Indice

1 Introduzione

Il seguente elaborato ha lo scopo di documentare la progettazione e lo sviluppo di una base di dati relazionale del DBMS PostgreSQL per il corso di Basi di Dati I. Il Database nasce per la gestione di un hackathon, ovvero una maratona di hacking.

1.1 Descrizione del problema

Il documento descrive la progettazione e lo sviluppo di una base di dati relazionale finalizzata alla realizzazione di un sistema per la gestione di hackathon, consentendo l'organizzazione degli eventi e la storicizzazione dei progetti realizzati dai vari team.
Il sistema consente la gestione completa di un hackathon, permettendo la registrazione degli utenti, la formazione dei team e la pubblicazione dei progetti. Ogni team collabora per risolvere il problema proposto, caricando periodicamente aggiornamenti relativi ai propri progressi, che possono essere visionati e commentati dai giudici. I giudici vengono selezionati dall'organizzatore tra gli utenti registrati alla piattaforma e, al termine dell'hackathon, assegnano una valutazione (da 0 a 10) a ciascun team. La piattaforma raccoglie le valutazioni e genera automaticamente la classifica finale. In questo modo, il sistema permette non solo di organizzare e monitorare le attività dell'hackathon, ma anche di storicizzare i progetti e i risultati dei vari team.

2 Progettazione Concettuale

In questo capitolo verrà affrontata la progettazione del database al suo livello più alto. Dall'analisi dei requisiti da soddisfare, si giungerà alla definizione di uno schema concettuale indipendente dalla struttura dei dati e dall'implementazione fisica. Tale schema è rappresentato con un Class Diagram UML, che metterà in evidenza le entità del problema, i relativi attributi, le relazioni tra esse e gli eventuali vincoli.

