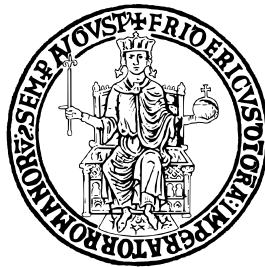


UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



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

<b>1</b>	<b>Descrizione del progetto</b>	<b>3</b>
1.1	Descrizione sintetica del problema . . . . .	3
<b>2</b>	<b>Progettazione Concettuale</b>	<b>4</b>
2.1	Introduzione . . . . .	4
2.2	Diagrammi UML e ER . . . . .	4
2.3	Ristrutturazione del modello concettuale . . . . .	5
2.4	Identificazione delle chiavi primarie . . . . .	5
2.5	Identificazione delle chiavi esterne . . . . .	5
2.6	Rimozione degli attributi multipli . . . . .	5
2.7	Rimozione delle gerarchie di specializzazione . . . . .	6
2.8	Analisi delle ridondanze . . . . .	6
2.9	Class Diagram ristrutturato . . . . .	6
2.10	Dizionario delle classi . . . . .	6
2.11	Dizionario delle associazioni . . . . .	6
2.12	Dizionario dei vincoli . . . . .	6
<b>3</b>	<b>Progettazione Logica</b>	<b>8</b>
3.1	Schema Logico . . . . .	8
<b>4</b>	<b>Progettazione Fisica</b>	<b>9</b>
4.1	Definizione tabelle . . . . .	9
4.1.1	Definizione della tabella TOPIC . . . . .	9
4.1.2	Definizione della tabella CHEF . . . . .	10
4.1.3	Definizione della tabella UTENTE . . . . .	10
4.1.4	Definizione della tabella CORSO . . . . .	11
4.1.5	Definizione della tabella SESSIONEINPRESENZA . . . . .	11
4.1.6	Definizione della tabella SESSIONEONLINE . . . . .	12
4.1.7	Definizione della tabella ADESIONE . . . . .	12
4.1.8	Definizione della tabella PREPARA . . . . .	13
4.1.9	Definizione della tabella RICETTA . . . . .	13
4.1.10	Definizione della tabella INGREDIENTE . . . . .	13
4.1.11	Definizione della tabella COMPOSIZIONE . . . . .	14
4.1.12	Definizione della tabella NOTIFICA . . . . .	14
4.1.13	Definizione della tabella RICEVE . . . . .	15
4.1.14	Definizione della tabella INSEGNA . . . . .	15
4.1.15	Definizione della tabella ISCRIZIONE . . . . .	16
4.2	Triggers, Procedures, Functions e Vincoli . . . . .	16
4.2.1	Funzione per calcolare il numero di adesioni . . . . .	16
4.2.2	Trigger per controllare che l'anno del corso sia valido . . . . .	17

4.2.3	Trigger per controllare che l'aula della sessione in presenza non sia occupata . . .	18
4.2.4	Trigger per controllare che l'utente che aderisce ad una sessione in presenza sia iscritto al corso . . . . .	19

