardinalità:

- (1,1) dal lato PRENOTAZIONI: A ogni prenotazione viene assegnato un solo tavolo.

- (1,N) dal lato TAVOLO: Un tavolo può essere assegnato a più prenotazioni in orari diversi.

---

3. DIPENDENTE - COMANDE (Relazione: CREA)
Descrizione della Relazione:
- La relazione CREA collega l'entità DIPENDENTE con l'entità COMANDE.

Cardinalità:
- (1,N) dal lato DIPENDENTE: Un dipendente può creare più comande.
- (1,1) dal lato COMANDE: Ogni comanda è creata da un solo dipendente.

Funzionalità:
- Traccia quale dipendente ha creato ciascuna comanda.

---

4. TAVOLO - COMANDE (Relazione: SI RIFERISCE)
Descrizione della Relazione:
- La relazione SI RIFERISCE collega l'entità TAVOLO con l'entità COMANDE.

Cardinalità:
- (1,1) dal lato COMANDE: Ogni comanda si riferisce a un solo tavolo.
- (1,N) dal lato TAVOLO: Un tavolo può avere più comande nel tempo.

Funzionalità:
- Traccia a quale tavolo è riferita una specifica comanda.

---

5. COMANDE - PORTATE (Relazione: CONTENUTO COMANDA)
Descrizione della Relazione:
- La relazione CONTENUTO COMANDA collega l'entità COMANDE con l'entità PORTATE.

Cardinalità:
- (1,N) dal lato COMANDE: Una comanda può contenere più portate.
- (0,N) dal lato PORTATE: Una portata può essere inclusa in più comande.

Attributi della Relazione:
- Quantità_Portate: Indica la quantità di una portata specifica inclusa nella comanda.

---

6. PORTATE - INGREDIENTI (Relazione: USATI)
Descrizione della Relazione:
- La relazione USATI collega l'entità PORTATE con l'entità INGREDIENTI. Cardinalità:
- (1,N) dal lato PORTATE: Una portata può utilizzare più ingredienti.
- (0,N) dal lato INGREDIENTI: Un ingrediente può essere utilizzato in più portate.

Attributi della Relazione:
- Quantità_Ingredienti: Indica la quantità specifica di un ingrediente usata per preparare una portata.

Funzionalità:
- Traccia il consumo di ingredienti per ciascuna portata.

---

7. INGREDIENTI - MAGAZZINO (STOCK)

Descrizione della Relazione:

- Questa relazione collega INGREDIENTI con MAGAZZINO.

Cardinalità:

- (0,1) dal lato INGREDIENTI: Un ingrediente può essere contenuto in un Magazzino.
- (1,N) dal lato MAGAZZINO: Un magazzino può gestire più voci di ingredienti

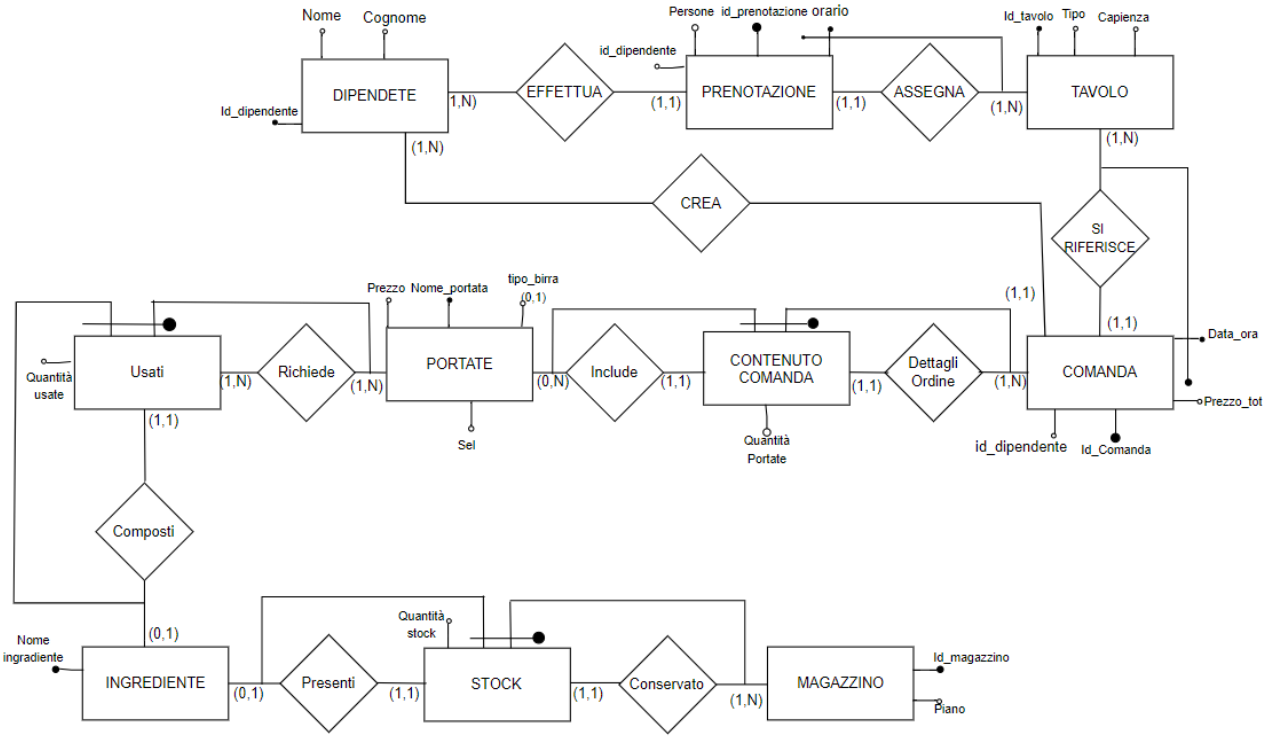
Attributi della Relazione:

- Quantità_Stock: Quantità attuale di quell'ingrediente.