2.1 Class Diagram

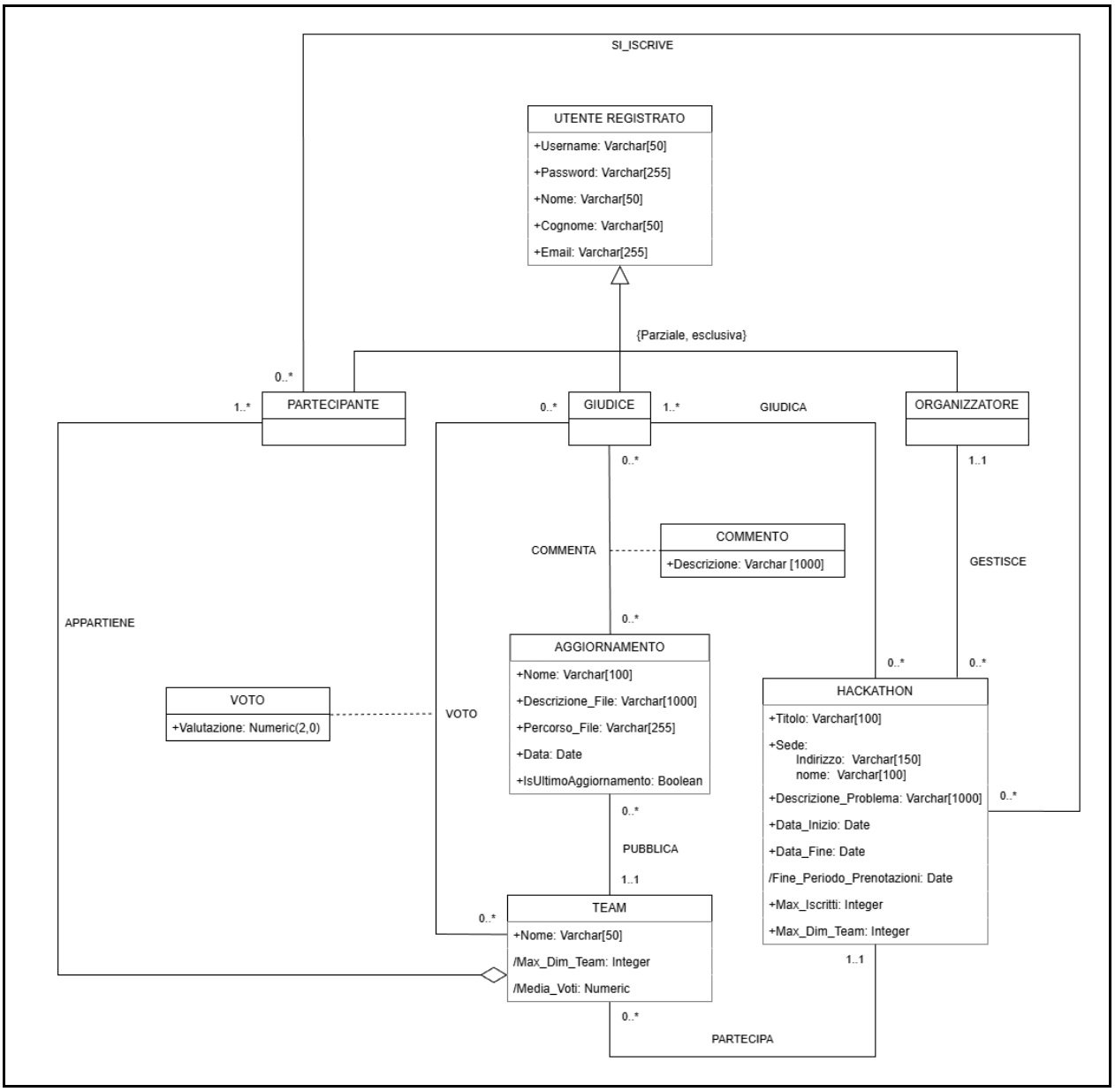

Figura 1: Class Diagram UML

2.2 Ristrutturazione del Class Diagram

Al fine di derivare lo schema logico, è necessario rielaborare il Class Diagram iniziale mediante un processo di ristrutturazione; questa fase ha lo scopo di analizzare il diagramma e rimuovere, se presenti, le criticità strutturali per garantire una corretta implementazione sul DBMS PostgreSQL. La fase di ristrutturazione seguirà i seguenti punti:

- Analisi delle chiavi
- Analisi degli attributi derivati
- Analisi delle ridondanze
- Analisi degli attributi strutturati
- Analisi degli attributi a valore multiplo
- Analisi delle gerarchie di specializzazione

Poiché il DBMS non distingue tra un associazione, un aggregazione e una composizione, in fase di ristrutturazione tali legami verranno mappati come associazioni.

2.2.1 Analisi delle chiavi

Per ottimizzare la gestione delle entità, si è optato per l'adozione di chiavi primarie surrogate: identificativi numerici privi di significato intrinseco. Tale scelta si è resa necessaria poiché non è stato possibile individuare, tra gli attributi già presenti, un candidato idoneo a svolgere la funzione di chiave primaria, garantendo cosi un'identificazione delle istanze univoca.
Nello specifico, sono state introdotte le seguenti chiavi surrogate:

- ID_hackathon per l'entità **hackathon** (preferito al titolo);
- ID_aggiornamento per l'entità **aggiornamento** (preferito al nome);
- ID_team per l'entità **team** (preferito al Nome del team).

Diversamente, per l'entità di un utente registrato, si è scelto di utilizzare l'attributo **username** come chiave primaria. Tale scelta è giustificata dall'unicità dello username in un sistema di autenticazione, che rende superfluo l'utilizzo di un codice numerico artificiale.

2.2.2 Analisi degli attributi derivati

Per ottimizzare le prestazioni, identifichiamo gli attributi derivati: informazioni che non necessitano di memorizzazione diretta, in quanto deducibili da altri dati.
Nel caso dell'entità **Team**, ad esempio, si individuano due attributi derivabili: la media dei voti (ottenibile dalla classe di associazione **Voto**) e la dimensione massima del gruppo (vincolo derivante dall'entità **hackathon**).
Parallelamente, per l'entità **hackathon**, è presente l'attributo derivato Termine Iscrizione. Quest'ultimo viene calcolato dinamicamente sottraendo due giorni dalla data di inizio dell'evento.
Si considerano impliciti, infine, altri attributi derivabili tramite operazioni di conteggio sulle associazioni. Tra questi rientrano, ad esempio, il numero di membri di un team, il totale degli iscritti a un hackathon, la quantità di aggiornamenti pubblicati e il numero di voti/commenti ricevuti; tutti valori calcolabili dinamicamente aggregando le relative occorrenze.

