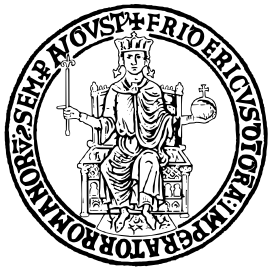
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Triennale in Informatica

UninaFoodLab Database

Anno Accademico 2024–2025

Indice

Capitolo 1

Descrizione del progetto

1.1 Descrizione sintetica del problema

Si vuole realizzare una base di dati relazionale per la gestione di corsi di cucina tematici. Gli chef possono registrare corsi su specifici argomenti, specificando una data di inizio e una frequenza delle sessioni. Ogni corso è articolato in più sessioni, che possono essere di due tipi: online, oppure in presenza, in cui gli utenti svolgono attività pratiche. Gli utenti possono iscriversi a più corsi e, nel caso delle sessioni pratiche, devono fornire una adesione esplicita per confermare la loro partecipazione. Ogni sessione pratica prevede la preparazione di una o più ricette, ciascuna delle quali richiede una specifica lista di ingredienti. Le adesioni vengono utilizzate per pianificare correttamente la quantità di ingredienti necessari, evitando così sprechi alimentari. Il sistema prevede la possibilità di notificare gli utenti iscritti ad un corso in caso di variazioni di programma.

Capitolo 2

Progettazione Concettuale

2.1 Introduzione

Una volta definito e analizzato il problema possiamo procedere con la progettazione concettuale della base di dati. In questa fase, l'obiettivo è quello di creare un modello concettuale che rappresenti le entità, le relazioni e le caratteristiche del dominio. Tale schema viene rappresentato mediante un diagramma delle classi UML e un diagramma Entità-Relazioni (ER).