# 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

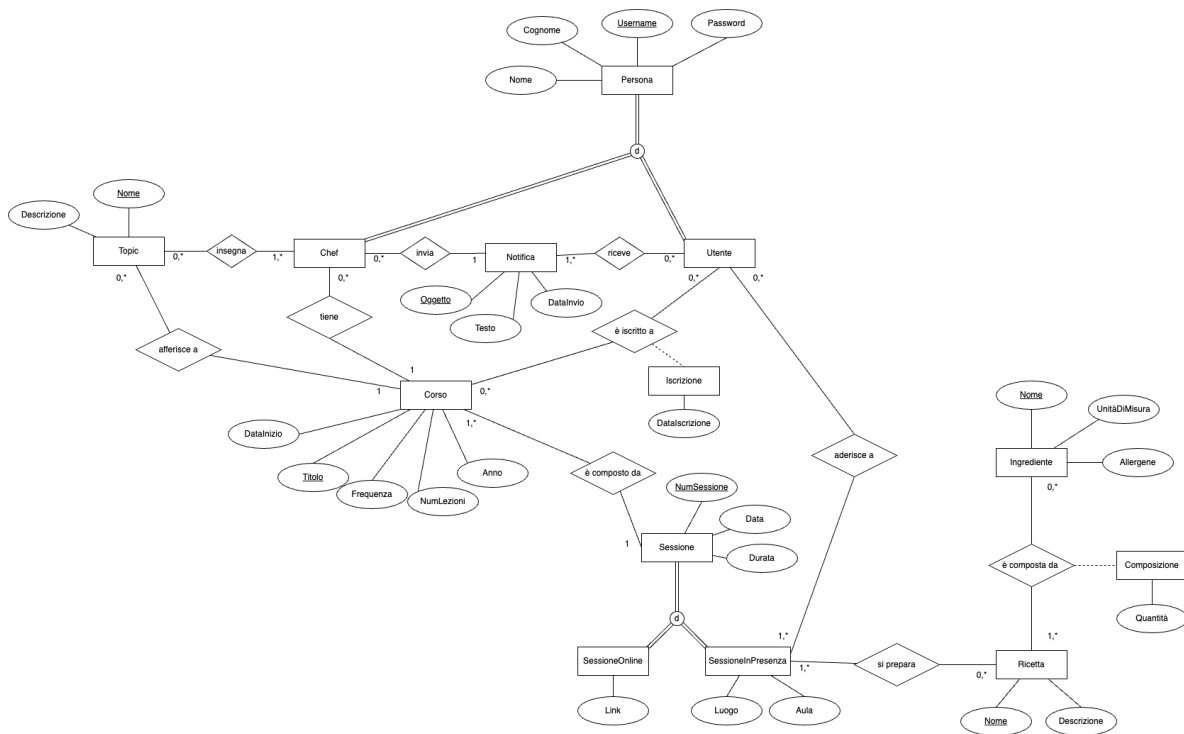


Figura 2.2: Diagramma Entità-Relazioni

## 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

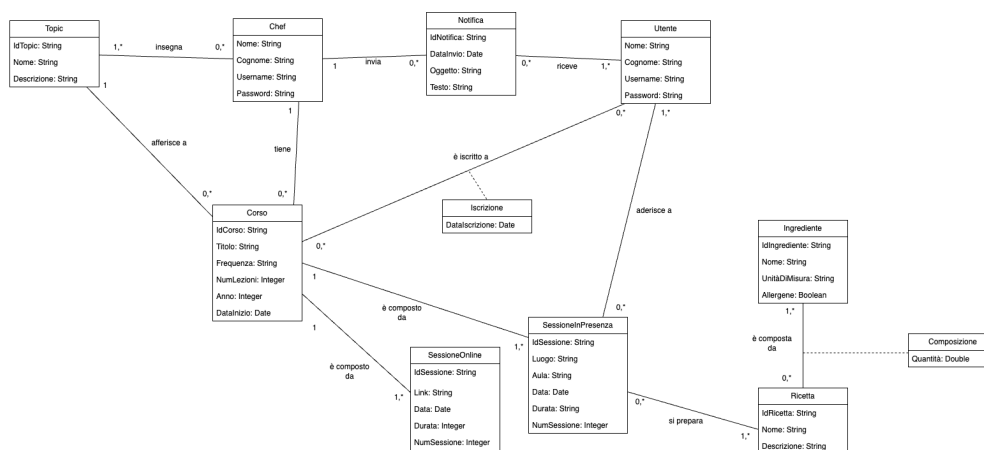


Figura 2.3: Diagramma UML ristrutturato

## 2.10 Dizionario delle classi

## 2.11 Dizionario delle associazioni

## 2.12 Dizionario dei vincoli

<b>Associazione</b>	<b>Descrizione</b>	<b>Classi coinvolte</b>
Insegna	Associa uno chef a un topic che insegna	Chef, Topic
Riceve	Associa un utente a una notifica ricevuta	Utente, Notifica
È iscritto a	Associa un utente a un corso a cui è iscritto	Utente, Corso, Iscrizione (Classe di associazione)
Aderisce a	Associa un utente a una sessione in presenza a cui partecipa	Utente, Sessione in Presenza
Si prepara	Associa una sessione in presenza a una ricetta che si prepara	Sessione in Presenza, Ricetta
È composta da	Associa una ricetta a un ingrediente che la compone	Ricetta, Ingrediente, Composizione (Classe di associazione)
Tiene	Associa uno chef a un corso che insegna	Chef, Corso
Afferisce a	Associa un corso al topic di appartenenza	Corso, Topic
È composto da (Online)	Associa un corso alle sue sessioni online	Corso, Sessione Online
È composto da (Presenza)	Associa un corso alle sue sessioni in presenza	Corso, Sessione in Presenza
Invia	Associa uno chef alle notifiche che invia	Chef, Notifica



## 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

```
1 CREATE TABLE Topic (  
2     idTopic SERIAL PRIMARY KEY,  
3     Nome VARCHAR(1000) NOT NULL UNIQUE,  
4     Descrizione VARCHAR(1000),  
5  
6     CONSTRAINT CK_Topic_Nome CHECK (Nome IN ('Cucina Italiana', 'Pizza  
7         Napoletana', 'Cucina Cinese',  
8         'Cucina Messicana', 'Cucina Giapponese', 'Cucina Indiana', 'Cucina  
