UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

CORSO DI LAUREA IN INFORMATICA

INSEGNAMENTO DI BASI DI DATI 1

A.A. 2024/2025

Progettazione e Sviluppo di una Base Di Dati per la Gestione di Corsi di Cucina Telematici

UninaFoodLab

A cura di

Antonio Esposito

Matricola N86005375

antonio.esposito167@studenti.unina.it

Docenti

Prof.ssa Mara Sangiovanni

INDICE

1. Introduzione 4

1.1 Descrizione del dominio 4

1.2 Struttura dell'elaborato 5

2. Progettazione concettuale 7

2.1 Introduzione ai diagrammi ER e UML 7

2.2 Diagramma delle classi 7

2.3 Lettura del diagramma delle classi 8

2.4 Diagramma Entità-Relazione 9

2.5 Differenze tra i due diagrammi 11

2.6 Ristrutturazione del diagramma delle classi 11

2.7 Diagramma delle classi ristrutturato 12

2.8 Criteri di ristrutturazione 13

3. Progettazione Logica 14

3.1 Dizionario delle classi 14

3.2 Dizionario delle associazioni 17

3.3 Dizionario dei vincoli 18

3.4 Schema logico relazionale 21

4. Progettazione fisica e realizzazione 22

4.1 Specifiche dell'ambiente PostgreSQL 22

4.2 Definizioni delle tabelle 24

4.2.1 Tabella Categoria 24

4.2.2 Tabella Frequenza_sessione 24

4.2.2 Tabella Chef 25

4.2.2 Tabella Utente 25

4.2.2 Tabella Corso 26

4.2.2 Tabella Sessione 26

4.2.2 Tabella Ricetta 27

4.2.2 Tabella Ingrediente 27

4.2.2 Tabella Iscrizione 28

4.2.2 Tabella Ingredienti_utilizzati 28

4.2.2 Tabella Realizzazione_ricetta 29

4.2.2 Tabella Partecipazione 29

4.3 Trigger Functions 30

4.3.1 check_data_partecipazione() 30

4.3.2 check_limite_sessioni() 30

4.3.3 check_sessione_in_presenza() 31

4.3.4 check_ricetta_sessione_in_presenza 31

4.4 Script di Popolamento 32

Capitolo 1

Introduzione

L'elaborato si propone di illustrare le scelte effettuate nella progettazione e realizzazione della base di dati a supporto dell'applicativo *UninaFoodLab*, un sistema dedicato alla creazione e gestione di corsi di cucina telematici.

Per lo sviluppo del progetto è stato utilizzato **PostgreSQL**, uno dei database relazionali più diffusi in ambito accademico grazie alla sua quasi completa aderenza allo standard SQL, affiancato da **pgAdmin 4**, strumento di amministrazione e gestione del database.

1.1 Descrizione del dominio

La traccia fornita dal committente è la seguente:

"UninaFoodLab è un sistema per la gestione di corsi di cucina tematici. Gli chef possono registrare corsi su specifici argomenti (es. cucina asiatica, pasticceria, panificazione), specificando una data di inizio e una frequenza delle sessioni (es. settimanale, ogni due giorni). Ogni corso è articolato in più sessioni, che possono essere di due tipi: online, oppure in presenza, in cui gli utenti svolgono attività pratiche.

Gli utenti possono iscriversi a più corsi e, nel caso delle sessioni pratiche, devono fornire una adesione esplicita per confermare la loro partecipazione. Ogni sessione pratica prevede la preparazione di una o più ricette, ciascuna delle quali richiede una specifica lista di ingredienti. Le adesioni vengono utilizzate per pianificare correttamente la quantità di ingredienti necessari, evitando così sprechi alimentari.

Si utilizzino le proprie conoscenze del dominio per definire eventuali dettagli non specificati nella traccia."

1.2 Struttura dell'Elaborato

Per illustrare il percorso di progettazione e sviluppo, il presente documento è articolato nei seguenti capitoli:

- **Capitolo 2 - Progettazione Concettuale:** Verranno presentati i modelli ER e UML che descrivono le entità del dominio, i loro attributi e le associazioni tra di esse.
- **Capitolo 3 - Progettazione Logica:** Si descriverà la traduzione dello schema concettuale in un modello logico relazionale, definendo le tabelle e i vincoli di integrità, e verranno presentati i dizionari delle classi, delle associazioni e dei vincoli.
- **Capitolo 4 - Progettazione Fisica e Implementazione:** Saranno illustrate le scelte implementative in ambiente PostgreSQL, con la presentazione del codice utilizzato per creare la struttura del database, inclusi creazione delle tabelle, vincoli e trigger.

Capitolo 2

Progettazione Concettuale

2.1 Introduzione ai diagrammi ER e UML

In questa fase della progettazione ci si mantiene volutamente distanti dal database e da scelte di tipo tecnologico. I modelli concettuali hanno l'obiettivo di fornire una rappresentazione astratta, logica e sistematica del problema, costituendo la base per le successive fasi di progettazione logica e fisica.