2.2.3 Analisi delle ridondanze

Si valuta, infine, la presenza di associazioni superflue, ovvero quelle deducibili attraverso altri percorsi già esistenti, al fine di snellire lo schema e rimuovere ambiguità.
L'unico caso di potenziale ridondanza è individuabile nell'associazione **SI_ISCRIVE** tra **Partecipante** e **hackathon**, data la presenza di un percorso indiretto che lega il **Partecipante** ad **hackathon** passando per l'entità **Team**.
Tuttavia, la relazione diretta è da considerarsi indispensabile e non ridondante: l'iscrizione individuale ad un hackathon costituisce, infatti, un vincolo necessario affinché un partecipante possa successivamente unirsi a un team.

2.2.4 Analisi degli attributi strutturati

Si procede ora all'identificazione di eventuali attributi strutturati. Il modello relazionale, infatti, non permette la loro implementazione diretta e necessitano di essere scomposti in attributi semplici (atomici).
Nel nostro caso, l'unico attributo strutturato è **sede**, presente nell'entità **hackathon**, in quanto composto da indirizzo e nome della struttura. Si è proceduto dunque alla sua decomposizione, sostituendo l'attributo originale con i due attributi semplici all'interno dell'entità.

2.2.5 Analisi degli attributi a valore multiplo

Si procede ora all'identificazione di eventuali attributi a valore multiplo. Tali attributi, infatti, non possono essere mappati direttamente in una colonna e richiederebbero la creazione di un'entità separata.
Tuttavia, dall'analisi dello schema concettuale corrente non sono emersi attributi di questo tipo, poiché tutti gli attributi sono a valore singolo, rendendo lo schema già conforme ai requisiti logici.

2.2.6 Analisi delle gerarchie di specializzazione

Si procede, infine, all'analisi delle gerarchie di specializzazione. Tali costrutti non hanno una rappresentazione diretta nei DBMS relazionali e richiedono pertanto l'adozione di specifiche strategie di mappatura per essere implementati correttamente.
Nello schema proposto è presente una singola gerarchia avente come superclasse **Utente Registrato** e come sottoclassi Partecipante, Giudice e Organizzatore. Essendo la generalizzazione definita come **parziale** (un utente può registrarsi senza assumere immediatamente un ruolo specifico) ed **esclusiva** (ogni utente può appartenere a una e una sola delle sottoclassi), si è deciso di risolvere la gerarchia adottando la strategia di accorpamento delle figlie nel padre.
Con questa soluzione, le entità figlie (Partecipante, Giudice, Organizzatore) vengono eliminate e assorbite nell'entità padre **Utente Registrato**.
Poiché le sottoclassi non presentano attributi specifici, l'operazione non comporta l'aggiunta di nuovi campi, ma solo il **trasferimento delle associazioni** verso la superclasse. Le relazioni che in precedenza legavano le sottoclassi ad altre entità (es. Partecipante con Team) vengono ora riferite direttamente a **Utente Registrato**.
Le relazioni specifiche di ogni ruolo sono state gestite introducendo apposite **classi di associazione**. Di conseguenza, il ruolo assunto dall'utente non viene memorizzato staticamente,

ma viene **dedotto dinamicamente**, verificando la presenza dell'istanza all'interno di queste tabelle di associazione.