2.2 Diagrammi UML e ER

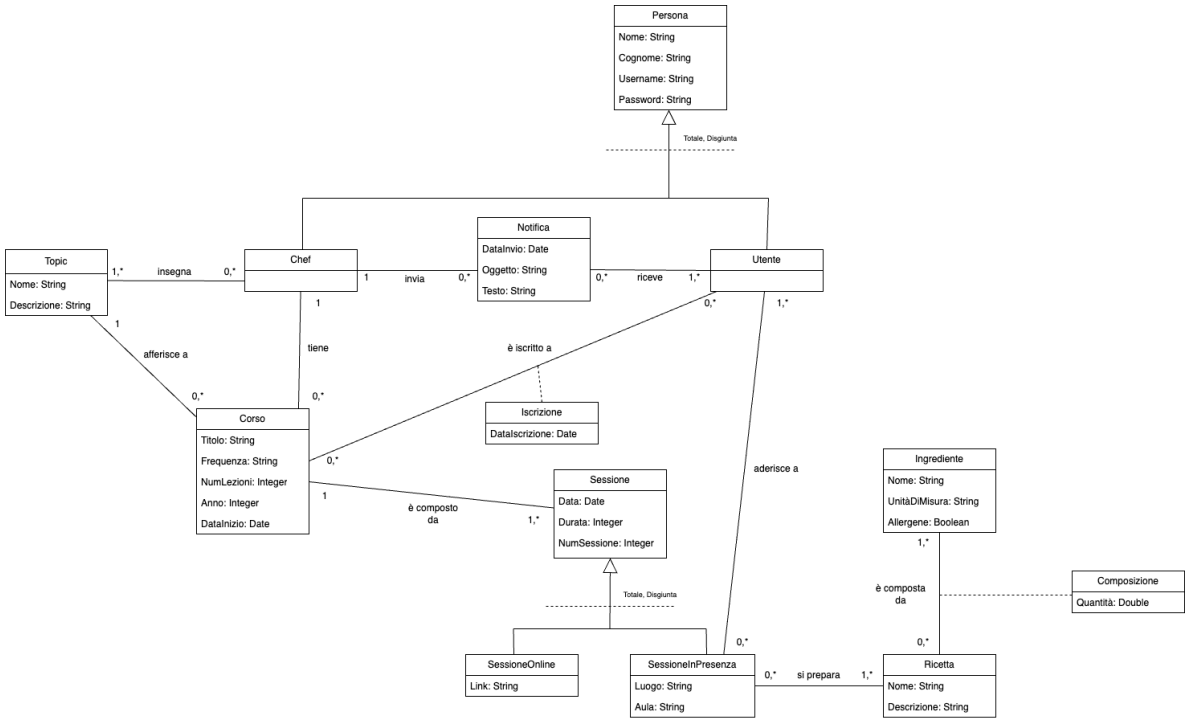


Figura 2.1: Diagramma delle classi UML

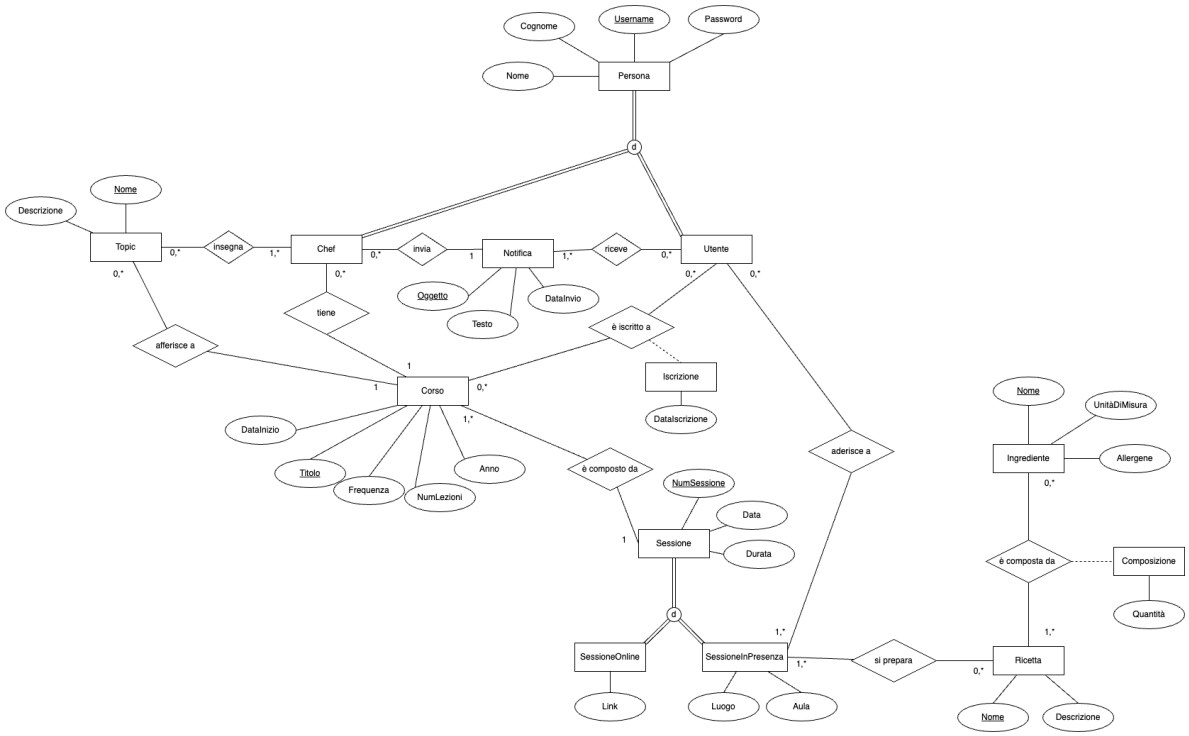


Figura 2.2: Diagramma Entità-Relazioni

2.3 Ristrutturazione del modello concettuale

Un'operazione fondamentale per il passaggio dal modello concettuale a quello logico è la ristrutturazione. Questo processo implica la trasformazione delle entità e delle relazioni del modello concettuale, al fine di renderne più agevole l'implementazione in un database relazionale. Durante questa fase, si identificano le chiavi primarie e le chiavi esterne, si rimuovono gli attributi multipli e le gerarchie di specializzazione, si normalizzano le tabelle per ridurre la ridondanza dei dati.

2.4 Identificazione delle chiavi primarie

Le chiavi primarie sono gli attributi che identificano in modo univoco ogni record in una tabella. Nella nostra ristrutturazione abbiamo fatto uso di chiavi surrogate, ossia chiavi artificiali create per garantire l'unicità dei record nel caso in cui non esistano chiavi naturali adatte. Queste chiavi non hanno un significato intrinseco, ma sono utilizzate per facilitare le operazioni sulle tabelle.

2.5 Identificazione delle chiavi esterne

Le chiavi esterne sono gli attributi che collegano le tabelle tra loro, rappresentando le relazioni tra le entità. Il loro scopo è quello di assicurare che i dati siano coerenti e che le relazioni siano mantenute.

2.6 Rimozione degli attributi multipli

Non sono presenti attributi multipli nel modello concettuale.

2.7 Rimozione delle gerarchie di specializzazione

Sono presenti due gerarchie di specializzazione:

- 'Persona' si specializza in "Chef" e "Utente"
- 'Sessione' si specializza in "Sessione Online" e "Sessione in Presenza"

2.8 Analisi delle ridondanze

Non sono presenti ridondanze significative nel modello concettuale.