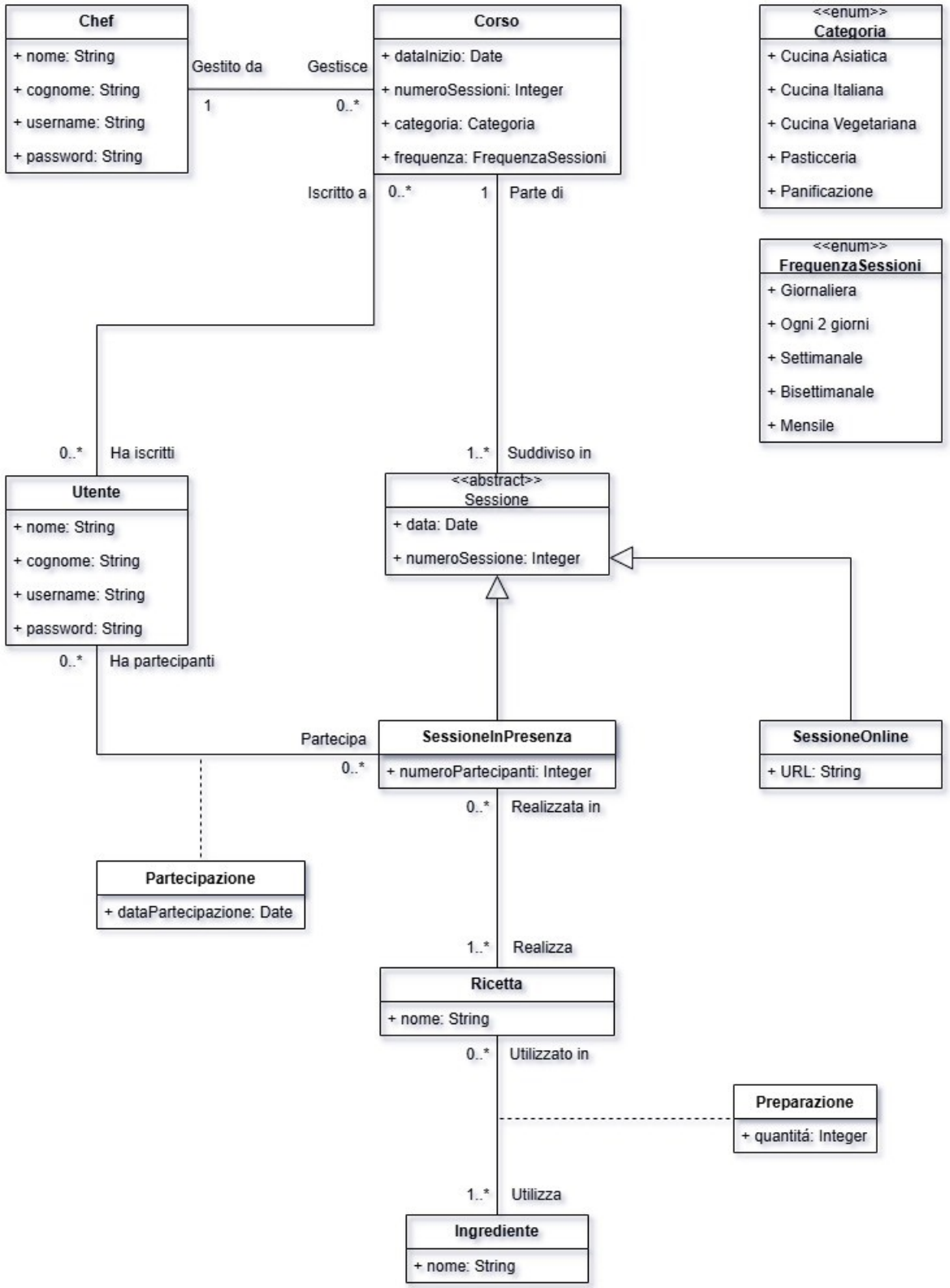
In questo capitolo verranno presentati due strumenti di modellazione:

- **Diagrammi ER (Entità–Relazione)**: nati specificamente per la progettazione di basi di dati relazionali, consentono di rappresentare in maniera immediata le entità rilevanti del dominio, i loro attributi e le relazioni che le collegano. Sono particolarmente adatti a tradurre i requisiti in uno schema concettuale pronto per la progettazione logica. In particolare, verrà adottata la versione **EER (Enhanced ER)**, che introduce notazioni aggiuntive, come l'ereditarietà tra classi aumentando il potere espressivo del modello.

- **Diagrammi UML (Unified Modeling Language)**: pur non essendo nati con un orientamento ai database, ma come linguaggio di modellazione software più generale, risultano molto utili per descrivere la struttura del sistema e le relazioni tra le entità che lo compongono. In questo contesto verranno utilizzati in particolare i **diagrammi delle classi**, in modo da offrire una prospettiva complementare a quella fornita dal modello EER.

L'impiego congiunto dei due approcci consente di ottenere una rappresentazione ricca e completa: da un lato centrata sulla struttura dei dati, dall'altro sulla modellazione delle relazioni e delle dinamiche del sistema.

2.2 Diagramma delle classi



2.3 Lettura del diagramma delle classi

Come già anticipato, a questo livello di astrazione non ci preoccupiamo di come andremo a rappresentare i dati nel nostro database, ma solo di definire a grandi linee le entità di cui ci occupiamo e il modo col quale dovranno essere collegate tra loro. Alcune brevi linee guida per una lettura corretta del diagramma:

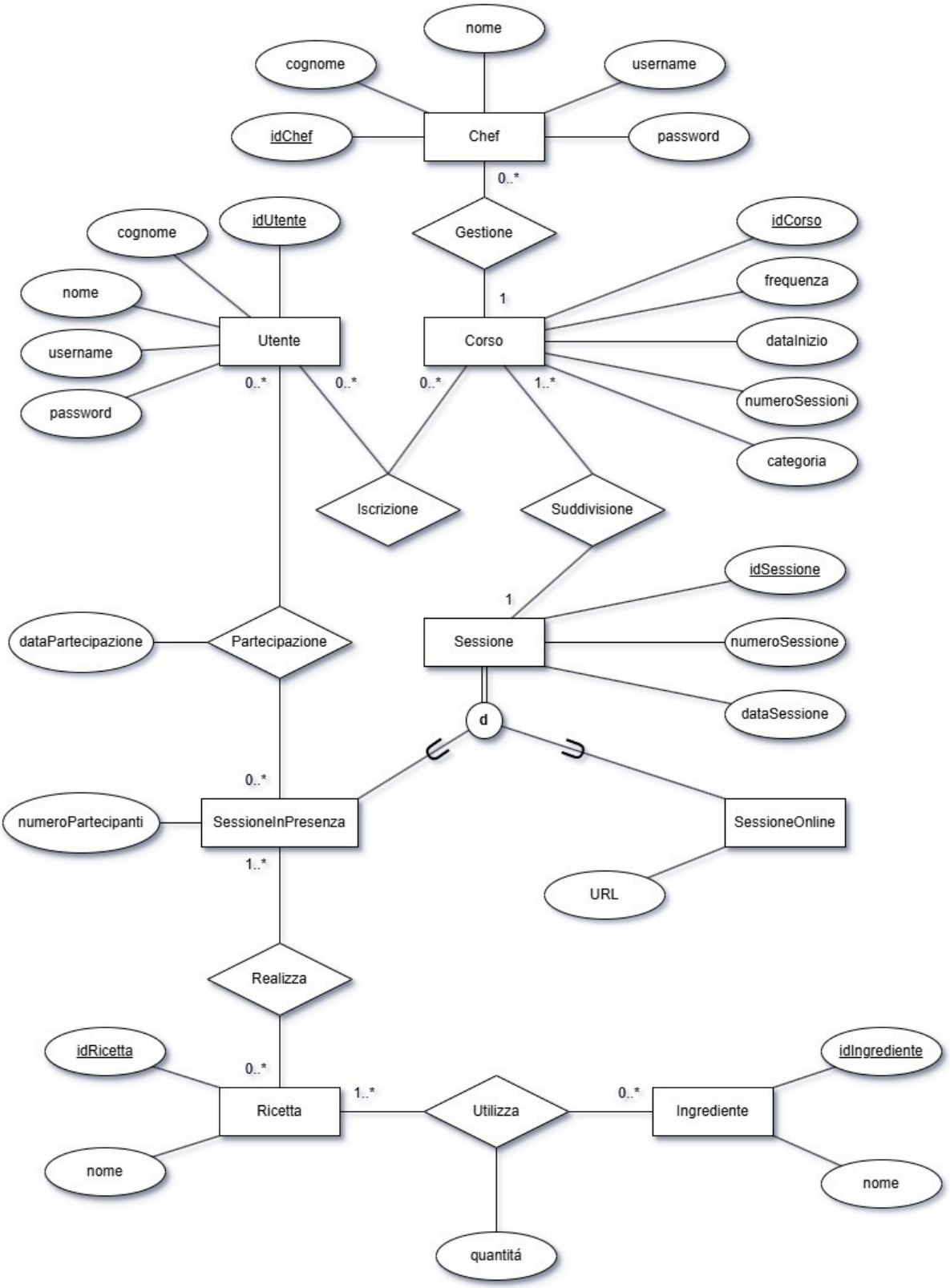
- **Classi e attributi**: ogni classe rappresenta un insieme di oggetti con caratteristiche comuni; gli attributi descrivono le proprietà principali di ciascuna classe. Ogni attributo ha un tipo, ossia un insieme finito o infinito di valori che può assumere.

- **Associazioni**: le linee tra le classi indicano come gli oggetti sono collegati tra loro. Le molteplicità specificano quante istanze di una classe possono essere collegate a quante istanze dell'altra.

- **Ereditarietà e generalizzazione**: le frecce con un triangolo vuoto indicano che una classe eredita proprietà e comportamenti da un'altra, permettendo di riutilizzare concetti comuni e definire strutture gerarchiche.

- **Elementi particolari**: Per ora la frequenza con la quale si svolgono le sessioni di un corso e la categoria sono rappresentate da enumerazioni, mentre la classe sessione è astratta (ossia non può esistere una sessione che non sia in presenza oppure a distanza). Da notare inoltre la presenza di alcune classi legate alle associazioni coi relativi attributi.

2.4 Diagramma Entità-Relazione



2.5 Differenze tra i due diagrammi

La logica della lettura del diagramma segue a grandi linee la stessa di quella del diagramma delle classi e verrà quindi volutamente omessa una spiegazione a riguardo, focalizzandoci invece sulle differenze tra i diagrammi.

Innanzitutto, essendo il diagramma ER per natura molto più vicino al modello fisico del database relazionale, nella sua stesura è stato fatto un passo in avanti verso quella direzione introducendo le **chiavi primarie**, ossia attributi che possono essere usati per identificare univocamente un'istanza di una classe. Nel diagramma ER queste chiavi vengono rappresentate con una sottolineatura, nel nostro caso sono gli ID e prendono il nome di **chiavi surrogate,** questo perché non esistono altri attributi che potrebbero altrimenti fare da chiave.