Funzionalità:

- Permette di monitorare le quantità disponibili degli ingredienti.
- Permette di vedere in quale magazzino è localizzato l’ingrediente.

SEMPLIFICAZIONE MODELLO E-R



MODIFICHE APPORTATE

COLLASSO VERSO L'ALTO NELLA TABELLA PORTATE

E' un tipo di semplificazione sempre possibile, non necessita di alcuna copertura specifica le entità figlie (birre,bevande,cibi) vengono "collassate" nell'entità padre Portate.
Che quindi mantiene i suoi attributi (prezzo,nome_portata) e aggiunge un attributo sel (selettore) che essendo una gerarchia con copertura totale ed esclusiva ha N valori, quante sono le entità figlie.(sel{birre,bevande,cibi}).
Gli attributi delle entità figlie diventano attributi opzionali (0,1) dell'entitò padre; come tipo_birra(0,1).

RETIFICAZIONI

1.RETTIFICA IN STOCK

Essendo STOCK un associazione con un attributo devo rettificare l'associazione creando un entità STOCK e due associazioni "PRESENTI" e "CONSERVATO"

Relazione con INGREDIENTE:

- La relazione "PRESENTI" collega Stock a Ingrediente con cardinalità (1,1) - (0,1).
    - Partendo da Stock cardinalità (1,1) ogni record di Stock deve riferirsi a un singolo Ingrediente (es. birra, hamburger).

- Partendo da ingrediente cardinalità (0,1) un Ingrediente può essere presente oppure no in uno Stock.

Relazione con MAGAZZINO:

- La relazione "CONSERVATO" collega Stock a Magazzino con cardinalità (1,1) - (1,N).
  - Partendo da Stock con cardinalità (0,1) ogni record di Stock può trovarsi, se non è finito il prodotto, in un singolo Magazzino.
  - Partendo da Magazzino con cardinalità (1,N) Un Magazzino può conservare più Stock, ovvero registrare più quantità di ingredienti diversi.

RETTIFICA IN USATI

Essendo USATI un'associazione con un attributo devo rettificare l'associazione, creando un'entità USATI e due associazioni "RICHIEDE" e "COMPOSTI"

Relazione con PORTATE:

- La relazione "RICHIEDE" collega Usati a Portate con cardinalità (1,N) - (1,N).
  - Partendo da Usati con cardinalità (1,N) ogni record di Usati appartiene a una o più Portate
  - Partendo da Portate con cardinalità (1,N) una Portata può richiedere più ingredienti, ciascuno registrato in un record distinto nella tabella Usati.

Relazione con INGREDIENTE:

- La relazione "COMPOSTI" collega Usati a Ingrediente con cardinalità (1,1) - (0,1).
  - Partendo da Usati con cardinalità (1,1) ogni record di Usati si riferisce a un singolo Ingrediente utilizzato.
  - Partendo da Ingrediente con cardinalità (0,1) un Ingrediente può essere utilizzato oppure no.

RETTIFICA IN CONTENUTO_COMANDA

Essendo CONTENUTO_COMANDA un associazione con un attributo devo rettificare l'associazione creando un entità CONTENUTO_COMANDA e due associazioni "INCLUDE" e "DETTAGLI_ORDINE"

Relazione con PORTATE:

- La relazione "INCLUDE" collega Contenuto_Comanda a Portate con cardinalità (1,1) - (0,N).
  - Partendo da Contenuto_Comanda con cardinalità (1,1) significa che ogni record di Contenuto_Comanda è riferito a una singola Portata
  - Partendo da Portata con cardinalità(0,N) Una Portata può essere inclusa o non inclusa in più comande attraverso più record in Contenuto_Comanda.

Relazione con COMANDA:

- La relazione "DETTAGLI ORDINE" collega Contenuto_Comanda a Comanda con cardinalità (1,1) - (1,N).
    - Partendo da Contenuto_comanda con cardinalità(1,1) Ogni record di Contenuto_Comanda appartiene a una singola Comanda.
    - Partendo da Comanda con cardinalità (1,N) Una Comanda può includere uno o più record di Contenuto_Comanda per registrare portate diverse e relative quantità.

PROGETTAZIONE LOGICA

Dopo aver completato la semplificazione del modello E/R, si passa alla fase di modellazione logica.
Durante questa fase, vengono immediatamente identificate le chiavi primarie, le chiavi alternative e le chiavi esterne.
Si adotta un approccio standard per la traduzione.