2.9 Class Diagram ristrutturato

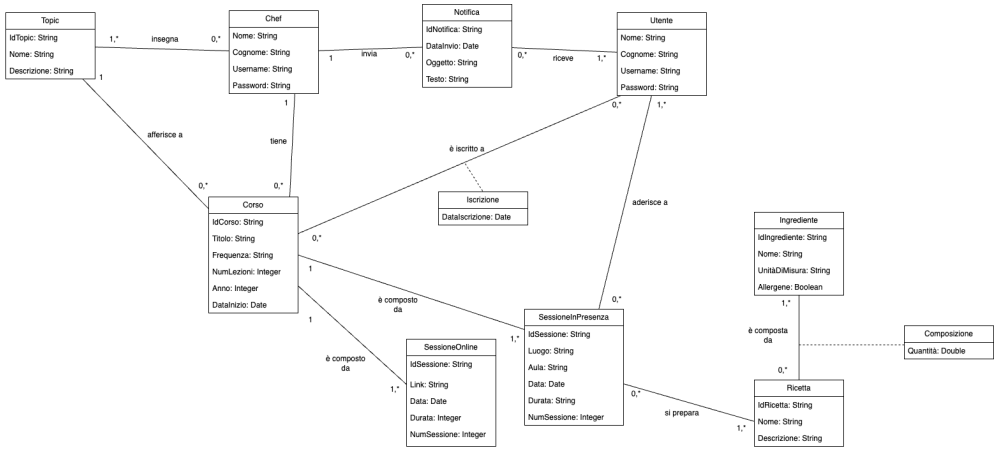


Figura 2.3: Diagramma UML ristrutturato

2.10 Dizionario delle classi

2.11 Dizionario delle associazioni

2.12 Dizionario dei vincoli

Classe	Descrizione	Attributi
Chef	Rappresenta gli chef della piattaforma, possono tenere corsi sugli argomenti di cui sono esperti	Username (string), Nome (string), Cognome (string), Password (string)
Utente	Rappresenta gli utenti della piattaforma, possono iscriversi ai corsi	Username (string), Nome (string), Cognome (string), Password (string)
Topic	Rappresenta gli argomenti dei corsi e le conoscenze degli chef	IdTopic (string), Nome (string), Descrizione (string)
Corso	Rappresenta i corsi tenuti dagli chef e ai quali gli utenti possono iscriversi	IdCorso (string), Titolo (string), Frequenza (string), NumLezioni (integer), Anno(integer), DataInizio (date)
Iscrizione	Rappresenta l'iscrizione di un utente a un corso	DataIscrizione (date)
Notifica	Rappresenta le notifiche che gli chef possono inviare agli utenti	IdNotifica (string), DataInvio (date), Oggetto (string), Testo (string)
SessioneOnline	Rappresenta le sessioni online dei corsi	IdSessione (string), Link (string), Data (date), Durata (integer), NumSessione (integer)
SessioneInPresenza	Rappresenta le sessioni in presenza dei corsi	IdSessione (string), Luogo (string), Aula (string), Data (date), Durata (integer), NumSessione (integer)
Ricetta	Rappresenta le ricette che possono essere preparate durante le sessioni in presenza	IdRicetta (string), Nome (string), Descrizione (string)
Ingrediente	Rappresenta gli ingredienti utilizzabili per le ricette	IdIngrediente (string), Nome (string), UnitàDiMisura (string), Allergene (boolean)
Composizione	Rappresenta le quantità di ogni ingrediente in una ricetta	Quantità(float)

Capitolo 3

Progettazione Logica

3.1 Schema Logico

TOPIC(IdTopic, Nome, Descrizione)

INSEGNA(IdTopic, UsernameChef)

CORSO(IdCorso, Titolo, Frequenza, NumLezioni, DataInizio, IdTopic, UsernameChef)

CHEF(Username, Nome, Cognome, Password)

UTENTE(Username, Nome, Cognome, Password)

NOTIFICA(IdNotifica, DataInvio, Oggetto, Testo, UsernameChef)

RICEVE(IdNotifica, UsernameUtente)

ISCRIZIONE(UsernameUtente, IdCorso, DataIscrizione)

SESSIONE ONLINE(IdSessione, Link, Data, Durata, NumSessione, IdCorso)

SESSIONE IN PRESENZA(IdSessione, Luogo, Aula, Data, Durata, NumSessione, IdCorso)

ADESIONE(UsernameUtente, IdSessionePresenza)