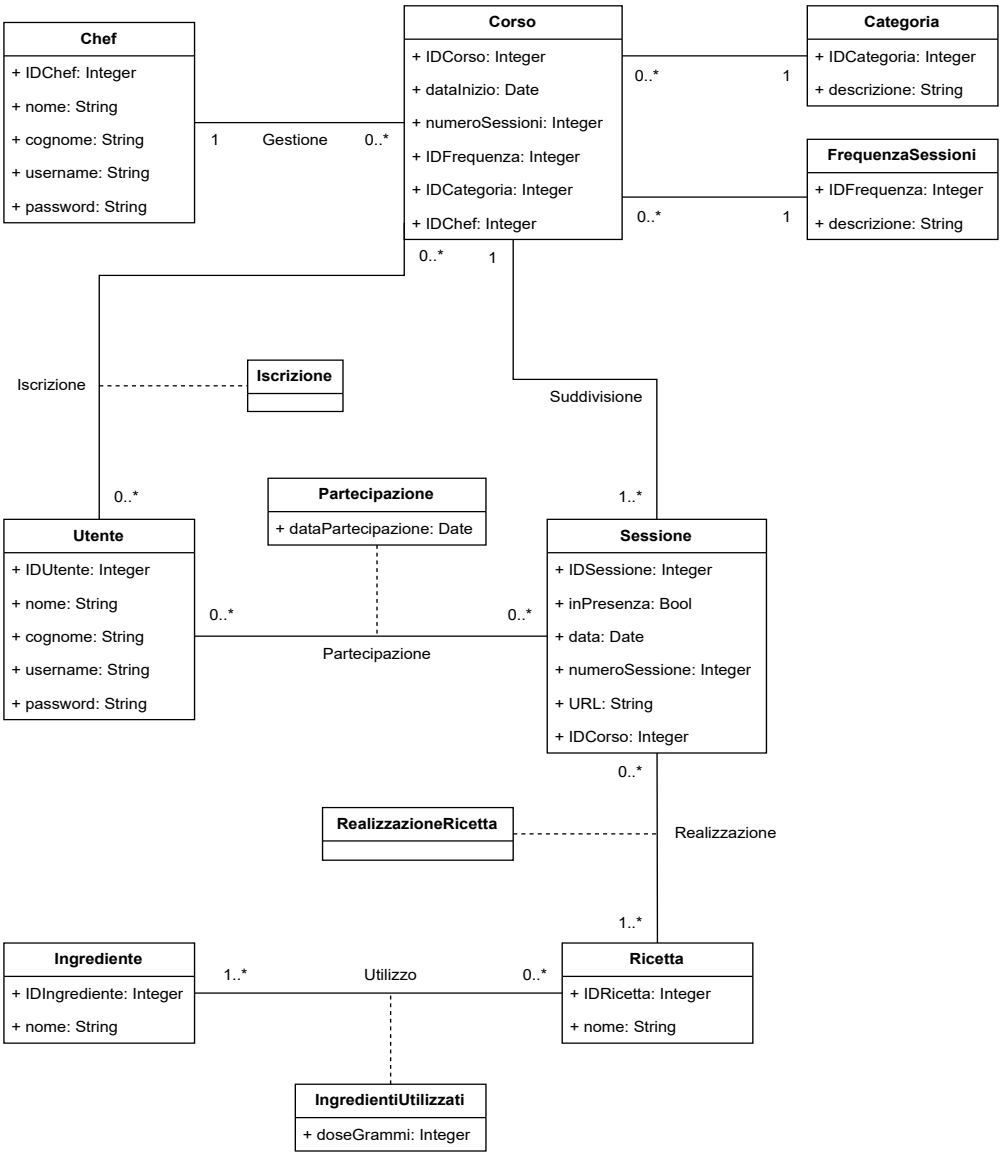
Inoltre, il diagramma ER si concentra maggiormente sulle relazioni, che vengono rappresentate esplicitamente come oggetti, mentre nel class diagram viene data più importanza alla classe e alla sua composizione e l'associazione ha il solo scopo di "navigabilità tra le classi", ossia di mostrare il modo in cui le classi sono collegate e interagiscono fra loro.

2.6 Ristrutturazione del diagramma delle classi

Per ristrutturazione si intende l'eliminazione e/o la modifica di alcune caratteristiche dei diagrammi concettuali al fine di rendere più immediata la traduzione tra il modello concettuale ed il modello logico, che verrà poi a sua volta tradotto nel database.

Nei prossimi paragrafi verrà quindi presentato il diagramma delle classi ristrutturato e saranno spiegate le scelte progettuali dietro alla ristrutturazione dello stesso.

2.7 Diagramma delle classi ristrutturato



2.8 Criteri di ristrutturazione

Di seguito le scelte prese durante la ristrutturazione del diagramma:

- **Classi Categoria e FrequenzaSessioni:** inizialmente concepite come enumerazioni, sono alla fine state "promosse" a classi autonome in relazione con la classe **Corso.** L'alternativa sarebbe stata inserire all'interno di quest'ultimo un attributo "frequenza" di tipo stringa, ma sarebbe stato troppo "debole" e facilmente suddetto ad errori. Ad esempio, trovare tutti i corsi con frequenza settimanale avrebbe potuto non dare il risultato sperato nel caso in cui ci fossero stati problemi di tipo lessicali (attributo scritto come settimanale, SETTIMANALE, Settimanale ecc.). Questa soluzione garantisce una maggiore consistenza semantica e semplifica le operazioni di interrogazione.

- **Classe Sessione:** le sottoclassi della suddetta sono state inglobate nella classe madre, questo perché l'ereditarietà era di tipo totale e disgiunta e perché le classi figlie avevano relativamente pochi attributi. Questa soluzione consente di mantenere una rappresentazione più compatta e flessibile, pur richiedendo l'introduzione di **vincoli** specifici per garantire la coerenza dei dati, che verranno esplicitati nel capitolo successivo nel **dizionario dei vincoli**.

- **Esplicitazione delle PK, FK e classi di associazione**: le chiavi primarie ed esterne sono state indicate esplicitamente come attributi, con l'obiettivo di facilitare la traduzione verso il modello logico. Nelle relazioni 1:N, la classe dipendente eredita come attributo la PK della classe forte, a evidenziare la propria dipendenza da essa. Per le relazioni M:N si è invece ricorso a classi di associazione, cioè classi dotate come PK della coppia di chiavi primarie delle entità collegate e, se necessario, di attributi propri dell'associazione (es. la classe **IngredientiUtilizzati**).