TRADUZIONE STANDARD

Dipendente (Id_dipendente, nome, cognome)
Prenotazione (Id_prenotazione Data_ora, Id_tavolo, numero_persone, id_dipendente)
FK: Id_tavolo references Tavolo NOT NULL
AK: Id_tavolo, Data_ora
Tavolo (Id_tavolo, capienza, tipo)
Comanda (Id_comanda,Data_orario, Id_tavolo, prezzo_totale,id_dipendente)
FK: Id_tavolo references Tavolo NOT NULL
AK: Data_orario, Id_tavolo
Contenuto Comanda (Id_comanda, nome_portata, quantità)
FK: Id_comanda references Comanda
FK: nome_portata references Portata
Portata (nome_portata, prezzo, sel, tipo_birra)
Usati (nome_portata, nome_ingrediente, quantità_usata)
FK: nome_portata references Portata NOT NULL
FK: ingrediente references Ingredienti
Ingrediente (nome_ingrediente)

Stock (nome_ingrediente, id_magazzino, quantità_stock)
FK: nome_ingrediente references Ingrediente
FK: id_magazzino references Magazzino
Magazzino (id_magazzino, piano)

MODELLO FISICO(DDL)

Utilizzando l’applicazione pgAdmin4, è possibile creare un programma che riesce a creare delle tabelle che contengono tutti i dati necessari e consente di eseguire operazioni e interrogazioni relative alla gestione della birrera.
La definizione di uno schema in pgAdmin4 avviene tramite una query che può essere eseguita attraverso il Query Tool.

