**Corso di Laurea in Informatica - Università degli Studi di Napoli Federico II**
**A.A. 2025/2026**

# UninaFoodLab

**Calone Francesco N86005555**

**D'Angelo Mario N86005477**

Codice gruppo: **OOBD39**

Insegnamento di Basi di Dati I

# Indice

# 1 Introduzione

## 1.1 Descrizione del progetto

Il seguente progetto, denominato UninaFoodLab, nasce con l'obiettivo di progettare e implementare un sistema informativo per la gestione di corsi di cucina tematici. Il sistema si propone di supportare gli chef nella creazione e gestione di corsi articolati in sessioni teoriche e pratiche, facilitando al contempo l'iscrizione e la partecipazione degli utenti.
La documentazione che segue illustra in dettaglio tutte le fasi della progettazione, analizzando le scelte architetturali, i modelli concettuali e logici, e le soluzioni tecniche adottate per garantire la correttezza, la coerenza e l'efficienza del sistema.

## 1.2 Analisi dei Requisiti rilenvanti per il Database

I seguenti requisiti funzionali sono stati analizzati ai fini della progettazione della base di dati. Essi definiscono le informazioni da memorizzare e le relazioni tra le entità del sistema.

1. Registrazione e autenticazione degli utenti.
   - Gli utenti devono poter creare un account con email e password.
   - Gli utenti devono poter effettuare il login.
2. Gestione dei profili dei partecipanti.
   - I partecipanti devono poter visualizzare e modificare le proprie informazioni personali.

- I partecipanti devono poter visualizzare i propri corsi.
- I partecipanti devono poter visualizzare i propri dati di pagamento.
- I partecipanti devono poter visualizzare le proprie sessioni pratiche.

3. Creazione e gestione dei corsi da parte degli chef.
   - Gli chef devono poter creare nuovi corsi.
   - Gli chef devono poter definire le sessioni teoriche e pratiche.
   - Gli chef devono poter modificare.
   - Gli chef possono aggiungere delle ricette alle sessioni pratiche.
   - Gli chef devono poter visualizzare il numero di ingredienti necessari per la sessione pratica
   - Gli chef devono poter visualizzare il loro report di guadagno e attività.
4. Iscrizione ai corsi da parte dei partecipanti.
   - I partecipanti devono poter consultare l'elenco dei corsi disponibili.
   - I partecipanti devono poter iscriversi a un corso.
   - I partecipanti devono dare conferma per la partecipazione alle sessioni pratiche.
5. Gestione delle presenze e dei pagamenti.
   - Gli chef devono poter registrare la presenza alle sessioni.
   - Il sistema deve gestire i pagamenti dei partecipanti.
   - I partecipanti devono poter vedere le proprie carte di credito.

# 2 Progettazione concettuale

## 2.1 Introduzione

Il modello concettuale rappresenta la struttura logica del database, definendo le entità, gli attributi e le relazioni tra di esse. In questa fase, si è proceduto a identificare le principali entità del sistema e a stabilire le relazioni che le collegano, garantendo così una visione chiara e coerente delle informazioni da gestire.

## 2.2 UML non ristrutturato

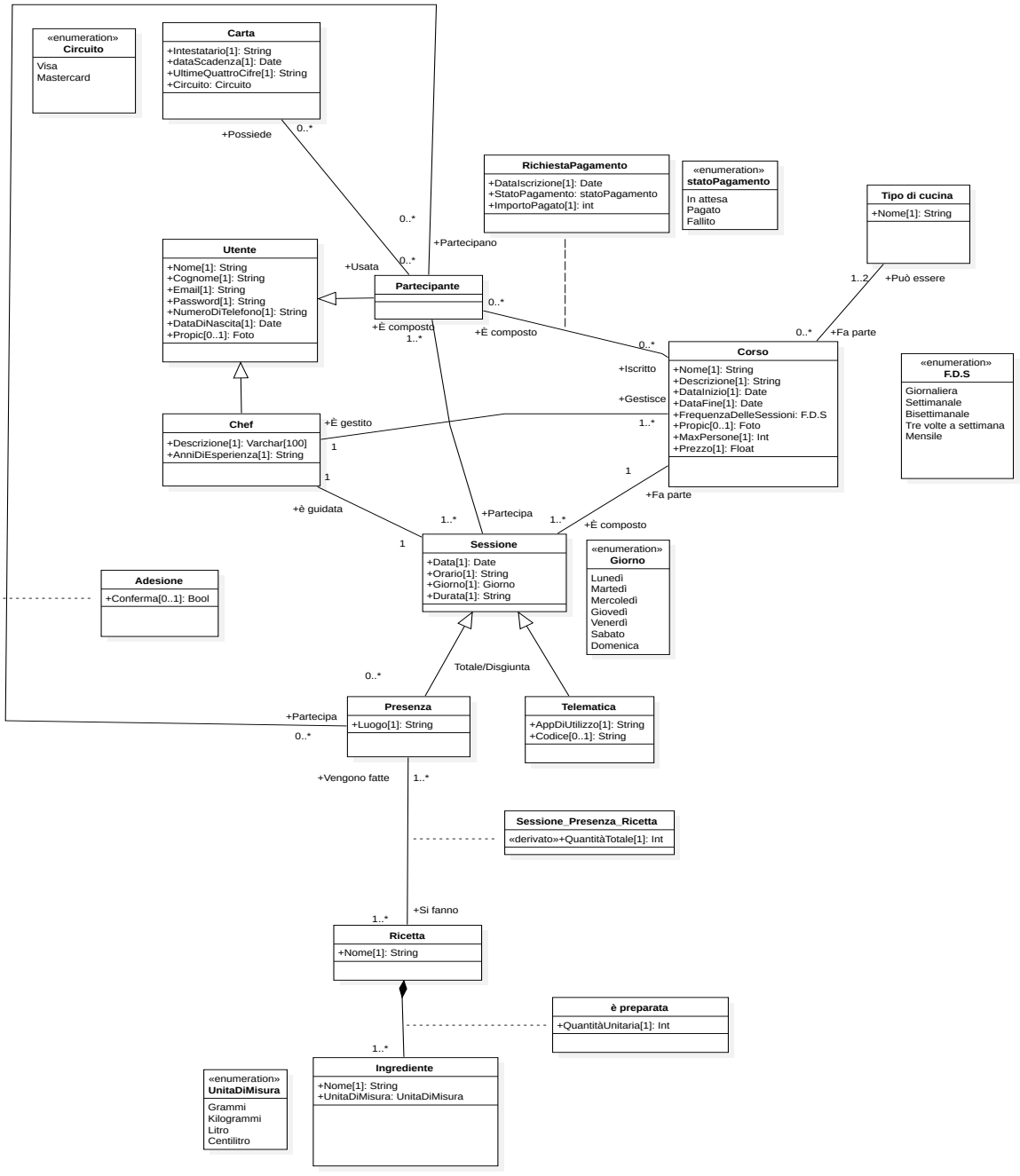


Figure 1: Diagramma UML non ristrutturato

### 2.2.1 Entità principali

Le entità principali identificate nel sistema sono:

- **Utente**: L'utente rappresenta il soggetto fruitore del sistema, che può iscriversi ai corsi e partecipare alle sessioni. I principali attributi includono nome, cognome, email, password, data di nascita e una foto di profilo. Ogni utente può essere associato a una o più carte di pagamento e può diventare partecipante a diversi corsi.
- **Chef**: Lo chef è un utente con il ruolo specifico di organizzare corsi. Ogni chef dispone di una descrizione e di un numero di anni di esperienza. Un chef può gestire più corsi, ma ogni corso è gestito da un solo chef.
- **Corso**: Il corso è l'entità centrale del sistema e rappresenta una proposta didattica su un tema gastronomico specifico. Contiene attributi quali nome, descrizione, identificativo, data di inizio/fine, frequenza delle sessioni, prezzo, massimo persone che possono iscriversi, immagine di copertina e tipo di cucina (gestito da un'altra entità). Ogni corso è composto da più sessioni e prevede una relazione molti-a-molti con i partecipanti.
- **Sessione**: Ogni corso è articolato in una o più sessioni, ciascuna delle quali ha una data, un orario, un insieme di giorni della settimana in cui si svolge, e una durata. Le sessioni sono specializzate in due sottotipi mutuamente esclusivi:
  - **Presenza**: Con attributi come luogo e attrezzature richieste.
  - **Telematica**: Con attributi relativi all'app utilizzata e al codice di accesso.
- **Partecipante e Adesione**: La partecipazione ai corsi è modellata tramite l'entità Partecipante, che collega utenti e corsi. La partecipazione a sessioni pratiche richiede un'adesione esplicita, rappresentata dall'entità Adesione, che contiene un attributo booleano di conferma.
- **Ricetta e Ingredientemento**: Ogni sessione pratica può includere la preparazione di una o più ricette. Ogni ricetta è composta da uno o più ingredienti, ciascuno dei quali ha un nome, e un'unità di misura (enumerata). La relazione tra Ricetta e Ingrediente è associativa e include l'attributo QuantitàTotale e QuantitàUnitaria, utile per calcolare la quantità necessaria in base alle adesioni.
- **Carta e RichiestaPagamento**: I partecipanti possono associare al proprio profilo una o più carte di pagamento, appartenenti a un circuito specificato tramite enumerazione (Visa, Mastercard). Le richieste di pagamento sono entità separate, con data, stato (in attesa, pagato, fallito) e importo.

### 2.2.2 Gerarchie e generalizzazioni

Nel modello concettuale, sono state identificate le seguenti gerarchie e generalizzazioni:

- **Sessione**: Le sessioni sono suddivise in due sottotipi: *Presenza* e *Telematica*. Questa specializzazione consente di gestire le specificità di ciascun tipo di sessione, come il luogo e le attrezzature per le sessioni in presenza, e l'app utilizzata e il codice di accesso per quelle telematiche.
- **Utente**: L'entità Utente può essere specializzata in due sottotipi: *Partecipante* e *Chef*. Questa distinzione permette di gestire le diverse funzionalità e attributi associati a ciascun ruolo nel sistema.

Entrambe le specializzazioni sono totali e disgiunte, di conseguenza ogni istanza di Sessione sia esclusivamente di uno dei due tipi e che ogni Utente sia o un Partecipante o uno Chef, ma non entrambi contemporaneamente.

### 2.2.3 Relazioni tra le entità

Le relazioni tra le entità sono state definite come segue:

- **Utente - Partecipante**: Un utente può essere un partecipante a più corsi, e ogni corso può avere più partecipanti. Questa relazione è molti-a-molti.
- **Chef - Corso**: Ogni chef può gestire più corsi, ma ogni corso è associato a un solo chef. Questa relazione è uno-a-molti.
- **Corso - Sessione**: Un corso può avere più sessioni, ma ogni sessione appartiene a un solo corso. Questa relazione è uno-a-molti.
- **Sessione - Partecipante**: Ogni partecipante può aderire a più sessioni pratiche, e ogni sessione può avere più partecipanti. Questa relazione è molti-a-molti, mediata dall'entità Adesione.
- **Corso - Ricetta**: Ogni corso può includere più ricette, e ogni ricetta può essere associata a più corsi. Questa relazione è molti-a-molti.
- **Ricetta - Ingrediente**: Ogni ricetta può includere più ingredienti, e ogni ingrediente può essere utilizzato in più ricette. Questa relazione è molti-a-molti, mediata dall'attributo QuantitàTotale.
- **Utente - Carta**: Un utente può avere più carte di pagamento associate al proprio profilo. Questa relazione è uno-a-molti.
- **Ricetta - Ingrediente**: Ogni ricetta può essere associata a più ingredienti, e ogni ingrediente può essere utilizzato in più ricette. Questa relazione è una composizione, mediata dall'attributo QuantitàTotale.

### 2.2.4 Motivazione delle scelte progettuali

Le scelte progettuali sono state guidate dalla necessità di garantire una rappresentazione chiara e coerente delle informazioni, facilitando la gestione dei corsi, delle sessioni e delle partecipazioni. La specializzazione delle sessioni in Presenza e Telematica consente di gestire le specificità di ciascun tipo di sessione, mentre la distinzione tra Partecipante e Chef permette di differenziare i ruoli degli utenti nel sistema. Inoltre, l'uso di relazioni molti-a-molti per gestire le adesioni alle sessioni pratiche e le associazioni tra ricette e ingredienti garantisce flessibilità e scalabilità nel modello.