RICETTA(IdRicetta, Nome, Descrizione)

PREPARA(IdSessionePresenza, IdRicetta)

INGREDIENTE(IdIngrediente, Nome, UnitàDiMisura, Allergene)

COMPOSIZIONE(IdRicetta, IdIngrediente, Quantità)

Capitolo 4

Progettazione Fisica

La base di dati è stata implementata utilizzando PostgreSQL, un sistema di gestione di basi di dati relazionali open source. La progettazione fisica della base di dati è stata effettuata seguendo le linee guida del modello relazionale, definendo le tabelle, le chiavi primarie e le chiavi esterne per garantire l'integrità referenziale tra le entità. Qui di seguito sono riportate le definizioni delle tabelle utilizzate nel database ed eventuali triggers, procedures, functions o vincoli implementati.

4.1 Definizione tabelle

Seguono le definizioni delle tabelle estratte dallo script di creazione del database.

4.1.1 Definizione della tabella TOPIC

```
CREATE TABLE Topic (
    idTopic SERIAL PRIMARY KEY,
    Nome VARCHAR(1000) NOT NULL UNIQUE,
    Descrizione VARCHAR(1000),

    CONSTRAINT CK_Topic_Nome CHECK (Nome IN ('Cucina Italiana', 'Pizza
        Napoletana', 'Cucina Cinese',
    'Cucina Messicana', 'Cucina Giapponese', 'Cucina Indiana', 'Cucina
        Francese',
    'Cucina Spagnola', 'Cucina Americana', 'Cucina Mediterranea',
        'Pasticceria', 'Panificazione')) -- controllo che il nome del topic
        sia un valore valido
);
```

Listing 4.1: Creazione tabella Topic

La tabella **Topic** rappresenta le categorie tematiche dei corsi di cucina. Ogni topic ha un identificatore univoco generato automaticamente e un nome che deve essere scelto tra i valori predefiniti specificati nel constraint CHECK.

4.1.2 Definizione della tabella CHEF

```
CREATE TABLE Chef (
    Nome VARCHAR(50) NOT NULL,
    Cognome VARCHAR(50) NOT NULL,
    Username VARCHAR(100) PRIMARY KEY,
    Password VARCHAR(100) NOT NULL
);
```

Listing 4.2: Creazione tabella Chef

La tabella **Chef** rappresenta gli chef che possono tenere corsi di cucina. Ogni chef ha un identificatore univoco (Username) e le informazioni personali come nome, cognome e password.

4.1.3 Definizione della tabella UTENTE

```
CREATE TABLE Utente (
    Nome VARCHAR(50) NOT NULL,
    Cognome VARCHAR(50) NOT NULL,
    Username VARCHAR(100) PRIMARY KEY,
    Password VARCHAR(100) NOT NULL
)
```

Listing 4.3: Creazione tabella Utente

La tabella **Utente** rappresenta gli utenti che possono iscriversi ai corsi di cucina. Ogni utente ha un identificatore univoco (Username) e le informazioni personali come nome, cognome e password.

4.1.4 Definizione della tabella CORSO

```sql
CREATE TABLE Corso (
    idCorso SERIAL PRIMARY KEY,
    Titolo VARCHAR(50) NOT NULL,
    Frequenza VARCHAR(50) DEFAULT 'Settimanale',
    NumLezioni Integer NOT NULL,
    Anno Integer NOT NULL,
    DataInizio Date NOT NULL,
    IdTopic INTEGER NOT NULL,
    UsernameChef VARCHAR(100) NOT NULL,

    CONSTRAINT FK_Corso_Topic FOREIGN KEY (IdTopic) REFERENCES
        Topic(idTopic), -- foreign key verso Topic
    CONSTRAINT FK_Corso_Chef FOREIGN KEY (UsernameChef) REFERENCES
        Chef(Username) ON DELETE CASCADE, -- foreign key verso Chef, su
        eliminazione di un chef elimina tutti i suoi corsi
    CONSTRAINT CK_Frequenza CHECK (Frequenza IN ('Settimanale', 'Ogni
        giorno', 'Ogni due giorni', 'Ogni tre giorni')), -- controllo che la
        frequenza del corso sia un valore valido
    CONSTRAINT CK_NumLezioni CHECK (NumLezioni BETWEEN 1 AND 100), --
        controllo che il numero di lezioni sia compreso tra 1 e 100
    CONSTRAINT CK_Anno CHECK (Anno = EXTRACT(YEAR FROM DataInizio)) --
        controllo che l'anno sia uguale all'anno di inizio
)
```

Listing 4.4: Creazione tabella Corso

La tabella **Corso** rappresenta i corsi di cucina. Ogni corso ha un identificatore univoco (IdCorso), un titolo, una frequenza, il numero di lezioni, una data di inizio e un riferimento al topic e allo chef che lo insegna.