--CREO LO SCHEMA:

```
CREATE SCHEMA IF NOT EXISTS "189632_simone"
    AUTHORIZATION postgres;

COMMENT ON SCHEMA "189632_simone"
    IS 'standard public schema';

GRANT ALL ON SCHEMA "189632_simone" TO PUBLIC;

GRANT ALL ON SCHEMA "189632_simone" TO postgres;

set search_path to "189632_simone";

--CREO TABELLE
DROP TABLE IF EXISTS Contenuto_Comanda CASCADE;
DROP TABLE IF EXISTS Comande CASCADE;
DROP TABLE IF EXISTS Usati CASCADE;
DROP TABLE IF EXISTS Portata CASCADE;
DROP TABLE IF EXISTS Prenotazioni CASCADE;
DROP TABLE IF EXISTS Tavolo CASCADE;
DROP TABLE IF EXISTS Stock CASCADE;
DROP TABLE IF EXISTS Ingrediente CASCADE;
DROP TABLE IF EXISTS Magazzino CASCADE;
DROP TABLE IF EXISTS Dipendente CASCADE;

-- Cliente (con clienti fedeltà e nuovi clienti)
CREATE TABLE Dipendente(
  Id_dipendente SERIAL PRIMARY KEY,
  nome VARCHAR(50) NOT NULL,
  cognome VARCHAR(50) NOT NULL
 );
-- Tavolo (con capienza e disponibilità)
CREATE TABLE Tavolo (
  Id_tavolo SERIAL PRIMARY KEY,--quando si esegue un INSERT in una tabella con una colonna SERIAL, non è necessario specificare manualmente il valore dell'ID - PostgreSQL si occupa di generarlo automaticamente.
```

```
  capienza INT NOT NULL,
  tipo VARCHAR(20) NOT NULL
);

-- Prenotazione (associata a uno o più tavoli)
CREATE TABLE Prenotazioni (
  Id_prenotazione SERIAL not null,
  Data_ora TIMESTAMP(0) NOT NULL,--contiene data e ora
  Id_tavolo INT NOT NULL,--Avendo dichiarato Id_tavolo come SERIAL nella
tabella Tavolo, i valori generati saranno numeri interi, quindi è corretto utilizzare il
tipo INT per la chiave straniera nella tabella Prenotazione.
  numero_persone INT NOT NULL,
  id_dipendente int,
  PRIMARY KEY (Id_prenotazione),
  FOREIGN KEY (Id_tavolo) REFERENCES Tavolo(Id_tavolo)
);

-- Creazione della tabella Comanda
CREATE TABLE Comande (
  id_comanda INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  id_dipendente int,
  Data_ora TIMESTAMP(0) NOT NULL,--contiene data e ora
  id_tavolo INT NOT NULL,
  prezzo_totale int,
  FOREIGN KEY (id_tavolo) REFERENCES Tavolo(id_tavolo),
  CONSTRAINT comanda_ak UNIQUE (Data_ora, id_tavolo) -- Alternate Key
);



-- Portata (cibi, bevande e birre con dettagli)
CREATE TABLE Portata (
  nome_portata VARCHAR(100) PRIMARY KEY,
  prezzo float NOT NULL,
  sel VARCHAR(20) NOT NULL CHECK (sel IN ('cibi', 'bevande','birra')),
  tipo_birra VARCHAR(50)
);
-- Contenuto Comanda (relazione molti a molti tra Comanda e Portata)
CREATE TABLE Contenuto_Comanda (
  id_comanda INT NOT NULL,
  nome_portata VARCHAR(100) NOT NULL,
  quantità INT NOT NULL,--almeno una
  PRIMARY KEY (id_comanda, nome_portata),
  FOREIGN KEY (id_comanda) REFERENCES Comande(id_comanda),
  FOREIGN KEY (nome_portata) REFERENCES Portata(nome_portata)
);
```

```
-- Ingrediente (utilizzati nelle portate)
CREATE TABLE Ingrediente (
  nome_ingrediente VARCHAR(100) PRIMARY KEY
);

-- Usati (relazione molti a molti tra Portata e Ingrediente)
CREATE TABLE Usati (
  nome_portata VARCHAR(100) NOT NULL,
  nome_ingrediente VARCHAR(100) NOT NULL,
  quantità_usata float NOT NULL,
  PRIMARY KEY (nome_portata, nome_ingrediente),
  FOREIGN KEY (nome_portata) REFERENCES Portata(nome_portata),
  FOREIGN KEY (nome_ingrediente) REFERENCES
Ingrediente(nome_ingrediente)
);

-- Magazzino (informazioni sul magazzino)
CREATE TABLE Magazzino (
  id_magazzino SERIAL PRIMARY KEY,
  piano INT NOT NULL
);

-- Stock (tracciando la disponibilità degli ingredienti)
CREATE TABLE Stock (
  nome_ingrediente VARCHAR(100),
  quantità_stock float NOT NULL,
  id_magazzino int NOT NULL,
  PRIMARY KEY(id_magazzino, nome_ingrediente),
  FOREIGN KEY (nome_ingrediente) REFERENCES
Ingrediente(nome_ingrediente),
  FOREIGN KEY (id_magazzino) REFERENCES Magazzino(id_magazzino)
);


/*TRIGGER STOCK*/

-- creo una funzione trigger che eseguo quando inserisco un nuovo record in
contenuto comanda
CREATE OR REPLACE FUNCTION update_stock_on_insert()
RETURNS TRIGGER AS $$
DECLARE -- variabili necessarie nel ciclo for
    var_nome_ingrediente text;
    total_usata numeric;
BEGIN
```

```
        --per ogni riga risultante dalla query assegno nome dell'ingrediente a
var_nome_ingrediente e la quantità totale a total usata
    FOR var_nome_ingrediente, total_usata IN (
        select nome_ingrediente, SUM(quantità_ingrediente) as total_usata-- somma
ingredienti tot usati
                from( -- somma ingredienti per portata, non serve group by perche se
ordino la stessa portata n volte ho solo un record con quantinta uguale a n
                        SELECT u.nome_ingrediente, u.nome_portata,
u.quantità_usata * cc.quantità as quantità_ingrediente
                        FROM "189632_simone".usati u
                        JOIN "189632_simone".contenuto_comanda cc ON
u.nome_portata = cc.nome_portata
                        WHERE cc.id_comanda = 29--calcolo la quantità totale di
ciascun ingrediente richiesto
                )
                GROUP BY nome_ingrediente
    )
        --aggiorno tabella stock
    LOOP
        UPDATE "189632_simone".stock
        SET quantità_stock = quantità_stock - total_usata
        WHERE nome_ingrediente = var_nome_ingrediente
                and id_magazzino = (select id_magazzino
                                                from "189632_simone".stock
                                                where nome_ingrediente =
var_nome_ingrediente
                                                limit 1);
    END LOOP;

    RETURN NEW;--restituisco la nuova riga inserita
END;
$$ LANGUAGE plpgsql;

-- condizione eseguo quando inserisco un nuovo record in contenuto comanda
CREATE OR REPLACE TRIGGER update_stock_after_insert
AFTER INSERT ON "189632_simone".contenuto_comanda
FOR EACH ROW
EXECUTE FUNCTION update_stock_on_insert();

-- conteollo se lo stock va negativo se si blocco la comanda
CREATE OR REPLACE FUNCTION check_stock_quantity()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.quantità_stock < 0 THEN
        RAISE EXCEPTION 'la quantità di Stock non può essere negativa per %',
NEW.nome_ingrediente;
```

```
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;-- utilizzato per delimitare una funzione di sql.
--condizione che si attiva il controllo prima dell'aggiornamento nella tabella stock
CREATE OR REPLACE TRIGGER prevent_negative_stock
BEFORE UPDATE ON "189632_simone".Stock
FOR EACH ROW
EXECUTE FUNCTION check_stock_quantity();
```