### 2.2.5 Diagramma ER

Il diagramma ER (Entity-Relationship) rappresenta graficamente le entità principali del sistema, i loro attributi e le relazioni tra di esse, fornendo una visione sintetica e formale della struttura informativa del database.

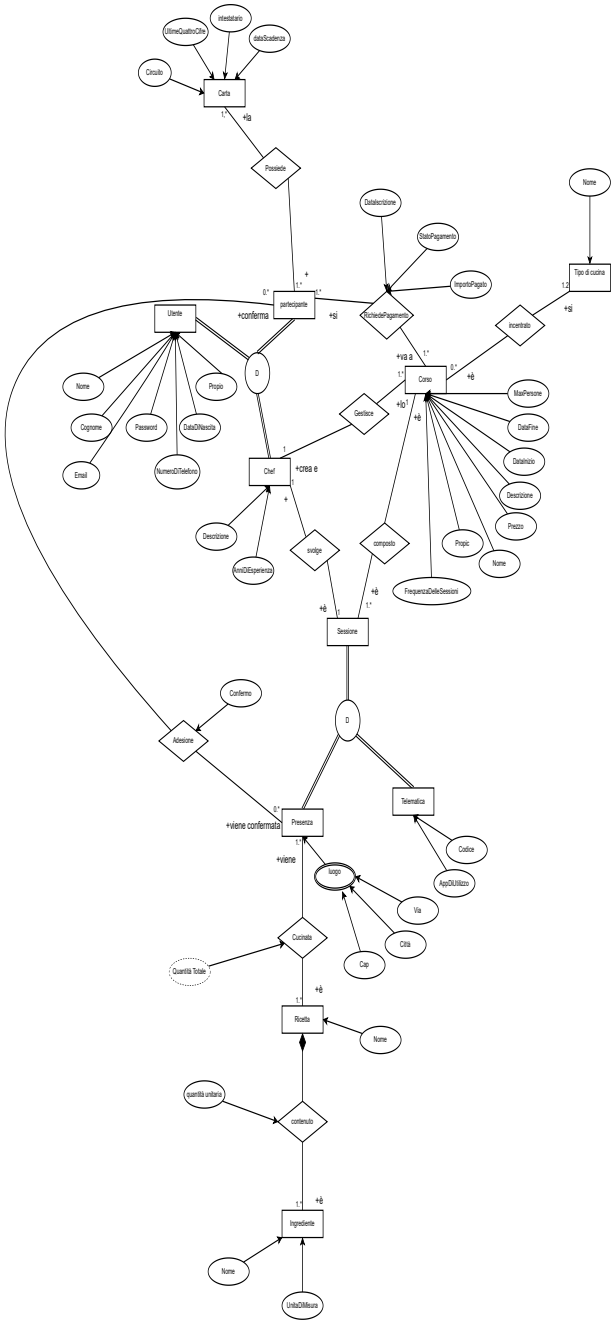


Figure 2: Diagramma ER del sistema

## 2.3 UML ristrutturato

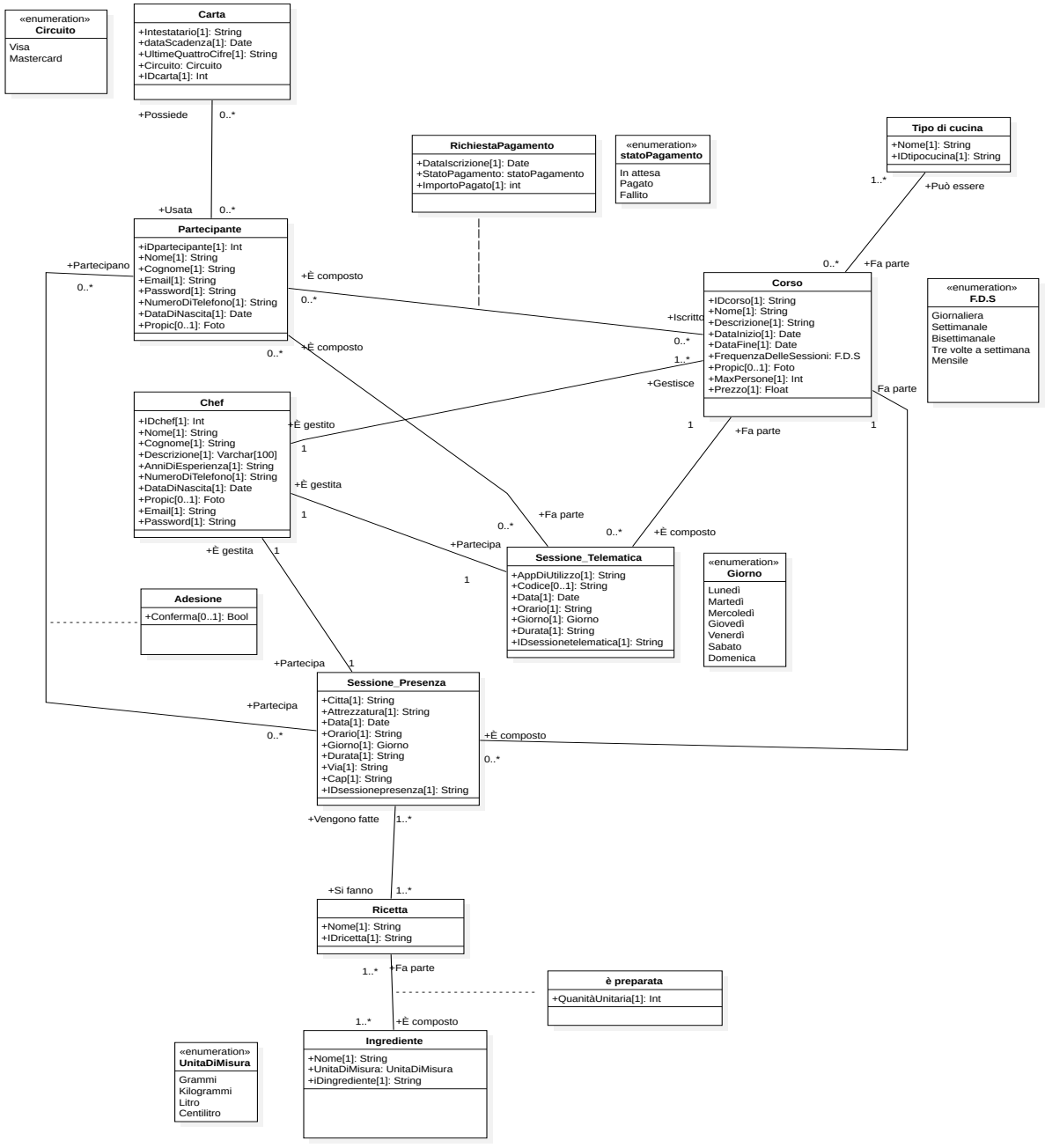


Figure 3: Diagramma UML ristrutturato del sistema

### 2.3.1 Modifiche rispetto al modello non ristrutturato

Il modello ristrutturato presenta alcune modifiche rispetto al modello non ristrutturato, adattandolo a un database relazionale. Le principali modifiche includono:

- **Rimozione delle gerarchie**: Le gerarchie tra le entità sono state rimosse, trasformando le specializzazioni in entità separate. Ad esempio, le sessioni di tipo Presenza e Telematica sono state trasformate in due entità distinte, mantenendo gli attributi specifici per ciascun tipo.
- **Attributi composti e multivalore**: Gli attributi composti e multivalore sono stati normalizzati. Ad esempio, l'attributo *Luogo* è stato suddiviso in attributi separati per *Città*, *Indirizzo* e *Cap*.
- **Aggiunta di chiavi primarie e esterne**: Ogni entità ha una chiave primaria univoca, e le relazioni tra le entità sono definite tramite chiavi esterne, garantendo l'integrità referenziale del database.

### 2.3.2 Comportamento del modello ristrutturato con le modifiche

Il modello ristrutturato mantiene le proprietà di specializzazione totale e disgiunta, adattandosi alle modifiche apportate. In particolare:

- **Specializzazione totale e disgiunta**: Le entità separate per le sessioni di tipo Presenza e Telematica garantiscono che ogni sessione appartenga esclusivamente a uno dei due tipi, rispettando la disgiunzione. Inoltre, ogni utente è classificato come Partecipante o Chef, assicurando che la specializzazione sia totale e disgiunta.
- **Integrità referenziale**: L'uso di chiavi primarie, esterne e l'aggiunta di chiavi surrogate garantisce che le relazioni tra le entità siano coerenti e che non si verifichino violazioni di integrità nel database.

### 2.3.3 Motivazioni delle scelte del modello ristrutturato

Le motivazioni alla base delle scelte progettuali del modello ristrutturato derivano dalla necessità di garantire una rappresentazione chiara, coerente e scalabile delle informazioni, adattandosi alle esigenze di un database relazionale. Una delle principali considerazioni riguarda la gestione delle specializzazioni totali e disgiunte, che hanno permesso di semplificare il modello concettuale senza perdere la coerenza logica.

Nel modello ristrutturato, le specializzazioni totali e disgiunte sono state gestite trasformando la classe generale in entità separate per ciascuna specializzazione. Questo approccio consente di incorporare gli attributi della classe generale direttamente nelle entità specializzate, eliminando la necessità di mantenere una gerarchia tra le entità. Ad esempio, nel caso delle sessioni, la classe generale "**Sessione**" è stata suddivisa in due entità distinte: "**Sessione_Presenza**" e "**Sessione_Telematica**". Ogni entità specializzata include gli attributi specifici del proprio tipo, come il luogo e le attrezzature richieste per le sessioni in presenza, e l'app utilizzata e il codice di accesso per quelle telematiche. In questo modo, si garantisce che ogni sessione appartenga esclusivamente a uno dei due tipi, rispettando la disgiunzione, e che tutte le sessioni siano rappresentate nel modello, rispettando la totalità, facilitando anche le associazioni tra le varie entità.

La stessa logica è stata applicata alla specializzazione dell'entità "Utente", suddivisa in "Partecipante" e "Chef". Ogni utente è classificato come Partecipante o Chef, ma non può appartenere a entrambe le categorie contemporaneamente. Gli attributi comuni agli utenti, come nome, cognome,

email e data di nascita, sono stati incorporati direttamente nelle entità specializzate, mentre gli attributi specifici, come la descrizione e gli anni di esperienza per gli Chef, sono stati aggiunti solo alla rispettiva entità. Questo approccio semplifica la gestione delle entità nel database, eliminando la necessità di un'entità generale "Utente" e garantendo che la specializzazione rimanga totale e disgiunta.

Per quanto riguarda le associazioni tra le entità, il modello ristrutturato utilizza chiavi primarie e chiavi esterne per definire le relazioni, garantendo l'integrità referenziale del database. Ad esempio, la relazione tra "Corso" e "Sessione" è uno-a-molti, con l'identificativo del corso utilizzato come chiave esterna nelle entità "Presenza" e "Telematica". Questo approccio consente di mantenere la coerenza delle informazioni e di semplificare le operazioni di query e aggiornamento. Analogamente, la relazione molti-a-molti tra "Sessione" e "Partecipante" è mediata dall'entità "Adesione", che include un attributo booleano di conferma per gestire la partecipazione alle sessioni pratiche.

Un altro esempio significativo riguarda la relazione tra "Ricetta" e "Ingrediente". Nel modello ristrutturato, questa relazione è stata normalizzata utilizzando un'entità associativa che include l'attributo "QuantitàTotale". Questo approccio consente di calcolare facilmente la quantità necessaria di ciascun ingrediente in base alle adesioni alle sessioni pratiche, garantendo flessibilità e scalabilità nel modello.

