UninaFoodLab

Antonino De Martino
N86005103

Simone Barbella
N86004906

Anno Accademico 2024/2025

Indice

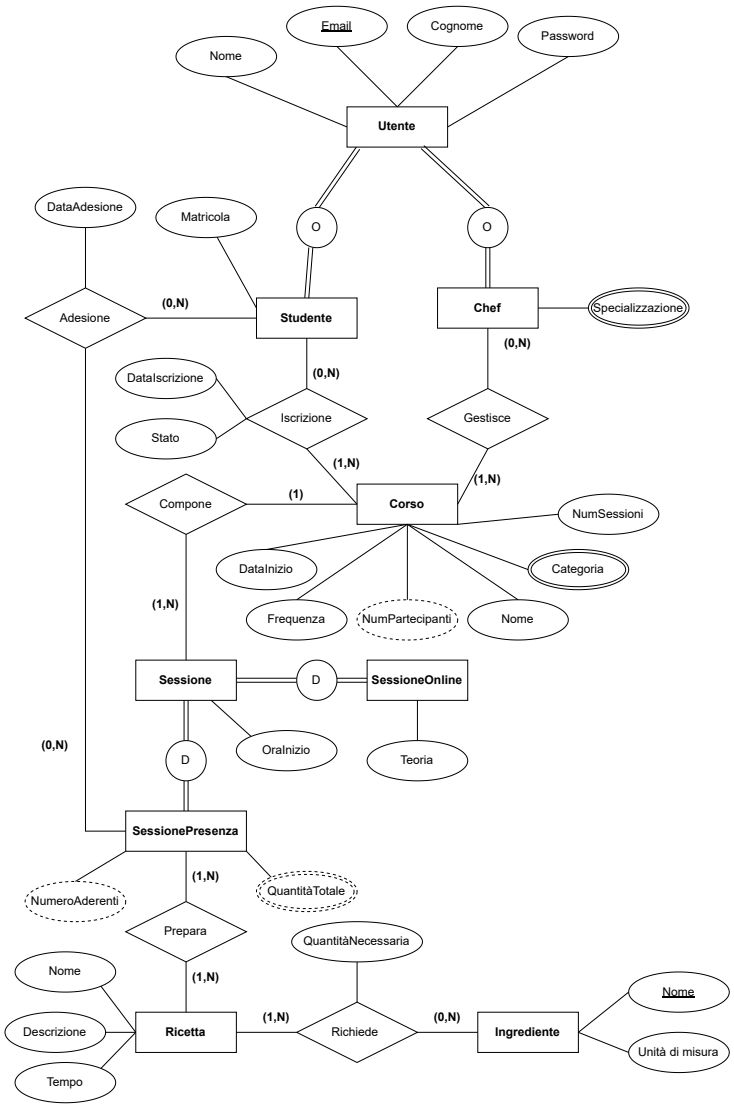
1 Presentazione del Progetto

1.1 Cos'è UninaFoodLab

Con UninaFoodLab, la didattica culinaria entra in una nuova era digitale, senza però perdere il contatto con la materia prima. Il progetto fonde la comodità dello studio teorico online con la necessità fondamentale della pratica in cucina, creando un ecosistema formativo completo e moderno.
L'aspetto più rivoluzionario del progetto risiede nella sua forte vocazione alla sostenibilità. L'obiettivo primario non si limita alla gestione di orari e corsi, ma punta dritto all'efficienza logistica: eliminare gli sprechi. Grazie a un sistema avanzato di adesioni e calcolo dei fabbisogni, UninaFoodLab trasforma la gestione della dispensa: non si acquista più in base a stime approssimative, ma in base alle reali necessità confermate. Questo approccio permette di ridurre drasticamente lo spreco alimentare, promuovendo una cultura del cibo più etica e consapevole all'interno dell'ambiente accademico.

2 Progettazione Concettuale

2.1 Diagramma ER



2.1.1 Descrizione Diagramma ER

Il diagramma Entità-Relazione (ER) proposto definisce l'architettura informativa della piattaforma UninFoodLab, progettata per digitalizzare i processi di un laboratorio culinario accademico. Il modello garantisce l'integrità referenziale e supporta le regole di business attraverso una struttura modulare.
La descrizione del diagramma è articolata nelle seguenti sei aree tematiche, che riflettono la logica sequenziale dei processi gestiti:

1. **Gestione Utenti e Gerarchie**: Definisce gli attori del sistema, implementando la specializzazione dei ruoli tra corpo docente (Chef) e discenti (Studenti);

2. **Corsi e Iscrizioni**: Modella l'offerta formativa e le relazioni di titolarità e fruizione, gestendo il ciclo di vita delle iscrizioni;

3. **Struttura delle Sessioni**: Dettaglia la scomposizione temporale dei corsi, introducendo la distinzione strutturale tra didattica online (teorica) e in presenza (pratica);

4. **Logistica in Presenza**: Approfondisce le dinamiche operative del laboratorio fisico, regolando i flussi di prenotazione degli studenti (Adesioni) e il calcolo puntuale dei fabbisogni di materie prime necessari per lo svolgimento della lezione;

5. **Scelta delle Chiavi Primarie**: Esplicita le decisioni progettuali riguardanti l'adozione di identificatori naturali per garantire una maggiore aderenza semantica al dominio applicativo

2.1.2 Gestione Utenti e Gerarchie

Il nucleo della gestione degli accessi e delle anagrafiche è rappresentato dall'entità padre **Utente**. Questa entità fattorizza le informazioni comuni a tutti gli attori del sistema, evitando ridondanze.

- **Attributi Comuni**: Ogni utente è definito da *Nome*, *Cognome*, *Email* e *Password*;

- **Gerarchia di Generalizzazione**: Il modello adotta una struttura gerarchica per gestire i ruoli. La relazione padre-figlio è definita dalle seguenti proprietà vincolanti:

  - **Totale**: Indica che l'unione delle entità figlie copre l'intera popolazione dell'entità padre. In termini operativi, non può esistere un "Utente

generico" nel sistema; ogni account registrato deve necessariamente appartenere ad almeno una delle categorie specifiche (**Studente** o **Chef**);

- **Overlapping** (Cerchio con "O"): Questa configurazione permette la sovrapposizione dei ruoli. Un singolo utente può esistere simultaneamente sia come **Studente** che come **Chef** (configurando il ruolo ibrido di "**ChefStudente**"). Questa scelta garantisce flessibilità, permettendo ad esempio a uno chef di iscriversi a corsi tenuti da colleghi per aggiornamento professionale.

- Le entità figlie estendono il padre con attributi specifici:
  - **Studente**: Identificato univocamente nel contesto accademico dalla *matricola*;
  - **Chef**: Qualificato professionalmente dall'attributo *specializzazione*.

2.1.3 Corsi e Iscrizioni

L'entità **Corso** rappresenta l'unità centrale dell'offerta didattica, fungendo da aggregatore per **Chef**e **Studente**. La sua esistenza è regolata da due relazioni fondamentali che definiscono, rispettivamente, la titolarità dell'insegnamento e la fruizione del servizio.

- Relazione **Gestisce** (Titolarità del Corso): Questa associazione collega l'entità **Chef** a **Corso**:
  - **Chef → Corso (0..N)** : Uno **Chef** può non avere corsi attivi in un determinato momento (0) oppure gestirne molteplici contemporaneamente (N);
  - **Corso → Chef (1..N) - Vincolo di Esistenza)**: Un **Corso**, per essere creato e mantenuto nel sistema, necessita obbligatoriamente di un responsabile. Il vincolo minimo "1" impedisce la presenza di "corsi orfani" senza docente, garantendo la qualità dell'offerta.

- Relazione **Iscrizione** (Fruizione Didattica): Collega l'entità **Studente** a **Corso**. Trattandosi di una relazione **Molti-a-Molti**, essa evolve in una tabella associativa dotata di attributi propri che storicizzano il legame:
  - Attributi della relazione:
    - *dataIscrizione*: Traccia temporalmente il momento dell'ingresso dello studente nel corso;
    - *stato*: Definisce la fase corrente del percorso (es. Attivo, Completato, Ritirato), permettendo di monitorare il progresso accademico.