9         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

```
1 CREATE TABLE Chef (  
2     Nome VARCHAR(50) NOT NULL,  
3     Cognome VARCHAR(50) NOT NULL,  
4     Username VARCHAR(100) PRIMARY KEY,  
5     Password VARCHAR(100) NOT NULL  
6 );
```

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

```
1 CREATE TABLE Utente (  
2     Nome VARCHAR(50) NOT NULL,  
3     Cognome VARCHAR(50) NOT NULL,  
4     Username VARCHAR(100) PRIMARY KEY,  
5     Password VARCHAR(100) NOT NULL  
6 );
```

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

```
1 CREATE TABLE Corso (  
2     idCorso SERIAL PRIMARY KEY,  
3     Titolo VARCHAR(50) NOT NULL,  
4     Frequenza VARCHAR(50) DEFAULT 'Settimanale',  
5     NumLezioni Integer NOT NULL,  
6     Anno Integer NOT NULL,  
7     DataInizio Date NOT NULL,  
8     IdTopic INTEGER NOT NULL,  
9     UsernameChef VARCHAR(100) NOT NULL,  
10  
11     CONSTRAINT FK_Corso_Topic FOREIGN KEY (IdTopic) REFERENCES  
12         Topic(idTopic), -- foreign key verso Topic  
13     CONSTRAINT FK_Corso_Chef FOREIGN KEY (UsernameChef) REFERENCES  
14         Chef(Username) ON DELETE CASCADE, -- foreign key verso Chef, su  
15         eliminazione di un chef elimina tutti i suoi corsi  
16     CONSTRAINT CK_Frequenza CHECK (Frequenza IN ('Settimanale', 'Ogni  
17         giorno', 'Ogni due giorni', 'Ogni tre giorni')), -- controllo che la  
18         frequenza del corso sia un valore valido  
19     CONSTRAINT CK_NumLezioni CHECK (NumLezioni BETWEEN 1 AND 100), --  
20         controllo che il numero di lezioni sia compreso tra 1 e 100  
21     CONSTRAINT CK_Anno CHECK (Anno = EXTRACT(YEAR FROM DataInizio)) --  
22         controllo che l'anno sia uguale all'anno di inizio  
23 )
```

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