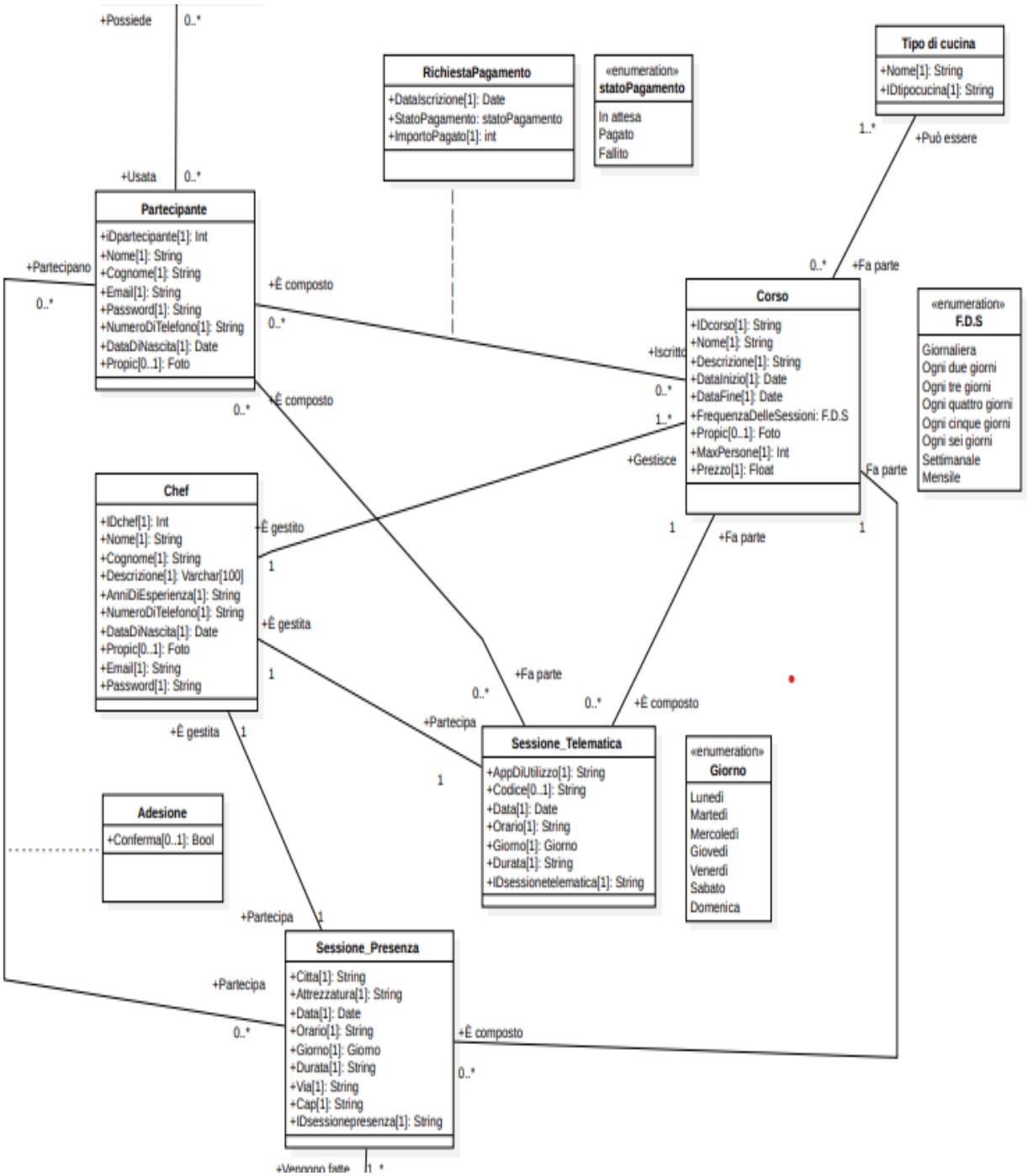


Figure 4: Scelte ristrutturato

## 2.4 Dizionari

### 2.4.1 Dizionario delle Classi (Entità)

**Entità Principali - Parte 1**

Dizionario delle Classi (Entità) - Parte 1

| Classe | Descrizione | Attributi |
|---|---|---|
| CARTA | Rappresenta una carta di pagamento associata a un partecipante. | IdCarta (PK), Intestatario, DataScadenza, UltimeQuattroCifre, Circuito |
| PARTECIPANTE | Utente che si iscrive e partecipa ai corsi di cucina. | IdPartecipante (PK), Nome, Cognome, Email, Password, DataDiNascita, Propic |
| POSSIEDE | Entità associativa che rappresenta il possesso di carte di pagamento da parte dei partecipanti. | IdPartecipante (FK, PK), IdCarta (FK, PK) |
| CHEF | Utente specializzato nell'organizzazione e gestione dei corsi di cucina. | IdChef (PK), Nome, Cognome, Email, Password, DataDiNascita, Descrizione, Propic, AnniDiEsperienza |
| CORSO | Rappresenta un corso di cucina tematico con tutte le sue caratteristiche. | IdCorso (PK), Nome, Descrizione, DataInizio, DataFine, FrequenzaDelleSessioni, Propic, MaxPersone, Prezzo, IdChef (FK) |
| RICHIESTA-PAGAMENTO | Gestisce le richieste di pagamento per l'iscrizione ai corsi. | DataRichiesta, StatoPagamento, ImportoPagato, IdCorso (FK), IdPartecipante (FK) |
| TIPODICUCINA | Definisce le tipologie di cucina disponibili per i corsi. | IDTipoCucina (PK), Nome |
| TIPODICUCINA_-CORSO | Entità associativa che collega i corsi alle tipologie di cucina. | IDTipoCucina (FK, PK), IDCorso (FK, PK) |

**Entità Principali - Parte 2**

Dizionario delle Classi (Entità) - Parte 2

| Classe | Descrizione | Attributi |
|---|---|---|
| SESSIONE_-PRESENZA | Rappresenta una sessione di corso che si svolge fisicamente in presenza. | IdSessionePresenza (PK), Giorno, Data, Orario, Durata, Citta, Via, Cap, Attrezzatura, Descrizione, IDCorso (FK), IDChef (FK) |
| SESSIONE_-TELEMATICA | Rappresenta una sessione di corso che si svolge online in modalità telematica. | IdSessioneTelematica (PK), Applicazione, CodiceChiamata, Data, Orario, Durata, Giorno, Descrizione, IDCorso (FK), IdChef (FK) |
| PARTECIPANTE_-SESSIONETELEMATICA | Entità associativa per la partecipazione alle sessioni telematiche. | IdPartecipante (FK, PK), IdSessioneTelematica (FK, PK) |
| ADESIONE_-SESSIONEPRESENZA | Gestisce l'adesione dei partecipanti alle sessioni in presenza. | Conferma, IdSessionePresenza (FK, PK), IDPartecipante (FK, PK) |
| RICETTA | Rappresenta una ricetta culinaria utilizzata durante le sessioni. | IdRicetta (PK), Nome |
| SESSIONE_-PRESENZA_-RICETTA | Entità associativa che collega le sessioni in presenza alle ricette utilizzate. | IdRicetta (FK, PK), IdSessionePresenza (FK, PK) |
| INGREDIENTE | Rappresenta un ingrediente utilizzato nelle ricette. | IdIngrediente (PK), Nome, UnitaDiMisura |
| PREPARAZIONE-INGREDIENTE | Entità associativa che definisce gli ingredienti necessari per ogni ricetta con le relative quantità. | IdRicetta (FK, PK), IdIngrediente (FK, PK), QuantitaTotale, QuantitaUnitaria |

### 2.4.2 Dizionario delle Associazioni (Relazioni) - Parte 1

Dizionario delle Associazioni (Relazioni) - Parte 1

| Associazione | Descrizione | Classi coinvolte |
|---|---|---|
| POSSIEDE | Relazione molti-a-molti che consente ai partecipanti di possedere multiple carte di pagamento. | **PARTECIPANTE** (0..n) ↔ **CARTA** (0..n) |
| CORSO - CHEF | Relazione uno-a-molti: ogni corso è gestito da un solo chef, ma ogni chef può gestire più corsi contemporaneamente. | **CORSO** (0..n) → **CHEF** (1) |
| CORSO - TIPODICUCINA | Relazione molti-a-molti implementata tramite l'entità associativa TIPODICUCINA_CORSO per gestire corsi con multiple tipologie. | **CORSO** (0..n) ↔ **TIPODICUCINA** (0..n) |
| CORSO - SESSIONI | Relazione uno-a-molti: ogni corso può avere multiple sessioni (sia in presenza che telematiche). | **CORSO** (1) → **SESSIONE_PRESENZA** (0..n), **SESSIONE_TELEMATICA** (0..n) |
| SESSIONE_PRESENZA - PARTECIPANTI | Relazione molti-a-molti tramite ADESIONE_SESSIONEPRESENZA per gestire la partecipazione alle sessioni in presenza. | **SESSIONE_PRESENZA** (0..n) ↔ **PARTECIPANTE** (0..n) |

### 2.4.3 Dizionario delle Associazioni (Relazioni) - Parte 2

Dizionario delle Associazioni (Relazioni) - Parte 2

| Associazione | Descrizione | Classi coinvolte |
|---|---|---|
| SESSIONE_TELEMATICA - PARTECIPANTI | Relazione molti-a-molti tramite PARTECIPANTE_SESSIONETELEMATICA per gestire la partecipazione alle sessioni online. | **SESSIONE_TELEMATICA** (0..n) ↔ **PARTECIPANTE** (0..n) |
| SESSIONE_PRESENZA - RICETTE | Relazione molti-a-molti tramite SESSIONE_PRESENZA_RICETTA per associare ricette alle sessioni pratiche. | **SESSIONE_PRESENZA** (0..n) ↔ **RICETTA** (0..n) |
| RICETTA - INGREDIENTI | Relazione molti-a-molti tramite PREPARAZIONE-INGREDIENTE per definire gli ingredienti necessari per ogni ricetta. | **RICETTA** (0..n) ↔ **INGREDIENTE** (0..n) |
| RICHIESTA-PAGAMENTO | Relazione molti-a-uno che collega ogni richiesta di pagamento a un partecipante e a un corso specifico. | **PARTECIPANTE** (0..n), **CORSO** (1) → **RICHIESTAPAGAMENTO** |

### 2.4.4 Dizionario dei vincoli

**Vincoli di Dominio**

Vincoli di Dominio

| Attributo | Dominio e Vincoli |
|---|---|
| **StatoPagamento** | $\in$ {"In Attesa", "Pagato", "Fallito"}. Rappresenta lo stato attuale di una transazione di pagamento. |
| **Circuito** | $\in$ {"Visa", "Mastercard"}. Definisce il circuito della carta di pagamento utilizzata. |
| **Email** | Deve rispettare il formato (es. utente@dominio.com). Garantisce la validità dell'indirizzo email per le comunicazioni. |
| **MaxPersone** | Intero positivo $> 0$. Definisce la capacità massima di partecipanti per corso. |
| **Prezzo** | Decimale $\geq 0.00$ con precisione di 2 cifre decimali. Rappresenta il costo del corso in Euro. |
| **Durata** | Intero positivo espresso in minuti, $> 0$ e $\leq 480$ (8 ore). Definisce la durata di una sessione. |
| **Età** | Data di nascita deve corrispondere a età compresa tra 18 e 100 anni. Validata tramite trigger per garantire utenti maggiorenni. |
| **TipoCucina** | Massimo 2 tipi di cucina diversi per corso. Implementato tramite trigger per limitare la varietà culinaria per corso. |

**Vincoli Intra-relazionali**

Vincoli Intra-relazionali

| Vincolo | Descrizione e Applicazione |
|---|---|
| **Chiavi Primarie** | Ogni entità ha una chiave primaria univoca seguendo la convenzione IdNomeEntità (es. IdUtente, IdCorso). Garantisce identificazione univoca di ogni istanza. |
| **Vincoli NOT NULL** | Attributi obbligatori non possono essere nulli: tutte le chiavi primarie, attributi Nome, Email degli utenti, DataInizio/-DataFine dei corsi, Titolo delle ricette. |
| **Vincoli di Unicità** | Email in UTENTE deve essere univoca nel sistema sia per chef che partecipante e tra loro . |
| **Vincoli CHECK** | DataInizio $<$ DataFine in CORSO. Prezzo $\geq 0$ in CORSO. MaxPersone $> 0$ in CORSO. Durata $> 0$ nelle sessioni. |
| **Vincoli via Trigger** | Implementati 16 trigger specifici per vincoli complessi (dettagliati nel Cap. 4.2): unicità email Chef/Partecipante, limite 2 tipi cucina per corso, validazione date/orari sessioni, controllo pagamenti, gestione modifiche corsi non iniziati, controllo adesioni solo per iscritti. |