OPERAZIONI SUI DATI (DML)

```
--INSERISCO I DATI
INSERT INTO Dipendente (nome, cognome)
VALUES
('Mario', 'Rossi'),
('Luigi', 'Bianchi'),
('Giovanni', 'Verdi'),
('Sara', 'Neri'),
('Anna', 'Gialli'),
('Paolo', 'Blu'),
('Francesca', 'Marroni'),
('Simone', 'Grigi');



-- 3 tavoli da 2 posti fuori
-- 3 tavoli da 2 posti fuori
INSERT INTO Tavolo (capienza, tipo)
VALUES
  (2, 'fuori'),
  (2, 'fuori'),
  (2, 'fuori');

-- 4 tavoli da 4 posti fuori
INSERT INTO Tavolo (capienza, tipo)
VALUES
  (4, 'fuori'),
  (4, 'fuori'),
  (4, 'fuori'),
  (4, 'fuori'),
  (8, 'fuori'),
  (8, 'fuori');
```

```
INSERT INTO Tavolo (capienza, tipo)
VALUES
  (2, 'dentro'),
  (2, 'dentro'),
  (2, 'dentro'),
  (2, 'dentro'),
  (2, 'dentro'),
  (8, 'dentro'),
  (8, 'dentro');

-- 8 tavoli da 4 posti dentro
INSERT INTO Tavolo (capienza, tipo)
VALUES
  (4, 'dentro'),
  (4, 'dentro'),
  (4, 'dentro'),
  (4, 'dentro'),
  (4, 'dentro'),
  (4, 'dentro'),
  (4, 'dentro'),
  (4, 'dentro');


-- Inserimento dati completi nella tabella Prenotazione
INSERT INTO Prenotazioni (Data_ora, Id_tavolo, numero_persone, id_dipendente)
VALUES
  ('2024-01-15 19:00:00', 1, 2, 1),
  ('2024-01-22 20:00:00', 2, 2, 2),
  ('2024-02-05 18:30:00', 3, 2, 3),
  ('2024-02-12 19:30:00', 4, 2, 4),
  ('2024-02-19 20:00:00', 5, 2, 5),
  ('2024-03-04 19:00:00', 6, 2, 6),
  ('2024-03-11 18:30:00', 7, 2, 7),
  ('2024-03-18 20:00:00', 8, 2, 8),
  ('2024-04-01 19:30:00', 9, 2, 4),
  ('2024-04-08 19:00:00', 10, 2, 1),
  ('2024-04-15 20:00:00', 11, 2, 1),
  ('2024-04-22 18:30:00', 12, 2, 1),
  ('2024-05-01 19:00:00', 13, 2, 3),
  ('2024-05-10 20:00:00', 14, 2, 8),
  ('2024-05-15 18:30:00', 15, 2, 7),

  -- Additional reservations with unique times
  ('2024-02-15 19:45:00', 16, 2, 16),
```

```
  ('2024-03-05 20:15:00', 1, 4, 1),
  ('2024-04-02 18:45:00', 2, 2, 2),
  ('2024-03-07 19:15:00', 3, 2, 3),
  ('2024-04-10 20:30:00', 4, 4, 4),
  ('2024-02-20 19:50:00', 5, 4, 5),
  ('2024-03-15 18:40:00', 6, 2, 6),
  ('2024-04-05 20:10:00', 7, 4, 7),
  ('2024-03-10 19:25:00', 8, 2, 8),
  ('2024-05-05 18:55:00', 9, 2, 9),

  -- More than 3 reservations
  ('2024-04-10 19:35:00', 10, 4, 10),
  ('2024-05-01 20:05:00', 11, 4, 11),
  ('2024-04-15 18:50:00', 12, 2, 12),
  ('2024-05-05 19:20:00', 13, 2, 13),
  ('2024-03-20 20:25:00', 14, 4, 14),
  ('2024-04-25 19:40:00', 15, 4, 15),
  ('2024-04-12 18:35:00', 16, 2, 16),
  ('2024-05-02 20:20:00', 1, 2, 1),
  ('2024-04-20 19:05:00', 2, 4, 2),
  ('2024-05-10 18:55:00', 3, 4, 3);




INSERT INTO Portata (nome_portata, prezzo, sel, tipo_birra)
VALUES
  -- Cibi
  ('Classic Burger', 8.00, 'cibi', NULL),
  ('Veggie Dream', 7.50, 'cibi', NULL),
  ('Classic Dog', 6.50, 'cibi', NULL),
  ('Classiche', 4.00, 'cibi', NULL),

  -- Birre (alla spina)
  ('Pilsner Light 33cl', 4.50, 'birra', 'Lager chiara'),
  ('Pilsner Light 50cl', 5.50, 'birra', 'Lager chiara'),
  ('Stout Dream 33cl', 6.00, 'birra', 'Birra scura al caffè'),
  ('Stout Dream 50cl', 7.00, 'birra', 'Birra scura al caffè'),
  ('Weiss Sun 33cl', 5.50, 'birra', 'Birra di frumento'),
  ('Weiss Sun 50cl', 6.50, 'birra', 'Birra di frumento'),

  -- Bevande (alla spina)
  ('Coca-Cola 33cl', 2.50, 'bevande', NULL),
  ('Coca-Cola 50cl', 3.50, 'bevande', NULL),
  ('Fanta 33cl', 2.50, 'bevande', NULL),
  ('Fanta 50cl', 3.50, 'bevande', NULL),
```

```
  -- Acqua (non alla spina, capacità standard)
  ('Acqua Naturale 33cl', 1.50, 'bevande', NULL),
  ('Acqua Frizzante 33cl', 1.50, 'bevande', NULL);

INSERT INTO Ingrediente (nome_ingrediente)
VALUES
  -- Classic Burger ingredients
  ('Hamburger di manzo'),
  ('Insalata'),
  ('Pomodoro'),
  ('Cheddar'),
  ('Salsa speciale'),
  ('Burger vegetariano'),
  ('Wurstel di maiale'),
  ('Senape'),
  ('Ketchup');

INSERT INTO Ingrediente (nome_ingrediente)
VALUES
  -- Ingredienti per Birre alla spina
  ('Birra Pilsner alla spina'),
  ('Birra Stout alla spina'),
  ('Birra Weiss alla spina'),

  -- Bevande
  ('Coca-Cola alla spina'),
  ('Fanta alla spina'),

  ('Acqua naturale 33cl'),
  ('Acqua frizzante 33cl');

-- Inserisci nella tabella Usati
INSERT INTO Usati (nome_portata, nome_ingrediente, quantità_usata)
VALUES
  -- Birre alla spina
  ('Pilsner Light 33cl', 'Birra Pilsner alla spina', 0.33),
  ('Pilsner Light 50cl', 'Birra Pilsner alla spina', 0.50),

  ('Stout Dream 33cl', 'Birra Stout alla spina', 0.33),
  ('Stout Dream 50cl', 'Birra Stout alla spina', 0.50),

  ('Weiss Sun 33cl', 'Birra Weiss alla spina', 0.33),