- Cardinalità della relazione:
  - **Studente → Iscrizione (0..N)**: Uno **Studente** registrato può non essere iscritto a nessun corso (0) o seguirne diversi in parallelo (N).
  - **Corso → Iscrizione (1..N) - Vincolo di Attivazione)**: Similmente alla gestione degli **Chef**, il modello impone che un corso esista solo se vi è una platea di destinatari. Un corso con 0 iscritti non è contemplato operativamente come istanza attiva nel sistema.

2.1.4 Struttura delle Sessioni

Il **Corso** non è un'entità monolitica, ma viene declinato in unità didattiche temporali attraverso la relazione **Compone**. Questa associazione definisce la scomposizione modulare del percorso formativo in singole lezioni, permettendo una pianificazione granulare del calendario accademico.

- **Relazione Compone** (Vincoli di Struttura): La relazione lega l'offerta formativa alla sua erogazione pratica. Le cardinalità imposte garantiscono la consistenza strutturale:
  - **Corso → Sessioni (1..N)**: Un corso non può esistere come "scatola vuota"; affinché sia valido, deve essere composto da almeno una sessione pianificata;
  - **Sessione → Corso (1..1**): Ogni sessione è strettamente dipendente dal corso di appartenenza. Non esistono sessioni "libere" o condivise tra più corsi; ogni lezione è un'istanza esclusiva del percorso formativo di riferimento.
- **Entità Sessione** (Unità Atomica): La Sessione rappresenta l'unità base di erogazione del servizio didattico.
  - Attributo *oraInizio*: Fondamentale per la calendarizzazione, questo attributo temporale definisce il collocamento cronologico della lezione, permettendo la gestione degli orari.
- **Gerarchia delle Sessioni** (Generalizzazione): Per gestire la natura ibrida della didattica, è stata modellata una gerarchia. Si tratta di una generalizzazione **Totale** e **Disgiunta** (cerchio con "D"):
  - **Disgiunta**: Impone una mutua esclusività rigida. Un'istanza di **sessione** non può possedere simultaneamente le caratteristiche di una lezione online e di una in presenza. Lo stato è binario: o la lezione avviene da remoto, o avviene in laboratorio.

La gerarchia si dirama nelle seguenti entità figlie:

- **SessioneOnline**: Rappresenta le lezioni erogate tramite piattaforme digitali.
  - Attributo *teoria*: Questo attributo specifico qualifica il contenuto della lezione (es. link alla piattaforma, argomento teorico trattato), focalizzandosi esclusivamente sulla componente concettuale e nozionistica, svincolata dalle necessità logistiche fisiche.
- **SessionePresenza** (Laboratorio Fisico): Rappresenta le lezioni pratiche svolte all'interno delle strutture dell'istituto.
  - Questa entità funge da snodo cruciale per l'intera gestione logistica (descritta nella sezione successiva), poiché è l'unico punto del sistema che abilita le associazioni con gli ingredienti, le ricette e la presenza fisica degli studenti.
  - Attributo Derivato Multivalore (*quantitàTotale*): A livello concettuale, è stato definito l'attributo *quantitàTotale* per rappresentare il fabbisogno complessivo di materie prime. Esso è classificato come multivalore (in quanto riferito a una lista eterogenea di ingredienti) e derivabile (rappresentato graficamente con linea tratteggiata).

2.1.5 Logistica in Presenza (Adesioni e Fabbisogni)

Questa porzione del diagramma modella le dinamiche operative che avvengono fisicamente all'interno del laboratorio, distinguendo le attività pratiche dal semplice apprendimento teorico. Qui vengono gestiti i flussi di prenotazione degli studenti e la definizione puntuale delle materie prime necessarie.

- **Adesione degli Studenti**: A differenza della generica iscrizione al **Corso**, la relazione **Adesione** rappresenta la prenotazione puntuale di uno studente a una specifica lezione pratica (**SessionePresenza**).
  - Attributo *dataAdesione*: Registra il momento esatto della prenotazione, fondamentale per gestire priorità o scadenze temporali per la partecipazione;
  - Attributo Derivato *numAderenti*: La SessionePresenza include questo attributo calcolato, che conta dinamicamente le istanze nella relazione **Adesione** associate a quella **sessione**.
- **Definizione dei Fabbisogni (Relazione Richiede)**: La relazione **Richiede** costituisce la distinta base tecnica del laboratorio, collegando ogni **Ricetta** ai singoli **Ingredienti** necessari per la sua realizzazione. Questa associazione è fondamentale per tradurre un piatto in una lista di spesa.

- Attributo *quantitàNecessaria*: Rappresenta la dose unitaria effettiva dell'ingrediente per la specifica ricetta . Questo attributo è il moltiplicatore base che, incrociato successivamente con il numero di aderenti alla sessione, permetterà al sistema di calcolare la *quantitàTotale* (attributo derivato multivalore).
- **Analisi delle Cardinalità (Vincoli di Struttura)**: Le cardinalità imposte sulla relazione **Richiede** garantiscono la consistenza delle schede tecniche:
  * **Ricetta → Ingredienti (1..N)**: Una **ricetta** non può esistere come entità astratta priva di componenti; affinché sia valida e "cucinabile", deve essere composta da almeno un **ingrediente**;
  * **Ingrediente → Ricette (0..N)**: Un **ingrediente** catalogato in magazzino (es. una spezia rara) potrebbe momentaneamente non essere utilizzato in nessuna delle ricette attive nel menu corrente (cardinalità minima 0), ma può potenzialmente comparire in infinite preparazioni diverse (cardinalità massima N).

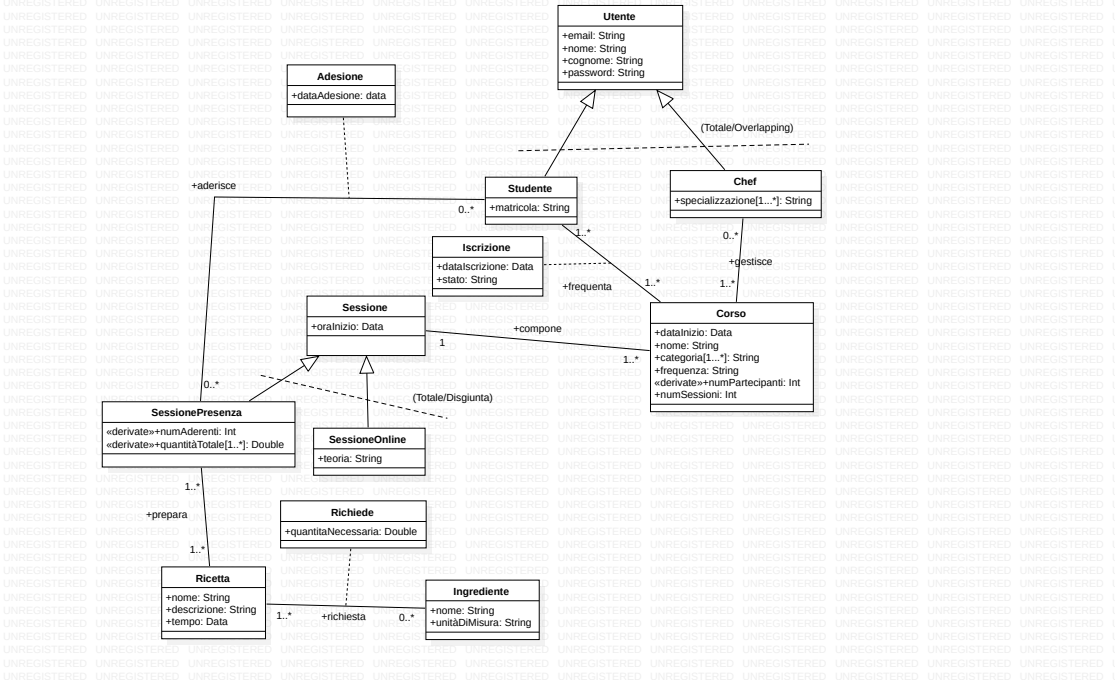
2.1.6 Scelta delle Chiavi Primarie (Identificatori Naturali)

Nel progettare lo schema, si è scelto di utilizzare, ove possibile, **chiavi naturali** (proprietà inerenti all'entità) anziché chiavi surrogate, al fine di mantenere una forte coerenza concettuale con il dominio applicativo.