**Vincoli Inter-relazionali**

Vincoli Inter-relazionali

| Vincolo | Descrizione e Applicazione |
|---|---|
| **Integrità Referenziale** | Tutte le chiavi esterne devono riferirsi a valori esistenti nelle tabelle padre. Implementata tramite FOREIGN KEY con CASCADE/RESTRICT appropriati. |
| **Specializzazione UTENTE** | Ogni UTENTE deve essere esattamente uno tra CHEF o PARTECIPANTE (specializzazione totale e disgiunta). Implementata tramite vincoli CHECK o trigger. |
| **Specializzazione SESSIONE** | Ogni SESSIONE deve essere esattamente una tra SESSIONE_PRESENZA o SESSIONE_TELEMATICA (specializzazione totale e disgiunta). |
| **Vincoli di Cardinalità** | Un CORSO deve avere almeno 1 SESSIONE. Un PARTECIPANTE può avere 0..n CARTE. Una RICETTA deve avere almeno 1 INGREDIENTE (tramite PREPARAZIONEINGREDIENTE). |
| **Vincoli di Capacità** | Il numero di iscrizioni per un CORSO non può superare MaxPersone del corso stesso. Il numero di carte per PARTECIPANTE è limitato (max 5). Impedimento eliminazione corsi con iscritti paganti. Implementati tramite trigger specifici. |
| **Vincoli Temporali** | Le SESSIONI di un CORSO devono avere Data compresa tra DataInizio e DataFine del corso. DataPagamento in RICHIESTAPAGAMENTO $\leq$ DataInizio del corso associato. Controllo orario e durata sessioni. Impedimento modifiche/disiscrizioni dopo inizio corso. Validazione corrispondenza importo pagato con costo corso. |
| **Vincoli di Conflitto** | Un CHEF non può essere assegnato contemporaneamente a sessioni di presenza e telematiche nello stesso orario. Un PARTECIPANTE può aderire solo a sessioni di corsi a cui è iscritto. Eliminazione automatica ingredienti quando viene eliminata la ricetta associata. |

**Vincoli di Molteplicità (n-plu)**

Vincoli di Molteplicità

| Relazione | Cardinalità e Vincoli |
|---|---|
| **CHEF - CORSO** | (1,n) : (1,1) - Ogni chef può creare più corsi, ogni corso è creato da un unico chef. |
| **PARTECIPANTE - ADESIONE** | (1,1) : (0,n) - Ogni adesione è di un partecipante, ogni partecipante può avere più adesioni. |
| **CORSO - ADESIONE** | (1,1) : (0,n) - Ogni adesione è per un corso, ogni corso può avere più adesioni (max MaxPersone). |
| **PARTECIPANTE - CARTA** | (1,1) : (0,n) - Ogni carta appartiene a un partecipante, ogni partecipante può avere più carte (max 5). |
| **RICETTA - INGREDIENTE** | (n,n) tramite PREPARAZIONEINGREDIENTE - Una ricetta usa più ingredienti, un ingrediente può essere in più ricette. |
| **SESSIONE_PRESENZA - RICETTA** | (n,n) tramite SESSIONE_PRESENZA_RICETTA - Una sessione può usare più ricette, una ricetta può essere usata in più sessioni. |

# 3 Progettazione Logica

In questa sezione viene presentata la progettazione logica della base di dati del sistema UninaFoodLab. Lo schema logico rappresenta la traduzione del modello concettuale in un insieme di tabelle relazionali, con particolare attenzione alle chiavi primarie (pk), chiavi esterne (fk) e alle associazioni tra le entità.

## 3.1 Schema Logico

Schema Logico - Entità Principali

| Entità | Attributi |
|---|---|
| CARTA | IdCarta, Intestatario, DataScadenza, UltimeQuattroCifre, Circuito |
| PARTECIPANTE | IdPartecipante, Nome, Cognome, Email, Password, DataDiNascita, Propic |
| POSSiEDE | IdPartecipante, IdCarta |
| CHEF | IdChef, Nome, Cognome, Email, Password, DataDiNascita, Descrizione, Propic, annidiesperienza |
| CORSO | IdCorso, Nome, Descrizione, DataInizio, DataFine, FrequenzaDelleSessioni, Propic, MaxPersone, Prezzo, IdChef |
| RICHIESTAPAGAMENTO | DataRichiesta, StatoPagamento, ImportoPagato, IdCorso, IdPartecipante |
| TIPODICUCINA | Nome, IDtipocucina |
| TIPODICUCINA_CORSO | IDtipocucina, IDcorso |
| SESSIONE_PRESENZA | IdSessionePresenza, giorno, Data, Orario, Durata, citta, via, cap, Attrezzatura, Descrizione, IDcorso, IDChef |
| SESSIONE_TELEMATICA | IdSessioneTelematica, Applicazione, Codicechiamata, Data, Orario, Durata, giorno, Descrizione, IDcorso, IdChef |
| PARTECIPANTE_SESSIONETELEMATICA | IdPartecipante, IdSessioneTelematica |
| ADESIONE_SESSIONEPRESENZA | Conferma, Idsessionepresenza, IDpartecipante |
| RICETTA | IdRicetta, Nome |
| SESSIONE_PRESENZA_RICETTA | Idricetta, idsessionepresenza |
| INGREDIENTE | IdIngrediente, Nome, UnitaDiMisura |
| PREPARAZIONEINGREDIENTE | IdRicetta, IdIngrediente, QuanititaUnitaria |

**Legenda:** Sottolineato singolo = chiave primaria, Doppio sottolineato = chiave esterna.

## 3.2 Guida alla Lettura dello Schema Logico

Ogni riga rappresenta una tabella, con i relativi attributi. Le chiavi primarie sono indicate con sottolineatura singola, le chiavi esterne con doppia sottolineatura. Le associazioni tra tabelle sono esplicitate tramite le chiavi esterne, che collegano le entità tra loro.

### 3.3 Traduzione e Associazioni

Di seguito si elencano tutte le chiavi esterne e le relative associazioni per ciascuna delle 17 tabelle:

- **CARTA**: Nessuna chiave esterna.
- **PARTECIPANTE**: Nessuna chiave esterna.
- **POSSiEDE**: IdPartecipante → PARTECIPANTE(IdPartecipante), IdCarta → CARTA(IdCarta)
- **CHEF**: Nessuna chiave esterna.
- **CORSO**: IdChef → CHEF(IdChef)
- **RICHIESTAPAGAMENTO**: IdCorso → CORSO(IdCorso), IdPartecipante → PARTECIPANTE(IdPartecipante)
- **TIPODICUCINA**: Nessuna chiave esterna.
- **TIPODICUCINA_CORSO**: IDtipocucina → TIPODICUCINA(IDtipocucina), IDcorso → CORSO(IdCorso)
- **SESSIONE_PRESENZA**: IDcorso → CORSO(IdCorso), IDChef → CHEF(IdChef)
- **SESSIONE_TELEMATICA**: IDcorso → CORSO(IdCorso), IdChef → CHEF(IdChef)
- **PARTECIPANTE_SESSIONETELEMATICA**: IdPartecipante → PARTECIPANTE(IdPartecipante), IdSessioneTelematica → SESSIONE_TELEMATICA(IdSessioneTelematica)
- **ADESIONE_SESSIONEPRESENZA**: Idsessionepresenza → SESSIONE_PRESENZA(IdSessionePresenz IDpartecipante → PARTECIPANTE(IdPartecipante)
- **RICETTA**: Nessuna chiave esterna.
- **SESSIONE_PRESENZA_RICETTA**: Idricetta → RICETTA(IdRicetta), idsessionepresenza → SESSIONE_PRESENZA(IdSessionePresenza)
- **INGREDIENTE**: Nessuna chiave esterna.
- **PREPARAZIONEINGREDIENTE**: IdRicetta → RICETTA(IdRicetta), IdIngrediente → INGREDIENTE(IdIngrediente)

In questo modo, per ogni tabella è esplicitato se sono presenti chiavi esterne e, in caso affermativo, a quale tabella e attributo si collegano. Le tabelle senza chiavi esterne sono comunque collegate tramite le tabelle associative sopra elencate, garantendo l'integrità referenziale e la rappresentazione fedele delle associazioni tra le entità del dominio applicativo.

### 3.4 Visualizzazione delle Chiavi Esterne e delle Associazioni

Per rendere più intuitiva la lettura delle relazioni tra le tabelle, di seguito vengono presentate tutte le tabelle che contengono chiavi esterne, evidenziando graficamente i collegamenti e le associazioni. In questo modo si ha una visione completa delle relazioni e dei vincoli di integrità referenziale.

**POSSiEDE**

**Struttura:** (IdPartecipante, IdCarta)
**Collegamenti:**
IdPartecipante ⟶ PARTECIPANTE(IdPartecipante)
IdCarta ⟶ CARTA(IdCarta)

### CORSO

**Struttura:** (IdCorso, ..., IdChef)
**Collegamenti:**
IdChef ⟶ CHEF(IdChef)

### RICHIESTAPAGAMENTO

**Struttura:** (DataRichiesta, StatoPagamento, ImportoPagato, IdCorso, IdPartecipante)
**Collegamenti:**
IdCorso ⟶ CORSO(IdCorso)
IdPartecipante ⟶ PARTECIPANTE(IdPartecipante)

### TIPODICUCINA_CORSO

**Struttura:** (IDtipocucina, IDcorso)
**Collegamenti:**
IDtipocucina ⟶ TIPODICUCINA(IDtipocucina)
IDcorso ⟶ CORSO(IdCorso)

### SESSIONE_PRESENZA

**Struttura:** (IdSessionePresenza, ..., IDcorso, IDChef)
**Collegamenti:**
IDcorso ⟶ CORSO(IdCorso)
IDChef ⟶ CHEF(IdChef)

### SESSIONE_TELEMATICA

**Struttura:** (IdSessioneTelematica, ..., IDcorso, IdChef)
**Collegamenti:**
IDcorso ⟶ CORSO(IdCorso)
IdChef ⟶ CHEF(IdChef)

### PARTECIPANTE_SESSIONETELEMATICA

**Struttura:** (IdPartecipante, IdSessioneTelematica)
**Collegamenti:**
IdPartecipante ⟶ PARTECIPANTE(IdPartecipante)
IdSessioneTelematica ⟶ SESSIONE_TELEMATICA(IdSessioneTelematica)

### ADESIONE_SESSIONEPRESENZA

**Struttura:** (Conferma, Idsessionepresenza, IDpartecipante)
**Collegamenti:**
Idsessionepresenza ⟶ SESSIONE_PRESENZA(IdSessionePresenza)
IDpartecipante ⟶ PARTECIPANTE(IdPartecipante)

### SESSIONE_PRESENZA_RICETTA

**Struttura:** (Idricetta, idsessionepresenza)
**Collegamenti:**
Idricetta ⟶ RICETTA(IdRicetta)
idsessionepresenza ⟶ SESSIONE_PRESENZA(IdSessionePresenza)

**PREPARAZIONEINGREDIENTE**

**Struttura:** (IdRicetta, IdIngrediente, QuantitaTotale, QuanititaUnitaria)
**Collegamenti:**
IdRicetta ⟶ RICETTA(IdRicetta)
IdIngrediente ⟶ INGREDIENTE(IdIngrediente)

# 4 Definizioni SQL

## 4.1 Introduzione

In questo capitolo viene presentata l'implementazione pratica del database UninaFoodLab attraverso la definizione completa del codice SQL sviluppato in PostgreSQL. La sezione fornisce una panoramica dettagliata di tutti gli elementi che compongono il database, dalla creazione delle tabelle all'implementazione delle funzioni.

## 4.2 Definizione degli enumerati

Gli enumerati sono tipi di dato definiti dall'utente che permettono di vincolare il valore di un attributo a un insieme finito di possibilità predefinite. Nel database UninaFoodLab vengono utilizzati per rappresentare domini chiusi come circuiti di carte, stati di pagamento, frequenza delle sessioni, giorni della settimana e unità di misura degli ingredienti. Questo garantisce coerenza e integrità dei dati, semplificando la gestione delle regole applicative.

```
CREATE TYPE Circuito AS ENUM ('Visa', 'Mastercard');

CREATE TYPE StatoPagamento AS ENUM ('In attesa', 'Pagato', 'Fallito');

CREATE TYPE FDS AS ENUM (
  'Giornaliera',
  'Settimanale',
  'Bisettimanale',
  'Tre volte a settimana',
  'Mensile'
);

CREATE TYPE Giorno AS ENUM (
  'Lunedì',
  'Martedì',
  'Mercoledì',
  'Giovedì',
  'Venerdì',
  'Sabato',
  'Domenica'
);
```

```
CREATE TYPE UnitaDiMisura AS ENUM ('Grammi', 'Kilogrammi', 'Litro',
    'Centilitro');
```

---

## 4.3 Definizione delle Tabelle

### 4.3.1 Tabella Partecipante

La tabella `Partecipante` rappresenta gli utenti che si iscrivono e partecipano ai corsi di cucina. Ogni partecipante ha un identificativo univoco generato automaticamente e deve fornire informazioni personali essenziali per la registrazione.