  ('Weiss Sun 50cl', 'Birra Weiss alla spina', 0.50),

  -- Bevande alla spina
```

```
  ('Coca-Cola 33cl', 'Coca-Cola alla spina', 0.33),
  ('Coca-Cola 50cl', 'Coca-Cola alla spina', 0.50),

  ('Fanta 33cl', 'Fanta alla spina', 0.33),
  ('Fanta 50cl', 'Fanta alla spina', 0.50),
  ('Acqua Naturale 33cl', 'Acqua naturale 33cl', 1),
  ('Acqua Frizzante 33cl', 'Acqua frizzante 33cl', 1),
    -- Classic Burger ingredients
  ('Classic Burger', 'Hamburger di manzo', 0.2),
  ('Classic Burger', 'Insalata', 0.1),
  ('Classic Burger', 'Pomodoro', 0.1),
  ('Classic Burger', 'Cheddar', 0.05),
  ('Classic Burger', 'Salsa speciale', 0.02),

  -- Veggie Dream ingredients
  ('Veggie Dream', 'Burger vegetariano', 0.2),
  ('Veggie Dream', 'Insalata', 0.1),
  ('Veggie Dream', 'Pomodoro', 0.1),
  ('Veggie Dream', 'Cheddar', 0.05),
  ('Veggie Dream', 'Salsa speciale', 0.02),

  -- Classic Dog ingredients
  ('Classic Dog', 'Wurstel di maiale', 0.15),
  ('Classic Dog', 'Senape', 0.02),
  ('Classic Dog', 'Ketchup', 0.02),
  ('Classic Dog', 'Insalata', 0.05);


INSERT INTO Comande (Data_ora, id_tavolo, prezzo_totale, id_dipendente)
VALUES
  ('2024-01-15 19:00:00', 1, 21.00, 1),
  ('2024-01-22 20:00:00', 2, 12.00, 2),
  ('2024-02-05 18:30:00', 3, 19.00, 3),
  ('2024-02-12 19:30:00', 4, 42.00, 4),
  ('2024-02-19 20:00:00', 5, 22.00, 5),
  ('2024-03-04 19:00:00', 6, 38.00, 6),
  ('2024-03-11 18:30:00', 7, 39.00, 7),
  ('2024-03-18 20:00:00', 8, 38.00, 8),
  ('2024-04-01 19:30:00', 9, 12.00, 1),
  ('2024-04-08 19:00:00', 10, 42.00, 2),
  ('2024-04-15 20:00:00', 11, 38.00, 3),
  ('2024-04-22 18:30:00', 12, 19.00, 4),
  ('2024-05-01 19:00:00', 13, 13.00, 5),
  ('2024-05-10 20:00:00', 14, 38.00, 6),
  ('2024-05-15 18:30:00', 15, 12.00, 7),
  ('2024-02-15 19:45:00', 16, 13.00, 8),
```

```
  ('2024-03-05 20:15:00', 1, 38.00, 1),
  ('2024-04-02 18:45:00', 2, 12.00, 2),
  ('2024-03-07 19:15:00', 3, 13.00, 3),
  ('2024-04-10 20:30:00', 4, 38.00, 4),
  ('2024-02-20 19:50:00', 5, 19.00, 5),
  ('2024-03-15 18:40:00', 6, 38.00, 6),
  ('2024-04-05 20:10:00', 7, 38.00, 7),
  ('2024-03-10 19:25:00', 8, 12.00, 8),
  ('2024-05-05 18:55:00', 9, 12.00, 1),
  ('2024-04-10 19:35:00', 10, 13.00, 2),
  ('2024-05-01 20:05:00', 11, 38.00, 3),
  ('2024-04-15 18:50:00', 12, 12.00, 4);



-- Inserimento Contenuto_Comanda - Piatti principali
INSERT INTO Contenuto_Comanda (id_comanda, nome_portata, quantità)
VALUES
  (1, 'Classic Burger', 1),
  (2, 'Veggie Dream', 1),
  (3, 'Classic Dog', 1),
  (4, 'Classic Burger', 1),
  (5, 'Veggie Dream', 1),
  (6, 'Classic Burger', 2),
  (7, 'Classic Dog', 2),
  (8, 'Veggie Dream', 2),
  (9, 'Classic Burger', 1),
  (10, 'Classic Dog', 2),
  (11, 'Classic Burger', 2),
  (12, 'Veggie Dream', 1),
  (13, 'Classic Dog', 1),
  (14, 'Classic Burger', 2),
  (15, 'Veggie Dream', 1),
  (16, 'Classic Burger', 1),
  (17, 'Classic Dog', 2),
  (18, 'Veggie Dream', 1),
  (19, 'Classic Burger', 1),
  (20, 'Classic Dog', 2),
  (21, 'Classic Burger', 2),
  (22, 'Veggie Dream', 1),
  (23, 'Classic Dog', 2),
  (24, 'Classic Burger', 1),
  (25, 'Veggie Dream', 1),
  (26, 'Classic Burger', 2),
  (27, 'Classic Dog', 2);
-- Inserimento Contenuto_Comanda - Bevande
```

```
INSERT INTO Contenuto_Comanda (id_comanda, nome_portata, quantità)
VALUES
  (1, 'Pilsner Light 33cl', 2),
  (2, 'Acqua Naturale 33cl', 2),
  (3, 'Coca-Cola 33cl', 2),
  (4, 'Pilsner Light 50cl', 4),
  (5, 'Fanta 33cl', 4),
  (6, 'Pilsner Light 50cl', 4),
  (7, 'Coca-Cola 33cl', 4),
  (8, 'Fanta 33cl', 4),
  (9, 'Acqua Naturale 33cl', 2),
  (10, 'Pilsner Light 50cl', 4),
  (11, 'Coca-Cola 33cl', 4),
  (12, 'Pilsner Light 33cl', 2),
  (13, 'Fanta 33cl', 2),
  (14, 'Pilsner Light 50cl', 4),
  (15, 'Acqua Naturale 33cl', 2),
  (16, 'Coca-Cola 33cl', 2),
  (17, 'Pilsner Light 50cl', 4),
  (18, 'Acqua Frizzante 33cl', 2),
  (19, 'Fanta 33cl', 2),
  (20, 'Coca-Cola 50cl', 4),
  (21, 'Pilsner Light 50cl', 4),
  (22, 'Acqua Naturale 33cl', 2),
  (23, 'Fanta 50cl', 4),
  (24, 'Pilsner Light 33cl', 2),
  (25, 'Coca-Cola 33cl', 2),
  (26, 'Weiss Sun 50cl', 4),
  (27, 'Stout Dream 50cl', 4);


-- Insert Magazzino data
INSERT INTO Magazzino (piano)
VALUES
  (1),  -- First floor
  (2);  -- Second floor

-- Insert Stock data with fixed quantities
INSERT INTO Stock (nome_ingrediente, quantità_stock, id_magazzino)
VALUES
  -- Burger ingredients
  ('Hamburger di manzo', 50.0, 1),
  ('Insalata', 30.0, 1),
  ('Pomodoro', 40.0, 1),
  ('Cheddar', 25.0, 1),
  ('Salsa speciale', 20.0, 1),
```