Capitolo 3

Progettazione Logica

3.1 Dizionario delle classi

Classe	Descrizione	Attributi
Chef	Rappresenta l'entità chef che crea e gestisce i corsi.	**IDChef (Integer)**: Chiave surrogata che identifica univocamente uno chef. **nome (String)**: Nome dello chef. **cognome (String)**: Cognome dello chef. **username (String)**: Nome utente per l'accesso al sistema. **password (String)**: Password per l'accesso al sistema.
Utente	Rappresenta l'entità utente che si iscrive e partecipa ai corsi.	**IDUtente (Integer)**: Chiave surrogata che identifica univocamente un utente. **nome (String)**: Nome dell'utente. **cognome (String)**: Cognome dell'utente. **username (String)**: Nome utente per l'accesso al sistema. **password (String)**: Password per l'accesso al sistema.
Corso	Rappresenta un corso di cucina con le sue caratteristiche principali.	**IDCorso (Integer)**: Chiave surrogata che identifica univocamente un corso. **dataInizio (Date)**: Data di inizio del corso. **numeroSessioni (Integer)**: Numero totale di sessioni previste per il corso.

		IDFrequenza (Integer): Chiave esterna che collega alla frequenza delle sessioni. **IDCategoria (Integer)**: Chiave esterna che collega alla categoria del corso. **IDChef (Integer)**: Chiave esterna che collega allo chef che tiene il corso.
Sessione	Rappresenta una singola sessione di un corso.	**IDSessione (Integer)**: Chiave surrogata che identifica univocamente una sessione. **inPresenza (Bool)**: Valore booleano che indica se la sessione è in presenza (true) o online (false). **data (Date)**: Data in cui si svolge la sessione. **numeroSessione (Integer)**: Numero progressivo della sessione all'interno del corso (es. Lezione 1, 2, 3...) **URL (String)**: Link per accedere alla sessione, se online. **IDCorso (Integer)**: Chiave esterna che collega la sessione al corso di appartenenza.
Categoria	Rappresenta la categoria di un corso.	**IDCategoria (Integer)**: Chiave surrogata che identifica univocamente una categoria. **descrizione (String)**: Nome/descrizione della categoria.

FrequenzaSessioni	Definisce la cadenza con cui si tengono le sessioni di un corso.	**IDFrequenza (Integer)**: Chiave surrogata **descrizione (String)**: Nome/Descrizione della frequenza.
Ricetta	Rappresenta una ricetta che può essere realizzata durante una sessione in presenza.	**IDRicetta (Integer)**: Chiave surrogata che identifica univocamente una ricetta. **nome (String)**: Nome della ricetta.
Ingrediente	Rappresenta un singolo ingrediente utilizzato nelle ricette.	**IDIngrediente (Integer)**: Chiave surrogata che identifica univocamente un ingrediente. **nome (String)**: Nome dell'ingrediente (es. "Farina 00").

3.2 Dizionario delle associazioni

Associazione	Entità	Cardinalità	Descrizione
Gestione	Chef (Gestore) → Corso (Gestito)	1 a 0...*	Uno Chef gestisce 0 o più corsi, un Corso è gestito da un unico Chef.
Appartenenza a Categoria	Categoria (Classificatore) → Corso (Classificato)	1 a 0...*	Una Categoria può "raggruppare" zero o più Corsi. Ogni Corso appartiene a una sola Categoria.
Definizione Frequenza	FrequenzaSessioni (Definitore) → Corso (Definito)	1 a 0...*	Rappresenta la cadenza con la quale si tengono le sessioni di un Corso. Le sessioni di un corso seguono una sola Frequenza, più corsi possono avere la stessa Frequenza.
Suddivisione	Corso (Insieme) → Sessione (Elemento)	1 a 1...*	Un Corso è suddiviso in una o più Sessioni. Ogni Sessione appartiene a un unico Corso.
Iscrizione	Utente (Iscritto) ↔ Chef (Titolare)	0...* a 0...*	Un Utente può iscriversi a più corsi e un corso può avere più utenti iscritti.
Partecipazione	Utente (Partecipante) ↔ Sessione (Evento)	0...* a 0...*	Traccia la partecipazione di un Utente a una Sessione in presenza.
Realizzazione	Sessione (Contesto) ↔ Ricetta (Oggetto)	0...* a 1...*	Indica quali Ricette vengono preparate in una Sessione.
Utilizzo	Ricetta (Composto) ↔ Ingrediente (Componente)	0...* a 1...*	Specifica quali Ingredienti ed in quali dosi sono necessari per realizzare una Ricetta.

3.3 Dizionario dei vincoli

Nome Vincolo	Tipo	Descrizione
categoria.descrizioneNotNullable	DOMINIO	La descrizione di una categoria non può essere nulla.
chef.nomeNotNullable	DOMINIO	Il nome dello chef non può essere nullo.
chef.cognomeNotNullable	DOMINIO	Il cognome dello chef non può essere nullo.
chef.usernameNotNullable	DOMINIO	Lo username dello chef non può essere nullo.
chef.passwordNotNullable	DOMINIO	La password dello chef non può essere nulla.
frequenza_sessioni.descrizioneNotNullable	DOMINIO	La descrizione di una frequenza non può essere nulla.
ingrediente.nomeNotNullable	DOMINIO	Il nome di un ingrediente non può essere nullo.
ingredienti_utilizzati.doseGrammiNotNullable	DOMINIO	La dose di un ingrediente in una ricetta non può essere nulla.
partecipazione.dataPartecipazioneNotNullable	DOMINIO	La data di partecipazione non può essere nulla.
ricetta.nomeNotNullable	DOMINIO	Il nome di una ricetta non può essere nullo.