---

```
CREATE TABLE Partecipante (
    IdPartecipante INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Nome VARCHAR(50) NOT NULL,
    Cognome VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Password VARCHAR(100) NOT NULL,
    DataDiNascita DATE NOT NULL,
    Propic TEXT
);
```

---

**Scelte progettuali:**

- **IdPartecipante**: Chiave primaria auto-incrementale utilizzando `GENERATED ALWAYS AS IDENTITY` per garantire unicità automatica
- **Email**: Constraint `UNIQUE` per evitare registrazioni duplicate
- **Password**: Campo di lunghezza fissa per supportare hash di password sicure
- **Propic**: Campo `TEXT` per supportare URL di immagini o dati base64

### 4.3.2 Tabella Carta

La tabella `Carta` memorizza le carte di pagamento associate agli utenti. Ogni carta ha un identificativo univoco, intestatario, data di scadenza, ultime quattro cifre e circuito di appartenenza.

---

```
CREATE TABLE Carta (
    IdCarta INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Intestatario VARCHAR(100) NOT NULL,
    DataScadenza DATE NOT NULL,
    UltimeQuattroCifre CHAR(4) NOT NULL,
    Circuito Circuito NOT NULL
);
```

---

**Scelte progettuali:**

- **IdCarta**: Chiave primaria auto-incrementale

- **Intestatario, DataScadenza, UltimeQuattroCifre, Circuito**: Attributi essenziali per identificare una carta
- **Circuito**: Vincolato tramite tipo enumerato per coerenza

### 4.3.3 Tabella Possiede

La tabella `Possiede` rappresenta la relazione molti-a-molti tra partecipanti e carte, indicando quali carte sono possedute da ciascun partecipante.

```
CREATE TABLE Possiede (
    IdPartecipante INT,
    IdCarta INT,
    PRIMARY KEY (IdPartecipante, IdCarta),
    FOREIGN KEY (IdPartecipante) REFERENCES Partecipante(IdPartecipante),
    FOREIGN KEY (IdCarta) REFERENCES Carta(IdCarta) on delete cascade
);
```

**Scelte progettuali:**

- **Chiave primaria composta**: (IdPartecipante, IdCarta) per garantire unicità della relazione
- **Vincoli di integrità**: Foreign key verso `Partecipante` e `Carta`
- **Cascata**: Eliminazione a cascata delle carte

### 4.3.4 Tabella Corso

La tabella `Corso` rappresenta i corsi di cucina offerti, con informazioni su nome, descrizione, periodo, frequenza, prezzo, chef responsabile e limiti di partecipazione.

```
CREATE TABLE Corso (
    IdCorso INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL,
    Descrizione VARCHAR(60) NOT NULL,
    DataInizio DATE NOT NULL,
    DataFine DATE NOT NULL,
    FrequenzaDelleSessioni FDS NOT NULL,
    Propic TEXT,
    MaxPersone INT CHECK (MaxPersone > 0),
    Prezzo DECIMAL(10, 2) CHECK (Prezzo >= 0),
    IdChef INT NOT NULL,
    FOREIGN KEY (IdChef) REFERENCES Chef(IdChef)
);
```

**Scelte progettuali:**

- **IdCorso**: Chiave primaria auto-incrementale
- **MaxPersone, Prezzo**: Vincoli di validità sui valori
- **IdChef**: Foreign key verso chef responsabile
- **FrequenzaDelleSessioni**: Vincolata tramite tipo enumerato

### 4.3.5 Tabella RichiestaPagamento

La tabella `RichiestaPagamento` registra le richieste di pagamento per i corsi, associando ogni richiesta a un partecipante e a un corso specifico, con informazioni su importo, stato e data.

```
CREATE TABLE RichiestaPagamento (
    DataRichiesta TIMESTAMP NOT NULL,
    StatoPagamento StatoPagamento NOT NULL,
    ImportoPagato DECIMAL(10, 2) CHECK (ImportoPagato >= 0),
    IdCorso INT,
    IdPartecipante INT,
    PRIMARY KEY (DataRichiesta, IdCorso, IdPartecipante),
    FOREIGN KEY (IdCorso) REFERENCES Corso(IdCorso),
    FOREIGN KEY (IdPartecipante) REFERENCES Partecipante(IdPartecipante)
);
```

**Scelte progettuali:**

- **Chiave primaria composta**: (DataRichiesta, IdCorso, IdPartecipante)
- **ImportoPagato**: Vincolo di non negatività
- **StatoPagamento**: Vincolato tramite tipo enumerato
- **Foreign key**: Collegamento a corso e partecipante

### 4.3.6 Tabella TipoDiCucina

La tabella `TipoDiCucina` elenca le tipologie di cucina disponibili, ciascuna con un identificativo univoco e un nome.

```
CREATE TABLE TipoDiCucina (
    IDTipoCucina INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Nome VARCHAR(50) NOT NULL UNIQUE
);
```

**Scelte progettuali:**

- **IDTipoCucina**: Chiave primaria auto-incrementale
- **Nome**: Unicità per evitare duplicati

### 4.3.7 Tabella TipoDiCucina_Corso

La tabella `TipoDiCucina_Corso` rappresenta l'associazione tra corsi e tipologie di cucina, permettendo di collegare più tipi di cucina a ciascun corso (fino a un massimo di due).

```
CREATE TABLE TipoDiCucina_Corso (
    IDTipoCucina INT,
    IDCorso INT,
    PRIMARY KEY (IDTipoCucina, IDCorso),
```

```
    FOREIGN KEY (IDTipoCucina) REFERENCES TipoDiCucina(IDTipoCucina),
    FOREIGN KEY (IDCorso) REFERENCES Corso(IdCorso)
);
```

---

**Scelte progettuali:**

- **Chiave primaria composta**: (IDTipoCucina, IDCorso)
- **Foreign key**: Collegamento a tipo di cucina e corso
- **Vincolo applicativo**: Massimo due tipi di cucina per corso (gestito da trigger)

### 4.3.8 Tabella Chef

La tabella `Chef` contiene le informazioni sugli chef che tengono i corsi. Oltre ai dati anagrafici, include una descrizione e gli anni di esperienza.

---

```
CREATE TABLE Chef (
    IdChef INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Nome VARCHAR(50) NOT NULL,
    Cognome VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE NOT NULL,
    Password VARCHAR(100) NOT NULL,
    DataDiNascita DATE NOT NULL,
    Descrizione VARCHAR(60),
    Propic TEXT,
    AnniDiEsperienza INT CHECK (AnniDiEsperienza >= 0)
);
```

---

**Scelte progettuali:**

- **IdChef**: Chiave primaria auto-incrementale
- **Email**: Unicità per evitare duplicati
- **AnniDiEsperienza**: Vincolo di non negatività
- **Propic**: Supporto per immagine profilo

### 4.3.9 Tabella Sessione_Presenza

La tabella `Sessione_Presenza` descrive le sessioni in presenza dei corsi, con dettagli su luogo, data, orario, durata e chef responsabile.

---

```
CREATE TABLE Sessione_Presenza (
    IdSessionePresenza INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Giorno Giorno NOT NULL,
    Data DATE NOT NULL,
    Orario TIME NOT NULL,
    Durata INTERVAL NOT NULL,
    Citta VARCHAR(50) NOT NULL,
    Via VARCHAR(100) NOT NULL,
```

```
    Cap CHAR(5) NOT NULL,
    Descrizione VARCHAR(60) NOT NULL,
    IdCorso INT,
    IdChef INT,
    FOREIGN KEY (IdCorso) REFERENCES Corso(IdCorso),
    FOREIGN KEY (IdChef) REFERENCES Chef(IdChef)
);
```

---

**Scelte progettuali:**

- **IdSessionePresenza**: Chiave primaria auto-incrementale
- **Attributi luogo**: Città, via, cap per identificare la sede
- **Foreign key**: Collegamento a corso e chef

### 4.3.10 Tabella Adesione_SessionePresenza

La tabella `Adesione_SessionePresenza` registra l'adesione dei partecipanti alle sessioni in presenza, con conferma di partecipazione.

---

```
CREATE TABLE Adesione_SessionePresenza (
    Conferma BOOLEAN,
    IdSessionePresenza INT,
    IdPartecipante INT,
    PRIMARY KEY (IdSessionePresenza, IdPartecipante),
    FOREIGN KEY (IdSessionePresenza) REFERENCES
       Sessione_Presenza(IdSessionePresenza),
    FOREIGN KEY (IdPartecipante) REFERENCES Partecipante(IdPartecipante)
);
```

---

**Scelte progettuali:**

- **Chiave primaria composta**: (IdSessionePresenza, IdPartecipante)
- **Conferma**: Indica la presenza effettiva
- **Foreign key**: Collegamento a sessione presenza e partecipante

### 4.3.11 Tabella Sessione_Telematica

La tabella `Sessione_Telematica` descrive le sessioni online dei corsi, con dettagli su applicazione, codice chiamata, data, orario, durata e chef responsabile.

---

```
CREATE TABLE Sessione_Telematica (
    IdSessioneTelematica INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Applicazione VARCHAR(100) NOT NULL,
    CodiceChiamata VARCHAR(100) NOT NULL,
    Data DATE NOT NULL,
    Orario TIME NOT NULL,
    Durata INTERVAL NOT NULL,
    Giorno Giorno NOT NULL,
```

```
    Descrizione VARCHAR(60) NOT NULL,
    IdCorso INT,
    IdChef INT,
    FOREIGN KEY (IdCorso) REFERENCES Corso(IdCorso),
    FOREIGN KEY (IdChef) REFERENCES Chef(IdChef)
);
```

**Scelte progettuali:**

- **IdSessioneTelematica**: Chiave primaria auto-incrementale
- **Applicazione, CodiceChiamata**: Dettagli per l'accesso online
- **Foreign key**: Collegamento a corso e chef

#### 4.3.12 Tabella Partecipante_SessioneTelematica

La tabella `Partecipante_SessioneTelematica` rappresenta la partecipazione dei partecipanti alle sessioni telematiche, associando ogni partecipante a una specifica sessione online.

```
CREATE TABLE Partecipante_SessioneTelematica (
    IdPartecipante INT,
    IdSessioneTelematica INT,
    PRIMARY KEY (IdPartecipante, IdSessioneTelematica),
    FOREIGN KEY (IdPartecipante) REFERENCES Partecipante(IdPartecipante),
    FOREIGN KEY (IdSessioneTelematica) REFERENCES
       Sessione_Telematica(IdSessioneTelematica)
);
```

**Scelte progettuali:**

- **Chiave primaria composta**: (IdPartecipante, IdSessioneTelematica)
- **Foreign key**: Collegamento a partecipante e sessione telematica

#### 4.3.13 Tabella Ricetta

La tabella `Ricetta` contiene le ricette che possono essere preparate durante le sessioni dei corsi.

```
CREATE TABLE Ricetta (
    IdRicetta INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL
);
```

**Scelte progettuali:**

- **IdRicetta**: Chiave primaria auto-incrementale
- **Nome**: Nome della ricetta, obbligatorio

#### 4.3.14 Tabella Sessione_Presenza_Ricetta

La tabella `Sessione_Presenza_Ricetta` associa le ricette alle sessioni in presenza, indicando quali ricette vengono preparate in ciascuna sessione.

```
CREATE TABLE Sessione_Presenza_Ricetta (
    IdRicetta INT,
    IdSessionePresenza INT,
    PRIMARY KEY (IdRicetta, IdSessionePresenza),
    FOREIGN KEY (IdRicetta) REFERENCES Ricetta(IdRicetta),
    FOREIGN KEY (IdSessionePresenza) REFERENCES
       Sessione_Presenza(IdSessionePresenza)
);
```

**Scelte progettuali:**

- **Chiave primaria composta**: (IdRicetta, IdSessionePresenza)
- **Foreign key**: Collegamento a ricetta e sessione presenza

#### 4.3.15 Tabella Ingrediente

La tabella `Ingrediente` elenca tutti gli ingredienti disponibili, con nome e unità di misura.

```
CREATE TABLE Ingrediente (
    IdIngrediente INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL,
    UnitaDiMisura UnitaDiMisura NOT NULL
);
```