```
1 CREATE TABLE SessioneInPresenza (  
2     IdSessione SERIAL PRIMARY KEY,  
3     Luogo VARCHAR(50) NOT NULL,  
4     Aula VARCHAR(50) NOT NULL,  
5     Data TIMESTAMP NOT NULL,  
6     Durata Integer NOT NULL,  
7     NumSessione Integer NOT NULL,  
8     IdCorso INTEGER NOT NULL,  
9  
10     CONSTRAINT FK_SessioneInPresenza_Corso FOREIGN KEY (IdCorso) REFERENCES  
11         Corso(idCorso) ON DELETE CASCADE, -- foreign key verso Corso, su  
12         eliminazione di un corso elimina tutte le sue sessioni  
13     CONSTRAINT CK_Luogo CHECK (Luogo IN ('MonteSantangelo', 'ViaClaudio',  
14         'PiazzaleTecchio', 'PortaDiMassa', 'ViaMarina')), -- controllo che il  
15         luogo sia un valore valido  
16     CONSTRAINT CK_Durata CHECK (Durata >= 60 AND Durata <= 180) -- controllo  
17         che la durata sia compresa tra 60 e 180 minuti  
18 )
```

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

```
1 CREATE TABLE SessioneOnline (
2     IdSessione SERIAL PRIMARY KEY,
3     Link VARCHAR(1000) NOT NULL,
4     Data TIMESTAMP NOT NULL,
5     Durata Integer NOT NULL,
6     NumSessione Integer NOT NULL,
7     IdCorso INTEGER NOT NULL,
8
9     CONSTRAINT FK_SessionOnline_Corso FOREIGN KEY (IdCorso) REFERENCES
10         Corso(idCorso) ON DELETE CASCADE, -- foreign key verso Corso, su
11         eliminazione di un corso elimina tutte le sue sessioni online
12     CONSTRAINT CK_Link CHECK (Link LIKE 'https://%'), -- controllo che il
13         link abbia un formato valido
14     CONSTRAINT CK_Durata CHECK (Durata >= 60 AND Durata <= 180) -- controllo
15         che la durata sia compresa tra 60 e 180 minuti
16 )
```

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

```
1 CREATE TABLE Adesione (
2     UsernameUtente VARCHAR(100) NOT NULL,
3     IdSessionePresenza INTEGER NOT NULL,
4
5     CONSTRAINT PK_Adesione PRIMARY KEY (UsernameUtente, IdSessionePresenza),
6     -- chiave primaria composta
7     CONSTRAINT FK_Adesione_Utente FOREIGN KEY (UsernameUtente) REFERENCES
8         Utente(Username) ON DELETE CASCADE, -- foreign key verso Utente, su
9         eliminazione di un utente elimina tutte le sue adesioni
10     CONSTRAINT FK_Adesione_SessionePresenza FOREIGN KEY (IdSessionePresenza)
11         REFERENCES SessioneInPresenza(IdSessione) ON DELETE CASCADE --
12         foreign key verso SessioneInPresenza, su eliminazione di una sessione
13         elimina tutte le adesioni ad essa collegate
14 )
```

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

```
1 CREATE TABLE Prepara (  
2     IdSessionePresenza INTEGER NOT NULL,  
3     IdRicetta INTEGER NOT NULL,  
4  
5     CONSTRAINT PK_Prepara PRIMARY KEY (IdSessionePresenza, IdRicetta), --  
6         chiave primaria composta  
7     CONSTRAINT FK_Prepara_SessionePresenza FOREIGN KEY (IdSessionePresenza)  
8         REFERENCES SessioneInPresenza(IdSessione) ON DELETE CASCADE, --  
9         foreign key verso SessioneInPresenza, su eliminazione di una sessione  
10        elimina tutte le preparazioni ad essa collegate  
11     CONSTRAINT FK_Prepara_Ricetta FOREIGN KEY (IdRicetta) REFERENCES  
12        Ricetta(IdRicetta) --foreign key verso Ricetta  
13 )
```

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

```
1 CREATE TABLE Ricetta (  
2     IdRicetta SERIAL PRIMARY KEY,  
3     Nome VARCHAR(100) NOT NULL UNIQUE,  
4     Descrizione VARCHAR(1000) NOT NULL  
5 )
```

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

```
1 CREATE TABLE Ingrediente (  
2     IdIngrediente SERIAL PRIMARY KEY,  
3     Nome VARCHAR(100) NOT NULL UNIQUE,  
4     Allergene BOOLEAN NOT NULL,  
5     UnitàMisura VARCHAR(50) NOT NULL  
6 )
```

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