4.1.5 Definizione della tabella SESSIONEINPRESENZA

```sql
CREATE TABLE SessioneInPresenza (
    IdSessione SERIAL PRIMARY KEY,
    Luogo VARCHAR(50) NOT NULL,
    Aula VARCHAR(50) NOT NULL,
    Data TIMESTAMP NOT NULL,
    Durata Integer NOT NULL,
    NumSessione Integer NOT NULL,
    IdCorso INTEGER NOT NULL,

    CONSTRAINT FK_SessioneInPresenza_Corso FOREIGN KEY (IdCorso) REFERENCES
        Corso(idCorso) ON DELETE CASCADE, -- foreign key verso Corso, su
        eliminazione di un corso elimina tutte le sue sessioni
    CONSTRAINT CK_Luogo CHECK (Luogo IN ('MonteSantangelo', 'ViaClaudio',
        'PiazzaleTecchio', 'PortaDiMassa', 'ViaMarina')), -- controllo che il
        luogo sia un valore valido
    CONSTRAINT CK_Durata CHECK (Durata >= 60 AND Durata <= 180) -- controllo
        che la durata sia compresa tra 60 e 180 minuti
)
```

Listing 4.5: Creazione tabella Sessione in Presenza

La tabella **SessioneInPresenza** rappresenta le sessioni in presenza dei corsi di cucina. Ogni sessione ha un identificatore univoco (IdSessione), un luogo, un'aula, una data, una durata, un numero di sessione e un riferimento al corso a cui appartiene.

4.1.6 Definizione della tabella SESSIONEONLINE

```
CREATE TABLE SessioneOnline (
    IdSessione SERIAL PRIMARY KEY,
    Link VARCHAR(1000) NOT NULL,
    Data TIMESTAMP NOT NULL,
    Durata Integer NOT NULL,
    NumSessione Integer NOT NULL,
    IdCorso INTEGER NOT NULL,

    CONSTRAINT FK_SessioneOnline_Corso FOREIGN KEY (IdCorso) REFERENCES
        Corso(idCorso) ON DELETE CASCADE, -- foreign key verso Corso, su
        eliminazione di un corso elimina tutte le sue sessioni online
    CONSTRAINT CK_Link CHECK (Link LIKE 'https://%'), -- controllo che il
        link abbia un formato valido
    CONSTRAINT CK_Durata CHECK (Durata >= 60 AND Durata <= 180) -- controllo
        che la durata sia compresa tra 60 e 180 minuti
)
```

Listing 4.6: Creazione tabella Sessione Online

La tabella **SessioneOnline** rappresenta le sessioni online dei corsi di cucina. Ogni sessione ha un identificatore univoco (IdSessione), un link, una data, una durata, un numero di sessione e un riferimento al corso a cui appartiene.

4.1.7 Definizione della tabella ADESIONE

```
CREATE TABLE Adesione (
    UsernameUtente VARCHAR(100) NOT NULL,
    IdSessionePresenza INTEGER NOT NULL,

    CONSTRAINT PK_Adesione PRIMARY KEY (UsernameUtente, IdSessionePresenza),
        -- chiave primaria composta
    CONSTRAINT FK_Adesione_Utente FOREIGN KEY (UsernameUtente) REFERENCES
        Utente(Username) ON DELETE CASCADE, -- foreign key verso Utente, su
        eliminazione di un utente elimina tutte le sue adesioni
    CONSTRAINT FK_Adesione_SessionePresenza FOREIGN KEY (IdSessionePresenza)
        REFERENCES SessioneInPresenza(IdSessione) ON DELETE CASCADE --
        foreign key verso SessioneInPresenza, su eliminazione di una sessione
        elimina tutte le adesioni ad essa collegate
)
```

Listing 4.7: Creazione tabella Adesione

La tabella **Adesione** rappresenta le adesioni degli utenti alle sessioni in presenza dei corsi di cucina. Ogni adesione ha un riferimento all'utente e alla sessione in presenza a cui si riferisce. Questa tabella è necessaria per calcolare il numero di ingredienti necessari per ogni sessione in presenza.