- **Email (Entità Utente)**: È stata selezionata l'attributo *email* come chiave primaria per l'entità **Utente**. Nel contesto di una piattaforma web, l'indirizzo email costituisce un identificatore univoco globale che garantisce l'assenza di duplicati per la stessa persona fisica. Questa scelta ottimizza anche il processo di autenticazione, poiché l'email funge sia da identificativo di accesso che da chiave di vincolo nel database;

- **Nome (Entità Ingrediente)**: Per l'entità **Ingrediente**, si è scelto l'attributo *nome* come chiave primaria. Si assume che nel dominio di riferimento (la dispensa del laboratorio) ogni ingrediente sia catalogato con una nomenclatura univoca (es. "Farina", "Zucchero a velo"). L'uso del nome come chiave evita ridondanze e semplifica le interrogazioni, rendendo le associazioni immediatamente leggibili senza necessità di operazioni di join per recuperare la descrizione dell'ingrediente.

2.2 Diagramma UML



2.2.1 Descrizione Diagramma UML

2.2.2 Tipi di Dato e Precisione Numerica

Il diagramma UML definisce esattamente come i dati devono essere memorizzati, sciogliendo le ambiguità generiche dell'ER:

- **Gestione delle Quantità**: Si nota una distinzione importante tra Integer e Decimal;
- Contatori come *numPartecipanti* e *numAderenti* sono Int, poiché contano unità discrete (persone, lezioni);
- Le misure fisiche come *quantitàTotale* e *quantitàNecessaria* sono definite come Decimal. Questo indica che il sistema gestirà valori decimali precisi (es. 1.5 kg, 0.25 litri), dettaglio fondamentale per le ricette che l'ER lasciava implicito.

- **Date e Orari**: i riferimenti temporali sono distinti in base al significato: *dataAdesione* e *dataIscrizione* sono Date, *oraInizio* è un DateTime (data+ora), mentre *tempo* è un Time (ora, minuti, secondi).

2.2.3 Rappresentazione degli Array (Attributi Multivalore)

L'UML traduce graficamente gli attributi multivalore dell'ER con la notazione delle parentesi quadre:

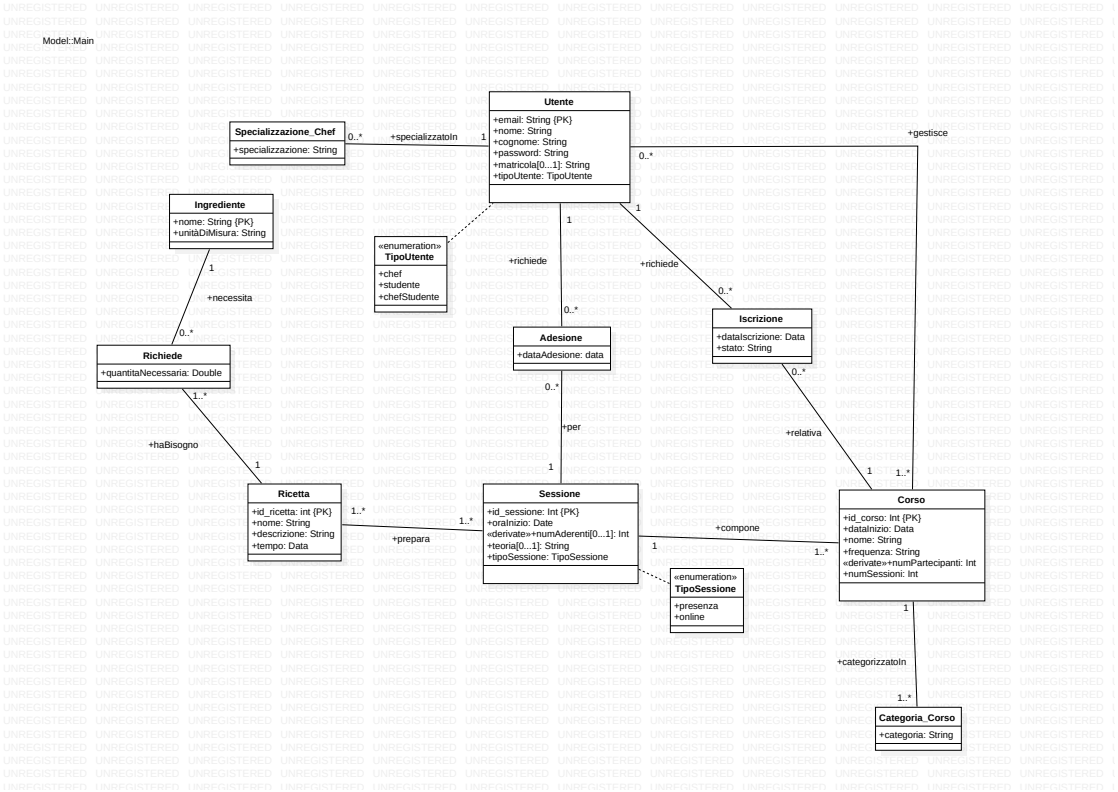
- +*specializzazione*[1...*]: String nella classe **Chef**;
- +*categoria*[1...*]: String nella classe **Corso**.

Questa notazione dice esplicitamente che posso avere più valori.
Un caso particolare è l'attributo della logistica: *«derivate» quantitàTotale*[1...*]: nella classe **SessionePresenza**. In questo caso, la notazione evidenzia due aspetti:

- Lo stereotipo «derivate»: indica che il dato non viene memorizzato, ma è calcolato dinamicamente dal sistema (aggregando le ricette).
- Le parentesi [1...*]: specificano che il risultato del calcolo non è un valore unico, ma una lista di ingredienti e quantità.

3 Progettazione Logica

3.1 Ristrutturazione Diagramma UML



3.1.1 Descrizione Delle Scelte

Nel passaggio dal modello concettuale a quello logico, si è scelto di semplificare la struttura delle gerarchie ereditarie adottando la strategia dell'accorpare nella classe genitore. Questa decisione mira a ridurre la complessità dello schema e a ottimizzare le prestazioni, evitando join eccessivi per recuperare informazioni su entità strettamente correlate.

3.1.2 Accorpamento della gerarchia Utente

Nel modello iniziale, l'entità **Utente** si specializzava nelle sottoclassi **Studente** e **Chef**. L'analisi ha evidenziato che la sottoclasse **Studente** possedeva un set di attributi propri molto ridotto, rendendo una separazione strutturale su tabelle dis-

tinte poco efficiente. Si è proceduto quindi alla fusione in un'unica entità **Utente**, gestendo le specificità tramite le seguenti modifiche:

- **Attributo Discriminante (Enum)**: È stato introdotto il campo *TipoUtente* basato su un'enumerazione. I valori previsti sono *chef*, *studente* e *chefStudente*. Quest'ultimo valore è cruciale per implementare logicamente la generalizzazione overlapping, permettendo al sistema di identificare gli utenti che ricoprono entrambi i ruoli senza duplicare i record.
- Gestione degli Attributi Specifici:
  - *matricola* [0..1]: L'attributo scalare è stato integrato nella tabella **Utente** con vincolo di nullabilità (nullable). Sarà valorizzato esclusivamente se il *TipoUtente* include il ruolo di *studente* o *chefStudente*;
  - *specializzazione* (Esternalizzazione): A differenza dell'attributo *matricola*, l'attributo *specializzazione* presentava una cardinalità multivalore (uno **Chef** può avere più competenze). Per rispettare la giusta gestione degli attributi multivalori, non è stato accorpato nell'entità **Utente**, ma è stato promosso a relazione indipendente (tabella **Specializzazione_Chef**), collegata all'utente tramite chiave esterna.

3.1.3 Accorpamento della gerarchia Sessione

Analogamente, la generalizzazione tra **SessioneOnline** e **SessionePresenza** è stata risolta accorpando le sottoclassi nell'entità padre **Sessione**. La scarsa complessità informativa della classe **SessioneOnline** non giustificava il mantenimento di una tabella dedicata.

- **Attributo Discriminante (Enum)**: È stato introdotto il campo TipoSessione con i valori *online* e *presenza*. Poiché la generalizzazione originale era disgiunta, questo attributo assume un valore mutuamente esclusivo.
- **Integrazione degli Attributi Condizionali**: La tabella unica **Sessione** accoglie ora l'unione degli attributi delle sottoclassi originali, resi opzionali per garantirne la coerenza semantica:
  - *teoria* [0..1]: Attributo testuale valorizzato solo se l'attributo discriminante *TipoSessione* è *online*;
  - **Attributi Logistici (*numAderenti*, *quantitàTotale*)**: Questi attributi sono pertinenti e validi esclusivamente se l'attributo discriminante *TipoSessione* è *presenza*. In caso di sessione online, tali campi restano nulli o non calcolabili, riflettendo l'assenza di logistica fisica.