```
1 CREATE TABLE Composizione (  
2     IdIngrediente INTEGER NOT NULL,  
3     IdRicetta INTEGER NOT NULL,  
4     Quantità FLOAT NOT NULL,  
5  
6     CONSTRAINT PK_Composizione PRIMARY KEY (IdIngrediente, IdRicetta), --  
7         chiave primaria composta  
8     CONSTRAINT FK_Composizione_Ingrediente FOREIGN KEY (IdIngrediente)  
9         REFERENCES Ingrediente(IdIngrediente), -- foreign key verso  
10        Ingrediente  
11     CONSTRAINT FK_Composizione_Ricetta FOREIGN KEY (IdRicetta) REFERENCES  
12        Ricetta(IdRicetta), -- foreign key verso Ricetta  
13     CONSTRAINT CK_Quantità_Positiva CHECK (Quantità > 0) -- controllo  
14        quantità ingrediente positiva  
15 )
```

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 uno o più ingredienti che la compongono, specificando la quantità necessaria per ciascuno di essi.

#### 4.1.12 Definizione della tabella NOTIFICA

```
1 CREATE TABLE Notifica (  
2     IdNotifica SERIAL PRIMARY KEY,  
3     UsernameChef VARCHAR(100) NOT NULL,  
4     Oggetto VARCHAR(1000) NOT NULL,  
5     Testo VARCHAR(1000),  
6     DataInvio TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
7  
8     CONSTRAINT FK_notifica_chef FOREIGN KEY (UsernameChef) REFERENCES  
9         Chef(Username) ON DELETE CASCADE -- foreign key verso Chef, su  
10        eliminazione di un chef elimina tutte le notifiche da lui inviate  
11 )
```

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

```
1 CREATE TABLE Riceve (  
2     IdNotifica INTEGER NOT NULL,  
3     UsernameUtente VARCHAR(100) NOT NULL,  
4  
5     CONSTRAINT PK_Riceve PRIMARY KEY (IdNotifica, UsernameUtente), -- chiave  
6         primaria composta  
7     CONSTRAINT FK_Riceve_Notifica FOREIGN KEY (IdNotifica) REFERENCES  
8         Notifica(IdNotifica) ON DELETE CASCADE, -- foreign key verso  
        Notifica, su eliminazione di una notifica elimina tutte le ricezioni  
        ad essa collegate  
9     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  
10 )
```

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

```
1 CREATE TABLE Insegna (  
2     IdTopic INTEGER NOT NULL,  
3     UsernameChef VARCHAR(1000) NOT NULL,  
4  
5     CONSTRAINT PK_Insegna PRIMARY KEY (IdTopic, UsernameChef), -- chiave  
6         primaria composta  
7     CONSTRAINT FK_Insegna_Topic FOREIGN KEY (IdTopic) REFERENCES  
8         Topic(idTopic), -- foreign key verso Topic  
9     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  
10 )
```

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

```
1 CREATE TABLE Iscrizione (
2     IdCorso INTEGER NOT NULL,
3     UsernameUtente VARCHAR(100) NOT NULL,
4     DataIscrizione TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
5
6     CONSTRAINT PK_Iscrizione PRIMARY KEY (IdCorso, UsernameUtente), -- chiave
7     CONSTRAINT FK_Iscrizione_Corso FOREIGN KEY (IdCorso) REFERENCES
8     Corso(idCorso) ON DELETE CASCADE, -- foreign key verso Corso, su
9     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

```
1 CREATE OR REPLACE FUNCTION CalcoloNumeroAdesioni(idSessione INTEGER) --
2     funzione per calcolare il numero di adesioni a una sessione in presenza
3 RETURNS INTEGER
4 LANGUAGE plpgsql
5 AS $$
6 DECLARE
7     numeroAdesioni INTEGER;
8 BEGIN
9     SELECT COUNT(*) INTO numeroAdesioni
10    FROM Adesione AS a
11   WHERE a.idSessione = idSessione;
12     IF numeroAdesioni = 0 THEN
13         RAISE EXCEPTION 'Nessuna adesione trovata per la sessione con ID %',
14             idSessione;
15     END IF;
16     RETURN numeroAdesioni;
17 END;
18 $$;
```