4.1.8 Definizione della tabella PREPARA

```
CREATE TABLE Prepara (
    IdSessionePresenza INTEGER NOT NULL,
    IdRicetta INTEGER NOT NULL,

    CONSTRAINT PK_Prepara PRIMARY KEY (IdSessionePresenza, IdRicetta), --
        chiave primaria composta
    CONSTRAINT FK_Prepara_SessionePresenza FOREIGN KEY (IdSessionePresenza)
        REFERENCES SessioneInPresenza(IdSessione) ON DELETE CASCADE, --
        foreign key verso SessioneInPresenza, su eliminazione di una sessione
        elimina tutte le preparazioni ad essa collegate
    CONSTRAINT FK_Prepara_Ricetta FOREIGN KEY (IdRicetta) REFERENCES
        Ricetta(IdRicetta) --foreign key verso Ricetta
)
```

Listing 4.8: Creazione tabella Prepara

La tabella **Prepara** rappresenta la relazione tra le sessioni in presenza e le ricette. Ogni record in questa tabella associa una sessione in presenza a una ricetta specifica.

4.1.9 Definizione della tabella RICETTA

```
CREATE TABLE Ricetta (
    IdRicetta SERIAL PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL UNIQUE,
    Descrizione VARCHAR(1000) NOT NULL
)
```

Listing 4.9: Creazione tabella Ricetta

La tabella **Ricetta** rappresenta le ricette associate alle sessioni in presenza dei corsi di cucina. Ogni ricetta ha un identificatore univoco (IdRicetta), un nome e una descrizione.

4.1.10 Definizione della tabella INGREDIENTE

```
CREATE TABLE Ingrediente (
    IdIngrediente SERIAL PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL UNIQUE,
    Allergene BOOLEAN NOT NULL,
    UnitàMisura VARCHAR(50) NOT NULL
)
```

Listing 4.10: Creazione tabella Ingrediente

La tabella **Ingrediente** rappresenta gli ingredienti utilizzati nelle ricette. Ogni ingrediente ha un identificatore univoco (IdIngrediente), un nome, un'unità di misura e un flag che indica se è un allergene.

4.1.11 Definizione della tabella COMPOSIZIONE

```
CREATE TABLE Composizione (
    IdIngrediente INTEGER NOT NULL,
    IdRicetta INTEGER NOT NULL,
    Quantità FLOAT NOT NULL,

    CONSTRAINT PK_Composizione PRIMARY KEY (IdIngrediente, IdRicetta), --
        chiave primaria composta
    CONSTRAINT FK_Composizione_Ingrediente FOREIGN KEY (IdIngrediente)
        REFERENCES Ingrediente(IdIngrediente), -- foreign key verso
        Ingrediente
    CONSTRAINT FK_Composizione_Ricetta FOREIGN KEY (IdRicetta) REFERENCES
        Ricetta(IdRicetta), -- foreign key verso Ricetta
    CONSTRAINT CK_Quantità_Positiva CHECK (Quantità > 0) -- controllo
        quantità ingrediente positiva
)
```

Listing 4.11: Creazione tabella Composizione

La tabella **Composizione** rappresenta la relazione tra le ricette e gli ingredienti. Ogni record in questa tabella associa una ricetta a varo ingredienti che la compongono, specificando la quantità necessaria per ciascuno di essi.

4.1.12 Definizione della tabella NOTIFICA

```
CREATE TABLE Notifica (
    IdNotifica SERIAL PRIMARY KEY,
    UsernameChef VARCHAR(100) NOT NULL,
    Oggetto VARCHAR(1000) NOT NULL,
    Testo VARCHAR(1000),
    DataInvio TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT FK_notifica_chef FOREIGN KEY (UsernameChef) REFERENCES
        Chef(Username) ON DELETE CASCADE -- foreign key verso Chef, su
        eliminazione di un chef elimina tutte le notifiche da lui inviate

)
```

Listing 4.12: Creazione tabella Notifica

La tabella **Notifica** rappresenta le notifiche inviate dagli chef agli utenti iscritti ai corsi. Ogni notifica ha un identificatore univoco (IdNotifica), una data di invio, un oggetto, un testo e un riferimento allo chef che l'ha inviata.

4.1.13 Definizione della tabella RICEVE

```
CREATE TABLE Riceve (
    IdNotifica INTEGER NOT NULL,
    UsernameUtente VARCHAR(100) NOT NULL,

    CONSTRAINT PK_Riceve PRIMARY KEY (IdNotifica, UsernameUtente), -- chiave primaria composta
    CONSTRAINT FK_Riceve_Notifica FOREIGN KEY (IdNotifica) REFERENCES Notifica(IdNotifica) ON DELETE CASCADE, -- foreign key verso Notifica, su eliminazione di una notifica elimina tutte le ricezioni ad essa collegate
    CONSTRAINT FK_Riceve_Utente FOREIGN KEY (UsernameUtente) REFERENCES Utente(Username) ON DELETE CASCADE -- foreign key verso Utente, su eliminazione di un utente elimina tutte le sue ricezioni
)
```

Listing 4.13: Creazione tabella Riceve

La tabella **Riceve** rappresenta la relazione tra le notifiche e gli utenti che le ricevono. Ogni record in questa tabella associa una notifica a un utente specifico, consentendo di tenere traccia delle notifiche inviate a ciascun utente.