3.1.4 Gestione degli Attributi Multivalore (Normalizzazione)

Durante la fase di traduzione dal modello concettuale al modello logico relazionale, è stata posta particolare attenzione alla gestione degli attributi definiti come **multivalore**. Il modello relazionale impone il rispetto della **Prima Forma Normale (1FN)**, la quale prescrive l'atomicità dei valori: ogni cella di una tabella deve contenere un singolo valore e non sono ammessi gruppi ripetitivi o array.
Per risolvere questa discrepanza strutturale, si è proceduto alla promozione degli attributi multivalore, trasformandoli in relazioni (tabelle) distinte collegate all'entità principale tramite vincoli di chiave esterna.

1. **Esternalizzazione delle Specializzazioni Chef**
   Nel modello concettuale, l'entità **Chef** possedeva l'attributo *specializzazione* con cardinalità [1..N], indicando che un docente può possedere competenze multiple. L'integrazione di questo dato direttamente nella tabella **Utente** avrebbe generato una ridondanza dei dati anagrafici (ripetendo le informazioni dell'utente per ogni specializzazione) o la creazione di campi non atomici, compromettendo l'integrità del database e violando la Prima Forma Normale. Quindi si è optato per la creazione della tabella dedicata **Specializzazione_Chef**, composta dalla coppia di attributi (*email_chef, specializzazione*) e avente una chiave primaria composta da entrambi gli attributi, garantendo che non ci siano duplicati della stessa competenza per lo stesso chef.

2. **Esternalizzazione delle Categorie Corso**
   Analogamente, l'attributo *categoria* associato all'entità **Corso** è stato identificato come multivalore (un corso può appartenere a più aree tematiche simultaneamente, es. "Primi Piatti" e "Tradizione Romana"), perciò è stata introdotta la tabella di collegamento **Categoria_Corso**, composta dalla coppia di attributi (*id_corso*, *categoria*) e avente una chiave primaria composta da entrambi gli attributi.
   Questa struttura permette una classificazione flessibile dell'offerta formativa, consentendo ricerche filtrate per categoria senza appesantire la tabella principale Corso con campi testuali ridondanti.

3.1.5 Gestione dell'attributo derivato multivalore quantitàTotale

Nella fase di ristrutturazione del diagramma per il passaggio al modello logico-relazionale, è stata posta particolare attenzione all'attributo *quantitàTotale*, definito nel modello concettuale come derivato e multivalore. La traduzione diretta di tale attributo all'interno della tabella **Sessione** avrebbe comportato due gravi violazioni delle forme normali:

- **Violazione della 1FN (Atomicità)**: Trattandosi di un attributo multivalore (una lista di ingredienti diversi per ogni sessione), non può essere memorizzato in un'unica cella atomica.

- **Ridondanza e Rischio di Inconsistenza**: Memorizzare un valore calcolato (il totale) che dipende da altri dati già presenti nel database (*quantitàNecessaria* in **Richiede** e *numAderenti* in **Adesione**) introdurrebbe una ridondanza. Se una dose venisse modificata, il totale memorizzato diventerebbe obsoleto (anomalia di aggiornamento).

Quindi si è optato all'utilizzo di una **Vista Logica (VIEW**).
Per risolvere queste criticità mantenendo l'accesso agevole al dato aggregato, l'attributo è stato rimosso dallo schema fisico delle tabelle ed è stato implementato tramite una Vista SQL (Virtual Relation).La vista, denominata **Vista_Fabbisogni_Sessione**, non memorizza fisicamente i dati, ma esegue una query di aggregazione a runtime che incrocia le seguenti entità:

- **Sessione**: Fornisce il moltiplicatore (*NumAderenti*).

- **Prepara (Tabella di collegamento Sessione-Ricetta)**: Funge da filtro, identificando quali ricette sono effettivamente previste per quella specifica sessione. Senza questa tabella ponte, sarebbe impossibile sapere quale "lista ingredienti" attivare per la lezione.

- **Richiede**: Fornisce la dose unitaria (*quantitàNecessaria*) per ogni ingrediente della ricetta selezionata.

3.1.6 Adozione delle Chiavi Surrogate (Identificativi Artificiali)

Mentre per le entità anagrafiche e stabili (come **Utente** e **Ingrediente**) si è optato per chiavi naturali, per le entità associative e di gestione eventi si è ritenuto necessario introdurre **chiavi surrogate** (o artificiali). Tali chiavi sono costituite da codici numerici autoincrementali privi di significato semantico intrinseco.
La scelta di utilizzare identificativi artificiali per entità come **Corso**, **Sessione** e **Ricetta** è motivata da tre criteri progettuali fondamentali:

1. **Stabilità e Indipendenza Semantica**
   Una Chiave Primaria deve soddisfare il requisito di immutabilità. Utilizzare attributi descrittivi (come il "Titolo del Corso" o il "Nome della Ricetta") come chiave primaria esporrebbe il database a gravi problemi di integrità referenziale nel caso in cui tali nomi dovessero subire modifiche o correzioni.

2. **Efficienza nelle Operazioni di Join**
   Dal punto di vista delle prestazioni, i database relazionali gestiscono i confronti tra numeri interi (INTEGER) in modo significativamente più rapido rispetto al confronto tra stringhe di testo (VARCHAR). Poiché le tabelle operative (come **Iscrizione** o **Richiede**) contengono migliaia di record che fanno riferimento ai corsi e agli ingredienti, l'uso di chiavi intere riduce la dimensione degli indici e velocizza le operazioni di JOIN.

3. **Semplificazione delle Chiavi Esterne**
   Per alcune entità deboli o dipendenti, l'uso di chiavi naturali avrebbe richiesto la creazione di chiavi composte complesse. Esempio:

   - **Sessione**: Senza una chiave surrogata, per identificare univocamente una sessione sarebbe stato necessario combinare *id_corso* + *OraInizio*. Utilizzando un semplice *id_sessione* univoco, si semplifica drasticamente la struttura delle tabelle associative (come **Adesione** e **Prepara**), che necessitano di importare una sola colonna numerica come chiave esterna anziché due.

3.2 Dizionari

3.2.1 Dizionario delle Classi

Classe	Descrizione	Attributi
Utente	Descrittore di ogni Studente o Chef	`email` [PK](String): è un indirizzo email unico e valido che permette di contattare l'utente. `nome` (String): è il nome dell'utente. `cognome` (String): è cognome dell'utente. `password` (String): una password permette all'utente di proteggere il suo account e le sue informazioni. `matricola` (String): string alfanumerica che identifica lo studente. `tipoUtente` (TipoUtente): indica il tipo di utente: (chef), (studente) oppure (chefStudente).
Corso	Descrittore di ciascun corso creato	`id_corso` [PK] (Int): identificativo univoco di Corso. `dataInizio` (Data): indica l'inizio del corso. `nome` (String): è il nome assegnato al corso. `frequenza` (String): indica ogni quanti giorni il corso si svolge. `numPartecipanti` (Int): Numero dei partecipanti al corso. `numSessioni` (Int): specifica quante sessioni ha un corso.
Iscrizione	Descrittore dell'iscrizione degli utenti	`dataIscrizione` (Data): indica l'inizio dell'iscrizione per un corso. `Stato` (String): indica lo stato dell'iscrizione di uno studente.