sessione.dataNotNullable	DOMINIO	La data di una sessione non può essere nulla.
sessione.numero_sessione NotNullable	DOMINIO	Il numero di una sessione non può essere nullo.
utente.nomeNotNullable	DOMINIO	Il nome dell'utente non può essere nullo.
utente.cognomeNot Nullable	DOMINIO	Il cognome dell'utente non può essere nullo.
utente.usernameNot Nullable	DOMINIO	Lo username dell'utente non può essere nullo.
utente.passwordNot Nullable	DOMINIO	La password dell'utente non può essere nulla.
corso.numeroSessioniNotL essThanZero	DOMINIO	Il numero_sessioni di un corso deve essere maggiore di 0 (o nullo).
ingredientiUtilizzati.dose GrammiGreaterThanZero	DOMINIO	La dose_grammi di un ingrediente deve essere maggiore di 0.
sessione.numeroSess-ioneGreaterThanZero	DOMINIO	Il numero_sessione di una sessione deve essere maggiore di 0.
sessione.checkUrl	ENNUPLA	Una sessione deve essere in presenza o, in alternativa, avere un url_meeting.
categoria.descrizioneKey	INTRARELAZIONALE	La descrizione di ogni categoria deve essere unica.
chef.usernameKey	INTRARELAZIONALE	Lo username di ogni chef deve essere unico.
frequenza.descrizioneKey	INTRARELAZIONALE	La descrizione di ogni frequenza_sessioni deve essere unica.

Ingrediente.nomeKey	INTRARELAZIONALE	Il nome di ogni ingrediente deve essere unico.
ricetta.nomeKey	INTRARELAZIONALE	Il nome di ogni ricetta deve essere unico.
sessione.idCorsoNumero-SessioneKey	INTRARELAZIONALE	La coppia (id_corso, numero_sessione) deve essere unica.
utente.usernameKey	INTRARELAZIONALE	Lo username di ogni utente deve essere unico.
dataPartecipazioneIsValid	INTERRELAZIONALE	La data nella quale un utente conferma la partecipazione ad una sessione non può essere maggiore della data della sessione.
partecipazioneSessione-Presenza	INTERRELAZIONALE	Non è possibile partecipare a una sessione che non è in presenza.
ricettaSessionePresenza	INTERRELAZIONALE	Una ricetta può essere associata solo ad una sessione in presenza.
checkLimiteSessioni	INTERRELAZIONALE	Il numero di una sessione non può essere maggiore del numero totale di sessioni di un corso.

3.3 Schema logico relazionale

Chef (idChef, nome, cognome, username, password)
Utente (idUtente, nome, cognome, username, password)
Categoria (idCategoria, descrizione)
FrequenzaSessioni (idFrequenza, descrizione)
Ingrediente (idIngrediente, nome)
Ricetta (idRicetta, nome)
Corso (idCorso, dataInizio, numeroSessioni, idFrequenza, idCategoria, idChef)
Sessione (idSessione, inPresenza, data, numeroSessione, URL, IDCorso)
Iscrizione (idUtente, idCorso)
Partecipazione (idUtente, idSessione, dataPartecipazione)
RealizzazioneRicetta (idSessione, idRicetta)
IngredientiUtilizzati (idRicetta, idIngrediente, doseGrammi)

NOTA: le PK sono rappresentate da una sottolineatura, le FK da una doppia sottolineatura.

Nelle tabelle di associazione la PK è data dall'insieme delle FK.

Capitolo 4

Progettazione Fisica e Realizzazione

4.1 Specifiche dell'ambiente PostgreSQL

In questo paragrafo verranno spiegate alcune delle caratteristiche specifiche del DBMS PostgreSQL che verranno usate nel codice SQL per creare il database.

- **Tipo SERIAL:** questo tipo non fa parte dello standard SQL, ma è esclusivo di PostgreSQL (altri DBMS hanno il loro equivalente). Il suo scopo è quello di semplificare la gestione di chiavi primarie di tipo numerico intero, e si presta perfettamente a chiavi surrogate ID, come nel nostro caso. Dichiarando un attributo di tipo SERIAL il DBMS crea automaticamente:

  1. Una colonna di tipo INTEGER.
  2. Una sequenza collegata alla colonna.
  3. Un default che usa nextval() sulla sequenza.

  Nel concreto, questo significa che viene creata automaticamente una colonna che si auto-incrementa ogni volta che viene aggiunto un nuovo record alla tabella, garantendo ID unici e semplificando la gestione delle chiavi primarie. È comunque possibile assegnare manualmente degli ID a dei record nel caso fosse necessario.

- **TRIGGER:** In postgreSQL i trigger seguono una logica divisa in 2 parti: una procedura e il trigger che la invoca. La *Trigger function* è una particolare funzione che non riceve niente in input e restituisce un valore di tipo TRIGGER, mentre il trigger è semplicemente un meccanismo che lega la funzione ad una tabella e definisce come e quando eseguirla.

Sintassi tipica di una trigger function:

```
CREATE OR REPLACE FUNCTION function()
RETURNS TRIGGER AS $$
BEGIN
    -- fai qualcosa
    RETURN NEW; --operazione riuscita/approvata, questa sarà
    la  nuova riga effettivamente inserita nel database
END;
$$ LANGUAGE plpgsql; --linguaggio default di PostgreSQL
```

Sintassi tipica di un trigger:

```
CREATE TRIGGER trigger
BEFORE INSERT OR UPDATE ON TABELLA
FOR EACH ROW
EXECUTE FUNCTION function();
```

La logica modulare dei trigger permette di separare la responsabilità di gestione della funzione del trigger dalla logica di quando/come usare la funzione, inoltre è molto comodo quando più trigger potrebbero aver bisogno di usare la stessa funzione.

Nei prossimi paragrafi verrà esposto tutto il codice sql usato per creare e popolare il database.