4.1.14 Definizione della tabella INSEGNA

```
CREATE TABLE Insegna (
    IdTopic INTEGER NOT NULL,
    UsernameChef VARCHAR(1000) NOT NULL,

    CONSTRAINT PK_Insegna PRIMARY KEY (IdTopic, UsernameChef), -- chiave primaria composta
    CONSTRAINT FK_Insegna_Topic FOREIGN KEY (IdTopic) REFERENCES Topic(idTopic), -- foreign key verso Topic
    CONSTRAINT FK_Insegna_Chef FOREIGN KEY (UsernameChef) REFERENCES Chef(Username) ON DELETE CASCADE -- foreign key verso Chef, su eliminazione di un chef elimina tutte le relative righe in Insegna
)
```

Listing 4.14: Creazione tabella Insegna

La tabella **Insegna** rappresenta la relazione tra i topic e gli chef che li insegnano. Ogni record in questa tabella associa un topic a uno chef specifico, consentendo di tenere traccia dei topic insegnati da ciascun chef.

4.1.15 Definizione della tabella ISCRIZIONE

```sql
CREATE TABLE Iscrizione (
    IdCorso INTEGER NOT NULL,
    UsernameUtente VARCHAR(100) NOT NULL,
    DataIscrizione TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT PK_Iscrizione PRIMARY KEY (IdCorso, UsernameUtente), -- chiave primaria composta
    CONSTRAINT FK_Iscrizione_Corso FOREIGN KEY (IdCorso) REFERENCES Corso(idCorso) ON DELETE CASCADE, -- foreign key verso Corso, su eliminazione di un corso elimina tutte le sue iscrizioni
    CONSTRAINT FK_Iscrizione_Utente FOREIGN KEY (UsernameUtente) REFERENCES Utente(Username) ON DELETE CASCADE -- foreign key verso Utente, su eliminazione di un utente elimina tutte le sue iscrizioni
)
```

Listing 4.15: Creazione tabella Iscrizione

La tabella **Iscrizione** rappresenta le iscrizioni degli utenti ai corsi di cucina. Ogni iscrizione ha un riferimento all'utente e al corso a cui si riferisce, insieme alla data di iscrizione. Questa tabella è necessaria per tenere traccia degli utenti iscritti a ciascun corso e per gestire le notifiche relative ai corsi.

4.2 Triggers, Procedures, Functions e Vincoli

Durante la progettazione fisica della base di dati, sono stati implementati alcuni triggers, procedures, functions e vincoli per garantire l'integrità dei dati e automatizzare alcune operazioni. Di seguito sono riportati alcuni esempi:

4.2.1 Funzione per calcolare il numero di adesioni

```sql
CREATE OR REPLACE FUNCTION CalcoloNumeroAdesioni(idSessione INTEGER) -- funzione per calcolare il numero di adesioni a una sessione in presenza
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE
    numeroAdesioni INTEGER;
BEGIN
    SELECT COUNT(*) INTO numeroAdesioni
    FROM Adesione AS a
    WHERE a.idSessione = idSessione;
    IF numeroAdesioni = 0 THEN
        RAISE EXCEPTION 'Nessuna adesione trovata per la sessione con ID %', idSessione;
    END IF;
    RETURN numeroAdesioni;
END;
$$;
```

Listing 4.16: Funzione per calcolare il numero di adesioni

Questa funzione calcola il numero di adesioni per una sessione in presenza. Necessaria per pianificare correttamente la quantità di ingredienti necessari per ogni sessione in presenza.