Classe	Descrizione	Attributi
Sessione	Descrittore di ciascuna sessione presente in un corso	`id_sessione` [PK] (Int): identificativo univoco di Sessione `oraInizio` (Data): orario di inizio della sessione. `numAderenti` (Int): numero di partecipanti alla sessione. `teoria` (String): argomento teorico spiegato in sessione. `tipoSessione` (TipoSessione): indica il tipo di sessione: (online) oppure (presenza).
Ricetta	Descrittore di ciascuna ricetta usata in una sessione	`id_ricetta` [PK] (Int): identificativo univoco di Ricetta. `nome` (String): nome della ricetta. `descrizione` (String): la preaparazione della ricetta. `tempo` (Time): tempo per la preparazione della ricetta.
Ingrediente	Descrittore di ciascun ingrediente usato per una ricetta	`nome` [PK] (String): nome dell'ingrediente. `unitàDiMisura` (String): specifica dell'unità di misura di un ingrediente ES: Kg, litri ecc.. .
Adesione	Classe associativa per gestire l'associazione tra Utente (studente/chefStudente) e Sessione (presenza)	`dataAdesione` (Data): data relativa all'adesione di uno studente ad una sessione.
Richiede	Classe associativa per gestire l'associazione tra Ingrediente e Ricetta	`quantitàNecessaria` (Decimal): quantità utilizzata di ogni singolo ingrediente per una ricetta.
Categoria_Corso	Classe per gestire l'attributo multivalore categoria del corso	`categoria` (String): una categoria associata a un corso.
Specializzazione_Chef	Classe per gestire l'attributo multivalore specializzazione dello chef.	`specializzazione` (String): una specializzazione associata a uno chef.

3.2.2 Dizionario Associazioni

Associazione	**Descrizione**	**Molteplicità delle Classi coinvolte**
richiede (adesione)	Richiesta di adesione da parte di uno studente	Utente [0..*] - Adesione [1]
richiede (iscrizione)	Richiesta di iscrizione da parte di uno studente	Utente [0..*] - Iscrizione [1]
gestisce	Gestione di un corso da parte di uno chef	Utente [0..*] - Corso [1..*]
compone	Un corso è composto da più sessioni	Corso [1] - Sessione [1..*]
prepara	Ad ogni sessione in presenza si prepara una ricetta	Sessione [1..*] - Ricetta [1..*]
haBisogno	ogni ricetta ha bisogno di sapere la quantità degli ingredienti	Ricetta [1] - Richiede [1..*]
necessita	Un ingrediente necessita di sapere la quantità per la ricetta	Richiede [1..*] - Ingrediente [1]
per	Richiesta di adesione per una sessione in presenza	Adesione [1] - Sessione [0..*]
relativa	Iscrizione di uno studente per un corso	Iscrizione [1] - Corso [1..*]
specializzatoIn	Specializzazione di uno chef	Utente [1] - Specializzazione_Chef [0..*]
categorizzatoIn	Categoria a cui appartiene un corso	Corso [1] - Categoria_Corso [1..*]

3.2.3 Dizionario delle Viste

Viste	Descrizione	Attributi
Vista_Fabbisogni_Sessione	Struttura virtuale che aggrega i dati logistici. Incrocia le adesioni e le ricette per calcolare il fabbisogno puntuale di materie prime	quantità_totale: (Derivato) Valore numerico che esprime il fabbisogno complessivo dello specifico ingrediente per la specifica sessione (calcolato come: Dose Unitaria × Numero Partecipanti).

3.2.4 Dizionario Vincoli

Vincolo	Tipo	Descrizione
email .. PRIMARY KEY	Integrità dell'entità	Identifica univocamente ogni utente. Garantisce l'unicità dell'email e che essa non sia nulla.
check_email	Dominio	L'attributo "email" della classe Utente deve contenere una combinazione non nulla di lettere, numeri e simboli, seguiti da una chiocciola ("@"), altre lettere/simboli, un punto ("."), e finire con almeno due lettere.
password VARCHAR(255) NOT NULL	Dominio	Impone che l'attributo "password" della classe Utente sia una stringa alfanumerica <= 255 caratteri e che sia obbligatoriamente non nullo.
check_password	Dominio	L'attributo "password" della classe Utente deve essere lungo almeno 8 caratteri, e avere almeno una lettera maiuscola, una minuscola, un numero e un carattere speciale.
nome VARCHAR(50) NOT NULL (Utente)	Dominio	Impone che l'attributo "nome" della classe Utente sia una stringa alfanumerica <= 50 caratteri e che sia obbligatoriamente non nullo.

Vincolo	Tipo	Descrizione
check_nome_utente	Dominio	L'attributo "nome" della classe Utente deve contenere solo stringhe di caratteri alfabetici.
cognome VARCHAR(50) NOT NULL	Dominio	Impone che l'attributo "cognome" della classe Utente sia una stringa alfanumerica <= 50 caratteri e che sia obbligatoriamente non nullo.
check_cognome	Dominio	L'attributo "cognome" della classe Utente deve contenere solo stringhe di caratteri alfabetici.
tipo_utente tipo_utente_enum NOT NULL	Dominio	L'attributo "tipoUtente" della classe Utente è un'enumerazione che può assumere solo i valori: chef, studente, chefStudente.
check_ruolo_dati	Tupla	Se l'attributo "tipoUtente" della classe Utente è studente o chefStudente, allora l'attributo "matricola" assume un valore alfanumerico <= 20.
matricola .. UNIQUE	Intrarelazionale	Impone l'assenza di duplicati per l'attributo, assicurando che ad ogni matricola corrisponda un solo profilo registrato nel sistema.
data_inizio DATE NOT NULL	Dominio	L'attributo "dataInizio" della classe Corso deve essere una data valida nel formato previsto, non può essere nulla.
nome VARCHAR(50) NOT NULL (Corso)	Dominio	Impone che l'attributo "nome" della classe Corso sia una stringa alfanumerica <= 50 caratteri e che sia obbligatoriamente non nullo.
check_nome_corso	Dominio	L'attributo "nome" della classe Corso deve contenere una stringa con solo caratteri alfabetici.

Vincolo	Tipo	Descrizione
frequenza VARCHAR(50) NOT NULL	Dominio	Impone che l'attributo "frequenza" ella classe Corso sia una stringa alfanumerica <= 50 caratteri e che sia obbligatoriamente non nullo.
trg_partecipanti	Interrelazionale	L'attributo "numPartecipanti" della classe Corso è calcolato automaticamente in base al numero di iscrizioni associate.
num_sessioni .. NOT NULL	Dominio	Impone che l'attributo "numSessioni" della classe Corso sia obbligatoriamente non nullo.
check_num_sessioni	Dominio	L'attributo "numSessioni" della classe Corso deve essere un intero positivo >0.
ora_inizio .. NOT NULL	Dominio	L'attributo "oraInizio" della classe Sessione deve essere una data valida nel formato previsto, non può essere nulla.
tipo_sessione tipo_sessione_enum NOT NULL	Dominio	L'attributo "tipoSessione" della classe Sessione è un'enumerazione che può assumere solo i valori: presenza o online.
check_logica_sessione	Tupla	Se l'attributo "tipoSessione" della classe Sessione è "presenza", allora "teoria" deve essere NULL. Se l'attributo "tipoSessione" è "online", allora "teoria" deve essere una stringa alfanumerica con lunghezza <=255.
trg_aderenti	Interrelazionale	Se l'attributo "tipoSessione" della classe Sessione è "presenza", allora "numAderenti" è calcolato automaticamente in base al numero di adesioni associate.

Vincolo	Tipo	Descrizione
trg_check_iscrizione	Interrelazionale	La relazione "Iscrizione" può coinvolgere solo utenti con tipoUtente = studente o chefStudente.
trg_check_adesione	Interrelazionale	La relazione "Adesione" può coinvolgere solo utenti con tipoUtente = studente o chefStudente e solo sessioni con tipoSessione = presenza. Inoltre verifica che l'utente sia iscritto al corso.
trg_check_prepara	Interrelazionale	La relazione "Prepara" può coinvolgere solo sessioni con tipoSessione = presenza. Ogni sessione in presenza prepara almeno una ricetta e ogni ricetta è preparata in almeno una sessione.
nome VARCHAR(50) NOT NULL (Ricetta)	Dominio	Impone che l'attributo "nome" della classe Ricetta sia una stringa alfanumerica <= 50 caratteri e che sia obbligatoriamente non nullo.
check_nome_ricetta	Dominio	L'attributo "nome" della classe Ricetta deve contenere una stringa alfabetica.
descrizione VARCHAR(250) NOT NULL	Dominio	Impone che l'attributo "descrizione" della classe Ricetta sia una stringa alfanumerica <= 250 caratteri e che sia obbligatoriamente non nullo.
check_descrizione	Dominio	L'attributo "descrizione" della classe Ricetta deve contenere una stringa alfabetica.
tempo TIME NOT NULL	Dominio	L'attributo "tempo" della classe Ricetta deve contenere un orario (ore, minuti, secondi) valido e non nullo.