**Scelte progettuali:**

- **IdIngrediente**: Chiave primaria auto-incrementale
- **UnitaDiMisura**: Vincolata tramite tipo enumerato

#### 4.3.16 Tabella PreparazioneIngrediente

La tabella `PreparazioneIngrediente` collega le ricette agli ingredienti necessari, specificando quantità totale e unitaria per ogni ingrediente in una ricetta.

```
CREATE TABLE PreparazioneIngrediente (
    IdRicetta INT,
    IdIngrediente INT,
    QuanititaUnitaria DECIMAL(10,2) NOT NULL CHECK (QuanititaUnitaria >=
       0),
    PRIMARY KEY (IdRicetta, IdIngrediente),
    FOREIGN KEY (IdRicetta) REFERENCES Ricetta(IdRicetta),
    FOREIGN KEY (IdIngrediente) REFERENCES Ingrediente(IdIngrediente)
);
```

**Scelte progettuali:**

- **Chiave primaria composta**: (IdRicetta, IdIngrediente)
- **QuantitaTotale, QuanititaUnitaria**: Vincoli di non negatività
- **Foreign key**: Collegamento a ricetta e ingrediente

## 4.4 Definizione dei Trigger

### 4.4.1 Trigger: Unicità Email tra Chef e Partecipante

Questo trigger garantisce che lo stesso indirizzo email non possa essere usato contemporaneamente da un Chef e da un Partecipante. La funzione viene richiamata sia sulle tabelle Chef che Partecipante, in inserimento e aggiornamento.

```
CREATE OR REPLACE FUNCTION verifica_unicita_email()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_TABLE_NAME = 'chef' THEN
        IF EXISTS (SELECT 1 FROM partecipante WHERE email = NEW.email)
           THEN
            RAISE EXCEPTION 'Errore: L''email "%" è già utilizzata da un
               Partecipante.', NEW.email;
        END IF;
    ELSIF TG_TABLE_NAME = 'partecipante' THEN
        IF EXISTS (SELECT 1 FROM chef WHERE email = NEW.email) THEN
            RAISE EXCEPTION 'Errore: L''email "%" è già utilizzata da uno
               Chef.', NEW.email;
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_verifica_email_partecipante
BEFORE INSERT OR UPDATE ON partecipante
FOR EACH ROW
EXECUTE FUNCTION verifica_unicita_email();

CREATE TRIGGER trg_verifica_email_chef
BEFORE INSERT OR UPDATE ON chef
FOR EACH ROW
EXECUTE FUNCTION verifica_unicita_email();
```

**Spiegazione:**

- `IF TG_TABLE_NAME = 'chef'`: Se il trigger è attivato dalla tabella Chef, controlla che l'email inserita o aggiornata non sia già presente nella tabella `partecipante` (campo `email`).
- `IF TG_TABLE_NAME = 'partecipante'`: Se il trigger è attivato dalla tabella Partecipante, controlla che l'email inserita o aggiornata non sia già presente nella tabella `chef` (campo `email`).

- `IF EXISTS (SELECT 1 ...)`: In entrambi i casi, se trova una corrispondenza, solleva un'eccezione e blocca l'inserimento/aggiornamento.
- `RAISE EXCEPTION`: Genera un errore se l'email è già in uso nell'altra tabella.
- Il trigger si attiva prima dell'inserimento o aggiornamento di un record (`BEFORE INSERT OR UPDATE`) sia su Chef che su Partecipante, garantendo l'unicità trasversale dell'email tra le due tabelle.

### 4.4.2 Trigger: Controllo Formato Email

Questo trigger verifica che l'indirizzo email inserito per Chef e Partecipante sia formalmente valido, ovvero contenga una chiocciola (@) e almeno un punto dopo la chiocciola, garantendo la correttezza sintattica degli indirizzi email memorizzati nel sistema.

```
CREATE OR REPLACE FUNCTION validate_email_full()
RETURNS TRIGGER AS $$
DECLARE
    at_pos INT;
    domain_part TEXT;
BEGIN
    at_pos := POSITION('@' IN NEW.Email);

    IF at_pos = 0 THEN
        RAISE EXCEPTION 'Email non valida: manca la chiocciola (@).';
    END IF;

    domain_part := SUBSTRING(NEW.Email FROM at_pos + 1);

    IF POSITION('.' IN domain_part) = 0 THEN
        RAISE EXCEPTION 'Email non valida: manca il punto dopo la
           chiocciola.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_validate_email_partecipante
BEFORE INSERT OR UPDATE ON PARTECIPANTE
FOR EACH ROW
EXECUTE FUNCTION validate_email_full();

CREATE TRIGGER trg_validate_email_chef
BEFORE INSERT OR UPDATE ON CHEF
FOR EACH ROW
EXECUTE FUNCTION validate_email_full();
```

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento o aggiornamento sulle tabelle `Chef` e `Partecipante`.

- La funzione controlla che l'email inserita contenga una chiocciola (@) e almeno un punto dopo la chiocciola.
- Se il formato non è valido, viene sollevata un'eccezione e l'operazione viene bloccata.
- In questo modo si garantisce che vengano accettati solo indirizzi email formalmente corretti.

### 4.4.3 Trigger: Eliminazione Ingredienti Associati a Ricetta

Quando una ricetta viene eliminata, questo trigger elimina automaticamente tutti gli ingredienti associati tramite la tabella PreparazioneIngrediente.

```
CREATE OR REPLACE FUNCTION elimina_ingredienti_della_ricetta()
RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM PREPARAZIONEINGREDIENTE
    WHERE IdRicetta = OLD.IdRicetta;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_elimina_ingredienti_associati
BEFORE DELETE ON RICETTA
FOR EACH ROW
EXECUTE FUNCTION elimina_ingredienti_della_ricetta();
```

**Spiegazione:**

- `DELETE FROM PREPARAZIONEINGREDIENTE WHERE IdRicetta = OLD.IdRicetta`: Elimina tutti gli ingredienti collegati alla ricetta che sta per essere cancellata.
- Il trigger si attiva prima della cancellazione di una ricetta (`BEFORE DELETE ON RICETTA`).
- `RETURN OLD`: Permette la cancellazione della ricetta dopo aver eliminato i riferimenti.

### 4.4.4 Trigger: Impedisci Inserimento di Carta Duplicata

Questo trigger impedisce l'inserimento o l'aggiornamento di una carta di pagamento se esiste già una carta con le stesse ultime quattro cifre, data di scadenza, circuito e intestatario, evitando duplicati nel sistema.

```
CREATE OR REPLACE FUNCTION impedisci_carta_duplicata_per_partecipante()
    RETURNS TRIGGER AS $$
    DECLARE
        existing_card_id INT;
    BEGIN
        IF TG_TABLE_NAME = 'carta' THEN
            IF EXISTS (
                SELECT 1
                FROM Carta
                WHERE UltimeQuattroCifre = NEW.UltimeQuattroCifre
                    AND DataScadenza = NEW.DataScadenza
```

```
                    AND Circuito = NEW.Circuito::Circuito
                    AND Intestatario = NEW.Intestatario
                    AND IdCarta != NEW.IdCarta
            ) THEN
                RAISE EXCEPTION 'Errore: Una carta con queste ultime 4
                    cifre, data di scadenza, circuito e intestatario
                    esiste già nel sistema.';
            END IF;
        END IF;
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_impedisci_dettagli_carta_duplicati
BEFORE INSERT OR UPDATE ON Carta
FOR EACH ROW
EXECUTE FUNCTION impedisci_carta_duplicata_per_partecipante();
```

---

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento o aggiornamento sulla tabella `Carta`.
- La funzione controlla se esiste già una carta con le stesse ultime quattro cifre, data di scadenza, circuito e intestatario, ma con un identificativo diverso.
- Se trova una corrispondenza, viene sollevata un'eccezione e l'operazione viene bloccata.
- Questo garantisce che non possano essere inserite carte duplicate nel sistema.

### 4.4.5 Trigger: Massimo 2 Tipi di Cucina per Corso

Questo trigger impedisce di associare più di due tipi di cucina a uno stesso corso, garantendo il rispetto del vincolo applicativo.

---

```
CREATE OR REPLACE FUNCTION verifica_max_tipi_cucina_per_corso()
RETURNS TRIGGER AS $$
DECLARE
    current_tipi_count INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO current_tipi_count
    FROM TipoDiCucina_Corso
    WHERE IDCorso = NEW.IDCorso;

    IF current_tipi_count >= 2 THEN
        RAISE EXCEPTION 'Errore: Il corso con ID % ha già raggiunto il
            limite massimo di 2 tipi di cucina associati.', NEW.IDCorso;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_max_tipi_cucina_corso
BEFORE INSERT ON TipoDiCucina_Corso
FOR EACH ROW
EXECUTE FUNCTION verifica_max_tipi_cucina_per_corso();
```

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento sulla tabella `TipoDiCucina_Corso`.
- La funzione conta quanti tipi di cucina sono già associati al corso specificato da `NEW.IDCorso`.
- Se il numero è già 2 o più, viene sollevata un'eccezione e l'inserimento viene bloccato.
- In questo modo si garantisce che ogni corso possa avere al massimo due tipi di cucina associati, come richiesto dal vincolo applicativo.

### 4.4.6 Trigger: Validazione Intervallo Date Corso

Questo trigger garantisce che la data di inizio di un corso non sia successiva alla data di fine e che la data di inizio non sia antecedente alla data corrente, assicurando la coerenza temporale dei corsi inseriti o aggiornati.

```
CREATE OR REPLACE FUNCTION verifica_intervallo_date_corso()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.DataInizio > NEW.DataFine THEN
        RAISE EXCEPTION 'Errore: La DataInizio del corso (%) non può
           essere successiva alla DataFine (%).',
            NEW.DataInizio, NEW.DataFine;
    END IF;

    IF NEW.DataInizio < CURRENT_DATE THEN
        RAISE EXCEPTION 'Errore: La DataInizio del corso (%) non può
           essere antecedente alla data corrente (%).',
            NEW.DataInizio, CURRENT_DATE;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

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

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento o aggiornamento sulla tabella `Corso`.
- La funzione controlla che la data di inizio non sia successiva a quella di fine e che non sia antecedente alla data corrente.

- Se una delle due condizioni non è rispettata, viene sollevata un'eccezione e l'operazione viene bloccata.
- In questo modo si garantisce la coerenza temporale dei dati relativi ai corsi.

### 4.4.7 Trigger: Importo Pagato Deve Corrispondere al Costo del Corso

Questo trigger garantisce che l'importo pagato per una richiesta di pagamento corrisponda esattamente al prezzo del corso associato, evitando discrepanze tra quanto dovuto e quanto effettivamente pagato.

```
CREATE OR REPLACE FUNCTION
   verifica_importo_pagamento_corrisponde_prezzo_corso()
RETURNS TRIGGER AS $$
DECLARE
    course_price DECIMAL(10,2);
BEGIN
    SELECT Prezzo INTO course_price
    FROM Corso
    WHERE IdCorso = NEW.IdCorso;

    IF NEW.ImportoPagato != course_price THEN
        RAISE EXCEPTION 'Errore: L''ImportoPagato (%.2f) deve
           corrispondere esattamente al Prezzo del corso (%.2f).',
           NEW.ImportoPagato, course_price;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_valida_importo_pagamento
BEFORE INSERT OR UPDATE ON RichiestaPagamento
FOR EACH ROW
EXECUTE FUNCTION verifica_importo_pagamento_corrisponde_prezzo_corso();
```

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento o aggiornamento sulla tabella `RichiestaPagamento`.
- La funzione recupera il prezzo del corso associato alla richiesta di pagamento tramite l'`IdCorso`.
- Se l'`ImportoPagato` non corrisponde esattamente al prezzo del corso, viene sollevata un'eccezione e l'operazione viene bloccata.
- In questo modo si garantisce che ogni pagamento sia coerente con il costo effettivo del corso.

### 4.4.8 Trigger: Età Compresa tra 18 e 100 Anni

Questo trigger garantisce che la data di nascita inserita per Chef e Partecipante produca un'età compresa tra 18 e 100 anni, assicurando che solo utenti con età valida possano essere registrati nel sistema.