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

```
1 CREATE OR REPLACE FUNCTION ControllAnnoCorso()
2 RETURNS TRIGGER
3 LANGUAGE plpgsql
4 AS $$
5 BEGIN
6     -- Controlla se l'anno del corso è compreso tra l'anno corrente e il
7     -- prossimo
8     IF NEW.Anno < EXTRACT(YEAR FROM CURRENT_DATE) OR
9     NEW.Anno > EXTRACT(YEAR FROM CURRENT_DATE) +1 THEN
10         RAISE EXCEPTION 'L'anno del corso deve essere compreso tra l'anno
11         -- corrente e il prossimo.';
12     END IF;
13     RETURN NEW;
14 END;
15 $$;
16
17 CREATE TRIGGER TriggerAnnoCorso
18 BEFORE INSERT OR UPDATE ON Corso
19 FOR EACH ROW
20 EXECUTE FUNCTION ControllAnnoCorso();
```

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

```
1 CREATE OR REPLACE FUNCTION ControllAulaOccupata()
2 RETURNS TRIGGER
3 LANGUAGE plpgsql
4 AS $$
5 BEGIN
6     -- Controlla se esiste già una sessione in presenza nello stesso luogo e
7     -- aula che si sovrappone con la nuova sessione
8     IF EXISTS (
9         SELECT 1
10        FROM SessioneInPresenza AS sip
11        WHERE sip.Luogo = NEW.Luogo
12        AND sip.Aula = NEW.Aula
13        AND (NEW.Data >= sip.Data AND NEW.Data < sip.Data + (sip.Durata
14            || ' minutes')::INTERVAL)
15        OR (NEW.Data + (NEW.Durata || ' minutes')::INTERVAL >
16            sip.Data AND NEW.Data + (NEW.Durata || '
17            minutes')::INTERVAL <= sip.Data + (sip.Durata || '
18            minutes')::INTERVAL)
19        OR (NEW.Data <= sip.Data AND NEW.Data + (NEW.Durata || '
20            minutes')::INTERVAL >= sip.Data + (sip.Durata || '
21            minutes')::INTERVAL)
22    ) THEN
23        RAISE EXCEPTION 'L''aula % nel luogo % è occupata in data %',
24            NEW.Aula, NEW.Luogo, NEW.Data;
25    END IF;
26
27    RETURN NEW;
28 END;
29 $$;
30
31 CREATE TRIGGER Tr_Insert_Aula_Sessione_Presenza
32 BEFORE INSERT OR UPDATE ON SessioneInPresenza
33 FOR EACH ROW
34 EXECUTE FUNCTION ControllAulaOccupata();
```

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

```
1 CREATE OR REPLACE FUNCTION ControllaAdesioniCorso()
2 RETURNS TRIGGER
3 LANGUAGE plpgsql
4 AS $$
5 DECLARE
6     corso_id INTEGER;
7 BEGIN
8     -- Recupera l'IdCorso dalla sessione in presenza
9     SELECT sip.IdCorso
10    INTO corso_id
11   FROM SessioneInPresenza AS sip
12  WHERE sip.IdSessione = NEW.IdSessionePresenza;
13
14     -- Controlla se l'utente non è iscritto al corso e nel caso solleva
15     -- un'eccezione
16     IF NOT EXISTS (
17         SELECT 1
18        FROM Iscrizione AS i
19       WHERE i.IdCorso = corso_id
20         AND i.UsernameUtente = NEW.UsernameUtente
21     ) THEN
22         RAISE EXCEPTION 'L''utente % non è iscritto al corso %.',
23             NEW.UsernameUtente, (SELECT Titolo FROM Corso WHERE idCorso =
24             corso_id);
25     END IF;
26
27     RETURN NEW;
28 END;
29 $$;
30
31 CREATE TRIGGER TR_Insert_Adesione
32 BEFORE INSERT ON Adesione
33 FOR EACH ROW
34 EXECUTE FUNCTION ControllaAdesioniCorso();
```

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.