Vincolo	Tipo	Descrizione
nome .. PRIMARY KEY (Ingrediente)	Integrità dell'entità	Identifica univocamente ogni ingrediente. Garantisce l'unicità del nome e che esso non sia nullo.
check_nome_ingrediente	Dominio	L'attributo "nome" della classe Ingrediente deve contenere una stringa alfabetica.
unita_di_misura VARCHAR(20) NOT NULL	Dominio	Impone che l'attributo "unitàDiMisura" della classe Ingrediente sia una stringa alfanumerica <= 20 caratteri e che sia obbligatoriamente non nullo.
check_unita_di_misura	Dominio	L'attributo "unitàDiMisura" della classe Ingrediente deve contenere una stringa alfabetica
trg_check_ specializzazione	Interrelazionale	La relazione "specializzatoIn" può coinvolgere solo utenti con tipoUtente uguale a chef o chefStudente. Un utente con ruolo di solo Studente non può possedere specializzazioni.
trg_check_gestisce	Interrelazionale	La relazione "Gestisce" può coinvolgere solo utenti con tipoUtente uguale a chef o chefStudente. Un utente con ruolo di solo Studente non può gestire Corsi o Sessioni.
trg_check_max_ sessioni_per_corso	Interrelazionale	Impedisce l'inserimento o la modifica di tuple nella tabella Sessione qualora ciò comporti il superamento del valore "numSessioni" definito nella tabella Corso associata.

Vincolo	Tipo	Descrizione
trq_check_num_ sessioni_corso_update	Interrelazionale	Inibisce l'aggiornamento del attributo "numSessioni" nella tabella Corso verso un valore inferiore alla cardinalità attuale delle tuple correlate nella tabella Sessione, preservando la consistenza dei dati esistenti.

3.3 Schema Logico

Prima di procedere all'implementazione fisica del database, è necessario definire lo schema logico. Tale modello si ottiene traducendo le entità e le associazioni dello schema concettuale in tabelle vere e proprie.
Di seguito la notazione utilizzata per la lettura:

- Sottolineatura singola per le chiavi primarie;
- Doppia sottolineatura per le chiavi esterne;
- Carattere corsivo per le parti costituenti una chiave primaria composta.

3.3.1 Traduzione delle classi

Dalla trasformazione del diagramma delle classi deriva la seguente organizzazione delle tabelle relazionali:
Utente (email, nome, cognome, password, matricola, tipoUtente)

Corso (id_corso, dataInizio, nome, frequenza, numPartecipanti, numSessioni)

Sessione (id_sessione, oraInizio, numAderenti, teoria, TipoSessione, id_corso)
Chiavi esterne: id_corso → Corso.id_corso

Ricetta (id_ricetta, nome, descrizione, tempo)

Ingrediente (nome, unitàDiMisura)

Adesione (dataAdesione, *email, id_sessione*, email, id_sessione)
Chiavi esterne: id_sessione → Sessione.id_sessione
email → Utente.email

Iscrizione (dataIscrizione, stato, *email, id_corso*, email, id_corso)
Chiavi esterne: id_corso → Corso.id_corso
email → Utente.email

Richiede (*id_ricetta*, *nomeIngrediente*, id_ricetta, nomeIngrediente, quantitàNecessaria)
Chiavi esterne: id_ricetta → Ricetta.id_ricetta
nomeIngrediente → Ingrediente.nome

Specializzazione_Chef (*email_chef, specializzazione*, email_chef)

Chiave esterna: email_chef → Utente.email

Categoria_Corso (*id_corso, categoria*, id_corso)

Chiave esterna: id_corso → Corso.id_corso

Gestisce (*email_chef, id_corso*, email_chef, id_corso)

Chiavi esterne: id_corso → Corso.id_corso
email_chef → Utente.email

Prepara (*id_sessione, id_ricetta*, id_sessione, id_ricetta)

Chiavi esterne: id_ricetta → Ricetta.id_ricetta
id_sessione → Sessione.id_sessione

Vista_Fabbisogni_Sessione (*id_sessione, nomeIngrediente*, id_sessione, nomengrediente, quantitàTotale)

Chiavi esterne: nomeIngrediente → Ingrediente.nome
id_sessione → Sessione.id_sessione

3.3.2 Traduzione delle associazioni

Associazione	Implementazione
adesione *(Utente - Adesione)*	**Importazione Chiave Esterna:** L'entità *Adesione* (lato N) importa la chiave primaria di *Utente* (`id_utente`) come chiave esterna.
iscrizione *(Utente - Iscrizione)*	**Importazione Chiave Esterna:** L'entità *Iscrizione* importa la chiave primaria di *Utente* (`id_utente`) come chiave esterna.
gestisce *(Chef - Corso)*	**Importazione Chiave Esterna:** Data la cardinalità 1:N, la tabella *Corso* ospita la chiave esterna dello *Chef* (`id_chef`).
compone *(Corso - Sessione)*	**Importazione Chiave Esterna:** La tabella *Sessione* importa la chiave primaria di *Corso* (`id_corso`) come chiave esterna.
prepara *(Sessione - Ricetta)*	**Nuova Tabella (Ponte):** Data la cardinalità Molti-a-Molti, si crea la tabella di raccordo *Prepara* composta dalle chiavi esterne delle due entità (`id_sessione`, `id_ricetta`).
haBisogno *(Ricetta - Richiede)*	**Importazione Chiave Esterna:** La tabella *Richiede* importa la chiave primaria della *Ricetta* (`id_ricetta`).
necessita *(Richiede - Ingrediente)*	**Importazione Chiave Esterna:** La tabella *Richiede* importa la chiave primaria dell'*Ingrediente* (`id_ingrediente`).
per *(Adesione - Sessione)*	**Importazione Chiave Esterna:** La tabella *Adesione* importa la chiave primaria della *Sessione* (`id_sessione`).
relativa *(Iscrizione - Corso)*	**Importazione Chiave Esterna:** La tabella *Iscrizione* importa la chiave primaria del *Corso* (`id_corso`).
specializzatoIn *(Chef - Specializzazione)*	**Nuova Tabella:** La tabella *Specializzazione_ Chef* viene creata per gestire l'attributo multivalore, importando la chiave primaria dello chef (`email_chef`).
categorizzatoIn *(Corso - Categoria)*	**Nuova Tabella:** La tabella *Categoria_ Corso* viene creata per gestire l'attributo multivalore, importando la chiave primaria del corso (`id_corso`).

3.3.3 Schema logico finale

Di seguito viene riportato lo schema logico finale risultante, completo di chiavi primarie, esterne e attributi:

Utente	email, nome, cognome, password, matricola, tipoUtente
Corso	id_corso, dataInizio, nome, frequenza, numPartecipanti, numSessioni
Sessione	id_sessione, oraInizio, numAderenti, teoria, TipoSessione, id_corso
Ricetta	id_ricetta, nome, descrizione, tempo
Ingrediente	nome, unitàDiMisura
Adesione	dataAdesione, *email, id_sessione*, email, id_sessione
Iscrizione	dataIscrizione, stato, *email, id_corso*, email, id_corso
Richiede	quantitàNecessaria, *id_ricetta, nomeIngrediente*, id_ricetta, nomeIngrediente
Specializzazione_Chef	*email_chef, specializzazione*, email_chef
Categoria_Corso	*id_corso, categoria*, id_corso
Gestisce	*email_chef, id_corso*, email_chef, id_corso
Prepara	*id_sessione, id_ricetta*, id_sessione, id_ricetta
Vista_Fabbisogni_Sessione	*id_sessione*, *nomeIngrediente*, id_sessione, nomengrediente, quantitàTotale

4 Implementazione della Base di Dati

In questo capitolo viene illustrata la fase di implementazione fisica del database, derivata direttamente dallo schema logico precedentemente definito. La codifica è stata realizzata utilizzando il linguaggio SQL (dialetto PostgreSQL), ponendo particolare attenzione alla robustezza e alla consistenza dei dati.
Oltre alla definizione delle strutture tabellari, una parte centrale dello sviluppo ha riguardato la traduzione delle regole di business in vincoli di integrità attivi. Sono stati implementati domini personalizzati, vincoli CHECK per la validazione dei formati e, soprattutto, TRIGGER per garantire il rispetto delle cardinalità minime e delle logiche di aggiornamento automatico.