```
CREATE OR REPLACE FUNCTION verifica_intervallo_eta()
RETURNS TRIGGER AS $$
DECLARE
    person_age_years INTEGER;
    min_age CONSTANT INTEGER := 18;
    max_age CONSTANT INTEGER := 100;
BEGIN
    IF NEW.DataDiNascita IS NULL THEN
        RAISE EXCEPTION 'Errore: La DataDiNascita non può essere NULL.';
    END IF;

    person_age_years := EXTRACT(YEAR FROM AGE(CURRENT_DATE,
       NEW.DataDiNascita));

    IF person_age_years < min_age THEN
        RAISE EXCEPTION 'Errore: L''età della persona (%, calcolata dalla
           DataDiNascita %) è inferiore all''età minima consentita (%).',
           person_age_years, NEW.DataDiNascita, min_age;
    END IF;

    IF person_age_years > max_age THEN
        RAISE EXCEPTION 'Errore: L''età della persona (%, calcolata dalla
           DataDiNascita %) è superiore all''età massima consentita
           (%).', person_age_years, NEW.DataDiNascita, max_age;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_valida_eta_chef
BEFORE INSERT OR UPDATE ON Chef
FOR EACH ROW
EXECUTE FUNCTION verifica_intervallo_eta();

CREATE TRIGGER trg_valida_eta_partecipante
BEFORE INSERT OR UPDATE ON Partecipante
FOR EACH ROW
EXECUTE FUNCTION verifica_intervallo_eta();
```

---

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento o aggiornamento sulle tabelle `Chef` e `Partecipante`.
- La funzione calcola l'età a partire dalla data di nascita fornita.
- Se l'età è inferiore a 18 anni o superiore a 100 anni, viene sollevata un'eccezione e l'operazione viene bloccata.
- In questo modo si garantisce che solo utenti con età valida possano essere registrati nel sistema.

### 4.4.9 Trigger: Controllo Superamento Numero Massimo di Partecipanti per Corso

Questo trigger impedisce che il numero di partecipanti paganti a un corso superi il limite massimo definito dal campo `MaxPersone` della tabella `Corso`. In questo modo si garantisce che non vengano accettate richieste di pagamento che eccedono la capienza prevista per ciascun corso.

---

```
CREATE OR REPLACE FUNCTION verifica_superamento_max_partecipanti_corso()
RETURNS TRIGGER AS $$
DECLARE
    course_max_people INTEGER;
    current_paid_count INTEGER;
    potential_paid_count INTEGER;
    course_name VARCHAR(255);
BEGIN
    SELECT MaxPersone, Nome INTO course_max_people, course_name
    FROM Corso
    WHERE IdCorso = NEW.IdCorso;

    IF course_max_people <= 0 OR course_max_people IS NULL THEN
        RETURN NEW;
    END IF;

    SELECT COUNT(*)
    INTO current_paid_count
    FROM RichiestaPagamento
    WHERE IdCorso = NEW.IdCorso
        AND StatoPagamento = 'Pagato';

    potential_paid_count := current_paid_count;

    IF TG_OP = 'INSERT' THEN
        IF NEW.StatoPagamento = 'Pagato' THEN
            potential_paid_count := potential_paid_count + 1;
        END IF;
    ELSIF TG_OP = 'UPDATE' THEN
        IF OLD.StatoPagamento != 'Pagato' AND NEW.StatoPagamento =
           'Pagato' THEN
potential_paid_count := potential_paid_count + 1;
        ELSIF OLD.StatoPagamento = 'Pagato' AND NEW.StatoPagamento !=
           'Pagato' THEN
            potential_paid_count := potential_paid_count - 1;
        END IF;
    END IF;

    IF potential_paid_count > course_max_people THEN
        RAISE EXCEPTION 'Errore: Il corso "%" (ID %) ha raggiunto il
           limite massimo di % partecipanti paganti. Impossibile
           accettare questa richiesta di pagamento (attuali: %s, limite:
           %s).',
                        course_name, NEW.IdCorso, course_max_people,
                           current_paid_count, course_max_people;
    END IF;

    RETURN NEW;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_controllo_max_partecipanti_corso
BEFORE INSERT OR UPDATE ON RichiestaPagamento
FOR EACH ROW
EXECUTE FUNCTION verifica_superamento_max_partecipanti_corso();
```

---

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento o aggiornamento sulla tabella `RichiestaPagamento`.
- La funzione recupera il numero massimo di partecipanti paganti previsto per il corso e il numero attuale di pagamenti confermati.
- In caso di inserimento o aggiornamento che porterebbe il totale dei paganti oltre il limite, viene sollevata un'eccezione e l'operazione viene bloccata.
- In questo modo si garantisce il rispetto della capienza massima prevista per ciascun corso.

### 4.4.10 Trigger: Controllo Orario e Durata Sessione Presenza

Questo trigger garantisce che l'orario di inizio e la durata di una sessione in presenza siano coerenti e rispettino i vincoli applicativi: la durata deve essere compresa tra 1 e 8 ore, l'orario deve essere valido (ore ¡ 23, minuti ¡ 60) e la fine della lezione non puo' superare le 23.

---

```
CREATE OR REPLACE FUNCTION check_orario_e_durata()
RETURNS TRIGGER AS $$
DECLARE
    orario_ora INTEGER;
    orario_minuti INTEGER;
    durata_ore NUMERIC;
    fine_lezione TIME;
BEGIN
    IF (EXTRACT(EPOCH FROM NEW.Durata)/3600) < 1 OR (EXTRACT(EPOCH FROM
       NEW.Durata)/3600) > 8 THEN
        RAISE EXCEPTION 'Durata deve essere maggiore di 1 e minore di 8';
    END IF;

    orario_ora := EXTRACT(HOUR FROM NEW.Orario);
    orario_minuti := EXTRACT(MINUTE FROM NEW.Orario);

    IF orario_ora >= 23 THEN
        RAISE EXCEPTION 'La parte intera dell''orario deve essere minore
           di 23';
    ELSIF orario_minuti >= 60 THEN
        RAISE EXCEPTION 'La parte decimale dell''orario deve essere
           minore di 60';
    END IF;

    fine_lezione := NEW.Orario + NEW.Durata;

    IF fine_lezione > TIME '23:59:59' THEN
```

```
        RAISE EXCEPTION 'L''orario di fine lezione non può superare le
            23:59';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_sessione_presenza
BEFORE INSERT OR UPDATE ON SESSIONE_PRESENZA
FOR EACH ROW EXECUTE FUNCTION check_orario_e_durata();
```

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento o aggiornamento sulla tabella `SESSIONE_PRESENZA`.
- Controlla che la durata sia maggiore di 1 e minore di 8 ore.
- Verifica che la parte intera dell'orario (ore) sia minore di 23 e la parte decimale (minuti) sia minore di 60.
- Calcola l'orario di fine lezione e verifica che non superi le 23.
- Se uno di questi vincoli non è rispettato, viene sollevata un'eccezione e l'operazione viene bloccata.
- In questo modo si garantisce la correttezza e la coerenza degli orari delle sessioni in presenza.
- **EXTRACT**: è una funzione SQL che estrae una parte specifica (ad esempio HOUR o MINUTE) da un valore di tipo data o orario. Nel trigger viene usata per ottenere le ore e i minuti dall'orario di inizio.
- **TIME**: è un tipo di dato SQL che rappresenta un orario (senza data). Nel trigger viene usato per confrontare l'orario di fine lezione con il limite massimo consentito (`TIME '23:59:59'`).
- L'operatore `+` tra un valore di tipo `TIME` e uno di tipo intervallo (`Durata`) permette di calcolare l'orario di fine lezione sommando la durata all'orario di inizio.

### 4.4.11 Trigger: Validazione Range Data Sessione rispetto al Corso

Questo trigger impedisce che la data di una sessione (sia in presenza che telematica) venga aggiornata a una data non compresa nell'intervallo di validità del corso associato, o a una data nel passato. Garantisce che tutte le sessioni si svolgano entro le date di inizio e fine del corso e che non vengano spostate retroattivamente.

```
CREATE OR REPLACE FUNCTION impedisci_aggiornamento_sessione_non_valida()
RETURNS TRIGGER AS $$
DECLARE
    data_inizio_corso DATE;
    data_fine_corso DATE;
    nuova_data DATE;
BEGIN
    SELECT DataInizio, DataFine INTO data_inizio_corso, data_fine_corso
```

```
    FROM CORSO
    WHERE IdCorso = NEW.IdCorso;

    nuova_data := NEW.Data;

    IF nuova_data < CURRENT_DATE THEN
        RAISE EXCEPTION 'Non è possibile modificare una sessione con una
           data nel passato (%).', nuova_data;
    END IF;

    IF nuova_data < data_inizio_corso OR nuova_data > data_fine_corso THEN
        RAISE EXCEPTION 'La data della sessione (%) deve essere compresa
           tra l''inizio e la fine del corso (% - %).',
                    nuova_data, data_inizio_corso, data_fine_corso;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_impedisci_aggiornamento_sessione_presenza_non_valida
BEFORE UPDATE ON SESSIONE_PRESENZA
FOR EACH ROW
EXECUTE FUNCTION impedisci_aggiornamento_sessione_non_valida();

CREATE TRIGGER trg_impedisci_aggiornamento_sessione_telematica_non_valida
BEFORE UPDATE ON SESSIONE_TELEMATICA
FOR EACH ROW
EXECUTE FUNCTION impedisci_aggiornamento_sessione_non_valida();
```

---

**Spiegazione:**

- Il trigger si attiva prima di ogni aggiornamento sulle tabelle `SESSIONE_PRESENZA` e `SESSIONE_TELEMATICA`.
- Recupera le date di inizio e fine del corso associato alla sessione.
- Impedisce di aggiornare la data della sessione a una data nel passato.
- Impedisce di impostare una data di sessione al di fuori dell'intervallo di validità del corso.
- Se una delle condizioni non è rispettata, viene sollevata un'eccezione e l'operazione viene bloccata.
- In questo modo si garantisce che tutte le sessioni si svolgano in un periodo valido e non retroattivo rispetto al corso.

### 4.4.12 Trigger: Impedire Eliminazione di Corsi con Iscritti Paganti

Questo trigger impedisce l'eliminazione di un corso se sono ancora presenti partecipanti che hanno già effettuato il pagamento, garantendo l'integrità delle iscrizioni e la correttezza dei dati gestiti dal sistema.

---

```
CREATE OR REPLACE FUNCTION impedisci_eliminazione_corso_se_iscritto()
RETURNS TRIGGER AS $$
DECLARE
    enrolled_count INTEGER;
    course_name VARCHAR(255);
BEGIN
    SELECT Nome INTO course_name
    FROM Corso
    WHERE IdCorso = OLD.IdCorso;

    SELECT COUNT(*)
    INTO enrolled_count
    FROM RichiestaPagamento
    WHERE IdCorso = OLD.IdCorso
        AND StatoPagamento = 'Pagato';

    IF enrolled_count > 0 THEN
        RAISE EXCEPTION 'Errore: Impossibile eliminare il corso "%" (ID
           %). Ci sono ancora % partecipanti paganti iscritti.',
                        course_name, OLD.IdCorso, enrolled_count;
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_impedisci_eliminazione_corso
BEFORE DELETE ON Corso
FOR EACH ROW
EXECUTE FUNCTION impedisci_eliminazione_corso_se_iscritto();
```

---

**Spiegazione:**

- Il trigger si attiva prima della cancellazione di un corso dalla tabella `Corso`.
- La funzione verifica se esistono partecipanti paganti ancora iscritti al corso.
- Se il numero di iscritti paganti è maggiore di zero, viene sollevata un'eccezione e l'eliminazione viene bloccata.
- In questo modo si garantisce che non vengano eliminati corsi con iscritti attivi, preservando la coerenza dei dati.

### 4.4.13 Trigger: Disiscrizione da un Corso Solo se non Iniziato

Questo trigger impedisce la disiscrizione (cancellazione della richiesta di pagamento) da un corso che è già iniziato.

---

```
CREATE OR REPLACE FUNCTION impedisci_disiscrizione_se_corso_iniziato()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
```

```
        FROM CORSO
        WHERE IdCorso = OLD.IdCorso
        AND DataInizio <= CURRENT_DATE
    ) THEN
        RAISE EXCEPTION 'Non è possibile disiscriversi da un corso già
            iniziato.';
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_impedisci_disiscrizione_se_corso_iniziato
BEFORE DELETE ON RICHIESTAPAGAMENTO
FOR EACH ROW
EXECUTE FUNCTION impedisci_disiscrizione_se_corso_iniziato();
```

**Spiegazione:**

- Il trigger si attiva prima della cancellazione di una richiesta di pagamento (`RICHIESTAPAGAMENTO`).
- Se la data di inizio del corso è già passata o è oggi, la disiscrizione viene bloccata.
- In questo modo si impedisce la disiscrizione da corsi già iniziati.

#### 4.4.14 Trigger: Modifica Corso Solo se non Iniziato

Questo trigger impedisce la modifica dei dati di un corso che è già iniziato.