4.2 Definizioni delle tabelle

4.2.1 Tabella Categoria

Query | Query History

```sql
CREATE TABLE IF NOT EXISTS public.categoria
(
    id_categoria SERIAL,
    descrizione VARCHAR(50) NOT NULL,
    CONSTRAINT categoria_pkey PRIMARY KEY (id_categoria),
    CONSTRAINT categoria_descrizione_key UNIQUE (descrizione)
)
```

4.2.2 Tabella frequenza_sessioni

Query | Query History

```sql
CREATE TABLE IF NOT EXISTS public.frequenza_sessioni
(
    id_frequenza SERIAL,
    descrizione VARCHAR(50) NOT NULL,
    CONSTRAINT frequenza_pkey PRIMARY KEY (id_frequenza),
    CONSTRAINT frequenza_descrizione_key UNIQUE (descrizione)
)
```

4.2.3 Tabella Chef

Query Query History

```
1
2  CREATE TABLE IF NOT EXISTS public.chef
3  (
4      id_chef SERIAL,
5      nome VARCHAR(50) NOT NULL,
6      cognome VARCHAR(50) NOT NULL,
7      username VARCHAR(50) NOT NULL,
8      password VARCHAR(50) NOT NULL,
9      CONSTRAINT chef_pkey PRIMARY KEY (id_chef),
10     CONSTRAINT chef_username_key UNIQUE (username)
11 )
12
```

4.2.4 Tabella Utente

Query Query History

```
1
2  CREATE TABLE IF NOT EXISTS public.utente
3  (
4      id_utente SERIAL,
5      nome VARCHAR(50) NOT NULL,
6      cognome VARCHAR(50) NOT NULL,
7      username VARCHAR(50) NOT NULL,
8      password VARCHAR(50) NOT NULL,
9      CONSTRAINT utente_pkey PRIMARY KEY (id_utente),
10     CONSTRAINT utente_username_key UNIQUE (username)
11 )
12
```

4.2.5 Tabella Corso

Query | Query History

```
CREATE TABLE IF NOT EXISTS public.corso
(
    id_corso SERIAL,
    data_inizio DATE,
    numero_sessioni INTEGER,
    id_frequenza INTEGER,
    id_categoria INTEGER,
    id_chef INTEGER,
    CONSTRAINT corso_pkey PRIMARY KEY (id_corso),
    CONSTRAINT corso_id_categoria_fkey FOREIGN KEY (id_categoria)
        REFERENCES public.categoria (id_categoria)
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT corso_id_chef_fkey FOREIGN KEY (id_chef)
        REFERENCES public.chef (id_chef)
        ON UPDATE NO ACTION
        ON DELETE CASCADE, --se viene eliminato lo chef gestore viene eliminato anche il corso
    CONSTRAINT corso_id_frequenza_fkey FOREIGN KEY (id_frequenza)
        REFERENCES public.frequenza_sessioni (id_frequenza)
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT corso_numero_sessioni_check CHECK (numero_sessioni IS NULL OR numero_sessioni > 0)
)

```

4.2.6 Tabella Sessione

Query | Query History

```
CREATE TABLE IF NOT EXISTS public.sessione
(
    id_sessione SERIAL,
    in_presenza BOOLEAN,
    data DATE NOT NULL,
    numero_sessione INTEGER NOT NULL,
    url_meeting VARCHAR(200),
    id_corso INTEGER NOT NULL,
    CONSTRAINT sessione_pkey PRIMARY KEY (id_sessione),
    CONSTRAINT sessione_id_corso_numero_sessione_key UNIQUE (id_corso, numero_sessione),
    CONSTRAINT sessione_id_corso_fkey FOREIGN KEY (id_corso)
        REFERENCES public.corso (id_corso)
        ON UPDATE NO ACTION
        ON DELETE NO ACTION, --se il corso viene cancellato la sessione potrebbe comunque restare valida
    CONSTRAINT check_url_presenza CHECK (in_presenza = true OR url_meeting IS NOT NULL),
    CONSTRAINT sessione_numero_sessione_check CHECK (numero_sessione > 0)
)

CREATE OR REPLACE TRIGGER trigger_check_limite_sessioni
    BEFORE INSERT OR UPDATE
    ON public.sessione
    FOR EACH ROW
    EXECUTE FUNCTION public.check_limite_sessioni();
```

4.2.7 Tabella Ricetta

Query Query History

```
CREATE TABLE IF NOT EXISTS public.ricetta
(
    id_ricetta SERIAL,
    nome VARCHAR(50) NOT NULL,
    CONSTRAINT ricetta_pkey PRIMARY KEY (id_ricetta),
    CONSTRAINT ricetta_nome_key UNIQUE (nome)
)
```

4.2.8 Tabella Ingrediente

Query Query History

```
CREATE TABLE IF NOT EXISTS public.ingrediente
(
    id_ingrediente SERIAL),
    nome VARCHAR(50) NOT NULL,
    CONSTRAINT ingrediente_pkey PRIMARY KEY (id_ingrediente),
    CONSTRAINT ingrediente_nome_key UNIQUE (nome)
)
```

4.2.9 Tabella Iscrizione

```
Query  Query History
1  CREATE TABLE IF NOT EXISTS public.iscrizione
2  (
3      id_corso INTEGER NOT NULL,
4      id_utente INTEGER NOT NULL,
5      CONSTRAINT iscrizione_pkey PRIMARY KEY (id_corso, id_utente),
6      CONSTRAINT iscrizione_id_corso_fkey FOREIGN KEY (id_corso)
7          REFERENCES public.corso (id_corso)
8          ON UPDATE NO ACTION
9          ON DELETE CASCADE, --se il corso viene cancellato l'iscrizione non ha senso
10     CONSTRAINT iscrizione_id_utente_fkey FOREIGN KEY (id_utente)
11         REFERENCES public.utente (id_utente)
12         ON UPDATE NO ACTION
13         ON DELETE CASCADE --stesso discorso se viene cancellato l'utente
14 )
```