4.1 Creazione e pulizia schema

```sql
-- PULIZIA AMBIENTE E CREAZIONE SCHEMA
DROP SCHEMA IF EXISTS uninafoodlab CASCADE;
CREATE SCHEMA uninafoodlab;
GRANT ALL ON SCHEMA uninafoodlab TO public;
SET search_path TO uninafoodlab, public;
```

4.2 Creazione delle enumerazioni

```sql
-- CREAZIONE TIPI ENUM
CREATE TYPE tipo_utente_enum AS ENUM ('chef', 'studente', 'chefStudente');
CREATE TYPE tipo_sessione_enum AS ENUM ('presenza', 'online');
```

4.3 Creazione delle tabelle

4.3.1 Tabella Utente

```sql
-- Creazione della tabella UTENTE
CREATE TABLE Utente (
    email VARCHAR(255) PRIMARY KEY,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    password VARCHAR(255) NOT NULL,
    matricola VARCHAR(20) UNIQUE, -- Obbligatoria solo se studente
    tipo_utente tipo_utente_enum NOT NULL
);
```

4.3.2 Tabella Corso

```sql
-- Creazione della tabella CORSO
CREATE TABLE Corso (
    id_corso INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    data_inizio DATE NOT NULL,
    nome VARCHAR(50) NOT NULL,
    frequenza VARCHAR(50) NOT NULL,
    num_partecipanti INTEGER DEFAULT 0, -- Aggiornato da Trigger
    num_sessioni INTEGER NOT NULL
);
```

4.3.3 Tabella Sessione

```sql
-- Creazione della tabella SESSIONE
CREATE TABLE Sessione (
    id_sessione INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    ora_inizio TIMESTAMP NOT NULL,
    num_aderenti INTEGER DEFAULT 0, -- Aggiornato da Trigger
    teoria VARCHAR(255),
    tipo_sessione tipo_sessione_enum NOT NULL,
    id_corso INTEGER NOT NULL,
    FOREIGN KEY (id_corso) REFERENCES Corso(id_corso) ON DELETE CASCADE
);
```

4.3.4 Tabella Ricetta

```
-- Creazione della tabella RICETTA
CREATE TABLE Ricetta (
    id_ricetta INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    nome VARCHAR(50) NOT NULL,
    descrizione VARCHAR(250) NOT NULL,
    tempo TIME NOT NULL
);
```

4.3.5 Tabella Ingrediente

```
-- Creazione della tabella INGREDIENTE
CREATE TABLE Ingrediente (
    nome VARCHAR(50) PRIMARY KEY,
    unita_di_misura VARCHAR(20) NOT NULL
);
```

4.3.6 Tabella Iscrizione

```
-- Creazione della tabella ISCRIZIONE (Utente -> Corso)
CREATE TABLE Iscrizione (
    email_utente VARCHAR(255),
    id_corso INTEGER,
    data_iscrizione DATE DEFAULT CURRENT_DATE,
    stato VARCHAR(20) DEFAULT 'Attivo',
    PRIMARY KEY (email_utente, id_corso),
    FOREIGN KEY (email_utente) REFERENCES Utente(email) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (id_corso) REFERENCES Corso(id_corso) ON UPDATE CASCADE ON DELETE CASCADE
);
```

4.3.7 Tabella Adesione

```sql
-- Creazione della tabella ADESIONE (Utente -> Sessione Pratica)
CREATE TABLE Adesione (
    email_utente VARCHAR(255),
    id_sessione INTEGER,
    data_adesione DATE DEFAULT CURRENT_DATE,
    PRIMARY KEY (email_utente, id_sessione),
    FOREIGN KEY (email_utente) REFERENCES Utente(email) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (id_sessione) REFERENCES Sessione(id_sessione) ON UPDATE CASCADE ON DELETE CASCADE
);
```

4.3.8 Tabella Gestisce

```sql
-- Creazione della tabella GESTISCE (Chef <-> Corso)
-- Regola: uno chef può gestire 0..N corsi; un corso deve essere gestito da >=1 chef.
CREATE TABLE Gestisce (
    email_chef VARCHAR(255),
    id_corso INTEGER,
    PRIMARY KEY (email_chef, id_corso),
    FOREIGN KEY (email_chef) REFERENCES Utente(email) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (id_corso) REFERENCES Corso(id_corso) ON UPDATE CASCADE ON DELETE CASCADE
);
```

4.3.9 Tabella Prepara

```sql
-- Creazione della tabella PREPARA (Sessione <-> Ricetta)
-- Regola: una sessione prepara 1..N ricette e una ricetta può essere preparata in 1..N sessioni.
CREATE TABLE Prepara (
    id_sessione INTEGER,
    id_ricetta INTEGER,
    PRIMARY KEY (id_sessione, id_ricetta),
    FOREIGN KEY (id_sessione) REFERENCES Sessione(id_sessione) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (id_ricetta) REFERENCES Ricetta(id_ricetta) ON UPDATE CASCADE ON DELETE CASCADE
);
```

4.3.10 Tabella Richiede

```sql
-- Creazione della tabella RICHIEDE (Ingredienti per Ricetta)
CREATE TABLE Richiede (
    id_ricetta INTEGER,
    nome_ingrediente VARCHAR(50),
    quantita_necessaria DECIMAL(10,2),
    PRIMARY KEY (id_ricetta, nome_ingrediente),
    FOREIGN KEY (id_ricetta) REFERENCES Ricetta(id_ricetta) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (nome_ingrediente) REFERENCES Ingrediente(nome) ON UPDATE CASCADE ON DELETE CASCADE
);
```

4.3.11 Tabella Categoria_Corso

```
-- Creazione della tabella CATEGORIA_CORSO (1:N)
CREATE TABLE Categoria_Corso (
    id_corso INTEGER,
    categoria VARCHAR(50) NOT NULL,
    PRIMARY KEY (id_corso, categoria),
    FOREIGN KEY (id_corso) REFERENCES Corso(id_corso) ON UPDATE CASCADE ON DELETE CASCADE
);
```

4.3.12 Tabella Specializzazione_Chef

```
-- Creazione della tabella SPECIALIZZAZIONE_CHEF (1:N)
CREATE TABLE Specializzazione_Chef (
    email_chef VARCHAR(255),
    specializzazione VARCHAR(50) NOT NULL,
    PRIMARY KEY (email_chef, specializzazione),
    FOREIGN KEY (email_chef) REFERENCES Utente(email) ON UPDATE CASCADE ON DELETE CASCADE
);
```

4.4 Creazione dei constraints

4.4.1 Su Corso

```
-- Corso
--  Numero sessioni positivo, nome con caratteri validi
ALTER TABLE Corso ADD CONSTRAINT check_num_sessioni CHECK (num_sessioni > 0);
ALTER TABLE Corso ADD CONSTRAINT check_nome_corso CHECK (nome ~ '^[A-Za-zÀ-ÿ ''-]+$');
```

4.4.2 Su Utente

```
--Utente
-- Email, nome, cognome, password e coerenza dati in base al ruolo
ALTER TABLE Utente ADD CONSTRAINT check_email CHECK (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');
ALTER TABLE Utente ADD CONSTRAINT check_nome_utente CHECK (nome ~ '^[A-Za-zÀ-ÿ ''-]+$');
ALTER TABLE Utente ADD CONSTRAINT check_cognome CHECK (cognome ~ '^[A-Za-zÀ-ÿ ''-]+$');
ALTER TABLE Utente ADD CONSTRAINT check_password CHECK (
    LENGTH(password) >= 8 AND password ~ '[A-Z]' AND password ~ '[a-z]'
    AND password ~ '[0-9]' AND password ~ '[^a-zA-Z0-9]'
);
ALTER TABLE Utente ADD CONSTRAINT check_ruolo_dati CHECK (
    ((tipo_utente IN ('studente', 'chefStudente')) AND matricola IS NOT NULL)
    OR
    (tipo_utente = 'chef' AND matricola IS NULL)
);
```

4.4.3 Su Ingrediente

```
-- Ingrediente
-- Nome e unità di misura con caratteri validi
ALTER TABLE Ingrediente ADD CONSTRAINT check_nome_ingrediente CHECK (nome ~ '^[A-Za-zÀ-ÿ ''-]+$');
ALTER TABLE Ingrediente ADD CONSTRAINT check_unita_di_misura CHECK (unita_di_misura ~ '^[A-Za-z]+$');
```