```
CREATE OR REPLACE FUNCTION impedisci_modifica_se_corso_iniziato()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.DataInizio <= CURRENT_DATE THEN
        RAISE EXCEPTION 'Il corso è già iniziato e non può essere
            modificato.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_impedisci_modifica_se_corso_iniziato
BEFORE UPDATE ON CORSO
FOR EACH ROW
EXECUTE FUNCTION impedisci_modifica_se_corso_iniziato();
```

**Spiegazione:**

- Il trigger si attiva prima di ogni aggiornamento sulla tabella `CORSO`.
- Se la data di inizio del corso è già passata o è oggi, la modifica viene bloccata.
- In questo modo si impedisce la modifica di corsi già iniziati.

#### 4.4.15 Trigger: Chef Non Può Essere Assegnato Contemporaneamente a Sessione Presenza e Telematica

Questo trigger impedisce che uno stesso chef sia assegnato contemporaneamente, nello stesso giorno, sia a una sessione in presenza che a una sessione telematica, garantendo la coerenza degli orari.

---

```
CREATE OR REPLACE FUNCTION verifica_sessioni_contemporanee_chef()
RETURNS TRIGGER AS $$
DECLARE
    conflicting_session_count INTEGER;
    chef_name VARCHAR(255);
    session_type_being_inserted TEXT;
    other_session_type TEXT;
BEGIN
    SELECT Nome || ' ' || Cognome INTO chef_name
    FROM Chef
    WHERE IdChef = NEW.IDChef;

    IF TG_TABLE_NAME = 'sessione_presenza' THEN
        session_type_being_inserted := 'presenza';
        other_session_type := 'telematica';
    ELSIF TG_TABLE_NAME = 'sessione_telematica' THEN
        session_type_being_inserted := 'telematica';
        other_session_type := 'presenza';
    ELSE
        RAISE EXCEPTION 'Errore interno del trigger: tabella sconosciuta
           (%).', TG_TABLE_NAME;
    END IF;

    IF session_type_being_inserted = 'presenza' THEN
        SELECT COUNT(*)
        INTO conflicting_session_count
        FROM Sessione_Telematica
        WHERE IDChef = NEW.IDChef
            AND Data = NEW.Data;
    ELSIF session_type_being_inserted = 'telematica' THEN
        SELECT COUNT(*)
        INTO conflicting_session_count
        FROM Sessione_Presenza
        WHERE IDChef = NEW.IDChef
            AND Data = NEW.Data;
    END IF;

    IF conflicting_session_count > 0 THEN
        RAISE EXCEPTION 'Errore: Lo Chef "%" (ID %) è già assegnato a una
           sessione di tipo "%" in data %. Impossibile assegnarlo anche a
           una sessione di tipo "%" nello stesso giorno.',
                        chef_name, NEW.IDChef, other_session_type,
                           NEW.Data, session_type_being_inserted;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_verifica_sessione_presenza_chef
BEFORE INSERT OR UPDATE ON Sessione_Presenza
FOR EACH ROW
EXECUTE FUNCTION verifica_sessioni_contemporanee_chef();

CREATE TRIGGER trg_verifica_sessione_telematica_chef
BEFORE INSERT OR UPDATE ON Sessione_Telematica
FOR EACH ROW
EXECUTE FUNCTION verifica_sessioni_contemporanee_chef();
```

**Spiegazione:**

- La funzione `check_partecipazione_corso` viene utilizzata come trigger per la tabella `ADESIONE_SESSIONEPRE`
- Si attiva prima di ogni inserimento su questa tabella.
- La funzione recupera l'ID del corso associato alla sessione di presenza (`SESSIONE_PRESENZA`) a cui il partecipante si sta iscrivendo.
- Verifica poi se il partecipante (`IDpartecipante`) ha almeno una richiesta di pagamento (`RICHIESTAPAGAMENTO`) associata a quel corso.
- Se non esiste alcuna richiesta di pagamento per quel partecipante e corso, viene sollevata un'eccezione e l'inserimento viene bloccato.
- In questo modo si garantisce che solo i partecipanti che hanno effettuato il pagamento (o almeno la richiesta di pagamento) possano iscriversi a una sessione di presenza di un corso.

### 4.4.16 Trigger: Partecipante può aderire solo se iscritto al corso

Questo trigger impedisce che un partecipante possa aderire a una sessione in presenza se non risulta iscritto (cioè non ha una richiesta di pagamento associata al corso della sessione).

```
CREATE OR REPLACE FUNCTION check_partecipazione_corso()
RETURNS TRIGGER AS $$
DECLARE
    corso_id INT;
    partecipante_id INT;
    pagamento_count INT;
BEGIN
    SELECT IDcorso INTO corso_id
    FROM SESSIONE_PRESENZA
    WHERE IdSessionePresenza = NEW.IdSessionePresenza;

    partecipante_id := NEW.IDpartecipante;

    SELECT COUNT(*) INTO pagamento_count
    FROM RICHIESTAPAGAMENTO
    WHERE IdCorso = corso_id
    AND IdPartecipante = partecipante_id;

    IF pagamento_count = 0 THEN
```

```
        RAISE EXCEPTION 'Il partecipante % non è iscritto al corso %.',
            partecipante_id, corso_id;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_partecipazione_corso
BEFORE INSERT ON ADESIONE_SESSIONEPRESENZA
FOR EACH ROW
EXECUTE FUNCTION check_partecipazione_corso();
```

**Spiegazione:**

- Il trigger si attiva prima di ogni inserimento o aggiornamento sulle tabelle `Sessione_Presenza` e `Sessione_Telematica`.
- La funzione controlla che lo stesso chef non sia già assegnato a una sessione di tipo diverso nello stesso giorno.
- Se viene rilevata una sovrapposizione, viene sollevata un'eccezione e l'operazione viene bloccata.
- In questo modo si garantisce che uno chef non possa essere assegnato contemporaneamente a sessioni di tipo diverso nello stesso giorno.

## 4.5 Definizione delle View

### 4.5.1 View: Statistiche Mensili Chef

Questa view fornisce una panoramica mensile delle attività di ciascun chef, utile per la generazione del grafico del report mensile. Riassume il numero di ricette preparate, i valori massimo, minimo e medio di ricette per sessione, il numero di corsi e sessioni tenuti, e il guadagno totale del mese corrente.

```
CREATE OR REPLACE VIEW vista_statistiche_mensili_chef AS
    SELECT
        ch.IdChef,
        ch.Nome,
        ch.Cognome,
        COUNT(spr.IdRicetta) AS totale_ricette_mese,
        COALESCE(MAX(spr_count.ricette_per_sessione), 0) AS
            max_ricette_in_sessione,
        COALESCE(MIN(spr_count.ricette_per_sessione), 0) AS
            min_ricette_in_sessione,
        COALESCE(AVG(spr_count.ricette_per_sessione), 0) AS
            media_ricette_in_sessione,
        COUNT(DISTINCT c.IdCorso) AS numero_corsi,
        COUNT(DISTINCT sp.IdSessionePresenza) AS numero_sessioni_presenza,
        COUNT(DISTINCT st.IdSessioneTelematica) AS
            numero_sessioni_telematiche,
```

```
    COALESCE(SUM(rp.ImportoPagato), 0) AS guadagno_totale,
    EXTRACT(MONTH FROM COALESCE(
        sp.Data,
        st.Data,
        rp.DataRichiesta,
        c.DataInizio,
        c.DataFine
    )) AS mese,
    EXTRACT(YEAR FROM COALESCE(
        sp.Data,
        st.Data,
        rp.DataRichiesta,
        c.DataInizio,
        c.DataFine
    )) AS anno
FROM CHEF ch
LEFT JOIN CORSO c ON ch.IdChef = c.IdChef
LEFT JOIN SESSIONE_PRESENZA sp ON c.IdCorso = sp.IDcorso AND
   sp.IDChef = ch.IdChef
LEFT JOIN SESSIONE_PRESENZA_RICETTA spr ON spr.IdSessionePresenza =
   sp.IdSessionePresenza
LEFT JOIN (
    SELECT spr2.IdSessionePresenza, COUNT(*) AS ricette_per_sessione
    FROM SESSIONE_PRESENZA_RICETTA spr2
    JOIN SESSIONE_PRESENZA sp2 ON spr2.IdSessionePresenza =
       sp2.IdSessionePresenza
    GROUP BY spr2.IdSessionePresenza
) spr_count ON spr_count.IdSessionePresenza = sp.IdSessionePresenza
LEFT JOIN SESSIONE_TELEMATICA st ON c.IdCorso = st.IDcorso AND
   st.IDChef = ch.IdChef
LEFT JOIN RICHIESTAPAGAMENTO rp ON rp.IdCorso = c.IdCorso
GROUP BY ch.IdChef, ch.Nome, ch.Cognome,
    EXTRACT(MONTH FROM COALESCE(sp.Data, st.Data, rp.DataRichiesta,
       c.DataInizio, c.DataFine)),
    EXTRACT(YEAR FROM COALESCE(sp.Data, st.Data, rp.DataRichiesta,
       c.DataInizio, c.DataFine));
```

**Spiegazione:**

- La view aggrega i dati per ogni chef, suddividendo i risultati per mese e anno tramite le funzioni `EXTRACT(MONTH...)` e `EXTRACT(YEAR...)`.
- Per ogni chef vengono calcolati:
  - Il totale delle ricette preparate nel mese (`totale_ricette_mese`).
  - Il massimo, minimo e media di ricette preparate per sessione (`max_ricette_in_sessione`, `min_ricette_in_sessione`, `media_ricette_in_sessione`).
  - Il numero di corsi, sessioni in presenza e sessioni telematiche tenute dallo chef.
  - Il guadagno totale del mese, calcolato come somma degli importi pagati per i corsi dello chef.
- Le `LEFT JOIN` collegano chef, corsi, sessioni, ricette e pagamenti per ottenere una panoramica completa delle attività mensili di ciascun chef.

- La view utilizza `COALESCE` per gestire i casi in cui non ci siano dati, restituendo zero dove necessario.
- Questa view e' pensata per essere utilizzata come base dati per la generazione di report e grafici mensili sulle performance degli chef.

### 4.5.2 View: Quantità Ingredienti per Sessione

Questa view calcola dinamicamente la quantità totale necessaria di ciascun ingrediente per ogni sessione in presenza, in base al numero di partecipanti che hanno confermato la presenza. È utile per aggiornare automaticamente la quantità totale da preparare ogni volta che viene registrata una nuova adesione confermata.

```
CREATE OR REPLACE VIEW QuantitaPerSessione AS
    SELECT
        pi.IdRicetta,
        pi.IdIngrediente,
        pi.QuanititaUnitaria,
        COUNT(asp.IDpartecipante) AS NumeroPartecipanti,
        pi.QuanititaUnitaria * COUNT(asp.IDpartecipante) AS
           QuantitaTotale,
        spr.Idsessionepresenza
    FROM PREPARAZIONEINGREDIENTE pi
    JOIN SESSIONE_PRESENZA_RICETTA spr ON pi.IdRicetta = spr.Idricetta
    JOIN ADESIONE_SESSIONEPRESENZA asp ON asp.Idsessionepresenza =
       spr.Idsessionepresenza
    WHERE asp.Conferma = true
    GROUP BY pi.IdRicetta, pi.IdIngrediente, pi.QuanititaUnitaria,
       spr.Idsessionepresenza;
```

**Spiegazione:**

- La view collega le tabelle delle preparazioni ingredienti, delle sessioni in presenza e delle adesioni confermate.
- Per ogni ingrediente di ogni ricetta associata a una sessione, calcola il numero di partecipanti confermati e la quantità totale necessaria (quantità unitaria moltiplicata per il numero di partecipanti).
- Ogni volta che viene registrata una nuova adesione confermata, la view restituisce automaticamente il valore aggiornato della quantità totale da preparare.
- Utile per la gestione logistica e l'approvvigionamento degli ingredienti in base alle presenze effettive.