2.3 Class Diagram ristrutturato

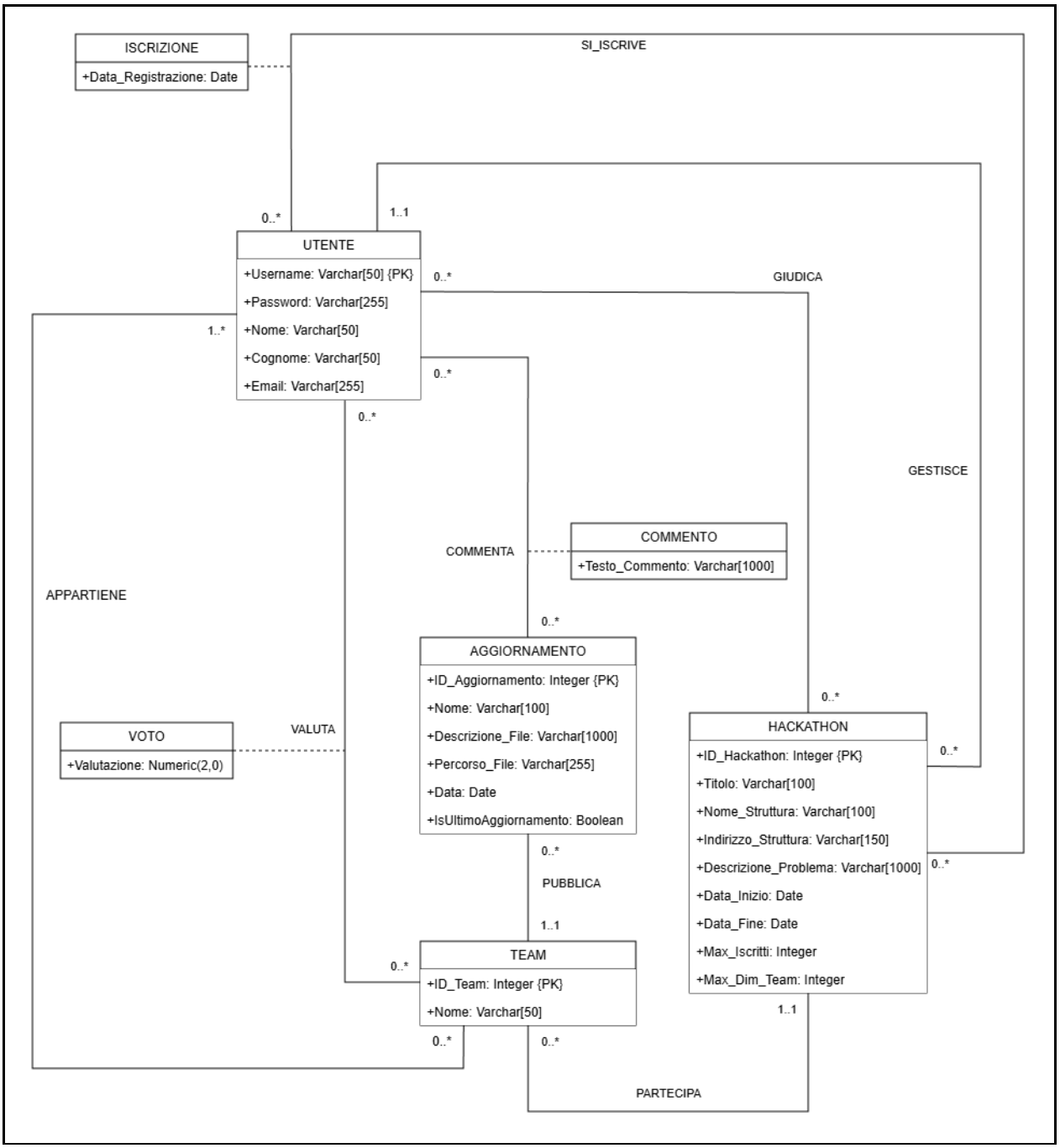

Figura 2: Class Diagram ristrutturato UML

2.4 Dizionario delle Classi

Classe	Descrizione	Attributi
UTENTE	Descrive ciascun utente iscritto alla piattaforma. Quest'ultimo può essere iscriversi ad un hackathon, organizzarne uno oppure essere nominato giudice da un organizzatore.	• **Username**(varchar(50)): Username identificativo per un utente. • **Password**(varchar(255)): Password di un utente. • **Nome**(varchar(50)): Nome di un utente. • **Cognome**(varchar(50)): Cognome di un utente. • **Email**(varchar(255)): Email di un utente.
HACKATHON	Descrive una maratona di hacking, organizzata da un utente, nella quale i team si affrontano, dando una propria risoluzione del problema proposto.	• **ID_hackathon**(Integer): Codice identificativo di un hackathon. • **Titolo**(varchar(100)): Nome dell'hackathon. • **Nome_struttura**(varchar(100)): Nome della struttura che ospita un hackathon. • **Indirizzo_struttura** (varchar(150)): Indirizzo della struttura. • **Descrizione_problema** (varchar(1000)): Descrizione del problema proposto. • **Data_inizio**(Date): Data di inizio hackathon. • **Data_fine**(Date): Data di fine hackathon. • **Max_iscritti**(Integer): Numero massimo di utenti iscritti. • **Max_dim_team**(Integer): Dimensione massima di un team.
ISCRIZIONE	Descrive un utente iscritto ad un hackathon.	• **Data_registrazione**(Date): Data di registrazione di un utente ad un hackathon.