4.2.2 Trigger per controllare che l'anno del corso sia valido

```
CREATE OR REPLACE FUNCTION ControlloAnnoCorso()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    -- Controlla se l'anno del corso è compreso tra l'anno corrente e il
        prossimo
    IF NEW.Anno < EXTRACT(YEAR FROM CURRENT_DATE) OR
       NEW.Anno > EXTRACT(YEAR FROM CURRENT_DATE) +1 THEN
        RAISE EXCEPTION 'L''anno del corso deve essere compreso tra l''anno
            corrente e il prossimo.';
    END IF;
    RETURN NEW;
END;
$$;


CREATE TRIGGER TriggerAnnoCorso
BEFORE INSERT OR UPDATE ON Corso
FOR EACH ROW
EXECUTE FUNCTION ControlloAnnoCorso();
```

Listing 4.17: Trigger per controllare l'anno del corso

Questo trigger viene attivato prima dell'inserimento o aggiornamento di un nuovo corso e verifica che l'anno del corso sia valido. Se l'anno non è valido, il trigger genera un errore e impedisce l'inserimento del corso. Questo è utile per garantire che i corsi siano pianificati correttamente e che le date siano coerenti con l'anno accademico corrente o successivo.

4.2.3 Trigger per controllare che l'aula della sessione in presenza non sia occupata

```
CREATE OR REPLACE FUNCTION ControlloAulaOccupata()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    -- Controlla se esiste già una sessione in presenza nello stesso luogo e
        aula che si sovrappone con la nuova sessione
    IF EXISTS (
            SELECT 1
            FROM SessioneInPresenza AS sip
            WHERE sip.Luogo = NEW.Luogo
            AND sip.Aula = NEW.Aula
                -- La nuova sessione inizia durante una sessione esistente
            AND (NEW.Data >= sip.Data AND NEW.Data < sip.Data + (sip.Durata
                || ' minutes')::INTERVAL)
                -- La nuova sessione finisce durante una sessione esistente
                OR (NEW.Data + (NEW.Durata || ' minutes')::INTERVAL >
                    sip.Data AND NEW.Data + (NEW.Durata || '
                    minutes')::INTERVAL <= sip.Data + (sip.Durata || '
                    minutes')::INTERVAL)
                -- La nuova sessione contiene completamente una sessione
                    esistente
                OR (NEW.Data <= sip.Data AND NEW.Data + (NEW.Durata || '
                    minutes')::INTERVAL >= sip.Data + (sip.Durata || '
                    minutes')::INTERVAL)
    ) THEN
        RAISE EXCEPTION 'L''aula % nel luogo % è occupata in data %',
            NEW.Aula, NEW.Luogo, NEW.Data;
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER Tr_Insert_Aula_Sessione_Presenza
BEFORE INSERT OR UPDATE ON SessioneInPresenza
FOR EACH ROW
EXECUTE FUNCTION ControlloAulaOccupata();
```

Listing 4.18: Trigger per controllare l'aula della sessione in presenza

Questo trigger viene attivato prima dell'inserimento o aggiornamento di una sessione in presenza e verifica che l'aula specificata non sia già occupata da un'altra sessione in presenza nello stesso orario. Se l'aula è occupata, il trigger genera un errore e impedisce l'inserimento della sessione. Questo è utile per evitare conflitti di programmazione e garantire che le sessioni in presenza possano essere svolte senza problemi logistici.

4.2.4 Trigger per controllare che l'utente che aderisce ad una sessione in presenza sia iscritto al corso

```
CREATE OR REPLACE FUNCTION ControlloAdesioniCorso()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    corso_id INTEGER;
BEGIN
    -- Recupera l'IdCorso dalla sessione in presenza
    SELECT sip.IdCorso
    INTO corso_id
    FROM SessioneInPresenza AS sip
    WHERE sip.IdSessione = NEW.IdSessionePresenza;

    -- Controlla se l'utente non è iscritto al corso e nel caso solleva
        un'eccezione
    IF NOT EXISTS (
        SELECT 1
        FROM Iscrizione AS i
        WHERE i.IdCorso = corso_id
        AND i.UsernameUtente = NEW.UsernameUtente
    ) THEN
        RAISE EXCEPTION 'L''utente % non è iscritto al corso %.',
            NEW.UsernameUtente, (SELECT Titolo FROM Corso WHERE idCorso =
            corso_id);
    END IF;

    RETURN NEW;
END;
$$;

CREATE TRIGGER TR_Insert_Adesione
BEFORE INSERT ON Adesione
FOR EACH ROW
EXECUTE FUNCTION ControlloAdesioniCorso();
```

Listing 4.19: Trigger per controllare l'iscrizione dell'utente

Questo trigger viene attivato prima dell'inserimento di una nuova adesione e verifica che l'utente che sta aderendo a una sessione in presenza sia iscritto al corso corrispondente. Se l'utente non è iscritto al corso, il trigger genera un errore e impedisce l'inserimento dell'adesione. Questo è utile per garantire che solo gli utenti iscritti ai corsi possano partecipare alle sessioni in presenza, mantenendo la coerenza dei dati e le regole del sistema.