('Burger vegetariano', 30.0, 1),
('Wurstel di maiale', 40.0, 1),
('Senape', 15.0, 1),
('Ketchup', 15.0, 1),

-- Beer ingredients
('Birra Pilsner alla spina', 200.0, 2),
('Birra Stout alla spina', 150.0, 2),
('Birra Weiss alla spina', 180.0, 2),

-- Soft drink ingredients
('Coca-Cola alla spina', 250.0, 2),
('Fanta alla spina', 200.0, 2),

-- Water
('Acqua naturale 33cl', 300.0, 1),
('Acqua frizzante 33cl', 300.0, 1);

QUERY SIGNIFICATIVE:

/*3 portate più prese*/

```
SELECT cc.nome_portata, SUM(cc.quantità) as quantità_venduta, SUM(p.prezzo *
cc.quantità) as entrate_totali
FROM "189632_simone".Contenuto_Comanda cc
INNER JOIN "189632_simone".Portata p ON cc.nome_portata = p.nome_portata
GROUP BY cc.nome_portata
ORDER BY quantità_venduta DESC LIMIT 3;
```

/*num prenotazioni, comande per mese con prezzo medio ordine*/

```
SELECT
    EXTRACT(MONTH FROM p.data_ora) AS mese,
    COUNT(DISTINCT p.Id_prenotazione) AS prenotazioni_totali,
    COUNT(DISTINCT c.id_comanda) AS ordini_totali,
    ROUND(AVG(c.prezzo_totale), 2) AS prezzo_medio_ordine
FROM
         "189632_simone".Prenotazioni p
LEFT JOIN
    "189632_simone".Comande c ON p.data_ora = c.data_ora AND p.Id_tavolo =
c.id_tavolo
GROUP BY
    EXTRACT(MONTH FROM p.data_ora)
ORDER BY
    mese;
```