AGGIORNAMENTO	Descrive il contenuto di un aggiornamento in merito al problema proposto, viene caricato da un team.	• **ID_Aggiornamento**(Integer): Codice identificativo di un aggiornamento. • **Nome**(varchar(100): Nome dell'aggiornamento. • **Descrizione_File**(varchar(1000)): Breve descrizione riguardante il contenuto dell'aggiornamento. • **Percorso_File**(varchar(255)): Stringa che indica il percorso del file. • **Data_caricamento**(Date): Data in cui un aggiornamento viene caricato. • **IsUltimoAggiornamento** (Boolean): una flag che indica se questo aggiornamento è l'ultimo che è stato caricato dal team
TEAM	Descrive un team di lavoro composto da utenti.	• **ID_Team**(Integer): Codice identificativo di un team. • **Nome**(varchar(50)): Nome di un team.
COMMENTO	Descrive un commento, scritto da un giudice, per un aggiornamento.	• **Testo_commento** (Varchar(1000)): Contenuto del commento scritto dal giudice.
VOTO	Descrive un voto assegnato da un giudice ad un team a fine hackathon.	• **Valutazione**(Numeric(2,0)): Voto assegnato da un giudice.

2.5 Dizionario delle associazioni

Nome	Descrizione	Classi coinvolte
GESTISCE	Esprime la gestione di un hackathon da parte di un utente (organizzatore).	• **UTENTE[1..1]**: Indica un utente (organizzatore) che gestisce un hackathon. • **HACKATHON[0..*]**: Indica un hackathon gestito da un certo utente (organizzatore).
GIUDICA	Esprime l'impiego di un utente (giudice) per un hackathon.	• **UTENTE[0..*]**: Indica un utente (giudice) che lavora per un hackathon. • **HACKATHON[0..*]**: Indica un hackathon per il quale un utente (giudice) lavora.
SI_ISCRIVE	Esprime l'iscrizione di un utente registrato in piattaforma ad un hackathon.	• **UTENTE[0..*]**: Indica un utente che si iscrive ad un hackathon. • **HACKATHON[0..*]**: Indica un hackathon al quale un utente si vuole iscrivere. **Classe di associazione:** ISCRIZIONE **Attributi:** Data_registrazione: Indica la data in cui un utente si è iscritto ad un hackathon.
APPARTIENE	Esprime l'appartenenza di un utente ad un determinato team dell'hackathon.	• **UTENTE[1..*]**: Indica un utente che si unisce ad un team. • **TEAM[0..*]**: Indica un team al quale un utente vuole unirsi.
PARTECIPA	Esprime la partecipazione di un team ad un hackathon.	• **TEAM[0..*]**: Indica un team che partecipa ad un determinato hackathon. • **HACKATHON[1..1]**: Indica un hackathon al quale un team vuole partecipare.

PUBBLICA	Esprime la possibilità da parte di un team di caricare un aggiornamento del proprio lavoro.	• **TEAM[1..1]**: Contenuto del commento scritto dal giudice. • **AGGIORNAMENTO[0..*]**: Indica un aggiornamento che viene caricato da un team.
COMMENTA	Esprime la possibilità, da parte di un giudice, di aggiungere un commento ad un aggiornamento pubblicato da un team.	• **UTENTE[0..*]**: Indica un giudice che commenta un aggiornamento di un team. • **AGGIORNAMENTO[0..*]**: Indica un aggiornamento che riceve un commento da parte di un giudice. **Classe di associazione:** COMMENTO **Attributi:** Testo_commento: Indica il testo del commento di un giudice per un determinato aggiornamento.
VALUTA	Esprime la possibilità da parte di un giudice di valutare un team.	• **GIUDICE[0..*]**: Indica un giudice che valuta un team. • **TEAM[0..*]**: Indica un team che viene valutato da un giudice. **Classe di associazione:** VOTO **Attributi:** Valutazione: Indica la valutazione che il giudice attribuisce ad un team.