4.2.10 Tabella Ingredienti_utilizzati

```
Query  Query History
1  CREATE TABLE IF NOT EXISTS public.ingredienti_utilizzati
2  (
3      dose_grammi INTEGER NOT NULL,
4      id_ricetta INTEGER NOT NULL,
5      id_ingrediente INTEGER NOT NULL,
6      CONSTRAINT ingredienti_utilizzati_pkey PRIMARY KEY (id_ricetta, id_ingrediente),
7      CONSTRAINT ingredienti_utilizzati_id_ingrediente_fkey FOREIGN KEY (id_ingrediente)
8          REFERENCES public.ingrediente (id_ingrediente)
9          ON UPDATE NO ACTION
10         ON DELETE CASCADE,
11     CONSTRAINT ingredienti_utilizzati_id_ricetta_fkey FOREIGN KEY (id_ricetta)
12         REFERENCES public.ricetta (id_ricetta)
13         ON UPDATE NO ACTION
14         ON DELETE CASCADE,
15     CONSTRAINT ingredienti_utilizzati_dose_grammi_check CHECK (dose_grammi > 0)
16 )
```

4.2.11 Tabella Realizzazione_ricetta

Query | Query History

```
CREATE TABLE IF NOT EXISTS public.realizzazione_ricetta
(
    id_ricetta INTEGER NOT NULL,
    id_sessione INTEGER NOT NULL,
    CONSTRAINT realizzazione_ricetta_pkey PRIMARY KEY (id_ricetta, id_sessione),
    CONSTRAINT realizzazione_ricetta_id_ricetta_fkey FOREIGN KEY (id_ricetta)
        REFERENCES public.ricetta (id_ricetta)
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT realizzazione_ricetta_id_sessione_fkey FOREIGN KEY (id_sessione)
        REFERENCES public.sessione (id_sessione)
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

CREATE OR REPLACE TRIGGER ensure_ricetta_sessione_presenza
    BEFORE INSERT
    ON public.realizzazione_ricetta
    FOR EACH ROW
    EXECUTE FUNCTION public.check_ricetta_sessione_presenza();
```

4.2.12 Tabella Partecipazione

Query | Query History

```
CREATE TABLE IF NOT EXISTS public.partecipazione
(
    data_partecipazione DATE NOT NULL,
    id_utente INTEGER NOT NULL,
    id_sessione INTEGER NOT NULL,
    CONSTRAINT partecipazione_pkey PRIMARY KEY (id_utente, id_sessione),
    CONSTRAINT partecipazione_id_sessione_fkey FOREIGN KEY (id_sessione)
        REFERENCES public.sessione (id_sessione)
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT partecipazione_id_utente_fkey FOREIGN KEY (id_utente)
        REFERENCES public.utente (id_utente)
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

CREATE OR REPLACE TRIGGER ensure_data_partecipazione
    BEFORE INSERT OR UPDATE
    ON public.partecipazione
    FOR EACH ROW
    EXECUTE FUNCTION public.check_data_partecipazione();

CREATE OR REPLACE TRIGGER ensure_partecipazione_sessione_presenza
    BEFORE INSERT
    ON public.partecipazione
    FOR EACH ROW
    EXECUTE FUNCTION public.check_partecipazione_sessione_presenza();
```

4.3 Funzioni trigger

4.3.1 check_data_partecipazione()

Query | Query History

```
CREATE OR REPLACE FUNCTION public.check_data_partecipazione()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.data_partecipazione >
     (SELECT data FROM sessione WHERE id_sessione = NEW.id_sessione) THEN
        RAISE EXCEPTION 'errore: la data partecipazione è successiva alla data di svolgimento della sessione';
  END IF;
  RETURN NEW;
END;
$$ language plpgsql;
```

4.3.2 check_limite_sessioni()

Query | Query History

```
-- funzione usata dal trigger che verifica se il numero di una sessione é valido
CREATE OR REPLACE FUNCTION check_limite_sessioni()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.numero_sessione > (SELECT numero_sessioni FROM corso WHERE id_corso = NEW.id_corso) THEN
        RAISE EXCEPTION 'Il numero sessione supera il numero di sessioni previste dal corso';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

4.3.2 check_sessione_in_presenza()

Query Query History

```sql
-- funzione usata dal trigger che verifica se la sessione a cui un utente partecipa é in presenza
CREATE OR REPLACE FUNCTION check_sessione_in_presenza()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT (SELECT in_presenza FROM sessione WHERE id_sessione = NEW.id_sessione) THEN
        RAISE EXCEPTION 'Impossibile registrare la partecipazione: la sessione non è una sessione in presenza.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

4.3.3 check_ricetta_sessione_in_presenza()

Query Query History

```sql
-- Funzione per il trigger che verifica che una sessione sia in presenza
-- prima di associarla a una ricetta.
CREATE OR REPLACE FUNCTION check_ricetta_sessione_in_presenza()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT (SELECT in_presenza FROM sessione WHERE id_sessione = NEW.id_sessione) THEN
        RAISE EXCEPTION 'Errore: Le ricette possono essere associate solo a sessioni in presenza.';
    END IF;

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

4.4 Script di popolamento

Per concludere, al fine di verificare il corretto funzionamento del database e di prepararlo per essere di supporto all'applicativo *UninaFoodLab*, è stato creato uno script di popolamento che va a riempire tutte le tabelle con dati di prova. Per chi fosse interessato, lo script debitamente commentato, assieme a tutti gli altri file relativi al progetto, è integralmente visualizzabile qui.