/*Calcolare il ricavo giornaliero e piatti, bevande, birre prese per giornali*/

```
SELECT
    DATE(c.Data_ora) AS giorno,--raggruppo solo per data senza orario
    SUM(c.prezzo_totale) AS ricavo_totale,
    SUM((SELECT SUM(cc2.quantità)
        FROM "189632_simone".Contenuto_Comanda cc2
        JOIN "189632_simone".Portata p2 ON cc2.nome_portata = p2.nome_portata
        WHERE cc2.id_comanda = c.id_comanda AND p2.sel = 'cibi')) AS
numero_piatti,
    SUM((SELECT SUM(cc2.quantità)
        FROM "189632_simone".Contenuto_Comanda cc2
        JOIN "189632_simone".Portata p2 ON cc2.nome_portata = p2.nome_portata
        WHERE cc2.id_comanda = c.id_comanda AND p2.sel = 'bevande')) AS
numero_bevande,
```

```
    SUM((SELECT SUM(cc2.quantità)
        FROM "189632_simone".Contenuto_Comanda cc2
        JOIN "189632_simone".Portata p2 ON cc2.nome_portata = p2.nome_portata
        WHERE cc2.id_comanda = c.id_comanda AND p2.sel = 'birra')) AS
numero_birre
FROM
    "189632_simone".Comande c
GROUP BY
    DATE(c.Data_ora)
ORDER BY
    giorno DESC;

/*Calcolare il totale delle vendite per ogni tipo di birra, includendo entrambi i formati
(33cl e 50cl):*/

SELECT p.tipo_birra,
       SUM(c.quantità) as totale_birre_vendute,
       SUM(c.quantità * p.prezzo) as ricavo_totale
FROM "189632_simone".Contenuto_Comanda c
JOIN "189632_simone".Portata p ON c.nome_portata = p.nome_portata
WHERE p.sel = 'birra'
GROUP BY p.tipo_birra
HAVING p.tipo_birra IS NOT NULL
ORDER BY ricavo_totale desc;


/*TRIGGER STOCK*/

-- creo una funzione trigger che eseguo quando inserisco un nuovo record in
contenuto comanda
CREATE OR REPLACE FUNCTION update_stock_on_insert()
RETURNS TRIGGER AS $$
DECLARE -- variabili necessarie nel ciclo for
    var_nome_ingrediente text;
    total_usata numeric;
BEGIN
        --per ogni riga risultante dalla query assegno nome dell'ingrediente a
var_nome_ingrediente e la quantità totale a total usata
    FOR var_nome_ingrediente, total_usata IN (
       select nome_ingrediente, SUM(quantità_ingrediente) as total_usata-- somma
ingredienti tot usati
                from( -- somma ingredienti per portata, non serve group by perche se
ordino la stessa portata n volte ho solo un record con quantinta uguale a n
                                SELECT u.nome_ingrediente, u.nome_portata,
u.quantità_usata * cc.quantità as quantità_ingrediente
                                FROM "189632_simone".usati u
```

```
                        JOIN "189632_simone".contenuto_comanda cc ON
u.nome_portata = cc.nome_portata
                        WHERE cc.id_comanda = 29--calcolo la quantità totale di
ciascun ingrediente richiesto
                )
                GROUP BY nome_ingrediente
    )
        --aggiorno tabella stock
    LOOP
        UPDATE "189632_simone".stock
        SET quantità_stock = quantità_stock - total_usata
        WHERE nome_ingrediente = var_nome_ingrediente
                and id_magazzino = (select id_magazzino
                                    from "189632_simone".stock
                                    where nome_ingrediente =
var_nome_ingrediente
                                    limit 1);
    END LOOP;

    RETURN NEW;--restituisco la nuova riga inserita
END;
$$ LANGUAGE plpgsql;

-- condizione eseguo quando inserisco un nuovo record in contenuto comanda
CREATE OR REPLACE TRIGGER update_stock_after_insert
AFTER INSERT ON "189632_simone".contenuto_comanda
FOR EACH ROW
EXECUTE FUNCTION update_stock_on_insert();

-- conteollo se lo stock va negativo se si blocco la comanda
CREATE OR REPLACE FUNCTION check_stock_quantity()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.quantità_stock < 0 THEN
        RAISE EXCEPTION 'la quantità di Stock non può essere negativa per %',
NEW.nome_ingrediente;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;-- utilizzato per delimitare una funzione di sql.
--condizione che si attiva il controllo prima dell'aggiornamento nella tabella stock
CREATE OR REPLACE TRIGGER prevent_negative_stock
BEFORE UPDATE ON "189632_simone".Stock
FOR EACH ROW
EXECUTE FUNCTION check_stock_quantity();
```

WEB APPLICATION

L'applicazione web sarà sviluppata utilizzando il linguaggio Python.
Grazie ai moduli Flask e psycopg2, sarà possibile interagire con il database attraverso un DBMS (Database Management System).
Questa interazione consentirà di eseguire operazioni sul database, sul programma python, tramite il linguaggio SQL, come l'inserimento, la modifica, l'eliminazione e la visualizzazione dei dati.
Il database sarà ospitato sullo stesso computer in cui viene eseguita l'applicazione web, accessibile tramite l'indirizzo localhost o 127.0.0.1:5432. L'applicazione web permette di gestire i dati presenti nel database attraverso un'interfaccia creata con HTML. Per migliorare l'aspetto grafico del sito, saranno utilizzati fogli di stile CSS.