2.6 Dizionario dei vincoli

Nome	Descrizione
Nome valido	Il nome di un utente registrato alla piattaforma deve essere unicamente composto da caratteri.
Cognome valido	Il cognome di un utente registrato alla piattaforma deve essere unicamente composto da caratteri.
Email valida	L'email di un utente registrato alla piattaforma deve seguire la struttura: "utente@dominio.estensione".
Unicità ruolo	Un utente può avere un unico ruolo attivo, non può esser organizzatore di un hackathon se è già un partecipante o giudice.
Unicità team	Un utente può afferire a un solo team per volta. Per potersi unire dovrà per prima cosa essere iscritto allo stesso hackathon del team.
Legame team-hackathon	Un team verrà creato esclusivamente per una certa competizione, non potrà passare da un hackathon ad un altro.
Unicità commento-giudice	L'inserimento di un commento a un aggiornamento è riservato esclusivamente ai giudici assegnati al relativo hackathon. Ogni giudice può inserire al massimo un unico commento per ciascun aggiornamento.
Unicità hackathon	Partecipanti, giudici o organizzatori possono essere associati ad un singolo hackathon attivo.
Termine caricamento aggiornamenti	Un aggiornamento può esser caricato solo dopo l'inizio dell'hackathon e non oltre le 23:59 del giorno antecedente la data di fine. Una volta pubblicato non potrà esser modificato.
Pubblicazione del problema	I giudici di un hackathon potranno pubblicare la traccia del problema da affrontare, ciò potrà avvenire solo ed esclusivamente dal giorno di inizio hackathon.

Controllo date hackathon	In fase di creazione, il sistema effettua dei controlli per garantire la validità delle date inserite. Nello specifico, la data di inizio deve garantire un preavviso di almeno 7 giorni rispetto alla data odierna e non può mai superare la data di fine. Inoltre, viene imposto un vincolo sulla durata complessiva dell'hackathon, che deve essere di almeno 14 giorni.
Modifica date hackathon	La data di inizio di un hackathon può essere anticipata o posticipata, a condizione che la nuova data garantisca un preavviso di almeno 7 giorni rispetto al momento della modifica. In ogni caso, l'evento dovrà sempre mantenere una durata complessiva minima di 14 giorni.
Scadenza iscrizioni	Il termine massimo per l'iscrizione a un hackathon è fissato a due giorni prima della data di inizio. Oltre questo limite, i team verranno considerati definitivi e non ne potranno esser creati di nuovi, inoltre non sarà più consentita l'adesione di nuovi partecipanti.
Controllo voto	L'inserimento di un voto a un team è riservato esclusivamente ai giudici assegnati alla relativa hackathon. Il giudice potrà assegnare un voto ad hackathon terminato, solo se il team avrà pubblicato almeno un aggiornamento. La valutazione avrà un valore compreso tra 0 e 10. Non potrà esser modificato o eliminato.
Limite dimensioni	Ogni competizione deve definire una capienza massima di partecipanti per l'intero evento e un limite massimo di componenti per i singoli team. Entrambi i valori devono essere strettamente positivi (maggiori di zero) e possono essere personalizzati dall'organizzatore per ogni singolo hackathon.
Termine hackathon	Il sistema conserva lo storico completo delle partecipazioni passate per tutti i ruoli. Tuttavia, un singolo utente non potrà esser coinvolto in più di un hackathon attivo contemporaneamente.
Unicità organizzazione	Non può esser modificato l'organizzatore di un hackathon creato.

3 Progettazione Logica

In questo capitolo analizzeremo la seconda fase della progettazione, riducendo il livello di astrazione per tradurre lo schema concettuale, opportunamente rifinito durante la fase di ristrutturazione, in uno schema logico. Tale rappresentazione risulterà strettamente dipendente dal modello di dati adottato: il modello relazionale puro.

3.1 Schema Logico

Il seguente schema logico definisce la struttura della base di dati. Al fine di distinguere i ruoli degli attributi, si è adottata la convenzione di sottolineare singolarmente le chiavi primarie e doppiamente le chiavi esterne.