4.4.4 Su Ricetta

```
-- Ricetta
-- Nome e descrizione con caratteri validi
ALTER TABLE Ricetta ADD CONSTRAINT check_nome_ricetta CHECK (nome ~ '^[A-Za-zÀ-ÿ ''-]+$');
ALTER TABLE Ricetta ADD CONSTRAINT check_descrizione_ricetta CHECK (descrizione ~ '^[A-Za-zÀ-ÿ ''-]+$');
```

4.4.5 Su Sessione

```
-- Sessione
-- Numero aderenti non negativo, coerenza dati in base al tipo di sessione
ALTER TABLE Sessione ADD CONSTRAINT check_num_aderenti CHECK (num_aderenti >= 0);
ALTER TABLE Sessione ADD CONSTRAINT check_logica_sessione CHECK (
    (tipo_sessione = 'online'
        AND teoria IS NOT NULL
        AND LENGTH(teoria) > 0
        AND LENGTH(teoria) <= 255
        AND teoria ~ '^[A-Za-zÀ-ÿ0-9 ''-]+$'

    )
    OR
    (tipo_sessione = 'presenza'
        AND teoria IS NULL
    )
);
```

4.4.6 Su Richiede

```
-- Richiede
-- Quantità necessaria positiva
ALTER TABLE Richiede ADD CONSTRAINT check_quantita_necessaria CHECK (quantita_necessaria > 0);
```

4.5 Triggers per il controllo dei vincoli

4.5.1 Su Adesione

```
-- Associazione: Adesione ammessa solo per studente/chefStudente e solo per sessioni di tipo presenza
CREATE OR REPLACE FUNCTION check_adesione() RETURNS TRIGGER AS $$
DECLARE
    t tipo_utente_enum;
    ts tipo_sessione_enum;
    corso_sessione INTEGER;
BEGIN
    SELECT tipo_utente INTO t FROM Utente WHERE email = NEW.email_utente;
    IF t IS NULL THEN
        RAISE EXCEPTION 'Utente % inesistente', NEW.email_utente;
    END IF;
    IF t = 'chef' THEN
        RAISE EXCEPTION 'Solo studente/chefStudente possono aderire alle sessioni (%).', NEW.email_utente;
    END IF;

    SELECT tipo_sessione, id_corso INTO ts, corso_sessione
    FROM Sessione
    WHERE id_sessione = NEW.id_sessione;
    IF ts IS NULL THEN
        RAISE EXCEPTION 'Sessione % inesistente', NEW.id_sessione;
    END IF;
    IF ts <> 'presenza' THEN
        RAISE EXCEPTION 'Adesione ammessa solo per sessioni in presenza (sessione=%).', NEW.id_sessione;
    END IF;

    IF NOT EXISTS (
        SELECT 1
        FROM Iscrizione i
        WHERE i.email_utente = NEW.email_utente
          AND i.id_corso = corso_sessione
    ) THEN
        RAISE EXCEPTION 'Adesione non consentita: utente % non iscritto al corso % (sessione=%).',
        NEW.email_utente, corso_sessione, NEW.id_sessione;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_adesione
BEFORE INSERT OR UPDATE ON Adesione
FOR EACH ROW EXECUTE FUNCTION check_adesione();
```

4.5.2 Su Iscrizione

```
-- Associazione: Iscrizione ammessa solo per studente/chefStudente
CREATE OR REPLACE FUNCTION check_iscrizione() RETURNS TRIGGER AS $$
DECLARE
    t tipo_utente_enum;
BEGIN
    SELECT tipo_utente INTO t FROM Utente WHERE email = NEW.email_utente;
    IF t IS NULL THEN
        RAISE EXCEPTION 'Utente % inesistente', NEW.email_utente;
    END IF;
    IF t = 'chef' THEN
        RAISE EXCEPTION 'Solo studente/chefStudente possono iscriversi ai corsi (%).', NEW.email_utente;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_iscrizione
BEFORE INSERT OR UPDATE ON Iscrizione
FOR EACH ROW EXECUTE FUNCTION check_iscrizione();
```

4.5.3 Su Compone

```
-- Impedisce di creare/spostare una Sessione su un corso che ha già raggiunto il limite.
CREATE OR REPLACE FUNCTION check_max_sessioni_per_corso() RETURNS TRIGGER AS $$
DECLARE
    limite INTEGER;
    conteggio INTEGER;
BEGIN
    SELECT num_sessioni INTO limite
    FROM Corso
    WHERE id_corso = NEW.id_corso;

    IF limite IS NULL THEN
        RAISE EXCEPTION 'Corso % inesistente', NEW.id_corso;
    END IF;

    SELECT COUNT(*) INTO conteggio
    FROM Sessione
    WHERE id_corso = NEW.id_corso;

    IF conteggio > limite THEN
        RAISE EXCEPTION 'Limite sessioni superato per corso % (sessioni=% / limite=%).', NEW.id_corso, conteggio, limite;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_max_sessioni_per_corso
AFTER INSERT OR UPDATE OF id_corso ON Sessione
FOR EACH ROW EXECUTE FUNCTION check_max_sessioni_per_corso();
```

```
-- Impedisce di ridurre num_sessioni sotto il numero di Sessioni già istanziate.
CREATE OR REPLACE FUNCTION check_num_sessioni_corso_update() RETURNS TRIGGER AS $$
DECLARE
    conteggio INTEGER;
BEGIN
    SELECT COUNT(*) INTO conteggio
    FROM Sessione
    WHERE id_corso = NEW.id_corso;

    IF NEW.num_sessioni < conteggio THEN
        RAISE EXCEPTION 'num_sessioni non può essere < delle sessioni già create (corso=%; sessioni=%; nuovo_limite=%).',
            NEW.id_corso, conteggio, NEW.num_sessioni;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_num_sessioni_corso_update
BEFORE UPDATE OF num_sessioni ON Corso
FOR EACH ROW EXECUTE FUNCTION check_num_sessioni_corso_update();
```

4.6 Triggers per la gestione dei dati derivabili

4.6.1 Di num_partecipanti

```
-- Creazione del trigger: Aggiorna numero partecipanti Corso
CREATE OR REPLACE FUNCTION aggiorna_partecipanti() RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'INSERT') THEN
        UPDATE Corso SET num_partecipanti = num_partecipanti + 1 WHERE id_corso = NEW.id_corso;
        RETURN NEW;
    ELSIF (TG_OP = 'DELETE') THEN
        UPDATE Corso SET num_partecipanti = num_partecipanti - 1 WHERE id_corso = OLD.id_corso;
        RETURN OLD;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_partecipanti AFTER INSERT OR DELETE ON Iscrizione
FOR EACH ROW EXECUTE FUNCTION aggiorna_partecipanti();
```

4.6.2 Di num_aderenti

```sql
-- Creazione del trigger: Aggiorna numero aderenti Sessione
CREATE OR REPLACE FUNCTION aggiorna_aderenti() RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'INSERT') THEN
        UPDATE Sessione SET num_aderenti = num_aderenti + 1 WHERE id_sessione = NEW.id_sessione;
        RETURN NEW;
    ELSIF (TG_OP = 'DELETE') THEN
        UPDATE Sessione SET num_aderenti = num_aderenti - 1 WHERE id_sessione = OLD.id_sessione;
        RETURN OLD;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_aderenti AFTER INSERT OR DELETE ON Adesione
FOR EACH ROW EXECUTE FUNCTION aggiorna_aderenti();
```

4.7 Creazione viste

```sql
CREATE VIEW Vista_Fabbisogni_Sessione AS
SELECT
    -- Chiavi di identificazione
    Prepara.id_sessione,
    Richiede.nome_ingrediente,

    -- Calcolo: (Dose Unitaria * Numero Studenti), sommato per ogni ricetta
    SUM(Richiede.quantita_necessaria * Sessione.num_aderenti) AS quantita_totale

FROM
    Prepara

    -- 1. JOIN con la tabella RICHIEDE (per sapere le dosi unitarie)
    JOIN Richiede ON Prepara.id_ricetta = Richiede.id_ricetta

    -- 2. JOIN con SESSIONE (num_aderenti è mantenuto dal trigger)
    JOIN Sessione ON Sessione.id_sessione = Prepara.id_sessione

GROUP BY
    Prepara.id_sessione,
    Richiede.nome_ingrediente;
```