- **UTENTE** (username, password, nome, cognome, email)

- **HACKATHON** (ID_hackathon, titolo, nome_struttura, indirizzo_struttura, descrizione_problema,
  data_inizio, data_fine, max_iscritti, max_dim_team, IDorganizzatore)
  IDorganizzatore → Utente.username

- **GIUDICA** (IDgiudice, IDhackathon)
  IDgiudice → Utente.username ; IDhackathon → Hackathon.ID_hackathon

- **TEAM** (ID_team, nome, IDhackathon)
  IDhackathon → Hackathon.ID_hackathon

- **AGGIORNAMENTO** (ID_aggiornamento, nome, descrizione_file, percorso_file, data_caricamento, isUltimoAggiornamento, IDteam)
  IDteam → Team.ID_team

- **VOTO** (IDgiudice, IDteam, valutazione)
  IDgiudice → Utente.username ; IDteam → Team.ID_team

- **COMMENTO** (IDgiudice, IDaggiornamento, testo_commento)
  IDgiudice → Utente.username ; IDaggiornamento → Aggiornamento.ID_aggiornamento

- **ISCRIZIONE** (IDutente, IDhackathon, data_registrazione)
  IDutente → Utente.username ; IDhackathon → Hackathon.ID_hackathon

- **APPARTIENE** (IDutente, IDteam)
  IDutente → Utente.username ; IDteam → Team.ID_team

4 Progettazione Fisica

Questo capitolo è dedicato alla fase di progettazione fisica del database: verrà documentato il processo di traduzione dello schema logico, delineato nel capitolo precedente, nell'equivalente modello fisico, il tutto mediante l'utilizzo del DBMS relazionale PostgreSQL.

4.1 Definizione tabelle

Nelle sezioni seguenti vengono riportate nel dettaglio le istruzioni per la creazione delle tabelle, includendo la definizione dei relativi vincoli intrarelazionali, fondamentali per garantire coerenza e correttezza dei dati, e delle eventuali strutture di supporto necessarie alla loro gestione.

4.1.1 Definizione della tabella UTENTE

```
1.  --Definizione tabella per la creazione di un utente
2.  CREATE TABLE UTENTE (
3.      username VARCHAR(50) PRIMARY KEY,
4.      password VARCHAR(255) NOT NULL,
5.      nome VARCHAR(50) NOT NULL
6.      CHECK (nome ~ '^[a-zA-ZÀ-ÿ\s'']+$'),
7.      cognome VARCHAR(50) NOT NULL
8.      CHECK (cognome ~ '^[a-zA-ZÀ-ÿ\s'']+$'),
9.      email VARCHAR(255) UNIQUE NOT NULL,
10.     CHECK (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z
            ]{2,}$')
11. );
```

4.1.2 Definizione della tabella ISCRIZIONE-UTENTE-HACKATHON

```
1.  --Definizione tabella per l'iscrizione di un utente ad un
        hackathon
2.  CREATE TABLE ISCRIZIONEUTENTEHACKATHON (
3.      id_utente VARCHAR(50) NOT NULL,
4.      id_hackathon INTEGER NOT NULL,
5.      data_registrazione DATE NOT NULL,
6.      CONSTRAINT iscrizioneUtenteHackathon_pkey PRIMARY KEY (
            id_utente, id_hackathon),
7.      CONSTRAINT fk_hackathon FOREIGN KEY (id_hackathon)
8.          REFERENCES HACKATHON(id_hackathon) MATCH SIMPLE
9.          ON UPDATE CASCADE
10.         ON DELETE CASCADE,
11.     CONSTRAINT fk_utente FOREIGN KEY (id_utente)
12.         REFERENCES UTENTE(username) MATCH SIMPLE
13.         ON UPDATE CASCADE
14.         ON DELETE CASCADE
15. )
```

4.1.3 Definizione della tabella HACKATHON

```
1.  --Definizione tabella per la creazione di un hackathon
2.  CREATE TABLE HACKATHON (
3.      id_hackathon INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
4.      titolo VARCHAR(100) NOT NULL,
5.      nome_struttura VARCHAR(100) NOT NULL,
6.      indirizzo_struttura VARCHAR(150) NOT NULL,
7.      descrizione_problema VARCHAR(1000) NOT NULL,
8.      data_inizio DATE NOT NULL,
9.      data_fine DATE NOT NULL,
10.     max_iscritti INTEGER NOT NULL,
11.     max_dim_team INTEGER NOT NULL
12.     CHECK (max_iscritti > 0 AND max_dim_team > 0),
13.     id_organizzatore VARCHAR(50) NOT NULL,
14.     CONSTRAINT fk_organizzatore FOREIGN KEY (id_organizzatore)
15.         REFERENCES UTENTE(username) MATCH SIMPLE
16.         ON UPDATE CASCADE
17.         ON DELETE RESTRICT
18. )
```